Beköszöntő

Május van, éljen a tavasz, izzadjanak a pingvinek!
Lássuk, mit tartogat ez a szám azoknak, akik a napsütés ellenére is hajlandóak olvasással tölteni egy kis időt.
Műszaki érdeklődésű olvasóink közül valószínűleg sokan tudják, mit neveznek a mérnöki gyakorlatban „léptékhatásnak". Ha kicsiben akarunk valamit megcsinálni, gyakran egészen más módszerekre és eszközökre van szükségünk, mint annak, aki „nagyban játszik". És ez néha az egészen egyszerűnek látszó dolgokra is igaz. Ha például mindössze annyi a feladat, hogy forraljunk tejet, akkor is előre tudnunk kell, hogy mekkora a „feldolgozandó" anyagmennyiség. Fél litert az ember egyszerűen feltesz egy fazékban a gázra. 2-3 liternél már résen kell lenni, mert ha nem kevergetjük, akkor az alja odakozmál. 100 liternél pedig keverhetjük, ahogy akarjuk, akkor is odaég. Ilyenkor a módszert és az eszközt kell lecserélni.
Ugyanez igaz az ügyfelek kezelésére is. Amelyik cégnek vagy vállalkozónak mindössze 5-10 főnyi állandó ügyfélköre van, az egy notesz segítségével is elboldogul. Több száz, vagy több ezer ügyfél esetében azonban már „célszerszám" kell a sikerhez. Ilyeneket lehet az adott célnak megfelelően fejleszteni, lehet komoly pénzért vásárolni, de akár ingyenes eszközt is találhat, aki jól tájékozott.
Horváth Ernő bemutat egyet azok közül a nyílt forrású ügyfélkezelő rendszerek (más néven CRM rendszerek) közül, amelyek jelentősen megkönnyíthetik egy-egy komolyabb ügyfélkörrel rendelkező cég életét. Aki ért hozzá, az néha talán érezte már úgy, hogy a számítógép nem csupán munkaeszköz, hanem egyfajta társ is. Némelyeknél olyasféle kapcsolat ez, mint ami motoros és kedves benzinparipája között szokott kialakulni. A szenvedélyes motorosok és autósok féltő gonddal fényezgetik a karosszériát, sőt néha még beszélnek is a gépükhöz. Olyan azonban, hogy a motor vagy autó visszaszóljon, még nem nagyon fordult elő. (Eltekintve persze ***Knight Ridertől***, meg azoktól, akik egyébként is rendszeresen szoktak hangokat hallani.)
Ebből is látszik, hogy a számítógép, az azért valami nagyon más. A digitális technika jelenlegi fejlettségi szintjén valójában nincs különösebb akadálya annak, hogy bármilyen szöveges tartalmat csaknem teljesen élethű emberi hangon olvasson föl nekünk a gép. Ez a lehetőség különösen nagy segítség lehet a látássérülteknek, bár eddig viszonylag drága programok kellettek hozzá. A közelmúltban azonban – mint megannyi más területen – itt is megjelent a nyílt forrású alternatíva. Az emberi beszéd előállításának lehetőségeit mutatja be ***Klein Eleonóra*** cikke.
A mostanában meglehetősen sokat emlegetett számítógépes betörések és elkövetőik általában amolyan „bizsergetően érdekes" témát jelentenek a médiának. Ebben a számban – ha nem is örökre – de mi is csatlakozunk a trendhez, ***Dr. Dudás Ágnes*** ugyanis a kalózkodást és annak jogi következményeit elemzi. Elöljáróban mi egyebet is mondhatnánk: reszkessetek betörők.
Természetesen folytatódnak korábbi sorozataink is. ***Fábián Zoltán*** az ***eFltk*** környezet, míg ***Illés Viktor*** a ***Samba*** használatát taglalja. ***Komáromi Zoltán*** az ***OpenOffice.org 2.0***-ás változatának új szolgáltatásait tekintette át, ***Garzó András*** pedig folytatja a számítógép hálózatok bemutatását.

Hasznos időtöltést, jó szórakozást kíván a Linuxvilág stábja.

Kódkert

Az *O'Reilly Media* és a nyílt forrású *IT* szolgáltatásokkal foglalkozó *SpikeSource* közös internetes forráskódtárat hoztak

létre. A *CodeZoo* oldalon az *O'Reilly* könyveiben, egyéb kiadványaiban, blogjaiban, illetve a kiadóhoz kötődő konferenciákon szereplő kódrészleteket, leírásokat, feljegyzéseket kívánják összegyűjteni. A *CodeZoo* tartalma jó néhány kategóriát fog majd át, kezdve a játékoktól kezdve az üzleti alkalmazásokon keresztül egészen a tudományos programokig; célja – ahogy minden hasonló gyűjteményé – a tanulás és a jobb minőségű programkódok készítésének elősegítése.
➲ http://www.codezoo.com/

Nyitott a Nero

A *Nero AG* linuxos változatban is megjelentette közismert és kedvelt *Nero CD-* és *DVD*-író alkalmazását.

A *NeroLINUX RPM* vagy *DEB* formátumú, szabványos csomagként tölthető le, jelenleg a *Red Hat 7.2 - 9.0* és *Enterprise Linux 3.0*, a *SuSE 8.0 - 9.2* és a *Debian 3.0* és *3.1* terjesztéseket támogatja. A *NeroLINUX* 2.4-es és 2.6-os rendszermag alatt futtatható, külső segédprogramokkal képes a menet közben végzett hangkódolásra és – dekódolásra, továbbá a belső meghajtók mellett az USB kapura csatlakozó külső egységeket is ismeri.
➲ http://www.nero.com/en/NeroLINUX.html

Adobe Reader 7.0 Linux alá

Szemfüles webezők felfedezték, hogy bár az *Adobe* letöltési oldalán nem dicsekednek vele, az *Adobe Reader* legújabb, 7.0-s változata *Linux* alá is elkészült. A *Linuxot* használók – már amennyiben ragaszkodtak az *Adobe* termékéhez – eddig egy ma már meglehetősen öregecske változattal, az 5.0.10-essel voltak kénytelenek beérni, miközben a windowsos tábor folyamatosan újabb és újabb kiadásokhoz juthatott hozzá. Az *Adobe Reader 7.0* linuxos kiadása az ➲ *ftp://ftp.adobe.com/pub/adobe/reader/unix/7x/7.0/enu/* címről tölthető le.

Bővülő NVIDIA 6000-család

Két új taggal bővült az *NVIDIA GeForce 6 GPU*-sorozata, a belépő szintű *6200*-as modellel és a felső kategóriába tartozó *6800 Ultra* lapka 512 MB memóriával ellátott változatával. Utóbbi a nagy mennyiségű *DDR3* memóriának köszönhetően még szebb, még jobb, még színesebb képet biztosít, és természetesen támogatja a két kártya együttes használatát lehetővé tévő *SLI* megoldást is; előbbi pedig a húszezer forint alatti piaci szegmensben biztosít viszonylag kifinomult grafikai szolgáltatásokat. Az új *GPU*-kra épülő kártyák áprilisban jelennek meg, várhatóan *AGP* és *PCI Express* csatlakozóval egyaránt kaphatók lesznek.

Kémelhárítás

Kémprogramok elleni védekezést segítő szoftverfejlesztő készletet adott ki az *InterMute*. A cég *SpySubtract* programjához épült készletet termékintegrátoroknak, hardver- és szoftverfejlesztőknek egyaránt ajánlják, célja az egyes termékek kémprogramok elleni védelemmel való bővítésének segítése. Alapját a *SpySubtract* motorja képezi, ezt a más termékekkel (tűzfalakkal, víruskeresőkkel, behatolásérzékelő alkalmazásokkal) való egybeépítést támogató *API*-készlet egészíti ki. Segítségével a motor asztali gépeken, kiszolgálókon és hálózati készülékeken egyaránt futtatható, míg a vállalati szintű megoldások kidolgozását távfelügyeleti és automatikus telepítési megoldások teszik lehetővé.
➲ www.intermute.com

Intel alsó-felső szinten

Az *Intel* a közeljövőben mind a belépő szintű, mind a kiszolgáló gépek piacára újdonságokat ígér. A hétköznapi felhasználók új, már a 64 bites kiterjesztéseket is támogató *Celeron* processzorokat kapnak. A *Celeron D* sorozatba tartozó lapkák órajele várhatóan 2,53 és 3,33 GHz közé esik majd, a lassanként követhetetlenné váló számozási rendszerben pedig a 326-355 tartományból kapnak azonosítókat. Az új *Celeronok Socket 775* foglalatba illeszkednek majd, 256 KB másodszintű gyorsítótárat kapnak, előoldali buszsebességük 533 MHz lesz. Biztosan támogatni fogják a futtatás letiltását lehetővé tévő jelzőbit használatát, míg az, hogy az órajel üzem közbeni állítását lehetővé tévő *Enhanced Intel SpeedStep Technology*-t is ismerni fogják-e, egyelőre kérdéses, ahogy egyelőre a pontos megjelenési ütemezés és az árak is homályban maradtak.
A kiszolgálók területén fontos újdonság lesz a legfeljebb négy darab *Xeon* processzor támogatására képes *E8500* lapkakészlet. Az *E8500* kettő darab előoldali buszt támogat majd, ezek órajele 667 MHz lesz. Mindkét buszra két-két processzor csatlakozhat majd, amivel a lapkakészlet jóval nagyobb teljesítményt biztosít, mint egyetlen 400 MHz-es buszra négy processzort csatlakoztató elődjei. Továbblépést jelent a négy memóriavezérlő támogatása, amelyek két csatornán keresztül *PC2100*, *PC2700* és *PC2-3200* típusú memóriákat kezelnek, akár memóriatükrözéssel, memória *RAID*-del, *ECC* alapú védelemmel és üzem közbeni cserével. A lapkakészlet 28 *PCI Express* csatornát lesz képes kezelni, ami például – elvileg – 28 darab *PCI Express x1* csatolókártya vagy három darab x8 és egy darab x4 kártya csatlakoztatását teszi lehetővé.
Az új lapkakészlethez új processzor is jár, a csúcs *Xeon* immár 3,33 GHz-es órajellel és 8 MB gyorsítótárral (valamint jóval félmillió forint feletti árral) üzemel, és esetében is számolhatunk az üzem közbeni órajelváltások lehetősége és a 64 bites kiterjesztések által kínált lehetőségekkel.

Kis helyen is elfér

A *Toshiba* után a *Hitachi* is bejelentette, hogy már ez év második felében szeretne függőleges adatrögzítési eljárást alkalmazó merevlemezeket piacra dobni. A merevlemezek kapacitását hosszú évek óta úgy növelik a gyártók, hogy a lemezek felületére írt mágneses jeleket egyre közelebb tolják egymáshoz. Az újfajta eljárásnál a mágneses hordozó réteget jóval mélyebben mágnesezik, így a korábbinál is közelebb tudják helyezni egymáshoz az egyes biteket hordozó jeleket – az újfajta rögzítési megoldás tehát kisebb technológiai ugrást jelent a merevlemezek világában. A váltás a felhasználók számára természetesen észrevétlen, ők csak annyit fognak tapasztalni az egészből, hogy hamarosan megjelennek a 100 GB feletti kapacitású, hordozható gépekbe szánt merevlemezek, az asztali egységek kapacitása pedig hamarosan akár az 1 GB-ot is elérheti.

Szárnyal a Firefox

Februárban továbbra is az *Internet Explorer* uralta a böngészők piacát, ám a *Firefox* fokozatosan erősítve immár 8,45 százalékos részesedést ért el. Bár ez nagyjából egytizede az *Explorer* részesedésének, ahhoz lassan elég lesz, hogy a weblapok fejlesztői ne csak *Explorerre* készítsék oldalaikat, de a *Firefox/Mozilla* vonal alatti megjeleníthetőséget is kénytelenek legyenek figyelembe venni. Üröm az örömben, hogy elsősorban a műszaki dolgok iránt érdeklődő felhasználók használják az alternatív böngészőket, a hétköznapi internetezők maradnak a kék **e** betűnél.
A *Firefox* terjedésének másik hátulütője az, hogy egyre könnyebben felszínre kerülnek a benne lévő hibák. Az 1.0.0 változat kiadását gyors ütemben követte a 17 hibajavítást magába foglaló 1.0.1, majd a további három biztonsági rést betömő 1.0.2, ám a közeljövőben további sebezhetőségek felismerésére lehet számítani. Aminek kapcsán felmerül az a kérdés is, hogy a *Firefox* vajon valóban biztonságosabb, mint a *Microsoft* böngészője, vagy csak egész egyszerűen senki nem foglalkozott eddig azzal, hogy hibákat keressen benne?

Megalakult a LiSoG

LiSoG (Linux Solutions Group) névvel új – vagy ha úgy tetszik, újabb – a *Linux* terjedését segítő szövetség alakult. Az *IBM*, a *MySQL*, a *Novell*, a *Red Hat*, a *Siemens* mellett számos további informatikai vállalat, továbbá egyetemek és felhasználók támogatását maga mögött tudó, irodáját a németországi Stuttgartban megnyitó, tevékenységét elsősorban a német nyelvterületekre összpontosító szövetség az ipari igényeket és a műszaki lehetőségeket egyaránt figyelembe vevő, szigorúan a tényleges felhasználói igények alapján készülő megoldások kidolgozását célozza.
A *LiSoG* elsősorban tudásanyagot kíván közvetíteni a leendő felhasználók felé, segíteni szeretné őket a döntések meghozatalában, illetve be kívánja mutatni nekik a különféle megoldások, üzleti alkalmazások működését, függetlenül az alkalmazott alaprendszertől, a gyártóktól és technológiáktól.
➲ www.lisog.org

Egy gombnyomásra a világ

A *Zoom Technologies* újabb tagokkal bővítette *ADSL* modemes termékcsaládját. A négy kapuval rendelkező *X6*

ADSL modem tulajdonképpen nem több, mint egy nagyon jó forgalomirányító és tűzfal, amely amellett, hogy internet-hozzáférést biztosít, vezeték nélküli hálózatokkal is képes kapcsolatot teremteni, valamint kiterjedt felügyeleti szolgáltatásokat nyújt. Az *X5v* és a *v3* modem vezeték nélküli kapcsolatok létesítésére ugyan nem képes, a vezetékes világ két részét, a telefonhálózatokat és az internetet azonban közelebb hozza egymáshoz. Ezek a készülékek rendelkeznek egy külön aljzattal, amelyhez hagyományos telefonkészüléket kell csatlakoztatni; ezt követően a felhasználó választhat, hogy kimenő hívását a hagyományos rendszeren szeretné bonyolítani – mert például a helyi hívások ingyenesek –, vagy rendkívül kedvező áron *VoIP* alapú hívást szeretne kezdeményezni. Utóbbit a *Global Village* szolgáltatás bonyolítja, mely a SIP szabvány szerint építi fel a kapcsolatokat, vagyis a hasonló jellegű készülékek és szolgáltatások túlnyomó részével képes együttműködni. Fizetni a *VoIP* alapú hívásokért csak akkor kell, ha a túlsó végponton kilépnek az internetes hálózatból, és hagyományos telefonállomáson végződnek; ha a beszélgetőpartner szintén internetes csatlakozást használ, akkor a beszélgetés ingyenes (sőt, a cég tervei szerint ez a jövőben is így marad).
A *Zoom* újdonságai a tengeren túl körülbelül 100 dollár körüli árakon vásárolhatók meg.
➲ www.zoom.com

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

Mi újság a rendszermag fejlesztése körül?

Linus Torvalds és *Andrew Morton* továbbra is keresik a *Linux* fejlesztésének legjobb módját. Az üzembiztos és a próbasorozatok ötlete életképtelennek bizonyult, miközben egyre erősebb az igény az egyes kiadások üzembiztossága iránt, ahogy ezt a 2.6.9-es és a 2.6.10-es kiadásnál is láthattuk Eközben sok felhasználó vonakodik a 2.6-os rendszermagok tesztelésétől, éppen az ezekbe kerülő fejlesztések hatalmas mennyisége miatt. *Linus*, *Andrew* és mások most azon törik a fejüket, vajon a kiszámíthatatlan ütemezésűvé vált hivatalos fa hogyan nyerhetné el minél több tesztelő bizalmát. Az egyik felvetés az üzembiztos és a próbaváltozatok váltakozó kiadásának visszahozása volt. A 2.6.11 tehát egy üzembiztos rendszermag volna, és csak az utóbbi hónapok hibajavításait tartalmazná, a 2.6.12 pedig az elmúlt hónapok újdonságait foglalná magába, és így tovább. Egy másik megoldás az lenne, ha egy negyedik számmal bővítenék a változatszámokat, így például a 2.6.11.2 és a 2.6.11.3 hibajavításokat tartalmazó kiadás lenne, az új fejlesztések pedig a 2.6.12-es kiadásba kerülnének. Eddig még semmi nem dőlt el, *Linus* és *Andrew* egyelőre az eddigi rendszer feladásának hatásait próbálják feltérképezni. Érdemes követni a fejleményeket.

Érdekes szerzői jogi kérdés merült fel, amikor *Adrian Bunk* felvetette, hogy a *ReiserFS* fájlokban van egy megjegyzés, amely szerint minden bővítés szerzői joga *Hans Reiserre* száll át. Bár a szerzők megtehetik, hogy hozzájárulásaikhoz külön nyilatkozatot mellékelnek, amellyel fenntartják jogaikat, ám *Adrian* mégis zavarosnak vélte a dolgot. *Linus Torvalds* támogatásáról biztosította *Hans* eljárását, valamint maga *Hans* ügyel arra, hogy minden hozzájáruló figyelmét felhívja a szerzői jogi kérdésekre. *Hans* szerint a kérdéses szöveg csak a forrásfájlokban szerepel, célja pedig csupán az, hogy védje magát például a *The SCO Group* rosszindulatától. *Christoph Hellwig* rámutatott, hogy az *SGI* is hasonló eljárást követ, amikor befogadja mások hozzájárulásait az *XFS* fájlrendszerhez. Megfelelő precedenssel, némi előzékenységgel és a fő-fő linuxos figura jóváhagyásával lehetséges, hogy ez a megoldás a rendszermag egyéb területeire is át fog terjedni.

Marcus Metzler felhívta a figyelmet arra, hogy az *iRiver* kiadott egy kizárólag futtatható formában elérhető, *Linux* alapú terméket, és határozottan elutasította a forráskód kiadását. Hirdetéseikben és útmutatóikban egy pillanatig sem csinálnak titkot abból, hogy multimédiás lejátszójuk *Linuxra* épül, ám a *GPL* szerződés már elmaradt, és sem weboldaluk, sem maga a termék semmilyen lehetőséget nem kínál a forrás beszerzésére.

A *SquashFS* tömörített fájlrendszer rendkívül közel került ahhoz, hogy a hivatalos rendszermagfa részévé váljon. *Phillip Lougher* kódja összefogott, jól működő és letisztult. Többen, köztük *Greg Kroah-Hartman* is biztatták a kód beépítésére, ám *Phillip* vonakodik. Számos új szolgáltatást szeretne még hozzáadni, és még nem tudta eldönteni, hogy ezeket a hivatalos rendszermagba való befogadás előtt vagy után lenne jobb megvalósítani. Jómagam biztos vagyok abban, hogy amikor *Phillip* elérkezettnek látja az időt, a *SquashFS* zökkenőmentesen fog beépülni a 2.6-os rendszermagba. A rendszermagot gondozó srácok már türelmetlenül várják.

A *FUSE* – ez egy felhasználói térben futó fájlrendszer – ellenben komoly problémákkal küzd, miközben a fő rendszermagfa részévé próbál válni. Különösen *Linus Torvalds* véli úgy, hogy a fájlrendszerek jellegüknél fogva nem igazán futhatnak felhasználói térben. A fájlrendszer leválasztása a rendszermagról szerinte olyan, mintha a mikrorendszermagok szemléletét követve elkülönítenénk egymástól a rendszer belső részeit. Amiért *Linus* a monolitikus rendszermagszerkezetben hisz, azért utasítja el a felhasználói térben futó fájlrendszer ötletét is. Más részről *Linus* úgy nyilatkozott, hogy hajlandó elfogadni a *FUSE*-t, ám csak korlátozott szolgáltatáskészlettel, a felhasználói térben futó fájlrendszerekhez nem illő jellemzők eltüntetése után. Hasonló korlátokat emelt annak idején a *DevFS* előtt is. A *DevFS* kapcsán viszont teljes zűrzavar alakult ki, részben azért, mert a */dev* könyvtár a *Linux* működésének egyik sarokköve. Talán soha más fájlrendszer nem lesz ennyire vitatott sorsú.

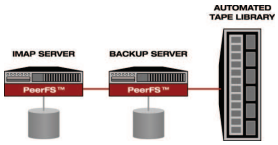
Zack Brown

Linux Journal 2005. április, 132. szám

Új termékek

PeerFS 3.0 változat

A *Radiant Data Corporation* kiadta a *PeerFS 3.0*-s változatát. A *PeerFS* az adatok folyamatos elérhetősé-



gét biztosítja Linux alapú vállalati alkalmazások számára. A *PeerFS* segítségével egyidejű tranzakciókat lehet végezni olyan kiszolgálókon, amelyek különböző helyeken vannak, és különálló, de azonos adattárolókkal rendelkeznek. A *PeerFS 3.0*-s kiadásának újdonsága a többféle terjesztés támogatása, ide értve a *Trustixot* és a *Debiant*; a 2.6-os rendszermag, a *SuSE Standard Server 9.0* és a *SuSE Enterprise Server 9.0* támogatása; az elveszett csomópontokra vonatkozó házirend, amely képes észlelni, ha a csoport egy vagy több csomópontja elérhetetlenné vált; valamint a kettőnél több csomópontból álló egységességi csoportok megadásának lehetősége. Emellett a *PeerFS* lemez nélküli ügyfelei új szolgáltatásokkal bővültek, a *mount* parancs a továbbiakban képes a terheléselosztásra és az állomáshoz kötésre.
www.radiantdata.com

1-Box for Linux 1.0

Az *1-Box for Linux 1.0* egy önálló program, tetszőleges *Linux* terjesztéshez hozzáadva egyetlen PC-ből egy akár tíz munkaállomásból álló hálózatot hoz létre. Ha a fő PC-t két monitor vezérlésére is képes videokártyákkal bővítjük, akkor a munkaállomásokhoz csak egy monitorra, valamint egy *USB*-s billentyűzetre és egérre van szükség. A felhasználók egyszerre böngészhetnek az interneten, egymástól függetlenül levelezhetnek, és a telepített programok bármelyikét szabadon futtathatják. Az *1-Box* a *Novell*, a *Mandrake*, a *Fedora Core* és a *Red Hat* terjesztéseket támogatja, a sor hamarosan a Sun Java Desktoppal bővül.
www.userful.com

WebScan for Linux

A *WebScan for Linux* víruskereső és tartalombiztonsági szolgáltatásokat egyesítve védi a hálózatot, mégpedig az átjáró vagy a proxykiszolgáló szintjén. A *WebScan* alkalmas arra, hogy a szervezetek szabályozzák az átjárón keresztül elérhető webes tartalom típusát, illetve védjék a hálózatot a proxykiszolgálókon keresztül bejutni próbáló vírusoktól. A *WebScan* képes a weblapok tartalmának házirend alapján történő ellenőrzésére, valamint a vírusok, férgek, trójaik és egyéb rosszindulatú programok felismerésére. *MIME* fájltípusokból álló feketelistát is összeállíthatunk, amelyre például a hang- és képfájlokat felvéve takarékoskodhatunk az internetkapcsolat sávszélességével. A *HTTP* alapú fájlfeltöltések letiltására is alkalmas, amivel megelőzhető a bizalmas adatok ellopása, kiszivárogtatása. Segítségével számos webhely engedély nélküli elérését megelőzhetjük olyan szervezetek minősítései alapján, mint a *RASCi*, *Safe Surf* és *ICRA*.
A rendszergazdák számára a *WebScan* széles körű jelentőrendszert biztosít a házirend áthágásainak figyelésére, a beállítások módosítását és a felügyeletet pedig grafikus felülettel segíti.
www.mwti.net

PostgreSQL 8.0

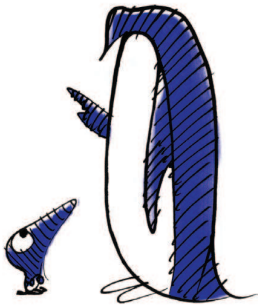
A *PostgreSQL Global Development* kiadta a *PostgreSQL* objektum alapú, relációs adatbázis-kezelő rendszer 8.0-s változatát. A 8.0-s változat fontos újdonsága az *SQL*-világban jól ismert mentési pontok támogatása, amelyek segítségével egy adatbázis-tranzakció bizonyos részei a teljes művelet visszavonása nélkül is visszavonhatók. Ugyancsak új a *PostgreSQL 8.0*-ban az időpontra való visszaállítás lehetősége, amely az önműködően és folyamatosan vezetett tranzakciós naplók alapján lehetővé teszi az adatok teljes visszaállítását, illetve az óránkénti és napi biztonsági mentések kiváltására is alkalmas. A 8.0-s változat a táblatereket is ismeri, így a nagyméretű táblákat és indexeket különálló lemezekre vagy tömbökre is képes elhelyezni, növelve a lekérdezések sebességét. Végül a *PostgreSQL* az *Adaptive Replacement Cache* algoritmus, az új háttéríró és a szintén új vákuum késleltetés révén továbbfejlesztett lemez- és memóriahasználatot ígér.
www.postgresql.org

IBM OpenPower 710

Az *IBM* bejelentette a *POWER5* processzorra épülő, *Linuxot* futtató *eServer OpenPower 710* kiszol-

gálót. Az *OpenPower 710* egy vagy két processzort tartalmazó, az *IBM* 64 bites *Power* géptípusára épülő, szekrénybe szerelhető rendszer, kiegészítő jelleggel a *POWER5* rendszerekre egyedileg jellemző, a nagygépek világát idéző virtualizációs és mikroparticionálási szolgáltatások támogatására is képes. Az *OpenPower 710* 1,65 GHz-es *POWER5* mikroprocesszorokkal és legfeljebb 32 GB memóriával vásárolható meg. Támogatja a *Novell SUSE LINUX Enterprise Server 9*-et és a *Red Hat Enterprise Linux AS 3*-at. Alapesetben a 710 1 GB memóriát tartalmaz, házában 73 GB-os 10000-es fordulatszámú merevlemezt és *DVD-ROM*-meghajtót találunk. A garancia három év, következő munkanapi javítást biztosít. A négy darab szabványos, menet közben cserélhető *Ultra320 SCSI* meghajtóhely révén a gép akár 570 GB-nyi belső tárhelyet is kaphat. A bővítést három *PCI-X* foglalat és két 10/100/1000 Mbps sebességű *Ethernet* aljzat segíti, a magas rendelkezésre állást pedig az igény szerint akár redundáns, üzem közben csatlakoztatható tápegységek és hűtők biztosítják.
www-1.ibm.com/servers/eserver/openpower

Linux Journal 2005. 132. szám



A Linux Journal honlapján számtalan gond megoldásához találhattok további segítséget. A Sunsite tükröoldalait, 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

A fájlleírók és a blokkméretek módosítása

Két kérdést szeretnék feltenni.

1) Vajon milyen hatással lesz a teljesítményre, ha egy linuxos lemezrészt az alábbi parancsokkal formázok meg:

```
mkfs.ext2 -i 1024 -b 1024 /dev/hda1
mkfs.ext3 -i -1024 -b 1024 /dev/hda2
```

Tudom, hogy a második paranccsal naplózó fájlrendszert hozok létre, de a nagy számú fájlleíró miatt nem fog lelassulni a rendszer? Egy tűzfalgépről van szó, amin ***Squid***, ***INN*** és ***qmail*** szolgáltatás futna.

2) Van két hasonló gépem, 66 MHz-es 486DX és 50 MHz-es 486SLC2 processzorral, mindkettőben 32 MB RAM van. Megoldható, hogy ***Red Hat Linux 9***-es változatát futtassam rajtuk? Vagy inkább tegyem fel a ***Red Hat 6.2***-es változatát, majd az up2date-tel frissítsem?

Lee Spivey,
➲ *tuskyhe@yahoo.com*

1) Az, hogy a fájlleírók mérete és száma mekkora hatást gyakorol a lemezelérési sebességre, attól függ, hogy milyen típusú fájlok vannak a lemezen. A fenti parancsokkal inkább a merevlemez kapacitásának jobb kihasználását lehet elérni – persze az sem rossz dolog. Ez különösen a nagyméretű merevlemezeknél igaz, náluk megsokszorozódik a fenti érték hatása.
A gyakorlatban azonban a weblapok és üzenetek mérete már túllépett az 1 KB-on. Ha a fájlrendszer blokkméretét ekkorára korlátozod, akkor a Linux a megfelelő adatok megtalálásához túlságosan sok fájlleírót lesz kénytelen bejárni és követni. Minél több fájlleíró van egy fájlban, annál tovább tart mindez. Figyelembe véve a jelenlegi merevlemezek MB-ra vetített árát, valamint azt, hogy a megtakarítás valószínűleg nem lesz több 100 MB-nál, inkább 4-8 KB-os érték használatát javaslom.

Chad Robinson,
➲ *chad@lucubration.com*

1) Ahogy ***Chad*** is rámutatott, a blokkméret befolyásolja a teljesítményt. Ha a használatban lévő fájlok jellemzően 1 KB feletti méretűek, akkor elérésükhöz több fájlleírót is be kell olvasni, ami teljesítményromláshoz vezet. Nem is arról van szó, hogy mennyi fájlleíród lesz, hanem arról, hogy a leggyakrabban használt fájlok beolvasásához hányat kell elérni. Vagyis ami fontos, az a fájlleíró/fájlméret arány, tehát pontosan az általad megadott `mkfs` parancsokban szereplő `-i` átadott érték fordítottja. A fájlrendszer tervezésekor ezt mindenképpen vedd figyelembe, és döntsd el, hogy sebességre vagy tárkapacitásra akarsz optimalizálni. Gondold át, hogy szerinted mi lesz a fájlok átlagos mérete, és melyek lesznek a leggyakrabban használt fájlok. Arra is ügyelj, nehogy túl kevés fájlleíróra korlátozd magad. Valószínű, hogy – persze attól függően, hogy mire használod a gépet – hosszú távon jóval több fájlod lesz, mint eredetileg gondoltad, ezért ne légy túl szűkmarkú. Ami az ***ext2*** és az ***ext3*** közötti teljesítménykülönbséget illeti, a naplózó fájlrendszerek mindig nagyobb terheléssel működnek, ám a sebességkülönbség elenyésző, különösen, ha a naplók meglétének előnyeivel vetjük össze.

Timothy Hamlin,
➲ *thamlin@nmt.edu*

2) Sem a ***Red Hat 9***-hez, sem a ***Red Hat 6.2***-höz nincs már támogatás, vagyis nem adnak ki hozzájuk biztonsági frissítéseket. Az utód, a ***Fedora*** futtatásához legalább ***Pentium*** processzor kell. Olyan terjesztést kell telepítened, amely a ***Pentiumnál*** régebbi processzorokat is támogatja – például ***Gentoot*** vagy ***Debiant***. Ne feledkezz el a biztonsági frissítésekről sem.
Tulajdonképpen mindegy, hogy mit teszel fel, ezek a gépek egy korszerű munkakörnyezet futtatásához túlságosan lassúak. Webkiszolgálónak, nyomtatókiszolgálónak, tűzfalnak, esetleg tanulási célra tudod használni őket.

Don Marti,
➲ *dmarti@ssc.com*

Régi Red Hat

Gondjaim vannak a ***Red Hat 7.2*** telepítésével egy 133 MHz-es PC-n, amit ***Smoothwall*** proxyként használok. A telepítés sikeresen megvolt, ám amikor újraindult a gép és megpróbáltam bejelentkezni, valami olyan üzenetet, kaptam, hogy `error in service mode`. Nehéz pontosan megmondani, mert csak egy pillanatra villan fel a képernyőn, aztán azonnal visszadob a bejelentkezési képernyőre. Ellenőriztem a fájlrendszert, a ***Bash*** telepítve van, valamint a környezeti elérési út is helyesen van beállítva. Valami mégis biztosan rossz, mert nem tudok bejelentkezni. Van valami ötletetek, hogy mi lehet a hiba oka, vagy – ami még jobb volna – tudjátok, mi lehet a megoldás? Nagyon megköszönném a segítségeteket.

Jeff,
➲ *jloyd1@comcast.net*

Amikor a rendszer elindult és megjelenítette a bejelentkezési képernyőt, nyomd le és tartsd lenyomva a CTRL és az ALT gombot, majd nyomd meg az F1 gombot. Ekkor kapsz egy parancssort. Be kell tudnod jelentkezni, mint root felhasználó. A 1-6-os szá-

mú konzolok között az ALT-F1 - ALT-F6 billentyűkombinációkkal tudsz váltogatni, az F7 pedig egy grafikus képernyő. Miközben a konzolok között lépkedsz, további részleteket is találsz a hibaüzenetről és/vagy a hozzá vezető eseményekről. Bejelentkezés után nézd át a */var/log/messages* fájlt és a */var/log* könyvtárban lévő egyéb naplófájlokat. Ennyivel már el tudsz indulni.

Usman S. Ansari,
➲ *usmannsari@yahoo.com*

Grafikus felületen próbálsz bejelentkezni? Ha igen, próbáld letiltani a grafikus felületet. Ehhez át kell írnod a */etc/inittab* fájlt, és 5-ös helyett 3-as futási szintet kell megadnod. A következő sort:

```
x:5:respawn:/etc/X11/prefdm -nodaemon
```

erre írd át:

```
x:3:respawn:/etc/X11/prefdm -nodaemon
```

vagy a rendszertöltőn keresztül adj meg ideiglenes beállításokat. Ha nem *xdm*-et futtatsz, akkor vizsgáld át a naplófájlokat, és keress hibajelzéseket. Külön felhívom a figyelmed a */var/log/messages* és a */var/log/secure* fájlra, illetve X használatakor az X-naplókat is át kell futni.

Timothy Hamlin,
➲ *thamlin@nmt.edu*

Melyik terjesztést?

Ostoba kérdésnek tűnhet, de azon gondolkodom, hogy 80 GB-os merevlemezemre második operációs rendszerként felteszem a *Linuxot*. Elsősorban multimédiás dolgokra, szövegszerkesztésre, film- és zenelejátszásra szeretném használni, ugyanis úgy hallottam, a *Linux* elég hatékonyan bánik az erőforrásokkal. A *Windowst* főleg játékra fogom megtartani. A gépemben *Athlon 64 3500+* processzor van, ezért a 64 bites adottságok kihasználására alkalmas rendszert keresek. Tudtok olyan terjesztést ajánlani, amivel a legjobban ki tudom használni a gépem 64 bites képességeit, és könnyű alatta médiafájlokat lejátszani, webezni stb.? Néztem a *Mandrake Linuxot*, de elég sok rosszat hallottam az *AMD64* processzorokra készült változatáról. Köszönöm, hogy időt szántok rám, várom válaszotokat.

Derek Allen,
➲ *sock_ferret@hotmail.com*

Lehet, hogy megköveznek érte, de szerintem ismerkedj meg a *Gentooval (www.gentoo.org)*, vele szinte minden géptípusból a lehető legtöbbet tudod kihozni, ugyanis képes arra, hogy telepítés közben natívan fordítsa le az összes csomagot. Ezt sokszor a *Gentoo* hátrányaként említik, mert a folyamat eltarthat egy ideig. Ugyanakkor a *Gentoo* csapat rengeteget dolgozott azon, hogy a lehető legtöbb géptípushoz készítsen bináris kiadásokat, köztük 64 bites gépekhez is, így ez a gond mára jelentéktelenné vált.
A *Gentoo* telepítésének folyamata rémálom, és bár a fejlesztők már elkezdték egy újabb telepítő összeállítását, nem biztos, hogy nem lesz ijesztő számodra, amit látni fogsz, amikor a géped megkezdi a betöltését. Ha mást szeretnél, akkor a *Red Hat* és a *Novell/SuSE* terjesztések is jó választásnak tűnnek. Mindkettőből vannak natív fordítások, és telepítőjük is könnyen kezelhető, áttekinthető. Ha teljesen szabad terjesztést akarsz, akkor a *Debiant* válaszd, fejlesztői az *AMD64*-es változatát az i386 után a legteljesebb átültetésnek nevezik. Az említett terjesztések mindegyike rendelkezik olyan csomagkezelővel, amellyel a rendszert naprakészen tudod tartani, valamint könnyedén tudsz új alkalmazásokat, például médialejátszókat telepíteni – vagy éppen be tudod szerezni a szükséges kodekeket.

Chad Robinson,
➲ *chad@lucubration.com*

A kezdőlap felkutatása

A *Red Hat 9.0*-s változatát futtatom 2.4.20-8-as rendszermaggal, és a terjesztéshez tartozó Apache kiszolgálóm használom. Amikor belépek a kiszolgálóra, egy tesztoldalt látok. A honlapom a */var/local/www/html* könyvtárban van, ahogy azt javasolják. Azt mondták, hogy a tesztoldalt cseréljem fel a honlapommal. Meg tudjátok mondani, hogy melyik fájlt kell átírnom ehhez? Kinyomtattam a *httpd.conf* fájl mind a 15 oldalát, és napokon keresztül tanulmányoztam, de nem jutottam semmire.

George Robertson,
➲ *grobertson29@earthlink.net*

A *Red Hat 9* alapértelmezett Apache-telepítésében a tesztoldal, azt hiszem, a */var/www/html/index.html*. Ha tehát le szeretnéd cserélni, akkor készíts róla egy biztonsági másolatot, majd tedd a helyére a saját fájlodat.

Timothy Hamlin,
➲ *thamlin@nmt.edu*

Az *Apache* beállító fájljában a `DocumentRoot` sort kell keresned. Az itt szereplő könyvtárban található a honlapod. Most tekints a `DirectoryIndex` sorra, ebben a fájl lehetséges nevei láthatók. Mielőtt azonban túlságosan sokat dolgoznál a rendszerrel, frissítsd a terjesztést a legújabb biztonsági foltokkal. A *Red Hat 9*-hez 2004. április 30-ig adtak ki biztonsági frissítéseket.
Most van a *Red Hat Múzeum Hét*, vagy mi?

Don Marti,
➲ *dmarti@ssc.com*

Távfelügyelet

Windows kiszolgálókat már jó ideje kezelek *VPN*-kapcsolaton keresztül. A linuxos rendszerek felügyeletére is van hasonló megoldás? Azt értem, hogy *VPN*-en keresztül be tudok jutni a linuxos rendszerekre, de milyen megoldást javasoltok a távelérésre és a felügyeleti teendők elvégzésére? Esetleg valamilyen könyvet tudtok ajánlani a témával kapcsolatban?

Ric Jones,
➲ *rictjones@wideopenwest.com*

A linuxos rendszerek távfelügyeletének hagyományos eszköze az *OpenSSH (www.openssh.com)*. Minden komolyabb terjesztésnek alapvető része, és titkosított megoldást biztosít parancsok futtatására és fájlok továbbítására, *VPN*-kapcsolat létesítése nélkül. Ha mégis *VPN*-t szeretnél, *Mick Bauer* következő írása kiválóan összefoglalja a témát: *www.linuxjournal.com/article/7881*.

Don Marti,
➲ *dmarti@ssc.com*

Intranet DNS

Az intranetem számára próbálok beállítani egy *bind* kiszolgálót. Van egy itthoni kábelmodemes forgalomirányítóm, ez játssza a *DHCP* kiszolgáló szerepét. Azt szeretném, ha lenne egy intranetes névterem a magán IP-címek feloldására, az internetes *DNS* kérések pedig az internetszolgáltató *DNS* kiszolgálói felé továbbítódnának. Odáig eljutottam, hogy a kiszolgáló válaszoljon a címrekordkérésekre (`ls -t`), de az egyes állomásnevek IP-címét nem adja vissza.

A gyökérzóna visszamutat ugyanazon gép *bind* kiszolgálójára. Beállítottam az *ort.cloud* tartományzónát, ez tartalmazza a *bind* kiszolgáló gazdagépét, a forgalomirányító IP-címét, továbbá a hálózati állomások IP-cím - állomásnév hozzárendeléseit és a kanonikus név - IP-cím leképezéseket. Egy másik zóna felelős a név - IP-cím és a kanonikus név - IP-cím összerendelésekért. Nem vagyok biztos abban, hogy a kettősség szükséges vagy sem, de egyelőre úgy tűnik, hogy működő megoldást ad.

Jeff,
➲ *jloyd1@comcast.net*

A *DNS* beállításával kapcsolatban talán a legjobb információforrás a *DNS-HOWTO*, ami a *www.tldp.org/HOWTO/DNS-HOWTO.html* címen érhető el. A leírás szerzője *Nicolai Langfeldt*, ő egy *DNS and Bind* című könyvet is írt, ami további részleteket és példákat is tartalmaz. Nekem is hasonló rendszerem van, mint aminek az összeállításával próbálkozol: belső *DNS* szolgálja ki a helyi, magánjellegű kéréseket, a külső feloldásokat pedig külső kiszolgálóhoz fordulva végzi el. Ha jól emlékszem – nem tegnap volt, hogy telepítettem – a *Google*-on a *„caching only nameserver”* (csak névkiszolgáló gyorsítótárazása) szövegre keresve jó pár példát és beállítást találtam.

Timothy Hamlin,
➲ *thamlin@nmt.edu*

Nem szabványos illesztőprogram összeomlik az új rendszermaggal

Egy ideig haboztam, hogy kérjek-e segítséget tőletek, de egyszerűen nincs ötletem, hogyan oldhatnám meg a gondomat. *Slackware 10.0* terjesztést használok, 2.6.9-es rendszermaggal és 3.3.4-es fordítóval, a rendszerindítást CD-lemezről, *isolinuxszal* végzem. A gond az, hogy az *Intel 536EP* modemlapkája *Linux* alatt nem támogatott. Az *Intel* által adott forráskód (*Intel-536ep-4.69-5.4.src.rpm*) rendben van, a modem működik is. Amikor az új rendszermagot használom, külön kell lefordítanom. A rendszerindítási folyamat során mindig az `Intel536: module license 'Proprietary' taints kernel` üzenetet kapom, de a modem működik. *KPPP*-t használok *KDE 3.2* alatt. Amikor a 2.6.10-es rendszermag kijött, megfoltoztam a saját gépem rendszermagját, lefordítottam ugyanazzal a *.config* fájllal, és persze újrafordítottam az 536ep kódját is, de a modem elnémult. Nem indul, nincs várakozás OK-ra az `ATZ` után, és nincs tárcsahang sem. Természetesen a régi, 2.6.9-es rendszermag még megvan, és azzal megy is minden. Örömmel venném, ha bármi segítséget, megjegyzést vagy egyéb adalékot kapnék tőletek a probléma megoldásához.

Werner Gerstmann,
➲ *WGerstmann@web.de*

Abban reménykedsz, hogy egy a fő rendszermagfába nem tartozó illesztőprogram hosszú távon is működőképes lesz a gépeden. A valóság azonban az, hogy a rendszermag *API*-jai a hibajavítások, a biztonsági foltozások és a szolgáltatások továbbfejlesztése nyomán folyamatosan változnak, ezért szinte biztos, hogy az illesztőprogram nem vagy nem sokáig fog működni.

A *www.kroah.com/log/linux/stable_api_nonsense.html* oldalon utánaolvashatsz, hogy a *Linux* rendszermagnak miért nincs stabil belső *API*-ja. Javaslom, hogy vedd fel a kapcsolatot az illesztőprogram írójával, és tőle kérj segítséget, ugyanis ő az a személy, aki a legjobban ismeri a kódot.

Greg Kroah-Hartman,
➲ *greg@kroah.com*

Linux Journal 2005. április, 132. szám

SugarCRM a gyakorlatban

Interjú Szigetvári Csabával, a program magyarítását végezte

A SugarCRM-nél akárcsak minden más, szabad szoftver esetén mindenki könnyen hozzáférhet a program dokumentációjához, és szabadon olvasgathatja, tesztelheti a szoftvert mind otthoni, mind pedig éles környezetben. Ez az egyik leglényegesebb érv, ami a szabad szoftverek mellett szól.

Nem csak egy demó programot kapunk egy-két hétre, korlátozott képességekkel, hanem magát a teljes értékű szoftvert tudjuk kipróbálni a bevezetés előtt. Így vagy még időben fény derül arra, hogy a szoftver nem felel meg a céljainknak, vagy ingyen jutottunk egy kiváló megoldáshoz.
Egy zárt forráskódú, kereskedelmi szoftver pedig, ha mégsem tudja a beígért funkciókat megvalósítani, amit például a demóból cselesen kihagytak, akkor jogosan becsapva érezhetjük magunkat, még ha esetleg később sikerül is visszaszereznünk a szoftver árát. Úgy gondoltam, hogy hasznos lenne utánajárni, annak, hogy mik a tapasztalatok a ***SugarCRM*** használatával. ***Szigetvári Csaba*** fordította magyarra, majd több helyre be is vezette a ***SugarCRM***-et, így őt kérdeztem arról, hogy milyen tapasztalatai vannak a szoftver használatával kapcsolatban.

– ***A SugarCRM-ről szóló cikk készítése során nagyon sok különböző leírást találtam arról, hogy mik a CRM rendszerek, illetve hogy milyennek is kell lennie egy CRM rendszernek. Neked mi erről a véleményed, milyen alapvető funkciókkal kell rendelkeznie egy jó CRM szoftvernek?***
– A ***CRM*** betűrövidítés az angol ***„Customer Relationship Management”*** kifejezésből származik, amit hozzávetőlegesen ügyfélkapcsolati ügyintézésként lehet fordítani. Olyan folyamatokra és módszertani alkalmazásokra utal, amelyek az ügyfél igényeinek, elvárásainak és szokásainak elemzésével igyekszenek megerősíteni az ügyfélkapcsolatokat. A ***CRM*** rendszerek általános célja az ügyfelek, az értékesítés, a marketing hatékonyságának, piaci folyamatok és visszajelzések adatainak kezelése és kiértékelése. A ***CRM*** rendszer segít az üzleti vállalkozásnak a műszaki lehetőségek kiaknázásában, valamint a meglévő és potenciális ügyfelek helyes megítélésben. Főbb összetevői lehetnek:
  – kapcsolattartás adatainak kezelése (ügyfél adatok, döntéshozatal hierarchiája, lehetséges üzletek várható nagyságrendje)
  – üzleti lehetőségek elemzése (folyamatok becslése, döntéshozatalt segítő előrejelzések)
  – kommunikációs eszközök (web-alapú ügyfélkapcsolat, e-mail és automatizált e-mail visszacsatolások, bejövő hívások nyilvántartása, kampánymenedzsment, honlap-analitika, kérdőívek)
  – szerződés nyilvántartások (szerződések, szerződés tervezetek, szándéknyilatkozatok tárolása)

Az elvárások ***CRM*** rendszerekkel szemben:
– ügyfélszolgálat szintjének javítása
– növelni az ügyfélforgalmat
– új ügyfelek szerzése
– termékértékesítés hatékonyságának növelése
– az üzletkötés előkészítési időtartamának csökkentése
– marketing és disztribúciós folyamatok áttekinthetővé tétele

A ***CRM*** rendszerek a csoportos munkát elősegítő és koordináló úgynevezett ***Workgroup*** szoftverek egy speciális ága. Sok tekintetben hasonlóak az ***BPM*** rövidítéssel jelölt ***„Business Process Management”*** típusú alkalmazásokhoz, amelyek az üzleti folyamatok általános meghatározását célozzák. A ***CRM*** rendszerek azonban sokkal konkrétabban egy jól meghatározott üzleti részterületre összpontosítanak.

– ***A következő kérdés azt hiszem meglehetősen magától értetődő: miért éppen a SugarCRM mellett tetted le a voksodat, illetve miért pont ezt a szoftvert fordítottad le? Mik voltak a leglényegesebb érvek és ellenérvek?***
– A SugarCRM 2004 őszén került a látótérbe, és körülbelül 2 hónapos helyi tesztelés után elég stabilnak bizonyult a felvállalt feladat ellátására. Ezt követően három olyan

cégnél került bevezetésre, ahol rendszergazdai tevékenység keretében érintve vagyok. Valamennyi cég kevesebb mint 5-20 munkatárssal dolgozik, ami azt jelenti, hogy szükség van a csoportos munka szervezését segítő szoftverre, viszont a LotusNotes és hasonló jellegű termékek minden tekintetben túllőnek a célon.
A SugarCRM-nél számos dolgot említhetek előnyként. ***AMP (Apache/MySQL/PHP)*** alapon fut, így csupán a szerver-telepítés igényel kiemelt figyelmet. Ugyanakkor platformtól függetlenül alkalmazható a felhasználók gépein (így egy ***Apple iBook*** éppúgy része lehet a csoportmunkának, mint a ***Windows*** és ***Linux*** felhasználói gépek). Forráskód tekintetében szabad szoftver, így nem alakul ki olyan függőség egy gyártó felé, amelynek hosszú távon nehezen kalkulálható anyagi és szervezeti vonzatai lehetnek (ha túl nagy a gyártó, akkor alig vehető rá egyedi kiegészítésekre; ha túl kicsi, akkor fennáll a veszély, hogy nem tudja fenntartani a termék folyamatos fejlesztését). Ez azt is jelenti, hogy egyedi kiegészítő fejlesztést önerőből is eszközölhetünk, ha a cég működése az általánostól eltérő követelményeket támaszt (az egyik felhasználó pl. egy utazási iroda, ahol hosszú távon szükség lesz bizonyos célirányú kiegészítésekre).
Az adatmentések, illetve adatok export/import funkciói a ***MySQL*** adatbázisból problémamentes folyamatok.
A ***SugarCRM*** működtetéséhez a szóban forgó létszámok mellett bőven megfelelnek szervergépnek azok a régi elfekvő ***Pentium I*** és ***II***-es gépek, amelyeken ***Windows*** rendszer már nem futtatható: a ***Windows98*** biztonsági támogatása teljesen megszűnt, A ***Win2k/NT*** esetében is hamarosan ez lesz a helyzet, a ***WinXP*** pedig keservesen köhög ezeken a gépeken. Ugyanezek a gépek ***Linux*** telepítéssel még akkor is ütemesek, ha ***X11*** fut rajta, a ***SugarCRM***-nek pedig szerverként a terminálos üzemmód is megteszi.
Ár tekintetében szabad szoftver, ami leginkább azért lényeges, mert 30-napos próbaverziókkal nem lehet egy cég keretében tesztelni szoftvert, ugyanakkor zsákbamacskára sem érdemes költeni. A ***SugarCRM*** mögött egy kommersz vállalkozás áll, ha tehát valaha adódna olyan helyzet, amikor nagyon gyorsan és megfelelő minőségű kiegészítő megoldásra volna szükség (ilyenkor általában a pénztárca is lazul), akkor van közvetlenül kihez fordulni. Ha megszűnne ez a cég vagy profilt váltana, akkor még mindig megmaradnak a szabad forráskód előnyei. Végül a munkafelület funkciói félreérthetetlenek, így a gyakorlatban nem igényel különösebb oktatást, betanítást.

– ***Nyilván szembekerültetek több nehézséggel is, például nem volt magyar fordítás a termékhez, vannak-e más olyan hiányosságai a szoftvernek, melyek nehézséget okoztak a bevezetés, használat során?***
– Magyar vállalatnál nem csupán illik, de többnyire célszerű is magyar felhasználói felülettel működő szoftver alkalmazása. Ez abban az adott pillanatban nem állt rendelkezésre, de felvállalható idő- és költségráfordítással megoldhatónak bizonyult.
A hozzáférési jogosultságok korlátozottak. A felhasználók adminisztrálásától eltekintve minden felhasználó bármilyen adatot felvezethet és törölhet. A belső levelezést is minden bejelentkezett felhasználó láthatja, nem csupán az akinek szól. Ebben a formában a belső levelezés inkább a munka kiosztására korlátozódik. Ennél valamivel árnyaltabb megoldás hasznos lenne. Ez az igény tudtommal a ***SugarCRM*** fejlesztői felé a kapcsolódó fórumokon keresztül már megfogalmazódott, de a jelenlegi ütemtervben még nem szerepel (az aktuális helyzetfelméréshez érdemes nyomon követni a fórumbeszélgetéseket). Jelenleg úgy küszöböltük át a helyzetet, hogy az egyik cégnél többször telepítettük fel a ***SugarCRM***-et, így minden munkacsoport a saját adatbázisával dolgozik. Ennek hátránya, hogy az átfedő adatokat többször kell felvezetni, külön-külön az egyes adatbázisokba.

– ***Sok helyen olvashatjuk, hogy a CRM rendszerek növelik a hatékonyságot, és ezáltal esetleg plusz bevételekhez is juthat a vállalat, ahol megfelelően használnak egy CRM szoftvet, mi a helyzet azoknál a cégeknél, ahová te vezetted be SugarCRM-et? Érezhető-e már valamilyen pozitív változás?***
– A ***SugarCRM*** eddig jól teljesíti az elvárásokat, de véglegesen nem szeretnék még ítéletet mondani róla, mert a hatékonysága akkor lesz csak bizonyítható, ha ténylegesen éltető eleme lesz a cégműködésnek. Ennek pedig az a feltétele, hogy napi használatú eszközzé váljon a munkatársaknak és (ami fontosabb) a főnököknek egyaránt. Ha a főnök használja, akkor az alkalmazottak is használni fogják.

– ***Három bevezetés után, akkor nyilván van már tapasztalatod arról, hogy hogyan érdemes elkezdeni a SugarCRM bevezetését?***
– Első lépésben egységesen ügyféladatokat és címlistákat vezetünk fel. Ez az alap. Ha első reflexként a ***CRM*** adatbázisban keresnek a munkatársak adatokat, akkor idővel a többi lehetőséget is kezdik kiaknázni.
Második lépés a belső információmozgás (belső levelezés) átirányítása a ***CRM*** keretein belülre. Ez kis létszámú cégnél nehézkes, mert egyszerűbb átmenni a másik irodai szobába megbeszélni valamit, vagy telefonon gyorsan átszólni. A belső levelezés tényleges célja az írásos folyamatdokumentáció, az ügyek utólagos nyomonkövethetősége. Ahol anyagi vonzata van döntéseknek, ott a felelősségeket is egyértelműen kell látni. Ennek a lépésnek a teljesítése tulajdonképpen nem a ***CRM*** működésének függvénye, hanem a munkaszervezésé. Az ilyesmi e-mail útján is megoldható, de felesleges egy belső használatú üzenetet világ körüli útra indítani ahhoz, hogy a másik szobába megérkezzen – mellesleg az adatbiztonsági elvárásoknak is jobban eleget tesz, ha csak az intraneten belül mozog.
Harmadik lépés lenne a ***CRM*** rendszer használata eredménykimutatásokra. A ***SugarCRM*** nyilván nem egy ügyviteli vagy számviteli rendszer, de megfelelő felhasználói ambíciókkal költségnem, költséghely és költségviselő jellegű összefüggések grafikonos kimutatására már a jelenlegi változatában is alkalmas. A ***SugarCRM*** alapvetően egy pénzügyi szemléletet tükröző logikai keretrendszer.

Az interjút készítette Horváth Ernő

Tér-idő feldolgozás linuxos stílusban

Ha új generációs vezeték nélküli kommunikációs készülékeket szeretnénk fejleszteni, akkor szükségünk lesz FPGA fejlesztői eszközökre, egy fürtre a szimulációkhoz, valamint egy beágyazott operációs rendszerre a mintapéldányokhoz. A Linux mindegyik célra megfelel.

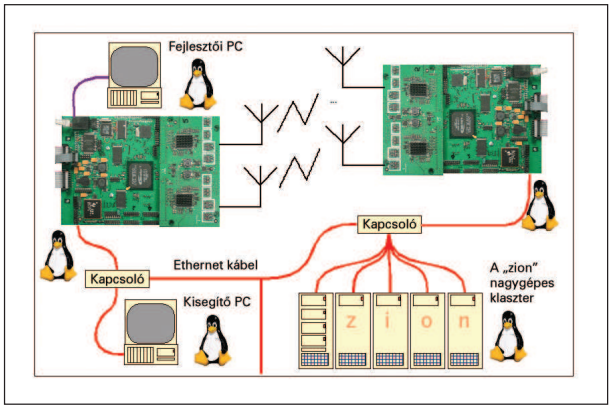
Munkahelyemen, az új-zélandi ***Christchurchben*** lévő ***Tait Electronics Group Researchnél*** az elmúlt időszakban szép csendesen kidolgoztunk egy fejlett vezeték nélküli hálózati megoldást, amit ***tér-idő (space-time, ST)*** feldolgozásnak neveztünk el. Az ***ST***-t sokan a vezeték nélküli rendszerek következő, a harmadik generáció utáni megoldásának tartják. A tér-idő lényege a mögöttes matematika, ebből fakadóan megvalósítása rendkívül bonyolult. Nem egy akadémiai kutató dolgozik ezen a területen, mégis talán mi vagyunk azok, akik ***ST*** tömbkutató (***STAR***) alaprendszerünkre építkezve el tudtuk készíteni a két első gyakorlati megvalósítást. Ezek az időinverziós ***tér-idő blokk-kódolás (time-reversal space-time block coding, TR-STBC)*** és a nyelvtörőnek is beillő nevű ***egyhordozós, adaptív többváltozós döntés-visszacsatolásos, kiegyenlített több bemenetű, több kimenetű (single carrier, adaptive multivariate decision feedback equalised multiple-in, multiple-out, SC-AMV-DFE-MIMO)*** kódolási séma. Kétségtelen, hogy eredményeinket jelentős mértékben a ***Linux*** kiváló teljesítményének és alkalmazkodókészségének köszönhetjük.

A továbbiakban szeretnénk elmondani, a ***Linux*** milyen – amúgy kulcsfontosságú – szerepet játszott matematikai megoldásaink kidolgozásában, illetve hogyan alkalmaztuk beágyazott feldolgozási és futási idejű vezérlési célokra. Olyan szerteágazó témákat fogunk érinteni, mint az ***üzenetátadó felület (message passing interface, MPI)***, a fürtözött adatfeldolgozás, a beágyazott ***Linux***, a ***PHP***, a héjparancsfájlok, a hálózati fájlrendszer (***NFS***), az ***SMB*** (kiszolgáló üzenetblokk) és a rendszermagmodulok használata és futtatása ***ARM***, ***Alpha*** és ***Intel*** géptípusokon. A fejlesztés részleteit át fogjuk ugrani, ám a rendszer jelenlegi működését ismertetni fogjuk, kiemelve azokat a valós életbeli előnyöket, amelyeket a Linux biztosított számunkra.

Alapesetben minden ***STAR*** eszköz egy többcsatornás adóegységből és egy többcsatornás vevőegységből áll, mindkettő egy megosztott ***LAN***-hoz csatlakozik, és mikrohullámú frekvenciákon bármilyen adattípust legfeljebb 200 Mbit/s sebességgel képes továbbítani. Jelenleg minden egységben 23 nyomtatott áramkör található, ezek megtervezése és legyártása 15 embernek egy évig tartott.

1. kódrészlet A fürtbeli csomópontok IP-címét SMB megosztáson tároló parancsfájl

```
#!/bin/sh
#
# Saját IP kiírása SMB megosztásra
#
# Központi SMB megosztás befűzése
smbmount //pelda/proba /mnt/proba
          ➥-o username=nev,password=jelszo >&
          ➥/dev/null#IP kiírása
#Az IP-cím lekérdezése az ifconfiggal
address/sbin/ifconfig | grep Bcast
  ➥| sed ´s/^.*addr://;s/Bcast.*//´ >
  ➥/mnt/proba/$HOSTNAME.ip
```



1. ábra Minden STAR alaprendszerben a kétirányú rádiós összeköttetés egy többcsatornás adót és egy többcsatornás vevőt használ

Az *1. ábrán* a rendszer egy megosztott ***Ethernet*** hálózathoz kétirányú rádiófrekvenciás összeköttetésként csatlakoztatva látható. Ez az összeállítás természetesen csak kísérleti célokat szolgál, a tényleges termékeknél az adó és a vevő között nem lesz megosztott ***Ethernet*** hálózat.

A rádiójelek feldolgozása a digitális kártyán történik, de nem az *ARM* processzor, hanem egy dedikált, ***egyedileg programozható kaputömb (field programmable gate array, FPGA)*** és egy ***digitális jelfeldolgozó processzor (DSP)*** végzi. Az *FPGA*-ban lévő kódot belső programnak nevezzük, esetében elég nehéz eldönteni, hogy program vagy vas alapú megvalósításról van-e szó. Ezzel a könnyed megoldással megengedhetjük magunknak, hogy mindenféle programozási és vastervezési szabványt figyelmen kívül hagyjunk.
A múlt év közepén, amikor a rendszert terveztük, a legnagyobb, leggyorsabb és legdrágább *FPGA*-t és *DSP*-t választottunk, amit csak be tudtunk szerezni, de azóta további két nagyméretű *FPGA*-t adtunk hozzá. Az *ARM* processzort alacsony szintű műveletekre nem használjuk, mert az *ST* feldolgozásokhoz több mint 50000 *MIPS* teljesítményre van szükségünk. Ilyen szintű bonyolultságnál még a leggyorsabb építőelemek is lassúnak bizonyulnak, ezért elég hamar eldöntöttük, hogy többprocesszoros működésre képes rendszerben gondolkodunk.
A *STAR* egységeket nagy sebességű, ***alacsony feszültségű, differenciális jelzéskezelésű (low-voltage differential signaling, LVDS)***, legfeljebb 1 Gbit/s sebességre képes kapcsolatokkal lehet bővíteni. Minden kártyán két darab ötcsatornás, kétirányú *LVDS* összeköttetés áll rendelkezésre a szomszédos kártyákkal történő kapcsolatteremtésre. Ugyanakkor minden kártya rendelkezik *Ethernet* kapcsolattal is. A nagy sebességű adatátvitel az *LVDS* összeköttetéseken, a vezérlőadatok továbbítása pedig az *Ethernet* kapcsolaton keresztül történik.

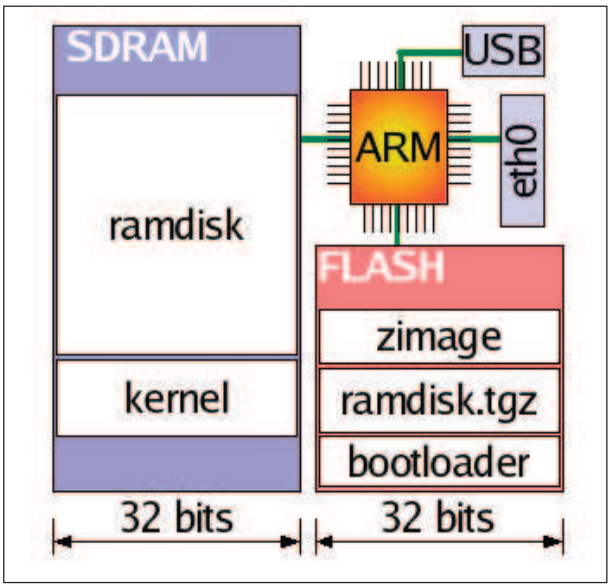
Fejlesztés

A feldolgozás túlnyomó része az *FPGA*-ban folyik, a kód *VHDL*-ben készült. *Linux* alá egyre több *VHDL* eszköz érhető el (lásd a széljegyzetet), mi az *Altera Quartust* használtuk. Rendszerünkben az algoritmusokat *GNU-Octave* alatt fejlesztettük ki, majd átültettük őket *VHDL* alá. Az *Octave* és a vele többé-kevésbé együttműködni képes *MATLAB Linux* és *Microsoft Windows* alá egyaránt elérhető, ám az *Octave*-nak van egy *MPI*-képes változata is, mely fürtökön is futtatható. Jó öreg *DEC Alpha* gépek fürtjére fordítottuk le, ezt egyébként *zionnak* neveztük el. Az *MPI-Octave* annak ellenére is nagyon szépen teljesített a *zionon*, hogy leggyorsabb processzora is csak 500 MHz-es volt.
A digitális kártyákhoz egy a *handhelds.org*-on talált *ARM Linux* eszközláncot használtuk a frissen foltozott 2.4.18-as rendszermagforrások lefordítására. *Russell King* az 1990-es évek elején *Acorn* számítógépekhez készített terjesztést, ekkor kezdett el dolgozni az *ARM Linuxon*. Jelenleg az *ARM* az egyik legjobb linuxos támogatású processzor, ami számunkra nagy könnyebbséget jelentett. Mindössze három napra volt szükségünk, hogy a *Linuxot* átültessük egyedi eszközeinkre, igaz, az *Ethernet* illesztőprogram beüzemelése még eltartott néhány napig. Az *ARM* a világon a legnagyobb példányszámban eladott processzorfajta, a *Linux* rajta való futtatásához elképesztő mennyiségű támogatást lehet szerezni. Ennek köszönhető, hogy a *Tait Electronics* úgy döntött, a jövőben is *ARM* processzorokat fog használni.
Amikor az *ARM Linux* már elindult, létrehoztunk neki egy *RAM*-lemezt. Az *ARM* 16 MB *SDRAM*-mal és 2 MB Flash

2. kódrészlet writeport.c
Egyszerű program 32 bites egész szám megadott fizikai memóriahelyre írására

```
#include <stdio.h>
#include <fcntl.h>      //az O_RDWR és az O_SYNC
                          ➥miatt szükséges
#include <sys/mman.h>   //a PROT_READ stb. miatt
                          ➥szükséges
#define GRAB_SIZE 1024UL
#define GRAB_MASK (GRAB_SIZE - 1)

int main(int argc, char **argv)
{
   void *grab_base, *virt_addr;
   unsigned int md, read_result, writeval;
   off_t phys_addr = strtoul(argv[1], 0, 0);
   /*memóriakezelő felület megnyitása*/
   if((md = open("/dev/mem", O_RDWR | O_SYNC))
   ➥== -1)
   {
      printf("HIBA - /dev/mem megnyitása
      ➥sikertelen\n");
      exit(1);
   }
   /* Lap leképezése a megadott fizikai címre*/
   if(grab_base = mmap(0, GRAB_SIZE, PROT_READ |
     PROT_WRITE, MAP_SHARED, md, phys_addr &
     ~GRAB_MASK), grab_base == (void *) -1)
   {
      printf("HIBA: leképezés sikertelen\n");
      exit(1);
   }
   /*írás a kért fizikai címre leképezett
   ➥képzetes memóriába*/
   *((unsigned long *)grab_base + (phys_addr &
     GRAB_MASK)) = strtoul(argv[2], 0, 0);
   /*memóriakezelő felület lezárása*/
   if(munmap(grab_base, GRAB_SIZE) == -1)
   {
      printf("HIBA - leképezés megszüntetése
      ➥sikertelen\n");
      exit(1);
   }
   close(md);
}
```

memóriával gazdálkodott, ezért úgy döntöttünk, hogy a tömörített rendszermag és a RAM-lemez egyaránt legfeljebb 1024 KB-os legyen, noha a tömörítetlen RAM-lemez mérete 4 MB. Mindkettőt a Flash memória tárolja, és külső beavatkozás nélküli rendszerindítást tesznek lehetővé.
A gyakran használt eszközöket, mint az `ls`, a `cd`, a `mount`, az `insmod` és a `ping` a ***BusyBoxból*** gyűjtöttük ki, a bejelentkezések és jelszavak kezelését pedig a ***TinyLoginra*** bíztuk. További segédprogramok végzik a memórialeképezett külső eszközök és a Flash kezelését, a ***TinyLogin*** szolgáltatásait használó ***telnet*** démon pedig a ***netkit-base*** csomagból szár-

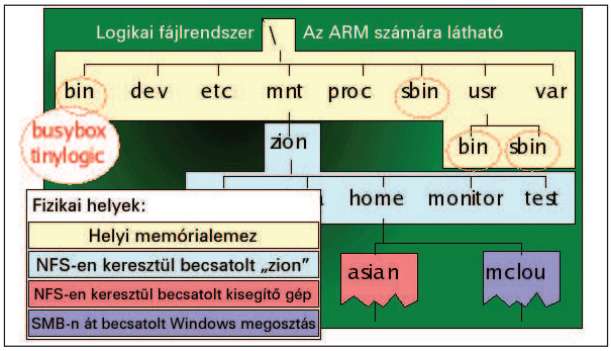


2. ábra Minden egység rendelkezik Flash alapú és RAM alapú fájlrendszerrel is

mazik. ***Linux-ARM*** fejlesztéseihez a ***Tait Electronics*** vásárolt egy ***MAC***-címtartományt az ***IEEE***-től, az egyes kártyák ***MAC***-címét a Flash memória tárolja.
Emellett jó néhány apróbb segédprogramot is készítettünk, illetve az ***Abyss*** webkiszolgálót is működésre bírtuk, de természetesen mindez egyszerre már nem fér el a Flash memóriában. A ***zionról*** ugyanakkor ***NFS***-en keresztül minden ***ARM***-on be tudunk fűzni egy könyvtárat, amivel több GB-nyi tárhelyhez jutunk. A 2. *ábrán* a fájlrendszerek elrendezése látható.
A ***zion*** alól ***SMB*** befűzésekkel a felhasználók megosztott meghajtóit is elérhetővé tettük, ezekhez az ***ARM*** kártyák szintén ***NFS***-en keresztül férhettek hozzá. Ha az ***ARM*** kártyákon futtattuk a webkiszolgálót, akkor végeredményként szerteágazó kapcsolatokkal rendelkező rendszert kaptunk.
A ***Windows***-felhasználók saját meghajtóikat webkiszolgálón keresztül is el tudták érni, és ez a webkiszolgáló olyan beágyazott ***ARM***-on futott, amely ***NFS***-en keresztül egy távoli fürtkiszolgálóhoz kapcsolódott.

Zion

2001 vége felé megvettük a kiöregedőben lévő ***DEC Alpha*** gépeket, és kíváncsian vártuk, mit tudunk kezdeni velük. Először is, módosítottuk a beépített rendszerindítót, és működésre bírtuk a 7.1-es ***Red Hatet***. Két fontosabb változtatást hajtottunk végre. Először kiválasztottunk egy mester gépet, majd hat darab ***SCSI*** merevlemezt és egy ***DVD-ROM***-meghajtót szereltünk bele. Öt darab lemezből egy ***RAID-5***-ös tömb lett (a ***raidtools*** segítségével), ez tárolja a kísérletek adatait, a fennmaradó lemez pedig a rendszerindítás és a helyreállítás céljait szolgálja.
A második módosítás az ***MPI*** telepítése volt az összes gépre. Bár maga a telepítés egyszerű volt, az összes IP-címet ***DHCP***-n keresztül kellett kiosztani, és ezzel bizony megszenvedtünk. Végül a ***zion*** nevű mester gépnek adtunk egy állandó IP-címet, a többi gépen pedig egy olyan indító parancsfájlt futtatunk, amely ***RPC*** használatával egy ***SMB*** megosztásra naplózza az egyes gépek címét.



3. ábra A fájlrendszerfa az egység oldaláról nézve

Amikor az ***MPI*** működőképes volt, letöltöttük a legújabb ***Octave*** változatot, és felraktuk rá az ***Octave-MPI*** foltot. Azóta az ***Octave-MPI*** fejlesztését a ***Transient Research*** vette át. (Lásd az internetes források részt.) ***MPI*** rendszerünket egy parancsfájl segítségével helyezzük üzembe, ez összegyűjti a már említett parancsfájl által elmentett IP-címeket, megpingeli őket, majd létrehoz egy ***rhosts*** fájlt. Amikor az ***rhosts*** dinamikus előállítása befejeződött, akkor a 4+1 gépen az alábbi parancsokkal indítjuk el az ***Octave-MPI***-t:

```
recon -v
lamboot
mpirun -v -c4 octave-mpi
```

Ha a rendszer életre kelt, az ***Octave*** terhelése eloszlik a fürt tagjain. Észrevettük, hogy az ***Alpha*** processzorok sokkal jobb lebegőpontos teljesítménnyel rendelkeznek, mint a ***Pentiumok***, ám az ***MPI***-üzenetek ***Ethernet*** feletti továbbítása lassította a fürtöt. Teljesítménytesteket ugyan nem végezünk, de az tény, hogy egy olyan szimuláció, amely egy 2 GHz-es Compaq PC-n néhány óra alatt fejeződik be, körülbelül 10 százalékkal gyorsabban futott az első, négy darab Alpha alapú gépből álló fürtön.
A ***GNU-Octave*** numerikus szimulációk elvégzésére kiváló eszköznek bizonyult. A `-traditional`, vagy, ha úgy tetszik, `-braindead` kapcsolóval futtatva a legtöbb ***MATLAB*** parancsfájlt képes értelmezni. Egyes esetekben az ***Octave*** jobb szolgáltatásokat nyújt, mint a ***MATLAB***; igaz, a ***MATLAB*** megjelenítési képességei fejlettebbek, mint az ***Octave*** által alapesetben használt ***gnuplot*** motorei.
Néhány mérnökünk inkább a windowsos fejlesztést kedvelte, ezért az ***MPI-Octave*** webes felületet is kapott. ***JavaScript*** alapú ***telnet*** ügyfelet használ, amelyet a ***zionon*** futó ***Apache*** és néhány háttérben dolgozó parancsfájl szolgáltat. A parancsfájlokat egy hálózati többszereplős játék motorjából vettük. A windowsos felhasználók megosztott meghajtóit a ***telnet*** ügyfél önműködően befűzi a ***zion*** fájlrendszerébe, majd ***telneten*** keresztül futtatja az ***Octave***-ot, valamint a rajzolást úgy állítja be, hogy a rajzok ***PNG*** formátumban a webkiszolgáló egyik könyvtárába kerüljenek, ahonnan böngészővel lehet megtekinteni őket. A felhasználók dönthetnek úgy is, hogy a rajzokat közvetlenül megosztott meghajtójukra mentik.
Hibakeresési célokra az ***ARM*** el tudja érni az ***FPGA*** nagyméretű hibakereső pufferét. Az ***FPGA***-t 32 bites, nagysebességű, aszinkron hozzáféréssel leképeztük a memóriába, így

felhasználói és rendszermag térből egyaránt elérhetővé vált az *ARM* számára. Nem meglepő, hogy a rendszermag térben fájlként elért karakteres illesztőprogram modult használunk. Ez löket módban akár 1024 adatszó nagysebességű továbbítására is alkalmas – természetesen folyamatos átvitelnél nincs ilyen jó sebessége –, miközben gondoskodik a jelzéskezelésről is. A felhasználói hozzáférést az `mmap` segítségével, a */dev/mem* felületen keresztül biztosítjuk – persze nem árt, ha nem feledkezünk meg a */dev/mem* létrehozásáról a beágyazott fájlrendszer alatt.

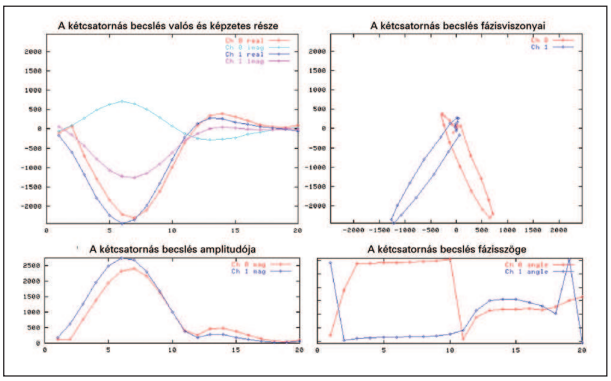
Mindezen eszközök segítségével fel tudjuk tölteni az ismert, az *Octave* által a *zionra* mentett tesztvektorokat a hibakereső pufferbe. Az *ARM* vezérlése alatt a hibakereső puffer kimenetét a tesztelés alatt álló blokk bemenetére irányítjuk, futtatjuk a rendszert néhány órajel erejéig, majd ugyancsak a hibakereső pufferben fogadjuk a kimenetet. Az eredményt az *Octave*-val elemezve el tudjuk dönteni, hogy a blokk működik-e.

A megjelenítést fontos tényezőnek tartottuk. Az első fontosabb lépések elvégzése után meghívtunk néhány embert, és megmutattuk nekik a rendszert. Hogy mit láttak? Zümmögő dobozokat, amelyeken néhány zöld *LED* jelezte, hogy minden rendben működik. Nem nagyon akartak lelkesedni, holott ez a műszaki megoldás nálunk működött elsőként a világon – kénytelen-kelletlen beláttuk tehát, hogy valami még hiányzik. Kitaláltuk, hogy – nagyjából valós időben, alkalmazásuk szerint – megjelenítjük a csatornamodelleket. A csatorna egy összetett útvonal a vevő és az adó között, különféle visszaverődésekkel, többszörös utakkal, szóródásokkal stb. Rendszerünk próbajelekkel vizsgálta a csatornát, mielőtt megkezdte volna az adatok továbbítását. A próbák révén egyfajta képet lehetett alkotni a csatornáról, és úgy döntöttünk, hogy ezt fogjuk megjeleníteni. Készítettünk egy *ARM* alapú parancsfájlt, ez időnként elindított egy programot, amely a vevőoldali *FPGA* hibakereső pufferéből kinyerte a csatornaadatokat, megformázta őket, és egy az *Octave* által is kezelhető *.mat* fájlba mentette a *zionra*. Az *Octave* a *zionon* nem interaktív módban futva rendszeresen kiolvasta a csatornaadatokat, elemezte őket, majd négy darab *PNG* képfájl formájában elkészítette a rajzokat. Ezeket a *zionon* futó webkiszolgáló egy *PHP* alapú weblappal jelenítette meg, négy másodperces frissítéssel. (*4. ábra*)

Támogatás

Linuxos digitális kártyáink normál esetben összetett feldolgozó algoritmusokat futtatnak, rádiójelek és *Ethernet* segítségével tartják a kapcsolatot más rendszerekkel. Mivel még laboratóriumi körülmények között is nehéz megmondani, hogy minden összetevő megfelelően működik-e, úgy döntöttünk, hogy a *Linux* erejére építve önellenőrző és önmegfigyelő megoldásokat készítünk. A megjelenítéshez hasonlóan itt is számos összetevőt kellett rugalmasan összekapcsolni.

A nem műszaki területen dolgozó felhasználók grafikus felületet akartak, amit *GTK*, *Tk*, *Qt* vagy valami hasonló eszközzel állíthattunk volna össze, mi azonban *PHP* alapú webes parancsfájlok használatát választottuk, így ugyanis a rendszer elérése tökéletesen gépfüggetlen. A *PHP*-t a *C* stílusú írásmódnak köszönhetően a *C* programozók is



4 ábra A csatornamódok megjelenítése (összetett útvonalak az adó és a vevő között)

könnyen tudják használni, és a felhasználók által megszokott és ismert webes felület is előnyt jelent. A legtöbb alkalmazás forráskódja 50 KB-nál kisebb, ami előnyös a hibakeresésnél, illetve ebből fakadóan több bizalmat helyezhetünk az egyes kódrészletekbe.

Az ilyen jellegű felhasználói felületeknek sajnos hátrányaik is vannak. Az egyik eset az, amikor valamilyen háttérbeli eseményre kell felhívni a felhasználó figyelmét, a másik pedig az, amikor a rendszernek közvetlen párbeszédet kell folytatnia a vassal. Az első kérdést folyamatosan frissülő üzenetek külön keretben való megjelenítésével lehet megoldani. A második gondot mi úgy hárítottuk el, hogy a *PHP*-vel a kapcsolatot fájl alapú felületen keresztül tartó, apró, alacsony szintű *C* programokat írtunk.

Nem mondom, szép szövevényes rendszerünk lett. Minden egység vezérlője egy *ARM* processzor, de a vezérlők vezérlője egy *PHP* parancsfájlokat kezelő webkiszolgáló. Önellenőrzés céljából a rendszer öt másodpercenként minden *ARM*-on lefuttat egy parancsfájlt, majd az eredményeket átadja a webkiszolgálónak. Az eredmények az egyes *ARM*-ok *Ethernet* csatolójának IP-címe szerint elnevezett fájlokba kerülnek. A figyelő parancsfájl feladata annak biztosítása is, hogy minden működő rendszer órája szinkronban legyen a *zionéval*. Az *NTP* használatát elvetettük, mert viszonylag nagyméretű kód kellett volna hozzá, és az órák együttállására csak a fájlrendszerek kezelésekor jelentkező hibák elkerülése miatt volt szükségünk. A *zion* egy parancsfájl és a `data` parancs segítségével az aktuális időt és dátumot öt másodpercenként egy fájlba írja; az *ARM*-ok ezt a fájl olvassák ki szintén öt másodpercenként, és tartalma alapján beállítják órájukat. Ezzel el tudjuk kerülni az időszinkronizálási gondokat, valamint biztosítani tudjuk a fájlok időbélyegeinek pontosságát. Egyben időzítőként is szolgál a több mint egy percet csúszó kártyák alapállapotba hozatalára.

Az önellenőrzésen túl a parancsfájl indításkor a *RAM*-lemezek és a rendszermagok változatszámát is lekérdezi és rögzíti egy állapotfájlba. A parancsfájl utolsó feladata a kártyákhoz egyedileg megadott parancsok végrehajtása. A parancsokat a *PHP* alapú weboldal szolgáltatja, ez szabja meg, hogy melyik *ARM* kártyának mit kell tennie. Az utasítások *PHP* alól héjparancsfájl formájában érkeznek, ezeket kell az IP-cím alapján kiválasztott kártyán lefuttatni.

Minden egység ugyanolyan rendszermaggal és *RAM*-lemezekkel dolgozik, ezek tárolása a helyi Flash memóriá-

ban történik, illetve a ***zionon*** lévő másolat alapján minden indításkor megtörténik sértetlenségük ellenőrzése. Bármilyen eltérésnél a rendszer újra bemásolja a Flash memóriába a megfelelő rendszermagot és ***RAM***-lemezt. Erre a célra egyedi Flash memóriakezelő eszközöket fejlesztettünk. A gyakorlatban tárolási rendellenességeket nem tapasztaltunk, ám ezzel a megoldással felhasználói beavatkozás nélkül tudjuk telepíteni a rendszermag és a ***RAM***-lemez újabb változatait.

A ***PHP*** alapú weboldalak az ***autoload HTML*** címke használatával öt másodpercenként frissülnek, természetesen csak akkor, ha legalább egy felhasználó megnyitotta böngészőjével az adott oldalt – egyébként lehetséges, hogy az oldalon szereplő adatokra éppen nincs is szükség. A vezetőknek szánt weboldalak szinte minden hasznos adatot elrejtenek, viszont jól néznek ki. A valódi felhasználóknak szánt oldalakon ellenben lehetőség nyílik az alsóbb szintek elérésére is, egészen az egyes kártyák működését irányító parancsfájlokig.

Hibakeresés és figyelés

Az ***ARM***-on futó linuxos program az ***FPGA*** által tárolt belső program segítségével kisméretű csomagokból álló adatfolyamot továbbít a vezeték nélküli összeköttetésen keresztül az ***FPGA*** hibakereső pufferébe. Ráérő idejében az ***ARM*** kibontja ezeket, és eltárolja őket a ***zionon***. A fájlokban önműködő eljárásokkal meg lehet keresni a hibákat (például a csupa nullából álló sorozatokat), és szükség esetén a weboldalakon keresztül riasztani lehet a felhasználókat.

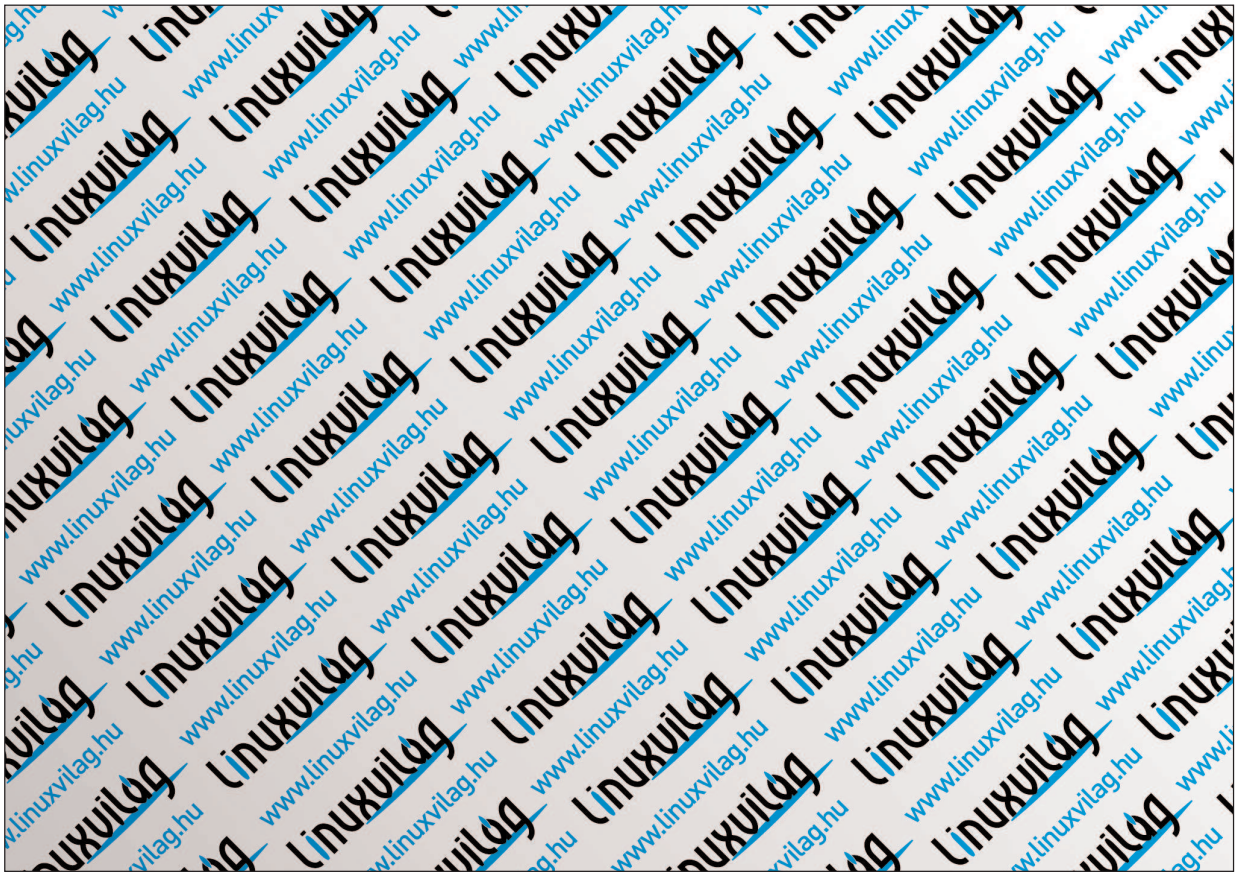
Összefoglalás

Legújabb tervünk az, hogy az elméleti kutatásokon túl termékek tervezésébe is belefogunk. Valószínűleg IP-csomagok küldéséről-fogadásáról lesz szó, és a termék vezérlését beágyazott ***Linuxra*** fogjuk bízni. Ennek nemcsak az az oka, hogy a fejlesztés során is a ***Linuxra*** támaszkodtunk, de az is, hogy számos kiváló lehetőséget biztosít az IP-csomagok kezelésére. A ***WinCE*** lassú és nagy, a ***VxWorks*** pedig drága és kevés protokollt támogat. A ***Tait Electronics*** nonprofit, az új-zélandi ***Christchurch*** alkalmazottait és közösségét segítő elektronikai egyesülés. ***Sir Angus Tait*** alapította, több mint 30 évvel ezelőtt. Mobil rádiós termékeinek 97 százalékát exportálja, értékesítései a világ több mint 200 országára terjednek ki.

Linux Journal 2004. szeptember, 125. szám

Ian McLoughlin 12 éve használja a Linuxot, szakterülete a jelfeldolgozás. Mielőtt kívándorolt volna Új-Zélandra, egyetemi előadó volt Szingapúrban, és mint az XSat műholdas program – indítását 2006-ra tervezik – vendégtudósa jelenleg is sokat tartózkodik ott.

Tom Scott a Mission Technologies Ltd. (www.missiontech.co.nz) igazgatója, elsősorban a dolgokhoz való lényegre törő, nyílt, szabatos és gyakorlatias hozzáállásáról ismert. Felesége és két gyermeke hittérítő.



Gyártófolyamatok automatizálása Linuxszal

Avagy hogyan kísérhetünk figyelemmel valós időben több gyártósort? A problémánkat saját magunknak oldottuk meg a Linux segítségével.

Egy termeléssel foglalkozó vállalat csak akkor termel hasznot, amikor működnek a gyártósorok, ezért létfontosságú, hogy a gyártószintről származó információk kellő időben rendelkezésre álljanak. A vállalatunk méretével együtt növekedett a bonyolultsága is, így rövid idő alatt kinőttük a termelés ellenőrzésére hivatott manuális, papír alapú módszereinket.

Cégünk, a ***Midwest Tool & Die*** elektronikus csatlakozóvégeket sajtol, és műanyag alkatrészeket formáz az autógyártás, elektronikai és fogyasztói ipar számára. A gyártófolyamatainkban rengeteg adat keletkezik. A nagysebességű préseink percenként akár 1200 alkatrész legyártására is képesek, és minden egyes darabnak megfelelőnek kell lennie. Minden gyártott alkatrésznek megvizsgáljuk a kritikus méreteit, folyamatosan figyelemmel kísérjük és ábrázoljuk a minőséget, az adatokat pedig a nyomon követhetőség érdekében archiváljuk.

Miért fontos az automatizálás?

A gyártófolyamataink továbbfejlesztéséhez szükségünk volt az összes adat kezelésére. A fő célunk az volt, hogy növeljük az üzemidőt és minél jobban megértsük az állásidő okait. Ezen felül reménykedtünk abban, hogy a költségeket is nyomon követhetővé és ellenőrizhetővé tehetjük, csökkenthetjük a papírmunkát és elkerülhetővé válik az emberi adatbevitelből származó hiba.

E célok alapján körvonalazódtak az új rendszerrel kapcsolatos elvárások. Elsődleges elvárás volt az adatok összegyűjtése a sokféle gépvezérlő egységből, érzékelőkből, automatikus vezérlőegységekből, a programozható logikai vezérlőkből (***PLC***) és az operátorként dolgozó munkatársaktól. A rendszernek megbízhatónak kellett lennie és képesnek arra, hogy a legnagyobb termelési sebesség mellett is összegyűjtse az adatokat. A következő elvárás az volt, hogy a rendszer meg is feleltesse az így begyűjtött adatokat, vagyis párbeszédképesnek kellett lennie a cég ***PostgreSQL*** adatbázisaival. A termelési adatok és a folyamat állapota a ***PostgreSQL***-nek adódik át megjelenítés és jelentéskészítés céljából.

Az új automatizáló rendszernek felhasználói felülettel is rendelkeznie kellett, amelyen a gépkezelők és a karbantartó személyzet naplózhatja a saját tevékenységét. Az állásidők és ezek okai rögzítésre kerülnének és továbbítódnának a vállalati adatbázis felé. Ezzel a megoldással kiváltható lenne a papír alapú naplózásra és manuális adatrögzítésre fordított munka.

1. ábra Két SmartPress gyártási automatizáló rendszer használat közben. Minden rendszer egy PC-ből, egy DAQ-kártyából, elektronikus leválasztó rendszerből, tartalék akkumulátorból és vonalkód-nyomtatóból áll.

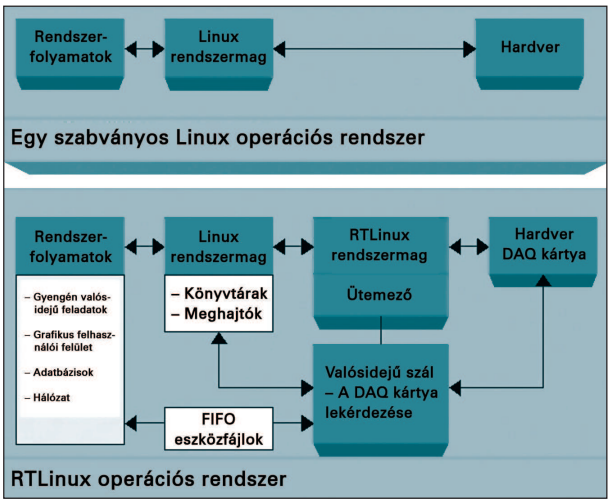
Végül a rendszernek rugalmasnak s könnyen frissíthetőnek kellett lennie továbbá alkalmasnak kellett lennie új gyártósorokra való átültetésre és a rendszerbemenetek megváltozásának kezelésére.

Miért a Linuxot választottuk?

A fent vázolt elvárások tükrében több megoldást is megvizsgáltunk. Az ipari ***PLC***-k megbízhatóan össze tudnák gyűjteni az adatokat, a hálózati megoldásaik azonban évtizedekre megrekedtek bizonyos nem szabványos, szabadalmilag védett eljárások szintjén. Az ***Ethernet*** kapcsolat ugyan elérhetővé vált, de az ilyen rendszerek drágák.

A felhasználói felület jellemzően a gyártófüggő megjelenítő hardverre készül. Minden gyártó kifejleszti a saját szabadalmazott fejlesztői felületét. Látható tehát, hogy az értékelés minden pontján megjelenik a gyártótól való függés.

A következő próbálkozásunk egy adatgyűjtő (***data acquisition, DAQ-***) kártyával felszerelt PC volt. Korábban egy táskagépet használtunk egy ***DAQ***-kártyával, ***Microsoft Windows***-t és ***Aglient VEE***-t. Ez az együttes jól működött az adatok gyors összegyűjtésének terén, ehhez csak kevés programozási munkára volt szükség. Ezzel a rendszerrel az adatok továbbítása az adatbázisunkhoz csak a ***Windows OLE***



2. ábra Az RTLinux-rendszerfolyam

felületén keresztül volt lehetséges. Írhattunk volna egyedi alkalmazásokat de a szabadalmazott felület a gyártóhoz kötött volna minket. A ***National Instruments*** szintén ajánlott teljes ***DAQ***-csomagot a PC-hez, de csak felár fejében.
Az igényeinknek legjobban megfelelő megoldás is PC-ből és DAQ-kártyából áll, az eltérés az ***RTLinux*** használata, amely egy ***Linuxra*** épülő valós idejű operációs rendszer. Ezzel korlátozhattuk a gyártóhoz való kötődést és ingyen létesíthettünk kapcsolatot a ***PostgreSQL*** adatbázissal és a ***TCP/IP*** hálózattal. A valós idejű operációs rendszer a ***PLC*** megbízhatóságát kínálta anélkül, hogy cserébe az összeköttetésekről le kellett volna mondanunk. Nem utolsó sorban pedig a felhasználó felületet is a saját tetszésünknek megfelelő nyelven készíthettük el. A nyílt forráskódú eszközök használatával rugalmas és könnyen frissíthető alkalmazásokat hozhattunk létre.

A SmartPress rendszer

A számítógépek, amelyeket az adatbegyűjtés és az adatok kezelésének feladatára választottunk, a manapság kapható gépekhez képest lassúak. Lehetőségünk adódott ugyanis a régi irodai gépeink újrahasznosítására a gyártási szinten. Ezek a 400 MHz-es ***Celeron*** processzoros gépek elég gyorsak ahhoz, hogy megfelelően elvégezzék a rájuk bízott feladatokat anélkül, hogy akadályoznák az adatok begyűjtésével kapcsolatos magas szintű elvárásainkat. Az általunk alkalmazott rendszerre induláskor a ***Red Hat 7.3*** rendszercsomag került a 2.4.18-as rendszermaggal.

Az RTLinux és a Linux rendszermagok

A rendszermag választja el a felhasználói szintű feladatokat a rendszer hardverétől. A szabványos ***Linux*** rendszermag minden egyes felhasználói kérés számára időszeleteket oszt ki és képes ezek lejártakor az adott feladat felfüggesztésére. Ez ahhoz vezethet, hogy a kritikus fontosságú feladatokban késést okozhatnak a rendszer nem kiemelt fontosságú folyamatai. Léteznek olyan parancsok, amelyekkel befolyásolhatjuk a ***Linux*** feladatütemező működését, bár a 2.4-es rendszermagban ezeknek a paramétereknek az átállítása sem eredményez igazi valós idejű rendszert. A 2.6 rendszermagban már fejlettebbek a valós-idejű képességek, de még ez sem elégíti ki egy ***szigorúan valósidejű rendszerrel*** (***hard real-time system***) szemben támasztott követelményeket.
Rengeteg nagyszerű kiadvány jelent már meg az ***RTLinux-szal*** kapcsolatban, sok ezek közül ***Michael Barbnovtól*** és ***Victor Yodaikentől*** származik, akik először valósították meg az ***RTLinuxot*** még 1996-ban, és azóta is fejlesztik. A ***Finite State Machine Labs, Inc. (FSMLabs)*** egy a tulajdonukban lévő szoftvercég, amely karbantartja a programot. Az évek során kidolgoztak olyan továbbfejlesztéseket, amelyeket az ***RTLinux*** kereskedelmi változataiba építettek be, de továbbra is biztosítanak egy ingyenes változatot ***RTLinux/Free*** néven, amely a ***GNU GPL*** és az ***Open RTLinux Patent Licence*** (***nyílt RTLinux szabadalmi licenc***) alapján használható. A projektünkhöz az ingyenes változatot használtuk, amelyhez nem jár támogatás az ***FSMLabs***-tól.
A ***Dinil Divakarannak*** köszönhető ***RTLinux HOWTO***-ban megtaláltuk az ***RTLinux*** telepítéséhez és futtatásához szükséges információ túlnyomó részét.

Az RTLinux-rendszerfolyam

Egy normál ***Linux*** rendszerben ha írunk egy függvényt, amely az adatgyűjtő kártyáról érkező adatokat meghatározott időnként beolvassa, nem kapunk kielégítő eredményt. Egy ilyen rendszer nem tudja garantálni a kapott ütemezési elsőbbséget. Amint a 2. *ábrán* látható, a szabványos ***Linux*** operációs rendszerben minden rendszerfolyamat a hardvertől elválasztva működik. Ez nem is lenne gond, ha az adatbeolvasó folyamatunk lenne az egyetlen, amit a rendszer éppen végrehajt. A mi projektünknek azonban felhasználói térben futó programra van szüksége, és garanciára, hogy az érzékelők bemeneteit minden ezredmásodpercben beolvashatja. A szigorúan valósidejű rendszerek biztosítani tudják, hogy az érzékelők bemeneti adatai egyszer sem vesznek el. Az ***RTLinux*** operációs rendszerben, amely szintén látható a 2. ábrán, a valós idejű feladat el van választva a többi rendszerfolyamattól, és modulként kerül megvalósításra. A modul közvetlen hozzáférést élvez a hardver és a ***DAQ***-kártya meghajtói felé, és a rendszer többi része felett is megkapja a szükséges prioritást. A modult egy adott feladat megoldására írják, amely megbízható eredményeket biztosít, és ezeket a ***FIFO*** meghajtó-fájlokon keresztül adja át a felhasználónak. A fejlesztők törekedtek arra, hogy a modul kódja minél egyszerűbb maradjon, csak azokat a feladatokat oldották meg vele, amelyeknek mindenképpen valós időben kell zajlaniuk. Az egyik ***FIFO*** meghajtó-fájlhoz hozzákapcsolva egy kezelőfüggvényt lehetőség nyílik arra, hogy az operátor kezelőprogramja ezen keresztül végezze az irányítást. Az ***RTLinuxnak*** ez a felépítése biztosítja, hogy a rendszermag másodlagos folyamatok futtatása miatt ne késleltethesse a fontos modulfeladatok végrehajtását.

Az adatgyűjtő szálak

Egyedül a digitális bemenet állapotát kell a szigorú valós idejű feltételekkel figyelnünk, ezért a valós idejű függvényünk kis méretű maradhatott. Az adatbemenetek lekérdezését megkönnyítette, hogy a ***United Electronics Industries*** a ***DAQ***-kártyáihoz ***RTLinux*** meghajtóprogramot is adott. Az ***ADLINK Technology, Inc. DAQ***-kártyáját szintén az ***RTLinuxhoz*** készült meghajtókkal teszteltük, a beállítással

1. lista Egy valós idejű modul kódjának váza

```
#include <pthread.h>
#include <rtl_fifo.h>
#include <rtl_core.h>
#include <rtl_time.h>
#include <rtl.h>
#include <rtl_fifo.h>

pthread_t thread_variable;

void thread_name(void)
{
    Struct Sched_param p;
    p.sched_priority = 1
    pthread_setschedparam(pthread_self(),
                          SCHED_FIFO, &p);
    pthread_make periodicnp(pthread_self(),
                            getrtime(),
                            ➥1000000);
    while(1) {
        // Real Time Task Code
        // Poll Data input lines, count low to
        // high transitions
        rtf_put() // Counts to be transferred by
                  ➥FIFO
        pthread_wait_np();
    }
}

int handler_function(){
    // Code tied to the handler FIFO
    // Variables for counting above are cleared
    // out
}

int init_module(void)
{
    ififo_status =  rtl_create(unsigned int fifo,
                               int size)
    pthread_create(&thread_variable, NULL,
                   thread_name, NULL);
    rtf_create_handler(FIFO_Number,
                       &handler_function)
}

int cleanup_module(void)
{
    rtf_destroy(unsigned int fifo)
    pthread_cancel(thrad_variable)
}
```

nem volt gondunk. Nem sok cég kínál ilyen szolgáltatást, jóllehet a ***Comedi Project*** a nyílt forrású meghajtók, segédprogramok és programkönyvtárak révén egy másik lehetőséget is kínál az adatgyűjtés megvalósítására.

A valós idejű feladatot egy betölthető modul formájában írtuk meg, amelynek legalább két függvénnyel kell rendelkeznie: Az egyik az `init_module` függvény, amelyet a modul rendszermagba történő beillesztése előtt hívunk meg, a másik a `cleanup_module`, amelyet közvetlenül az eltávolítás előtt hívunk (*1. lista*).

Miután a modul alapszerkezete elkészült, szükségünk volt a modulban egy szál létrehozására, amely a valós idejű feladatot végzi el. A szálat az `init_module` belsejében hoztuk létre, és úgy állítottuk be, hogy a megfelelő ***RTLinux*** prioritásokkal fusson. A megfelelő prioritás és futási sebesség beállítása a szál számára a valós idejű feladat létrehozásának fontos lépését jelentette.

A FIFO meghajtófájlok

A kiszámítható időzítésű futtatás megvalósításához szükség volt valós idejű memóriára az adatátvitelhez. A felhasználói szintű folyamatnak hozzá kellett férnie az összegyűjtött adatokhoz és a valós idejű folyamat vezérléséhez. A valós idejű ***FIFO***-k olyan várakozási sorok, ahonnan a ***Linux*** folyamatok adatokat olvashatnak ki és írhatnak bele vissza. A valós idejű ***FIFO*** meghajtók fordítása az ***RTLinux*** telepítésekor történik, a létrehozása pedig a valós idejű modulok inicializálásakor. Ekkor egy kezelőprogram hozható létre és köthető az egyik ***FIFO*** meghajtóhoz. A kezelőt úgy állíthatjuk be, hogy akkor fusson le, amikor a felhasználói felületről egy 1-es érték érkezik a kezelő ***FIFO***-ba mint a szükséges modul vezérlője. A modult a rendszermagba valós idejű folyamatként való telepítés céljából hoztuk létre. A valós idejű modul bonyolult része a keretrendszer létrehozása volt. A valós idejű feladat magától értetődő volt, bár igen hosszú, ezért ide csak a valós idejű modul vázát illesztettük be (1. lista). Láthatjuk, hogy milyen egyszerű a valós idejű elvárásokat megvalósító kód.

A felhasználói felület

A valós idejű folyamat üzembe helyezésének következő lépése az alkalmazás felhasználói felületének létrehozása volt. A programunkban a grafikus felület volt annak a számos feladatoknak az egyike, amelyeknek nem kellett szükségszerűen teljesíteniük a szigorú valós idejű kikötéseket. Nem kellett különösebben törődnünk a rendszererőforrásokkal, mivel a valós idejű modulunk a már megvalósított valós idejű környezetünkben fog futni. A választásunk a ***KDevelop IDE*** és a ***Trolltech Qt Designer*** grafikus felület készítő eszközére esett. A ***Qt Designer*** lehetővé tette egy olyan grafikus felület kifejlesztését, amely a ***KDevelopban*** elkészített program függvényeivel képes volt a jelekkel és csatolókkal megvalósított kapcsolattartásra. Ennek a két eszköznek az együttes használatával létrehozott felület tökéletesen megfelelt a programunk számára. Rövid időn belül képesek voltunk egy felhasználóbarát felület létrehozására.

A program két forrásból képes a rendszer számára információkat fogadni: a ***DAQ***-kártya felől jövő digitális bemenetről és a grafikus felületen keresztül a kezelőszemélyzettől. Ennek a két forrásnak az egyesítésére volt szükség ahhoz, hogy megőrizhessük az adatok sértetlenségét. Például a kezelő a munkafolyamathoz beírja a gyártandó alkatrész számát, a futás alatt összegyűjtött adatok pedig ehhez az alkatrészszámhoz rendelődnek. Ez helyettesíti a felhasználó korábbi kötelezettségét a nyilvántartások kézzel történő kitöltésére vonatkozóan.

Az adatok kezelése

Az adatok összegyűjtése csak a rendszer és a program első lépését jelentette. A legfontosabb az volt, hogy az információkat használhatóvá tegyük a cég minden osztálya számára. A tervezési, beszerzési és karbantartó osztály csak néhány példa azokra a helyekre, ahol ezeknek az információknak jó hasznát vehetik.

A *SmartPress* program az összegyűjtött adatokat egy *PostgreSQL* adatbázisban helyezte el. A *PostgreSQL* rendszer képezi a vállalati szintű háttéradatbázisunkat is, így a jövőben kifejlesztendő alkalmazás egész vállalati szinten láthatóvá teszi majd a *SmartPress* adatait.

Összegzés – „csináld magad" információtechnológia

A *SmartPress* rendszerünk a tesztelőszobából kikerült a termelési szintre. Visszatekintve elmondhatjuk, hogy sikerült létrehoznunk egy rugalmas gyártásellenőrző rendszert. A *Linux* használatával egy olyan bővíthető alkalmazást kaptunk, amely a cég igényeitől függően szabható testre. Lehetőség van a *SmartPress* új gyártófolyamatokra történő bevezetésére is. A rendszer egyszerűen frissíthető, a jövőben tervezzük is a program új rendszermagra történő átültetését. A frissítéshez valószínűleg a *Fedora* magot fogjuk *Linux*-alapként használni.

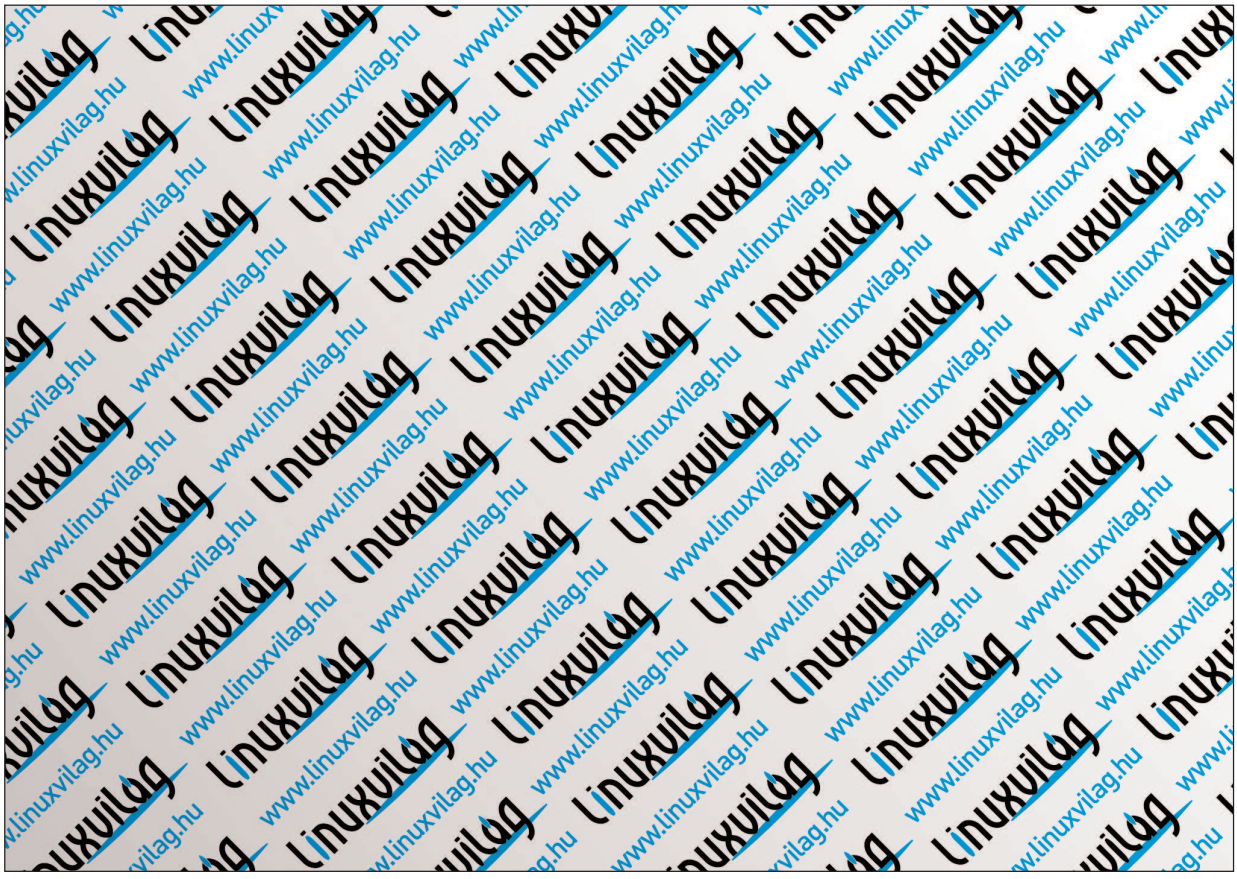
A *SmartPress* alacsony hardverköltségei fontos szempontot jelentenek számunkra, mivel minden egyes préssor számára egy-egy külön rendszert telepítünk. A költségeket személyi számítógépek és a *Linux* által támogatott *DAQ*-kártyák használatával tudtuk alacsony szinten tartani.

A programfejlesztésünk szintén olcsó volt, az idő, amelyet *Ryan Walsh* ezzel a projekttel töltött, körülbelül annyi, mint amennyit egy kereskedelmi vezérlőnyelv megtanulásával és az abban történő fejlesztéssel kellett volna eltöltenie. *Ryan* mostanra teljesen otthonosan mozog a rendszermag-modulok, a valós idejű operációs rendszerek, a *PostgreSQL* és a grafikus felületek fejlesztése terén, ezek a készségek pedig sokkalta használhatóbbak, mint egyetlen gyártó programozási nyelvének elsajátítása. Számunkra a „csináld magad" módszer választása azt eredményezte, hogy alacsonyabb költséggel, gyártóhoz való kötöttség nélkül és fejlesztői ismereteink gyarapításával sikerült elérnünk a céljainkat.

Linux Journal 2004. december, 128. szám

Craig Swanson (craig.swanson@slssolutions.net) hálózattervező és Linux tanácsadó az SLS Solutions cégnél. A Midwest Tool & Die cégnél Linux programok fejlesztésével foglalkozik. 1993 óta használ Linuxot.

Ryan Walsh (ryan.walsh@midwest-tool.com) a Midwest Tool & Die hálózati mérnöke. Szabadidejében repülőgépekből kiugrálva élvezi a szabadesés élményét.



MyHDL: egy Python alapú hardverleíró nyelv

Az alkatrésztervezés végre belépett a XXI. századba. Ez az új eszköz a Python olvasható kódját ötvözi az extrém programozás tudományágával az alkatrész-projektjeinkben.

A digitális alkatrészterveket általában speciális leíró nyelven, az úgynevezett ***alkatrészleíró nyelven (hardware description language, HDL)*** készítik. Ezt a megközelítés azon a feltételezésen alapul, hogy az alkatrésztervezés különleges követelményeket támaszt. Manapság a legkedveltebb *HDL* nyelv a *Verilog* és a *VHDL*.
A *MyHDL* projekt szakít a hagyománnyal és egy magas szintű, általános célú nyelvet a *Python*-t teszi alkalmassá az alkatrésztervezésre. Ez a megközelítés lehetővé teszi az alkatrésztervezőknek, hogy kihasználják egy jól megtervezett, széles körben használt nyelv és a mögötte meghúzódó nyílt forrású modell előnyeit.

Fogalmak

A *HDL* használ néhány fogalmat amelyek az alkatrészek természetét írják le. A legtöbb leíró fogalom az erős párhuzamosítás modelljére vonatkozik. A *HDL* leírása rengeteg apró szálat tartalmaz, amelyek szoros kapcsolatban állnak egymással. Ez a körülmény megköveteli, hogy a szálásítás a lehető legegyszerűbb legyen. A *HDL* szerkezeteink egy kifejezetten erre a célra kialakított környezetben, a szimulátorban futnak.
A *MyHDL* tervezésekor a *Python* szellemiségére oly annyira jellemző egyszerűségre törekedtem, ami persze egyébként is hasznos dolog. A *MyHDL* egyik fontos része a *Python* felhasználási modellje. A másik rész a *myhdl Python* csomag, amely a *HDL* fogalmak objektumait tartalmazza. A következő *Python* kód néhány *MyHDL* objektumot importál, amelyeket hamarosan használni is fogunk:

```
from myhdl import Signal, Simulation, delay, now
```

A *MyHDL* a *Python* nyelvbe nemrég bevezetett `generator` típusú függvényekkel (lásd a hálózati forrásokat) szimulálja a párhuzamosságot. Ezek nagyon hasonlóak a hagyományos függvényekhez, azzal az eltéréssel, hogy van nem végzetes visszatérési értékük is. Amikor generátorfüggvényt hívunk meg, egy generátort ad vissza, azaz a minket érdeklő objektumot. A generátorok folytathatók és állapotukat az egyes meghívások között is megtartják, így kiválóan felhasználhatjuk őket szuper-könnyűsúlyú szálakként.

Következő példánk egy óragenerátort modellező generátorfüggvényt mutat be:

```
def clkgen(clk):
        “”” Clock generator.
        clk – clock signal
        “””
        while 1:
            yield delay(10)
            clk.next = not clk
```

A függvény nagyon hasonlít a hagyományos Python függvényekhez. Figyeljük meg, hogy a kód a while 1 ciklussal kezdődik; ezzel a pórias megoldással tartjuk örökre életben a generátorunkat. A hagyományos és a generátor függvény között a yield utasítás jelenti a lényegi különbséget. Nagyon hasonlóan működik a return utasításhoz, attól eltekintve, hogy a generátor a yield után tovább él és folytathatja futását erről a pontról. Továbbá a `yield` utasítás késleltető objektumot ad vissza. *MyHDL* alatt ezzel a mechanizmussal adjuk át a vezérlési információkat a szimulátornak. Jelen esetben a szimulátort arról értesítettük, hogy tíz időegység múltán kell visszatérnie a végrehajtáshoz.
A `clk` paraméter jelképezi az órajelet. *MyHDL* alatt a generátorok közötti kommunikáció ilyen jelzésekkel történik. A jelzés fogalmát a *VHDL*-től kölcsönöztük. A jelzés objektum két értékkel rendelkezik: a csak olvasható aktuális értékkel valamint a következő (next) értékkel, amely a `.next` attribútumhoz rendelve állíthatunk be. Példánkban az órajel úgy váltakozik, hogy a következő értéket az aktuális érték inverzére állítjuk be.
Az órajelgenerátor modellezéséhez először is előállítjuk az óra jelzést:

```
clk = Signal(bool(0))
```

A `clk` jelzéshez a logikai `zero` kezdeti értéket rendeljük. Most már a generátorfüggvény meghívásával létrehozhatjuk az órajel-generátorunkat:

```
clkgen_inst = clkgen(clk)
```

1. lista MyHDL SPI szolga modell

```
from myhdl import Signal, posedge, negedge, intbv

ACTIVE_n, INACTIVE_n = bool(0), bool(1)
IDLE, TRANSFER = bool(0), bool(1)

def toggle(sig):
    sig.next = not sig

def SPISlave(miso, mosi, sclk, ss_n,
             txdata, txrdy, rxdata, rxrdy,
             rst_n, n=8):
    """ SPI Slave model.

    miso – „master in, slave out" soros kimenet
    mosi – „master out, slave in" soros kimenet
    sclk –  eltolási óra bemenet (shift clock
    ➥input)
    ss_n – aktív alacsony szolga választó bemenet
    txdata – n-bites bemenet a küldendő adattal
    txrdy – ha új txdata fogadható, megváltozik
    rxdata – n-bites kimenet a fogadott adattal
    rxrdy – megváltozik, ha van elérhető rxdata
    rst_n – aktív alacsony reset bemenet
    n – paraméteres adat

    """

    cnt = Signal(intbv(0, min=0, max=n))

    def RX():
        sreg = intbv(0)[n:]
        while 1:
            yield negedge(sclk)
            if ss_n == ACTIVE_n:
                sreg[n:1] = sreg[n-1:]
                sreg[0] = mosi
                if cnt == n-1:
                    rxdata.next = sreg
                    toggle(rxrdy)

    def TX():
        sreg = intbv(0)[n:]
        state = IDLE
        while 1:
            yield posedge(sclk), negedge(rst_n)
            if rst_n == ACTIVE_n:
                state = IDLE
                cnt.next = 0
            else:
                if state == IDLE:
                    if ss_n == ACTIVE_n:
                        sreg[:] = txdata
                        toggle(txrdy)
                        state = TRANSFER
                        cnt.next = 0
                else: # TRANSFER
                    sreg[n:1] = sreg[n-1:]
                    if cnt == n-2:
                        state = IDLE
                    cnt.next = (cnt + 1) % n
                miso.next = sreg[n-1]

    return RX(), TX()
```

A legegyszerűbb de még használható szimuláció létrehozásához készítsünk egy másik generátort, amely megfigyeli, és kiírja az óra jelzéseinek változását:

```
def monitor():
    print "time: clk"
    while 1:
        print "%4d: %s" % (now(), int(clk))
        yield clk
```

A `yield clk` utasításból láthatjuk, miképpen vár a generátor a jelzés értékének megváltozására.
MyHDL-ben a szimulátort a `Simulation` objektum konstruktorral készíthetjük el amely tetszőleges számú generátort vár paraméterként:

```
sim = Simulation(clkGen_inst, monitor())
```

a szimulátor elindításához meghívjuk a `run` metódust:

```
sim.run(50)
```

Ezzel 50 időegységen keresztül futtatjuk a szimulátort. A kimenet a következő lesz:

```
$ python clkgen.py
time: clk
   0: 0
  10: 1
  20: 0
  30: 1
  40: 0
  50: 1
```

Most már tudjuk, hogyan működik a szimulátor. A szimulátor algoritmusát a ***VHDL*** ihlette, amely ugyan kevésbé népszerű ***HDL*** mint a ***Verilog***, viszont jobb követendő példa. Az összes generátor párhuzamos végrehajtását a szimulátor esemény vezérelt algorit mussal oldja meg. A generátor `yield` utasításában megadott objektum határozza meg, azt az eseményt amelyre a következő meghívásáig várakozni szeretne. Tegyük fel az egyik szimulációs lépésnél néhány generátor

2. lista Teszkörnyezet az SPI Slave Module-hoz

```
import unittest
from random import randrange

from myhdl import Signal, intbv, traceSignals

from SPISlave import SPISlave, ACTIVE_n,
➥INACTIVE_n

def TestBench(SPITester, n):

    miso = Signal(bool(0))
    mosi = Signal(bool(0))
    sclk = Signal(bool(0))
    ss_n = Signal(INACTIVE_n)
    txrdy = Signal(bool(0))
    rxrdy = Signal(bool(0))
    rst_n = Signal(INACTIVE_n)
    txdata = Signal(intbv(0)[n:])
    rxdata = Signal(intbv(0)[n:])

    SPISlave_inst = traceSignals(SPISlave,
        miso, mosi, sclk, ss_n, txdata,
        txrdy, rxdata, rxrdy, rst_n, n=n)

    SPITester_inst = SPITester(
        miso, mosi, sclk, ss_n, txdata,
        txrdy, rxdata, rxrdy, rst_n, n=n)

    return SPISlave_inst, SPITester_inst
```

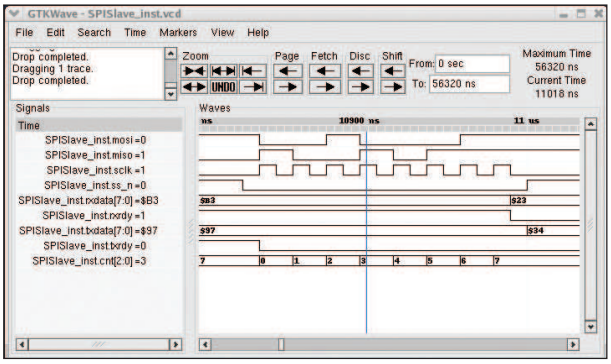
elindul, ugyanis bekövetkezik az esemény amire vártak. A szimulációs fázisban az összes generátor lefut az aktuális jeleket használva, közben pedig beállítják a következő jelet (next signal). A második fázisban az aktuális jelértékeket lecseréljük a következő jelben lévőkkel. A jelértékek változása következtében néhány generátor ismét aktívvá válik és a szimulációs ciklus megismétlődik. A mechanizmus garantáltan determinisztikus, ugyanis az aktív generátorok futtatási sorrendje a modell viselkedési szempontjából lényegtelen.

Valós tervezési példa

Az elvek bemutatása után készen állunk rá, hogy egy valódi ***MyHDL*** tervezési példával is megismerkedjünk. A soros külső csatolófelület (SPI) szolga alkatrész modulját választottam. Az ***SPI*** népszerű szinkron soros vezérlőfelület amelyet eredetileg a ***Motorola*** tervezett.
Egyetlen ***SPI*** mester több szolgát is irányíthat. Három általános I/O kaput használhatunk: a ***mosi***, azaz a mester ki, szolga be (master-out slave-in) soros vonalat, a miso, azaz a mester-be, szolga ki (master-in slave-out) soros vonalat, valamint az ***sclk***-t azaz a mester által vezérelt soros órát. Ezen kívül minden szolga rendelkezésére áll a szolga-választó (***ss_n***) vonal. Az ***SPI*** kommunikáció mindig egyszerre zajlik mindkét irányban. Az adatváltozást kiváltó óra határt általában szabadon beállíthatjuk. Ebben a példában felfutó élet használunk.
Az ***SPI*** szolga ***MyHDL*** kódját a az *1. listában* mutatjuk be. Az ***SPISlave*** nevű klasszikus ***Python*** függvény segítségével

3. lista Teszt eset SPI-on keresztül történő adatfogadáshoz

```
import unittest
from random import randrange

from myhdl import Simulation, join, delay,
➥intbv, downrange

from SPISlave import SPISlave, ACTIVE_n,
➥INACTIVE_n
from SPISlaveTestBench import TestBench

n = 8
NR_TESTS = 100

class TestSPISlave(unittest.TestCase):

    def RXTester(self, miso, mosi, sclk, ss_n,
                 ➥txdata, txrdy, rxdata, rxrdy,
                 ➥rst_n, n):

        def stimulus(data):
            yield delay(50)
            ss_n.next = ACTIVE_n
            yield delay(10)
            for i in downrange(n):
                sclk.next = 1
                mosi.next = data[i]
                yield delay(10)
                sclk.next = 0
                yield delay(10)
            ss_n.next = INACTIVE_n

        def check(data):
            yield rxrdy
            self.assertEqual(rxdata, data)

        for i in range(NR_TESTS):
            data = intbv(randrange(2**n))
            yield join(stimulus(data),
            ➥check(data))

    def testRX(self):
        """ Test RX path of SPI Slave """
        sim = Simulation(TestBench
             ➥(self.RXTester, n))
        sim.run(quiet=1)

if __name__ == '__main__':
    unittest.main()
```



1. kép A gtkwave segítségével a tesztkészlet futása során megjeleníthetjük az összes jelet

modellezzük az alkatrész modult. A függvény az összes jelzést paraméterként kapja meg és két generátort ad vissza. A kód jól mutatja hogyan modellezzük *MyHDL* alatt a hierarchiát: a magasabb szintű függvény meghívja az alacsonyabb szintűt és a visszaadott generátorokat beépíti saját visszatérési értékébe.

A modul csatolófelület néhány további jelzést és paramétert is tartalmaz. A `txdata` az átadandó bemeneti adat szó, a `txrdy` pedig akkor változik ha új adatot fogadhatunk. Hasonlóképpen, az `rxdata` tartalmazza a fogadott adatszót, és az `rxrdy` változik ha új szót fogadtunk. Végül találunk egy bemeneti `reset` jelet (`rst_n`), és az bemeneti szószélességet meghatározó n paramétert.

Az *SPI* szolga modul belsejében létrehozzuk a `cnt` jelzést amiben a soros ciklusszámot tartjuk nyilván. Ez kezdeti értékként az `intbv` objektumot használja fel. Az `intbv` alkatrész orientált osztály amely bitszintű képességekkel rendelkező egészként viselkedik. A *Python* indexelő és szeletelő képességével elérhetjük az egyes biteket és szeleteket. Továbbá az `intbv` objektumnak lehet minimum és maximum értéke.

Az `RX` generátor függvények a fogadási viselkedést írják le. Valahányszor az `ss_n` szolga választó vonal aktív alacsony, a `mosi` bemenetet az `sreg` eltolási regiszterbe toljuk. A `yield negedge(sclk)` utasítás azt jelenti, hogy a művelet az ereszkedő órajel ciklusban hajtódik végre. Az utolsó soros ciklusban az eltolási regiszter az `rxdata` kimenetre kerül az `rxrdy` pedig átvált.

A `TX` generátor függvény valamivel bonyolultabb, mivel egy apró állapotgépre van szüksége a protokoll irányításához. A `yield` utasítás ebben az esetben két esemény jelöl, így a generátor az elsőként bekövetkező esemény hatására folytatja futását. Amikor a `reset` bemenet aktív alacsony, a `cnt` és az állapot alaphelyzetbe kerül. A másik esetben a művelet az állapottól függ. Az `IDLE` állapotban megvárjuk amíg a `select` vonal aktív alacsony és csak akkor fogadjuk el a szót átvitelre és lépünk be a `TRANSFER` állapotba. A `TRANSFER` állapotban a `shift` regisztert sorosan kitoljuk. Az állapotgép fenntartja a helyes soros ciklust és az utolsó kitolás után visszatér az `IDLE` állapotba.

Ellenőrzés

Az *SPI* szolga modult a megvalósításhoz nagyon közeli szinten modelleztük. Ez jó lehetőséget adott nekünk a *MyHDL* fogalmainak bemutatására. Ugyanakkor a *MyHDL*-t ilyen célra használva nem jutunk túl sok előnyhöz a hagyományos *HDL*-ekhez képest. A *MyHDL* igazi ereje abban rejlik, hogy a teljes *Python* képességkészletet elérhetővé teszi az alkatrésztervezők számára. A *Python* kifejezőereje, rugalmassága és kiterjedt könyvtára már jóval túlmutat a hagyományos *HDL*-ek képességein.
Az egyik terület, ahol a *Python*-szerű képességek nagyon jól jönnek: az ellenőrzés. Akárcsak a programoknál az alkatrésztervezésnél is az ellenőrzés a keményebb dió. Általános vélemény, hogy a hagyományos *HDL*-ek nem igazán alkalmasak erre a feladatra. Következésképpen egy újabb nyelv jelent meg a színen, az ***alkatrész ellenőrző nyelv (hardware verification language avagy HVL)***. A *MyHDL* ezzel a megközelítéssel szemben a *Python* támaszkodik.
Az alkatrész ellenőrző környezet kialakításakor először is létrehozzuk a tesztasztalt. Ez az az alkatrész modul, amely a ***tesztelés alatt álló tervünk (design under test, azaz DUT)*** , valamint az adatgenerátorok és ellenőrzők összességét megjeleníti számunkra. A *2. lista* az *SPI* szolga modul tesztasztalát mutatja be. Itt az *SPI* szolga modult, és az *SPI* tesztelő modult láthatjuk, amely a csatolófelület tűit vezérli. Hogy egy időben több, a tervet különböző szempontok alapján vizsgáló *SPI* tesztelőt is alkalmazni tudjunk, az *SPI* teszt modul a tesztasztalunk paramétere lesz.
Magukhoz a tesztekhez tesztelő keretrendszert használunk. Az egységtesztelés a ***extrém programozás*** (*XP*) sarokköve. Ez a modern programfejlesztési módszer a józan ítélőképesség és radikálisan új elképzelések érdekes keveréke. Az eredeti *XP* megközelítés szerint először a tesztet kell kifejleszteni, még a megvalósítás előtt. Bár az *XP* igen hasznos módszertan, az alkatrészfejlesztők közössége általában mégis figyelmen kívül hagyja tanulságait. *MyHDL* alatt a *Python* egységtesztelő keretrendszere a ***unittest***, amely jó szolgálatot tesz nekünk a teszt-vezérelt alkatrészfejlesztésben.
A *3. lista* az *SPI* szolga modul teszt kódját mutatja be. A teszteket a `unittest.TestCase` osztály alosztályaiként adjuk meg. Az összes test előtaggal jelölt metódus a tényleges tesztre vonatkozik, a többi a teszt támogatására szolgál. Egy tipikus tesztben számos tesztet és tesztesetet találunk, itt azonban mintaképpen csak egytelen tesztet mutatunk be.
Az `RXTester` generátor függvény metódus célja az *SPI* szolga fogadási oldalának tesztelése. Tartalmaz egy stimulus nevű helyi generátor függvényt, amely mesterként továbbítja az adatszót az *SPI* buszon. Másik helyi generátor függvénye a `check`, amely ellenőrzi, hogy a szolga helyesen fojgadta-e az adatszót. A teljes teszt adott számú véletlen szó átviteléből áll. Minden adatszó esetében létrehozunk egy stimulus és egy `check` generátort. A *MyHDL* lehetővé teszi, hogy a `yield` utasításba helyezzük őket, így megvárhatjuk a befejezésüket. A helyes szinkronizáció érdekében csak akkor akarunk továbblépni ha már mindkét függvény kilépett. Ezt a feladatot a `join` függvény látja el.
A tesztprogram futtatásakor a kimenetben láthatjuk melyik teszt lett hibás melyik ponton. Amennyiben minden működik, apró példánk kimenete a következő lesz:

```
$ python test_SPISlave.py -v
Test RX path of SPI Slave ... ok
-----------------------
Ran 1 test in 0.559s
```

Hullámforma nézet

A *MyHDL* támogatja az alkatrész viselkedés népszerű vizsgálati módszerét, a hullámforma nézetet is. A 2. listában az *SPI* szolga modul létrehozását a `traceSignals` függvényhívásba csomagoltuk. Mellékhatásként a szimuláció alatt történt jelváltozások szabványos formátumú fájlba íródnak. Az *1. ábrán* a minta-hullámformánkat látjuk a nyílt forráskódú `gtkwave` hullámforma-nézegető megjelenítésében.

Kapcsolat más HDL rendszerekkel

A *MyHDL* praktikus megoldás, amely más *HDL* rendszerekkel is képes a kapcsolattartásra. A *MyHDL* támogatja a segéd-szimulációt más *HDL* szimulátorokkal, amennyiben azok rendelkeznek csatolófelülettel az operációs rendszerhez. Minden szimulátorhoz létre kell hozni egy hidat. Ez a nyílt forrású *Icarus* és *cver Verilog* szimulátorokhoz már el is készült.
Ezen túl, a *MyHDL* megvalósítás-orientált kódrészei automatikusan konvertálhatók *Verilog* alá. Ezt a `toVerilog` függvény segítségével érhetjük el, amelyet ugyanúgy használunk, mint a korábban bemutatott `traceSignals` függvényt. Az eredményül kapott kódot felhasználhatjuk a szokásos tervezési folyamatban például automatikusan szintetizálhatjuk megvalósításunkba.

Epilógus

Tim Peters, a híres *Python* guru, a következő paradox kijelentéssel jellemzi szeretett *Python* nyelvét: ***„a Python kódot könnyű kidobni.”*** Hasonló szellemben, a *MyHDL* célja olyan kedvelt alkatrésztervező nyelvvé válni, amelyben egyszerűen próbálhatunk ki új ötleteket.

Linux Journal 2004. november, 127. szám

Jan Decaluwe 18 évig dolgozott ASIC tervezőmérnökként és vállalkozóként. Jelenleg elektronikus tervezési és automatizálási tanácsadó. A jan@jandecaluwe.com címen érhető el.

KAPCSOLÓDÓ CÍMEK

- www.jandecaluwe.com/Tools/MyHDL/Overview.html
- sourceforge.net/projects/myhdl
- news.gmane.org/gmane.comp.python.myhdl
- www.python.org/peps/pep-0255.html
- www-106.ibm.com/developerworks/linux/library/l-pythrd.html
- www.embedded.com/story/OEG20020124S0116
- www.mct.net/faq/spi.html

Makacs hibák megkeresése sokatmondó hibainformációkkal

Amikor egy felhasználó egy olyan hibát jelez, amit nem tudunk megismételni, bízzuk inkább a programunkra a hiba megkeresését. Hozzunk létre egy eseménynaplót, és legyünk hibakereső bajnokok a szoftver alkalmazása után.

Egy hiba nyomon követése gyakran a programfejlesztés egyik legbonyolultabb folyamata. A felhasználók sokszor a fejlesztőétől eltérő helyzetben használják a programot, és a programhibák, amik a felhasználónak nagy gondot okoznak, a fejlesztő gépén esetleg meg sem jelennek. Néha a hibák maguktól elmúlnak, vagy a hálózatos programokban csak akkor jelentkezik a tünet, amikor egy bizonyos kiszolgálóval vagy ügyféllel folytatunk adatcserét. Ebben a cikkben olyan módszereket ismertetek, amelyek segítségével a programfejlesztők könnyebben kinyomozhatják egy hibajelenség eredetét.
Először bemutatok két módszert a hibajelentések könnyebb fogadására és kezelésére, majd megmutatom, miképpen érhetjük el, hogy a programunk jobban használható hibakereső kimenetet adjon. Ezután szólni fogok a kellemetlen hibák felderítéséről, végül bemutatok néhány fogást arra, hogyan előzhetjük meg a programhibák kialakulását.
A cikkben bemutatandó eljárások közül sokat alkalmaztam az ***OfflineIMAP*** fejlesztésekor.

A programhibák felderítése

Mielőtt megvizsgálnánk, hogyan tehetünk szert jobb hibajelentésekre, nagyon fontos, hogy biztosak legyünk benne, hogy egyáltalán foglalkozni tudunk-e a kapott hibajelentésekkel. Kisebb projektek esetében elegendő lehet, ha egyszerűen egy e-mail címet tartunk fenn erre a célra, habár a legtöbb projekt esetében ennél többre van szükség. A fejlesztők gyakran elfoglaltak és feledékenyek. A hibák kijavítása meglehetősen bonyolult is lehet, és több ember közreműködését is igényelheti, de az is előfordulhat, hogy elárasztanak a hibajelentések.
A ***hibakövető rendszerek (BTS, bug-tracking system)*** remek eszközt biztosítanak ahhoz, hogy egy hibáról se feledkezzünk el. A legtöbb BTS lehetőséget ad a levelezés követésére, a csatolt állományok kezelésére és a hibákkal kapcsolatos feladatok megosztására. Néhány támogatja a hiba súlyosságа, a felhasználói környezet és bizonyos összetevők alapján történő osztályozást is.
Amennyiben a projektünket a ***SourceForge*** vagy ***Savannah*** weboldalakhoz hasonló tárhelyeken helyezzük el, akkor ezzel biztosítva van számunkra egy ***BTS***-rendszer is. Ezt mindenképpen érdemes használnunk és a felhasználóinkat is arra kell ösztönöznünk, hogy a levelezőlisták helyett ezt a felületet használják a hibák jelzésére.
Ha ennél nagyobb rugalmasságra tartunk igényt, kereshetünk a ***Linuxhoz*** készített ***BTS***-rendszert is. Néhány a legnépszerűbb ingyenes ***BTS***-programok közül:

- ***Bugzilla***: A ***Mozilla Projekt*** által használt ***BTS*** egy rugalmas rendszer, amelyet elsődlegesen a webes felületén keresztül alkalmazhatunk.
- A ***Request Tracker*** használható akár hibakövető, akár támogatás-nyilvántartó rendszerként. Rendelkezik webes és levelezői felülettel is, bár bizonyos rendszergazdai funkciók csak a webes felületen keresztül érhetőek el.
- A ***Jitterbug*** a ***Samba Projekt*** által is használt ***BTS***. Az alkalmazott elv hasonló a Bugzillaban használthoz, de annál kisebb programról van szó.
- A ***Debbugs*** a ***Debian Projekt*** által használt ***BTS***. A ***Debbugs*** rendelkezik egy webes felülettel, de ez csak olvasható, minden művelet e-mail-en keresztül történik. A ***Debbugs*** a nagy projektekhez a legmegfelelőbb az egyértelműen azonosítható összetevőivel és ezek hatáskörével.

Tegyük egyszerűbbé a hibák bejelentését

Előfordul, hogy kellemetlen hibát találok egy programban és erről tájékoztatni akarom a fejlesztőt, de ehhez egy részletes kérdőívet kell kitöltenem és olyan dolgokat megadnom, amiket nem nagyon szeretnék. Egyszerűvé kell tennünk a felhasználók számára a hibák és a felderítésükhöz szükséges információk elküldését. Ha a bejelentéseket a Világhálón keresztül fogadjuk, a folyamat legyen minél egyszerűbb. Ne követeljünk túl sok információt és olyan bejelentéseket is fogadjunk el, amikről esetleg hiányzik néhány adat. Ne várjuk a felhasználóktól, hogy tisztában legyenek a program különböző összetevőivel, vagy hogy tudják, melyik fejlesztőnek kell foglalkoznia az adott problémával.

A napló használata

Problémák felderítése közben gyakran szeretnénk tudni, hogy milyen környezetben működik a program, máskor

pedig esetleg arra van szükség, hogy ismerjük a hiba jelentkezését megelőző, azt kiváltó eseményeket. Mivel a felhasználók nem feltétlenül ismerik az általunk kódolásra és hibakeresésre használt eszközöket, elterjedt a naplózás használata erre a célra. A naplózás egyszerűen egy rekord rögzítését jelenti az elvégzett műveletekről. Az egyszerű programok esetleg csak a képernyőre írják az adatokat, de valószínű, hogy ennél kicsit jobb eszközre lesz szükségünk.

A nem interaktív programok – amilyenek például a hálózati kiszolgálók – nem rendelkeznek olyan kijelzővel, amin megjeleníthetnék ezeket az információkat. Ezek a programok a bejegyzéseiket rendszerint naplófájlba rögzítik, vagy a ***Linux*** és ***UNIX*** rendszerekbe beépített ***syslog*** eszközt használják.

Az interaktív programok vagy a képernyőre vagy egy fájlba írhatják az információkat. A naplófájl megkönnyíti a hibajelentések elkészítését, mert a felhasználó egyszerűen csatolhatja ezt a fájlt a bejelentéséhez.

Sokszor meglehetősen sok adatra van szükség annak megállapításához, hogy pontosan mi is történik egy adott rendszeren, de ez az adatmennyiség lehetetlenné is teheti a normál működést, elboríthatja a felhasználó képernyőjét, vagy megtöltheti a merevlemezét. Éppen ezért sok programban találkozhatunk a naplózási szint fogalmával, ami annyit jelent, hogy a felhasználó futási időben meghatározhatja a naplózott információ mennyiségét. Néhány program osztályozza is az eseményeket, így a felhasználó azt is megadhatja, hogy milyen típusú információk kerüljenek be a naplóba. Az ***OfflineIMAP*** ezt a megközelítési módot használja. Kellemetlenebb problémák esetén a felhasználó bekapcsolhatja a kapcsolattartási naplót, amely minden, az ***IMAP***-kiszolgáló felé elküldött vagy onnan érkező adatot naplóz.

A ***Python 2.3*** változatában megjelent egy hasznos modul, aminek ***naplózás (logging)*** a neve. Ez a modul egységes felületet biztosít az üzenetek naplózásának számos különböző módszeréhez. A támogatott naplózási módszerek közt megtaláljuk a naplófájlok használatát, a hálózati szolgáltatásokat, a ***syslog***-ot, az e-mail üzeneteket és számos egyéb módszert. Nézzünk egy egyszerű példát a naplózó modul használatának bemutatására:

```
#!/usr/bin/env python
import logging, sys
# A naplózó objektum létrehozása
l = logging.getLogger('testlog')
# Egy kezelő létrehozása és az objektumhoz rendelése
handler = logging.StreamHandler(sys.stderr)
l.addHandler(handler)
# A szintek:  DEBUG, INFO, WARNING, ERROR,
# CRITICAL.
# Beállítjuk az alapértelmezett szintet.
# Az e szint alatti
# naplóüzeneteket figyelmen kívül hagyjuk.
l.setLevel(logging.INFO)
# Kipróbáljuk.
l.debug("Debug message -- system initialized.")
l.info("Here's some info.  I've just debugged.")
l.warning("I don't have many messages left.")
l.error("Only one more message to go.")
l.critical("Nothing else to do!")
```

A program a naplózás alapbeállításával kezdődik. A naplóüzenetek szabványos hibakimenetre történő kiírását a `StreamHandler` kezelő végzi. A naplózás szintjét `INFO`-ra állítjuk. Öt naplóüzenetet naplózunk, de a program futásakor csak az utolsó négyet látjuk. A `debug` üzenetet a naplózási szint `INFO`-ra állítása kiszűrte. Sok program biztosít beállítási lehetőséget vagy parancssori paramétert a naplózási szint futásidőben történő beállítására. Ettől eltérő naplózási módszert is használhatunk egyszerűen egy másik kezelőt adva a `Logger` objektumunkhoz. A ***Python*** leírásában megtaláljuk az összes használható kezelő ismertetését.

A bemenet ellenőrzése

Bizonyosodjunk meg arról, hogy a kapott bemenet megfelelő. Például ha a parancssoron keresztül várunk valamit, ellenőrizzük a paraméterek megfelelő számát, mielőtt megpróbálnánk alkalmazni (vagy a kapott kivételt elcsípni). Ezzel a módszerrel jobb hibaüzenethez juthatunk. Íme egy ***Python*** példaprogram ennek bemutatására:

```
#!/usr/bin/env python
import sys
try:
    print "You supplied: %s" % sys.argv[1]
except IndexError:
    print "You forgot an argument."
```

A kivételek kezelése

Számos programozási nyelv – többek közt a ***Java***, ***Python*** és az ***OCaml*** – biztosít lehetőséget a kivételek kezelésére. A kivételek használatával a kívánt helyen csíphetjük el a hibákat ahelyett, hogy minden esetlegesen problémát okozó hívást ellenőriznünk és az előforduló hibákat kezelnünk kellene. Időnként nem okoz gondot a kivételek lekezelés nélküli átengedése, de rendszerint nem ez a helyzet, a kivételeket el kell fogni és megfelelően kezelni. Habár megfelelő megoldás lehet felfüggeszteni a program működését, amikor nem tudjuk megnyitni a felhasználó által kért fájlt, de ezt jobb egy megfelelő hibaüzenettel tennünk, amely megadja a probléma jellegét és a fájl nevét, ahelyett, hogy a felhasználó csak a kivétel csúnya üzenetét kapná meg.

A kivételek elfogása

Még a tényleg végzetes kivételek esetén is érdemes lehet a kivételt elfogni. Ez lehetővé teszi például a kivétel naplófájlba való írását vagy egy felugró ablakban a grafikus felületen történő megjelenítését. Ez megkönnyíti a felhasználók számára, hogy a rögzített nyomokat visszaküldjék számunkra. Használhatunk általános kivételelfogót is, amely egyéb műveleteket is végrehajthat, például az átmeneti tárak rögzítését, amely megkönnyíti az események visszakövetését. Az alábbi példa minden kivételt elkap és naplóz néhány egyéb, a futó programmal kapcsolatos információval együtt. Ezután újra kiváltja a kivételt és kilép.

```
#!/usr/bin/env python
import logging, sys, StringIO, traceback, os
l = logging.getLogger('testlog')
handler = logging.StreamHandler(sys.stderr)
l.addHandler(handler)
```

```
formatter = logging.Formatter("LOG: %(message)s")
handler.setFormatter(formatter)
l.setLevel(logging.INFO)
def logexception():
    sbuf = StringIO.StringIO()
    traceback.print_exc(file = sbuf)
    excval = sbuf.getvalue()
    l.critical(" *** Exception Detected ***")
    l.critical("Current PID: %d" % os.getpid())
    l.critical("Program name: %s" % sys.argv[0])
    l.critical("Command line: %s" % \
                str(sys.argv[1:]))
    for line in excval.split("\n"):
        l.critical(line)
def main():
    print "Hello, I'm running."
    raise RuntimeError("Oops! I've had a problem!")
try:
    main()
except:
    logexception()
    raise
```

A program futtatásakor valami ilyesmit kell látnunk a képernyőn:

```
Hello, I'm running.
LOG:  *** Exception Detected ***
LOG: Current PID: 28441
LOG: Program name: /tmp/logerror.py
LOG: Command line: []
LOG: Traceback (most recent call last):
LOG:   File "/tmp/logerror.py", line 30, in ?
LOG:     main()
LOG:   File "/tmp/logerror.py", line 27, in main
LOG:     raise RuntimeError("Oops! I've had
         ➥a problem!")
LOG: RuntimeError: Oops! I've had a problem!
LOG:
```

A kivételkezelő megtalálta a kivételt, begyűjtötte a kapcsolódó információkat és sikeresen rögzítette a naplóban. Másodszor is láthatjuk a ***visszakövető (traceback) információkat***. A program végén lévő `raise` utasítás újra előidézi a kivételt és lehetővé teszi annak normál módon történő lekezelését. Ez azt jelenti, hogy egy visszakövetéssel félbeszakítja a programunkat. Az elvárásainktól függően egy `sys.exit()` műveletet is használhatunk ehelyett.

A bejelentett hibák megkeresése

Most, hogy már rendelkezünk néhány módszerrel arra, hogy a felhasználókat segítsük a minél jobb hibajelentések elküldésében, vizsgáljuk meg ezeknek a jelentéseknek a felhasználási módjait. Egy hibanaplóval és talán a visszakövetési információkkal felfegyverkezve a következő kérdéseket tehetjük fel magunknak:

- Elő tudom-e idézni a hibát a saját futtatókörnyezetemben? Ha ez a saját gépünkön is sikerül, közel kerültünk ahhoz, hogy ki is javítsuk. Használjunk egy hibakeresőt vagy valamilyen más eszközt az ok megkereséséhez.
- Az elvártnak megfelelő volt a bemenet és a kimenet? Előfordulhat, hogy a felhasználó olyan értéket használt, amire nem gondoltunk a program írásakor. Esetleg egy hálózati ügyfélgép vagy kiszolgáló kezel egy kicsit eltérően valamilyen protokollt. Talán csak a bemenet vagy a kimenet maga torzult és a hiba nem is a mi programunkban van. Nagyon hasznos tud lenni ezen a ponton egy olyan hibakereső napló, amely az összes be- és kimenetet tartalmazza.
- A vártnak megfelelően alakult a program futási folyamata? Amennyiben a naplónk különböző függvények és eljárások hívását tartalmazza, szükséges, hogy egy programmal követni tudjuk a futási folyamatot. Előfordulhat, hogy valamilyen feltételek együttesen vezettek egy élő kódrészlet figyelmen kívül hagyásához, amely később vezetett a hibához.
- Mi a helyes futás utolsó pontja? Ez lehet a hibát közvetlenül megelőző rész, de az is előfordulhat, hogy már a hibát valamennyi idővel megelőzően történt egy rossz adatátadás. Ha megkeressük azt a legkésőbbi helyet, ahol a program még megfelelően működött, akkor könnyebben behatárolhatjuk a félresiklás pontos helyét.
- Ha kéznél vannak a visszakövetési információk (traceback), ellenőrizzük, hogy a verem tartalma megfelelőnek tűnik-e. Vizsgáljuk meg, hogy a függvényhívások és a kapott paraméterek a vártnak megfelelőek-e.

A hibák megelőzése

Ugyan hasznosak a fent ismertetett módszerek, de önmagukban még nem elegendőek. Fontos az is, hogy olyan fejlesztési gyakorlatot folytassunk, amely segít csökkenteni a hibák előfordulási valószínűségét. Néhány megfontolásra érdemes módszer ezek közül:

- Alkalmazzunk az ***egységek tesztelését (unit testing)***. A ***Java***, ***Python***, ***OCaml***, ***Perl*** és ***C*** nyelvek mindegyike rendelkezik egységtesztelő keretrendszerrel. Használjuk ki ezeket és a lehető legtöbb futási esetet vizsgáljunk meg velük. Ez különösen az olyan nyelveknél fontos, mint a ***Python***, amelynél egy bizonyos futáskor még a kódértelmezés sem terjed ki a teljes kódra. Fontos ez a Java esetében is, ahol a futásidejű kivételek nem megfelelő objektumváltást eredményezhetnek.
- Kerüljük a globális változókat. A globális (vagy osztályszinten globális) változók segítenek a problémák körülhatárolásában és megóvnak a többszálú programok szinkronizációs gondjaitól. A globális változók nem várt és nehezen visszakövethető mellékhatásokat okozhatnak a függvényhívásokban.
- Használjuk a legmegfelelőbb eszközt az adott munkához. A programozási nyelvek mindegyikének megvan a maga erőssége és gyengéje, nincs olyan nyelv, ami minden feladatra a legmegfelelőbb. Például ***Perlben*** programozva könnyű a körülhatárolt szövegfájlok szabályos kifejezésekkel történő elemzése, az ***OCaml*** pedig olyan eszközökkel rendelkezik, amelyeket kifejezetten ***fordítóprogramok (compiler)*** írására használhatunk. Az egyik nyelvben könnyen kifejezhető probléma sokkal bonyolultabb lehet egy másikban.
- Ne használjunk túl sok különböző eszközt sem. A legtöbb projektnek előnyére válik a használt eszközkészlet

behatárolása. Válasszuk ki a legmegfelelőbb nyelvet és programkönyvtárat és csak akkor nyúljunk új eszközhöz, ha erre alapos okunk van.

- Használjunk karakter- és memóriakezelő programokat. Sok nyelv, többek közt a ***Java***, ***Python***, ***OCaml***, ***Perl*** és ***Ruby*** háttérben futó memóriakezelést biztosít, így nincs szükség a memória lefoglalására és felszabadítására. Nem kell az explicit karakterlánc-végződésekkel és a karakterláncokra vonatkozó hosszkorlátokkal foglalkoznunk, melyek mindegyik gyakori probléma a *C* nyelv használatakor és futásidejű hibákhoz, vagy biztonsági résekhez vezethet. Ha mindenképpen *C*-t kell használnunk, fontoljuk meg egy hulladékgyűjtő- vagy memóriatartalék-programkönyvtár használatát.
- Először tegyük a programot működővé és utána keressük meg a lehető legjobb megoldást. Sok esetben jobb kifejleszteni egy működő kódot és később optimalizálni. Sokan először optimalizálnak, ami működik is olykor, mégis rendszerint fontosabb egy egyszerű, hibátlan kód, mint egy olyan, ami a lehető leggyorsabb.
- Kódoljunk tisztán, készítsünk függvényeket az egyes kódrészletekhez. Használjunk megjegyzéseket. Készítsünk leírást arról, hogy melyik függvénynek mi a szerepe és milyen hatással van a környezetre.

Esettanulmány: egy hiba az OfflineIMAP-ben

Az ***OfflineIMAP*** egy olyan program, amely kapcsolatot tart az ***IMAP***-kiszolgálókkal és összehangolja az ***IMAP*** mappaszerkezetet a helyi faszerkezettel. Sokféle ***IMAP***-kiszolgáló létezik, amelyek nem teljesen azonos módon működnek. Kétéves pályafutása során az ***OfflineIMAP*** egyre nagyobb arányban alkalmazta a cikkben bemutatott hibakereső eljárásokat. Azok a problémák, amelyekkel a felhasználók szembesülnek, gyakran előidézhetetlennek bizonyulnak az én egyedi rendszeremen, ezért elengedhetetlen, hogy a program részletes naplózást folytasson. Néha maguk az ***IMAP***-kiszolgálók sem hibátlanok, ezért a hibajelentések jelentős részénél az első megválaszolandó kérdés az, hogy egyáltalán az ***OfflineIMAP*** rendszerben lévő hibáról van-e szó. Az esetek meglepően nagy százalékában nemleges a válasz. Az ***OfflineIMAP*** számos olyan ***IMAP***-szolgáltatást használ, amelyet az ***IMAP***-ügyfelek többsége nem, és ezek a tulajdonságok néhány kiszolgálón elég gyengén tesztelt állapotban vannak.

Szeretnék bemutatni az ***OfflineIMAP*** egy különösen makacs hibáját, amivel egyszer dolgom akadt. Körülbelül egy évvel ezelőtt egy felhasználó egy hibát jelzett a programban, amelyet a ***Debian*** hibakereső rendszerével fedezett fel. Sajnos nem tudtam újra előidézni a hibát és az eredeti bejelentőnek éppen nem volt a naplózás bekapcsolva, amikor a hiba jelentkezett. Hibakereső információkat szintén nem tudott biztosítani. A kapott információk birtokában, amelyek közt egy hibaüzenet is volt, a cikkben korábban vázolt lépéseket követve sikerült némi információt összegyűjtenem. A be- és kimenet nem állt rendelkezésemre, de a program folyamata és a verem tartalma jónak tűnt. Végül már tudtam, hogy a program melyik részén következett be az esemény, de azt nem, hogy miért, így a kódban egy ideig bennmaradt a hiba. A helyzetet bonyolította, hogy a jelenség nem jött elő állandóan, a program néha jól működött, időnként azonban bedobta a törölközőt.

Később egy második felhasználó is tapasztalta ugyanezt a hibát, észrevette a korábbi debianos hibajelentést és elküldte a saját információit vele kapcsolatban. Az ***OfflineIMAP*** végzetes hiba esetén megpróbálja kiíratni a hibakereső napló részeit, ennek a felhasználónak pedig sikerült elcsípnie ezeket a kimeneteket. Az ***OfflineIMAP*** e tulajdonsága hasznosnak bizonyult tehát, mivel gyakran lehetetlen egy problémához vezető helyzet utólagos reprodukálása.

Ebben az esetben segített a kapott információ, most már képes voltam felidézni, mit csinált az ***OfflineIMAP*** közvetlenül a hiba felbukkanása előtt. Ahhoz azonban kevés volt, hogy a probléma okára is rájöjjek, minden normálisnak tűnt. A hiba csak időnként bukkant fel, és további információk nem álltak rendelkezésre.

Végül egy harmadik felhasználó is tapasztalta ugyanezt a hibát. Neki is voltak információi, de ahhoz kevés, hogy választ kapjak a kérdésemre. Valaminek még történnie kellett, ezért a kérdéses kódrészhez még részletesebb naplózást készítettem. Remélhetőleg a következő alkalommal a részletesebb információk alapján majd visszakövethetem a probléma okát.

A folyamat során számos tényező játszott fontos szerepet. Az első, hogy az ***OfflineIMAP*** végzetes hiba esetén mindig létrehoz egy használható verem-képet. Még e legkevésbé részletes jelentés is pontosan megmutatta, hogy milyen állapotban volt a program az összeomláskor. Másodszor, a hibanaplók ugyan hasznosak, de kevéssé vehetjük hasznukat akkor, amikor egy bizonyos hibát nem tudunk könnyen előidézni. A hibakereső információk kiíratása a program összeomlásakor vagy hibás működésekor hasznos lehet a probléma leküzdésében.

A hibakövető rendszer is fontos szerepet játszhat a probléma felderítésében. Mivel a ***Debian*** hibajelentései nyilvánosak, a három említett hibabejelentő felismerhette a fennálló hibajelentéseket és hozzátehette a saját információját. Ez mindenkinek segített az adott témával kapcsolatos információk kezelésében és kiindulópontot biztosított azoknak a felhasználónak, akik először botlottak a problémába.

Összegzés

Számos módja van annak, hogy segítsük a felhasználóinkat a hibák bejelentésében és visszakövetésében, de ezek önmagukban még nem elegendőek. Ne feledkezzünk meg arról, hogy minél könnyebb legyen a hiba bejelentése és követése, és hogy törekedjünk a kód tisztaságára. Végül azt se feledjük, hogy önmagában egyik lépés sem varázstöltény. Együtt alkalmazva egyszerűsíthetik a hibakereső folyamatot és segíthetnek a problémák felderítésében, de nem feltétlenül oldanak meg minden kérdést.

Linux Journal 2005. január, 129. szám

A cikkhez kapcsolódó források a ➲ ***www.linuxjournal.com/aricle/7747*** címen érhetőek el.

John Goerzen hosszú ideje Linux-programozó, a Foundations of Python Network Programming (A Python hálózati programozásának alapjai) szerzője. A szoftverágazat igazgatójaként dolgozik a Public Interest, Inc. cégnél. A hozzászólásokat a jgoerzen@complete.org címre várja.

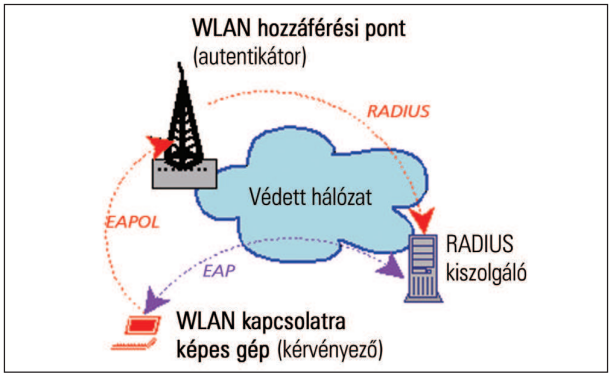
WLAN-ok védelme WPA és FreeRADIUS alkalmazásával (1. rész)

Vezeték nélküli hálózatunkat a kiöregedett, biztonságot nem nyújtó WEP helyett védjük az új szabvány szerint, s egyben építsük össze a hitelesítést linuxos hálózatunkkal.

Aggódunk ***802.11b vezeték nélküli helyi hálózatunk (wireless local area network, WLAN)*** biztonsága miatt, mert még a jó öreg ***WEP***-et (wired equivalent privacy, vezetékes egyenértékű adatvédelem) használjuk? Aki kizárólag a ***WEP***-re bízza magát, bizony ***jól teszi, ha fél***: komoly és jól ismert sebezhetőségek vannak benne, amelyek révén a kalózok néhány órányi hallgatózás után, nyers erőből végzett töréssel könnyedén visszafejthetik ***WEP***-kulcsainkat. De van remény! A ***WPA (Wi-Fi protected access, Wi-Fi védett elérés)*** új hitelesítési eljárásokkal és továbbfejlesztett kulcselőállítási képességgel ruházza fel a ***802.11b*** hálózatokat, és a ***WPA*** támogatására képes ***WLAN***-készülékek szinte pillanatok alatt megjelentek az üzletekben. További örömre ad okot, hogy a ***WPA-kérvényezők (ügyfélrendszerek)***, a ***hitelesítők (hozzáférési pontok)*** és a ***kiszolgálók (RADIUS hitelesítő kiszolgálók)*** számára linuxos eszközök is rendelkezésre állnak. Következő két cikkemben a ***WPA***-t és alkotó protokolljait, illetve ezek együttműködését fogom ismertetni, valamint szólok arról is, hogy a ***FreeRADIUS*** csomag segítségével hogyan helyezhetünk üzembe egy ***Linux***-alapú ***WLAN*** hitelesítő kiszolgálót.

Áttekintés

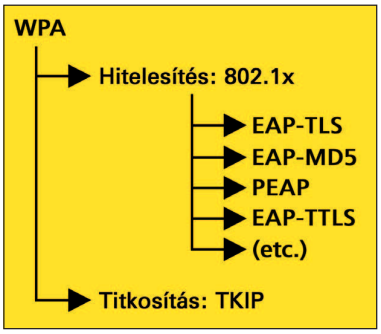
Mi is alapvetően a baj a ***802.11b*** hálózatok biztonságával? Röviden, a ***802.11b*** szabvány ***WEP*** protokolljában van két súlyos hiba. Az első, a titkosítási-megvalósítási hiba lehetetlenné teszi, hogy a gyakorlatban 40 bitnél erősebb kulcsot használjunk, még ha rendszerünk elvileg képes is lenne a hosszabb kulcsok kezelésére. A második a ***WEP*** titkosítási kulcs származtatási eljárásában keresendő, miatta a támadó megfelelő számú csomag elfogása után meg tudja határozni a hálózat titkos ***WEP***-kulcsát – azt a titkosítási kulcsot, amelyet a hálózat minden egyes állomása használ.
A jelenleg fejlesztés alatt álló ***802.11i*** protokoll teljes értékű, erőteljes biztonsági keretrendszert fog biztosítani a ***WLAN***-okhoz. Természetesen véglegesítése után is kell némi idő ahhoz, hogy a protokoll széles körben elérhetővé váljon a kereskedelmi termékekben és a szabad programokban. Térjünk rá a ***WPA***-ra. A ***WPA*** a ***802.11i*** két kulcsfontosságú összetevőjét emeli át a ***802.11b***-be. Az első a rugalmas és nagyteljesítményű hitelesítési szolgáltatásokat biztosító ***802.1x*** hitelesítő protokoll. A második a ***TKIP*** protokoll, melynek segítségével egyedi ***WEP***-kulcsokat tudunk hozzárendelni az egyes ***WLAN***-ügyfelekhez, majd dinamikusan tudjuk elvégezni ezek újraegyeztetését, így a ***WEP***-nél látott kulcsszármaztatási sebezhetőség megszűnik.



1. ábra A WPA felépítése

Az *1. ábrán* a ***WPA*** alapú rendszerek összetevői közötti kapcsolatokat szemléltettem. Először is, van egy ***WLAN***-képes ügyfélrendszerünk, ennek ***WPA*** ügyfélprogramját kérvényezőnek nevezzük. Az ügyfél/kérvényező csatlakozik egy ***vezeték nélküli hozzáférési ponthoz (access point, AP)***, amely hitelesítő szerepet játszik, miközben gyakorlatilag proxyzza a hitelesítési párbeszédet a kérvényező és a háttérben üzemelő hitelesítő kiszolgáló között. Az 1. ábrán ez a hitelesítő kiszolgáló egy ***RADIUS*** kiszolgáló, de ***TACACS***-t is használhatunk.
A kérvényező és a kiszolgáló közötti hitelesítésproxyzás mellett az ***AP/hitelesítő*** a ***TKIP (Temporal Key Integrity Protocol, ideiglenes kulcsintegritás protokoll)*** alkalmazásával adatokat közöl a hitelesítő kiszolgálóról, ezzel segítve a ***WEP*** munkamenetkulcs beszerzését. Ezután átadja a kulcsot a kérvényezőnek. A kérvényezőnek rendszeres időközönként újra kell hitelesítenie magát, ilyenkor új ***WEP*** kulcsot kap.
A hitelesítő (***RADIUS***) kiszolgáló elhagyható. Egy másik lehetőség az ***előre megosztott kulcs (pre-shared key, PSK)*** mód használata, ennél az egyes ***WPA*** kérvényező rendszerek egyedi kulcsát kézzel adjuk meg az ***AP***-nek, majd ***RADIUS***

helyett ezt használjuk fel a hitelesítésre. Ez az eljárás is jobb, mint a ***WEP***, mert magát a megosztott kulcsot nem használjuk titkosító kulcsként. Feladata csupán a dinamikus *WEP* kulcsokat szállító *TKIP* tranzakciók védelmének megalapozása.
A *WPA*-t jelenleg az újabb *WLAN* csatolók és hozzáférési pontok széles köre támogatja, sőt, a belső programok frissítésével néhány régebbi, *802.11b*-s terméken is elérhetővé vált. A linuxos világban támogatását az ügyfél oldalon a *wpa_supplicant (hostap.epitest.fi/wpa_supplicant)*, a linuxos hozzáférési pontokon a *hostapd (hostap.epitest.fi/hostapd)*, míg a hitelesítő kiszolgálók oldalán a *FreeRADIUS (www.freeradius.org)* biztosítja.
Mielőtt rátérnénk szűkebb témánkra, s egyben következő cikkem tárgyára, a *WPA*-képes *FreeRADIUS* kiszolgálók építésére, vizsgáljuk meg részletesebben is a *WPA* hitelesítési és titkosítási megoldásait.



2. ábra A WPA protokolljai0

WPA hitelesítés: 802.1x, EAP és RADIUS

Mindenki tud követni? Csak azért, mert a *WPA* valójában egy kicsit bonyolultabb annál, amit az 1. ábra tartalma sugall. Tehát: a *WPA* használatakor az ügyfélrendszernek (a kérvényezőnek) még az előtt kell hitelesítenie magát, hogy engedélyt kapna a csatlakozásra, amikor is hozzájut egy rendszeresen lecserélésre kerülő titkosítási munkamenetkulcshoz.
A dolog attól kezd bonyolódni, hogy a *WPA* hitelesítésre használt *802.1x* protokoll számos különböző módszer használatát teszi lehetővé a kérvényezők hitelesítésére – ami persze jó dolog. Moduláris, bővíthető hitelesítő megoldást alkalmazva nem kell attól tartanunk, hogy a *WPA*, a *802.1x* vagy a *802.11i* egy csapásra elavulttá válik, ahogy a különféle hitelesítő protokollok divatba jönnek és elfelejtődnek.
A *802.1x* modularitása és bővíthetősége a számos változatban létező *Extensible Authentication Protocol (bővíthető hitelesítő protokoll, EAP)* hozománya. Ejtsünk néhány szót a legnépszerűbb változatokról.

Az a számtalan protokoll!

Nem véletlen, hogy egy egész cikket szentelek a *WPA* működésének boncolgatására, és nem vetem bele magam a *FreeRADIUS WPA* használatára való beállításának témájába – annyi a *WPA* felépítésében részt vevő protokollal és alprotokollal találkozhatunk ugyanis, hogy megfelelő alapismeretek hiányában pillanatok alatt összezavarodunk. Aki már most elveszítette a fonalat az talán a *WPA* protokolljait hierarchikus formában szemléltető *2. ábra* alapján rendet tud teremteni gondolatai között.
Az *EAP-MD5* egyszerű, *MD5* kivonat alapú azonosítóadatcserét végez. A kérvényező közöl egy felhasználónevet és egy *MD5* kivonatolt jelszót a kiszolgálóval, amely összehasonlítja ezeket az adatbázisában tárolt adatokkal. Sajnos hallgatózással meg lehet szerezni a *WPA* kérvényező által elküldött kivonatot, és offline, szótár alapú támadással meg lehet határozni az előállításához használt jelszót. Gondot jelent az is, hogy bár az *EAP-MD5* hitelesíti a kérvényezőt a kiszolgáló felé, ám semmit nem tesz annak érdekében, hogy a kiszolgálót is hitelesítse a felhasználó számára, például tanúsítvánnyal, ahogy azt az *SSL* esetében is láthatjuk. Az *EAP-MD5* tehát *WPA* környezetben meglehetősen gyenge választás *802.1x* hitelesítésre.
Az *EAP-TLS* alapvető hitelesítésre a *TLS* titkosító protokollt, az *SSL* leszármazottját használja. Ez egy erőteljesnek mondható hitelesítési mód, hiszen használatakor a kiszolgáló és az ügyfél egyaránt saját digitális tanúsítványával – ezek adják a hitelesítési transzakciók alapját – igazolja magát. Nagyobb számú felhasználónak digitális tanúsítványt adni, illetve ezeket a tanúsítványokat kezelni ugyanakkor összetett és időrabló teendő lehet. Elég, ha arra gondolunk, hogy a szervezetünket elhagyó személyek tanúsítványát vissza kell vonni. Az *EAP-TLS* használatához általában teljes értékű *nyilvános kulcs infrastruktúrát (Public Key Infrastructure, PKI)* kell fenntartani, ami viszont egy kisebb vagy akár közepes méretű szervezet számára megterhelő lehet. Ne feledjük el azt sem, hogy a hitelesítések kezdeményezésekor a felhasználónevek nyílt szövegként továbbítódnak, ami apró kis hiányosság ugyan, de nem árt megjegyezni.
A *PEAP (Protected, védett EAP)* eredetileg a *Microsoft* fejlesztése, célja a gyengébb, de egyszerűbb hitelesítési eljárások, például az *MD5* és az *MS-CHAP TLS* titkosítással való védelme. A *PEAP* használatakor az azonosító adatok továbbítása előtt egy titkosított csatorna jön létre a kérvényező és a kiszolgáló között. Általában a webes alkalmazások is hasonló módon használják a *TLS*-t: segítségével felépítenek egy titkosított alagutat, ezen keresztül biztonságosan le lehet bonyolítani az egyszerű, felhasználónév és jelszó alapján végzett hitelesítéseket, és nincs szükség a TLS biztonságosabb, ám jóval bonyolultabb, ügyféltanúsítványra épülő eljárásainak alkalmazására. A *PEAP* fő hátránya *Microsoft*-központúsága. Bár a szabad programok között is találni a *PEAP* támogatására képest, a legtöbben nem látják a *Microsoft* szándékát arra, hogy biztosítsa az együttműködés lehetőségét más gyártók *WPA* termékeivel vagy megoldásaival.
Az *EAP-TTLS* lényegében egy nem microsoftos *PEAP*-alternatíva. Esetében is létrejön egy titkosított *TLS* alagút, ezen keresztül *TLS* alapú vagy egyéb, gyengébb hitelesítési eljárást lehet folytatni. Fő előnye a *PEAP*-pal szemben, hogy nincs kitéve egy nagyvállalat szeszélyeinek. Jelenleg hitelesítési módszerből is többet támogat – igaz, a *PEAP*-ot is úgy tervezték, hogy a jelenleg megvalósítottaknál jóval több módszer támogatására legyen képes. Néhányan úgy látják, a *Microsoft* támogatása nélkül az *EAP-TTLS* nem lesz olyan sikeres, mint a PEAP.
További *EAP*-variánsok az *EAP-SIM*, a *Microsoft EAP-MSCHAPv2*-je és a *Cisco Lightweight EAP*-ja (*LEAP*).
Álljunk csak meg, bizonyára ezt kérik néhányan; hát a *RADIUS* nem hitelesítő protokoll? Hogyan illeszkedik a képbe? A *RADIUS* az a protokoll, amely felett a hitelesítő, vagyis az *AP* a hitelesítő kiszolgálóval tartja a kapcsolatot.
A *802.1x* és a *WPA* témakörében a *RADIUS*-ra mint szállítóeszközre gondolhatunk, amely felett a hitelesítő továbbítja az *EAP*-üzeneteket a kiszolgálónak. Tehát a végfelhasználó, mint kérvényező *EAP* nyelven beszél a hitelesítővel, a hite-

lesítő pedig *RADIUS*-csomagokba ágyazva továbbítja a kérvényt a kiszolgáló felé. Van egy további protokoll is, mely hasonló szerepet játszik, vagyis a kérvényező és a hitelesítő között közvetít, és ez az *EAPOL*, azaz az *EAP Over LAN*. Ez egy tökéletesen átlátszó protokoll, ugyanis a kérvényező és a hitelesítő oldali programba van beépítve, és mint ilyennek, beállításokat sem kell megadnunk neki. Mondhatnám úgy is, amíg nem *WPA* alapú programot akarunk írni, addig semmi érdemlegeset nem kell tudnunk az *EAPOL*-ról. Attól kezdve, hogy egy kérvényező csatlakozást kezdeményez az *AP* felé, az *AP kizárólag EAP* alapú forgalmat engedélyez. Csak a hitelesítés – a kiszolgáló válaszán alapuló – teljes befejezése után kap a kérvényező *DHCP* bérletet, illetve engedélyt a *WLAN*-hoz való teljes értékű csatlakozásra. A sikeres hitelesítés másik folyománya egy *WEP*-kulcs kiosztása a kérvényezőnek.

A TKIP és a WEP kulcsozás

Ha egy kérvényezőt *EAP-TLS* vagy más titkosított *EAP*-változat segítségével hitelesítünk, akkor a hitelesítési forgalom titkosított. Maguk a vezeték nélküli hálózat keretei viszont nem, hiszen ennek előfeltétele a *WEP* engedélyezése a kérvényező rendszer és a hozzáférési pont közötti kapcsolaton. A megvalósítást végző szempontjából érdekes módon ez a *WPA* használatának legegyszerűbb mozzanata. A sikeres hitelesítés után a kiszolgáló, a hitelesítő és a kérvényező a *TKIP*-t alkalmazzák a hitelesítő és a kérvényező rendszer közötti kapcsolaton érvényes *WEP*-kulcsok egyeztetésére és biztonságos továbbítására. A folyamat nagyrészt átlátszó, a feladat elvégzéséhez sem a kiszolgálón, sem a kérvényezőn nem kell megadnunk semmilyen beállítást. A legtöbb hozzáférési ponton ugyanakkor, ide értve a linuxos *hostapd*-t is, meg lehet adni egyedi beállításokat, például a *WEP* kulcsfrissítési időközt.
A *TKIP* kapcsán érdemes még megjegyezni, hogy, mint említettem is, esetében a kiszolgáló elhagyható. Ha a kérvényezőket és a hitelesítőt előre megosztott kulcsos üzemmódra állítottuk be, a *TKIP*-t akkor is használhatjuk a *WEP*-titkosítás kulcsozására és a kulcsok frissítésére a kérvényező és a hozzáférési pont között.

Összefoglalás – egyelőre

Dióhéjban ennyit a *WPA*-ról. A következő alkalommal a most megismert fogalmakat a *FreeRADIUS* használata kapcsán fogjuk alkalmazni, és összeállítunk egy *Linux* alapú, *WPA*-s hitelesítő kiszolgálót. Aki nem tud addig várni, tanulmányozza az internetes forrásokat. Első a biztonság!

Linux Journal 2005. április, 132. szám

A cikk forrásai: ➲ *www.linuxjournal.com/article/8017*

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.

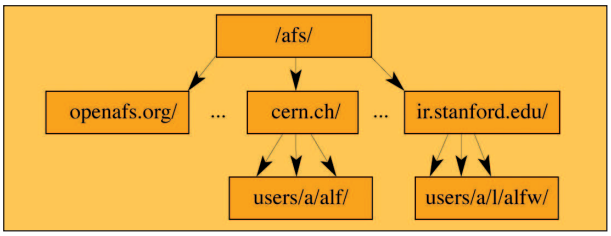
Címtárszolgáltatások (3. rész)
Az AFS – Biztonságos, elosztott fájlrendszer

Egyszeri bejelentkezésre épülő rendszerünket többféle géptípusról is elérhető, elosztott fájlrendszerrel tehetjük teljessé.

Az *Andrew File System (AFS)* egy elosztott, biztonságos, átfogó, az adatok számára helyfüggetlenséget, méretezhetőséget és áttelepítési lehetőséget biztosító fájlrendszer. Az *AFS* sokféle operációs rendszer alól elérhető, és számos nagyobb helyen használják hosszú évek óta. Az *AFS* – közel 20 éves életkora ellenére – egyedi, más elosztott fájlrendszerek által nem biztosított szolgáltatásokat nyújt. Lehet, hogy öregsége miatt egyeseknek nem annyira vonzó, ám miután az *IBM* 2000-ben nyílt forrásúvá tette, használata és fejlesztése új lendületet kapott. Írásomban az *AFS* szolgáltatásait szeretném ismertetni, és szeretnék kedvet csinálni a kipróbálásához.

Az AFS szolgáltatásai és előnyei

Az *AFS* ügyfélprogram *Linux*, illetve a *HP-*, a *Compaq-*, az *IBM-*, a *Sun-* és az *SGI-*féle *UNIX*-változatokhoz, továbbá *Microsoft Windows* és az *Apple Mac OS X*-e alá érhető el. Az *AFS* tehát tökéletes megoldás, ha helyi vagy nagy távolságú hálózaton keresztül többféle operációs rendszer vagy géptípus között szeretnénk adatokat megosztani.
Minden *AFS* ügyfélgép rendelkezik helyi gyorsítótárral. A gyorsítótár-kezelő feladata a gépet igénybe vevő felhasználók követése, valamint a tőlük érkező adatkérések kezelése. Az adatok gyorsítótárazása fájldarabonként történik, a rendszer ezeket az *AFS* fájlkiszolgálóról a helyi lemezre másolja. A gyorsítótárat a gép minden felhasználója igénybe veheti, tartalma az újraindítások során is érintetlen marad. A helyi gyorsítótárazásnak köszönhetően csökken a hálózati forgalom, és a gyorsítótárazott adatok ismételt elérése is felgyorsul.
Az *AFS* egy átfogóan egyedi névtér szerint szerveződik. Az *AFS* fájltér átfogó nézetére láthatunk példát az *1. ábrán*. A fájlokhoz vezető elérési utak neve minden esetben azonos, függetlenül attól, hogy az adatokat honnan érjük el. Az elérési utak nem tartalmaznak kiszolgálóra utaló részt. Mondhatnám úgy is, hogy az *AFS* használója nem tudja, hogy a kívánt adat melyik fájlkiszolgálón található. Ez úgy érhető el, hogy az *AFS* az adatok helyét tároló adatbázist replikálja minden ügyfélre. A megoldás merőben eltér a *Network File Systemtől (NFS)*, amelynél az ügyfélnek ismernie kell az *NFS* adott részének helyt adó fájlkiszolgálót. A különféle, független *AFS* tartományokat sejteknek nevezzük. A sejtek *Kerberos* tartományokhoz tartoznak. Egy jellegzetes *AFS* elérési út így néz ki: */afs/cern.ch/felhasznalo/a/alf/tervezetek/*. Az elérési útban szerepel ugyan az *AFS* sejt neve, ám a fájlkiszolgálóé nem.



1. ábra Az AFS fájltér mindenhonnan azonosnak látszik, használatakor az ügyfélnek nem kell tudnia, hogy melyik könyvtár melyik kiszolgálón található

A helyfüggetlenségnek köszönhetően az *AFS* rendszergazdák úgy mozgathatnak át fájlokat az egyik *AFS* kiszolgálóról a másikra, hogy a felhasználók ebből semmit nem észlelnek. Ugyancsak ez alapozza meg az *AFS* kiváló méretezhetőségét. Ha *AFS* fájlkiszolgálóinkon elfogy a hely, akkor elég egy újat üzembe állítanunk, majd áttelepítenünk rá az adatok egy részét. Az ügyfelek mindebből semmit nem vesznek észre. Az *AFS* a fájlkiszolgálónkénti felhasználók száma tekintetében is kiválóan méreteződik. Egy korszerű gépen egy *AFS* fájlkiszolgáló akár ezer ügyfelet is gond nélkül képes kiszolgálni.
A felhasználók szemszögéből az *AFS* fájltér pontosan úgy néz ki, mint az összes többi fájlrendszer. *Kerberos* hitelesítő adataikkal *AFS* alatt tárolt fájljaikat a világ bármely pontjáról elérhetik, köszönhetően az átfogóan egyedi névtérnek. Nézzünk egy példát: ha a svájci *CERN*-ben lévő kezdőkönyvtáramból a kaliforniai *SLAC*-ben lévő kezdőkönyvtáramba szeretnék másolni valamit, akkor két különböző *AFS*-sejt felé kell hitelesítenem magam:

```
% kinit --afslog alfw@ir.stanford.edu
alfw@ir.stanford.edu's Password:
% kinit -c /tmp/krb5cc_5828_1 --afslog alf@cern.ch
alf@cern.ch's Password:
```

Az *AFS* `tokens` parancsával meg tudjuk jeleníteni a hitelesítő adatokat:

```
% tokens Tokens held by the Cache Manager:
User's (AFS ID 388) tokens for afs@cern.ch [Expires
➥Apr  2 10:30]
User's (AFS ID 10214) tokens for
➥afs@ir.stanford.edu [Expires Apr  2 09:49]
--End of list--
```

A hitelesítés után mindkét kezdőkönyvtáramat el tudom érni:

```
% cp /afs/cern.ch/felhasznalo/a/alf/
➥tervezetek/X/src/hello.c \
   /afs/ir.stanford.edu/felhasznalok/a/l/alfw/
   ➥tervezetek/Y/src/.
```

Az *AFS* fájlkiszolgálók az adatokat különleges, */vicepXX* nevű lemezrészeken tárolják, ahol az XX az „a-zz" tartományba esik, vagyis egy-egy kiszolgálón összesen 256 lemezrész lehet. A lemezrészek mindegyikén adatkonténerek találhatók, ezeket köteteknek nevezzük. A kötetek a legkisebb mozgatásra, replikálásra vagy biztonsági mentés készítésére használható egységek. A köteteken belül könyvtárak és fájlok vannak. A kötetek csak azt követően válnak láthatókká, hogy befűzzük őket az *AFS* fájltérbe. Ezek a befűzési pontok pontosan olyanok, mint a könyvtárak.
Az *AFS* az ügyféloldali gyorsítótárazásnak köszönhetően különösen jól használható csak olvasható adatok, például a */usr/local/* fa tárolására. Az ilyen megoldások működésének javítását, üzembiztosságának fokozását az *AFS* azzal segíti, hogy lehetővé teszi a csak olvasható adatok másolatainak különböző *AFS* fájlkiszolgálókon való elhelyezését. Ha a példányokat tároló kiszolgálók valamelyike leáll, az ügyfél észrevétlenül átvált az adatok egy másik példányát tároló kiszolgálóra. Ugyanezzel a replikációs megoldással történik a földrajzilag távol lévő kiszolgálók közötti adatmásolás is. Az ügyfeleket be lehet úgy állítani, hogy lehetőleg a közeli másolatot használják, a távoli példányhoz csak meghibásodás esetén forduljanak. Az *openafs.org AFS* sejt például két kiszolgálóról fut, az egyik a *Pennsylvania* állambéli *Pittsburgh*-i *Carnegie Mellon Egyetemen*, a másik svédországi, stockholmi *Royal Institute of Technology (KTH)* intézetben található.
Az *AFS* pillanatképek készítésének lehetőségével segíti a biztonsági másolatok létrehozását. A pillanatképek létrehozása az általunk kiválasztott időpontban történik, és mindig kötetekről készülnek. A pillanatképeket ezután az egyéb felhasználói műveletek zavarása nélkül menthetjük le például szalagra, de külön befűzési pontokon, *AFS* kezdőkönyvtárukon belül akár a felhasználók számára is elérhetőkké tehetjük őket. Ezzel az egyszerű trükkel megelőzhetjük, hogy állandóan mentési és visszaállítási kérésekkel zaklassanak a felhasználók, hiszen az utolsó éjszaka készült pillanatképekben lévő fájlok szabadon hozzáférhetők.
Az *AFS* adatcsere protokollját nagy távolságú hálózatokhoz tervezték. Saját ***távoli eljáráshívás (remote procedure call, RPC)*** megvalósítást használ, ennek neve *Rx*, és *UDP* felett működik. A protokoll minden csomagkötetből csak az esetlegesen meghibásodott csomagokat küldi újra, és más protokollokhoz képest több nyugtázatlan csomag kint létét engedi meg.
Az *AFS* felügyelete bármelyik *AFS* ügyfélről elvégezhető, tehát semmi szükség nincs arra, hogy bármelyik *AFS* kiszolgálóra bejelentkezzünk. A rendszergazda tehát „szorosra zárhatja" az *AFS* kiszolgálókat, ami biztonsági szempontból kétségtelenül előnyös. Az *AFS* alatt tárolt adatok helyfüggetlensége szintén megkönnyíti a felügyelhetőséget; például egy *AFS* fájlkiszolgálót a köteteket más kiszolgálókra mozgatva akár teljesen – ráadásul a felhasználók számára észrevétlenül – ki tudunk üríteni. Az üres kiszolgálón ezután elvégezhetjük a szükséges karbantartásokat, legyen szó akár az operációs rendszer frissítéséről, akár a hardver javításáról. Amikor végeztünk, csak – ismét átlátszó módon – vissza kell mozgatnunk rá a köteteket.
Az *AFS* belül *Kerberost* alkalmaz a felhasználók hitelesítésére. Ez eredetileg a *Kerberos 4*-es változata, ám a Kerberos 5 bármelyik jelentősebb megvalósítása mellett is dönthetünk, ha fokozni szeretnénk a biztonságot. Az ***AFS hozzáférés-vezérlési listák (access control list, ACL)*** alapján korlátozza a könyvtárak elérését. Az *ACL*-ekben csak *Kerberos* főfiókok és ezek csoportjai szerepelhetnek, ellentétben az *NFS*-sel, amelynél kizárólag unixos felhasználóazonosítókkal történik a hitelesítés. Emellett egy további jogosultságkezelő szolgáltatás, a ***védelmi szolgáltatás (protection service, PTS)*** is figyelemmel követi az egyes *Kerberos* főfiókokat és főfiókcsoportokat.

Az AFS összetevői

Mindezen szolgáltatások biztosításához az *AFS*-nek számos különböző összetevőre van szüksége. Az *AFS* ügyfélprogramnak minden az *AFS* fájltér elérésére használni kívánt számítógépen futnia kell. Az *AFS* kiszolgáló program négy alaprészből áll. Hitelesítésre *Kerberost* használ, a jogosultságkezelést a *PTS* végzi, a kötethely kiszolgáló a helyfüggetlenséget teszi lehetővé, az adatszolgáltatásért pedig egy fájl- és egy kötetkiszolgáló felelős. Az egyes folyamatokat az *AFS* kiszolgálókon az ***alapszintű felügyelő (basic overseer, BOS)*** kiszolgáló kezeli. Mindezen feltétlenül szükséges összetevők mellett további démonok is léteznek az *AFS* kiszolgálók karbantartására és tartalmuk biztonsági mentésére. Az *AFS* kiszolgálók telepítése sajnos túlmutat témánkon. A különféle kiszolgáló összetevők miatt az *AFS* megismerésének első lépései egyenesen riasztók. Mégis érdemes megküzdeni vele, mára sok helyen létezni sem tudnának nélküle. Ha egy sejtet telepítettünk, akkor az *AFS* napi szintű karbantartását körülbelül 2 órában el tudjuk végezni, még nagyméretű telepítéseknél is.
Akit részletesebben is érdekel, hogy mások, például a *Morgan Stanley* és az *Intel* hogyan és mire használják az *AFS*-t, az vessen egy pillantást a legutóbbi *AFS Best Practices Workshopon* szerepelt bemutatóra (lásd az internetes forrásokat).

Az AFS ügyfelek telepítése

Az *AFS* kipróbálásához nincs szükség saját *AFS* kiszolgálóra. Elég, ha telepítjük az *OpenAFS* ügyfélprogramot, majd egy különleges, az idegen *AFS* sejtek nyilvánosan elérhető tereinek használatát engedélyező kapcsolóval indítjuk el az *AFS* ügyféldémont (`afsd`). Az *AFS* ügyfelek telepítésének legnehezebb mozzanata a szükséges rendszermagmodul beszerzése. Ha *Red Hat* vagy *Fedora* rendszert használunk, töltsük le a megfelelő *RPM*-eket (lásd a forrásokat). A rendszermagmodul mellett az *AFS* ügyfélnek egy felhasználói térbeli démonra (`afsd`), valamint az *AFS* parancskészletre is szüksége van. Ezek két további *RPM*-ben találhatók meg.
Ha megvannak a modulok, a következő lépés az *AFS*-ügyfél igényeink szerinti beállítása. Elsőként azt kell megadnunk,

hogy számítógépünk melyik sejt tagja legyen. Az *AFS* sejt neve a */usr/vice/etc/ThisCell* fájlban van megadva. Ha saját *AFS* kiszolgálókkal nem rendelkezünk, akkor a név tetszőleges, egyébként viszont az általuk kiszolgált sejt nevét kell beállítanunk. A következő beállítási érték az *AFS* helyi gyorsítótárával kapcsolatos. Az AFS-ügyfeleken az ügyfélprogramot külön lemezrészre kell telepíteni, a gyorsítótárat viszont bárhova tehetjük. A gyorsítótár helyét és méretét a */usr/vice/etc/cacheinfo* fájlban adhatjuk meg. A gyorsítótár alapértelmezett helye a */usr/vice/cache*, mérete pedig 100 MB, ami egy átlagos asztali vagy hordozható számítógép esetében bőségesen elegendő. Az *openafs-client RPM* telepítése után ezekkel a beállításokkal kezdünk, ilyenkor a *cacheinfo* fájlban az alábbi beállítást láthatjuk:

```
/afs:/usr/vice/cache:100000
```

Következőként az `afsd`, vagyis az *AFS* ügyféldémon a */etc/sysconfig/afs* fájlban található beállításait kell testreszabnunk. Az `OPTIONS` meghatározáshoz adjuk hozzá a `-dynroot` kapcsolót, így az AFS-ügyfél saját *AFS* kiszolgáló nélkül is el tud indulni.
Egy további fontos beállítás a `-fakestat`. Hatására az `afsd` meghamisítja a */afs/* könyvtárban lévő bejegyzések `stat(3)` adatait. Hiányában az *AFS* ügyfél világgá indulna, és minden számára ismert *AFS* sejttel kapcsolatba lépne. Ez jelenleg 133 sejtet jelent, amint a */afs/* könyvtárban kiadott hosszú listázással (`/bin/ls -l`) magunk is meggyőződhetünk róla.
Mivel az *AFS Kerberost* használ a hitelesítésre, gépünkön vagy gépeinken az időt is szinkronizálnunk kell. Az *AFS* ugyan saját szinkronizálási megoldással is rendelkezett, ám ez mára elavult, használatát nem javaslom. Kapcsoljuk is ki; ehhez a */etc/sysconfig/afs* fájl `OPTIONS` meghatározásához a `-nosettime` kapcsolót kell hozzáadnunk. Ha nincs kész megoldásunk az idő szinkronizálására, akkor ismerkedjünk meg a *hálózati idő protokollal (Network Time Protocol, NTP*; lásd a forrásokat).
Mindezen módosítások végrehajtása után a */etc/sysconfig/afs* fájl `OPTIONS` meghatározásának így kell kinéznie:

```
OPTIONS="$MEDIUM -dynroot -fakestat -nosettime"
```

Az utolsó lépés az *AFS* fájlrendszer befűzési pontjának létrehozása, amit a

```
% sudo mkdir /afs
```

parancs kiadásával végezhetünk el. Ezután a

```
% sudo /etc/init.d/afs start
```

paranccsal indíthatjuk el az AFS-ügyfelet. Az indítás eltart néhány másodpercig, ugyanis az `afsd`-nek indulása előtt fel kell töltenie a helyi gyorsítótárat. Mivel a gyorsítótár az újraindítások során is megőrzi tartalmát, a későbbi indítások már gyorsabbak lesznek.

Az AFS-világ felfedezése

Ha nincs is saját *AFS*-kiszolgálónk, de *AFS*-ügyfelünk beállításait a fentiek szerint módosítottuk, akkor az *AFS* használatával kapcsolatos parancsok egy részét, illetve a világszintű *AFS* teret egyedül is megismerhetjük.
Egy gyors ellenőrzéssel láthatjuk, hogy egyetlen *AFS* sejtben sem vagyunk hitelesítve:

```
% tokens
Tokens held by the Cache Manager:
--End of list--
```

A listában – a korábbi példával ellentétben – nincsenek hitelesítő adatok.
Első lépésként kérdezzük le a */afs/* könyvtár tartalmát. Ekkor az *AFS* ügyfelünk által ismert *AFS* sejtek mindegyike megjelenik. Most lépjünk át a */afs/openafs.org/software/openafs* könyvtárba, és listáztassuk ki ennek tartalmát is. A következőket kell látnunk:

```
% ls -l
total 10
drwxrwxrwx  3 root  root  2048 Jan 7  2003  delta
drwxr-xr-x  8 100   wheel 2048 Jun 23 2001  v1.0
drwxr-xr-x  4 100   wheel 2048 Jul 19 2001  v1.1
drwxrwxr-x 17 100   101   2048 Oct 24 12:36 v1.2
drwxrwxr-x  4 100   101   2048 Nov 26 21:49 v1.3
```

Lépjünk be valamelyik könyvtárba. Például:

```
% cd v1.2/1.2.10/binary/fedora-1.0
```

Vessünk egy pillantást a könyvtárban lévő *ACL*-ekre:

```
% fs listacl .
Access list for . is
Normal rights:
  openafs:gatekeepers rlidwka
  system:administrators rlidwka
  system:anyuser rl
```

A fentiekben két csoportot láthatunk, mindkettőhöz mind a hét jogosultság hozzá van rendelve: olvasás (`r`, *read*), keresés (`l`, *lookup*), beillesztés (`i`, *insert*), írás (`w`, *write*), teljes, önkéntes érvényű fájlzárolás (k) és ACL-módosítás (a).
A `system:anyuser` egy különleges, az AFS-hez tartozó csoport, olvasási (r) és keresési (l) joggal rendelkezik, így lényegében mindenkinek hozzáférést biztosít.
Egy csoport tagjait a `pts`, a védelmi szolgáltatás megfelelő parancsával adhatjuk meg:

```
% pts member openafs:gatekeepers -cell openafs.org
➥-noauth
Members of openafs:gatekeepers (id: -207) are:
  shadow
  rees
  zacheiss.admin
  jaltman
```

A `-noauth` kapcsolót azért használjuk, mert a parancsot erre a sejtre nézve hitelesítő adatok nélkül futtatjuk.
Az *AFS* hitelesítési részének – amely amúgy normál *Kerberos* – felderítéséhez különleges felügyeleti jogokra van szükség, ezért ettől most eltekintünk. Inkább nézzük meg, hogy a jelenlegi könyvtár fizikailag hol található:

```
% fs whereis .
File . is on hosts andrew.e.kth.se
➥VIRTUE.OPENAFS.ORG
```

A kimenet szerint a könyvtárból két példány létezik, az egyik az *andrew.e.kth.se*, a másik a *VIRTUE.OPENAFS.ORG* címen érhető el.

A
```
% fs lsmount /afs/openafs.org/software/openafs
?/v1.2/1.2.10/binary/fedora-1.0
/afs/openafs.org/software/openafs/v1.2/1.2.10/
➥binary/fedora-1.0
? is a mount point for volume #openafs.1210.f10
```

parancs kimenete szerint ez a könyvtár valójában az `openafs.1210.f10` nevű ***AFS*** kötet befűzési pontja.
A kötetet egy másik paranccsal vizsgálhatjuk meg:
```
% vos examine openafs.1210.f10 -cell openafs.org
➥-noauth
```

A fenti parancs az ***openafs.org AFS***-sejtben található `openafs.1210.f10` kötet írható-olvasható változatáról szolgáltat részletesebb adatokat. A kimenetnek így kell alakulnia:

```
openafs.1210.f10       536871770 RW   25680 K
➥On-line
    VIRTUE.OPENAFS.ORG /vicepb
    RWrite  536871770 ROnly  536871771 Backup     0
    MaxQuota          0 K
    Creation    Fri Nov 21 17:56:28 2003
    Last Update Fri Nov 21 18:05:30 2003
    0 accesses in the past day (i.e., vnode
    ➥references)

    RWrite: 536871770     ROnly: 536871771
    number of sites -> 3
       server VIRTUE.OPENAFS.ORG partition /vicepb
       ➥RW Site
       server VIRTUE.OPENAFS.ORG partition /vicepb
       ➥RO Site
       server andrew.e.kth.se partition /vicepb RO
       ➥Site
```

A kimenet értelmében a kötet a `VIRTUE.OPENAFS.ORG` állomás */**vicepb*** lemezrészén található. A következő sorban az írható-olvasható és a csak olvasható kötetek kötetazonosítói láthatók, némi statisztika társaságában. Az utolsó három sor a kötet egy írható-olvasható (`RW Site`) és két csak olvasható (`RO Site`) másolatának helyét adja meg.
Ha kíváncsiak vagyunk rá, vajon hány további ***AFS*** lemezrész található a `VIRTUE.OPENAFS.ORG` kiszolgálón, alkalmazzuk a következő parancsot:
```
% vos listpart VIRTUE.OPENAFS.ORG -noauth
```

Segítségével a következő lemezrészekről szerezhetünk tudomást:
```
/vicepa      /vicepb      /vicepc
Total: 3
```

Vagyis a gépen összesen három */**vicep*** lemezrész van.
Ha látni szeretnénk, hogy hány kötet található a kiszolgáló */**vicepa*** lemezrészén, adjuk ki az alábbi parancsot:
```
% vos listvol VIRTUE.OPENAFS.ORG /vicepa -noauth
```

A parancs végrehajtása eltart egy kis ideig, végül 275 kötet listáját látjuk. A lista első néhány eleme a következő:

```
Total number of volumes on server VIRTUE.OPENAFS.ORG
➥partition /vicepa: 275
openafs.10.src           536870975 RW   11407 K On-line
openafs.10.src.backup    536870977 BK   11407 K On-line
openafs.10.src.readonly  536870976 RO   11407 K On-line
openafs.101.src          536870972 RW   11442 K On-line
openafs.101.src.backup   536870974 BK   11442 K On-line
openafs.101.src.readonly 536870973 RO   11442 K On-line
```

Érdemes még ismerni a `bos` parancsot, amely egy sejt alapszintű felügyelő kiszolgálójával veszi fel a kapcsolatot, és meghatározza a rajta futó ***AFS*** kiszolgáló folyamatok állapotát. Az `fs`, a `pts`, a `vos` és a `bos` parancshoz számos további alparancs is tartozik. Szerencsére az AFS parancsainak mindegyike érti a `help` kapcsolót (elé nem kell kötőjelet írni), ennek segítségével megjeleníthetjük az alparancsokat is. Az `fs <alparancs> -help` (itt már kell a kötőjel) segítségével az egyes alparancsok szintaxisát is lekérdezhetjük.

Az AFS jövője

Jelenleg is számos az ***AFS*** fejlesztését célzó tervezet van folyamatban. A legfontosabb ezek közül az ***AFS***-nek a 2.6-os ***Linux*** rendszermagok alatti működését lehetővé tévő. Ezeknél a rendszermagoknál már nem érhető el szabadon a ***syscall*** tábla. Egy másik tervezet a kapcsolat nélküli üzemmód támogatását célozza, ennél az ***AFS***-ügyfelek hálózati kapcsolatuk megszakítása után is folytathatnák az ***AFS*** használatát, az ***AFS*** tér és a fájlok tartalma a kapcsolat ismételt létrejötte után kerülnének szinkronizálásra.

Összefoglalás

Bár az ***AFS***-t minden oldaláról egy csapásra megismerni gyakorlatilag lehetetlen, az ***AFS*** sejtek üzembe helyezésének folyamatát megismerni pedig komoly munka, az ***AFS***-t mégis kifizetődő saját infrastruktúránkon belül használni. Segítségével a biztonságos, gépfüggetlen, világszintű fájlmegosztás ugyanolyan könnyedén megoldható, mint a */**usr**/**local**/* könyvtárág vagy a unixos kezdőkönyvtárak tárolásának leegyszerűsítése. Ráadásul hosszú távon felügyeleti terheink növekedésével sem kell számolnunk.

Linux Journal 2005. április, 132. szám

A cikkhez tartozó források elérhetősége:

➲ ***www.linuxjournal.com/article/8079***

Dr. Alf Wachsmann 1999 óta a Stanford Linear Accelerator Center (SLAC) munkatársa. Ő felelős az önműködő Linux-telepítések minden mozzanatáért, egyaránt ide értve a farmok csomópontjainak, a kiszolgálóknak és az asztali gépeknek a kezelését. Munkája során elsősorban az aktív fájlkészletek (AFS) támogatásával, a Kerberos 5-re való áttéréssel, egy felhasználónyilvántartó tervezettel és felhasználói tanácsadással foglalkozik.

Szaktekintély, immár századszor

Eleinte minden aprósághoz CGI parancsfájlokat írtunk; mára viszont eljutottunk a webes eszközök és keretrendszerek pazar bőségéhez. Vajon milyen lesz a webes fejlesztések jövője?

Köszöntök mindenkit a Szaktekintély rovat századik írásának megjelenése alkalmából! Bizony, ez már a századik cikk, amit a ***Linux Journal*** számára, illetve korábban a SSC ***Websmith*** című kiadványa számára 1996 tavasza óta írtam. Az évek során rendkívül sok örömömet leltem abban, hogy hónapról hónapra a webes és kiszolgáló oldali megoldások egy-egy újabb elemét mutathattam be. Ebben a hónapban szeretnék kicsit visszatekinteni a kiszolgáló oldali és a web/adatbázis-programozás múltjára – így talán a jelenlegi helyzetet is jobban tudjuk majd értékelni. Ezután megvizsgáljuk a web mostani állapotát, és megpróbáljuk felmérni, vajon mire számíthatunk az eljövendő években.

Visszatekintés

Manapság a webet és az internetet mint természetesen meglévő dolgot fogadjuk el. A banki ügyeimet weben keresztül intézem; könyveket vásárlok online boltokból; webes ***RSS***-olvasóval webnaplókat olvasok; a webes újságokat gyakrabban látogatom, mint ahogy nyomtatott változatukat kézbe vettem; azonnali üzenetküldő programokon keresztül csevegek a barátaimmal és ismerőseimmel; sőt, még a fizetségeimet is a ***PayPalon*** keresztül kapom. Sokszor hallottam, hogy Manhattan lakóinak soha nem muszáj kimozdulniuk otthonukból, mert mindent házhoz lehet szállíttatni. Nem akarom megítélni, hogy ez jó vagy sem, de tény, hogy az internet világszerte egyre több ember számára teszi mindezt elérhetővé.

Az internet üzleti és szórakoztató jellege egy dinamikus folyamat eredményeként alakult ki. A webkiszolgálók eredetileg előre összeállított, nyers vagy ***HTML*** formázású, szöveges dokumentumok megosztására szolgáló megoldások voltak. Röviddel az után, hogy a weben közzétett, viszonylag korlátozott számú dokumentum felfedezése egyre népszerűbb tevékenység lett, valaki rájött, hogy a ***HTTP*** ügyfél-kiszolgáló jellegének köszönhetően a dokumentumokat dinamikusan, a beérkező kérésekre válaszul is elő lehetne állítani. Amikor egy ***HTTP***-ügyfél elküldi adott dokumentumra vonatkozó kérését a kiszolgálónak, akkor még nem tudja, hogy a dokumentum hónapok óta ott várakozik a kiszolgáló fájlrendszerében, vagy a kérésre válaszul állítják elő. Ez a szemlélet aztán örökre átalakította a webet, egyszerű, statikus anyagok megosztott tárháza helyett valós idejű dokumentumok és alkalmazások rendszerévé formálva.

A dinamikus forradalom kezdete meglehetősen egyszerű volt. Az első dinamikusan előállított tartalmak inkább burkolók voltak olyan unixos parancsokhoz, mint a ***mail*** és a ***finger***. Barátaimmal együtt készített első programjaim egyike például egyszerűen arra volt alkalmas, hogy kereséseket végezzünk újságunk online archívumában. Természetesen megtehettük volna, hogy különleges ***HTTP***-kiszolgálókat írunk a kívánt szolgáltatások biztosítására. Szerencsénkre azonban – és a többi webes fejlesztő szerencséjére – az ***NCSA httpd***, az ***Apache*** elődjének tervezői a ***közös átjárófelület (common gateway interface, CGI)*** révén az összes a kiszolgálón található program számára lehetővé tették a ***HTTP*** alapú kommunikációt. A ***CGI*** révén a kiszolgáló bármely programját elérhetővé tudtuk tenni a weben keresztül, egész egyszerűen egy ***CGI*** programba burkolva.

Az első években kemény idők járták. Mindenki feltételezte, hogy a web eleve állapot nélküli, és mindenki kitörő örömmel fogadta, amikor a Netscape bejelentette a sütiket (cookie), amelyek segítségével a kiszolgálók figyelemmel követhették a felhasználókhoz egyedileg rendelt adatokat. Nem voltak a webes forgalom mérésére alkalmas programok, nem is beszélve a webes programozás alacsonyabb szintjeit elfedő könyvtárakról. A hibakeresés nagyjából a webkiszolgáló hibanaplójának figyelésében merült ki. Minden olyan dolognak a használata, amely bonyolultabb volt egy egyszerű szöveges fájlnál, már fejlett, különleges adattárolási megoldásnak számított.

Itt és most

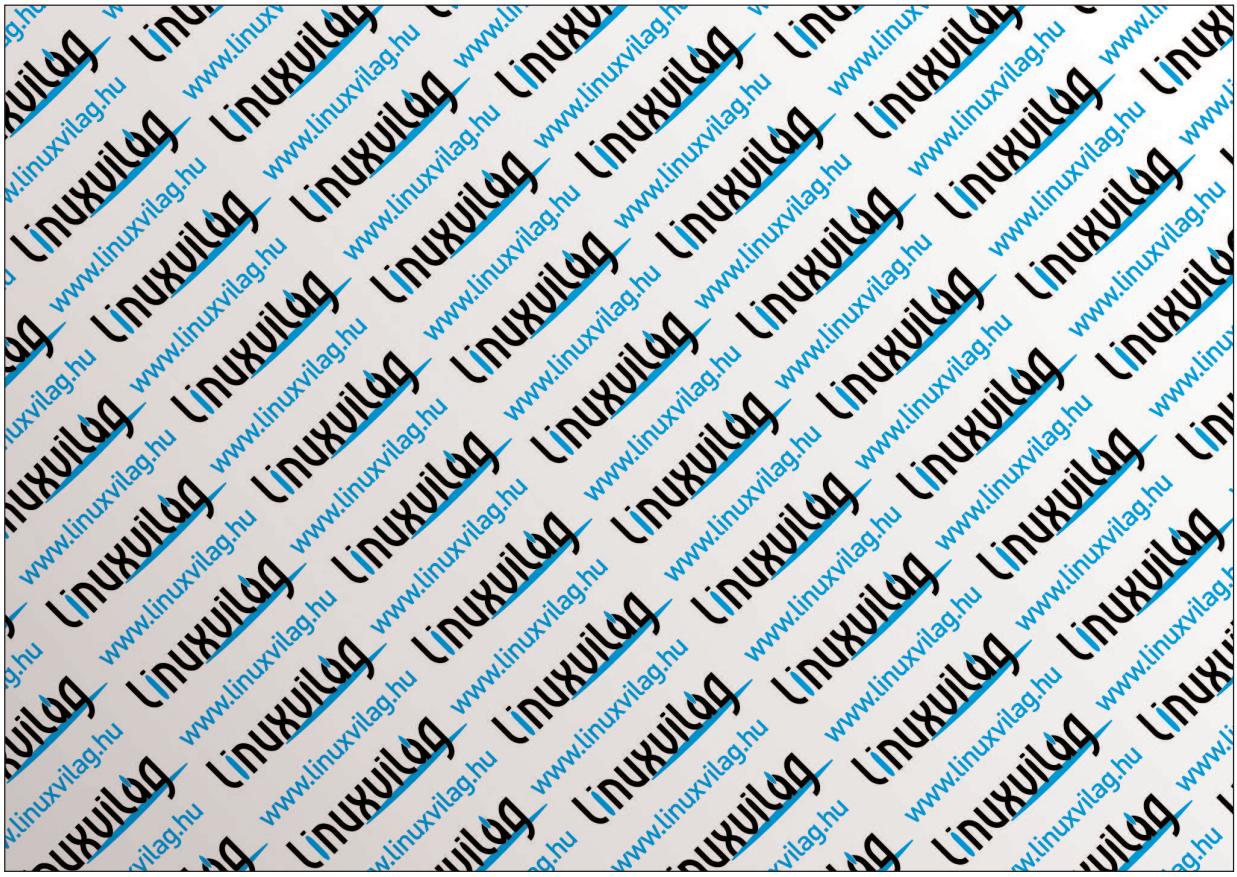
Napjainkra a webes fejlesztés teljesen megváltozott. Az Apache legújabb változatának letöltése és telepítése gyerekjáték, a ➲ ***www.apache.org*** oldal megnyitása után néhány perccel már profin beállított webkiszolgáló lehet a gépünkön. Relációs adatbázis nélkül, még ha esetleg nem is akarjuk bevallani, ma már nem létezik normálisabb webes alkalmazás. A legtöbb időt azzal takaríthatjuk meg, hogy többé nem is kell saját programokat írnunk – a rendelkezésünkre álló, a web és adatbázis alapú programok készítését segítő alkalmazások, könyvtárak és keretrendszerek száma egészen lenyűgöző. Korábban égen-földön kutatnunk kellett, ha találni akartunk egy az igényeinknek megfelelő, nyílt forrású alkalmazást. Kétségtelen, hogy a leginkább megfelelő alkalmazás megtalálása most sem

feltétlenül könnyű, ám ennek csak az az oka, hogy rengeteg gyenge vagy nem odaillő programot kell megismernünk, mire rálelünk az igazi megoldásra. Mindemellett a fejlesztői közösség is rengeteget fejlődött az évek során. A kiszolgáló oldali programozás világába kezdőként belépők jóindulatban és segítségben sosem szenvedtek hiányt, ám a kellő tapasztalat felhalmozódásához idő kellett. A webes programozás egykor kutatólaborok hálózatához hasonlított, amelyben minden résztvevő megosztotta tapasztalatait a közösség többi tagjával. Mára komoly tapasztalat gyűlt össze, a nyílt közösségben és a vállalatok falai mögött egyaránt. Ha egy ifjú programozó új alkalmazásokat szeretne írni, akkor szinte végtelen mennyiségű könyv, weboldal és forráskód áll rendelkezésére a tanuláshoz.

Arról sem szabad elfeledkezni, hogy az ezen a területen használt, népszerű programozási nyelvek, mint a *Perl*, a *Python*, a *PHP* és a *Java* az elmúlt évek során rengeteget fejlődtek. Engem ugyanakkor ezeknek a nyelveknek és könyvtáraiknak fejlődése kevésbé lepett meg, mint az, hogy az iparág egyre inkább a magas szintű nyelvek használata felé halad.

A webes világ hajnalán a legtöbben *C* és *C++* nyelven fejlesztettek. Akik magas szintű nyelveket, például *Perlt* vagy *Pythont* használtak, azokat szinte kontároknak tekintették, a „rendes” nyelveket használókhoz képest komolytalannak számítottak. A web mindezt megváltoztatta: ma már az is lehet komoly alkalmazásfejlesztő, aki csupán *PHP*-ben dolgozik. Természetesen a lefordított *C* programok ma is gyorsabban futnak, mint az azonos feladatokat ellátó, de magas szintű nyelveken készültek, ám utóbbiaknak a fejlesztési és hibakeresési időben észlelhető előnye viszont annyira nagy, hogy ma már szinte senki nem ír *C*-ben webes alkalmazást. Szintén jól látható folyamat, hogy a nagyvállalatok egyre inkább magas szintű nyelveket alkalmaznak, és a nyílt forrású programokat is elfogadják. Sok vállalat – például az Amazon vagy az *eBay* már rájött, hogy programozói jóval termelékenyebbek, ha magas szintű nyelvekkel dolgoznak. Az a tény, hogy ma már a *Java* és *C#* a két legalacsonyabb szintű webes fejlesztésre használt nyelv, elég sokat elmond arról, hogy hová tart az iparág. Olyan nyelvek vették át az uralmat, amelyek lehetővé teszik, hogy a programozó végre a valódi ötletekkel foglalkozzon, és ne biteket és bájtokat piszkáljon naphosszat. A *Java*, azt hiszem, kijelenthetjük, mint asztali programozási nyelv megbukott, ám a *C#* a jelek szerint, köszönhetően a *Microsoft .NET* kezdeményezésének, egyre népszerűbb; így nem kizárt, hogy néhány év múlva a legtöbb asztali alkalmazás olyan nyelveken készül, amelyek nem tartalmaznak mutatókat, az automatikus szemétgyűjtést ellenben elvégzik.

Természeten az ilyen nyelvek terjedésének okai szerteágazók, műszaki és pénzügyi jellegűeket egyaránt találunk közöttük. Biztos vagyok abban, hogy a web kiemelkedő szerepet játszott az irányváltásban. A magas szintű nyelvek, mint a *Perl*, kiválóan megfelelnek a webes környezet igényeinek, mint furcsa adattípusok kezelése, adatbázis-kapcsolat, könnyű használat, sokoldalú szövegkezelési lehetőségek és könyvtárak. A web nem más, mint hálózatra kihajított szövegek halmaza, márpedig ezeket a legmesszebbre magas szintű, nyílt forrású nyelvek segítségével hajíthatjuk.



Elképesztő növekedés állt be a kiszolgáló oldali alkalmazások készítésére szolgáló keretrendszerek számában is. Hiába rendelkezik valaki magas szintű programozási nyelvvel, ha saját rendszert kell írnia a felhasználók, a csoportok, az engedélyek, a tartalom és az üzenetek kezelésére. Valamelyik meglévő keretrendszert használva megúszhatjuk ezt a munkát, és felhasználhatjuk valaki másnak a tapasztalatait. A keretrendszerek két alapvető irányt követnek: egy részük tartalomkezelésre, híroldalak és magazinok oldalainak valós idejű összeállítására szolgál, mások viszont alkalmazáskiszolgálóként üzemelnek, vagyis alkalmazások készítésére használható eszközkészletekkel látják el a fejlesztőket.
Felületesen szemlélve azt hinnénk, hogy az olyan alkalmazási keretrendszerek, mint a ***HTML::Mason***, a ***Zope***, az ***OpenACS*** és a ***Java servletek/JSP***-k kevés közös tulajdonsággal rendelkeznek. Aki azonban egynél többet is megismer közülük, az hamar rájön, hogy bár mindegyik sajátos szemléletet követ, sokban azonosak. Igaz, hogy egyik keretrendszerről a másikra áttérni továbbra is fájdalmas, ám aki már többet is felfedezett közülük, annak egy-egy újabb megismerése rutinműveletté válik.
Igen, webes fejlesztőnek lenni 2005-ben messze kellemesebb ahhoz képest, amit tíz évvel ezelőtt ki kellett állnunk. A szoftverek egyre kiforrottabbak, a közösség kiterjedt és segítőkész, nem kell többé minden héten feltalálni a kereket, és a web felé nyitó szervezetek száma egyértelműen azt jelzi, hogy a munkákra van igény a piacon.

A jövő

Miután ennyit áradoztam a jelenlegi helyzetről, vajon mi vár ránk a jövőben? Milyen folyamatok fognak felgyorsulni a 2005-ös év során? Először is, szerintem egyértelmű, hogy a web, ami alatt itt a ***HTTP***, a ***HTML*** és az ***URL***-ek együttesét értem, felbomlik alkotó elemeire. Mindig is úgy gondoltam, hogy a web szokatlanul sokoldalú, hiszen három önmagában is nagy tudású megoldásból – ezek lennének a ***HTTP***, a ***HTML*** és az ***URL***-ek – áll össze. Tisztán látható, hogy mindhárom megoldás önmagában is megállja a helyét, és más területeken is meg fog jelenni.
Különösen érdekesek a webszolgáltatások, amelyek egy a böngészőktől különböző programok számára készült, új, sokoldalú és nyílt kommunikációs protokollt képviselnek. Amikor először megjelentek, azt hittem, hogy valami egyszerű dologról van szó, aminek kiötlői megpróbálják kihasználni a web sikerét és nevét. Az lehet, hogy a szegényes névválasztás tekintetében igazam volt, sőt, talán a hátterelméletek sem túl összetettek, ám mindez mit sem változtat azon, hogy a webszolgáltatások valóban komoly lehetőségeket kínálnak. Az az ötlet, hogy adott alkalmazás operációs rendszertől és programozási nyelvtől függetlenül kapcsolódhasson egy másikhoz, egyszerű, mégis zseniális. Bár a webszolgáltatások igazán jó használatára még ritkán látunk példát, az ***Amazon***, a ***Google*** és a ***Bloglines*** már igazolta, hogy belső ***API***-jaink ügyfelek és más külső személyek számára való elérhetővé tétele nem feltétlenül veszélyezteti üzletmenetünk biztonságát.
Hasonló irányzat a webböngészők asztali alkalmazások belső összetevőjeként való használata. A súgók már ***HTML*** alapúak, és mini webböngészőkkel működnek, de vannak teljes alkalmazások is, mint például az ***ActiveState Komodoja***, amelyek például a ***Mozilla*** motorra épülnek. Sokszor mondtam, hogy a Mozilla az új ***Emacs***. Noha a ***Mozilla*** alapú fejlesztés jóval nehezebb, mint az ***Emacs*** testreszabása valaha is volt, az a tény, hogy a ***Mozilla*** többféle operációs rendszer felett is futó, programozható környezetet biztosít sokoldalú asztali alkalmazások készítéséhez, önmagában is figyelemre érdemes. Ígéretes alkalmazás a ***Sunbird***, a ***Mozilla*** naptárprogramja is, amelyet a saját gépemen én is hónapokon keresztül használtam. A ***Sunbird*** ugyan számos hibától és problémától terhes, mégis roppant rokonszenvessé teszi, hogy az ***iCalendar*** szabványt használja különféle naptáraknak az internetről, ***HTTP***-n keresztül végzett letöltéséhez. Bizony! Pontosan erről volt szó: olyan asztali alkalmazás, amely ***Mozilla*** alapú, ***HTTP***-n keresztül ***URL***-eket kérdez le, mégsem webböngésző!
Kiszolgáló oldalon az együttműködés egyre hangsúlyosabb szerepet kap. Bár egy kereskedelmi enciklopédia színvonalát nem feltétlenül éri el, én mégis a ***Wikipediához*** fordulok először, ha valamilyen témáról bővebb ismereteket szeretnék gyűjteni. Köszönettel tartozunk a több ezer szerkesztőnek, munkájuk eredménye mindennapos használatra több mint kiváló. Egy ilyen jellegű együttműködést vezényelni nem kis feladat, és a ***WikiMedia Foundation PHP*** és ***MySQL*** alapú ***MediaWiki*** alkalmazása szép csendben élvonalbeli közös írást és szerkesztést segítő megoldássá vált.
Végül, hibakereső és tesztelő keretrendszerekre mindig is szükség lesz. A jövőben egyre kiterjedtebb tesztelésekre, sőt, tesztelés alapú programozásra kell felkészülnünk. A részegységek ellenőrzése alapján soha nem lehet egyértelműen megállapítani, hogy vajon a teljes alkalmazás megfelelően fog-e működni, ám ennek ellenére nyilvánvaló, hogy minden eljárást tüzetesen ellenőrizni kell, mielőtt összeépítenénk őket. A tesztelés alapú fejlesztés előtérbe kerülése az utóbbi néhány év egyik legfontosabb módszertani változása, és hiszem, hogy népszerűsége az alkalmazások bonyolultságával párhuzamosan fog növekedni.

Összefoglalás

Őszintén örülök, hogy alkalmat kaptam immár száz cikk megírására. Ám a fentiekből is kitűnik, a webes megoldásokkal, adatbázisokkal foglalkozó fejlesztőket számos újabb kihívás várja, vagyis bőségesen lesz témám további száz íráshoz. A következő hónapok során több itt említett ötletet is meg fogunk vizsgálni, ide értve az ***iCalendart***, a ***Wiki*** szoftvert, a webszolgáltatásokat és a tesztelés alapú fejlesztést. Lehet, hogy elmúlt tíz éves, a webbel dolgozni mégis szórakoztató, izgalmas és sok fejtörést okoz.
A ***reuven@lerner.co.il*** címen bárki levelét szívesen fogadom, ha meg kívánja velem osztani a web jövőjével kapcsolatos gondolatait, esetleg közölni szeretné, hogy mely tervezetekkel, megoldásokkal és irányzatokkal kapcsolatban szeretne bővebben is olvasni az eljövendő hónapok és évek során.

Linux Journal 2005. április, 132. szám

Reuven M. Lerner hosszú ideje web/adatbázis tanácsadóként és fejlesztőként dolgozik, jelenleg a Northwestern University oktatásmódszertan kurzusának hallgatója. Weblogja az altneuland.lerner.co.il, ő maga pedig a reuven@lerner.co.il címen érhető el.

Kedvenc Bash trükkjeim

Rengeteg gépelést takaríthatunk meg néhány hasznos bash trükkel amelyek a régivágású UNIX héjprogramokból még hiányoztak.

A ***bash***, azaz ***„Bourne again shell”***, a legtöbb ***Linux*** terjesztésben alapértelmezett héjprogram. A ***bash*** héjprogram népszerűsége a ***Linux*** és ***UNIX*** felhasználók között nem a véletlen műve. Igen sok funkciója van amely a felhasználóbarát jelleget és a termelékenységet erősíti. Sajnos nem igazán tudjuk kihasználni ezeket a lehetőségeket, ha a létezésükről sem tudunk.
Amikor először kezdtem el ***Linuxot*** használni, az egyetlen ***bash*** képességet használtam. Nevezetesen, hogy a parancssorban a felfele nyíl segítségével vissza lehet lépkedni a parancstörténetben. Hamarosan további lehetőségekre is rábukkantam, másokat figyelve vagy kérdezősködve. Cikkemben az évek alatt megismert kedvenc ***bash*** trükkjeimet szeretném megosztani mindenkivel.
Ebben a cikkben nem akarjuk a ***bash*** összes képességét összefoglalni; ahhoz egy könyvre lenne szükség, és szép számmal találunk is könyveket, melyek ezzel a témával foglalkoznak. Ilyen például a ***Learning the bash Shell O'Reilly*** könyv. Ebben a cikkben inkább azokat a ***bash*** trükköket szeretném összegyűjteni amelyeket a leggyakrabban használok és amelyek nélkül elveszettnek érezném magam.

Zárójel kiegészítés

Kedvenc ***bash*** trükköm egyértelműen a ***zárójel bővítés (brace expansion)***. A zárójel bővítés vesszővel elválasztott karaktersorozatokból készít nekünk különálló paramétereket. A listát kapcsos-zárójelek, azaz { és } közé tesszük, a vesszők környékén pedig nem hagyunk szóköz karaktereket. Például:

```
$ echo {one,two,red,blue}
one two red blue
```

Az előbbi egyszerű példában bemutatott zárójel bővítés nem igazán nyújt túl sokat a felhasználónak. Tulajdonképpen a fenti példa kettővel több karakter begépelését igényli, mint ha egyszerűen csak ennyit írnánk:

```
echo one two red blue
```

ami azonos eredményt ad. Ugyanakkor a zárójel bővítés rendkívül hasznos tud lenni, ha a zárójelbe foglalt lista közvetlenül egy másik karakter sorozat előtt, mögött vagy annak közepében foglal helyet:

```
$ echo {one,two,red,blue}fish
onefish twofish redfish bluefish
$ echo fish{one,two,red,blue}
fishone fishtwo fishred fishblue
$ echo fi{one,two,red,blue}sh
fionesh fitwosh firedsh fibluesh
```

Figyeljük meg, hogy a zárójelek és a csatlakozó karaktersorozatok között nincsen szóköz karakter. Ha kitesszük a szóközt, a dolgok megbolondulnak:

```
$ echo {one, two, red, blue }fish
{one, two, red, blue }fish
$ echo “{one,two,red,blue} fish”
{one,two,red,blue} fish
```

Ugyanakkor, ha idézőjelet használunk a zárójeleken kívül vagy a listában, akkor szóközöket is beírhatunk:

```
$ echo {“one “,”two “,”red “,”blue “}fish
one fish two fish red fish blue fish
$ echo {one,two,red,blue}” fish”
one fish two fish red fish blue fish
```

A zárójeleket akár egymásba is ágyazhatjuk, de ügyelnünk kell a formára:

```
$ echo {{1,2,3},1,2,3}
1 2 3 1 2 3
$ echo {{1,2,3}1,2,3}
11 21 31 2 3
```

E példák után, biztos sokan azt mondják magukban: ***„Nahát, ezek aztán tényleg ügyes szalontrükkök, de miért jó nekem ez a zárójelbővítés?”*** A zárójel bővítés hasznos lehet, ha biztonsági másolatot akarunk készíteni egy állományról. Ez az egyik kedvenc héjtrükköm. Szinte miden nap használom, amikor egy beállításállományról változtatás előtt másolatot készítek. Például, amikor megváltoztatom az Apache beállításállományt, egy kis gépelést megtakarítva a következőket írhatom:

```
$ cp /etc/httpd/conf/httpd.conf{,.bak}
```

Figyeljük meg hogy semmilyen karakter nem álla a nyitó zárójel és a vessző között. Az ilyesmit teljesen elfogadott és hasznos amikor betűket szeretnénk egy létező fájlnévhez fűz-

ni vagy ha az egyik paraméter részhalmaza a másiknak. Aztán ha később kíváncsi vagyok milyen változtatást végeztem, könnyen megtudhatom a `diff` parancs segítségével a karaktersorozatokat fordított sorrendben megadva a zárójelben:

```
$ diff /etc/httpd/conf/httpd.conf{.bak,}
1050a1051
> # I added this comment earlier
```

Parancs helyettesítés

A másik ***bash*** trükk amit szeretek használni a parancshelyettesítés. A parancshelyettesítés használatához egy szabványos kimenetre író parancsot helyezzünk zárójelek közé, majd a nyitó zárójel elé tegyünk egy dollár jelet, `$(command)`. A parancshelyettesítés nagyon hasznos ha értéket akarunk adni egy változónak. Elég általános a héjprogramokban, ahol gyakori művelet a dátum vagy idő hozzárendelése egy változónévhez. Akkor is hasznos lehet, ha az egyik parancs kimenetét egy másik program paramétereként szeretnénk felhasználni. Például, amikor a dátumot szeretnénk egy változóhoz rendelni, a következőt írjuk:

```
$ date +%d-%b-%Y
12-Mar-2004
$ today=$(date +%d-%b-%Y)
$ echo $today
12-Mar-2004
```

Gyakran használom a parancskiegészítést amikor több ***RPM*** csomagról szeretnék egyszerre információt kapni. Például ha azon ***RPM*** csomagok fájllistájára vagyok kíváncsi amelyek nevében szerepel a `httpd` szó, egyszerűn a következő parancsot adom ki:

```
$ rpm -ql $(rpm -qa | grep httpd)
```

A belső parancs az `rpm -qa | grep httpd`, valamennyi `httpd` szót tartalmazó nevű csomagot listázza. A külső parancs az `rpm -ql`, pedig a csomagban található állományokat jeleníti meg. Most a tapasztaltabb ***Bourne*** héj használók rámutatnának, hogy a parancshelyettesítést úgy is végre lehet hajtani, ha a parancsot ***fordított aposztrófok (back-tick)*** közé helyezzük. A ***Bourne***-stílusú parancshelyettesítéssel, a fenti dátum értékadás a következő alakot ölti:

```
today2=`date +%d-%b-%Y`
$ echo $today2
12-Mar-2004
```

Az újabb ***bash***-stílusú parancshelyettesítés formátumnak azonban van néhány fontos előnye. Először is könnyebben egymásba ágyazható. Mivel a nyitó és záró jelek különbözőek, a belső jeleket nem kell backslash-el védeni. Másodszor könnyebb olvasni, különösen ha beágyazva használjuk. Még ***Linuxon*** is, ahol a ***bash*** az általános, könnyen találkozhatunk a régi, ***Bourne*** stílusú formátumot használó héjprogramokkal. Ennek oka a hordozhatóság biztosítása a különféle ***UNIX*** verziók között, ahol nem mindig van elérhető ***bash*** viszont van ***Bourne*** héj. A ***bash*** visszafelé együttműködik a ***Bourne*** héjjal, így megérti a régebbi formátumot.

Szabványos hiba átirányítása

Előfordult már, hogy egy állományt kerestünk a `find` paranccsal, ám a keresett állomány eltűnt a `permission denied` hibaüzenetek tengerében melyek pillanatok alatt elözönlötték a terminálablakunkat?
Ha mi vagyunk a rendszergazdák, átjelentkezhetünk root felhasználóként, hogy úgy adjuk ki újra a `find` parancsot. Minthogy a root bármilyen állományt olvashat, többé nem kapunk hibaüzeneteket. Sajnos nem mindenki rendelkezik root jogokkal azon a rendszeren amit használ. Emellett meglehetősen rossz gyakorlat root-ként dolgozni hacsak nemfeltétlen szükséges. De vajon mi mást tehetünk? Az egyik megoldás, ha a kimenetet egy állományba irányítjuk. A szokásos alap kimenet átirányítás nem lehet újdonság annak aki bizonyos időt töltött ***UNIX*** vagy ***Linux*** héjkörnyezetben, így a kimenet átirányítás részleteibe most nem mennék bele. A `find` parancs hasznos kimenetének elmentéséhez a kimenetet a következőképpen irányíthatjuk egy állományba:

```
$ find /  -name foo > output.txt
```

A hibaüzeneteket továbbra is látni fogjuk a képernyőn, ám a keresett állomány elérési útját már nem. Ez ugyanis az `output.txt` állományba kerül. Amikor a `find` parancs befejeződik, kiírathatjuk a `output.txt` állomány tartalmát a `cat` paranccsal és máris láthatjuk a keresett fájl(ok) elérési helyét. Ez ugyan elfogadható módszer, de van jobb megoldás is. Ne a szabványos kimenetet irányítsuk át egy állományba, hanem a hibaüzeneteket. Ezt úgy érhetjük el, hogy közvetlenül az átirányító jel elé a 2-es számot írjuk. Ha nem érdekelnek bennünket a hibaüzenetek, nyugodtan elküldhetjük őket a */dev/null*-ba:

```
$ find /  -name foo 2> /dev/null
```

Így a bosszantó `permission denied` üzenetek nélkül nézhetjük meg a `foo` állomány elérési útját, feltéve persze, hogy létezik. Szinte mindig így hívom meg a `find` parancsot. A 2-es szám a szabványos hibacsatornát jelenti. A legtöbb program erre a szabványos hiba csatornára küldi a hibaüzeneteit. A szokásos (nem-hiba) kimenet a szabványos kimenetre kerül, amit egyébként az 1-es szám jelez. Minthogy a leggyakrabban átirányított csatorna a szabványos kimenet a kimenet átirányítás alapértelmezés szerint a szabványos kimenet folyamot irányítja át. Következésképpen a következő két parancs egyenértékű:

```
$ find / -name foo > output.txt
$ find / -name foo 1> output.txt
```

Előfordul, hogy a a hibaüzeneteket és a szabványos kimenetet is el szeretnénk menteni egy állományba. A `cron` folyamatok esetében gyakran van szükség ilyesmire, amikor minden kimenetet egy naplófájlba szeretnénk menteni. Ezt is megtehetjük, ha mindkét kimeneti folyamot egyazon állományba irányítjuk:

```
$ find / -name foo > output.txt 2> output.txt
```

Működik ugyan, de megint csak van egy jobb módszer. Az és jel segítségével a szabványos hibafolyamot a szabványos kimenethez köthetjük. Miután ezt megtettük, a hibaüzenetek ugyanoda mennek ahová a szabványos kimenetet irányítottuk:

```
$ find / -name foo > output.txt 2>&1
```

Egy dologra azonban oda kell figyelni, a kötés műveleti jele mindig a kimenetet készítő parancs végére kerül. Ez akkor

lehet fontos, ha a kimentet egy másik parancsba vezetjük át. A következő sor úgy működik ahogy várjuk:

```
find -name test.sh 2>&1 | tee /tmp/output2.txt
```

Ez viszont nem:

```
find -name test.sh | tee /tmp/output2.txt 2>&1
```

és a következő sem:

```
find -name test.sh 2>&1 > /tmp/output.txt
```

A kimenet átirányítás bemutatásához a `find` parancsot használtam példaként, majd valamennyi későbbi példa is ezt a parancsot használta. A megoldás azonban természetesen nem csak a `find` paranccsal működik. Rengeteg másik parancs is készít hibaüzeneteket amelyek elfedhetik az eredményül várt egy-két soros kimenetet.
A kimenet átirányítás sem ***bash*** különlegesség. Minden ***UNIX/Linux*** héj ugyanilyen alakban támogatja a kimenet átirányítást.

Keresés a parancstörténetben

A ***bash*** héj egyik legnagyszerűbb képessége a parancstörténet, melynek segítségével könnyedén navigálhatunk oda-vissza a már kiadott parancsok között a fel és lefelé nyilak segítségével. Mindez kiváló amíg csak az utóbbi 10-20 kiadott parancs között keresgélünk, de igen fárasztó, ha a parancs 75-100 parancsnyira van a parancstörténetben. A dolgok felgyorsítása érdekében interaktívan keresgélhetünk a parancstörténetben a ***Ctrl-R*** kombináció leütésével. Amint ezt megtettük a parancssor a következőre változik:

```
(reverse-i-search)`': 
```

Gépeljük be a keresett parancs néhány betűjét és a ***bash*** megmutatja melyik az a legfrissebb parancs amely tartalmazza az eddig begépelt betűket. Amit gépelünk, a ` és ‘ jelek között jelenik meg a parancssorban. Az alábbi példában, a `htt` szót gépeltem be:

```
(reverse-i-search)`htt': rpm -ql $(rpm -qa | grep
➥ httpd)
```

Ez azt jelenti, hogy a legutoljára begépelt parancs amely tartalmazza a `htt` karaktereket a:

```
rpm -ql $(rpm -qa | grep httpd)
```

amennyiben újra végre akarom hajtani a parancsot, egyszerűen csak Entert ütök. Ha inkább szerkeszteni szeretnénk, a jobb vagy bal nyíl segítségével ezt is megtehetem. Ilyenkor a parancssorban mutatott parancs a szokásos prompt sorba kerül, ahol ugyanúgy szerkeszthetem mintha begépeltem volna. Ez igazi időmegtakarítás lehet ha sok paraméterrel ellátott parancsot szeretnék a parancstörténet mélyéről előásni.

Ciklusok használata parancssorból

Az utolsó tipp amit be szeretnék mutatni, az, hogyan tudunk ciklusokat használni a parancssorban. A parancssor nem igazán az a hely ahol egymásba ágyazott ciklusokat és elágazásokat tartalmazó összetett szkripteket szokás írni. Kis ciklusok használatával azonban egész sok időt takaríthatunk meg. Sajnos nem sok emberrel találkoztam aki ki is használná ezeket a lehetőségeket. Inkább az a jellemző, hogy az emberek minden ciklusban visszalépkednek a parancstörténetben és módosítják az eredetileg beírt parancsot. Amennyiben valaki nem nagyon ismeri a `for` vagy egyéb ciklustípusok készítésének módozatait, érdemes utánaolvasni. Sok jó héjprogramozásról szóló könyv tárgyalja a kérdést. A `for` ciklusok általános tárgyalása önmagában is megérdemelne egy cikket.
Kétféleképpen lehet interaktív szkriptet írni. Az első, általam előnyben részesített módszer, ha minden sort pontosvesszővel választunk el. A könyvtárban található valamennyi fájlról háttérmentést készítő egyszerű ciklus a következőképpen nézne ki:

```
$ for file in * ; do cp $file $file.bak; done
```

A másik lehetőség, ha pontosvessző beszúrása helyett minden sor után Enter-t ütünk. A ***bash*** a `for` kulcsszóból felismeri hogy ciklust készítünk, és a másodlagos prompt alkalmazásával kéri be a következő sorokat. A `done` kulcsszóból meg tudja állapítani, hogy befejeztük a ciklust:

```
$ for file in *
> do cp $file $file.bak
> done
```

És most valami egészen más

Amikor eredetileg rászántam magam erre a cikkre, a ***„Bolondos bash trükkök”*** címet akartam adni neki, ahol bemutatok néhány különös, ezoterikus ***bash*** parancsot amelyeket megismertem. A cikk hangvétele azóta sokat változott, de egy bolond ***bash*** trükk azért megmaradt amit be szeretnék mutatni. Körülbelül öt évvel ezelőtt egy ***Linux*** rendszer, amelyért én feleltem, kifutott a memóriából. Még az olyan egyszerű parancsok is, mint az `ls` `insufficient memory` hibával álltak le. A probléma nyilvánvaló orvoslása egy egyszerű reboot lett volna. Az egyik rendszergazda azonban szeretett volna megnézni egy állományt amely esetleg a problémával kapcsolatos nyomokat tartalmazhat, de nem emlékezett a pontos fájlnévre. A könyvtárakba be tudtunk lépni, hiszen a `cd` parancs a ***bash*** része, de a fájllistát már nem tudtuk lekérdezni, hiszen még az `ls` sem működött. A probléma megkerülésére a másik rendszergazda készített egy egyszerű ciklust amely megmutatta a könyvtárban található állományokat:

```
$ for file in *; do echo $file; done
```

Ez olyankor is működött amikor az `ls` nem, hiszen az `echo` a ***bash*** héj része, így már eleve a memóriába töltve található. Érdekes megoldása egy nem szokványos problémának. Meg tudja valaki mondani, hogyan lehet egy állomány tartalmát megjeleníteni kizárólag ***bash*** beépített parancsokat használva?

Összefoglalás

A ***bash*** héj rengeteg remek szolgáltatással könnyíti meg felhasználói életét. Remélem, legkedveltebb ***bash*** trükkjeim rövid összefoglalója mutatott néhány új lehetőséget, hogy jobban kihasználhassuk a ***bash*** igazi erejét.

Linux Journal 2005. április, 132. szám

Prentice Bisbal

Emberi beszéd mesterséges előállítása és gépi felismerése

Az emberi hangok mesterséges előállítása a világon először a magyar Kempelen Farkasnak sikerült. Az ő készüléke mechanikus úton imitálta az emberi hangképzést az 1700-as évek végén.

A számítógépek őskorában, az 1960-as években újra feléledt a gondolat: hogyan állíthatnánk elő emberi beszédet mesterségesen? A gépi hangok emberi beszédként való felismeréshez még az 1980-as években is „sok jóindulat" kellett. Annyira rossz volt a hang minősége, hogy a hallgatónak önkéntelenül az az érzése támadt, a számítógépnek sürgősen meg kellene csináltatnia a fogait. Ennek az oka többek között a kimenő csatorna kis frekvenciaszélessége volt. A gépek növekvő teljesítménye mára lehetővé tette élvezhető és érthető hang előállítását számítógép segítségével. Ugyanakkor egy hibátlanul hangsúlyozó, valódi emberi hangon beszélő, az érzelmeket és hangerőváltozásokat hűen visszaadó rendszerig, amilyeneket néha filmekben látunk, még hosszú utat kell megtennünk.

Magyar fejlesztések

Már a 80-as évek óta létezik egy magyar fejlesztésű hangszintetizátor, amit a ➲ ***http://speechlab.ttt.bme.hu/*** címen találhatunk meg. A ➲ ***www.világhallo.hu*** webhelyen kis is próbálhatjuk a rendszert, és tapasztalhatjuk, hogy a minősége kifejezetten jó. Először ***multivox*** néven egy tisztán szintetizált hangokból álló beszédszintetizátort fejlesztettek ki, majd egy emberi hangra alapuló változatot, mely ***profivox*** néven ismert. Sajnos, mindkettő kereskedelmi termék, a ***profivox*** része az angol árjegyzék szerint körülbelül 850 dollárba kerülő ***jaws*** csomagnak, melynek demó változata a ➲ ***http://www.vakalap.hu*** oldalról tölthető le. Ugyanezt a ***http://www.vilaghallo.hu*** ingyenes felolvasóprogramként terjeszti, de ez a megoldás egyrészt állandó hálózatot követel, másrészt csak néhány mások által előre kiválasztott könyvet képes felolvasni, ami sokakat biztosan nem elégít ki. 2001 óta létezik a szintén magyar fejlesztésű ***speakboard*** (➲ ***http://www.speecht.com/speakboard/intro.php)***, mely saját állítása szerint nyelvfüggetlen beszédszintézis-rendszer. Magyar és angol változata vásárolható meg, ára körülbelül húszezer forint. Az előállított beszéd meg is hallgatható a fenti oldalon, véleményem szerint szintén jól érthető.

Ingyenes illetve nyílt forráskódú beszédszintetizátorok

Mi kell tehát ma ahhoz, hogy számítógépünk beszélni, vagy akár énekelni tudjon anélkül, hogy emiatt nagy pénzt kelljen külön kiadnunk? Egy hangkártya, körülbelül 20 MB hely a merevlemezen és egy beszédszintetizáló program. A jelenleg létező beszédszintetizáló rendszerekről a ➲ ***http://www.speech.cs.cmu.edu/comp.speech/Section5/Q5.5.html*** címen találhatunk összefoglalót. A dolog gyakorlati oldalát tekintve az átlagos érdeklődő számára olyan rendszer jöhet számításba, amely ingyenes, amelyhez támogatja az új nyelvek felvételét, amelynek elfogadható a minősége, és amely lehetőleg Linuxon is és Windows alatt is működik. Ezeknek a feltételeknek jelenleg egyetlen rendszer felel meg, az ***mbrola*** beszédszintetizátor. A nevét helyesen ***„embérolának"*** kell ejteni, amúgy az angol umbrella (esernyő) szóból ered. A rendszert a ➲ ***http://tcts.fpms.ac.be/synthesis/mbrola.html*** helyen találhatjuk meg.

Az mbrola beszédszintetizátor

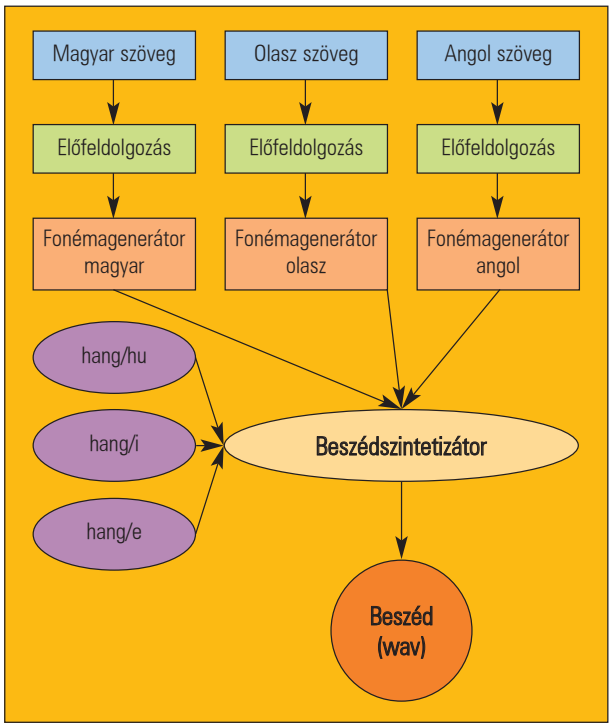
Az ***mbrola*** ugyan nem szabad forráskódú program, de a katonai és a kereskedelmi célú alkalmazásokat kivéve bárki ingyen használhatja. A programot a ***Belgiumban*** levő ***Monsi Műszaki Egyetem*** hangtechnika tanszéke írta. Körülbelül 10 éves, így ma már érettnek mondható, amit az is bizonyít, hogy 33 nyelvhez, köztük a japánhoz és koreaihoz is van már hangja. Egy adott nyelv támogatásához valakinek gépbe kell mondania az összes előforduló kettős hangzót, szaknyelven ***difonémát***. Ha megvan ez az adatbázis, a szintetizátor képes az adott nyelven beszélni.

A difonéma adatbank

Egy difonéma adatbank elkészítése nem triviális feladat, erről részletes beszámoló olvasható a ➲ ***http://tkltrans.sourceforge.net/magyar/bmrolamod.htm*** címen. Ezt természetesen csak egyetlen egyszer kell megtenni, hiszen később a hangadatbank minden felhasználó rendelkezésére áll. A magyar nyelvhez 2005. januárja óta van adatbank, és további magyar adatbankok vannak készülőben.

Szöveg-fonéma átalakító

Ha kész a difonéma-állomány, még szükséges egy nyelvfüggő fonémaelőállító program is, mely az elmondandó szöveget difonémákká alakítja. Ez a magyar nyelvhez



1. ábra

a ➲ ***http://tkltrans.sf.net/magyar/hunpho.tar.gz*** címen található. A ***„Helló világ!"*** szöveg például így néz ki fonémákká alakítva:

```
_ 50  0 160
h 70  0 170  30 200
E 90
l 65
l 65
o: 148
_ 10  0 160
_ 150  0 160
_ 10  0 160
v 70  0 210  30 200
i 90
l 65
a: 148
g 75
_ 10  0 160  30 190
_ 350  0 160
```

A fonémasorozatban egy sor értelmezése: Az első jel a kimondandó betű úgynevezett ***sampa jelölésben.*** Erről bővebben a ➲ ***http://phon.ucl.ac.uk/home/sampa*** címen olvashatunk. A második jel a kimondás hossza ms-ban, majd a hanglejtés (pitch) megadása következik, amely számpárokból áll: Az első szám a fonémán belüli hanglejtés kezdetét adja meg, a második a frekvenciát hertzben (Hz).
A példa első sora (_ 50 0 160) azt jelenti, hogy 50 ms-nyi szünetet tartunk úgy, hogy ennek 0%-ánál a frekvencia 160 Hz kell legyen. A hanglejtési pontok egy lineáris hanglejtési görbét adnak meg.

A hangerősség változtatására a ***.pho*** fájlon belül nincs lehetőség. Ugyanakkor ez is megoldott például a ***de6*** és ***de7*** hangadatbankokban. Az ötlet annyi, hogy ezek a hangadatbankok terjedelmüket megháromszorozva, minden difonémát hangos és halk módban is bemondanak.
A fonémasorozatot az ***mbrola*** szövegszintetizáló program közvetlenül ***.wav*** állománnyá képes alakítani, melyet a számítógép közvetlenül a ***play*** segédprogrammal vagy például az ***mplayerrel*** (***Linux***) vagy ***Windows*** esetén bármelyik lejátszó programmal (például ***Media Player***) le tud játszani.
Az ***mbrolával*** való hangelőállítás menetét talán legvilágosabban ez az ábra szemlélteti:
Mint látható, a hangelőkészítést és a szöveg fonémákká átalakításának nyelvspecifikus részét nem az ***mbrola*** szintetizátor végzi, hanem külső modulok. Ez a modularizált felépítés nagy szabadságot jelent, és lehetővé teszi, hogy a fejlesztő mindig csak az adott nyelvre koncentráljon, hiszen a hallható beszéddé alakítás végső soron független a célnyelv szabályaitól, sajátosságaitól.

A szövegszintetizáló rész

A felhasználó számára az ***mbrola*** program és a fonémaképzéshez szükséges eszközök telepítése talán a legnehezebb a megoldandó feladatok közül. A programot és a hangadatbázist a ➲ ***http://tcts.fpms.ac.be/synthesis/mbrola/mbrcopybin.html*** kell letölteni.
Egy linuxos gépen az ***mbr301.zip*** tartalmát kicsomagolás után a ***/usr/local/mbrola*** könyvtárba kell írni, és ez alatt létre kell hozni egy ***hu1*** alkönyvtárat. Ide kerül a hu1 adatbank, vagyis ide kell kitömöríteni a ***hu1.zip*** nevű fájlt.
Az egész folyamat a ***Linux*** alatt így néz ki (***/en/konyvtaram*** az a hely, ahová a letöltött csomagokat helyezzük).

```
bash# cd /usr/local
bash# mkdir mbrola
bash# cd mbrola
bash# unzip /en/konyvtaram/mbr301h.zip
bash# unzip /en/konyvtaram/hu1.zip
```

A ***/usr/local/bin*** könyvtárban hozzunk létre egy a végrehajtható ***mbrola*** állományra mutató szimbolikus linket:

```
bash# cd /usr/local/bin
bash# ln -s /usr/local/mbrola/mbrola-linux-i386
mbrola
```

A szöveg fonémákká alakítása

Ezek után a szöveg-fonéma átalakítót (***http://tkltrans.sf.net/magyar/hunpho.tar.gz***) kell beüzemelnünk. Ennek a részegységnek szüksége van a Perl nyelvre, valamint az awk-ra.
A feldolgozandó szövegnek egyszerű szövegként kell rendelkezésre állnia, vagyis a .doc, .pdf, és egyéb ehhez hasonló fájlokat előbb tiszta szöveggé kell átalakítanunk.
A szöveg két szűrőn halad át, mielőtt hanggá (vagyis fonémák sorozatává) válik.
A ***szam.awk*** kiszűri a zárójeleket és idézőjeleket, az önálló betűket, egyes rövidítéseket kiejthetővé tesz, problémás szókapcsolatokat szétválaszt a számjegyeket pedig szavakká alakítja.

Az *xttp.pl* aztán az így megszűrt szöveget fonémákká alakítja. Az keletkezett fonéma állományt kell átadni az *mbrola* programnak úgy, hogy az a hu1 adatbázist használja.

```
awk -f szam.awk <$1.txt >$11.txt
perl xttp.pl $2 < $11.txt >$1.pho
mbrola /usr/local/mbrola/hu1/hu1 $1.pho $1.wav
play $1.wav
rm -f $11.txt
```

`xttp.pl` paramétere, melyet nem kötelező megadni, lehet `m`, `f1`, `f2` vagy `f3`. A paraméter hiánya a legmélyebb női hangot jelenti, azaz megfelel az `f1` paraméternek, `f2` és `f3` egyre magasabb hangokat jelent, `m` pedig a férfihang.
Ha egy szövegállományt sikeresen átalakítottunk `szoveg.pho` formátumba, akkor az `mbrola` azt *wav* formátummá alakítja, amit aztán egy tetszőleges hanglejátszó program le tud játszani. A legegyszerűbben a

```
play szöveg.wav
```

paranccsal tehetjük hallhatóvá művünket. (***Alsa*** meghajtó esetén az `aplay` parancsot kell használni).
Íme két példa a `do.sh` (vagy `do2.sh`) használatára:

```
sh do.sh szoveg f1
sh do.sh szoveg
```

Eljutottunk tehát odáig, hogy bármely szöveget fel tudunk olvastatni a géppel. Most koncentráljunk a finomabb igényekre. A ***mbrola*** fel tud olvasni hosszabb szövegeket, tud énekelni, a szöveg hanglejtését pedig grafikusan is megjeleníthetjük vele. Ez utóbbi szolgáltatás a fonémaátalakító program javításánál jól használható.

Hosszabb szöveg felolvasása

Mivel hosszabb szövegek feldolgozása nagyon nagy *wav* állományt eredményezne, célszerű ezeket kisebb darabokba szabdalni és úgy fölolvastatni. Erre való a ***mesel.pl*** program, amely a ***segedeszkozok*** könyvtárból futtatható. Az olvasandó állomány nevét a program paramétereként adhatjuk meg (a ***.txt*** végződés nélkül), az eredménynek pedig hozzuk létre a ***test*** nevű alkönyvtárat.
A program kizárólag olvassa a neki átadott állományt, tehát annak tartalmát semmilyen módon nem változtatja meg. Az éppen fölolvasott részt a ***kwrite*** segítségével a képernyőn is mutatja. A megjelenítéshez használt program a `$szerkeszto` változóval ízlés szerint változtatható. A `$sor` belső változó az egyszerre fölolvasott sorok száma, szintén ízlés szerint beállítható. A ***mesel.pl*** program hívása, ha a fölolvasandó szöveg a ***szoveg/olvasnivalo.txt*** állomány:

```
perl mesel.pl szoveg/olvasnivalo
```

A mellékelt `mesel.sh` állományt használva:

```
sh mesel.sh szoveg/olvasnivalo
```

A ***mesel.pl*** a ***mesel_sorok.txt*** fájlban feljegyzi, hogy éppen hol tart az olvasásban. Ha indításakor a második paraméter a `last` szó, akkor a ***mesel_sorok.txt*** állományban levő szám által kijelölt sornál kezdi el a szöveg olvasását. Ha a harmadik paraméter szám, akkor az annyiadik sorban kezdi meg a fölolvasást. Íme néhány példa:

```
perl mesel.pl szoveg/olvasnivalo last
```

Ez esetben ***mesel_sorok.txt*** tartalma szabja meg, hogy hol kezdi el az olvasást.

```
perl mesel.pl szoveg/olvasnivalo 256
```

Ekkor a 256-odik sornál kezdi el a fölolvasást.

1. táblázat *A zenei alaphangok frekvenciái*

c3	d3	e3	f3	g3	a3	h3	c3
262	294.5	327.5	349	393	440	491	524

2. táblázat *„Boci, boci, tarka...”*

{Singing:	8,c2,	8,e2,	8, c2,	8, e2,	4,g2	4,g2	}.
	bo	ci	bo	ci	tar	ka	

Éneklés

Mivel a számítógép szemszögéből az éneklés sem más, mint a hangképzés egy különleges esete, a mbrola énekelni is tud. A ***hunpo.tar.gz***-ben mellékelt ***boci.txt*** állomány például a boci-boci tarkát tartalmazza. A további mellékelt dalok a damdam (***damdam.txt***), a király és a bolond (***bolond.txt***), szeretnék szántani (***szantani.txt***). Ha valakinek netán komponálni támadt volna kedve, annak ajánlom figyelmébe az *1. táblázatot*, amely az alaphangok frekvenciáit tartalmazza (hertzben). Gépünket a következő egyszerű paranccsal fakaszthatjuk dalra:

```
sh do.sh szoveg/boci.txt
```

A rendszer hangzókészlete négy oktávnyi távolságot fog át, ami egy géptől egészen szép teljesítmény. A hangok `cn`, `ciszn`, `dn`, `diszn`, `en`, `fn`, `fiszn`, `gn`, `giszn`, `an`, `aiszn`, `hn` alakban adhatók meg. Az `n` egy szám, ami a hangmagasságot jelzi. 1 a legalacsonyabb, 4 a legmagasabb oktáv. A normál zenei á hang (440 Hz) jele tehát `a3`. Minden hang előtt meg kell adni a hosszúságát is. 1 a leghosszabb, 16 a legrövidebb hang. Ezzel a jelölésrendszerrel a „Boci boci tarka” a következőképpen fest:

```
{Singing: 8,c2,8,e2,8,c2,8,e2,4,g2,4,g2}.
```

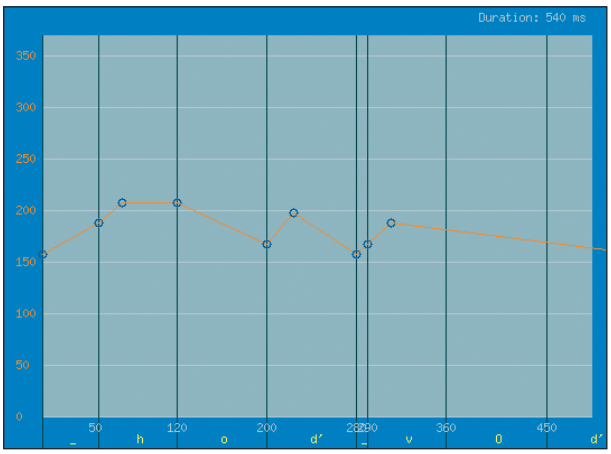
Az egyes szakaszok kiénekelt formáját a *2. táblázat* mutatja. Látható, hogy az első helyen a `{Singing:` betűsorozatnak kell állnia, majd ezt követik a hosszúság/hang párok, vesszővel elválasztva. A sorozatot egy „`}.`” jelpár zárja. (A pont fontos a sor végén!) A `{Singing:` kezdetű sort követő sorban van az énekelendő szöveg. Ennek ugyanannyi szótagból kell állnia, mint ahány hangot megadtunk, és ezt is pont zárja a végén. Szintén lényeges, hogy a `{Singing:`-et megelőző szövegrészt ponttal kell lezárni, különben ezt lenyeli a program.

Egy mondat hanglejtésének grafikus megjelenítése

Néha nagy segítséget jelenthetnek a a show.pl illetve a graph5.pl nevű Perl programok, amelyeket szintén

3. táblázat *Az mbrola program kapcsolói:*

Kapcsoló	Leírás	Példa
h	segítség a kapcsolókhoz	`mbrola -h`
i	fonéma-adatbank információ	`mbrola -i /usr/local/mbrola/hu1/hu1`
e	nem létező difonéma ne okozzon megállást	`mbrola -e /usr/local/mbrola/hu1/hu1 test.pho test.wav`
v	hangerő 1: alapállás, <1: halkabb, >1: hangosabb	`mbrola -v 1.1 /usr/local/mbrola/hu1/hu1 test.pho test.wav`
f	hangmagasság 1: alapállás, <1: magasabb, >1: mélyebb	`mbrola -f 0.6 /usr/local/mbrola/hu1/hu1 test.pho test.wav`
t	beszédsebesség 1: alapállás, <1: gyorsabb, >1: lassabb	`mbrola -t 1.2 /usr/local/mbrola/hu1/hu1 test.pho test.wav`



2. ábra

a hunpho.tar.gz csomagban találunk. Ezek a megadott mondat hanglejtését egy képfájlba írják ki, amit aztán bármelyik böngészővel meg lehet tekinteni:

```
show.pl:teszt.png graph5.pl:file3.png
```

Ezek az eszközök a jobb fonetika kidolgozásában segítenek, vagyis az xttp.pl program javításában. A háló kedvelői számára a szolgáltatás online is elérhető a következő helyen: ***http://pascal.kgw.tu-berlin.de/expressive-speech/online/synthesis/hungarian/en/ia-en.php***. (Köszönet ***Astrid Paeschkének*** a ***Berlini Egyetemről*** a magyar adatbank integrálásáért!).

A fonéma-átalakító és a nyelvi adatbázis továbbfejlesztése

Első lépésben a hangsúlyozást kell tökéletesíteni, hogy a kérdő, felszólító, óhajtó, stb... mondatok intonációja helyes legyen. A hely és időbeli viszonyt kifejező szavakat nem hangsúlyozzuk, (például „asztal alatt"). így a fonéma-átalakító már ma is helyesen kezeli ezt.
A német fonetika-átalakító (txt2pho) képes egyes szavak különféle hangsúlyozására. A ***„felástam a kertben a zöldségágyást"*** mondat esetében például a hangsúlyozandó szót ki lehet emelni a szórenddel, de a hangsúlyozással is.
A következő lépés a különféle érzelmeket (kétely, félelem, letörtség, jókedv, buzdítás, stb) kifejező szöveg helyes hangsúlyozása lenne.

A szegény ember beszédszintetizátora

A mbrola mögött megbúvó zseniális ötlet tulajdonképpen a difonémák használata. Ha van valaki, aki összeállítja egy adott nyelv difonémakészletét, azt kellő monotonitással a gépbe mondja (azaz ***wav*** állományokat hoz létre) és ezekből adatbankot készít, akkor ezzel a rendszer tetszőleges szöveg felolvasására képessé válik. Ráadásul ehhez elegendő például egy egyszerű Perl program, mely a felolvasandó szöveg difonémáit leképezi az adatbank difonémáira.
Tegyük fel például, hogy az adatbank tartalmazza a következő difonémákat:

_b, _a, ab, ba, aa, bb, _a, _b,

ahol a _ jel szünetet jelent.
Ezzel az adatbankkal a gép ki tudja mondani a „baba", „abba", „a", „bab", „ba" szakavak és szóelemeket, illetve minden más, csak „a"-t és „b"-t tartalmazó szót. A megfelelő Perl program mondjuk a „baba" szóhoz kikeresi a

_b ba ab ba a_

difonémákhoz tartózó wav jeleket, majd ezeket egyetlen wav állománnyá alakítja. És ezzel meg is oldotta a baba szó kimondását.
Problémát jelent, hogy az egyes difonémák összekötésekor a találkozási pontban egyes esetekben keletkezhet egy kattanás, ha a hangerő vagy a hangmagasság nem ugyanaz. Nyilván célszerű úgy kialakítani a szoftvert, hogy állítható legyen a hangmagasság (pitch), a beszéd sebessége, valamint a hangerő. Ezeknek a paramétereknek a helyes megválasztásával kelthetjük a beszéd valódiságának érzetét. Mindezt az ***mbrola*** program már most is lehetővé teszi.

Más, nyílt forráskódú beszédszintetizátorok

Létezik nyílt forráskódú beszédszintetizátor is. Ilyen például a http://www.cstr.ed.ac.uk/projects/festival/ helyen elérhető ***Festival***. Ehhez jelenleg csak angol, welsh és spanyol difonémakészlet létezik, valamint egy olyan kapcsolat, melynek segítségével az ***mbrola*** hangadatbankjait és szintetizáló képességét is felhasználja a ***Festival***.

Ehhez nagyon hasonló a CMU egyetem ***Festvox*** nevű terméke (***http://festvox.org***), melyhez szintén csak a fenti három hangadatbank létezik, de a dokumentációja leírja, hogyan lehet japán adatbankot készíteni hozzá (***http://festvox.org/bsv/bsv-jpdiphone-ch.html***).
A Festival szintetizátor Mandrake Linuxon 10.0 alatt nekem rögtön működött, de a hang kétszeres sebességű, vagyis csipogó volt. Elolvasva aztán a gyakran ismételt kérdések oldalt rögtön a második kérdésére adott válaszban megtaláltam ennek az okát (***http://www.cstr.ed.ac.uk/cgi-bin/cstr/lists.cgi?config=festival_faq&entry=arunning_festival/speed.html).*** A trükk annyi, hogy hogy létre kell hozni a lib könyvtárban egy siteinit.scm állományt a következő tartalommal:

```
(Parameter.set 'Audio_Method 'Audio_Command)
(Parameter.set 'Audio_Command "sox -t raw -sw -r
$SR $FILE -c2 -t ossdsp /dev/dsp")
```

Alsa hangmeghajtás esetén a második sor:

```
(Parameter.set 'Audio_Command "aplay -fS16_LE -r
$SR $FILE ")
```

A ***Festival*** beszédszintetizáló és a ***Szfinx*** beszédfelismerő programcsomagok azoknak az elszántabb embereknek jók, akik szeretnének a forráskódhoz hozzájutni és akiket érdekelnek azok a programtechnikai eszközök, melyeket ezek a programcsomagok használnak, illetve akik kíváncsiak mondjuk arra, hogy más hangadatbankok milyen egyszerű fonémákból, trifonémákból vagy fölvett szövegekből állnak, és ezekkel milyen hangminőséget eredményeznek.
Mivel ezek a programcsomagok temérdek lehetőséget kínálnak, megértésük lényegesen időigényesebb, mint a most bemutatott ***mbrola*** használata.
sAz beszédszintetizáló alkalmazások száma manapság egyre nagyobb, képességeik pedig egyre jobbak. Könyvek, cikkek, emailek fölolvasása mellett lehetővé teszik, hogy a számítógépet vakok is kezelhessék. Ez utóbbira több linuxos projekt is létezik:

http://developer.gnome.org/projects/gap/AT/Gnopernicus/
http://www.kde-apps.org/content/show.php?content=18806
http://emacspeak.sourceforge.net/

Ezek közül talán a ***Gnopernicus*** a legelőrehaladottabb, de még ez sem kész.

Beszédfelismerés

A gépi beszéddel rokon, de lényegesen bonyolultabb téma a gépi beszédfelismerés. Műszaki szempontból, illetve nehézségét tekintve a beszédfelismerés a kézírásos szöveg gépi felismerésével egyenértékű feladat. (A kézírásos szövegnél az írások különbözősége mellett az igazi nehézséget az jelenti, hogy az egyes szavak közti üres jel nincs mindenhol bejelölve.)
A ***Carnegie-Mellon Egyetem*** több projektje is foglalkozik a beszédtechnikával. (***http://www.speech.cs.cmu.edu/***, ***http://cmusphinx.sourceforge.net/html/cmusphinx.php***).
A beszédfelismerés módszertana mára már annyira érett, hogy nemsokára bizton számíthatunk a gyakorlatban is használható beszédfelismerők megjelenésére. Ezek a gépbe mondott szöveget írott alakra hozzák, ami szintén sok esetben hasznos lehet. Egy értekezleten például a gép vezetheti a jegyzőkönyvet, de diktálhatunk a gépi titkárnőnek leveleket és más szövegeket is.
Összehasonlításként talán nem árt áttekinteni a kereskedelmi termékek piacát sem.
Magyarul összesen egyetlen olyan program létezik, amely beszédfelismerésre és gépi szövegbevitelre képes. Ez a ***Philips SpeechMagic*** nevű programja, mely azonban ármegjelölés nélkül szerepel a magyar weben. Az idegen nyelvű változat 850 dollárba kerül.
Angol és német nyelven az ***IBM*** illetve a ***Scansoft Dragon*** nevű programja versenyez egymással, az ***IBM*** terméke a ***ViaVoice*** körülbelül 160 dollárba kerül, a ***Dragon*** pedig kiépítéstől függően 90 és 850 dollár közötti áron kapható.
A fölismerési pontosság cégek szerint ma még csak 95%-os, ami nyilvánvalóan komoly utólagos feldolgozást tesz szükségessé. A pontosság a program betanítása után általában javul, de ezt minden felhasználónak külön-külön kell elvégeznie.

Nyelvi megfontolások, algoritmusok

A magyar nyelv esetén a ragozott alakok fölismerése nyilvánvalóan nem triviális, de mint a magyar helyesírásellenőrzők minősége mutatja, egyáltalán nem lehetetlen feladat. Szerencsére az egyértelmű kiejtés a szükséges szótár méretét nagyon lecsökkenti (különleges kiejtést kívánó nevek mint Széchenyi, Desewffy, vagy idegen szavak mint pszichológia tartoznak a magyar szótárba.). Az angol nyelv esetén a ragozott szavak problematikája ugyan minimális, a szótárnak viszont a teljes angol szókészletet fel kell ölelnie a kiejtés szóspecifikus volta miatt.
Az összehasonlításra az elterjedt módszer a ***HMM (Hidden Markov Model)***, mely alapjában statisztikai módszer. Előnye a hullámösszehasonlítási módszerekkel szemben a gyorsaság. A szavak határának megállapítására a szintén statisztikai ***Viterbi*** algoritmus terjedt el, de elképzelhetőek természetesen más módszerek is. A szintetizálásra az egyszerűbb ***LPC (Linear predicting code)*** vagy a komplikáltabb ***PSOLA (Pitch-Synchronous-OveraLap-Add)*** módszerek a legelterjedtebbek. A szintetizálási módszerek iránt érdeklődők jó bevezetőt találhatnak a ***http://tcts.fpms.ac.be/synthesis/introtts.html*** címen.

Korlátozott felismerés illetve szövegszintézis

Olyan területeken, ahol kevés szó felismerése elegendő, már próbaképpen bevezették a gépi szövegfelismerést. Ilyen például a vonat- vagy repülőjegy telefonos megrendelése, vagy a menetrendi tájékoztatók. Egyes bankok a korlátozott szövegfelolvasást használják például a folyószámla állásának lekérdezésére. Ilyenkor a felhasználó a telefon billentyűin keresztül adja be a folyószámlaszámot, a gép pedig a fölolvassa a folyószámla állását.

Klein Eleonóra

Ügyfeleink kezelése nyílt forráskódú CRM szoftverrel

Használjuk ki a nyílt forráskódú, platform független CRM szoftverek előnyeit az üzleti életben is.

Ahogyan a *Linux* egyre nagyobb teret hódít nem csak a kiszolgálók, hanem a munkaállomások területén is, így jogosan merül fel az igény olyan szoftver termékek iránt, melyekkel kedvenc operációs rendszerünket nem csak szövegszerkesztésre, levelezésre, böngészésre tudjuk használni, hanem szó szerint „munkára" is foghatjuk. Sajnálatos módon jelenleg a piacon lévő üzleti célú szoftverek nagy része csak egy operációs rendszert támogat, így amennyiben egy másik operációs rendszert szeretnénk használni, akkor szinte biztosak lehetünk abban, hogy le kell cserélni például a számlázó, vagy a raktárkészlet nyilvántartó programunkat is. Pedig ennek nem feltétlenül kell így lennie.
Ebben a cikkben most egy olyan *CRM* szoftverrel ismerkedünk meg, mely nem csak, hogy nyílt forrású, hanem platform független is. Jelenleg szerintem már csak olyan vállalati szoftverekben van jövő, melyek több platform alatt is képesek működni, vagy könnyen portolhatóak egyik platformról a másikra. Ezt úgy érzem, hogy mindenképpen érdemes szem előtt tartani, amikor olyan szoftvereket vezetünk be, melyeket hosszú távon szeretnénk használni. Mivel a *CRM* szoftverek hazai piaca csak most kezd kibontakozni, ezért még nem késtünk el azzal, hogy platform független, nyílt forráskódú szoftvert válasszunk erre a célra.

Mi is az a CRM?

A *CRM*, azaz *Customer Relationship Management* egy összetett fogalom, magyarul talán a legmegfelelőbb fordítása az *Ügyfélkapcsolat Menedzsment* lenne. Ellenben, hogy ez a fogalom mit is jelent pontosan, arra úgy érzem, hogy a következő két idézet teljes mértékben rávilágít.
„A CRM olyan koncepció, amely a legértékesebb ügyfelek azonosítására, megszerzésére és megtartására, valamint az ilyen ügyfélkör kibővítésére szolgál, lehetővé téve a folyamatos növekedést és az üzleti értékek fejlesztését." (Atos Origin)

„A CRM olyan az egész vállalatot átfogó stratégia, melynek célja a jövedelmezőség, a bevétel és a vevői elégedettség növelése. A CRM ezt a célt az ügyfélközpontú magatartás támogatásával, minden vevőcsatorna összekötésével éri el, úgy, hogy a vállalatot az ügyfelek igényei szerint szervezi át." (GartnerGroup)

A fenti idézetek inkább hasonlítanak egy-egy reklám szövegére, sem mint pontos definícióra, ez nem véletlen, hiszen a CRM rendszer egyben egy marketing eszköz is, amely ezen felül javítja az értékesítés és az ügyfélszolgálat hatékonyságát.

Miért lehet szükségünk a CRM rendszerekre?

Most egy pillanatra vonatkoztassunk el a fenti definícióktól és gondoljuk végig, hogy ha van egy vállalkozásunk, akkor annak nyilván van valamiféle terméke, ez a termék lehet fizikai termék, vagy szolgáltatás is. Tehát vannak ügyfeleink, akik megveszik a mi termékünket, használják őket, közben esetleg kérdéseik, problémáik merülnek fel. Ha elégedett ügyfeleket szeretnénk, akkor az ügyfelek problémáival, kérdéseivel foglalkoznunk kell, célszerű nyilvántartani azt, hogy az egyes problémák megoldása folyamatban van-e, illetve, hogy azokat megoldottuk-e már. Kellemetlen tud lenni egy elfelejtett, vagy időben nem megoldott probléma. Vegyük észre azt is, hogy a problémát nem feltétlenül az a személy oldja meg, akivel az ügyfél egyeztetett, sőt lehet, hogy több ember kooperatív munkája szükséges egy probléma megoldásához. Tekintsünk most el a problémáktól, nyilván termékeink iránt mindig lesznek érdeklődők, célszerű az érdeklődő ügyfelekkel történő megbeszéléseket, tárgyalásokat is nyilvántartani, hiszen egyes érdeklődő személyek, cégek később komoly üzleti partnereink lehetnek. Természetes követelmény lehet az is, hogy szeretnénk látni beosztottjaink, kollégáink időbeosztását is, így sokkal hatékonyabban tudunk előre tervezni, láthatjuk, például a következő hét fő feladatait, így egy nagyon zsúfolt hétre például inkább nem vállalunk el újabb munkákat, tárgyalásokat. Megfigyelhetjük kollégáink hatékonyságát is, illetve lehetőségünk van előre látni lehetséges bevételeinket is.
A fent említett esetek természetesen csak példák, a *CRM* rendszerek ennél sokkal komplexebbek, és egy-egy *CRM* szoftver lehetőségeinek bemutatására sem ennek a cikknek, sem pedig ennek az újságnak a terjedelme nem volna elegendő. Ugyanakkor megismerkedünk azokkal a minimum követelményekkel, mellyel minden *CRM* szoftvernek rendelkeznie kell, és végül röviden bemutatom a *SugarCRM 2.5 OpenSource* változatát.

Ismerkedés a CRM rendszerekkel

Természetesen a *CRM* sem csodaszer, hiába oldjuk meg a problémákat gyorsan és hatékonyan, ha állandóan csak problémák vannak a termékünkkel, nem garantálja azt sem, hogy bevezetése után duplájára fog emelkedni a forgalom, és egyszer s mindenkorra piacvezetővé lépünk elő. De segítségével lehetőségünk van bizonyos folyamatok optimalizálására, szorosabb kapcsolatot tudunk kialakítani a partnereinkkel, meglévő partnereinket nagyobb valószínűséggel tudjuk megtartani, és termékeinket nagyobb eséllyel tudjuk értékesíteni. Azért is választottam a *SugarCRM* szoftvert, mert a jelenleg elérhető nyílt forrású *CRM* szoftverek közül csak ennek volt megfelelő magyar fordítása. Ugyanakkor szeretném megemlíteni, hogy nagyon sok hasonló szintén szabad forráskódú *CRM* szoftvert lehet találni, de a cikk írása közben, és előtte is nekem úgy tűnt, hogy jelenleg ez a termék fejlődik a legdinamikusabb módon, tehát nem azért esett rá a választás, mert ez lenne az összes közül a legjobb, látni fogjuk, vannak jelenleg hiányosságai is. Tekintsük meg tehát, hogy mik a *SugarCRM 2.5* alapfunkciói:

Ügyfélkapcsolat menedzsment:

- ***Felhasználók*** létrehozása, és kezelése. ***Ügyfelek*** hozzárendelése a felhasználókhoz.
- ***Aktivitások*** listája (találkozók, hívások, feladatok, jegyzetek opcionális csatolt fájl kezeléssel, és e-mailek) követése kapcsolatok, ügyfelek, érdeklődök, és üzletek esetén.
- A felhasználókhoz ***Feladatok*** hozzárendelése, és lehetőség automatikus e-mail értesítésre az új feladatok esetén.

Értékesítési támogatás:

- Összefoglaló nézet a következő ***aktivitásokról***, a legjobb ***üzletekről***, a nyitott ***feladatokról***, az érdeklődő ***ügyfelekről***, az értékesítési folyamatok vizsgálata, többféle ***naptár*** nézet, új kapcsolatok gyors felvitele.
- ***Érdeklődők*** létrehozása, és követése, az ***érdeklődők*** üzletekké való konverziója.
- Grafikus ***kimutatások*** készítése, az üzleti csatornák mutatása, ***érdeklődők*** esetén az érdeklődés forrás nyilvántartása, és az érdeklődés eredményének nyilvántartása.

Ügyfélszolgálati támogatás:

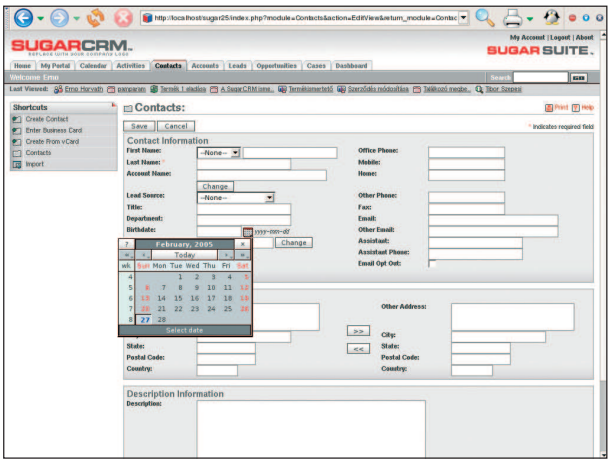
- Különbözőféle ***ügyek*** nyilvántartása, mely segíti a felhasználóknak az ***ügyfelek*** problémáinak, és döntéseinek a követését mivel minden ügyet mint egy folyamatot tárol a rendszer.
- Mindegyik ***ügyet*** lehetőségünk van egy ***felhasználóhoz***, ***ügyfelekhez***, feljegyzésekhez csatolni, és akár több ***hívás*** és személyes találkozó is kapcsolódhat az ***ügyhöz***.

Közös, megosztott naptár:

- A ***naptár*** megtekinthető napi, heti, havi vagy éves bontásban mindegyik ***aktivitás*** a hozzárendelt feladat listával
- Bepillantási lehetőség a többi ***felhasználó*** naptárába, hogy elkerüljük a konfliktusokat.

SugarCRM, te édes

Most ismerkedjünk meg a *SugarCRM* rendszer főmenüjével, illetve, hogy az egyes menüpontnál milyen funkciót érhetünk el. Itt a jelenlegi magyar fordításból indulok ki, még akkor is, ha az néhány esetben nem pontosan fedi az angol fogalmat.



1. ábra A kezdőlapon a legfontosabb adatainkról látunk összefoglaló képet

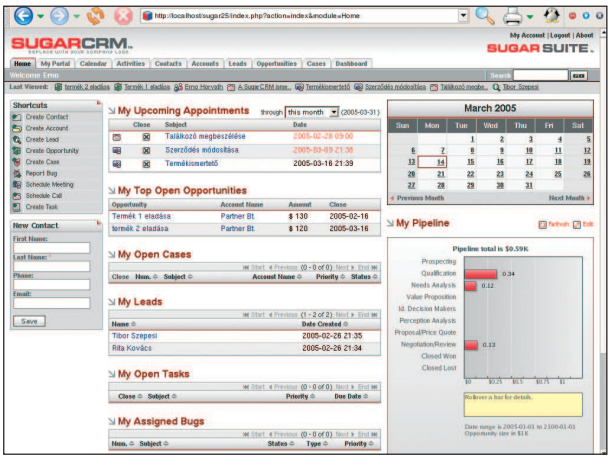
A ***Honlap (Home)*** menüpont alatt egy összefoglaló oldalt találunk az aktivitásokról, üzletekről, naptárról, és a teendőinkről, illetve hivatkozásokat tartalmaz új ügyek, ügyfelek, feladatok felvitelére. A ***Honlap menüpont*** alatt egy összefoglaló nézetet látunk arról, hogy milyen eseményekre kell koncentrálni az adott napon, a következő napon, az adott héten, vagy éppen a következő héten. Az *1. ábrán* a ***Home menüpont*** alatt található oldalt láthatjuk.

A ***Saját Portál (My Portal)*** menüpont alatt lehetőségünk van megadni a kedvenc weboldalainkat, így azokat könnyen és gyorsan elérjük a *SugarCRM*-ből, bár a weboldal a *SugarCRM*-en belül jelenik meg, ezért kisebb a látható felülete.

A ***Naptár (Calendar)*** menüpontban az előre tervezett aktivitásokat láthatjuk többféle nézetben, illetve lehetőségünk van a találkozók, feladatok, és hívások megtekintésére is. A ***Naptár*** lehetőséget biztosít arra, hogy a felhasználók megosszák egymással az időbeosztásukat.

Az ***Aktivitások (Activities)*** menüpont alatt lehetőségünk van felvinni új aktivitásokat, módosítani egy már meglévő aktivitást, vagy éppen keresni az aktivitások között. Az ***Aktivitásokat*** kezelhetjük a ***Felhasználók***, ***Kapcsolatok***, ***Érdeklődők***, ***Üzletek***, vagy az ***Ügyek*** menüpontok alatt is. Ezáltal a *SugarCRM* lehetőséget biztosít arra, hogy hívásokat, találkozókat, feladatokat, feljegyzéseket tudjunk követni annak érdekében, hogy egy komplex munkát elvégezzünk. A ***Feladatok*** lehetőséget biztosítanak olyan eseményeket nyilvántartására, melyeknek egy bizonyos időpontig be kell fejeződnie. A ***Jegyzetek*** segítségével feljegyzéseket tudunk készíteni, illetve a jegyzetek mellé fájlokat is van lehetőségünk csatolni. A ***Hívások*** segítségével a telefonhívásainak tudjuk nyilvántartani. A ***Tárgyalások*** hasonlóak a ***Hívásokhoz***, de nyilván tudjuk tartani a tárgyalás helyszínét is.

A ***Kapcsolatok (Contacts)*** menüpont alatt lapozható listát látunk a kapcsolatainkról, illetve kereshetünk is közöttük. Egy kapcsolatra kattintva rögtön megtekinthetjük annak főbb tulajdonságait, egy kapcsolat rekordból linkek segítségével eljuthatunk a kapcsolódó Ügyfelek, ***Aktivitások***, ***Érdeklődők*** vagy ***Ügyek*** listájához. A ***Kapcsolatok*** olyan emberek, melyekkel a mi vállalatunk üzletet köt. Kapcsolatainkról sok hasznos információt tudunk eltárolni, például



2. ábra Új kapcsolat létrehozása, a felugró ablakban könnyen átlátható a naptárral

a személy beosztását, címét, vagy e-mail címét, ezeket a személyeket általában egy Ügyfélhez kapcsolódva tároljuk el, de nem szükséges minden Kapcsolatot ügyfelekhez kötni. Az 2. *ábrán* láthatunk egy üres kapcsolat oldalt, ahol lehetőségünk van új kapcsolatot rögzíteni.

Az ***Ügyfelek (Accounts)*** menüpont alatt lapozható listát látunk az ügyfeleinkről, illetve kereshetünk is közöttük. Egy ügyfélre kattintva rögtön megtekinthetjük annak főbb tulajdonságait, egy ügyfél rekordból linkek segítségével eljuthatunk a kapcsolódó ***Kapcsolatok***, ***Aktivitások***, ***Érdeklődők*** vagy ***Ügyek*** listájához. Az ***Ügyfelek*** olyan vállalatok, melyekkel a mi vállalatunk üzletet köt. Ügyfeleinkről sok hasznos információt tudunk eltárolni, például a címét, alkalmazottaik számát, és tevékenységi körüket is. Lehetőségünk van egy ügyfél alá több ügyfelet felvenni, ezáltal láthatjuk ügyfeleink közötti kapcsolatokat is.

Az ***Érdeklődők (Leads)*** menüpont alatt lapozható listát láthatunk az érdeklődőkről, illetve kereshetünk is közöttük. Egy érdeklődőt kiválasztva részletesen is megtekinthetjük a hozzá kapcsolódó aktivitásokat, illetve az egymás után következő aktivitások listáját. Az ***Érdeklődők*** olyan emberek, vagy vállalatok, akikkel a saját vállalatunk előre láthatólag üzletet fog kötni. Asz ***Érdeklődők*** tipikusan a marketing osztály által az értékesítési osztálynak átadott lehetséges vásárlók. Az ***Érdeklődők*** modul úgy lett kialakítva, hogy követhetővé váljanak a lehetséges vásárlóval történő első lépések. Az Érdeklődők közé felvehetünk ügyfeleket automatikusan is, például olyan személyeket, akik regisztrálják magukat a honlapunk.

Az ***Üzlet (Opportunities)*** menüpont alatt egy lapozható listát kapunk az üzleteinkről, illetve kereshetünk is listában. Egy üzletre rákattintva láthatjuk az üzlethez kapcsolódó nyitott aktivitások listáját, illetve az előzményeket is. Az ***Üzlet*** menüpont alatt lehetőségünk nyílik követni egy árú, vagy szolgáltatás értékesítési lépéseit egy potenciális vevőnek. Amikor egy érdeklődővel kezdődnek meg az üzleti tárgyalások, akkor már célszerű ügyfélként, vagy kapcsolatként tovább kezelni az eseményeket.

Az ***Ügyek*** menüpont alatt egy lapozható listát kapunk az ügyekről, vagy éppen kereshetünk is közöttük. Egy ügyre kattintva láthatjuk az adott ügy részleteit, ebben a nézetben hivatkozásokat találunk az összes kapcsolódó ***Aktivitásra***, és ***Kapcsolatra***. Az ***Ügyek*** az értékesítés és az ügyfélszolgálat közötti kapcsolatot teszik könnyebbé. Az ***Ügyek*** megkönnyítik az ügyfélszolgálat munkáját, a kérdések, és a problémák kezelhetővé, követhetővé válnak, mindegyik ügyhöz lehetőség van prioritást, és státuszt rendelni. Az ügyhöz tartozó lezárt, és nyitott Aktivitások is megjeleníthetőek.

SugarCRM: érvek és ellenérvek

Minden egyes komplex szoftvertermék esetében sokféle érvet és ellenérvet tudunk felsorakoztatni a használat mellett, illetve ellen, most megpróbálom a szoftvert, nem mint ***CRM*** termék, hanem mint felhasználói szoftver tekinteni. Újfent szeretném kiemelni azt a tényt, hogy a ***SugarCRM*** platformfüggetlen, mind ügyfél, mind pedig kiszolgáló oldalon, sőt kliensként bármilyen böngésző szoftver alkalmas arra, hogy elérjük vele az adatainkat. Mivel a ***SugarCRM*** már eleve internetböngészőn keresztül érhető el, ezért megfelelő titkosítás esetén akár otthonról is képesek vagyunk rákapcsolódni, és ugyanolyan teljes értékű felületet érhetünk el, mint az irodából. A szoftver egyik előnye a platform függetlenség, ugyanakkor sajnos most meg kell említenem, hogy a program egyelőre csak ***MySQL*** adatbázis kezelővel képes működni, sajnálatos, hogy egyéb nyílt forrású adatbázis kezelők nem támogatottak. Ha a program használhatóságáról beszélünk, akkor érdemes megemlíteni, hogy a kezelői felület nagyon szépen, logikusan van felépítve, kimondottan a Honlap menüpont tetszik, melyen rögtön láthatjuk az adott napi teendőinket, és egyéb ügyeinket, így szinte percek alatt újra munkába tudunk lendülni. Hiányolom, hogy a nyomtatás csak azt jelenti, hogy egy sallangoktól mentes ***HTML*** lapot kapunk, melyet kinyomtathatunk a böngészővel, sokkal jobb lenne, ha mondjuk ***PDF*** formátumban, vagy ***PS***-ben is lehetőségünk lenne megkapni az listákat. Ellenben ***CSV***-be tudunk exportálni, így mondjuk például a Kapcsolatok könnyen kimenthetőek, és nyomtathatóak ***OpenOffice*** segítségével.

A 2.5-ös verzióban megjelent a ***Bug Report***, modul, ami szerintem hasznos lehet egy szoftver fejlesztő cégnek, de más tevékenységű cégek szerintem nem használják, ezért próbaképpen egy-pár felhasználónak kikapcsoltam a menüből ezt az opciót. A menüből el is tűnt a hivatkozás rá, de például az a ***Honlap*** menüpont alatti eszköztárban ettől még fel tudtam vinni új hibát, és szintén a Honlap menüpont alatt látható volt az általam felvitt Bug-ok listája is. Számomra ez elég furcsa volt, hogy így a modulokat nem tudom kikapcsolni, csak a főmenüből kiszedni.

További hiányossága a szoftvernek, hogy egyelőre nincs megoldva benne a jogosultság kezelés, tehát a marketing osztályunk láthatja az ügyfélszolgálaton történő eseményeket, természetesen a fejlesztők tervezik ennek a résznek a teljes kidolgozását, de egyelőre még ezzel várnunk kell.

Horváth Ernő (he305@hszk.bme.hu)

KAPCSOLÓDÓ CÍMEK

SugarCRM: ➲ http://www.sugarcrm.com/

SugarCRM (Sourceforge): ➲ http://sourceforge.net/projects/sugarcrm

Samba Windowsban is otthon (4. rész)

A cikksorozat előző részeiben az Olvasó megismerkedhetett a Samba alapvető használatával, azokkal a funkciókkal, amelyekkel egy Windows-os hálózatot ellátó kiszolgálót fel lehet építeni. Így az alapfunkciók tárgyalásának befejeztével rátérnék azokra a funkciókra, amelyekkel az eddigieknél magasabb szinten tudjuk a rendszert a felmerülő igényekhez alakítani.

A hálózat tallózása

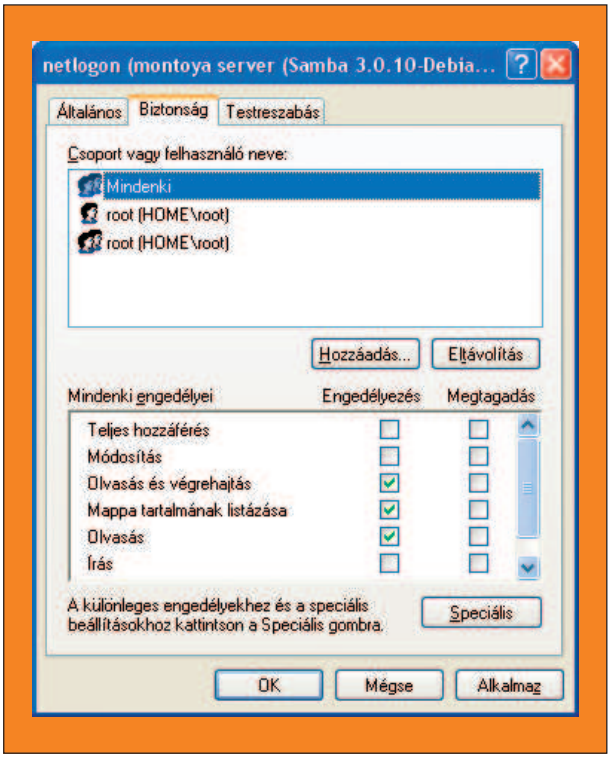
A hálózat tartalmának tallózása egy pofon egyszerű dolognak tűnik. Megnyitjuk a ***Network neighborhood*** – a magyar ***Windows*** alatt ***Hálózatok*** – ablakot és ott megtalálható a hálózatban jelenlévő gépek listája. A listából egy tetszőleges gépet kiválasztunk, ráklikkelünk és máris tallózhatjuk az adott gép tartalmát. A működés bár tényleg egyszerűnek tűnik, rengeteg szolgáltatás egyidejű együttműködését követeli meg.

Először is a ***Windows*** kliensnek regisztrálnia kell magát a hálózatban és tudatni a rendszerrel, hogy mostantól ő is aktív résztvevője annak. Majd ezek után a gépnek minden másik gép tudomására kell hozni a jelenlétét, hogy elérhetővé váljon. Egy, vagy több gépnek a hálózatban listába kell gyűjtenie a hálózatban elérhető gépeket, és a belépő munkaállomásnak ennél a gépnél regisztrálnia kell magát. A kliens gépeknek képeseknek kell lenniük arra, hogy a gépnév alapján meghatározzák egymás IP címét, hiszen egy ***TCP/IP*** alapú hálózatban a gépek címzése csak ezen a módon történhet. Végül pedig a kliens gépünk ezeken a szolgáltatásokon keresztül csatlakozni tud a kiszemelt célgéphez és elérheti annak erőforrásait.

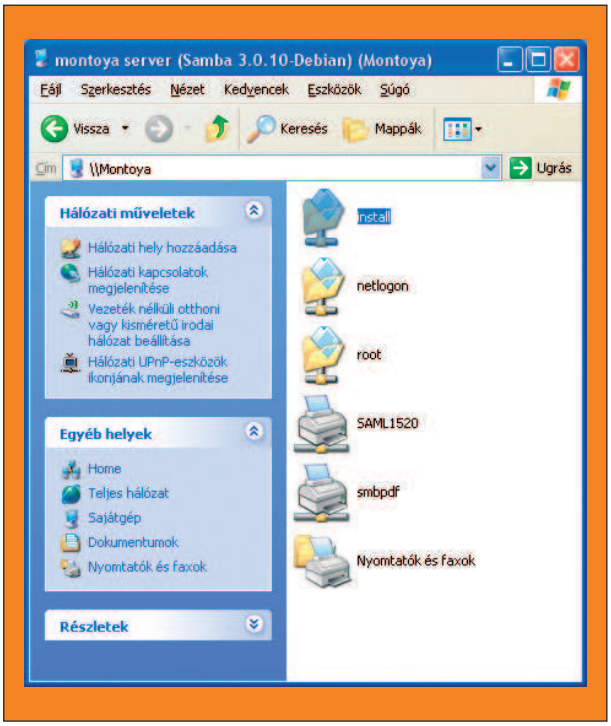
A ***Samba*** rendszer ***nmbd*** komponense biztosítja, hogy a hálózat tallózása, a gépek listába gyűjtése megtörténjen. A ***Samba*** kiszolgálón, a konfigurációs állományban pedig állítható akár kézzel is, hogy melyik gép végezze a listába gyűjtést. Mivel alapesetben ezt a szerepet (úgynevezett ***master browser***) bármelyik hálózati gép megkaphatja annak terhelésétől függően, ezért az adott gép kiválásával elképzelhető, hogy pillanatnyi fennakadás állhat elő- Különösen igaz ez akkor, ha például munkaidő végén a munkaállomások tömegesen hagyják el a hálózatot. Éppen ezért célszerű – természetesen a terhelések figyelembevételével – ezt a szerepet a kiszolgálóink valamelyikére kiosztani.

A NetBIOS protokoll

A ***Samba*** a ***Windows*** hálózatokban használt ***NetBIOS*** protokollon keresztüli kommunikációra is fel van készítve, hasonlóan a ***Windows*** kliensekhez a ***Samba*** is képes a ***TCP/IP*** fölött folytatni a ***NetBIOS*** protokollon keresztüli kommunikációt. A Windows kliensek hálózati kommunikációjának egy része – így többek között a gépek felderítése – üzenetszórással történik a hálózatban. Mivel azonban az útválasztók ezeket az úgynevezett ***broadcast üzeneteket*** nem engedik át, így amennyiben nagyméretű, de legalábbis útválasztók által szabdalt hálózat esetén a ***Sambát*** úgy kell beállítani, hogy ***unicast*** üzeneteket használjon. Ezen beállítások az ***smb.conf*** állományban végezhetőek el. ***Unicast*** üzenetek használata esetén a ***Sambát*** úgy kell beállítani, hogy támogassa a ***WINS*** alapú névfeloldást. Amennyiben több kiszolgálónk van a hálózatban, úgy lehetőségünk van a ***WINS*** kiszolgáló replikálására a hálózatban terheléselosz-



1. ábra Egy megosztás jogosultságai Windows alól nézve



2. ábra Egy Debian kiszolgáló tartalma Windows alól

tás és tartalékképzés céljából. Sajnos azonban jelenleg ***Windows*** és ***Samba*** kiszolgálók között ez a megoldás még nem támogatott, így amennyiben ***Samba*** kiszolgáló is van a hálózatban és az „master browser" szerepet tölt be, úgy csak az az egy ***nmbd*** entitás futhat.
Végezetül jegyezzük meg, hogy a névlisták összeállítása 15 perces intervallumonként automatikusan létrejövő hálózati üzenetek alapján történik, így egy-egy pontos, jól működő lista előállítása nagyobb hálózat esetén akár 45 percet is igénybe vehet.
Nézzük meg gyorsan, hogy milyen úton is halad végig egy név feloldása egy ***Windows*** munkaállomásnál. Először a kliens megnézi a lokális ***hosts*** állományt, amely teljesen hasonló a ***Linux*** rendszereken az ***/etc/hosts*** állományhoz, csak ez a ***%SystemRoot%\System32\Drivers\etc*** könyvtárban található. Amennyiben a ***hosts*** állomány alapján nem végezhető el a névfeloldás, akkor a kliens a beállított ***DNS*** kiszolgálóhoz fordul a keresett gép ***IP*** címéért. Amennyiben a ***DNS*** sem tartalmaz bejegyzést a keresett géphévhez, akkor harmadik lépésben a ***NetBIOS cache*** kerül vizsgálatra, majd pedig a beállított ***WINS*** kiszolgáló. Amennyiben egyik kiszolgáló sem tudott információval szolgálni, akkor egy ***broadcast*** üzeneteken alapúló keresés indul és minden gép megszólításra kerül, hátha válaszol a címzett. Amennyiben ez sem vezet eredményre, úgy utolsó lépésben a teljes ***LMHOSTS*** lekérdezésre kerül, amely alapértelmezésben a ***%SystemRoot%\System32\Drivers\etc*** könyvtárban helyezkedik el.

Kommunikáció TCP/IP felett NetBIOS használata nélkül

Az újabb ***Windows*** kliensek, mint a ***Windows 2000*** és ***Windows XP*** már támogatják a lokális hálózat használatát a ***NetBIOS*** protokoll nélkül, teljes egészében a szabványos ***DNS*** struktúrára épülve. Ebben az esetben a kliensek a hozzájuk tartozó tartományban automatikusan elvégzik a szükséges bejegyzések módosítását. ***Active Directory*** használata esetén ez olyannyira működik, hogy ***Active Directory*** létrehozásához szükség van egy megfelelően beállított és működő ***DNS*** kiszolgálóra.
A dinamikus ***DNS*** használata több szempontból is jó dolog, egyfelől egy szabványos, minden hálózati alkalmazás által támogatott protokollt használ, másfelől nekünk a munkaállomásnak csak egy nevet kell adni, valamint egy ***DHCP*** által kiosztott ***IP*** címet és a cím-név összerendelés automatikusan megtörténik. Ez a módszer egy tökéletesen működő megoldás ***Active Directory*** esetén, azonban mivel jelenleg a ***Samba*** nem működik ***Active Directory Serverként***, ezért ***Samba*** tartománykiszolgáló használata esetén kénytelenek vagyunk ***NetBIOS***-t használni. Amennyiben a ***Samba*** kiszolgálónk egy ***Active Directory*** tagja, akkor természetesen a dinamikus ***DNS*** használata megoldható. (A linuxos dinamikus ***DNS*** használatához először tájékozódjunk a ***BIND9*** leírásában!)

Munkacsoport tallózás beállítása

Amennyiben a hálózatunkon munkacsoportokba gyűjtjük a munkaállomásokat és szeretnénk lehetővé tenni a hálózaton keresztüli tallózást, úgy a ***Samba*** kiszolgálónkat a hálózat úgynevezett ***Domain Master Browserévé*** kell tennünk. Ez nem azonos a tartományvezérlővel, bár az elsődleges tartományvezérlő a hálózatban szintén ellátja ezt a feladatot. A ***Domain Master Browser*** feladata, hogy összegyűjtse a ***Local Master Browserektől*** az adott alhálózatban szereplő gépek listáját. A ***Domain Master Browser*** szolgáltatás hiányában a hálózatunk nem lesz több, mint lokális alhálózatok csoportja, amelyek egymással való kommunikációra ***Windows*** hálózat használatával egyszerűen képtelenek.
Egy munkacsoportos környezetben a ***Domain Master Browsernek*** a ***Samba*** kiszolgálót kell kijelölni és egy adott munkacsoporthoz csak egy ***Domain Master Browser*** tartozhat. Ennek a szerepnek a kiosztását a ***Samba*** konfigurációs állományában tehetjük meg a `domain master = yes` paraméter megadásával. A ***Samba*** kiszolgálót érdemes úgy beállítani, hogy a saját alhálózatának ő legyen a ***Domain*** és ***Local Master Browsere*** egy személyben. Ehhez a globális beállításokat egészítsük ki a következő bejegyzésekkel:

```
[global]
domain master = yes
local master = yes
preferred master = yes
os level = 65
```

Ha ezzel megvagyunk, akkor győződjünk meg arról, hogy minden alhálózatunknak is megvan a saját ***Local Master Browser*** gépe. ***Local Browser*** szerepére nem feltétlenül kell kijelölnünk egy ***Samba*** kiszolgálót, ezt a szerepet bármely Windows munkaállomás betöltheti. Amennyiben azonban úgy döntünk, hogy a ***Samba*** végezze ezt a feladatot is, akkor a ***Local Master Browser*** gép beállításai közé vegyük fel az alábbi paramétereket:

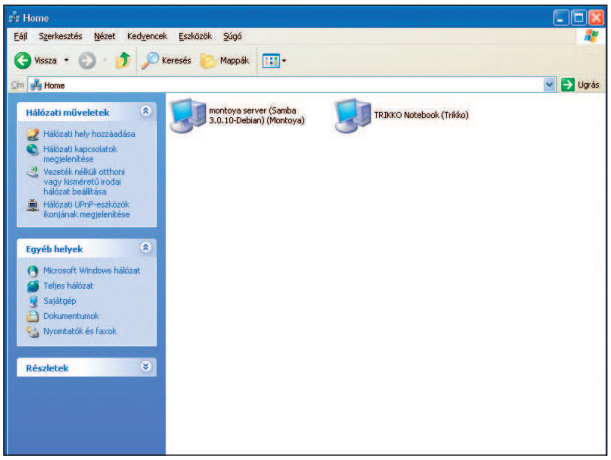
```
[global]
domain master = no
local master = yes
preferred master = yes
os level = 65
```

Fontos azonban, ahogy ***Domain Master Browserből*** is csak egy lehet a hálózatban, ***Local Master Browserből*** is csak egy lehet az adott alhálózaton. Ha tehát beállítunk még egy ***Samba*** kiszolgálót erre a feladatra, akkor a két szerver csúnyán össze fog veszni, aminek az lesz a következménye, hogy szép ki hálózati forgalmat generálnak, mellesleg nem is fognak rendesen működni.
Ha végeztünk a hálózati struktúra kialakításával, akkor most nézzük meg, hogy miként is tudjuk a megosztásokon a hozzáférési jogokat beállítani, akár megosztásonként különböző módon. Erre nagy szükségünk lehet egy bonyolultabb felhasználói struktúra esetén és az adatok biztonság érdekében nagy odafigyelést igényel.

Tartományi felhasználók és munkaállomások

Amennyiben ***Samba*** kiszolgálót használunk ***Windows*** tartomány kezelésére, úgy a rendszerben természetes módon létre kell hozunk azokat a felhasználókat, akiknek a későbbiekben tartományi elérést szeretnénk biztosítani. Ekkor a felhasználó egyfelől bekerül a ***Linux passwd*** állományába, másfelől a ***Samba*** háttér adatbázisába. A felhasználók kezelése mellett szükség van továbbá azokra a munkaállomásokra is, amelyeknek lehetőséget szeretnénk biztosítani a tartományi eléréshez, így ezeket a munkaállomásokat is nyilván kell tartani. A munkaállomások nyilvántartása egy helyen történik a felhasználók tárolásával, annyi különbséggel, hogy ezek az objektumok speciális felhasználói fiókként jönnek létre. A specialitás abban jelentkezik, hogy a felhasználói név $ karakterrel kezdődik, valamint az alapértelmezett munkakörnyezet a ***/bin/false***, tehát ilyen felhasználó a ***UNIX*** rendszerbe nem tud belépni, valamint a saját könyvtára a ***/dev/null***, tehát semmilyen személyes adat nem kerül tárolásra.
A munkaállomás felhasználói fiókjának létrehozása akkor történik, amik a ***Windows***-zal csatlakoztatjuk a munkaállomást a tartományhoz. Ezt a műveletet minden esetben csak tartományi rendszergazda jogokkal bíró felhasználó teheti meg – mint alapesetben a root felhasználó.
Windows rendszerekben megszokott dolog, hogy bizonyos felhasználói körökhöz speciális jogokat lehet rendelni, így például a tartományhoz való munkaállomás csatolást. Ilyen egyedi jogkörök kialakítása jelenleg még nem támogatott, de a ***Samba 3.0.11***-es verziótól kezdve folyamatosan fognak ezek a funkciók is bekerülni a rendszerbe.
A felhasználók jogosultsági szintekhez való hozzárendelése úgy fog kinézni, hogy egyfelől engedélyezni kell majd a ***Samba*** beállításai között ezt a funkciót a `enable privileges = yes` paraméter beállításával, valamint az egyes jogosultsági szintekhez hozzá kell adni a felhasználókat.
Jelenleg (3.0.11-es változat) a következő öt szint létezik:

- `SeMachineAccountPrivilege` – Ezek a felhasználók gépeket csatolhatnak a tartományhoz.



3. ábra Egy tartomány, ahol egy Debian a kiszolgáló

- `SePrintOperatorPrivilege` – Ezek a felhasználók menedzselhetik a nyomtatókat.
- `SeAddUsersPrivilege` – Ezzel a jogosultsági szinttel rendelkező felhasználók új felhasználókat és csoportokat adhatnak a tartományhoz.
- `SeRemoteShutdownPrivilege` – Ezzel a joggal rendelkező felhasználók használhatják a tartományban a távoli leállítás funkciót.
- `SeDiskOperatorPrivilege` – Ezek a felhasználók pedig szabályozhatják a lemezeke kiosztását.

Jogosultságok megadása a megosztásokra

Az előbb áttekintettük, hogy miként lehet az egyes felhasználóinkhoz jogosultsági csoportokat rendelni, most pedig nézzük meg, hogy miként lehet az egyes megosztásokhoz a felhasználóink hozzáférését szabályozni.
Azt szeretnénk elérni, hogy a különböző megosztásokat a rendszerben az egyes felhasználók láthassák, vagy éppen ne láthassák, az ott lévő állományokat megnyithassák olvasásra, vagy éppen írásra, továbbá hasznos lenne az is, ha adott megosztásra a megosztást elérő felhasználók egy adott felhasználói jogosultsággal végeznének munkát.
Mivel a windowsos és a ***UNIX***-os felhasználói jogosultság filozófiája jelentősen különbözik egymástól, ezért a jogosultságok megfeleltetés és kialakítása nem egyszerű feladat. Látnunk kell, hogy a két rendszer alapfelfogása különböző. Míg a ***UNIX*** rendszerekben a történelmi gyökerek, az akadémiai felhasználás miatt egy-egy dokumentum tulajdonosa és ezáltal a jogosultságok kiosztására hivatott felhasználó az adott dokumentumot létrehozó egyén, addig a ***Windows*** rendszerekben a kereskedelmi struktúra miatt egy-egy dokumentum nem a létrehozójának, hanem a rendszer felügyelője által beállított személy, vagy csoport tulajdonába kerül. Ez a szemléletbeli különbség kis problémát jelent, ám ügyes szervezéssel könnyen áthidalható, méghozzá oly módon, hogy egyes vélemények szerint a végén egy könnyebben átlátható és karbantartható rendszert kapunk.

1. táblázat

Paraméter	Leírás	Használat
`admin users`	Ebbe a listába felsorolt felhasználók az adott megosztásra rendszergazdai jogokkal rendelkeznek, így minden olyan utasítást végre tudnak hajtani, amit a `super-user` végrehajthat.	`admin users = user1, user2`
`force group`	Ennek a paraméternek adott érték azt a UNIX csoportot jelenti a rendszerben, amelynek tulajdonába fog kerülni minden az adott megosztáson elhelyezett állomány.	`force group = users`
`force user`	Hasonlóan működik a `force group`-hoz, csak itt felhasználót kell megadni.	`force user = root`
`guest ok`	Amennyiben ez a paraméter aktív, akkor az adott megosztás elérésére jelszó és hitelesítés nélkül is lehetőség van. Érdemes a nyilvános mappák használatánál alkalmazni.	`guest ok = yes/no`
`invalid users`	Ez egy nagyon hasznos beállítási lehetőség, felsorolhatjuk azokat a felhasználókat akiket explicit nem akarunk hozzáférési joggal felruházni. Nagyon hasznos, ha sok felhasználó esetén egy-két felhasználót kell letiltani.	`invalid users = user1, user2`
`only user`	Ez szintén egy bináris kapcsoló, bekapcsolva beállíthatjuk, hogy csak csak olyan felhasználókkal kezdeményezett kapcsolatok épüljenek ki, ahol a felhasználó neve szerepel a felhasználói listában.	`only user = yes/no`
`read list`	A `read list` listában felsorolt felhasználók jogosultak az adott megosztáson olvasási joggal hozzáférni a tárolt adatokhoz. A `read only` paraméter beállításától teljesen függetlenül ezek a felhasználók csak olvasási joggal fognak a megosztáson rendelkezni.	`read list = user1, user2`
`valid user`	Amennyiben implicit módon szeretnénk egy megosztásra definiálni azon felhasználók körét, akik hozzáféréssel rendelkeznek a megosztáshoz, akkor a `valid user` listába kell őket felvenni. Azok a felhasználók akik nem szerepelnek a listában hozzáférés megtagadása hibaüzenetet fognak kapni.	`valid user = user1, user2`
`write list`	A `read list`-hez hasonlóan ebben a listában azokat a felhasználókat kell felsorolni, akik írási joggal is rendelkeznek a szóban forgó megosztáson.	`write list = user1, user2`

Talán a legegyszerűbb megoldás, ha a rendszerben meglévő dokumentumokat két részre bontjuk. Az egyik a minden felhasználó által elérhető publikus anyagok, a másik az egyes felhasználók által használt privát, vagy bizalmas anyagok. Érdemes ezek után a struktúrát oly módon kialakítani, hogy a publikus, közös anyagokat egy külön kiosztás keretében oly módon tesszük közzé, hogy ahhoz minden felhasználó korlátlan módon hozzáférjen.
A bizalmas jellegű, vagy csak néhány felhasználó által használt anyagokat pedig olyan könyvtárstruktúrában helyezzük el, amit aztán a ***Linux*** kiszolgálón az adott felhasználó személyes mappájába – a ***HOME*** kiosztásba – linkelés útján helyezünk el.
Ez a megoldás már egy egész jó jogosultságkezelést biztosít, ám előfordulhat, hogy szeretnénk olyan megosztásokat készíteni, amit egyes felhasználók csak olvashatnak, mások írhatnak, míg megint mások nem is láthatnak. Erre szolgálnak az alábbi paraméterek, amelyeket a ***Samba*** konfigurációs állományában az adott megosztásnál kell elhelyezni.

Ezzel sikerült is áttekintenünk a ***Samba*** és a ***Windows*** rendszerek közötti jogosultságkezelési megoldásokat, amelyek segítségével a legbonyolultabb felépítésű hálózat kialakítása sem jelenthet problémát. Mielőtt azonban erőből nekiesünk egy-egy hozzáférés kialakításának, fordítsunk időt a tervezésre, mert egy felhasználók által elfogadott és programokba beállított struktúra átalakítása sok problémával és fejfájással járhat. Jobb az ilyet messze elkerülni.
A sorozat következő részében folytatjuk a ***Samba*** haladó funkcióinak áttekintését, addig is mindenkit arra ösztönzök, hogy próbálgassa a fent leírtakat.

Illés Viktor (viktor@ei.hu)
23 éves, a BME műszaki informatikus szakának hallgatója, mellette weblapokkal, linuxos és windowsos rendszerekkel foglalkozik. Szabadidejét legszívesebben a szabadban tölti, teniszezik és kerékpározik.

FreeBSD – a szomszéd vár (7. rész)
Tűzfal készítése

A fájlrendszer megtekintése után elérkeztünk a FreeBSD igazi területéhez, a hálózat kiszolgálásához. Ezen belül először egy olyan területet vizsgálunk meg, amely szinte minden hálózati rendszer esetén alapfeladat: tűzfallal védeni egy számítógépet. Ehhez alapvetően szükségünk lesz egy hálózati csomagszűrő programra, vagy tűzfal programra.

Mi az a tűzfal?

A tűzfalak két hálózatot elválasztó eszközök, bővebb értelemben ezek teszik lehetővé, hogy megszűrjük a rajtuk átfolyó a be-, illetve kimenő forgalmat. Ennek értelmében a tűzfalak feladata a belső (személyes) hálózat számára különféle szolgáltatások nyújtása, illetve ésszerű korlátozások alkalmazása. Gondoljunk csak általánosságban arra az esetre, amikor egy cég a széles sávnak becézett vékonyka ***ADSL*** kapcsolattal tartja a világhálóval a kapcsolatot. Az ***ADSL*** sávszélessége ugyanis nagyon sok mindent kibír, amíg meg nem jelenik a belső hálózaton néhány notórius (film)letöltögető kolléga, akik miatt a többi munkatárs esetleg a munkájában lesz akadályozva. Érdemes ebben az esetben egyfajta letöltési korlátozást bevezetni, így egyenlő esélyt adni minden dolgozónak. Szolgáltatás tekintetében megfontolandó a web adatforgalmát gyorstáron átvezetni, ezzel – kismértékben ugyan, de – gyorsíthatjuk a böngészést.
A tűzfalak fő feladata a csomagok szűrése, s ez sok feltételtől függhet: a feladó vagy a címzett IP címétől, a forrás vagy cél porttól, a csomagok állapotától vagy az integritásától. A szűrés mindenképpen magasabb biztonságot jelent, bár az túl erőteljes korlátozás a munkát is zavarhatja.

Milyen tűzfalak vannak?

Az elkészített tűzfalak két nagy csoportba sorolhatók: megengedő és tiltó. Az előbbi csoport alapvetően minden forgalmat enged, és ezt tudjuk szabályokkal szűkíteni. A tiltó tűzfal mindent tilt, és szabályokkal lehet engedélyezni a meghatározott forgalmat. Az előbbi kényelmesebb, a második biztonságosabb. A másik csoportosítási mód szerint léteznek úgynevezett „***stateful***” tűzfalak, amelyek a kapcsolat kiépítésétől egészen annak megszűnéséig követik az adatokat, ezzel egy sor biztonsági problémát megakadályoznak (csak olyan hálózati csomagokat engednek át, amelyek egy új kapcsolatot nyitnának, illetve egy már létezőhöz tartoznak). Megkülönböztetünk személyes és vállalati tűzfalakat is. Az előbbiek egy-egy gépet védenek a „külvilágtól”, a vállalati tűzfalak a vállalat minden számítógépét a külvilágtól. Szokás a két tűzfalat kombinálni, így a belső hálózat felől induló támadások ellen is védettek a számítógépek, és a vállalaton kívüli támadásoktól is. Ebben a cikkben inkább a személyes tűzfalak kialakításáról szólok, a a vállalati tűzfalakkal következő részben foglalkozunk.

Milyen eszközeink vannak a megvalósításhoz?

Linux esetén minden feladatot a rendszermagba ágyazott modulokkal, és a modulokat kezelő ***iptables*** programmal tudunk megoldani. ***FreeBSD*** esetén a feladatok szét vannak választva, illetve több eszközt is használhatunk, ezek az ***IPFilter (IPF)***, az ***IPFirewall (IPFW)*** és a ***PacketFilter (PF)***. Az ***IPFW*** tartalmaz ***sávszélesség korlátozó eszközt (DummyNet)***, amely kényelmesen használható. Az ***IPF*** ilyet nem tartalmaz, de használhatjuk hozzá az ***AltQ*** programot a ***ports*** adatbázisból. A ***PF*** az ***OpenBSD*** rendszerből származik, szintén kényelmes eszköz a tűzfal elkészítéséhez. Én az ***IPFW*** programot használom, így ezt fogom bemutatni, de a többi eszköz is pont ennyire használható.

Az IPFW beállítása

A rendszermag már tartalmazza modulként a jelenlegi ***FreeBSD*** terjesztések esetén a szükséges komponenseket, így nem kell újrafordítanunk a rendszermagot.
A */etc/rc.conf* állományhoz kell a

```
firewall_enable="YES"
```

sort hozzáfűznünk, és a következő induláskor már működik is az ***IPFW*** rendszermag modul. Természetesen nem kell újraindítani a kiszolgáló gépünket, mivel az ***ipfw*** nevű modul betöltésével azonos eredményt érünk el

```
$ kldload ipfw.ko
```

így a következő induláskor az ***rc.conf*** szerint fog a modul betöltődni. Ellenőrizzük le, hogy működik-e a tűzfal program, a rendszermag üzenetei között meg kell látnunk az alábbi sort:

```
ipfw2 initialized, divert disabled, rule-based
➥forwarding disabled, default to deny, logging
➥disabled
```

Különféle finomítások

A tűzfal beállítása során a legfontosabb, hogy értesüljünk a szabályok működéséről, illetve a program üzeneteiről. Ehhez a naplózást be kell állítani, és ideiglenesen bőbeszédű naplózást is beállíthatunk. Ehhez a `sysctl` programmal a

```
$ sysctl net.inet.ip.fw.verbose=1
```

parancsot kell kiadnunk (vagy ezt be is írhatjuk a */etc/sysctl.conf* állományba).
Az *rc.conf* állományban megadhatjuk a tűzfalszabályokat tartalmazó állomány nevét, amely alapértelmezés szerint a */etc/rc.firewall*.

```
firewall_script="/etc/ipfw.rules"
```

Az ipfw program

A rendszermag modulja valósítja meg a tűzfalat, illetve kezeli annak szabályait. A modul működését egy programmal tudjuk, amely `ipfw` névre hallgat. Ha betöltöttük a modult, akkor azonnal elvágtuk magunkat a külvilágtól, mivel a program minden forgalmat tilt, amely a hálózati csatolófelületeken át folyna. A modul betöltésétől óvakodjunk olyan gépen, amelytől fizikailag messze vagyunk, mivel izzasztó meglepetés érhet minket: nem férünk hozzá a géphez a hálózaton át. Ezt az alapvető viselkedést a rendszermag `IPFIREWALL_DEFAULT_TO_ACCEPT` opciójával tudjuk elkerülni, de ez biztonsági megfontosálok okán nem ajánlatos.
Az alapértelmezett viselkedésről a parancs `list` opciójával győződhetünk meg:

```
$ ipfw list
65535 deny ip from any to any
```

A 65535 szám már sejteni enged egy korlátot: a szabályok maximális száma mindössze ennyi lehet. A számok szerint kerül ellenőrzésre az adott szabály, a tiltás tehát a legutolsó a sorban; ha lenne előtte bármilyen másik szabály, akkor először azt értékeli ki. A tiltás ténye igen látványos a programok felől, ugyanis jogosultsági problémára panaszkodnak majd a hálózatot használni kívánó programok:

```
$ ping -c 1 10.1.1.211
PING 10.1.1.211 (10.1.1.211): 56 data bytes
ping: sendto: Permission denied

--- 10.1.1.211 ping statistics ---
1 packets transmitted, 0 packets received, 100%
➥packet loss
```

Új szabály hozzáadása nagyon beszédes formában történik, az `ipfw` parancssora olyan, mintha angolul elmondanánk a kívánságainkat:

```
$ ipfw add 1 allow ip from me to any
00001 allow ip from me to any
$ ipfw list
00001 allow ip from me to any
65535 deny ip from any to any
```

Ezzel a szabállyal engedélyeztük a saját címről (`me`) az összes másik cím (`any`) felé induló csomagok továbbítását.

Gondolhatnánk, hogy ezzel megoldottunk minden szükséges beállítást, azonban ez kevésnek bizonyul:

```
$ ping -c 1 10.1.1.211
PING 10.1.1.211 (10.1.1.211): 56 data bytes

--- 10.1.1.211 ping statistics ---
1 packets transmitted, 0 packets received, 100%
➥packet loss
```

A hiba kiírása ugyan megszűnt, a csomagok vígan átjutnak a tűzfalon, elérik a célként megnevezett gépet is, onnan a válaszcsomag visszaindul, a tűzfalunkon azonban fennakad. Hozzá kell vennünk a szabályokhoz egy újabb szabályt, amely engedi a forgalmat a mi gépünk felé:

```
$ ipfw add 2 allow ip from any to me
00002 allow ip from any to me
$ ipfw list
00001 allow ip from me to any
00002 allow ip from any to me
65535 deny ip from any to any
```

S már működik is a kommunikáció, a csomagok oda-vissza átjutnak a személyes tűzfalon.

```
PING 10.1.1.211 (10.1.1.211): 56 data bytes
64 bytes from 10.1.1.211: icmp_seq=0 ttl=64
➥time=0.308 ms
```

Naplózás beállítása

A napló vezetése nagyon fontos a hibakeresésnél és a visszakereshető dokumentálás okán. Általában a visszautasított kapcsolatokat szoktuk naplóba vezetni, a legális fogalom naplózása feleslegesen rabolja a gépidőt, illetve időrabló az elemzése is. A feladatunk mindössze annyi, hogy a naplózni kívánt szabálynál az `allow` (`accept`, stb) kulcsszó után felvesszük a `log` kulcsszót is:

```
$ ipfw add 3 allow log ip from 10.1.1.211 to me
```

A fenti szabály működéséhez azonban az előtte lévő másik szabályt át kell mozgatni e szabály utáni pozícióra, ugyanis az elkapja az összes (`any`) beérkező csomagot. Ezt egy törléssel majd egy új hozzáadással tudjuk megtenni:

```
$ ipfw show
00001  981  77572 allow ip from me to any
00002   88  39287 allow ip from any to me
00003  577 160451 allow log ip from 10.1.1.211 to me
65535 1687 294213 deny ip from any to any
$ ipfw delete 2
$ ipfw add 4 allow ip from any to me
00004 allow ip from any to me
$ ipfw show
00001 1313 101607 allow ip from me to any
00003   14   5486 allow log ip from 10.1.1.211 to me
00004    0      0 allow ip from any to me
65535 1879 326108 deny ip from any to any
```

Ha mindent jól csináltunk, akkor a */var/log/security* állományban megjelennek a naplózott forgalmak:

```
ipfw: 3 Accept TCP 10.1.1.211:110 10.1.1.210:63187
➥in via fxp0
last message repeated 8 times
```

Éles helyzetben jól jöhet, hogy megadhatjuk a maximális számát a naplózott soroknak (gondoljunk csak egy jól kivitelezett *DoS* támadásra). Ezen sorok számát lehetőségünk van globálisan, illetve szabályokra lebontva meghatározni. Globálisan a

```
$ sysctl net.inet.ip.fw.verbose_limit=100
```

parancs állítja be ezt, s így minden egyes szabály csak 100 sorral szaporítja a naplóállományunkat, ezzel elegendő információt szolgáltatva a hibákról (vagy támadásokról). Érdemes rendszeresen nullázni a naplózott sorok számát, amely a

```
$ ipfw resetlog
```

parancs hatására fog bekövetkezni, így például minden nap tiszta lappal indulhat a naplózás.
Egyes szabályok esetén külön megadhatjuk a naplózott sormennyiséget, egyszerűen a `logamount n` paramétert kell megadnunk:

```
$ ipfw add 10 allow log logamount 4 ip from
➥10.1.1.211 to me
```

Szűrés port szerint

Személyes tűzfal esetén érdemes szűrni néhány portra, így meghatározhatjuk a legális szolgáltatásokat, amelyeket a helyi hálózat számára használni engedünk. A megoldáshoz egyszerűen csak a megadott IP címek mögé kell írni a port számát:

```
$ ipfw add 2 allow log ip from any to me 22
00002 allow log ip from any to me dst-port 22
$ ipfw list
00001 allow ip from me to any
00002 allow log ip from any to me dst-port 22
00003 allow log ip from 10.1.1.211 to me
00004 allow ip from any to me
65535 deny ip from any to any
```

A naplózást tehát kicseréljük, és csak azon csomagokat kérjük a naplóba írni, amelyek a gépünk 22-es portjára (SSH) érkeznek, s a ***security*** állományban látnunk kell a kéréseket:

```
ipfw: 2 Accept TCP 10.1.1.211:55978 10.1.1.210:22
➥in via fxp0
```

Ha naplózás és engedélyezés helyett eldobnánk a csomagot (`drop`), vagy hibát küldenénk vissza (`unreach`), akkor egy kis változtatással ez is könnyedén mehet:

```
$ ipfw delete 2
$ ipfw add 2 unreach net log ip from any to me 22
00002 unreach net log ip from any to me dst-port 22
$ ipfw list
00001 allow ip from me to any
00002 unreach net log ip from any to me dst-port 22
00003 allow log ip from 10.1.1.211 to me
00004 allow ip from any to me
65535 deny ip from any to any
```

Ha kapcsolódni szeretne valaki, akkor erről értesülni fogunk:

```
ipfw: 2 Unreach 0 TCP 10.1.1.211:58164
➥10.1.1.210:22 in via fxp0
```

Miközben a kapcsolódni vágyó az

```
ssh: connect to host 10.1.1.210 port 22: No route
➥to host
```

üzenetet kapja. Az `unreach` paraméterét (`net`) változtatva szinte bármilyen „hibát” elő tudunk állítani.

Tűzfal programocska

A `firewall_script` paraméterében megadott állományt feltölthetjük a kialakított tűzfal szabályokkal, így minden rendszerinduláskor megfelelően fog működni a személyes tűzfalunk. Az adott állomány tartalma lehet például a következő pár sor:

```
ipfw -q -f flush
ipfw add 100 allow ip from me to any
ipfw add 200 unreach net log ip from any to me
➥dst-port 22
ipfw add 300 allow log ip from 10.1.1.211 to me
ipfw add 400 allow ip from any to me
```

Érdemes az egyes szabályok számozását megfelelően nagy különbségre hagyni, így könnyedén kettő meglévő szabály közé tudunk egy újabbat szűrni különösebb nehézség nélkül:

```
ipfw add 200 unreach net log ip from any
➥to me dst-port 22
ipfw add 250 allow log ip from 10.1.1.211
➥to me dst-port 80
ipfw add 300 allow log ip from 10.1.1.211
➥to me
```

További lehetőségek

Az `ipfw` parancs többi lehetősége a kézikönyve oldalain megtalálható, illetve a következő részben – egy a vállalati tűzfal ismertetésével – sok más hasznos tulajdonságát ismerhetjük meg.

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

KAPCSOLÓDÓ CÍMEK

A FreeBSD projekt honlapja: ➲ http://www.freebsd.org

A magyar FreeBSD honlap: ➲ http://www.freebsd.hu

A magyar BSD honlap: ➲ http://www.bsd.hu

A kézikönyv magyar fordítása
➲ http://www.enaplo.hu/FreeBSD/handbook/

Az eFltk bemutatása (2. rész)

Alapvető elemek és ablakok.

Egy grafikus elemkészlet megismerésénél fontosnak tartom, hogy megvizsgáljuk a legtöbb elem alapját képező osztályokat, mint amilyen az ***eFltk***-ban a `Fl_Widget`. Amint az később is látható lesz majd, ebben az eszközkészletben az elemek elnevezése bizonyos megszokásokat mutat. Mivel az ***eFltk*** őse az ***Fltk***, így az abban megtalálható elnevezéseket követték a szerzők, vagyis az `Fl_` karakterek után következik az elem működésére utaló név. Kezdjük tehát az egyik legfontosabb elemmel, hiszen ennek megismerése a legtöbb elem esetén használható osztályfüggvényeket biztosít számunkra. Ez az elem az `Fl_Widget` nevet viseli.

Az osztály létrehozására alkalmas konstruktor legkevesebb négy, de legfeljebb öt paramétert vár. Az első kötelezően meghatározza az elem kezdeti helyzetét és méretét, és ötödikként megadhatjuk még az elem feliratát meghatározó karakterláncot.

Az első néhány függvény a grafikus elem aktív állapotára vonatkozik (`active(); active_r(); activate(); deactivate()`). Ha egy elem aktív állapotban van, akkor képes az események kezelésére és azokat meg is kapja az eseménykezelő ciklusban. Az `active()` és az `active()` függvények a lekérdezésre szolgálnak (az `_r` itt a rekurzióra utal vagyis az `active_r()` függvény csak akkor tér vissza igaz értékkel, ha az adott elem és annak minden szülője is képes fogadni az eseményeket.

Az elemek feliratának igazítását az `align()` eljárással állíthatjuk be. Ahogyan a grafikus tervezőben is láthatjuk a feliratokat többféle módon igazíthatjuk a vezérlőn belül. Ezeket az igazítási módokat az `FL_ALIGN_` kezdetű változók határozzák meg.

Az egyik legfontosabb függvény a `callback()` mellyel meghatározhatjuk az eseménykezelő függvényt. Általában egy eljárásról vagyis visszatérési érték nélküli `void` típusú függvényről van szó, amint az alábbi listán is látható.

```
void call_back(Fl_Widget* w, void* data)
{
        eseménykezelő_utasítások();
}
```

A függvénynek két paramétere van, az elsőben kapja meg, hogy melyik grafikus elem hívta meg az adott eseménykezelő eljárást, míg a másodikban az adott elemhez tartozó felhasználói adatokat kapjuk. Így hasonló elemeket az eseménykezelő eljárásban az adatai alapján tudunk megkülönböztetni. Lássunk egy példát arra is, hogy hogyan tudjuk beállítani az eseménykezelő eljárást. A példában egy gomb lenyomásához tartozó eljárást állítjuk be:

```
button1->callback((Fl_Callback*)call_back, NULL);
```

Vagyis látható, hogy típuskonverziót kell végeznünk, hogy beállíthassuk az esemény kezeléséhez készített eljárásokat és függvényeket. Az elemek színének beállítására szolgál a `color()` osztályfüggvény. Paramétereként egy `Fl_Color` típusú értéket vár, melyet előállíthatunk az `fl_rgb_color()` függvény segítségével. Ez utóbbi függvénynek paraméterként a kívánt vörös, zöld és kék összetevőket kell átadni, visszatérési értéke pedig a megfelelő típusú szín lesz.

Az `image()` és a `deimage()` osztályfüggvények `Fl_Image` típust várnak, és meghatározható segítségükkel, hogy az elem aktív- és inaktív állapotában milyen képet jelenítsen meg a grafikus elem.

A `label()` metódus segítségével adhatjuk meg az elem címkéjének feliratát, míg a `labelcolor()` (szintén `Fl_Color` típusú paraméterrel) a címke színét határozza meg.

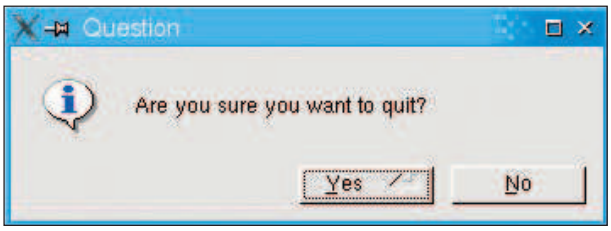
A `show()` és a `hide()` függvények segítségével megjeleníthetjük vagy elrejthetjük az adott grafikus elemet, míg a `visible()` és a `visible_r()` függvényekkel lekérdezhetjük a láthatóságát meghatározó információkat. Itt szintén a `_r` változat szolgál a szülők állapotának lekérdezésére.

A grafikus elem mérete és helyzete beállítható a `size()` és a `position()` eljárások segítségével, míg ezen információk lekérdezésére a `w()` (szélesség), a `h()` (magasság), `x()` (x-koordináta) és az `y()` (y-koordináta) függvények alkalmasak.

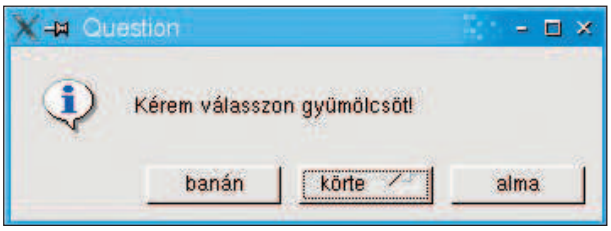
A `resize()` eljárással a méretet és a helyzetet egyszerre állíthatjuk be, az első két paraméterben átadva az új helyzetet meghatározó értékeket, majd a harmadik és a negyedik paraméterben az új méretet adjuk meg.

A `redraw()` és a `redraw_label()` eljárások segítségével ismételten kirajzolhatjuk a grafikus elemet vagy annak címkéjét, míg a `damage()` függvény visszaadja, hogy szükség van-e újbóli kirajzolásra. Amennyiben az elemet valami eltakarta az idők során, akkor a `damage()` függvény nullától különböző értéket ad vissza.

Fontos még megismerni a `tooltip()` függvényt is, hiszen ennek segítségével hozhatjuk létre vagy kérdezhetjük le a grafikus elemhez rendelt gyorstippet. Amennyiben az eljárásnak egy karakterláncot adunk át paraméterként, akkor

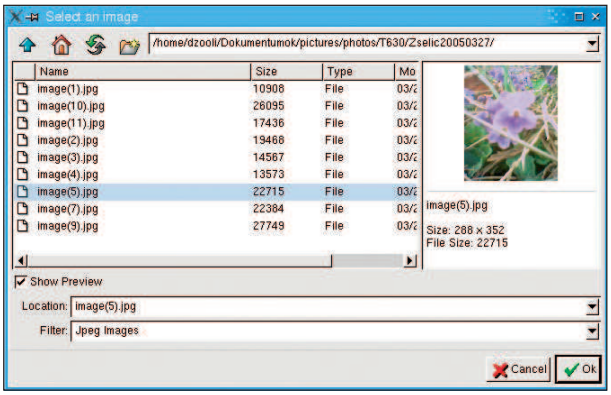


1. ábra eFltk kérdezőablak



2. ábra eFltk többszörös választás

3. ábra eFltk beviteli ablak



4. ábra Állományválasztó

a gyorstipp beállításáról gondoskodik, míg ha nem adunk paramétert, akkor annak lekérdezéséről. Ez utóbbi megoldás a legtöbb ***eFltk***-ban használatos osztályfüggvényre igaz. Hasznos lehet még a `window()` osztályfüggvény, melynek segítségével lekérdezhetjük a grafikus elemhez tartozó `Fl_Window` típusú objektumot. Egy valódi ablak esetében azonban ügyelni kell arra, hogy a függvény nem az adott ablakhoz tartozó mutatót adja vissza (hiszen annak segítségével hívtuk meg), hanem az aktuális ablak szülőablakát kapjuk visszatérési értékként.

A gyakorlatban majd látni fogjuk, hogy valójában az objektum-orientált megközelítés mennyire leegyszerűsíti a programozók dolgát. Nem kell más csak egy jó referencia és egy ábra az osztályhierarchia felépítéséről és máris tisztában vagyunk a lehetőségeinkkel és eldönthetjük, hogy valóban erre az eszközkészletre van-e szükségünk. Javaslatom szerint, ahol kicsi és gyors programokra van szükség, ott mindenképpen érdemes fontolóra venni az ***eFltk*** használatát.

Így röviden áttekintve a fontosabb osztályfüggvényeket, egy nagyon hasznos lehetőségre szeretném felhívni a kedves olvasók figyelmét. Az ***eFltk***-ban és már az elődjében is találkozhatunk előre elkészített párbeszédablakokkal. Következzen ezek áttekintése, megismerése és alkalmazása. Az ismerkedés után már megpróbálhatjuk elkészíteni az első programunkat, ami ugyan kezdetben nem csinál majd semmi hasznosat, de azt legalább felhasználói azonosító és jelszó bekérése után teszi. Itt láthatjuk majd igazán az előre elkészített párbeszédablakok használatát.

Lássuk tehát az ***eFltk***-ban használható párbeszédablakokat szép sorjában. Ahhoz, hogy használatba vehessük ezt a lehetőséget be kell illesztenünk a megfelelő fejléc állományt. Ezt az alábbi makró segítségével tehetjük meg:

```
#include <efltk/fl_ask.h>
```

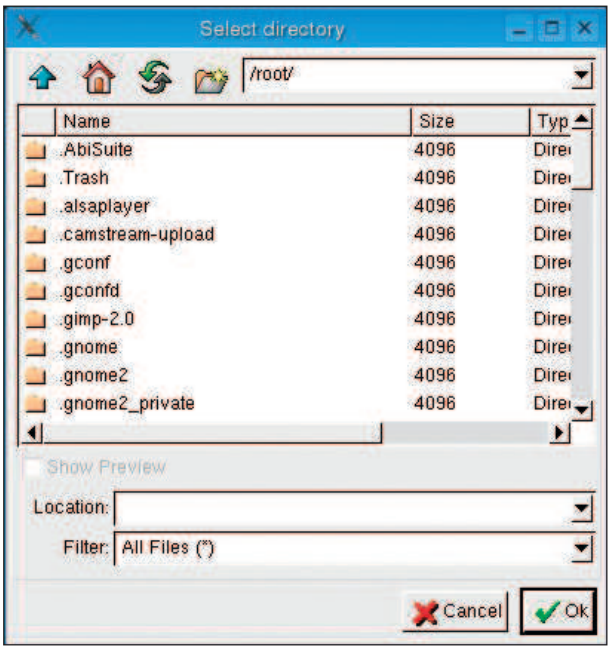
Az első, és legegyszerűbb az eldöntendő kérdések megjelenítésére alkalmas IGEN/NEM párbeszédablak. Az ablakot elővarázsolhatjuk egy egyszerű `fl_ask()` hívással. A függvénynek csak a kérdés szövegét kell megadnunk paraméterként, majd a visszatérési érték alapján eldönthetjük, hogy mit választott a felhasználó. Amennyiben ez az érték 1, akkor a felhasználó vagy az ENTER billentyűvel vagy a ***Yes*** gombra kattintva válaszolt, míg 0 visszatérési érték esetén a ***No*** gombot alkalmazta a felhasználó. Ez utóbbi eset megfelel az 'ESC' billentyű lenyomásának. Az *1. ábrán* látható egy ilyen párbeszédablak.

A következő ilyen párbeszédablak az `fl_alert()` függvény segítségével érhető el és egy egyszerű figyelmeztető üzenet megjelenítésére alkalmas. A függvény paramétere az üzenet szövege.

Több lehetőséget is felajánlhatunk a kedves felhasználóknak a programok használata során, mégpedig az `fl_choice()` függvény segítségével. A függvény első paramétere a kérdés szövege, majd a további három paraméter határozza meg a választások szövegét. Így összesen három lehetőséget ajánlhatunk fel a felhasználóknak. Az alapértelmezett választás a középső gomb, melyet szintén az ENTER billentyűvel választhatunk ki. Ebben az esetben a függvény visszatérési értéke 1 lesz. Az ESC billentyű hatására a visszatérési érték 0, és szintén kiválasztható a jobb oldali gomb segítségével. A bal oldali gomb lenyomásával a visszaadott érték 2 lesz. A *2. ábrán* láthatjuk a többszörös választás megjelenítésére szolgáló ablak képét.

A következő egyszerű párbeszédablak arra szolgál, hogy a felhasználótól adatokat kérjünk be. Itt egyszerűen szöveges adatokat olvashatunk be, az `fl_input()` függvény segítségével. A függvény első paramétere a beviteli mező felirata, második paramétere az alapértelmezett érték. Amennyiben a felhasználó megszakítja a bevitelt, a visszatérési érték `NULL` lesz. A *3. ábrán* látható egy egyszerű adatbeviteli párbeszédablak.

Jelszavak bevitelére alkalmas az `fl_password()` függvény. Paraméterezése megegyezik az `fl_input()`-nál megadottakkal, de a felhasználó által leütött karakterek helyén * karaktereket jelenít meg az ***eFltk***.



5. ábra Könyvtárválasztó

Az `fl_message()` megegyezik a korábban bemutatott `fl_alert()` ablakkal, de ebben az esetben nem figyelmeztetést jelző felkiáltójelet, hanem egy információt jelző 'i' ikont láthatunk a párbeszédablakban.
A párbeszédablakok ikonját megváltoztathatjuk, ha az `fl_message_icon()` függvény által visszaadott Fl_Widget objektumban a képet kicsréljük egy általunk beolvasott képre. Ahogyan korábban már láttuk, erre szolgál az `Fl_Widget->image()` osztályfüggvény.
Természetesen az üzenetek betűkészletének megváltoztatására is van lehetőség, mégpedig az `fl_message_font()` függvénnyel. A függvény két paramétere a betűkészlet (`FL_HELVETICA`, `FL_HELVETICA_BOLD`, `FL_HELVETICA_ITALIC`, `FL_TIMES`, `FL_TIMES_BOLD`, `FL_TIMES_ITALIC`, `FL_COURIER`, `FL_COURIER_BOLD`, `FL_COURIER_ITALIC`) és a betűméret. Érdemes megjegyezni, hogy a betűkészlet és az ikonok megváltoztatása csak a változás után megjelenített párbeszédablakokra van hatással.
A párbeszédablakok sorában a leginkább összetett az állományok kiválasztására alkalmas állományválasztó ablak. Megjeleníthető az `fl_select_file()` függvény meghívásával, melynek három paramétert adhatunk át. Az első paraméter a kiindulási könyvtár, a második az állományok szűrésére alkalmas minta, a harmadik paraméter az ablak címe. A *4. ábrán* látható egy ilyen állományválasztó ablak, melyben éppen JPG képet keresünk a könyvtárakban.
Természetesen az eFltk készítői nem csupán egyes állományok kiválasztására szolgáló párbeszédablakot készítettek, hanem a meglévő eszközök segítségével több állományt is kijelölhetün. Erre szolgál az `fl_select_files()` függvény, melynek paraméterei megegyeznek az előbbiekben bemutatott `fl_select_file()` paramétereivel egyezik meg.
A függvény visszatérési értéke egy karakterlánc lista vagy NULL, amikor nem választunk ki egyetlen állományt sem. Amennyiben a felhasználói programban könyvtár kiválasztására van szükség, használhatjuk az `fl_select_dir()` függvényt. Első paramétere a kiindulási útvonal, a másodikban adhatjuk meg az ablak feliratát. Amennyiben a felhasználó kiválaszt egy könyvtárat a függvény visszatérési értéke ennek elérési útja lesz.
Mindegyik állomány- és könyvtár választó párbeszédablakról elmondható, hogy ha nem választunk ki semmit, vagyis a felhasználó az Esc billentyűvel vagy a '***Mégsem***' gomb használatával bezárja az ablakot, akkor az előbbi függvények NULL értékkel térnek vissza.
Az állományok kiválasztására szolgáló ablakok működését néhány előre definiált változóval befolyásolhatjuk.
Az `fc_initial_w` egész érték határozza meg, hogy milyen széles legyen a párbeszédablak, míg az `fc_initial_h` a magasságot befolyásolja. A képek előnézetének megjelenítését az `fc_initial_preview` logikai típusú változó megfelelő beállításával kapcsolhatjuk be vagy ki.
Most lássunk egy egyszerű példát az alkalmazásunk indítása előtti felhasználó azonosításra. Az előző rész alapján a `main()` függvényben jelenítjük meg az alkalmazás fő ablakát, azonban célszerű az ellenőrzést még az ablak megjelenítése előtt megtenni. A korábbi példában a főprogram így kezdődött.

```
int main (int argc, char **argv) {

  make_window();
  window->show();
}
```

Az ellenőrzés legyen a lehető legegyszerűbb, lássunk is hozzá. Első lépésben ellenőrizzük a felhasználónevet a `make_window()` előtti sorokban, majd amennyiben azt helyesen adja meg a felhasználó, következhet a jelszó bevitele.

```
const char* pass=NULL;
const char* azon=
   fl_input((const char*)"Kerem adja meg
   ➥a nevet:",(const char*)NULL);
if (!strcmp(azon, (const char
➥*)"rendszergazda")) {
   pass=fl_password((const char *)"A jelszot
   ➥is kerem:", "");
   if (strcmp(pass, (const char*)"jojelszo")) {
      fl_alert("Nem jo jelszo! Kilepes!");
      abort();
   }
} else {
   fl_alert("Nem jo azonosito! Kilepes!");
   abort();
}
```

Amint látható ismét nem túl bonyolult a megoldás, köszönhetően annak, hogy az ***eFltk***-ban a készítők nagy tapasztalattal rendelkező programozók, akik tisztában vannak egy grafikus elemkészlet készítése során felmerülő igényekkel.

Fábián Zoltán

Egy csipetnyi Swing és egy leheletnyi JFreeChart

Hogyan készítsünk mutatós felhasználói felületet pillanatok alatt Javaban? Tippek és trükkök a platformfüggetlenség jegyében.

A grafikus alkalmazások a számítógéppel most ismerkedők számára mindig barátságosabb eszközöknek bizonyulnak, mint a parancssoros programok. Éppen ezért, ha feltesszük, hogy a világegyenletet megoldó szoftverünknek nem kizárólag a legszakavatottabb kezek alatt is jó szolgálatot kell tennie, érdemes felruháznunk valamilyen ***grafikus felhasználói felülettel (GUI, graphical user interface)***. Nézzünk körül, milyen fejlesztési eszközök állnak rendelkezésünkre!

A ***Linuxvilág*** hűséges olvasói számos megoldást láttak már a problémára, mind más írók, mind jómagam tollsából. A lap hasábjain jelentek meg írások a ***Tk*** eszközkészletről, és ennek több programozói környezetben előforduló változatáról, így a ***Perl/Tk***-ról és a ***Tcl/Tk***-ról. Olvashattunk a ***Qt*** könyvtárról és a ***GTK***-ról is, melyek ***C/C++***-ból felhasználható függvénykönyvtárak. Természetesen a felsorolás a teljesség igénye nélkül készült.

Ezen eszközkészletek egy része szkriptekből felhasználható. Ezek egyik problémája, hogy nem előfordíthatók, így a futási idő ezeknél a lehető legnagyobb. Léteznek kerülőutak ennek kiküszöbölésére, de ezek a módszerek még nem kiforrottak. Az előfordított nyelvekkel az a gond, hogy még a forráskód szintű hordozhatóság biztosítása is nagy körültekintést és komoly programozói gyakorlatot igényel. Létezik ezek után megoldás?

Nem tettem volna fel a költői kérdést, ha nem igen volna a válasz. A megoldást pedig a ***Java*** programnyelv jelenti. Látom lelki szemeim előtt, ahogy most százan és ezren húzzák fel egyszerre az orrukat. Be kell látni, a Java nem egy paripa. Igen, lassabb a ***C++***-nál, és sokkal lassabb a ***C***-nél. Cserébe viszont bájtkód szinten hordozható a különböző operációs rendszerek között, és egy nagyon kényelmesen használható, gazdag függvénykönyvtárral bíró objektum-orientált környezet. Továbbá az elvakult szkripthívők is beláthatják, hogy a Java gyorsabb egy átlagos szkriptnyelvnél.

Fontos, hogy belássuk, a cél határozza meg az eszközt, és ennek mindig így kell lennie. Ha követelmény a hordozhatóság, kézenfekvő megoldást jelenthet a ***Java***. A bájtkóddá történő fordítás természetesen nem egyetemes válasz minden kérdésre, így sebességben sohasem fogja utolérni a natív kódot. Ezzel szemben a Java használatával az, hogy mely operációs rendszerek képesek futtatni alkalmazásunkat, kizárólag annak a függvénye, hogy elkészült-e már a ***Java Virtuális Gép (JVM, Java Virtual Machine)*** az adott platformra.

Nézzük tehát, mit kínál a ***Java***. Kezdetben grafikus felhasználói felületek létrehozására az ***AWT (Abstract Windowing Toolkit)*** állt rendelkezésünkre. Az eszközkészlet absztrakt jellegét az adja, hogy kizárólag olyan elemek találhatók meg benne, amelyek minden, ***JVM***-et futtató operációs rendszer grafikus felületén előfordulnak, és ezek megjelenítéséért a gazda rendszer felelős. Vagyis egy nyomógomb ***Windows*** alatt ***windowsosan***, ***Unix*** alatt „motifosan” néz ki. Ez azt is jelenti sajnos, hogy a grafikus felületek által biztosított grafikus elemek legszűkebb metszete érhető csak el a programozó számára.

Ezt követte a ***Swing*** megjelenése. A ***Swing*** egy olyan részhalmaza a ***Java*** függvénykönyvtárnak, amelyet teljesen ***Javaban*** írtak. Egyáltalán nem használja a gazda rendszer nyújtotta elemeket, ami azt jelenti, hogy mindent saját maga rajzol ki. Ebből eredően a ***Swinget*** használó Java alkalmazások minden rendszer alatt ugyanúgy néznek ki. A kinézet stílusok révén befolyásolható, ez az úgynevezett ***„Look&Feel”***. Ezzel jelen írás nem foglalkozik, alkalmazásunk az alapértelmezett ***Metal*** kinézettel bír.

Aki használta már az ***AWT***-t, könnyen megbarátkozik a ***Swinggel*** is, mivel a régi elemek új neve csak annyiban változik, hogy ott áll egy „J” betű az elején. Vagyis például `Frame` helyett `JFrame`-et fogunk használni. A névrokonság nem önkényes, nagyon sok elem régi metódusai nem változtak, és ezért ugyanúgy használhatók, mint korábban. Viszont néhány helyen alapvető változások történtek, így a `JFrame` esetében is. Nem hívható meg többek között egy `JFrame`-re az `add()` metódus egy elem hozzáadásához, mint a régi `Frame` esetében. Ezért mindig legyen nyitva egy böngészőablakban a ***Java API*** programozás közben!

Jelen írás keretében egy menüvel rendelkező grafikus alkalmazást fogunk létrehozni. A menüből modulokat lehet kiválasztani, és mindig a kiválasztott modul jelenik meg az ablakban. Ezért a főprogram mellett minden modul önálló osztályt fog képviselni. Lesz egy olyan modulunk, amely egy diagrammot rajzol ki, ráadásul dinamikusan változó adatsorral. Vágjunk is rögtön bele, mert elég sok dolgunk van!

A főprogram

```
// Monitor.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Monitor implements ActionListener {

    private JMenuItem kilepes;
    private JCheckBoxMenuItem altalanos, memoria,
    ➥nevjegy;
    private Container ablakTartalom;
    private CardLayout elrendezesKezelo;

    private static final String ALTALANOS =
    ➥"Általános információk";
    private static final String MEMORIA = "Memória
    ➥használat";
    private static final String NEVJEGY = "Névjegy";

    public Monitor() {

      JFrame ablak = new JFrame("Monitor");

      JMenuBar menuSor = new JMenuBar();
      JMenu fajlMenu = new JMenu("Fájl");
      JMenu modulMenu = new JMenu("Modul");
      ButtonGroup modulCsoport = new ButtonGroup();
      altalanos = new JCheckBoxMenuItem(ALTALANOS,
      ➥true);

altalanos.setAccelerator(KeyStroke.getKeyStroke
➥("control A"));
      altalanos.addActionListener(this);
      modulCsoport.add(altalanos);
      modulMenu.add(altalanos);
      memoria = new JCheckBoxMenuItem(MEMORIA);
      memoria.setAccelerator(KeyStroke.getKeyStroke
      ➥("control M"));
      memoria.addActionListener(this);
      modulCsoport.add(memoria);
      modulMenu.add(memoria);
      nevjegy = new JCheckBoxMenuItem(NEVJEGY);
      nevjegy.setAccelerator(KeyStroke.getKeyStroke
      ➥("control N"));
      nevjegy.addActionListener(this);
      modulCsoport.add(nevjegy);
      modulMenu.add(nevjegy);
      fajlMenu.add(modulMenu);
      fajlMenu.addSeparator();
      kilepes = new JMenuItem("Kilépés");
      kilepes.setAccelerator(KeyStroke.getKeyStroke
      ➥("control K"));
      kilepes.addActionListener(this);
      fajlMenu.add(kilepes);
      menuSor.add(fajlMenu);
      ablak.setJMenuBar (menuSor);

      ablakTartalom = ablak.getContentPane();
      elrendezesKezelo = new CardLayout();
      ablakTartalom.setLayout(elrendezesKezelo);
      ablakTartalom.add(new MonitorAltalanos(),
      ➥ALTALANOS);
      ablakTartalom.add(new MonitorMemoria(),
      ➥MEMORIA);
      ablakTartalom.add(new MonitorNevjegy(),
      ➥NEVJEGY);

      ablak.setDefaultCloseOperation
      ➥(JFrame.EXIT_ON_CLOSE);
      ablak.setBounds(100, 100, 640, 480);
      ablak.setVisible(true);

    }

    public void actionPerformed(ActionEvent esemeny) {

      Object forras = esemeny.getSource();

      if (forras == altalanos) {
          elrendezesKezelo.show(ablakTartalom,
          ➥ALTALANOS);
      } else if (forras == memoria) {
          elrendezesKezelo.show(ablakTartalom,
          ➥MEMORIA);
      } else if (forras == nevjegy) {
          elrendezesKezelo.show(ablakTartalom,
          ➥NEVJEGY);
      } else if (forras == kilepes) {
          System.exit(0);
      }

    }

    public static void main(String[] args) {

      new Monitor();

    }

}
```

Bár kizárólag a ***Swing*** által biztosított eszközkészletből dolgozunk, vannak osztályok, amelyek kizárólag a jó öreg ***AWT***-ből érhetők el. Így az eseménykezeléshez használt `ActionListener` interfész, amely a `java.awt.event` csomag része, vagy a `CardLayout` nevű, elrendezéskezelőt osztály, melyet a `java.awt` csomagban találunk. Ezért importáljuk a névtérbe a `java.awt`, a `java.awt.event`, és a `javax.swing` csomagok összes elemét.
`Monitor` nevű osztályunk megvalósítja az `ActionListener` interfészt, mivel ennek lesz a felelőssége a menüben egy elemen történő kattintás eseményének a kezelése. Három publikus tagfüggvénye van: a konstruktor, amely a grafikus felületet építi fel, az `actionPerformed()`, amely az előbb említett eseményeket dolgozza fel, illetve egy statikus `main()`, melynek nincs más feladata, mint hogy létrehozzon egy példányt az alkalmazásból. Az osztály privát tagváltozóit első használatukkor, a konstruktor bemutatásánál részletezem.
A konstruktor feladata a grafikus felület felépítése. Első sorában létrehoz egy `JFrame` objektumot, ez fogja képviselni az ablakunkat. Az átadott paraméter az ablak címe.

Ezt követi egy nagyobb rész, amiben a menüt hozzuk létre. Egy igen egyszerű menüsort építünk fel, egyetlen `Fájl` menüvel, amiben van egy `Modul` lenyíló menü valamint egy `Kilépés` menüpont. A `Modul` menüből további menüpontok választhatók ki.

Egy ***Swing*** menü a következőképpen fest: egy `JMenuBar` objektum képviseli a teljes menüsort, ennek létrehozása után kell hozzárendelni az ablakhoz annak `setJMenuBar()` metódusával. A menüsor menükből áll, melyeket egy-egy `JMenu` objektum képvisel. Egy menü további menüket is tartalmazhat, ekkor ezek lenyíló menükként jelennek meg. Az egyes menüpontokat `JMenuItem` objektumok jelentik. A `JMenuItem` ősosztálya az `AbstractButton`, emiatt van `addActionListener()` metódusa, és eseménykezelése egy közönséges nyomógombhoz hasonló.

Alkalmazásunkban a Modul almenü jelölőnégyzettel ellátott menüpontokból áll, melyeket a `JCheckBoxMenuItem` osztály valósít meg. Ez a `JMenuItem` osztályból származik. A menüpontok közül mindig pontosan egy aktív, hiszen a felhasználó az ablakban mindig egy modult lát, a menü az ezek közti váltást valósítja meg. Ezt a `ButtonGroup` osztály használatával értem el. Egy `ButtonGroup` objektum egy gombcsoportot képvisel, melyhez az add() metódussal lehet hozzávenni egy nyomógombot. Az objektum felelőssége, hogy a csoportba tartozó gombok közül mindig csak az egyik legyen kiválasztva.

Nem menü a menü gyorsbillentyűk nélkül. A `JMenuItem` osztály, s így a `JCheckBoxMenuItem` osztály is, rendelkezik egy `setAccelerator()` metódussal. Ez egy `KeyStroke` objektumot vár paraméterként és a megadott billentyűkombinációt hozzárendeli ahhoz a menüponthoz, amelyre meghívták. A `KeyStroke` objektum létrehozása jelen esetben az osztály `getKeyStroke` statikus metódusának meghívásával történik, amely a paraméterében szövegesen megfogalmazott billentyűkombinációból készít egy `KeyStroke` objektumot.

Lássuk tehát sorról sorra, mi történik a menü felépítésekor! Elsőként készítünk egy menüsort, melyet `menuSor`-nak nevezünk el. Ezután létrehozunk két menüt, az egyiket `fajlMenu`-nek, a másikat `modulMenu`-nek nevezzük el. Ne felejtsük el, egy menün belül található lenyíló menü is ugyanolyan, mint az őt tartalmazó. Ezután létrehozunk egy gombcsoportot, és mivel a Modul menü elemeit fogjuk majd egybe ezzel, `modulCsoport`-nak nevezzük el.

Ezután három, nagyon hasonló lépéssorozat következik, a `Modul` almenü három menüpontjának létrehozásához. Előbb létrehozzuk a menüpontot, mint egy `JCheckBoxMenuItem` objektumot. Az első konstruktorában jelezzük, hogy már ki van választva. A menüpontokhoz beállítunk egy gyorsbillentyűt. Ezután jelezzük, hogy a menüpont kiválasztásához tartozó eseménykezelőt jelen osztály tartalmazza. Hozzávesszük a gombcsoporthoz a menüpontot. Végül hozzáadjuk a `modulMenu`-höz. Miután mindhárom `JCheckBoxMenuItem` elkészült, s ezzel feltöltöttük a `modulMenu`-t, a `fajlMenu` objektum `add()` metódusát meghívjuk a `modulMenu`-vel és így hozzávesszük a menühöz. Ezután az `addSeparator()` tagfüggvény segítségével beteszünk egy elválasztó vonalat a Modul menü alá. Végül elkészítjük a Kilépés menüpontot, amely mindössze annyiban különbözik a Modul menü elemeitől, hogy nem rendelkezik jelölőnégyzettel, és így nem is tartozik a gombcsoportba.

A menü létrehozását követő rövidebb rész az ablak tartalmát építi fel. Fontos különbség az ***AWT***-ben található `Frame`-hez képest, hogy a `JFrame`-re nem hívható meg közvetlenül az `add()` metódus. Először le kell kérdezni az ablaktartalmat képviselő `Container` objektumot és ennek lehet használni az add() tagfüggvényét. Először tehát ezt kérdezzük le. Majd létrehozunk egy `CardLayout` elrendezéskezelőt. Ennek az a különlegessége, hogy a tartalmazott objektumok közül mindig csak egyet mutat, a többit háttérben tartja. Neve is onnan ered, hogy hasonlítható egy kártyapaklihoz, melynek csak a legfelső eleme látható.

Az ablaktartalom elrendezéskezelőjét beállítjuk a létrehozott objektumra a `setLayout()` metódussal. Majd egyesével felvesszük a modulokat az `add()` metódussal, mindegyiknek azt a szöveges nevet adva, amely a menüben is szerepel. Tehát a nagybetűs állandók, az `ALTALANOS`, a `MEMORIA`, és a `NEVJEGY` nem csupán a `JCheckBoxMenuItem` létrehozásakor játszottak szerepet, mint a menüpontok feliratai, hanem a hozzájuk tartozó kártyalap nevei is egyben. Ezzel a névvel lehet később hivatkozni arra a kártyalapra, amelyet felülre kívánunk tenni.

Minden modulunk a `JPanel` osztályból ered, ezért adhattuk őket hozzá az ablaktartalomhoz ilyen egyszerűen. A konstruktor végén még néhány egyszerűbb beállítást találunk. Meghatározzuk az alapértelmezett műveletet akkor, ha az ablak bezárását kezdeményezi a felhasználó az ablakkezelőn keresztül. Beállítjuk az ablak méreteit, pozícióját, és végül láthatóvá tesszük.

Az `actionPerformed()` metódus a menüpontok valamelyikén történő kattintáskor fut le. Előbb meghatározzuk az esemény forrását, majd ettől függően elvégezzük a szükséges műveletet. Megjelenítjük a kívánt modult, vagy kilépünk a programból. Látható, hogy a menüpontok feliratával azonos nevű oldalak segítségével ez milyen egyszerűen megvalósítható.

Lássuk most az „Általános információk modult”!. Ez táblázatos formában mutatja be a rendszertulajdonságokat leíró változókat.

```
import java.util.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;

public class MonitorAltalanos extends JPanel {

    private class AdatModell extends
    ➥AbstractTableModel {

      private Vector adatok;

      public AdatModell() {
          Properties rendszerTulajdonsagok =
          ➥System.getProperties();
          adatok = new
```

```
        ➥Vector(rendszerTulajdonsagok.size());
        Enumeration kulcsok =
        ➥rendszerTulajdonsagok.keys();
        while (kulcsok.hasMoreElements()) {
      Object kulcs =
      ➥kulcsok.nextElement();
      Vector adat = new Vector(2);
      adat.add(kulcs);
      adat.add(rendszerTulajdonsagok.get(kulcs));
      adatok.add(adat);
        }
    }

    public int getRowCount() { return
    ➥adatok.size(); }

    public int getColumnCount() { return 2; }

    public Object getValueAt(int sor, int oszlop) {
        Vector sorVektor = (Vector)
        ➥adatok.get(sor);
        return sorVektor.get(oszlop);
    }

    public String getColumnName(int sor) {
        if (sor == 0) return "Kulcs";
        return "Érték";
    }

  }

  public MonitorAltalanos() {

    setLayout(new BorderLayout());
    JTable tablazat = new JTable(new
    ➥AdatModell());
    JScrollPane gorgethetoNezet = new
    ➥JScrollPane(tablazat);
    add(gorgethetoNezet, BorderLayout.CENTER);

  }

}
```

A modul szép példája a `JTable` osztály használatának. A feladat egy görgethető táblázat létrehozása. Ezt a mondatot a konstruktorban írjuk le. Beállítjuk az elrendezéskezelőt egy `BorderLayout` objektumra. Ezután létrehozunk egy táblázatot egy adatmodellel. Az adatmodellt egy belső osztály írja le. Ezt követően megalkotjuk a táblázat görgethető nézetét. Végül a panel közepére betesszük ezt az elemet.
A belső osztály az adatmodellt írja le. Ezzel elkülönül maga a `JTable` és az adatokat tartalmazó objektum. Ez azért nagyon hasznos, mert így az adatok ábrázolása teljes mértékben ránk van bízva, olyan szerkezetet használunk, amilyet csak szeretnénk. Ebben az adatmodellben egy `Vector` objektumot veszünk igénybe, amely további `Vector`-okat tartalmaz. Ezen `Vector`-ok pedig két `String` elemből, egy kulcsból és egy értékből állnak.

Az `AdatModell` konstruktora a `System` osztály `getProperties()` metódusa segítségével lekérdezi a rendszertulajdonságokat. Sajnos ez közvetlenül nem használható fel adatforrásnak, mert a benne található elemek nem rendezhetők sorba. Ezért átírjuk az összes elemét a fentebb említett `Vector`-ba, ami már egyértelmű sorrendet jelent. A lekérdezés után tehát létrehozunk egy akkora `Vector`-t, ahány eleme van a `rendszerTulajdonsagok`-nak.
Ezt követően lekérdezzük a `rendszerTulajdonsagok` kulcsait, és egy ciklusban bejárjuk őket. Minden iterációban létrehozunk egy 2 hosszú `Vector`-t, amit megtöltünk adattal, és hozzáadjuk az adatok nevű `Vector`-unkhoz.
Miután az `AdatModell` osztály kiterjeszti az `AbstractTableModel`-t, néhány metódust biztosítanunk kell. Ezek: a `getRowCount()`, ami a sorok számát adja vissza, a `getColumnCount()`, ami az oszlopok számát adja meg, illetve a `getValueAt()`, ami egy konkrét elem értékével szolgál. A `getColumnName()` nem kötelezően megvalósítandó, de ha nem definiáljuk felül, az oszlopok címei A, B, C, stb. lesznek.
Lássuk most a „Memória használat” modult! Ez egy dinamikus diagrammon mutatja be a ***Java Virtuális Gép*** memóriahasználatát.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import org.jfree.chart.*;
import org.jfree.data.time.*;

public class MonitorMemoria extends JPanel
➥implements ActionListener {

    private Runtime kornyezet;
    private TimeSeries osszesMemoria,
    szabadMemoria;

    public MonitorMemoria() {

      kornyezet = Runtime.getRuntime();

      osszesMemoria = new TimeSeries("Összes
      ➥Memória", Millisecond.class);
      osszesMemoria.setHistoryCount(30000);
      szabadMemoria = new TimeSeries("Szabad
      ➥Memória", Millisecond.class);
      szabadMemoria.setHistoryCount(30000);
      TimeSeriesCollection adatSor = new
      ➥TimeSeriesCollection();
      adatSor.addSeries(osszesMemoria);
      adatSor.addSeries(szabadMemoria);

      JFreeChart diagramm =
      ➥ChartFactory.createTimeSeriesChart(
          "A Java Virtuális Gép memória
          ➥használata",
          "Idő",
          "Memória",
```

```
            adatSor,
            true, true, false
        );

        ChartPanel diagrammPanel = new
        ➥ChartPanel(diagramm);

        setLayout(new BorderLayout());
        add(diagrammPanel, BorderLayout.CENTER);

        Timer utemezo = new Timer(100, this);
        utemezo.start();

    }

    public void actionPerformed(ActionEvent
    ➥esemeny) {
        szabadMemoria.add(new Millisecond(),
        ➥kornyezet.freeMemory());
        osszesMemoria.add(new Millisecond(),
        ➥kornyezet.totalMemory());
    }

}
```

Ez a modul felhasználja a `JFreeChart` *Java* csomagot, melyet a ***http://www.jfree.org/*** oldalon keresztül lehet beszerezni. Használatához mindössze a ***jcommon-x.x.x.jar*** és a ***jfreechart-x.x.x.jar*** állományok elérési útvonalát kell betenni a `CLASSPATH` környezeti változóba, vagy paranccsorban mind a fordítónak, mind a futtatókörnyezetnek a `-cp` kapcsolóval átadni. A `JFreeChart` egy nagyon gazdag környezet mindenféle grafikon egyszerű előállításához és ráadásul ingyenes is.
Tehát miután importáltuk a névtérbe az `org.jfree.chart` és az `org.jfree.data.time` csomagok elemeit, nézzük meg, hogyan is működik ez az osztály. Megvalósítja az `ActionListener` interfészt, mivel az felhasznált memóriára vonatkozó adatok periodikus lekérdezéséhez a ***Swing*** `Timer` osztályát használjuk, és ez a megadott időközönként meghívandó eljárást egy olyan osztály képében várja, amely rendelkezik az `actionPerformed()` tagfüggvénnyel.
Az osztály egy konstruktorból és egy `actionPerformed()` metódusból áll. Vannak továbbá privát tagváltozói, ezek a `kornyezet`, az `osszesMemoria`, és a `szabadMemoria`.
A konstruktor lekérdezi a környezetet, létrehozza a diagrammot, beállítja és elindítja az időzítőt.
Az `actionPerformed()`, amely minden tizedmásodpercben lefut, lekérdezi a környezettől az összes- és a szabad memóriát, és ezeket az értékeket hozzáadja az `osszesMemoria` és a `szabadMemoria` adatforrásokhoz.
A konstruktor mindenek előtt beállítja a `kornyezet` változót a `Runtime` osztály `getRuntime()` metódusa alapján. Létrehozunk két `TimeSeries` objektumot, melyek az adatforrásokat fogják jelenteni a `osszesMemoria`, illetve a `szabadMemoria` változóknak. Mindkettőnek beállítjuk, hogy a 30 másodpercnél régebbi adatokat ne vegye figyelembe. Létrehozunk továbbá egy `adatSor` változót, melyhez hozzáadjuk a két adatforrást.

Ezután létrehozzuk a diagrammot, megadva a címét, a tengelyek címeit, az adatsort, és néhány apróságot. Készítünk a diagrammból egy panelt, beállítjuk az elrendezéskezelőt, és hozzáadjuk a panelt. Ezután készítünk egy ütemezőt, amely a megadott objektum `actionPerformed()` metódusát hívja meg tizedmásodpercenként, majd elindítjuk az ütemezőt.
Végezetül a teljesség kedvéért lássuk a névjegy panelt. Noha meg vagyok győződve, hogy ezek után egyetlen sora sem jelent újdonságot.

```
import java.awt.*;
import javax.swing.*;

public class MonitorNevjegy extends JPanel {

    public MonitorNevjegy() {

        final String sorTores =
        ➥System.getProperty("line.separator");

        setLayout(new BorderLayout());

        JTextArea szoveg = new JTextArea();
        szoveg.setEditable(false);
        szoveg.append("Monitor v0.1" + sorTores);
        szoveg.append("==========" + sorTores +
        ➥sorTores);
        szoveg.append("Java alkalmazás a Linuxvilág
        ➥olvasói számára");
        szoveg.append(sorTores + sorTores + "Fülöp
        ➥Balázs" + sorTores);
        szoveg.append("2005.");

        add(szoveg, BorderLayout.CENTER);

    }

}
```

Egy közönséges szövegdobozba írom bele a névjegy tartalmát. Ami talán érdekes lehet, az az, hogy a sortörést nem „\n"-el oldottam meg, bár a ***Java*** intelligenciája miatt úgy is működött volna bármely operációs rendszer alatt, hanem lekérdeztem a sortörést jelentő szövegfüzért a rendszertulajdonságok táblából.
Ebben az írásban elég sok kódot láthatsz, amit érdemes tanulmányozni. Ezt azért hangsúlyozom, mert kizárólag a leírásból nem fogsz rájönni mindenre. Feltétlenül töltsd le a ***Java API***-t a ***Sun*** oldaláról (***http://java.sun.com/***), ha még nem tetted meg, és próbálkozz! Ne csak azzal, amit itt látsz, hanem azzal is, amit kigondolsz! Sok örömet kívánok a ***Java***-hoz.

Fülöp Balázs (admin@guardware.com)
21 éves, imádja a Túró Rudit, a Debian Linuxot és a teheneket. Kedvenc írója Slawomir Mrožek. Leginkább a számítógépes hálózatok biztonsága érdekli. A BME VIK műszaki informatikus szak hallgatója.

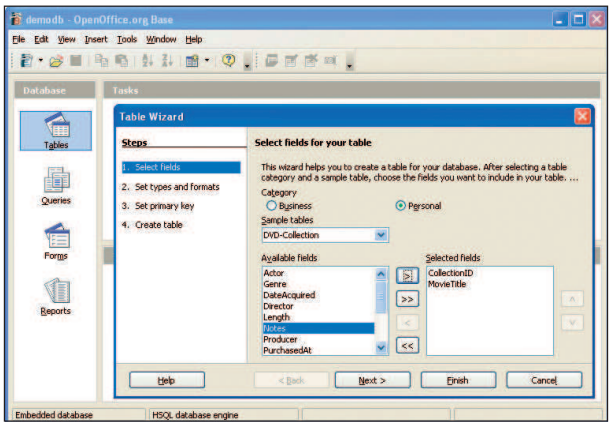
OpenOffice.org 2.0

Újabb mérföldkövek egy jól ismert irodai programcsomag életében.

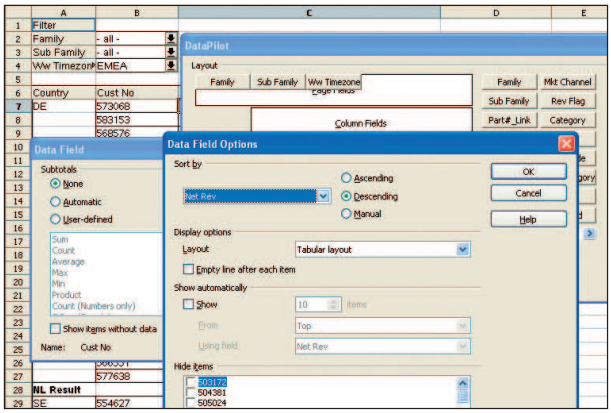
A *StarOffice*-ból sarjadt *OpenOffice* az egyik legsikeresebb nyílt forrású projekt. Sikerének kulcsa az ingyenesség mellet, hogy valóban használható dologról van szó: képes helyettesíteni a többi nagy irodai szoftvercsomagot, s mellette szól még, hogy az összes elterjedt platformon futtatható.
Talán pont ez a népszerűség ihlette a fejlesztőket arra, hogy teljes erőbedobással tökéletesítsék a jelenlegi 1.x-es változatot. A megfeszített munka eredménye most már egyértelműen körvonalazódik a 2.0-s változat formájában, amelyből ízelítőt is kaphatunk az 1.9.79-es számmal kiadott béta változaton keresztül. Bár az alkalmazás még nem végleges, az újítások listája már teljes, ezek szerepelnek is a már emlegetett béta programban. Új szolgáltatás tehát már nem várható a stabil csomag megjelenéséig, csupán a jelenlegiben előforduló hibák javítására számíthatunk. Ennek következtében már most nyugodtan végignézhetjük, hogy mivel lett több illetve kevesebb az új *OpenOffice.org*.

1. mérföldkő: telepítés

Nos, a telepítés némiképp megváltozott: az újítások listájában benne van, hogy az *OpenOffice* telepítője ezentúl a natív módot támogatja. Ez a gyakorlatban azt jelenti, hogy a rendszerünk csomagkezelő-rendszerére van bízva az előre összeállított csomagok telepítése, magyarul az új *OpenOffice*-nak igazából nincs is telepítője, csak csomagjai. Ez alapvetően nem rossz dolog: a baj csak annyi, hogy a fejlesztők építenek *MSI* illetve *RPM* csomagokat, és ezzel a lista végére is értünk. Ha tehát mi nem *Windowst*, illetve *Redhat*/*SuSE* terjesztést használunk, akkor bizonyos kellemetlen nehézségebe ütközhetünk: legrosszabb esetben például abba, hogy nem fogjuk tudni telepíteni a programot.
A telepítési útmutatóban az szerepel, hogy az *OpenOffice* a későbbiekben sem fogja támogatni a többi terjesztést, mivel azokba licenszelési okokból ez vagy az nem fér bele, így minden terjesztésnek magának kell összeállítania az OO „telepítőjét”.
Nem kell azért megijedni, jelenleg is létezik az összes nagyobb terjesztésen belül „saját”, natív módon telepíthető csomag, szóval ez nem lesz nagy változás: a terjesztések a továbbiakban is el fogják ezeket készíteni, mi pedig használjuk, csak úgy, mint eddig.
Ami engem zavar, hogy ezt „újításként” említik a fejlesztők, pedig igazából visszalépésről van szó. A régi telepítő lehetővé tette, hogy bármikor, a legfrissebb változatot letöltve, azt akármilyen környezetben telepíthessük. A terjesztés által rendelkezésemre bocsátott csomagokkal csak annyi a baj, hogy 1-2 hónappal is akár le vannak maradva a legfrissebb kiadástól (ennyivel később készülnek el az adott kiadásból a csomagok). Ezentúl tehát a terjesztés összeállítóira lesz bízva, hogy mikor használhatjuk a legfrissebb változatot. Jeles példája ennek, hogy én *Debian* alatt szerettem volna kipróbálni a béta változatot, és nem ment egyszerűen, szenvedni kellett érte. Először a egy nem hivatalos *apt*-forrás hozzáadása mellett döntöttem, ám az itt lévő,



1. ábra Az Openoffice.org új adatbázis-kezelő eszköze, a Base



2. ábra A Calc PivotTables nevű adatértelmező bővítménye akció közben

meglehetősen régi kiadás majdnem teljesen működésképtelen volt. Ez után következett az *RPM* csomagok *aliennel* történő átalakítása *DEB* csomagokká, majd az imádkozás, hogy minden gond nélkül települjön. Az imám meghallgatásra talált, mert az *OpenOffice* elindult. Ez így első hallásra nem is annyira bonyolult, de ha ehhez még hozzávesszük azt, hogy az új *OpenOffice* nem fér meg a régivel, akkor újra elszomorodhatunk: a régi 1.1-es változatot javasolt eltávolítani a rendszerből, mielőtt az újat telepítenénk, s ha esetleg a béta változat túlságosan gazdag hibákban, akkor a régi változatra történő visszaállás is ugyanilyen körülményes. Egyszóval: aki nem szeret szenvedni, annak bizony várnia kell – vagy átmegy *Windowsba*, és telepíti ott.

2. mérföldkő: új, szabványos fájlformátum

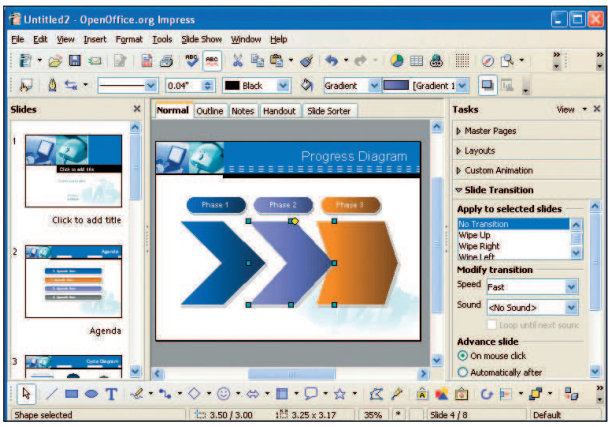
A 2.0-s változattól az *OpenOffice* az *XML* alapú, az *Európai Bizottság* ajánlásában is szereplő *OASIS OpenDocument* formátumot használja alapértelmezetten a dokumentumok tárolására. Ez egy terjesztés és implementáció-független formátum, amely a különböző dokumentum-kezelő programok közötti szabványos együttműködést hivatott elősegíteni. Ezt a formátumot használja egyébként a *Koffice* is. Ezzel együtt természetesen változnak a „kiterjesztések" is. A korábbi *.SX* kezdetű fájlvéget *.OD* kezdetűek váltják, így például a régi *SXW* végű szöveges dokumentum *ODT* végű lett, s igyekeztek következetesen kiosztani a fájlok neveit. Az *OASIS OpenDocument* formátumról a ➲ *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office* oldalon olvashatunk bővebben.

3. mérföldkő

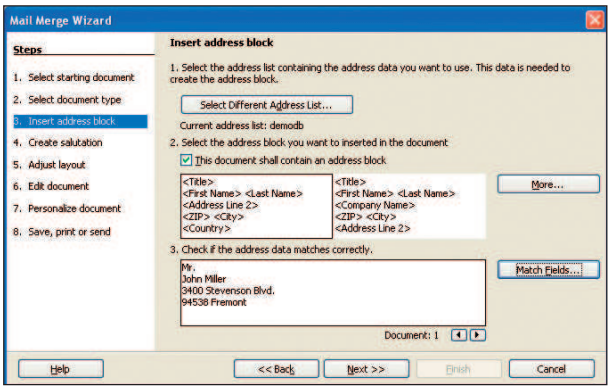
Előtérbe került az adatbázis-kezelő felület is: a többi alkalmazáshoz hasonlóan itt is a *Fájl menü*, *Új* menüpontjára kattintva lehet új adatbázist készíteni. A fejlesztők odafigyeltek a kezdő vagy épp az adatbázis-kezelésben, *SQL*-ben kevésbé jártas felhasználókra is. Egy új táblakészítő varázsló segítségével bárki könnyedén hozhat létre adatbázisokat, s utána űrlapokat, jelentéseket, stb készíthet, ahogy azt már megszokhattuk Microsoft Access esetében, amelynek versenytársa kíván lenni az irodai programcsomag *Base* nevű része. Ennek megfelelően tud is sokmindent: űrlapokat, jelentéseket, lekérdezéseket kezel az ott megszokott formában.
Ha már itt tartunk: hasonlóan az *Access*-hez, itt is adatbázis-dokumentumokat hozunk létre. Az új adatbázis-felület ugyanis a *Java* alapú *HSQLDB* adatbázis-kezelőre támaszkodik, amelynek következtében nincs szükség a háttérben futó adatbázis-kiszolgáló használatára, a program az adatokat (vele együtt az űrlapokat, jelentéseket, lekérdezéseket) egy *XML* fájlban tárolja.
A program ezen része is természetesen a már emlegetett *OpenDocument XML* formátumot használja.

4. mérföldkő: digitális aláírások használata

Lehetőség nyílik a dokumentumaink digitális aláírására, illetve a mások által aláírt dokumentumok hitelességének ellenőrzésére. Az ehhez szükséges tanúsítványokat az alkalmazás a rendszer által használt tanúsítvány-tárolóból szedi.



3. ábra Az Imress prezentáció-készítő felosztott felülete

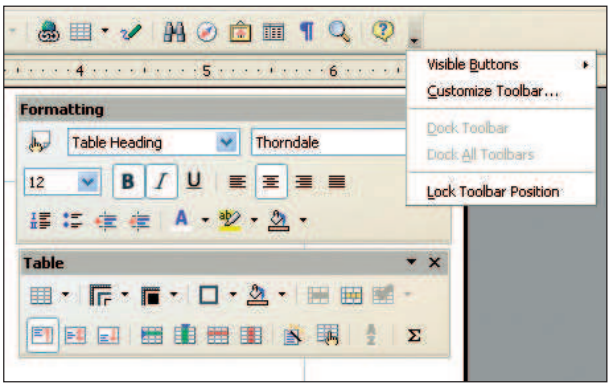


4. ábra A sokszínű körlevél-készítő varázsló

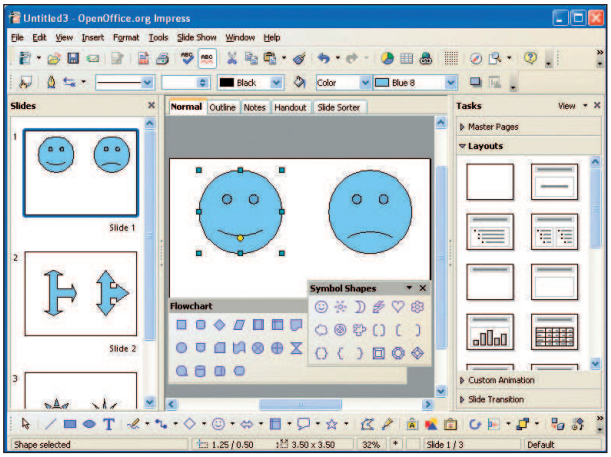
Az *OpenOffice Windows* alatt a *Microsoft* hitelesítési eljárását használja, egyéb operációs rendszerek alatt pedig a *Mozilla/NSS (Network Security Services)* függvénykönyvtárat. Ez utóbbi segítségével lehetővé válik rengeteg tanúsítvány és titkosítási forma használata (igazából minden, amit az *NSS* megenged). Nem szerepel viszont a leírásokban sehol sem utalás arra, hogy a linuxos körökben elterjedten használt *GnuPG* kulcsait és a bizalmi hálót tudja-e használni az *OpenOffice*. Az *OpenSSL*-lel természetesen képesek vagyunk például *X.509*-es tanúsítvánnyá konvertálni a szükséges adatokat (amelyet az *NNS* könyvtárral megetethetünk), de ez ugye mégsem egy kényelmes módszer, és elveszítjük a *PGP* hitelesítési mechanizmusát is – magyarán, hogy megbízhatunk-e egy idegen kulcsban, illetve ennek leellenőrzése meglehetősen körülményes.
Sajnos a béta változatnak ezen része defektes, így azt nem sikerült kipróbálnom, hogy a *PGP* kulcskarikámon szereplő „tanúsítványokat" be tudja-e olvasni, s használni a program.
Az *NSS* függvénykönyvtárról az alábbi címen tájékozódhat a téma iránt érdeklődő olvasó:
http://www.mozilla.org/projects/security/pki/nss/

5. mérföldkő: újítások a táblázatkezelőben

A *Microsoft PivotTable*-jének mintájára az *OpenOffice* annak idején kitalálta a *DataPilot* névre keresztelt adatelemző segédprogramot, amelynek segítségével akár külső adatforrásból , akár a táblázatban valahol szereplő értékekből von-



5. ábra A méretezhető, variálható úszó ablakok



6. ábra Az AutoShapes nevű vektorgrafikus ikonkészlet

hatunk le mindenféle következtetéseket. Külső adatforrás lehet az *OpenOffice.org*-ban regisztrált bármilyen (az OO által ismert) adatforrás. Egy ilyen létrehozásának legegyszerűbb módja, hogy a *Base* segítségével csinálunk egy adatbázist, s mentéskor rá fog kérdezni, hogy szeretnénk-e ezt az adatforrást regisztrálni. Ha az igenre kattintunk, a későbbiek folyamán elérhetjük a *DataPiloton* keresztül (már meglévő adatbázist annak megnyitását követően regisztrálhatunk).
A szolgáltatás természetesen már az előző változatban is szerepelt, az újítás az elérhető lehetőségek tárházának gazdagságában rejlik: létrehozhatunk csoportokat, szűrési feltételeket adhatunk a kijelölt adathalmazra, s mindenféle az adathalmazhoz relatív mutatókat számoltathatunk ki vele. Másik fontos fejlesztés, hogy kiterjesztették a tábla sorainak méretét: mostantól az *Excel*-hez hasonlóan 65536 (2^16) sora lehet a *Calcban* létrehozott tábláknak is. Ennek leginkább kompatibilitási okai vannak: hogy meg tudjuk nyitni a nagy *Excel* fájlokat is.

6. mérföldő: Az Impress újításai

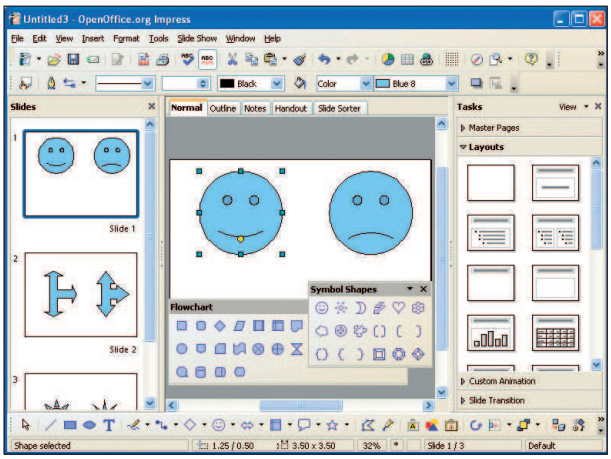
Bár én már eddig is egy igen használható eszköznek tartottam az *OpenOffice* prezentáció-készítő programját, a mostani fejlesztések mégis nagyot löknek rajta. Egyik ilyen újítás a több részre felosztott prezentáció-szerkesztő felület, amelynek köszönhetően minden fontos funkció megtalálható közvetlenül a képernyőn, nem kell érte különböző menükben mászkálnunk. A fejlesztők azt írják, hogy aki dolgozott már *PowerPointtal*, bizonyára nagyon fogja értékelni ezt az új felületet. Megjegyzem: tényleg jó, bár nekem az előző pont az egyszerűsége miatt tetszett nagyon.
Rengeteg új oldalátmeneti és animációs hatással is felvértezték az alkalmazást, csak hogy még jobban hasonlítson a konkurenciára, és hogy minél pontosabban tudja megjeleníteni a már említett versenytárs segítségével készített prezentációkat – amivel az előzőekben voltak gondok (pontatlan megjelenítés, szétcsúszás, stb).
S ami nem látszik: nem csak a hatások gazdagodtak, de a háttérben teljesen kicserélték oldalátmenetek kezeléséért felelős motort, s ezen túlmenően az egész prezentációkészítő motor is teljesen megújult.

7. mérföldkő: a Writer és lehetőségei

Bár a táblázatokról már beszéltünk a *Calc* kapcsán, most essen szó egy másik érdekességről: szinten főként kompatibilitási okokból megoldották az egymásba ágyazott táblázatokat a *Writerben*. Mostantól táblázat egy cellája is tartalmazhat másik táblázatot. Közben fejlődött a táblázatok tudása is: tudunk például a cellákban elhelyezett adatokra számozást alkalmazni, vagy felsorolásként kezelni azokat. (Természetesen egy cellában csak egy érték szerepel, azaz a felsorolás és számozás funkció kilát a cella határain túlra).
Nem tűnik nagy újításnak, hogy megváltozott a szavak számára vonatkozó statisztika, s most már lehetőségünk van egy kijelölésre is lekérni a szavak és karakterek számát, mégis azt kell mondjam, hogy már nagyon vártam. Régen ezt csak úgy lehetett, ha a kijelölt szöveget vágólapra másolva beillesztettem egy üres dokumentumba, és ott kértem szavak száma statisztikát – s pechemre, erre gyakran szükségem volt.
Megújult a ***körlevél funkció (Mail Merge)*** is. Nem csak a levélküldés paramétereinek sokkal jobb beállítási lehetőségét kaptuk, de a kezdő felhasználók számára készült egy varázsló, amely egész ügyesen végigvezeti a felhasználót címjegyzék kiválasztásától egészen a dokumentum mentéséig tartó nem túl egyszerű folyamaton. A *JavaMail API* segítségével nem csak körlevél-dokumentumokat készíthetünk, de közvetlenül e-mailben is elküldhetjük a körlevelünket, megúszva egy sor felesleges munkát.

8. mérföldkő: Kinézet

Mostantól az ablakkezelő-rendszerbe illeszkedő kinézet együtt jön a programmal. Régebben is voltak hasonló bővítmények, amelyeket telepítve az *OpenOffice* elkezdte az ablakkezelőnk által rendelkezésére bocsátott elemkészletet használni. Ez nem csak azért lényeges, mert egységes a felhasználói környezet, hanem azért is, mert számomra például a GTK-s elemkészlet használata sebességnövekedést és pontosabb viselkedést is eredményezett.
A másik újdonság, hogy megoldották az eszköztárak lebegtetésének problémáját (úszó eszköztárak). Ez azt jelenti, hogy szabadon húzgálhatjuk őket a képernyő széléről, s a már jól ismert kis ablakokban a monitor bármelyik területén megjeleníthetők, ahol épp nem zavarnak a munkában. Nem csak a helyváltoztatás jobb, de az ablakok átméretezhetők, s egyszerűsödött az elemek testreszabása is.



7. ábra Így néz ki az XForms támogatás az OpenOffice-ban

9. mérföldkő: Egyéb újdonságok

A fejlesztők szívügyüknek érezték, hogy a *Corel WordPerfect* szövegszerkesztője által létrehozott fájlokat is tudjuk nyitni az *OpenOffice*-ba importálni illetve onnan exportálni. Ehhez az nyílt forrású közösség által fejlesztett *WordPerfect* szűrőt használják. Továbbfejlesztették a *Lotus 1-2-3* szűrőt is, amely most már a 9.7-es változatig képes megnyitni a *Lotus* fájlokat.

Az 1.1-es változatban bemutatkozott a *PDF* export, kényelmesebbé téve az addigi *PostScript* formátumba nyomtatást, majd az abból *PDF*-be konvertálást (Sőt, *Windows* alatt lehetővé tette, hogy egyáltalán meg tudjuk ezt csinálni egyéb segédprogramok nélkül). A 2.0-s változat továbbfejlesztette a *PDF*-generátor képességeit: megadhatjuk a beágyazott képek tömörítésének mértékét, a kép felbontását, s az ígéret szerint ez már jól kezeli a hiperlinkeket és az előnézeti oldalképeket.

Ismét csak elsődlegesen kompatibilitási okokból, az *OpenOffice* megteremtette a *CustomShapes* bővítményt, amely a *Microsoft AutoShapes* megoldásának itteni megfelelője. Ezek „apró” vektorgrafikus „ikonok”, amelyeket a *Draw* programban használhatunk. A legfőbb vívmány mégis az, hogy az *AutoShapes* „ikonokat” jól beolvassa és meg is jeleníti a *Draw*, tehát megnyithatjuk az *AutoShapes*-t használó, *MS Office*-szal készült dokumentumokat.

Szintén a nagy riválissal történő együttműködés elősegítése érdekében az *OpenOffice*-ból megnyithatók a jelszóval védett *MS Office* fájlok, természetesen csak abban az esetben, ha a felhasználó tudja a jelszót a dokumentumhoz. A megnyitás során automatikusan felpattan a jelszót bekérő ablak, tehát pontosan ugyanúgy működik, mint az *MS Office* esetében.

S ha már a globális kompatibilitási újításoknál tartunk: alapértelmezetten része az új változatnak a *Microsoft XML* formátumainak, a *Spreadsheetml* és a *Wordml* fájljainak írása és olvasása. Mint tudjuk, az *MS Office*-ban importálhatunk, exportálhatunk *XML* formátumba dokumentumokat, amelyeket egy *XSLT (XML Stylesheet Translator)* „szűrő” alakít át a megfelelő formába a dokumentum újbóli megnyitás során. Mostantól ezek a műveletek *OpenOffice*-ból is elvégezhetők az Eszközök menü *XML* szűrő beállítások menüponton keresztül. Érdemes próbálkozni, hogy a helytelenül megnyitott dokumentumot először exportáljuk *MS Office* alatt *XML* formába, majd ezt az *XML*-t próbáljuk meg aztán *OpenOffice*-ból megnyitni.

A *Writerrel* lehetőségünk nyílik a *W3C XForms* szabványának megfelelő űrlapok készítésére, amelynek segítségével programozási nélkül is gyerekjáték egyszerű logikát csempészni az űrlap-kitöltés folyamatába. Ennek egyik nagy előnye, hogy olyan űrlapok jönnek létre, amelyek szabványosak, tehát más alkalmazások is használhatják (nincs benne *VisualBasic*, vagy egyéb gyártófüggő megoldás), így egyik objektumból a másikba könnyedén átvihető.

Az *XForms* egyébként egy új üdvöske, a jelenlegi *HTML/XHTML* űrlapokat hivatott hosszú távon helyettesíteni, lecserélni. Egy *XML* alapú megoldásról van szó, amely a tartalmat és megjelenést külön kezeli, és az űrlaphoz rendelhető *XML* adatszerkezetek használatával teszi kényelmessé és teljessé az űrlapok használatát.

A témában „mélyen érintettek” a *http://www.w3.org/MarkUp/Forms/* címen olvashatnak bővebben róla.

Összegzés

Az itt szereplő újítások listája természetesen a teljesség igénye nélkül, fontossági sorrendben készült. Számtalan egyéb apró változtatás van még, amely számunkra nem annyira érdekes. Ilyenek például, hogy bizonyos menüpontok máshová kerültek, néhány művelet során alapértelmezetté váltak olyan beállítási lehetőségek, amiket régebben külön ki kellett választani, vagy épp más lett a betűk igazításának módja, és még sorolhatnám. Alapjaiban igaz ezekre a változtatásokra, hogy a következetesség és kompatibilitás jegyében történtek.

Hasonlóan igaz ez a fenti mérföldkövekre is, bár ott nem elhanyagolható az újítási szándék sem. Rengeteg új funkció áll rendelkezésünkre, amelyek nagy része rendkívül hasznos. Más megoldások inkább csak abból a szempontból fontosak, hogy közeledtek a *MS Office* által kínált lehetőségekhez, amelynek itt valóban nem az utánzás volt az elsődleges célja, hisz a felhasználók igen jelentős része a szolgáltatások felét sem használja, még az *MS Office*-ban sem. Ezek tényleg azért kellettek, hogy jobb együttműködést biztosítsanak a konkurencia megoldásaival. Abból is látszik, hogy nem a konkurencia utolérése a cél, hogy az *OpenOffice* néhány szolgáltatásával egyenesen előzni próbál. Ilyen például a beépített digitális aláírás kezelő, vagy a beépített *PDF*-készítő.

Alapvetően jól eltalálták a fejlesztéseket, leszámítva talán a telepítés körüli problémákat. S ha már ilyen jól idomult a versenytársakhoz, őszintén remélem, hogy méltó ellenfele lesz azoknak a mindennapos felhasználás során.

Az újítások, változtatások teljes és részletes listája megtalálható a következő webhelyen:
http://marketing.openoffice.org/2.0/featureguide.html
A béta változatot
a *http://download.openoffice.org/2.0beta/index.html*
címről tölthetjük le.

Komáromi Zoltán

Hálózatok (17. rész)

Forgalomirányítás mozgó gépek esetén, adatszóró, többesküldéses forgalomirányítás, torlódásvédelem

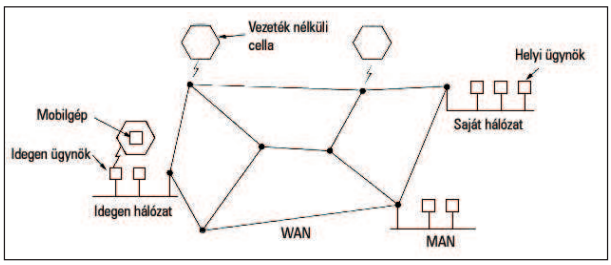
A sorozatnak ebben a részében a mozgó gépekkel kapcsolatos problémákat feszegetjük, majd megnézzük, milyen módszerek vannak arra, hogy egyszerre ne csak egy hoszt számára küldhessünk csomagokat. Ezenkívül szó lesz még a hálózatok legnagyobb mumusaként számon tartott jelenségről is: a torlódásról.

Forgalomirányítás és vándorló gépek

Eddig olyan forgalomirányítási mechanizmusokat tárgyaltunk, amelyek feltételezték azt, hogy a gépek „otthonülő életmódot" folytatnak, azaz nem vándorolnak szerte a világban. A mai mobilizálódó világban azonban ez elképzelhetetlen, hiszen egyre több felhasználó érzi magát kellemetlenül, ha a buszon utazva mobiltelefonjának segítségével nem nézhetné meg e-mail-jét, vagy esetleg nem léphetne be SSH-val az otthoni számítógépére.

Az *1. ábrán* egy nagy kiterjedésű hálózatot láthatunk. Tegyük fel, hogy ez a hálózat az egész világot behálózza, vagy legalábbis azt a területet, ahol a gépek vándorolhatnak. Ezt a hálózatot sokféle felhasználó használja. Vannak olyanok, akik a hálózathoz réz, illetve optikai kábel segítségével csatlakoznak, és sohasem hagyják el az otthonukat. Olyan felhasználó is akad, aki ugyan mindenfelé bolyong a világban, mégis csak úgy használja a hálózatot, hogy előbb fizikailag kapcsolódik hozzá (azaz a hálózaton tartózkodás ideje alatt nem változtatja a helyzetét). És persze vannak az „örök utazók" is, akik úgy szeretnének a hálózaton ügyködni, hogy közben folyamatosan mozogásban vannak. A felhasználók utóbbi két csoportját együtt ***mozgó felhasználónak (mobile user)*** nevezzük. Nézzük, miként juttathatjuk célba a mozgó felhasználóknak szánt csomagokat.

Az alapötlet az, hogy minden felhasználó rendelkezik egy állandó lakcímmel. A feladat az, hogy ha a felhasználó nem tartózkodik otthon, akkor a neki szánt csomagokat célba juttassuk, bárhol is legyen a felhasználó az adott pillanatban. Ehhez persze elengedhetetlen, hogy valamiképp rátaláljunk. Ehhez a behálózandó területet fel kell osztanunk körzetekre, vagy valamilyen más egységekre. Fontos, hogy a körzetek egymással diszjunktak, azaz a felhasználó egyszerre csak egy körzetben lehet. Minden ilyen körzetben két ügynök dolgozik: a ***hazai (home agent)*** és az ***idegen ügynök (foreign agent)***. A hazai ügynök azokat a felhasználókat tartja nyílván, akiknek az állandó lakhelyük a körzetében van, viszont jelenleg nem tartózkodnak otthon. Az idegen ügynök azokkal a felhasználókkal foglalkozik, akik ugyan máshol laknak, de jelenleg az ügynök körzetében tartózkodnak. Amikor egy felhasználó új körzetbe lép, akkor mindig fel kell vennie a kapcsolatot a helyi idegen ügynökkel. Először tehát meg kell tudnia az ügynök címét. Ez vagy úgy történik, hogy az idegen ügynök folyamatosan küldi szét a saját címét, vagy arra vár, hogy az újonnan belépő gazdagép megkérdezze, „van-e errefelé egy idegen ügynök?".



1. ábra Egy nagy kiterjedésű WAN, amelyhez MAN-ok, WAN-ok illetve vezeték nélküli cellák kapcsolódnak

Amint megvan a cím, a mozgó felhasználónak regisztrálnia kell magát. Ez gyakorlatilag abból áll, hogy elküldi az állandó lakcímét, az aktuális adatkapcsolati réteg címét, illetve szükség szerint valamiféle hitelesítőt. Ezután az idegen ügynök elküld egy csomagot a felhasználó lakcíméhez tartozó hazai ügynöknek, amelyben közli, hogy itt van egy felhasználója. A hazai ügynök ezután felírja az idegen ügynök címét, és elvégzi a hitelesítő ellenőrzését, hogy megbizonyosodjon, nem hazudott-e a felhasználó a kilétével kapcsolatban. Ha ez rendben van, a hazai ügynök visszaküld egy nyugtát. Amint ez visszaérkezik az idegen ügynökhöz, az tudatja a felhasználóval, hogy a regisztráció sikeresen befejeződött.

Ezek után egy tetszőleges forrás és a mozgó felhasználó között a kommunikáció a (*2. ábra*) szerint fog zajlani. Az első lépésben a forrás a felhasználónak csak az otthoni címét ismeri, így a csomagot oda küldi, amit a hazai ügynök elfog. A következő lépésben az ügynök ezt a csomagot egy másik

csomagba ágyazza, és ezt elküldi az idegen ügynök számára. (A csomagok más csomagokba való ágyazását alagút továbbításnak nevezzük. Ezzel az eljárásban a sorozat egy későbbi részében majd részletesen is megismerkedünk). Amint ez megérkezik az idegen ügynökhöz, az kiveszi az eredeti csomagot, és átadja a felhasználó számára. Ezután az idegen ügynök felveszi a kapcsolatot a forrással, és közli vele, hogy a továbbiakban a felhasználónak szánt csomagokat ne az otthoni címre továbbítsa, hanem inkább ágyazza be őket egy olyan csomagba, amelynek a címzettje az idegen ügynök. Innentől kezdve a kommunikáció már nem a hazai ügynökön keresztül fog zajlani, hanem közvetlenül a forrás és az idegen ügynök között.

A gyakorlatban nem ez az egyetlen megoldás a mozgó gépek forgalomirányításának problémájára. Rengeteg ilyen protokoll létezik, és ezek rengeteg szempontban térnek el egymástól. Ilyen szempont például az, hogy a munka oroszlánrészét kire bízzuk: az útválasztókra, vagy a gépekre. A protokollok különböznek még a csomagok másik címre történő irányításának mikéntjében is. Egyes protokollok például nem használják az alagút továbbítást, hanem egyszerűen csak átírják a csomag célcímét. A mozgó gép protokollok közötti különbségeket sok-sok oldalon keresztül tárgyalhatnánk.

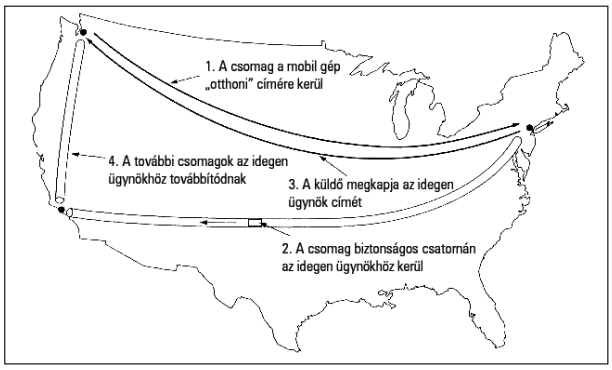
Adatszóró forgalomirányítás

Az ***adatszórás (broadcasting)*** fogalmával már találkozhattunk a közegelérési alréteg tárgyalásakor. Ott olyan LAN-okkal foglalkoztunk, ahol a gépek egy közös csatornát használtak, így mindenki hallott mindenkit, de csak akkor figyelt, ha az üzenet neki szólt. A hálózati rétegben is szükség lehet adatszórásra. Tegyük fel, azt szeretnénk, hogy a hálózatban lévő összes gép órája pontosan járunk. Elmegyünk tehát, és szerzünk egy atomórát, majd beállítjuk úgy, hogy bizonyos időközönként minden gépnek elküldje a pontos időt. (Persze a gépek órája így sem fog teljesen pontosan járni, késni fognak, mivel a csomagok továbbítása időbe telik. Most azonban a példa kedvéért tekintsünk el ettől a problémától).

Miként lehet azonban egy csomagot egyszerre minden gépnek elküldeni? Alkalmazhatjuk például a „favágó” módszert, azaz egyenként minden gép számára elküldjük a pontos időt tartalmazó csomagot. Ez azonban rendkívül sávszélesség pazarló megoldás, arról nem is beszélve, hogy az atomóránknak rendelkeznie kell egy listával, amely az összes gép címét tartalmazza. A módszernek mégis van egy nagy előnye: nem igényel semmiféle alhálózat oldali támogatást, tehát ez minden típusú hálózatban alkalmazható. Sőt, az is lehet, hogy az adatszórás megvalósítására ez lesz az egyetlen járható út. Ha azonban van más megoldás is, akkor inkább azt válasszuk.

A másik nagyon kézenfekvő ötlet az elárasztás alkalmazása, azaz a csomagot az útválasztó az összes kimenetén továbbítja. Sajnos a probléma itt is ugyanaz mint a forgalomirányítás esetében: a csomagok hihetetlen mértékben elszaporodnak, és ez a sávszélesség rovására megy.

Hatékonyabb megoldásnak ígérkezik a ***többcélú forgalomirányítás (multidestination routing)***, ahol egy csomagnak nem csak egy címzettje lehet. Ha egy útválasztó egy több címzettet tartalmazó csomagot kap, akkor először kijelöli azokat a kimeneteit, amelyeken a csomagot továbbítania kell. Egy



2. ábra Forgalomirányítás mozgó felhasználó esetén

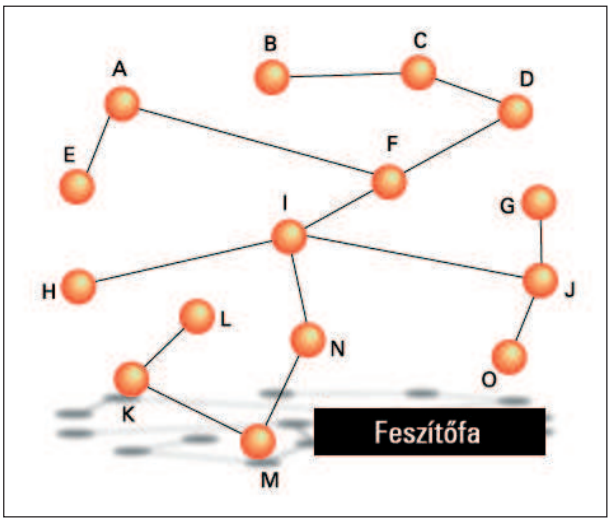
kimenet csak akkor lesz kijelölve, ha van olyan célcíme a csomagnak, amely felé azon a kimeneten vezet a legjobb út. Ezután az útválasztó minden kijelölt port számára előállítja a csomag másolatát, de úgy, hogy csak azokat a célcímeket hagyja benne, amelyeknek az adott kimeneten kell igénybe venniük. Az utolsó ugrásnál, még mielőtt a csomag a gépet elérné, már csak egy címet fog tartalmazni, így a gépek már csak egy közönséges, egy célcímet tartalmazó csomagot fognak kapni. Ha belegondolunk, ez az eljárás az előbb említett favágó módszer továbbfejlesztett változata. Itt is szükség van ugyan a gépek címeit tartalmazó listára, viszont az egy irányba tartó csomagokat csak egyszer kell továbbítanunk.

A legjobb eredményt mégis akkor érjük el, ha az útválasztók az adatszórásra kijelölt csomagot csak azokon a portjukon továbbítják, amely része az alhálózat valamely feszítőfájának, például az adatszórást kezdeményező útválasztóhoz tartozó nyelőfának. (Ha visszaemlékszünk, a feszítőfa az alhálózatnak egy olyan része, amelyben minden router benne van, viszont nem tartalmaz hurkokat. Minden útválasztóhoz csak egy út tartozik). Kivéve persze azt a portot, amelyen az adatszórásra ítélt csomag megérkezett. Ez a módszer azonban feltételezi, hogy az összes útválasztó ismeri legalább egy feszítőfát az alhálózatból. Távolságvektor alapú forgalomirányítás esetén sajnos az útválasztók számára nem áll rendelkezésre ilyen információ, így ott ez a módszer nem használható.

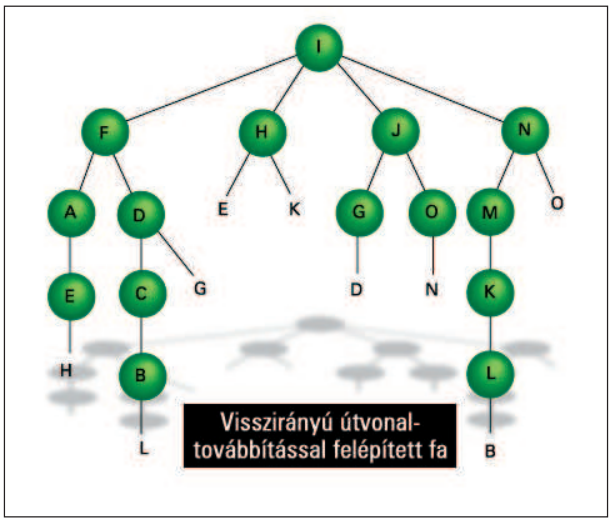
Létezik azonban egy olyan egyszerű és hatékony algoritmus, ahol nincs szükség arra, hogy az útválasztók ismerjenek legalább egy feszítőfát is. Ez az eljárás ***visszirányú továbbítás (reverse path forwarding)*** néven híresült el.

Az alapötlet az, hogy ha az útválasztó egy adatszórásra ítélt csomagot azon a porton kap, amelyik egybeesik az adatszórás forrása felé vezető legjobb úttal (vagy másképp fogalmazva, az adatszórás forrása felé menő csomagokat az útválasztó erre a kimenetre irányítaná), akkor valószínűsíthető, hogy most találkozik először ezzel a csomaggal. Mivel ez az első példány, amit megkapott, elárasztja. Ha azonban ez a csomag egy olyan porton érkezik, amelyik nem esik egybe a forrás felé vezető legjobb úttal, ésszerű feltételezni, hogy ez a csomag viszont másodpéldány, így nyugodtan megszabadulhatunk tőle.

Hogy senki se kételkedhessen az algoritmus működésének helyességében, bemutatunk egy egyszerű példát. A *3. ábrán* láthatunk egy alhálózatot, pontosabban annak az I szerinti nyelőfáját. Ez a gráf azt mutatja, hogy az I-ből miként jut-



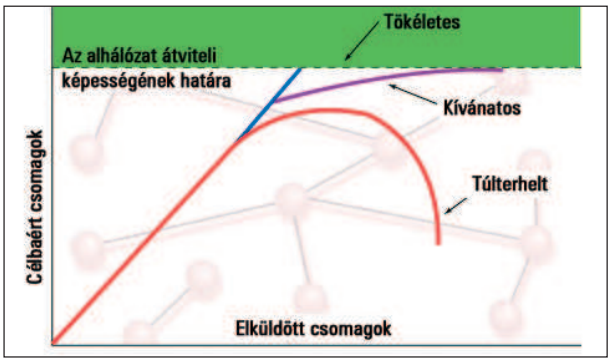
3. ábra Egy alhálózat I csomópont szerinti nyelőfája



4. ábra A visszirányú továbbítás algoritmusának működése

hatunk el a többi csomópontba a lehető legkisebb költségű úton. A *4. ábra* az algoritmus működését szemlélteti. Tegyük fel, hogy az I adatszórást kezdeményez, így összes szomszédjának elküldi a csomagot. Ezt látjuk a 4. ábrán szereplő fa második sorában. Mivel az I szomszédaihoz a csomagok a lehető legrövidebb úton érkeztek meg, ezért egyrészt őket az ábrán egy nagy zöld pötty segítségével kiemeltük, másrészt ők is elküldik minden szomszédukhoz a csomagot (3. sor). Ezek közül megint kijelöltük azokat a pontokat, amelyekhez a csomag az I-től a legrövidebb úton érkezett. A következő lépésnél már csak ezek az útválasztók végzik el az elárasztást.

Érdekes, hogy az E útválasztóhoz az adatszórásra ítélt csomag már az algoritmus harmadik lépésénél megérkezik, az E útválasztó mégsem végzi el az elárasztást, ugyanis az EH él nem része a nyelőfának (habár része az alhálózati topológiának). Amikor azonban az E útválasztó az A-tól kapja a csomagot, akkor elvégzi az elárasztást, hiszen az AE él



4. ábra Ha az alhálózaton a forgalom a kapacitás alatt van, akkor az elküldött és a kézbesített csomagok egymással egyenesen arányosak. Ellenkező esetben torlódás lép fel, és a csomagok nagy többsége sosem fog célba jutni.

szerepel a feszítőfában. Az algoritmus hatékonysága tehát némileg elmarad az előbb bemutatott, a nyelőfát pontosan követő algoritmustól, viszont nem is ad sokkal rosszabb eredményt. A nyelőfát követve az algoritmus 4 lépés alatt végezne, szemben a mostani 5 lépéssel. Ez ugyan jobb eredmény, de viszont az útválasztóknak nem kell ismerniük az alhálózat egyetlen feszítőfáját sem. Ezenkívül sokkal jobb megoldás, mint az észnélküli elárasztás használata, mivel magától leáll, és nincs szükség belső mechanizmusra, amely leállítaná a végtelenig tartó csomagképződést.

Többesküldéses forgalomirányítás (multicasting routing)

Mi a helyzet akkor, ha nem mindenkinek, hanem csak a gépek egy kisebb csoportjának szeretnénk csomagot küldeni. Ilyesmire is gyakran szükségünk lehet, például ha egy adatbázist elosztottan szeretnénk tárolni. (Tipikusan ilyen adatbázist használ a tartományneveket az IP címekkel összekapcsoló DNS is, amelyről sorozatunk utolsó szakaszában, az alkalmazási réteg tárgyalásakor foglalkozunk). Ekkor az adatbázis kezelését végző folyamatoknak időnként szüksége lehet arra, hogy a többi, a munkában szintén részt vevő folyamatoknak üzeneteket küldjenek.
Ha a munkában közösen résztvevő folyamatokat futtató gépek száma viszonylag kicsi, akkor a probléma megoldható azzal, hogy a folyamatok küldhetnek egymásnak egyszerű két pont közötti üzeneteket. Ha a csoport nagy, akkor a módszer nem hatékony. Próbálkozhatunk adatszórással is, kivéve ha a hálózatban szereplő gépek száma nem elenyészően kicsi a teljes hálózathoz viszonyítva. Ilyenkor ugyanis az üzenet a gépek legnagyobb részét nem is érdekli, az adatszórás tehát rendkívül pazarló lenne.
A megoldást a ***többesküldésnek (multicasting)*** nevezett technológia jelenti, amely lehetővé teszi, hogy csomagot küldhessünk a gépek egy jól meghatározott csoportjának. Ehhez persze először valami módon meg kell határoznunk ezeket a csoportokat, ez azonban a forgalomirányítás szempontjából nem túlzottan érdekes, így erre most nem térünk ki külön. Ha egy folyamat belép egy csoportba, akkor azt tudatnia kell a gépével. A többesküldéses forgalomirányításban alapvetően fontos, hogy minden útválasztó tudja, melyik hoszt melyik csoportba tartozik. Itt rögtön fel is merül egy elvi kérdés: a gépnek kelljen-e szólni az útválasztónak, hogy egy új csapatnak kötelezte el magát, vagy az útválasztók kérdezzék ki gépeiket hovatartozásukról? Bárhogy is valósuljon ez meg a gyakorlatban, az útválasztók ezt az információt közlik az alhálózat további csomópontjaival.
A forgalomirányítás úgy működik, hogy az útválasztók kiszámítják az alhálózat egy feszítőfáját. Amikor egy többesküldésre szánt csomag megérkezik az első útválasztóhoz, az a saját feszítőfájából eldobja azokat az utakat, amelyek olyan útválasztókhoz vezetnek, akiknek nincs olyan gépük, amelyik tagja lenne a csoportnak. A csomagot ezután az így létrejött megcsonkított feszítőfa mentén kell továbbítani.

Torlódásvédelem

Egy ***torlódás (congestion)*** kialakulásánál nehezen lehet elképzelni nagyobb katasztrófát egy alhálózat életében. Nem csak arról van szó, hogy az alhálózat a csomagokat lassabban fogja célba juttatni, hanem az is előfordulhat, hogy egyszerűen képtelen lesz ellátni a feladatát, és szinte egy csomag sem fog eljutni a rendeltetési helyére.
Az *5. ábrán* látható grafikonon a kézbesített csomagokat ábrázoltuk az alhálózatba beadott csomagok számának függvényében. Ha ez a szám kisebb, mint az alhálózat maximális kapacitása, akkor a két érték között egyenes arányosság van. Ha azonban nagyobb, akkor fellép a torlódás, és az alhálózat teljesítőképessége nagyban visszaesik.
Torlódást sokféle esemény kiválthat. A leggyakoribb oka az, hogy egy helyen az alhálózatba hirtelen nagy mennyiségű csomag kezd beáramlani. Ha egy útválasztó hirtelen sok csomagot kap, akkor kialakul egy várakozási sor, azaz a beérkező csomag először az útválasztó memóriájába kerül, és ott várakozik egészen addig, amíg az útválasztó fel nem szabadul, és el nem kezdheti feldolgozni azt. Ha azonban túl sok csomag érkezik, előfordulhat, hogy az útválasztó memóriája betelik, és az ezután érkező csomagok mind elvesznek. (Érdekes, hogy a probléma nem oldódna meg azzal, ha több memóriát pakolnánk az útválasztóba, például végtelen mennyiségűt. Sőt, a helyzet ezzel csak rosszabbodna! Mire a router elérné a sor végi csomagokat, addig a forrás időzítője már rég, minden bizonnyal többször is lejárt, így az útválasztó sora már másodpéldányok sokaságát is tartalmazza. Az útválasztó persze ezeket is továbbítani fogja, ezzel is növelve a terhelést a már amúgy is teljesen lelassult hálózaton).
A torlódás másik gyakori oka az, hogy nincs egyensúlyban a vonalak sávszélessége és az útválasztók számítási kapacitása. Ha az útválasztók lassú CPU-val rendelkeznek, akkor úgy is kialakulhat torlódás, ha közben rendelkezésre áll szabad vonalkapacitás. A dolog fordítva is igaz: hiába képes az útválasztó gyorsan végezni a feladatát, ha a vonalak túl lassúak.
Ha már egy torlódás kialakult, akkor nagyon könnyen szétterjedhet az alhálózat többi részére is. A torlódásnak van egy öngerjesztő hatása. Ha bizonyos csomagok elvesznek, mivel már nincs hely az útválasztó memóriájában, az adó újra és újra megpróbálja elküldeni ugyanazt a csomagot, ezzel másodpéldányokat zúdítva a már amúgy is leterhelt útválasztója.
A torlódások elleni védekezés tehát létkérdés a hálózatok életében. A következő részben részletesen megvizsgáljuk, milyen módszerek léteznek a torlódások elkerülésére.

Garzó András
garzo@interware.hu

A főszakács gyűjteménye

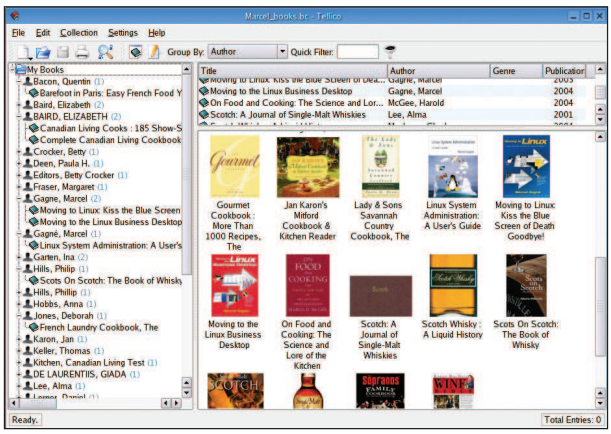
Katalogizáljuk könyveinket – vagy bármi mást – egy olyan alkalmazással, amelynek csak az ISBN-azonosítót kell megadnunk, a többi adatot önműködően lekérdezi.

Aha! Szóval itt van a 2005-ös borenciklopédiám! ***Mon Dieu***, François, már mindenhol kerestem. Várj csak egy percet! Itt a párizsi szakácskönyvem, a toszkánai gyűjteményem és a provance-i fűszerekről szóló leírásom is! Mégis hány könyvem van itt nálad!? Nem, ***mon ami***, semmi másra nem célozgatok, azon kívül, hogy egy ideje már keresem őket. Igen, kétségtelenül igazad van, legalább nem vesztek el. Na mindegy, inkább igyekezz, és teríts meg, ***mon ami***, vendégeink bármelyik pillanatban betoppanhatnak.

Ó, késő, hiszen már meg is érkeztek. Üdvözöllek benneteket, ***mes amis*** a ***Chez Marcelben***, ahol a legfinomabb linuxos fogások a menü állandó tagjai, és a borospince a világ legjobbjai közé tartozik! Foglaljatok helyet, érezzétek magatok otthon, ***François*** máris hozza a bort. Kérlek, ***mon ami***, szaladj le a pince északi szárnyába, és hozd vissza azt a 2000-res bordóit, amit nemrég… khm, szóval minőségellenőrzésnek vetettünk alá. A „2010-ig nem nyitható ki” feliratú ***Margaux*** mellett van. ***Vite***, François. ***Mozogj***!

Amíg készséges segítőm visszaér az itallal, hadd ismertessem mai menünket. Mint tudjátok, a ***Chez Marcel*** az elmúlt évek során már számos recepttel szolgált; és persze borból sem kis mennyiség fogyott. Bármennyire is szeretnék visszaemlékezni mindenre, a valóság az, hogy nem megy. Éppen ezért több polcnyi a ***Linuxról***, a főzésről és a borokról szóló könyv sorakozik a konyhában, a pincében és az irodában egyaránt. Innen kezdve az egész csak szervezés kérdése, éppen ezért szükségem van egy adatbázisra.

De miféle adatbázis legyen ez? Nyilván valami elképesztően rugalmas, mégis könnyen kezelhető dolog. Például a ***Tellico***. ***Robby Stephenson Tellicoja*** elvileg gyűjteménykezelésre született, ám szerintem inkább egy roppant sokoldalú, személyi könyvtárrendszer. Nagyszerű eszköz receptes könyveink katalogizálására, de mellettük a linuxos kiadványok, a tudományos-fantasztikus regények és a krimik is kiválóan megférnek. (*1. ábra*) Arra is kiválóan megfelel, hogy figyelemmel kövessük, barátainknak és családtagjainknak mely könyveinket adtuk kölcsön. Nem tudom, ti hogy’ vagytok vele, ***mes amis***, de én elég sok könyvet kölcsönadtam az elmúlt években, és jó részük bizony sose került hozzám vissza. Akik kölcsönkérték a könyveimet, elfelejtették, hogy kitől kapták őket, és én is elfelejtettem már, hogy kinek mit adtam oda. Kivéve persze ***François***-t. Róla külön listát vezetek.

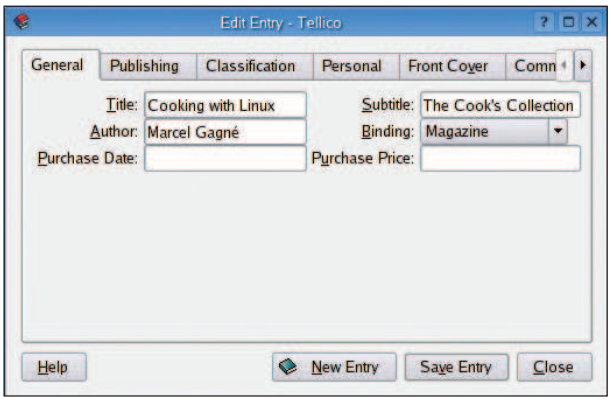


1. ábra A Tellico egy nagyszerű személyi könyvtárrendszer, ráadásul jól is néz ki

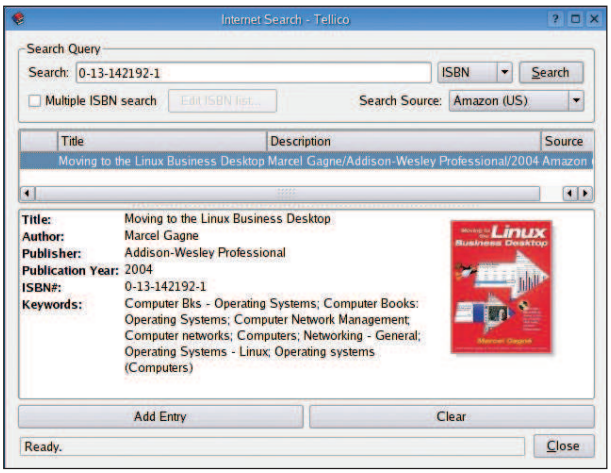
A ***Tellico*** sablonok segítségével már gyűjtemények – videók, zenék, érmék, bélyegek stb. – rendszerezésére is használható. Még borospincékhez is van hozzá sablon. Saját gyűjteményeket is létrehozhatunk alatta, továbbá a meglévő űrlapokat is módosíthatjuk. Szeretném megmutatni, hogyan lehetséges mindez. Előfordított csomagokat szinte minden fontosabb terjesztéshez – ***Fedora***, ***SuSE***, ***Mandrake***, ***Slackware*** stb. – találunk. A forrást is letölthetjük (lásd az internetes forrásokat), ekkor a megszokott öt lépésből álló telepítési folyamatot kell végigjátszanunk:

```
tar -xzvf tellico-0.13.1.tar.gz
cd tellico-0.13.1
./configure –prefix=/usr
make
su -c “make install”
```

A ***Tellico*** egy a ***KDE 3.1***-es vagy újabb kiadása alá készült csomag, működéséhez szükség van a megfelelő ***Qt*** és ***KDE*** fejlesztői könyvtárakra. Ha forrással dolgozunk, akkor



2. ábra Könyv adatainak beírása egy gyűjteménybe



3. ábra Internetelérés birtokában a könyvek adatainak megadása gyerekjáték

néhány további, elhagyható könyvtár lefordítását is fontoljuk meg. A ***taglib*** fejlesztői könyvtárcsoport az első ilyen, segítségével zenei fájlokból ki tudjuk olvasni adataikat; erről még lesz szó. Egy másik kiegészítő a ***yaz***. Ha ezzel együtt fordítjuk le a ***Tellicot***, akkor alóla ***Z39.50*** kereséseket tudunk indítani.

Amikor elindítjuk a ***Tellicot*** – a `tellico` paranccsal –, akkor közhelyesen szólva, tiszta lappal kezdünk. Nagyítsuk kényelmes méretre az ablakot, és máris nekiláthatunk első gyűjteményünk megadásának. Ha könyvgyűjteményt akarunk megadni, akkor kattintsunk a ***Tellico*** menüsor ***File (Fájl)*** parancsára, majd a ***New (Új)*** és a ***New Book Collection (Új könyvgyűjtemény)*** parancsra. Mint említettem, a ***Tellico*** egy kiváló személyi könyvtárrendszer, segítségével figyelemmel követhetjük, hogy milyen könyveink vannak, mikor és hol vásároltuk őket, valamint kinek adtuk őket kölcsön. Azonban mielőtt használhatnánk ezeket a szolgáltatásait, rögzítenünk kell gyűjteményünk tagjainak adatait. (*2. ábra*)

A különféle lapokon megadhatjuk a szerző nevét és a könyv címét, de beírhatjuk a kiadót, a megjelenés dátumát, hogy hányadik kiadásról van szó, a műfajt, a sorszámot, a kiadvány állapotát, hogy dedikált példányról van-e szó, hogy éppen kölcsönadtuk-e valakinek, valamint kiolvastuk-e. Rengeteg további mező is van, ezek felfedezését az olvasóra hagyom; talán még a borítókép megadásának az 1. ábrán is látható lehetőségére szeretném felhívni a figyelmet. Ha mindezen adatok beírása túlságosan hosszadalmasnak tűnik, akkor van egy másik lehetőség is. Nem, nem arra gondolok, hogy fel kell bérelni valakit. Mindössze internetkapcsolatra van szükségünk, a ***Tellico*** tökéletes kényelmet biztosít. Kattintsunk az ***Edit (Szerkesztés)*** menü Internet ***Search (Internetes keresés)*** parancsára.

Amikor az ***Internet Search*** párbeszéd megjelenik (*3. ábra*), adjuk meg a könyv címét, szerzőjét és ***ISBN***-azonosítóját (***International Standard Book Number, nemzetközi szabványos könyvazonosító***) vagy az általunk kiválasztott kulcsszót. A keresések az ***Amazon.com*** adatbázisa alapján történnek, de brit, japán és német oldalakon is tudunk keresni. Ha ***ISBN***-azonosító alapján akarunk keresni, akkor jó esetben csak egy találatot fogunk kapni, míg másfajta kereséseket indítva többet is. Válasszuk ki a megfelelőt, majd kattintsunk az ***Add Entry (Bejegyzés hozzáadása)*** gombra.

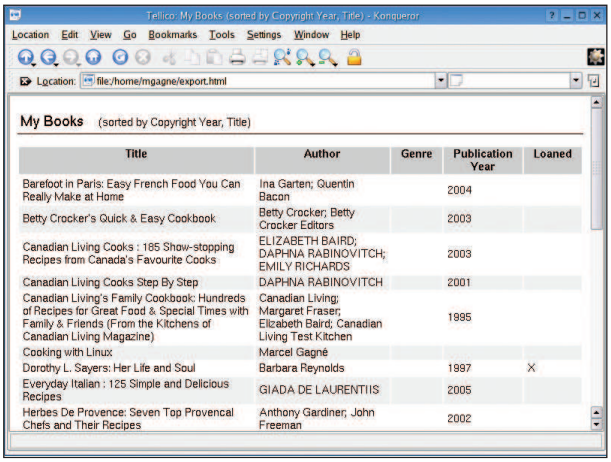
Ekkor frissül az adatbázisunk, sőt, a borító képe is belekerül. A ***Tellico*** képes az intelligens, adott címre vagy címtartományra vonatkozó keresésre is. Az adatokat a jobb oldalon látható listákban szereplő, a mezőknek megfelelő oszlopok hozzáadásával vagy eltávolításával kedvünk szerint érhetjük el. Ha például mindig tudni szeretnénk, hogy mi van kölcsönadva, akkor kattintsunk az egér jobb gombjával a mezősorra, és adjuk hozzá a ***Loaned (Kölcsönadva)*** mezőt. Ekkor a kölcsönadottként megjelölt könyvek mellett egy zöld pipa jelenik meg.

Az említettek mellett egyéb lehetőségeink is vannak a ***Tellico*** adatbázisának feltöltésére. Nyissuk meg a ***File*** menü Import almenüjét. Itt választhatunk, hogy milyen formátumú fájlból akarunk adatokat beemelni: ***CSV***, ***Alexandria***, ***Bibtex*** stb.

Az exportálási szolgáltatás még érdekesebb – közben térjünk át a jelentéskészítés témájára. Persze egy-egy bejegyzést bármikor kiírathatunk, de az exportálási szolgáltatás ennél jóval többre is alkalmas. A ***HTML*** exportálást választva például készíthetünk olyan ***HTML*** oldalt, amelyen minden könyvünkről szerepelnek az általunk kiválasztott adatok. Megadhatjuk, hogy mi legyen a ***HTML*** fájl neve, hogy hogyan akarjuk formázni az egyes bejegyzéseket vagy mezőket, és így tovább. Végeredményként egy szépen formázott, letisztult ***HTML*** oldalt kapunk. (*4. ábra*)

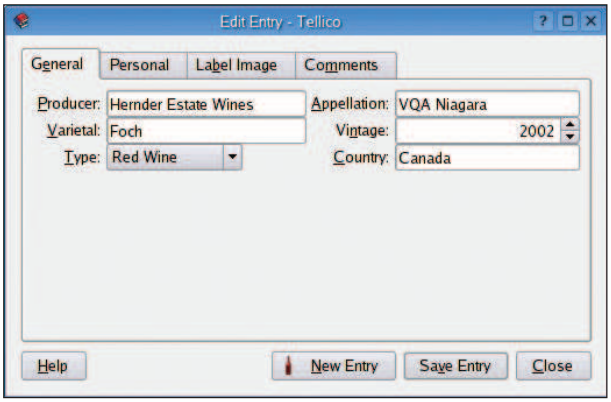
Mielőtt továbblépnénk, szeretnék megemlíteni egy másik exportálási lehetőséget, ami különösen a szívemhez nőtt. Ha a listáról az ***Export to PilotDB*** elemet választjuk, akkor ***PDB*** formátumú, ***Palmon*** is olvasható jelentést tudunk előállítani. A ***hotsync***-kel másoljuk fel a dokumentumot, és bármikor kéznél lesz a katalógusunk.

Korábban említettem, hogy a ***Tellicohoz*** más gyűjteményekhez, például zeneiekhez is léteznek sablonok. Aki még nem rendelkezik semmilyen katalógussal, egyszerűen válassza a ***File*** menü ***New Music Collection (Új zenegyűjtemény)*** parancsát. A ***CD***-k adatainak megadása hasonló a könyveknél látotthoz, csak a mezők különböznek. ***CD***-gyűjteményünk katalogizálása még könnyebb, ha a ***taglib*** kiterjesztések telepítve vannak a gépünkre. Ekkor csak tegyük be a kívánt zenei CD-lemezt a ***CD-ROM***- vagy ***DVD***-meghajtónkba, és kattint-



4. ábra HTML formátumú jelentés

5. ábra A Tellico közvetlenül a CD-lemezekről is képes beolvasni a címeket, az előadók nevét és a zeneszámok adatait



6. ábra Akár borospince-adatbázist is készíthetünk

sunk a *File* menü *Import*, majd *Import Audio CD Data (Zenei CD adatainak importálása)* parancsára. (*5. ábra*)
A program beolvassa a *CD* adatait, majd beemeli őket a katalógusba.
Ha az adatok beolvasása megtörtént, lépjünk vissza, és egészítsük ki az adatokat, ahogy kívánjuk. Természetesen, ha bakeliten vagy szalagon van a gyűjteményünk, akkor marad a gépelés. A könyvekhez hasonlóan itt is megjelölhetjük a kölcsönadott anyagokat. Ha pontosan tudjuk, milyen könyveink és zenéink vannak, és éppen hol találjuk őket, akkor garantáltan megnyugszik a lelkünk. Dőljünk hátra, lazítsunk – így még egy pohárka bor is jobban fog esni.
És újra a bornál találjuk magunkat, na nem mintha ez olyan rossz téma volna. Mi van a borospincével? Hihetetlen, de a *Tellico* még itt is szóba jön. Ahogy könyveinket és zenéinket katalogizáltuk, úgy borgyűjteményünk adatait is összegyűjthetjük. Kattintsunk a *File* menü *New* pontjára, majd a *New Wine Collection (Új borgyűjtemény)* parancsra. Kattintsunk a *Collection (Gyűjtemény)*, majd a *New Entry (Új bejegyzés)* parancsra, és szép egyenként meg is kezdhetjük boraink adatainak bevitelét. (*6. ábra*)
Sajnos nincs semmiféle varázslatos mód arra, hogy borgyűjteményünk jellemzőit egy csapásra beillesszük az adatbázisba – minden egyes üveget kézbe kell vennünk, és be kell gépelnünk az adatait. Persze ez legyen a legnagyobb bajunk, hiszen ha lent lehetünk a pincében, és szöszmötölhetünk a gyűjteményünkkel, az nem is teher, hanem szórakozás.
Végül, ha valaki egyedi adatbázisok kezelésére keres megoldást, ismét a *Tellicot* tudom ajánlani. Ha az előre megadott sablonok egyike sem felel meg igényeinknek, hozzunk létre saját gyűjteményt. Az alapértelmezett gyűjteménymezők ekkor rendkívül egyszerűek, csak a címre hagyatkoznak, de bátran módosítsuk őket. Miután létrehoztuk egyedi gyűjteményünket, kattintsunk a menüsor *Collection (Gyűjtemény)* elemére, majd válasszuk a *Collection fields (Gyűjteménymezők)* parancsot. Itt további mezőket is megadhatunk, ezek szövegesek, numerikusak vagy bármi egyebek lehetnek, amit csak szeretnénk.
Most látom, *mes amis*, az óra mutatója bizony már a záróra után ballag. Most, hogy borospincénk tökéletesen naprakész lett, *François* minden figyelmét ratok fordíthatja, és örömmel tölti újra poharatokat. A következő alkalomig, *mes amis*, igyunk egymás egészségére!
A *votre santé! Bon appétit!*

Linux Journal 2005. április, 132. szám

Marcel Gagné díjnyertes író, az ontarioi Mississaugában él. Legújabb, immár harmadik könyve a Moving to the Linux Business Desktop (ISBN 0-131-42192-1, Addison-Wesley). Hobbipilóta, Top-40-es lemezlovas volt, tudományos-fantasztikus írások szerzője, jelenleg egy kisebb origami T-Rexen dolgozik. A mggagne@salmar.com címen érhető el. Weboldalán számos további érdekességet találni, többek közt kiváló boros hivatkozásokat is: www.marcelgagne.com.

KAPCSOLÓDÓ CÍMEK

- www.periapsis.org/tellico
- www.marcelgagne.com

Betörések és betörési kísérletek I.

Gyakran használunk a köznapi nyelvben olyan kifejezéseket, melyeket a jogászi világ nem ismer. Például azt mondjuk: „betörés”, és a tényállás, amire gondolunk valójában „lopás, dolog elleni erőszakkal”.[1] Vagy ha az első kijelentést informatikusként tesszük, akkor – bár talán nem is sejtjük – a számítástechnikai rendszer és adatok elleni bűncselekményre utaltunk.

Vegyük alapul a következő szituációt: egy körülbelül 100 gépből álló hálózat rendszergazdái vagyunk, van egy remek tűzfalunk, és csak hobbiból gyűjtjük a betörési kísérleteket tartalmazó naplókat.
A kísérletek alapvetően két csoportba sorolhatóak:

1. Az interneten szanaszét szóródott kíváncsiskodó programok, melyek, bezörgetnek, aztán, mikor látják, hogy a tűzfal nem egy „trivial joke” tovább állnak, ártani sem akarnak nagyon, csak éppen ott lenni.

2. A másik szélsőséget azok a speciálisan irányított programok jelentik, amik sokkal agyafúrtabbak és „felügyelet alatt” próbálnak hozzáférkőzni a rendszerünkhöz. Konkrét céljuk van, adatot keresnek, módosítanak, vagy éppen tüntetnek el. Ilyen pl. mikor egy banki rendszert feltörve saját számlájukra utal(tat)nak a ckrackerek.

Az elkövetés módja

Lássunk egy példát az első variációra!
Egy nap egy arra leszünk figyelmesek, hogy a naplóbejegyzések között egy bizonyos nyom túl gyakran szerepel. A kísérlet első pillanatra nem tűnik komolynak, de a próbálkozások száma aggasztó. Eltelik további két nap (nyilván a logokat most már gyakrabban figyeljük), a betörési kísérletek száma nő.
Jogászi szemmel nézve az elkövetési magatartás: ***számítástechnikai rendszer elleni bűncselekmény kísérlete***. Ha valóban sikerült volna feltörnie a rendszerünket, akkor a tényállás minden elemét megvalósítva, elköveti a vétséget (alapesetben vétség, és egy évig terjedő szabadságvesztéssel, közérdekű munkával vagy pénzbüntetéssel büntetendő). Így azonban a cselekménye a kísérlet szintjén megrekedt.[2]
A kísérletnek több típusa van, lássunk ezekre vonatkozóan egy-egy példát.

- ***Befejezett kísérlet***: az elkövető saját részéről mindent megtett, hogy a bűncselekmény megvalósulásához szükséges eredmény (jelen esetben a jogosulatlan belépés) bekövetkezzen. Tehát tudatosan elindított egy programot a neten, ami arra szolgál, hogy mások rendszerébe beférkőzzön.

- ***Befejezettlen a kísérlet***, ha nem követett el mindent, például megpróbálja elindítani a programot, de nehézségekbe ütközik (tegyük fel nem tud azon a nyelven, amelyiken a programot írták, és rosszul adja ki a parancsot).
A kísérletre fő szabályként a befejezett bűncselekmény büntetési tételét kell alkalmazni, azonban korlátlanul enyhíthető a büntetés, ha a kísérletet alkalmatlan tárgyon vagy alkalmatlan eszközzel próbálják megvalósítani.

- ***Alkalmatlan tárgyon***: a célrendszer, amit megpróbál feltörni valójában nem létezik, csak a barátai ültették a bogarat és az ip címet a fülébe.

- ***Alkalmatlan eszköz***: ilyen esetben egy hálózat nélküli, asztali gépről próbálja meg futtatni a programot, amely nyilván nem jut ki a szobájából, sőt a gépéből sem.

Nem büntethető kísérlet miatt, akinek elállása folytán marad el a bűncselekmény befejezése, továbbá az sem, aki az eredmény bekövetkezését önként elhárítja.
Ezek az esetek még szemléletesebbek:

- ***Elállás***: Tehát a cracker az asztalánál ül, ujja az enteren, épp készül rászabadítani a világra egy tucat vírust, amikor megszólal a lelkiismerete és a „start” parancs helyett a „delete” parancsot adja ki.

[1] Btk. 316.§ e,
[2] Btk. 16.§

- *Elhárítás (meglehetősen légből kapott)*: Miután elindította a programokat, eszébe jut a Btk, és gyorsan végigtelefonálja a megcélzott intézményeket, mindenütt éppen véletlenül sikerül is beszélnie a rendszergazdákkal, akiknek pár perc alatt (esetlegesen programsorok bediktálásával, vagy megküldésével) elmondja, hogyan is lehet az ő vírusait megállítani.

Az első eset persze tovább súlyosbítható, hiszen nem csak belépni, jogosultságot túllépve vagy megsértve bent maradni lehet, hanem – ha már ott van – az elkövető

1. jogosulatlanul megváltoztathatja, törölheti vagy hozzáférhetetlenné teheti a rendszerben tárolt, feldolgozott, kezelt vagy továbbított adatot (pl. egy nagyobb vállalatnál a karácsonyi címlistát összekeveri, és az üdvözletek ennek hatására sosem érkeznek meg)

2. adat bevitelével, továbbításával, megváltoztatásával, törlésével, illetőleg egyéb művelet végzésével a számítástechnikai rendszer működését jogosulatlanul akadályozza, (olyan méretű leveleket küldözget (kitömörített kernel), hogy a teljes levelezést lefagyasztja.

Ezekben az esetekben a magatartás még mindig vétségnek minősül, de a büntetési tétel felső határa két év, a közérdekű munka illetve pénzbüntetés marad.
Még eggyel súlyosabban esik a latba, ha a fentebbi cselekedeteket jogtalan haszonszerzés végett teszi és cselekedetével kárt okoz. A cselekmény ebben az esetben már bűntettnek minősül, a büntetési tétel maximuma pedig 3 év, de a kár nagyságának függvényében emelkedhet:

- egy évtől öt évig terjedő szabadságvesztés, ha a bűncselekmény jelentős kárt okoz,

- két évtől nyolc évig terjedő szabadságvesztés, ha a bűncselekmény különösen nagy kárt okoz,

- öt évtől tíz évig terjedő szabadságvesztés, ha a bűncselekmény különösen jelentős kárt okoz.

A kár mértéke

Amint láttuk, az okozott kár nagy mértékben befolyásolja a bűncselekmény súlyát. Az összegekkel kapcsolatosan a törvénykönyv maga rögzíti a határokat,[3] melyeken belül a bíróság csupán a büntetési tétel határai között mozogva veheti figyelembe, hogy az elkövető alig lépte át a határt, illetve, hogy 200 forint híján másik kategóriába esik.
Lássuk ezt a rendszert:

- ***kisebb***, ha tízezer forintot meghalad, de kétszázezer forintot nem halad meg,

- ***nagyobb***, ha kétszázezer forintot meghalad, de kétmillió forintot nem halad meg,

- ***jelentős***, ha kétmillió forintot meghalad, de ötvenmillió forintot nem halad meg,

- ***különösen nagy***, ha ötvenmillió forintot meghalad, de ötszázmillió forintot nem halad meg,

- ***különösen jelentős***, ha ötszázmillió forintot meghalad.

A technikai intézkedések kijátszása, és következményei

Egy másik, tényállásként rögzített bűncselekmény a „számítástechnikai rendszer védelmét biztosító technikai intézkedés kijátszása." Ez szorosan összefügg az előzőekben ismertetettel, hiszen itt a számítástechnikai rendszer és adatok elleni bűncselekmény elkövetése céljából, az ehhez szükséges vagy ezt könnyítő számítástechnikai program elkészítése, a jelszó, a belépési kód, vagy számítástechnikai rendszerbe való belépést lehetővé tevő adat megszerezése illetve forgalomba hozása, az azzal való kereskedés, vagy más módon hozzáférhetővé tétel jelenti magát a cselekményt.
Ezzel kapcsolatban életszerű példa az a pár éve történt eset, mikor az interneten közzétették egy szolgáltató előfizetői névsorát, felhasználói nevekkel, jelszavakkal együtt. Ez a cselekmény tehát a mai szabályozás szerint minimum vétségnek minősülne és 2 évig terjedő szabadságvesztéssel, pénzbírsággal vagy közérdekű munkával lenne büntethető.
Mentesül azonban az elkövető, aki még mielőtt fentebbi cselekedete a hatóság (többnyire rendőrség vagy nemzetbiztonsági hivatal) tudomására jutott volna tevékenységét a hatóság előtt felfedi, és az elkészített dolgot a hatóságnak átadja, valamint lehetővé teszi a készítésben részt vevő más személy kilétének megállapítását.
Mostani utolsó elemként egy szintén érdekes – talán sokak szerint ismeretlen részlet: Bűncselekmény az is, ha valaki a számítástechnikai program, jelszó, belépési kód, vagy valamely számítástechnikai rendszerbe való belépést lehetővé tevő adat készítésére vonatkozó gazdasági, műszaki, szervezési ismereteit másnak a rendelkezésére bocsátja. (elkövetővé válik például az, aki lediktálja valakinek a kódot, vagy elmondja, ő hogyan készítené, vagy az is, ha részegen a kocsmában kifecsegi, hogy milyen biztonsági lyukak vannak az általa épített vagy üzemeltetett rendszerben.)

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

[3] Btk. 138.§