Hírek

Csuklóra rögzíthető PDA Linuxszal

A *Eurotech* bemutatta csuklóra rögzíthető számítógépét, amely a megrendelő igényétől függően ***Linuxot*** vagy ***Windows CE***-t futtat. Az eszköz körülbelül 200 gramm súlyú és hiánypótló lehet például kórházakban, logisztikai és őrző-védő cégeknél. A lehetséges felhasználási területeknek csak a képzelet szabhat határt. ***32 megabájt flash*** memóriával és ***64 megabájt*** programmemóriával rendelkezik, míg ***Secure Digital*** kártyával 1 gigabájtig bővíthető. A készülék litium-polimer akkumulátora 6-8 órás üzemidőt nyújt, ami hosszúnak mondható, ha figyelembe vesszük, hogy 3,5 hüvelykes érintőkijelzővel bír, valamint nem hiányozhat belőle a ***Bluetooth*** és a ***Wifi*** sem. Az már csak hab a tortán, hogy egy 16 csatornás ***GPS vevő*** is van benne.

A Google AMD-re vált

A hírek szerint *AMD*-re ***cseréli*** szervereit a ***Google***. Ez több, mint ***kétszázezer kiszolgálót*** jelent világszerte. Beszerzésekor már csak 64 bites ***Opteron*** processzorral szerelt szerverek szerepelnek a megrendeléseken. Míg az ***Intelnek*** ez komoly ökölcsapás, addig az *AMD* szárnyra kaphat – jelzi előre több tőzsdei tanácsadó.

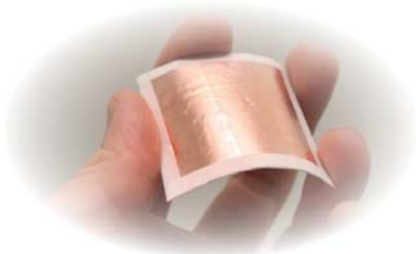
Titkosított VoIP

Philip Zimmermann – a *PGP* írója – elkészítette a ***Zfone***-t, amely képes bármely *SIP*-alapú *IP* telefonhívás titkosítására. Az első kiadás ***Linux*** és *Mac* alá készült el. ***Zimmermann*** reméli, hogy az oly népszerű ***OpenWengo***,ill. ***Gizmo*** fejlesztői integrálják a szoftvereikbe a megoldást, hogy ezáltal még triviálisabb legyen a használata. A kód jogállása még nem dőlt el, de várhatóan nyílt forrású lesz. A ***Zfone*** windowsos verzióját áprilisra ígéri ***Philip***.
➲ ***http://www.philzimmermann.com/EN/zfone/index.html***

Havi 20 dollárért Wifi Philadelphiában

Philadelphiában hamarosan kiépítik az egész várost – mintegy 350 négyzetkilométert – lefedő ***wifi hálózatot***. A város vezetése a technológiai haladást jelölte meg fő oknak a döntés meghozatalakor. Havi 20 dollárért az ügyfelek mintegy egy megabites elérést kapnak. Körülbelül 50-80 ezer ügyfélre számítanak, míg a beruházás 2007 elejére készül el.

Laposelem a javából



A finn *Enfucell* kifejlesztett egy elemet, amely már igazán lapos. Stabil energiát szolgáltat, így például elektronikus papírok energiaforrása is lehet a jövőben. Maga az elem hajlékony, mint egy papír, olcsó, környezetkímélő és szinte bármilyen környezetben használható.
➲ ***http://www.enfucell.com/***

146 dolláros PC Kínából

A kínai ***Municator*** bejelentette – főképp irodai felhasználásra szánt – PC-jét. A dobozban minden benne van, amit egy átlagos irodista várhat: 40 gigabájtos merevlemez, 400/800 MHz-es ***Godson C2-es processzor***, 256 megabájt memória, 4 darab ***USB 2.0***-ás port. Az egész doboz nem nehezebb 400 grammnál, ami pedig a fogyasztását illeti: 45 wattal is beéri, akár egy laptop.
A gyártó Linuxszal (***Thinix OS***) kínálja, amely tartalmaz ***Firefox***-ot, ***Thunderbird***-öt, ***Gaim***-et, ***Red Office***-t, az ***mplayer***-t, de nem hiányozhat a ***Skype*** sem.
➲ ***http://yellowsheepriver.org/***

Flash alapú laptop merevlemez

A ***Samsung*** bejelentette ***32 gigabájtos*** laptop merevlemezét, mely ***flash*** technológián alapul. Az ***1.8 hüvelykes*** merevlemez már használható sebességű, hiszen 32 és 57 megabájt per másodperc között változik, azonban lényegesen kevesebb energiát használ, mintegy hagyományos merevlemez. Azt sem szabad elfelejteni, hogy teljesen ***hangtalan***, hiszen nincs benne mozgó alkatrész. A ***Samsung*** szerint az új flash merevlemezek alkalmazása akár ***20%-al is meghosszabbíthatja*** a laptopok üzemidejét. A meghajtó ára egyelőre ismeretlen.

Linuxos médialejátszó

A koreai *Cowon* legújabb – A2-re keresztelt – médialejátszója *Linuxot* futtat. Az eszköz gyomrában egy 1.8 hüvelykes 20 vagy 30 gigabájtos merevlemez rejtőzik, míg a videók egy 4 hüvelykes, 480x272 képpont felbontású 14 bites színmélységű *LCD*-n jelennek meg. Az energiaéhséget egy 4300 mAh kapacitású akkumulátor elégíti ki.
A készülék tömege 300 gramm, azonban egy töltéssel ***10 órányi videót*** vagy ***18 órányi hangot*** játszhat le rajta a felhasználó. Az extrák között találhatunk ***rádiót, ébresztőórát***, illetve lehetőségünk van egyszerűbb ***elektronikus könyvek*** – e-book-ok – olvasgatására. Az adatok gyors áttöltéséhez ***USB 2.0***-es csatolóval látták el az eszközt, valamint támogatja az ***USB Mass Storage*** szabványt. További extra, hogy ***USB hostként*** is funkcionál, ami lehetővé tesz pl. a fotók áttöltését megfelelő digitális fényképezőgépről. Természetesen – amint azt már megszokhattuk az ilyen eszközöknél – a beépített szoftvere frissíthető.
➲ http://eng.iaudio.com/

Google Page Creator

A *Google* bemutatta a *Google Page Creatort*, amellyel másodpercek alatt összeállíthatunk letisztult *HTML* oldalakat a rendelkezésre álló képekből. Munkánkat számos előre elkészített sablon segíti. Az elkészült oldalak a *Gmail* tárhelyünkre kerülnek.

Az Intel Vietnámban

A *Vietnámi* kormány jóváhagyta az *Intel 605* millió dolláros befektetési tervét. Az új gyár *Ho Si Minh* városban épül és *Vietnám* legnagyobb számítástechnikai üzeme lesz, melyben processzorok és más számítógép alkatrészek készülnek majd. Jelenleg mintegy 260 amerikai cég képviselteti magát az országban összesen mintegy 1,45 milliárd dollár összbefektetéssel.

Az év olvasói díjai

Nem csak a magyarországi ***HUP***, de a ***Linuxquestions.org*** is kiosztja évről-évre olvasói szavazatok alapján a legjobb programoknak járó címet. Lássuk hát, idén mik ezek?

Kategória	Nyertes
Disztribúció	*Ubuntu*
Adatbázis-kezelő	*MySQL*
Irodai alkalmazás	*OpenOffice.org*
Böngésző	*Firefox*
Levelező	*Thunderbird*
Szövegszerkesztő	*vi/vim*
Szabad játék	*Frozen Bubble*
Ablakkezelő	*Fluxbox*
Munkakörnyezet	*KDE*
Multimédia (audio)	*amaroK*
Multimédia (video)	*mplayer*
Üzenőprogram	*Gaim*
Biztonsági program	*nmap*
Live disztribúció	*Knoppix*
Windows Linux alatt	*Wine*
Fájlkezelő	*Konqueror*
Fejlesztői környeze	*Eclipse*
Grafikus program	*GIMP*
Héj (shell)	*Bash*
Webszerkesztő	*Quanta*

➲ ***http://www.linuxquestions.org***

Négymagos Intel

Bemutatta a sajtónak két új ***négymagos*** processzorát az Intel. A ***Clovertown*** kódnevűt elsősorban szerverekbe, míg a ***Kentsfield*** kódnevűt munkaállomásokba szánják. Mindkét processzor ***65 nanométeres*** technológiával készült, azonban piacra csak 2007 első negyedévében kerülnek.
➲ ***http://www.intel.com/pressroom/archive/releases/20060307corp_a.htm***

Medve Zoltán
(e-medve@e-medve.hu)

2001-ben kezdett „Linuxolni”, de már korábban is ismerkedett a szabad szoftverek világával.
Ha éppen nem a gép előtt ül, akkor fotózgat, olvasgat vagy bicajozik.

Mi újság a rendszermag fejlesztése körül

John W. Linville bejelentette, hogy létrehozott egy ***git*** tárat az általa fejlesztett ***Fedora Core*** kernel számára. Az ezen a helyen található anyag alapja mindig hivatalos ***Fedora*** rendszermag, amit ***Linville*** kiegészít a ***kernel.org***-on található hálózati foltokkal. A cél egyrészt az, hogy a ***Fedora*** felhasználók is hozzáférhessenek a legújabb eszközök meghajtóihoz, amelyekre másként várniuk kellene, amíg bekerülnek a hivatalos kiadásba, másrészt pedig egy tesztelői közösség összekovácsolása, amely különösen a hálózati eszközök kipróbálására koncentrál majd.
Eric Dean Moore elvállalta a korábban a támogatott eszközök listáján nem szereplő ***LSI Logic MPT Fusion SCSI*** meghajtó karbantartását. Ez a meghajtó korábban egységesen támogatta a ***SCSI*** és ***Fibre Channel*** eszközöket, nemrég azonban szétvált több olyan meghajtóra, amelyek csak egy-egy felületet támogatnak. Az egészben az igazán ironikus momentum az, hogy bár ezt a váltást maguk a fejlesztők javasolták, az ***MPT Fusion*** melletti legfőbb érv, illetve hírnevének alapja egészen idáig az volt, hogy egyetlen felületen keresztül támogatja a ***SCSI***-t és ***Fiber Chanelt***.
Alessandro Di Marco Sluggard néven megkezdte egy olyan speciális fájlrendszer fejlesztését, amely a kernelforrás letöltését hivatott egyszerűbbé tenni. Ahogy a rendszermag növekszik, a forrás letöltése egyre több és több időt és helyet igényel. ***Sluggard*** erre egy viszonylag egyszerű megoldást talált ki: az ***rsync*** segítségével egyszerűsíti a letöltés folyamatát. A helyi gépen a felhasználó egy speciális fájlrendszert olvas, amely transzparens módon követi a forrásfájlok változásait, és a hozzáférés előtt automatikusan frissíti, amit kell. Ezzel a módszerrel elmondása szerint neki körülbelül 200 MB helyet sikerült spórolnia, de szerinte ennél többre is van esély attól függően, hogy milyen szolgáltatásokat akar valaki belefordítani a rendszermagba.
Andrey Volkov elkészítette az ***ST M41T85*** valósidejű órajelchipet támogató kódot. Munkájához ***Mark A. Greer ST M41T100*** chiphez készült meghajtóját használta fel alapként. Ami azt illeti, egyesek épen azért azt javasolták, hogy a két kódot egyesíteni kellene. Ugyanakkor a két hardver közötti számos apró eltérés miatt egyelőre nem világos, hogy melyik megközelítés a gazdaságosabb, célszerű volna ugyanis elkerülni a rengeteg ***#if*** ág megjelenését a forrásban. Akár az is kiderülhet tehát, hogy a két kód egyesítése nem hogy nem egyszerűsíti a helyzetet, hanem kifejezetten bonyolítja. Mindazonáltal mind ***Andrey*** mind ***Mark*** egyetért az egyesítéssel, ha ez tűnik a helyes útnak.
Mivel a ***Sun*** nyilvánossá tette a ***ZFS*** fájlrendszer forráskódját, ***Tarkan Erimer*** nemrég azt a kérdést kezdte feszegetni, vajon azt jelenti-e ez, hogy valamikor a jövőben ***Linux*** alatt is meg lehet valósítani a ***ZFS*** támogatását. Sajnos azonban ez nem tűnik túl valószínűnek, mivel a ***Sun*** a ***CDDL*** licenszet használta, amely megengedi, hogy az így kibocsátott kódot zárt forráskódú alkalmazásokhoz is linkelni lehessen. Ez pedig összeegyeztethetetlen a ***GPL*** elveivel. Hacsak a Sun úgy nem dönt, hogy mind ***CDDL***, mind ***GPL*** alatt kibocsátja a kódot, a ***ZFS*** támogatását legálisan nem lehet beilleszteni a hivatalos kernelfába. Annak persze továbbra sincs akadálya, hogy valaki a ***Sun*** kódja nélkül teljesen újraírja ennek a fájlrendszernek a kódját.
Ha valakinek netán kétségei lettek volna afelől, mit is gondol ***Linus Torvalds*** a ***CVS*** rendszerről, nos az megnyugodhat. Amikor nemrég valaki azt javasolta, hogy a rendszermag dokumentációjába meg kellene említeni a jelenleg létező ***CVS*** tárakról is, ***Linus*** a következőket válaszolta: ***„Semmi értelmét nem látom. A CVS úgy rossz, ahogy van, és bárki, aki ezzel tartja karban az általa fejlesztett kódot, fel kell készüljön azokra a nehézségekre, amelyek a kernelfával való egyesítés során majd felmerülnek. És ez nem holmi elmélkedés, hanem gyakorlati tapasztalat, amivel nekünk már évekkel ezelőtt tisztába kellett jönnünk, amikor hasonló dologgal próbálkoztunk. Éppen ezért nem akarok látni egyetlen külső CVS fát sem, és nem is vagyok hajlandó ezekkel semmilyen szinten foglalkozni."*** Hát ennyi.
A ***Linux*** fejlesztése során alkalmazott filozófia egyik alaptétele az, hogy a rendszenek nem kell egyetlen szabványhoz sem vakon alkalmazkodnia csupán azért, mert az a szabvány létezik. A szabványoknak értelmeseknek kell lenniük, ha pedig a ***Linux*** nem követ egy amúgy jól átgondolt előírást, akkor az ahhoz való visszatérésnek valamiféle gyakorlati haszonnal kell szolgálnia. Ez a dilemma a közelmúltban is felmerült, amikor ***Matthew Wilcox*** megpróbálta kijavítani a ***NO_IRQ*** definícióját, hogy az egyezzen a ***PCI*** szabvánnyal. ***Linus*** rámutatott, hogy ezt a szabványt maguk a hardvergyártók is évek óta figyelmen kívül hagyják, így a ***NO_IRQ*** definíciójának megváltoztatása számos jelenleg jól működő meghajtó működésképtelenné válásához vezethet. Ami azt illeti, a ***Linux*** kernel sokkal áttekinthetőbb lenne, ha a különböző architektúrákat legalább nagyjából fedésbe hoznák, illetve ha az összes meghajtót sikerülne úgy módosítani, hogy tükrözze ezt a változást. Ez lenne a leghelyesebb. Ezzel együtt ***Linus*** elutasította a javaslatot, mondván igaz ugyan, hogy ez eredeti felépítés hibás, a kijavítása azonban olyan mennyiségű járulékos problémát vonna maga után, ami megkérdőjelezi az egész akció értelmét.

Linux Journal 2006. 144. szám

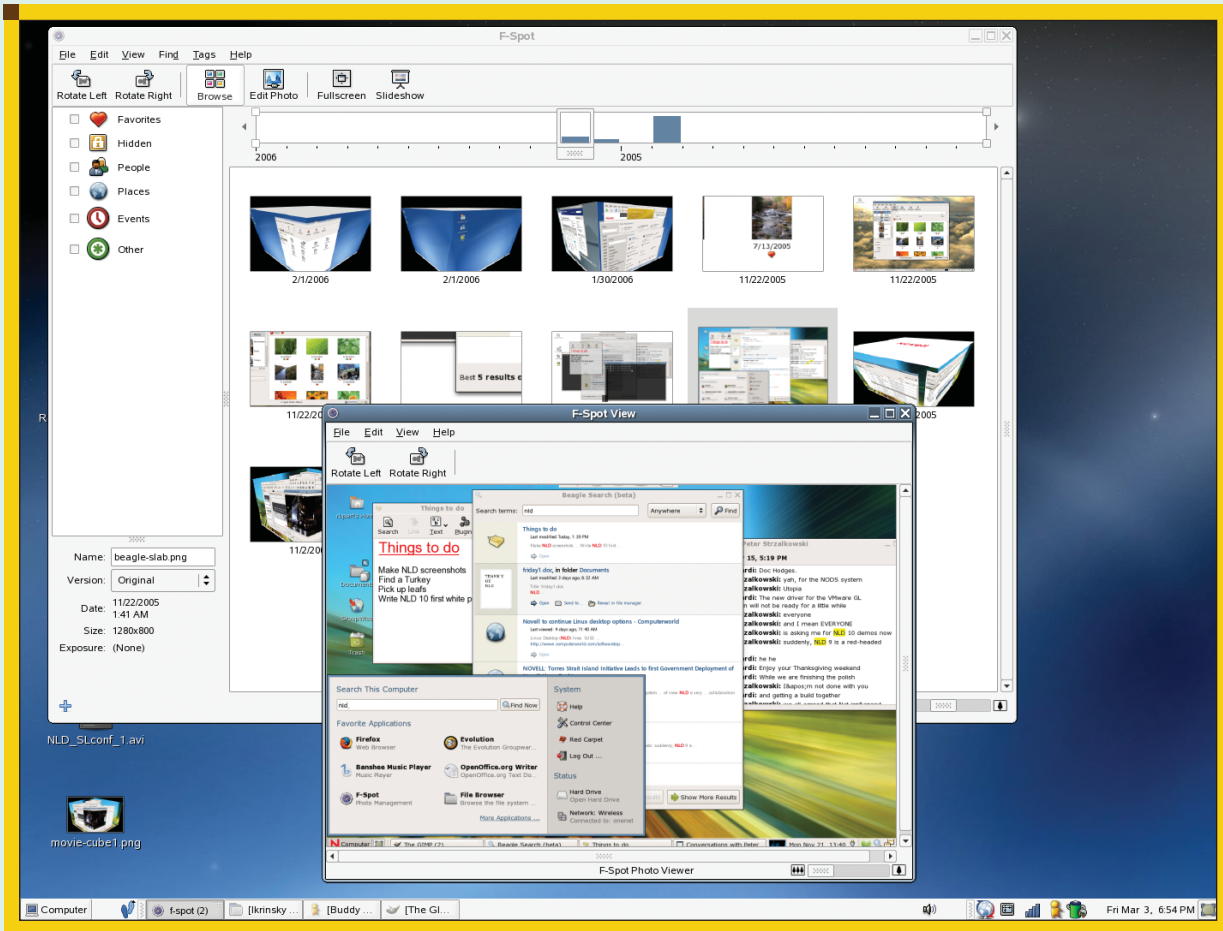
Zack Brown

A Novell újgenerációs Linux desktop megoldása átrendezi az asztali munkaállomások piacát

„A fejlesztés megkezdésekor az üzleti felhasználók számára elérhető legjobb asztali rendszer megalkotását tűztük ki célul”- ezekkel a szavakkal jelentette be a Novell a CeBIT-en újgenerációs vállalati asztali Linux-rendszerét.

A *SUSE Linux Enterprise Desktop* megjelentetésével a *Novell* beváltotta tavaly tett ígéretét, miszerint nemcsak a tapasztalt linuxosokat, hanem az üzleti felhasználókat is megcélzó asztali rendszert dob a piacra. A könnyű és egyszerű felhasználás problémáját oldotta meg és egy olyan asztali környezetet készített, amely a piac legfontosabb pontjának, az általános irodai dolgozóknak a kiszolgálására is alkalmas.

A *SUSE Linux Enterprise Desktop* néven piacra kerülő termék egy teljes körű asztali környezet, amelyben olyan technológiai újítások találhatóak, mint a továbbfejlesztett energiagazdálkodás, a beépített asztali keresőrendszer, a nagy teljesítményű grafikus felületek és a különféle alkalmazások nyílt forráskódú fejlesztésekből származó újdonságai. Ennek köszönhetően a *SUSE Linux Enterprise*



1. ábra A SUSE Linux Enterprise Desktop beépített fotó menedzsment könyvtára

2. ábra A SUSE Linux Enterprise Desktop 3 dimenziós megjelenítése

Desktop új mércét állít fel az üzleti asztali környezetek világában: egyszerű használhatóságot, nagy teljesítményű, beépített irodai funkciókat és sziklaszilárd stabilitást kínál a termelékenység növeléséhez – a hagyományos asztali termékek árának töredékéért.

A *SUSE Linux Enterprise Desktop* tervezése közben a *Novell* több száz felhasználói felmérést végzett és közel 1500 órányi videót forgatott arról, hogyan is használják az irodai dolgozók munkaállomásaikat. A *SUSE Linux Enterprise Desktop* minden egyes funkcióját – legyen az akár az asztali beállítások módosítása, fájlok keresése, alkalmazások elindítása, külső eszközök (például *USB*-s meghajtók) elérése, az Internet használata vagy éppen csatlakozás a helyi és a vezeték nélküli hálózatokhoz – gondosan tesztelték. A körültekintő tervezésnek köszönhetően a lehető legjobb teljesítményre számíthatnak a céges, vagy intézményi irodai felhasználók.

Nagy teljesítményű asztali funkciók

A *SUSE Linux Enterprise Desktop* része a gyors rendszertöltési technológia, a továbbfejlesztett energiagazdálkodás és a rendszer felfüggesztése, a vezeték nélküli eszközök még kifinomultabb integrációja, *Bluetooth*-támogatás és az alkalmazások biztonságosabb használata a *Novell AppArmor* segítségével. Szemben a hagyományos megoldásokkal, ahol további szoftverek vásárlására van szükség egy hasonló funkcionalitású asztali rendszer kialakításához, a *SUSE Linux Enterprise Desktop* minden szükséges komponenst tartalmaz: egy megoldásban kínál biztonságos operációs rendszert, irodai csomagot, webböngészőt, kliensprogramot azonnali üzenetküldéshez, fájlkezelőt, fotó- és zeneszerkesztő szoftvert, beépített asztali keresőt és automatizált szoftverfelügyeleti eszközöket. A fájl- és tartalomkeresést teljes egészében a felhasználói környezetbe integráló *SUSE Linux Enterprise Desktop* használatakor csupán egy kulcsszó megadásával azonnal elérhetők a dokumentumok, grafikák, e-mail üzenetek és beszélgetések, weboldalak és minden más tartalom. A *Novell* felhasználói tesztjein a válaszadók túlnyomó része jobbnak találta a *SUSE Linux Enterprise Desktop* keresési lehetőségeit a hagyományos megoldásokhoz képest.

Integrált OpenOffice 2.0

A *SUSE Linux Enterprise Desktop* az első olyan, teljes mértékben támogatott vállalati asztali környezet, amelynek része az *OpenOffice.org 2.0*, a vezető nyílt forráskódú irodai megoldáscsomag. Az *OpenOffice* része egy táblázatkezelő, egy üzleti prezentációkészítő és egy szövegszerkesztő is. Az *OpenOffice.org Novell*-változata számos *Visual Basic* makrót kezel és ezzel megszünteti az *OpenOffice.org* és a *Microsoft Office* közötti korábbi kompatibilitási problémát. Az *OpenOffice.org 2.0* képes megnyitni és elmenteni a *Microsoft Office* formátumban készült dokumentumokat, így az *Excel* adattáblákat is, és jelenleg ez az egyetlen olyan irodai csomag, amely teljes mértékben támogatja a dokumentumfájlok új, nyílt formátumát, az *OpenDocument* fájlformátumot. Mivel az *OpenDocument* egy nyílt szabvány, amelyet a nyílt forráskódú közösség tart karban, használatával megszűnik a gyártóhoz való kötöttség, hiszen a táblázatokba, dokumentumokba és bemutatókba mentett adatok szabadon elérhetők minden *OpenDocumentet* támogató alkalmazásból.

Együttműködés a Windows-zal, Office-szal, Exchange-dzsel

A *Linuxra* való átállás megkönnyítése érdekében a *SUSE Linux Enterprise Desktop* együttműködik a *Microsoft Windowszal*, a *Microsoft Office*-szal és a *Microsoft Exchange*-dzsel. Támogatja az összes szokásos hálózati és nyomtatási protokollt és problémamentesen integrálódik a meglévő *Active Directory* környezetekbe. Az *OpenOffice.org 2.0* olvassa és írja a *Microsoft* dokumentumformátumait, a *Novell Evolution* e-mail kliens pedig beépített csatolót tartalmaz a *Microsoft Exchange*-hez. Az *IBM Workplace Client Technology* termékéhez ígért *Lotus Notes Application* bővítőmoduljával a *Lotus Notes* és *Domino* felhasználók elérhetik a *Notes*-alkalmazásokat, -adatbázisokat és -levelezést. A *Novell GroupWise* pedig természetesen teljes mértékben támogatott a *SUSE Linux Enterprise Desktop* alatt is.

Teljes körű ügyfélmegoldás

A *SUSE Linux Enterprise Desktop* egyszerűen üzembe helyezhető és üzemben tartható. Az előfizetési időszak alatt a Novell a felhasználók számára teljes körű hozzáférést biztosít a *Novell Customer Centerhez*, egy on-line portálhoz, amelyen azonnal, automatizált módon elérhetők az új szoftverkiadások, javítások és műszaki frissítések. A nagyobb rendszerek támogatása érdekében a *SUSE Linux Enterprise Desktop* maximálisan integrálódik a *Novell ZENworks Linux Management* megoldásával.


Linuxvilág
www.linuxvilag.hu

Interjú a madárral (1. rész)

A közelgő május 10-i Madarak és Fák Napja alkalmából felkerestük madarunkat, Pápaszemes Tóbiást az Állatkert Kisziklájának fája alatt, és arról érdeklődtünk, hogyan telt a napja.

H.L.: Kedves Tóbiás, bizonyára hallotta a hírt, mely szerint hasonló alakú fajrokonait a japán állatkert szigorú gondozói napi többszöri testgyakorlásra kényszerítették, hogy a bőrük alatt felhalmozott szalonnaréteg téli vastagságából veszejtsenek. Érték Önt és társait hasonló atrocitások a tavasz beköszöntével?

P.T.: Minálunk úgy tartják, a kövérség ott kezdődik, amikor barátaid azt kezdik pedzegetni, régen mintha magasabb lettél volna… Amíg megközelítem az Ön indiai fajrokonának, *Gul Mohammednek* 57 centiméteres magasságát, nem aggódom a méreteim miatt.

H.L.: Apropó, magasság. Minap olvastam, hogy 40 millió éves óriáspingvin-csontokat találtak Új-Zélandon, és e lelet valaha másfél méterre nyúlt madarat testesített meg – igaz, többet nyomott 100 kilónál.

P.T.: Hát az Eiffel-torony magassága is változik a hőmérséklet függvényében… Amúgy is nehézkes négy ujjban végződő bokán egyensúlyozni ezt a pár kilós frakkot, de az alacsony termetnek is megvannak az előnyei. Például minket ér utoljára az eső.

H.L.: Egyik ismerősöm szerint ne bízzak feltétlen olyan egyénben, aki rövid lábakon áll, mert túl közel van az agya az ülepéhez. Önnek azonban világhírű az ülepe, pontosabban az eme testtájékát elhagyó terméke. Többféleképpen hasznosítják, guano termelésében, erősáramú vezetékek forgalmazásában, Nobel-díj szerzésében vagy éppen mesékben…

P.T.: A hatótávolság szempontjából megoszlanak a székletemmel szembeni megítélések. Ha szemben székelek egy erősáramú vezetékkel, szennyezéses meghibásodásról panaszkodnak. Ha azonban a nyomásviszonyokra koncentrálok erősen, nem az irányra, továbbá kihagyom számításaimból kloákám átmérőjét – máris garantáltan Nobel-díjhoz segítem a madárlesőket. Sőt egy ausztrál mesekönyvbe is beszékeltem magam nagy dolgaim végeztével.

Szívesen faggattam volna tovább madarunkat a magasság, tömeg és Nobel-díj összefüggéseiről, azonban ő hirtelen hátat fordított (én 40 centit arrébb ugrottam), és eltotyogott uzsonnaosztásra, hogy heringadagját belakomázva újabb nyomást gyakorolhasson a főemlősök ama szervére, mi testsúlyuk 2 százalékát teszi ki, nevezetesen az emberi agyra. Pedig még annyi kérdésem lett volna napelemes pingvinhűtésről, szürkehályog-madárműtétről, álcázott homoszexualitásról – no, majd legközelebb.

Halusz Léna

Itt az idő, hogy mindent újragondolj!

Az EFL alapjai

Az EFL egy függvénykönyvtár, és hasonlóan a Gnome, és a KDE keretrendszereihez, ez az Enlightenment DR17 (E17) alapja. Minek még egy, hiszen már van nekünk GTK, QT, és még sorolhatnánk a grafikus eszköztárakat?! Ez sem lesz szebb, gyorsabb, vagy egyszerűbb! Nos, aki így vélekedik, annak itt az idő, hogy mindent újragondoljon.

„*Itt az idő, hogy mindent újragondolj!*" Ez ugyanis a szlogen. És valóban, az *E17*-et készítő csapatnak ez sikerült is. Az *EFL* tagjai úgy lettek tervezve, hogy a lehető legkevesebb erőforrást használják fel számítógépünkből, egyszerűen kezelhetőek legyenek mind programozói, mind felhasználói szemszögből, és nem utolsó sorban elálljon a lélegzetünk az grafikus effekteknek köszönhetően. Jó, jó, hiszen mindenki ezt mondja a saját programjáról! Nos, az a helyzet, hogy a csapat egyik tagja egy 100 Mhz-es *Pentium I*-esen teszteli rendszeresen az alkalmazásokat, elég gyorsak-e azon is, és meg van elégedve az eredménnyel.
Mik is alkotják hát az *EFL*-t? Kezdjük az alapokkal.

Evas

Az *Evas* egy nagyon jól portolható és okos képernyő-leképező. Gyorsaságát főleg neki köszönheti az egész *EFL*, hiszen ezen alapul. Nem csak *X* szerveren használhatjuk, hiszen fut *Framebufferen*, *DirectFB*-n, *OpenGL*-lel is gyorsíthatjuk működését, sőt „okostelefonokra" és *PDA*-kra is létezik. Ellát minket jó minőségű képátméretezéshez, élsimított (anti-aliased) betűk létrehozásához, színátmenetek készítéséhez, egyszerű vonalak előállításához való eszközökkel, és ez még nem minden. Az embernek amiatt sem kell törnie a fejét, hogy hány színt képes kezelni az *X* szerverünk. Elég okos hozzá, hogy megtalálja a maximálisan megjeleníthető színek számát, így akár egy fekete-fehér kijelzőn is fut. A megfelelő *Doxygen* dokumentáció a ➲ *http://enlightenment.sourceforge.net/doxy/evas/* címen érhető el.

Ecore

Ez a könyvtár felelős az eseménykezelésekért, a húzd-és-ejtsd (*drag-and-drop*) rendszerért, a kijelölések kezeléséért, és az alapszintű rendszerekért. Nagyon jól bővíthető, és gyors. Használhatjuk alapjául az *Evast*, de közvetlen *Framebufferen* és *X* szerveren is fut. Tartalmaz modulokat szövegkonverzióra, folyamatok közti kommunikációra, kliens-kiszolgáló kapcsolatokra, és még sok minden másra is.

Embryo

Az *Embryo* egy *C* szerű szintaxissal rendelkező szkriptnyelv. Minden *Embryo* program egy védett környezetben (*sandbox*) fut, ami meggátolja abban, hogy rendszerünkben kárt tegyen, viszont ezáltal nem alkalmazható olyan feladatokra, mint amilyeneket *Bash*, *Perl*, vagy *Python* programokkal oldanánk meg. Ebből a védett környezetből nem tud fájlokat kezelni, memóriaterületet foglalni, más folyamatokat elérni, és a hálózathoz csatlakozni sem. Az *Embryo* kód egy nagyon kicsi, és gyors virtuális gépben fut. Az egész virtuális gép kódja nem több, mint háromezer sor, ami szép teljesítmény.

Edje

Az *Edje* egy olyan könyvtár, ami megadja a lehetőséget, hogy a grafikus felhasználói felületet elválasszuk

a tényleges munkát végző kódtól. Ami különlegessé teszi, az az a tulajdonsága, hogy mi magunk csinálhatjuk meg a grafikus elemeket, amiket később elláthatunk olyan *Embryo* kódokkal, amik azt animálják, vagy éppen átalakítják a képeket. Így készíthetünk akár *Flash* animáció szerű programokat is. Az *Embryo* forrásfájlokat a képekkel együtt *EET* kiterjesztésű állományokban tartja.

EET, EDB

A két rövidítés egy-egy állománytípust jelöl. Az ilyen fajta fájlok bináris formában tartalmaznak kulcsérték párokat. Az *EET zip*-szerű, tehát tömörítheti is a bevitt adatot, és inkább fájlok tárolására való, ám egy *zip*-nél jóval egyszerűbb felépítésű.
Az *EDB* fájlok beállításokat tárolnak. A kulcs-érték párokhoz kapcsolódik még egy típus is, ami lehet `int` (egész), `str` (karakterlánc), `float` (lebegőpontos szám), vagy `data` (bináris adat).
Az *EET Doxygen*-je a ➲ *http://enlightenment.sourceforge.net/doxy/eet/* címen érhető el, az *EDB*-nek viszont nincs ilyenje, ezért a jó minőségű, megjegyzésekkel tűzdelt fejállományt kell használnunk.

Epeg, Epsilon

Két képkicsinyítőről, villámnézet (*thumbnail*) készítőről van szó. Az *Epeg* (a készítők szerint) a világ leggyorsabb *JPEG* kicsinyítő könyvtára. Az *Epsilon* képes saját maga kezelni PNG formátumú képeket is, és az Imlib2 könyvtárral együttműködve ez a sor még kiegészül az XCF és TIFF formátumokkal is. Az Epeggel kombinálva fantasztikus sebességgel készít villámnézetet még a nagy felbontású képekből is.
Az *Epeg Doxygen* dokumentációja a ➲ *http://enlightenment.sourceforge.net/doxy/epeg/* oldalon érhető el, az *Epsiloné* pedig ugyanitt de a /epsilon könyvtár alatt.

Etox

Az *Etox* nagyon hasznos, ha szövegek formázásáról van szó. Képes mindenféle effektre, mint szövegeket színezése, képek körülfolyatása, átméretezése, rétegekbe rendezése, vagy akár mozgatása. Mindezt egy sima ablakban, szövegszerkesztő nélkül. Szintén nincs *Doxygenje*, de a fejállományok itt is segítségünkre vannak.

Emotion

Az *Emotion* mozgóképek, videók kezelésére való. Alapjául a *libxine*, vagy a *Gstreamer* használható, így minden formátumot képes lejátszani, amit azok is. Segítségével akár 18 sorban írhatunk egy *DVD* lejátszót. Nincs *Doxygen* dokumentációja, a fejállományokat kell tanulmányoznunk.

EWL

Ez az *EFL* grafikus eszköztára. Bár létezik egy *ETK* nevű hasonló funkciójú könyvtár is, arról egy szót sem ejt az *Enlightenment* hivatalos oldala. Elég nehézkes a *Doxygen* böngészése, sok mindent nehéz benne megtalálni, ezért ajánlom a közvetlen fejállományokat tanulmányozni. A *Doxygen* elérhető a ➲ *http://enlightenment.sourceforge.net/doxy/ewl/ címen.*

Lássuk mindezt a gyakorlatban!

Most, hogy megismerkedtünk az *EFL* építőelemeivel, lássuk, hogyan is kell használni őket. A következő példákban az *EWL*, *EDB*, és *EET* könyvtárak lesznek a főszereplők. A programokat a

```
gcc -o program prog.c
➥`ewl-config -cflags -libs`
➥`eet-config -cflags -libs`
➥`edb-config -cflags -libs`
```

paranccsal fordíthatjuk le.

1. Lista hello.c

```
 1: #include <Ewl.h>
 2: #include <Eet.h>
 3: #include <Edb.h>
 4: #include <stdio.h>
 5:
 6: void vege__ (Ewl_Widget *hivo, void
    ➥*esemeny, void *extra) {
 7: ewl_widget_destroy(hivo);
 8: ewl_main_quit();
 9: }
10:
11: void eet__ (Ewl_Widget *hivo, void *esemeny,
    ➥void *extra) {
12: char *fajlnev;
13: char *szoveg;
14: Eet_File *fajl;
15: Ewl_Widget *bevitel;
16:
17: bevitel = (Ewl_Widget *)extra;
18: fajlnev = ewl_text_text_get
    ➥(EWL_TEXT(bevitel));
19:
20: fajl = eet_open(fajlnev,
    ➥EET_FILE_MODE_WRITE);
21: if(!fajl) return;
22:
23: szoveg = "Ez egy mondat, amit eet-vel írtunk
    ➥ki.";
24: eet_write(fajl, "eleresi_kulcs", szoveg,
    ➥(strlen(szoveg)+1), 1);
25: eet_close(fajl);
26: }
27:
28: void edb__ (Ewl_Widget *hivo, void *esemeny,
    ➥void *extra) {
29: char *fajlnev;
30: char *szoveg;
31: E_DB_File *fajl;
32: Ewl_Widget *bevitel;
33:
34: bevitel = (Ewl_Widget *)extra;
35: fajlnev = ewl_text_text_get
    ➥(EWL_TEXT(bevitel));
36:
```

1. Lista folytatás

```
37: fajl = e_db_open(fajlnev);
38: if(!fajl) return;
39: e_db_property_set(fajl, "program", "Hello");
40: e_db_str_set(fajl, "/eleresi/kulcs", "Ez egy
    ➥mondat, edb-ben tárolva");
41: e_db_close(fajl);
42: }
43:
44: int main (int argc, char **argv) {
45: Ewl_Widget *ablak, *vbox, *hbox,
46:       *cimke, *bevitel,
47:       *gomb_eet, *gomb_edb;
48:
49:
50: if(!ewl_init(&argc, argv)) return 1;
51:
52: /** Az ablak **/
53: ablak = ewl_window_new();
54: ewl_window_title_set(EWL_WINDOW(ablak),
    ➥"Hello, mi?");
55: ewl_window_name_set(EWL_WINDOW(ablak),
    ➥"EwlDemo");
56: ewl_window_class_set(EWL_WINDOW(ablak),
    ➥"EwlDemo");
57: ewl_object_size_request(EWL_OBJECT(ablak),
    ➥300, 50);
58: ewl_callback_append(ablak,
    ➥EWL_CALLBACK_DELETE_WINDOW, vege__,NULL);
59:
60: /** A vbox tároló **/
61: vbox = ewl_vbox_new();
62:
63: /** A hbox tároló **/
64: hbox = ewl_hbox_new();
65:
66: /** A címke **/
67: cimke = ewl_text_new();
68: ewl_text_text_set(EWL_TEXT(cimke), "A fájl
    ➥elérési útja:");
69: ewl_object_alignment_set(EWL_OBJECT(cimke),
    ➥EWL_FLAG_ALIGN_RIGHT);
70:
71: /** A beviteli mező **/
72: bevitel = ewl_entry_new();
73:
74: /** Az eet-be író gomb **/
75: gomb_eet = ewl_button_new();
76: ewl_button_label_set(EWL_BUTTON(gomb_eet),
    ➥"Eet-be írás");
77: ewl_callback_append(gomb_eet,
    ➥EWL_CALLBACK_CLICKED, eet__, bevitel);
78:
79: /** Az edb-be író gomb **/
80: gomb_edb = ewl_button_new();
81: ewl_button_label_set(EWL_BUTTON(gomb_edb),
    ➥"Edb-be írás");
82: ewl_callback_append(gomb_edb,
    ➥EWL_CALLBACK_CLICKED, edb__, bevitel);
83:
84: /** Elemek hozzáadása a tárolókhoz **/
85: ewl_container_child_append
    ➥(EWL_CONTAINER(ablak), vbox);
86: ewl_container_child_append
    ➥(EWL_CONTAINER(vbox), cimke);
87: ewl_container_child_append
    ➥(EWL_CONTAINER(vbox), bevitel);
88: ewl_container_child_append
    ➥(EWL_CONTAINER(vbox), hbox);
89: ewl_container_child_append
    ➥(EWL_CONTAINER(hbox), gomb_eet);
90: ewl_container_child_append
    ➥(EWL_CONTAINER(hbox), gomb_edb);
91:
92: /** Elemek megjelenítése **/
93: ewl_widget_show(ablak);
94: ewl_widget_show(vbox);
95: ewl_widget_show(hbox);
96: ewl_widget_show(cimke);
97: ewl_widget_show(bevitel);
98: ewl_widget_show(gomb_eet);
99: ewl_widget_show(gomb_edb);
100:
101:    ewl_main();
102:    return 0;
103: }
```

1-5. sorban betöltjük a szükséges fejállományokat. Az ***Ewl.h***, ***Eet.h***, ***Edb.h*** fájlok értelemszerűen az ***EWL***, ***Eet***, és ***EDB*** könyvtárakhoz kellenek.
A 7-43. sorig a visszahívásokat definiáljuk, ezekre később térek ki.
A `main` függvény a 45. sorban kezdődik. A paraméterekre az ***EWL*** inicializálásakor van szükségünk, ugyanis van néhány opció, amiket a könyvtár maga kezel le. Ezek az opciók a téma beállításához, a hardveres ***OpenGL***, vagy szoftveres ***X11*** leképezés megválasztásához használhatók. Ez az inicializálás az 51. sorban történik meg.
Az 55. sorban hozzuk létre az ablakunkat. A függvény `Ewl_Widget *` típussal tér vissza. Ez azért van, mert egy objektumnak vannak saját, és általános tulajdonságai. Az általános tulajdonságokat minden egyes objektumnál ugyanazokkal a függvényekkel állítjuk be, így azok ugyanolyan típusú mutatókat várnak paraméterül. Az átalakítást a fejállományokban definiált állandók végzik. Általános tulajdonságnak hívhatjuk az `ewl_object` és `ewl_callback` kezdetű függvényeket.
Az objektum saját tulajdonságait csak egy adott típusra állíthatjuk be. Például egy beviteli mezőre nem állíthatunk be ablakosztályt, ellenkező esetben az alkalmazásunk futtatáskor hibával áll le. Ablakunkhoz tartozó saját tulajdonságokat az 55-57. sorokban

állítjuk be. Az `ewl_window_title_set` értelemszerűen az ablak címét állítja be, esetünkben `Helló, mi?`-re. Az 56-57. sorokban beállított tulajdonságok az ablakkezelőnek szükségesek, például az ***E17*** ez alapján társítja az ikonokat futás közben az alkalmazásokhoz. Az 58. sorban beállítjuk az ablak méretét. Ezt a függvényt bármilyen objektumra meghívhatjuk, ahogy az 59. sorban található hívást is, ami visszahívást rendel az ablakunkhoz. Láthatjuk, hogy ehhez a híváshoz nem kell átalakítanunk a mutató típusát. Az `ewl_callback_append` függvény második paramétere a visszahívás típusa, jelen esetben az ablak bezárása. A harmadik kapcsoló a meghívandó függvény, a negyedik pedig egy extra objektum, amit átadhatunk. Esetünkben erre nincs szükség, így `NULL`.
Így el is érkeztünk a visszahívásokhoz, ugorjunk tehát a 7. sorra, a `vege__` függvény kezdetéhez. Egy visszahívás három változót vár paraméterül: az első, `Ewl_Widget` típusú maga a hívó. A második az esemény típusa, a harmadik pedig a fent említett „extra”. Minden visszahívás `void` típusú, tehát nem tér vissza értékkel.
A `vege__` függvény `ewl_widget_destroy` hívása az ablak, és összes gyermeke megsemmisítéséről gondoskodik. Az `ewl_main_quit` hívás az ***EWL*** megjelenítési ciklusát állítja le. Ezt a ciklust az `ewl_main` hívással indítjuk majd el.
A 62. és 65. sorban létrehozott `vbox` és `hbox` tárolókhoz adott elemek

1. ábra A futó program

függőlegesen, illetve vízszintesen egymás alatt, illetve mellett lesznek. Használhatnánk bonyolultabb elrendezési mechanizmust alkalmazó tárolókat is, de ezek céljainknak tökéletesen megfelelnek.
Az ***EWL*** `Text` típusa egy nem írható szövegmezőt, címkét jelöl. Ezt hozzuk létre a 68. sorban. A címke szövegét a következő sorban állítjuk be, de azt akár az `ewl_text_new` függvénynek átadott mutatóval is megtehetnénk.
Az `ewl_object_alignment_set` hívás szintén minden típusra alkalmazható, és az objektum elhelyezkedését szabja meg, az átadott állandó segítségével. A mi esetünkben az ablak jobb oldalára igazítja a widgetet.
A 73. sorban inicializált `Entry` típusú objektum egy beviteli mezőt jelöl. Vele egyelőre nincs más dolgunk.
A 76. és 81. sorban két gombot hozunk létre, amik szinte ikrei egymásnak, csak nevükben, feliratukban térnek el. A gombok címkéjét nem `Text` típusúra átalakítva kezeljük, mint a beviteli mezőket, hanem egy saját függvény segítségével.
A gombok visszahívásai, az `eet__` és `edb__` függvények a gombokra való kattintáskor hívódnak meg, és ***eet*** illetve ***edb*** adatformátumban kiírnak egy mondatot a bevitel objektumba beírt fájlnevet felhasználva.
Akkor tehát vizsgáljuk meg az `eet__` függvényt. Az első négy sorban a szükséges változókat deklaráljuk, majd a következő sorban az extra változót típuskényszerítéssel átalakítjuk `Ewl_Widget`-re, így a változó típusát a 19. sorban tovább tudjuk kényszeríteni `Ewl_Text`-re, ezt végzi igazából az `EWL_TEXT` (bevitel) kifejezés.
A 21. sorban nyitjuk meg a beviteli mezőbe írt névhez tartozó fájlt, majd rögtön az utána lévő sorban leellenőrizzük, hogy ez a művelet sikerült-e.
A 25. sorban kiírjuk a mondatunkat: az második paraméter az elérési kulcs, amivel később mondatunkra hivatkozni tudunk majd, a harmadik maga a tárolni kívánt adat. A negyedik kapcsoló a tárolni kívánt adat hossza, esetünkben a karakterláncé, beleértve a végét jelző 0 karaktert is. Az utolsó várt kapcsoló 0 vagy 1 értéket kaphat aszerint, hogy szeretnénk-e, hogy adatunk tömörítve legyen. Mi természetesen akarjuk! A 26. sorban végül lezárjuk eet fájlunkat.
Az `edb__` függvényben az egyetlen új dolog az ***edb*** fájlok kezelése.
A 38. sorban megnyitjuk a fájlt mindenféle műveletre.
Az `e_db_open_mode` függvénnyel meghatározhatnánk a megnyitás módját az `open` függvényhez hasonlóan, az `O_` állandókkal.
A 39. sorban egy „tulajdonságot” állítunk be. Ez a mező nem szerepel a többi között, ha lekérjük a kulcsok listáját, és nem elérhető egy egyszerű, valamilyen `get` hívással. Mondhatni „rejtett”, de mégsem. Arra használható, hogy az alkalmazások megtudják: miféle adatbázissal van dolguk.
Tudni kell, hogy nem csak egy ilyen állítható be, ilyen módon sok mindent tárolhatunk.
A következő sorban viszont egy „igazi” mezőt készítünk. Az elérési kulcs, mint láthatjuk, némileg más formátumú, mint az ***eet*** esetében, azonban ez nem az adatformátumtól függ. Az ***E17***, és az ***EFL***-t használó programok inkább ez utóbbi formát használják, így egyszerűen csoportosíthatóak a beállítások, tehát érdemes ez utóbbira szavazni, bár ízlés szerint bármilyen kulcs használható.

2. Lista eet_olvas.c

```
 1: #include <Edb.h>
 2: #include <stdio.h>
 3:
 4: int main (int argc, char
    ➥**argv) {
 5: int kulcs_sz, i;
 6: char **kulcsok;
 7: E_DB_File *fajl;
 8:
 9: if(argc != 2) { printf("Használat:
    ➥\n\t%s [fáljnév]\n", argv[0]);
10: return 1; }
11:
12: fajl = e_db_open_read(argv[1]);
13: if(!fajl) { printf("Nem tudtam megnyitni
    ➥a fájlt"); return 1; }
14: kulcsok = e_db_match_keys(fajl,"*",
    ➥&kulcs_sz);
15:
16: for (i = 0; i < kulcs_sz; i++) {
17: printf("típus: %s ", e_db_type_get(fajl,
    ➥kulcsok[i]));
18: printf("| %s : \"%s\"\n", kulcsok[i],
    ➥e_db_str_get(fajl,kulcsok[i]));
19: }
20:
21: e_db_close(fajl);
22: free(kulcsok);
23: return 0;
24: }
```

3. Lista edb_olvas.c

```
 1: #include <Edb.h>
 2: #include <stdio.h>
 3:
 4: int main (int argc, char **argv) {
 5: int kulcs_sz, i;
 6: char **kulcsok;
 7: E_DB_File *fajl;
 8:
 9: if(argc != 2) { printf("Használat:\n\t%s
    ➥[fáljnév]\n", argv[0]);
10: return 1; }
11:
12: fajl = e_db_open_read(argv[1]);
13: if(!fajl) { printf("Nem tudtam megnyitni
    ➥a fájlt"); return 1; }
14: kulcsok = e_db_match_keys(fajl,"*",
    ➥&kulcs_sz);
15:
16: for (i = 0; i < kulcs_sz;
    ➥i++) {
17: printf("típus: %s ", e_db_type_get(fajl,
    ➥kulcsok[i]));
18: printf("| %s : \"%s\"\n",
    ➥kulcsok[i],
    ➥e_db_str_get(fajl,kulcsok[i]));
19: }
20:
21: e_db_close(fajl);
22: free(kulcsok);
23: return 0;
24: }
```

A 85. sortól a tárolókhoz adjuk a widgeteket, majd a 93. sortól megjelenítjük őket. Figyeljük meg, hogy az objektumok sorrendje futáskor nem a megjelenítési sorrenden múlik, hanem a tárolókhoz adáskor határozzuk meg azt.
Végül a 101. sorban elindítjuk az ***EWL*** megjelenítési ciklusát, amit majd az ablak lezárásakor léptetünk ki, hogy programunk befejeződjön.

Eet fájlok olvasása

Az ***Eet*** és ***Edb*** könyvtárak telepítésekor megkapjuk az ilyen típusú fájlok kezeléséhez szükséges programokat is, ***eet***, és `edb_ed` parancsok formájában. Az ***Eet*** fájlok, mint említettem inkább más állományok tárolására szolgálnak, hasonlóan egy ***zip*** tömörítésű fájlhoz, ezért az ***eet*** program a kulcsokat fájlnevekként kezeli, tartalmukat nem a képernyőre írja, hanem az kulcs által meghatározott fájlba. Ennek kiküszöbölésére, és annak szemléltetésére, hogy hogyan lehet ***C*** programból ezeket a fájlokat kezelni, íme egy újabb példa (*2. Lista*).
A 12. sorban megnyitjuk a programnak átadott fájlt, majd a 14. sorban lekérjük az elérhető kulcsok listáját. A második paraméter egy szabályos kifejezés, amivel meghatározzuk, hogy mi alapján akarjuk szűrni az eredményül kapott kulcsokat. Jelen esetben nincs ilyenről szó, így megteszi minden kulcs.
A 16-18. sorban végigmegyünk a kulcsokon, és egyenként kiírjuk őket

```
kulcs : "érték"
```

formában. Az értékeket az `eet_read` függvénnyel kapjuk meg. Harmadik paraméternek egy mutatót vesz, amibe beleírja a visszatérési adat méretét. Most erre nincs szükségünk, így nyugodtan lehet `NULL`.
A 22. sorban, ahogy azt illik, felszabadítjuk a kulcsokat tartalmazó tömb által lefoglalt memóriát, és a 23. sorban befejezzük a program futását. Az ***Eet Doxygen*** külön felhívja a figyelmet, hogy egyenként a tömbben lévő mutatókat ne szabadítsuk fel!

Edb fájlok olvasása

Mint említettem, létezik egy `edb_ed` nevű program, amivel az *edb* fájlokat kezelni tudjuk. A létező kulcsok listáját, és azok típusát például az

```
edb_ed valami.db match "*"
```

paranccsal kérhetjük le, ahol a zárójelek között egy szabályos kifejezést adhatunk meg.
Ha viszont egy programból szeretnénk ezeket a fájlokat kezelni, egy hasonló kódra lesz szükségünk (*3. Lista*). Csak olvasására megnyitni egy fájlt az `e_db_open_read` függvénnyel tudunk, ahogy azt láthatjuk a 11. sorban.
A 14. sorban lévő `e_db_match_keys` szintén egy szabályos kifejezést, és egy egész mutatót vár, előbbit a kívánt kulcsok szűrésére, utóbbit a visszaadott kulcsok számának tárolására használja.
A 17. sorban lekérjük a kulcshoz tartozó típust, és kiírjuk a képernyőre. A 18. sorban aztán karakterlánc típusúként kérjük le az adatot, és a kulccsal együtt kiírjuk a képernyőre. Így a teljes kimenet elemenként így néz majd ki:

```
típus: str | kulcs : "érték"
```

A 21. sorban lezárjuk a fájlt, utána felszabadítjuk a kulcsokat tartalmazó tömb által lefoglalt memóriaterületet, és befejezzük a programunkat.

Összegzés

Az *E17* és az *EFL* úgy lettek tervezve, hogy ha bármire szükség van a fejlődéshez, egyszerűen lehessen őket bővíteni. Jelenleg mindkét komponens aktív fejlesztés alatt áll, így folyamatosan változik.
Az *Enlightenment* hivatalos oldalán (➲ *http://enlightenment.org*) lévő könyvek már mind elavultak, így csak a *Doxygenre*, és a fejállományokra bízhatjuk magunkat.
Jelenleg nincsen stabil verzió egyik *EFL* könyvtárból sem, így a *CVS* fából kell őket kinyernünk, és lefordítanunk, de létezik néhány gyakran frissülő hely, ahol bináris csomagokat érhetünk el Debian rendszerekhez. A *Gentoo Linux Portage* fájában alapból benne vannak az *EFL* csomagok, telepítési útmutató ért keressük fel a ➲ *http://gentoo-wiki.org*-ot.
Található továbbá egy nem-hivatalos tár, ahová éjjelente kerülnek fel a napi *CVS* lenyomatok *.tar.gz* formátumban, így nem kell beleásnunk magunkat a *CVS* kezelésébe, és egyszerűen nekiállhatunk a fordításnak. A komponenseket a következő sorrendben kell lefordítanunk:

eet, edb, evas, ecore, embryo, imlib2, edje, e(vagy enlightenment), epeg, epsilon, esmart, entrance, ewl

Azt hiszem ennyi kiindulásnak elég volt, talán még sok is. Akinek van kedve fejleszteni programokat az *EFL*-re, tegye bátran, a csapat szívesen lát mindenkit!

Párkányi Péter
(peter.parkanyi@gmail.com)
1991-ben születtem, és idén elsős vagyok egy pécsi gimnáziumban. Szeretem a jó zenét, szeretek kerékpározni, és kosárlabdát is szívesen játszom. Jelenleg Gentoo Linuxot használok.

Twisted és Python alapú eseményvezérelt programozás

Mielőtt kiszolgáló alkalmazásunkat ezernyi folyamatból álló szörnyeteggé vagy kusza szálak gombolyagává változtatnánk, ismerjünk meg egy letisztult, logikus, eseményvezérelt módszert. Töltsük le a proxykiszolgálót mindössze 600 sorban megvalósító példaprogramot, és próbáljuk magunkévá tenni szellemiségét!

Kezdetekben a kiszolgálók `fork` hívásokkal osztódtak. Utánuk a többszálú kiszolgálók következtek. A több példányban vagy szállal futó kiszolgálók kisebb számú párhuzamos kapcsolatot jól kezelnek, ám amikor a kapcsolatok száma több százra vagy több ezerre nő, akkor túlságosan sok erőforrásigényes folyamatot indítanak ahhoz, hogy hatékonyan működhessenek. Napjainkra kialakult egy jobb megoldás, ezek az aszinkron kiszolgálók. Az eseményvezérelt programozás egykor bonyolult világát mára harmadik generációs programozási nyelvek keretrendszerei teszik barátságossá.

A *Python* közösség egyik fényesen ívelő csillaga a *Twisted*, mely egyszerűvé és elegánssá teszi az aszinkron programozást, miközben eseményvezérelt segédosztályok széles választékát is rendelkezésre bocsátja. Írásomban az aszinkron, eseményvezérelt programozásról és *Twisted* alapú megvalósításáról lesz szó. Ha csak elmélkedünk a kódról, azzal nem jutunk messzire, ezért egy valódi, kifejezetten a cikk céljaira készített *Twisted* alkalmazásból fogok példákat mutatni. Egy egyszerű proxykiszolgálóról van szó, amely letiltja a nem kívánt sütiket, képeket és kapcsolatokat. A teljes forráskód beszerzésével kapcsolatban a források között szerepel részletesebb tájékoztatás.

Mi a Twisted?

A *Twisted Project*, mint hálózati alkalmazások készítésére használható, nagyteljesítményű, egyre üzembiztosabb megoldás növekvő népszerűségnek örvend. Alapjában véve a *Twisted* egy aszinkron hálózatkezelő keretrendszer. A hasonló keretrendszerektől eltérően a *Twisted* a fontosabb protokollok és programozási feladatok kezeléséhez, például a felhasználók hitelesítéséhez vagy éppen a távoli objektumközvetítéshez számos beépített könyvtárral rendelkezik. A *Twisted* mögött álló filozófiák egyike az eszközkészletek közötti hagyományos határok eltörlése; például ugyanaz a kiszolgáló webes tartalom szolgáltatására és *DNS*-lekérdezések feloldására egyaránt alkalmas lehet.

Noha maga a csomag meglehetősen nagy, az alkalmazásoknak nem muszáj a *Twisted* minden összetevőjét beemelniük, vagyis a futtatási többletterhelés elenyésző.

Ahogy a *Python*, úgy a *Twisted* felhasználói bázisa is az oktatási szektor felől terjeszkedett a kereskedelmi és a kormányzati szereplők felé. A *Zotonál* mi a *Twistedet* egy elosztott fényképtároló és -kezelő alkalmazásban használjuk, általa ugyanis rövid idő alatt, közismerten termelékeny nyelven – ez volna a *Python* – tudunk méretezhető hálózati alkalmazásokat készíteni. Napi programozási munkám során újra és újra elismerést vált ki belőlem a *Twisted* lenyűgöző eszközkészlete és a mögötte álló közösség segítőkészsége. A más közösségi, nyílt forrású tervezetekhez hasonlóan *Twisted* üzleti eszközként is biztonságosan alkalmazható, léte ugyanis nem egyetlen vállalat vagy intézet támogatásának függvénye.

Mi az aszinkron programozás?

Ugye ismerős a helyzet, hogy sorban állunk egy vegyesbolt gyorspénztáránál, egyetlen üveg ásványvízzel, az előttünk lévő viszont megkérdőjelezi valamelyik tétel árát, és a mögötte állók mindannyian hosszú percekig kénytelenek várakozni az ár ellenőrzésére? Az aszinkron programozást számtalan módon szokták magyarázni, ám szerintem a legszemléletesebb példa az, hogy sorban állunk egy tétlen pénztárosnál. Ha a pénztáros aszinkron módon dolgozna, akkor megtehetné, hogy félreállítja az előttünk lévő személyt, és kiszolgál bennünket, amíg az ár ellenőrzése be nem fejeződik. Sajnos a pénztárosok elég ritkán működnek aszinkron módban. A szoftverek világában ugyanakkor az eseményvezérelt kiszolgálók gazdálkodnak a legjobban a rendelkezésre álló erőforrásokkal, ugyanis esetükben nincsenek értékes memóriát foglaló, a foglalatok forgalmára várakozó szálak. Maradva a vegyesbolt példájánál, a többszálú kiszolgáló további pénztárosok munkába állításával próbálja csökkenteni a sorban állás idejét, az aszinkron megoldás viszont lehetővé teszi, hogy egy-egy pénztáros egyszerre több vevőnek is segítsen.

1. kódrészlet A Twisted példakiszolgáló elküldi az időt, majd lezárja a foglalatot

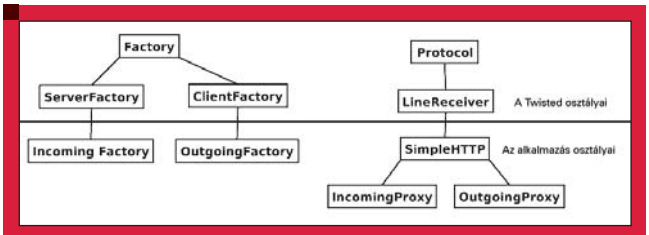
```
import time
from twisted.internet import protocol, reactor
class TimeProtocol(protocol.Protocol):
    def connectionMade(self):
        self.transport.write(
            'Szia! A pontos idő: %s' %
            ➥time.time())
        self.transport.loseConnection()

class TimeFactory(protocol.ServerFactory):
    protocol = TimeProtocol
reactor.listenTCP(1100, TimeFactory())
reactor.run()
```

Természetesen szó sincs arról, hogy a többszálú modellnek ne lennének előnyei. Például mikroszálak alkalmazásával az egyes szálak által lefoglalt erőforrások mennyisége lényegesen csökkenthető. Az aszinkron programozás szükségszerűen bonyolult, különösen, ha egymást követő blokkoló műveleteket kell végeznünk. ***Python*** alatt a szálakra osztásból fakadó előnyöket csökkenti a ***Python Global Interpreter Lock (globális értelmező zár, GIL)***.
A többszálú programozás ***Python*** alatt üdítően egyszerű, ugyanis a ***Python*** belső műveletei kivétel nélkül „szálbiztosak". Ha hozzá akarunk adni egy elemet egy listához, vagy be akarunk állítani egy szótárkulcsot, akkor semmilyen zárra nincs szükségünk ahhoz, hogy el tudjuk kerülni a szálak közötti versenyhelyzeteket. Mindezt egy a teljes értelmezőre kiterjedő zár teszi lehetővé, melyet sajnos a ***Python*** értelmezője meglehetősen nagylelkűen használ. Vagyis egyszerre két szál ugyanahhoz a listához biztonságosan tud elemet hozzáfűzni, ám ugyanez a zár lép érvénybe akkor is, ha két különböző listával dolgoznak. Mivel a többszálú ***Python*** alkalmazások teljesítménye emiatt romlik, az aszinkron, egyszálú programozás a ***Pythonhoz*** hasonló nyelvek esetében hangsúlyozottan előnyös lehet.

Csatlakozások fogadása és válaszok küldése

Kezdésként nézzünk egy egyszerű, a kapcsolatokat az 1100-as kapun fogadó példakiszolgálót. A kiszolgáló minden csatlakozásnál elküldi a ***UNIX*** időt, majd lezárja a foglalatot. Több kapcsolatot egyetlen szállal kezelni rendkívül bonyolult feladat – éppen ezt a kérdést célozzák meg a ***Twistedhez*** hasonló keretrendszerek. A hálózati kapcsolatokat a `twisted.internet.protocol.Protocol` osztály alosztályai képviselik, minden `Protocol` példány egy-egy hálózati kapcsolatot jelenít meg. Ezeket az objektumokat a `twisted.internet.protocol.Factory`-ból származó `Factory` objektumok hívják életre.
A `twisted.internet.reactor` egymaga végzi a piszkos munkát, vagyis a foglalatok lekérdezését és az események meghívását. ***Twistedben*** a `reactor.run()` meghívásával indul el az eseményhurok. A `run()` akkor lép ki, amikor az alkalmazás futása befejeződik, hasonlóan a ***GTK*** vagy a ***Qt*** eseményhurkaihoz.



1. ábra Egy proxykiszolgáló osztálydiagramja. A Protocol osztályok kezelik az egyes kapcsolatokat, és a Factory osztályok hozzák létre őket.

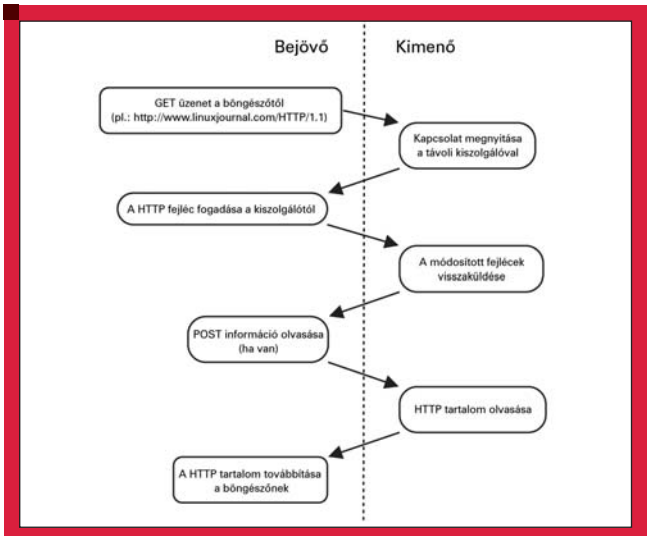
A proxykiszolgálós példa

Proxykiszolgálónk kétféle hálózati csevegőkapcsolatot ismer: bejövő ***HTTP***-kéréseket és a velük párosuló kimenő válaszokat. Mivel a ***HTTP*** egy csevegés jellegű protokoll, `protocol` osztályunkat a ***Twisted LineReceiveréből*** származtatjuk. Ez a `Protocol` alosztálya, a csevegés jellegű – például ***HTTP*** – kapcsolatokhoz külön szolgáltatásokat is biztosít. A ***Twisted*** valójában rendelkezik kifejezetten ***HTTP***-kérések indítására és kezelésére alkalmas osztályokkal is. Azért írunk mégis sajátot, mert a ***Twisted*** beépített osztályai proxyszolgáltatást nem biztosítanak, valamint a feladat programozási gyakorlatnak sem utolsó.
Az *1. ábrán* látható az az osztályszerkezet, amit alkalmazni fogunk. A ***Twisted*** a `Factory` osztályok példányait használja az egyes létrejövő kapcsolatokhoz tartozó `Protocol` példányok létrehozására. Létrehozunk egy `SimpleHTTP` osztályt, majd belőle származtatjuk a bejövő és a kimenő forgalmat kezelő osztályokat. Mivel a ***HTTP*** az ügyfél és a kiszolgáló oldalon szinte azonos, a tennivalók túlnyomó részét rábízhatjuk egy szuperosztályra, a maradékot pedig alosztályokra hagyjuk – pontosan így működnek a ***Twisted*** saját ***HTTP*** osztályai is.

A visszahívások kezelése

Eseményvezérelt programozásnál azok a műveletek, amelyeket egyébként egy vagy két eljárással el tudnánk végezni, több visszahívást is szükségessé tesznek. Ökölszabály, hogy minden olyan blokkoló művelet, amelyre várnunk kell, saját kódunkon kívül, azaz két eljárásunk között folyik. Proxykiszolgálónk esetében a kérések kezelésének fázisait számos további lépésre bontjuk. A proxykiszolgáló tevékenysége jelentős részben a böngészőtől érkező adatok beolvasásából, az adatok kismértékű módosításából, majd a távoli webkiszolgálónak való továbbküldéséből áll. A ***HTTP/1.1*** megjelenése óta egyetlen hálózati kapcsolaton keresztül több webes kapcsolatot is lehet kezelni. A *2. ábrán* látható, hogy mi történik az egyes kérésekkel. Ne feledjük, hogy minden ***HTTP***-kapcsolaton belül több kérést is lehet indítani. A négyszögeket összekötő nyilak az események sorrendjét jelzik.
Egy blokkoló jellegű programban szinte természetes, hogy ha meg kell nyitnunk egy távoli kapcsolatot, majd el kell küldenünk rajta egy sornyi szöveget, akkor ezt az alábbi módszerrel végezzük el:

```
connection = socket.open(remote_server,
➥remote_port)
connection.write(get_string)
response = connection.readline()
```



2. ábra A proxykérések feldolgozásának általános lépései

2. kódrészlet A műveletek halasztása Twistedben olyan, mintha tartásba helyeznénk őket, amíg a blokkoló utasítás végrehajtása be nem fejeződik.

```
d = self.outgoing_proxy_cache.getOutgoing
    (host, int(port))
d.addCallback(self.outgoingConnectionMade, uri)
d.addErrback(self.outgoingProxyError, uri)
```

3. kódrészlet Hagyományos hibakezelés Pythonban

```
try:
    (kérdéses kódrész)
except ValueError:
    (hibakezelő kód)
except MyError:
    (hibakezelő kód)
```


Mindannyian láttunk már ilyet, de vajon ***Twistedben*** mi az, ami másképp történik? Az eseményvezérelt programokban nem várjuk meg a kapcsolat felépülését, egyszerűen megadjuk, hogy milyen kódot szeretnénk futtatni, amikor a távoli kiszolgáló végre foglalkozni tud velünk. ***Twistedben*** az ilyen jellegű ütemezést, halasztást a `twisted.internet.defer.Deferred` osztály egy példányával intézzük, mely helyőrzőként szolgál ahhoz az eredményhez, amelyet egy blokkoló művelettől egyébként várnánk. Proxykiszolgálónkban például akkor alkalmazunk `Deferred` objektumot, amikor távoli kapcsolatot kezdeményezünk. (*2. kódrészlet*)
A `self.outgoing_proxy_cache.getOutgoing` eljárás egy kimenő proxykapcsolatot kezdeményez. A kapcsolat létrejöttét azonban nem várja meg, azonnal visszatér a hívóhoz. Minden hívás a lehető leghamarabb visszatér; ez az a megközelítés, mely lehetővé teszi az egyszálú kiszolgáló működését. Az eljárások a lekötött processzoridőt tehát valóban érdemi műveletekre fordítják, és nem arra, hogy külső események bekövetkeztére várjanak.
Érdemes megfigyelni, hogy a kapcsolati objektumot helyettesítő `Deferred` objektum visszaadása hogyan történik.
A `Deferred` objektumon az `addCallback` és az `addErrback` eljárást meghívva utasítások jövőbeli végrehajtását írjuk elő. Például a kimenő kapcsolat felépülése után a `self.outgoingConnectionMade` eljárás kerül meghívásra. Az `addCallback` második átadott értéke az uri, amivel azt közöljük a ***Twisteddel***, hogy a `self.outgoingConnectionMade` eljárást úgyszintén az urival, mint átadott értékkel kell meghívni.

Hibakezelés

Hiba esetén a hibakezelést biztosító `self.outgoingProxyError` eljárás kerül meghívásra egy `Failure` objektummal. A ***Python*** hagyományos hibakezelése kivételekkel dolgozik; a fogalom más magas szintű nyelvekből, például a Javából is ismerős lehet. (*3. kódrészlet*)
Bár a ***Python*** kivételkezelése szinkron programoknál – játsszunk kicsit a szavakkal – kivételesen jól működik, aszinkron programok kezelésére nincs felkészítve.

Például, amikor kimenő ***HTTP***-kapcsolatot kezdeményezünk, a ***Twisted*** a kapcsolat felépüléséig más események feldolgozásával foglalkozik. Nyilván azt szeretnénk, ha ettől függetlenül a kapcsolat felépítése közben esetlegesen előforduló hibák bármelyikét kezelni tudnánk. Szerencsére a ***Twisted*** készítői – áldjuk a nevüket – erre is gondoltak. Kódot nemcsak úgy lehet időzíteni, hogy blokkoló jellegű művelet véget érésekor fusson, hanem úgy is, hogy hiba felmerülésekor kapjon szerepet.
A ***Twisted*** az eseményhurkon belül felmerülő hibákat is kezeli, a kivételek kézben tartásában és naplózásában csatlakozási pontokkal segíti a fejlesztőket. Mindennek van egy járulékos előnye is: egy-egy kivétel ugyan meggátolhatja egy-egy művelet befejezését, a teljes kiszolgáló leállását viszont nem okozza, még akkor sem, ha egyetlen betűnyi kivételkezelő kódot sem írtunk.

A Twisted osztályai és a kivételkezelés

A ***Twisted*** osztályok egy részének használatakor, ilyen a most alkalmazott `LineReceiver` osztály is, számos eseményt úgy tudunk kezelni, hogy egyszerűen hozzáadjuk a megfelelő nevű eljárásokat az osztályokhoz. Minden alkalommal, amikor a protokoll fogad egy sort, a `lineReceived` eljárás kerül meghívásra, átadott értékként a sor szövegével. `SimpleHTTP` osztályunk, melyet a ***HTTP***-kapcsolatok alapszintű kezelésére készítettünk, a következő eljárásokkal rendelkezik:

- `startNewRequest`: Az egyes kérések kezdetekor kerül meghívásra.
- `lineReceived`: Célja a csevegés jellegű protokollok használatának segítése. Minden alkalommal, amikor újabb sornyi szöveg érkezik a foglalaton keresztül, ez az eljárás hívódik meg.
- `rawDataReceived`: Bináris fájl vagy nyers adatfolyam küldésekor nincs értelme új sor karakterekkel elválasztott részek feldolgozásáról beszélni. Éppen ezért a `LineReceiver` lehetővé teszi, hogy nyers átviteli módba váltsunk, ilyenkor a `lineReceived` helyett a `rawDataReceived` kerül meghívásra.

4. lista Események ütemezése Twistedben

```
def outgoingConnectionMade(self, outgoing_proxy,
                           uri):
    """
    Akkor kerül rá a vezérlés, amikor egy kimenő
    ➥proxykapcsolat
    felépült. Ez is egy Twisted visszahívó
    ➥eljárás.
    """
    assert(outgoing_proxy, OutgoingProxy)
    self.outgoing_proxy = outgoing_proxy
    outgoing_proxy.incoming_proxy = self

    # HTTP parancs küldése és a válasz visszaadása
    outgoing_proxy.write('%s %s %s' % \
        (self.http_command,
         uri,
         self.http_version) \
         + self.delimiter)

    outgoing_proxy.firstline_sent_def.addCallback(
        self.outgoingFirstlineReceived)

    # A sorba állított adatok elküldése
    self.flushOutgoingBuffer()

    # Visszahívók hozzáadása a fejlécek
    # megérkezésének idejére

outgoing_proxy.headers_finished_def.addCallback(
        self.outgoingHeadersReceived)

outgoing_proxy.request_finished_def.addCallback(
        self.handleOutgoingRequestFinished)
```

- `handleFirstLine`: A *HTTP* úgy működik, hogy minden kérést egy különálló sorral indít. Általában az ügyfél egy *GET* vagy *POST* kérést küld el egy *URI*-val, a kiszolgáló pedig egy állapotkóddal válaszol. A `handleFirstLine` mindegyik eset kezelésére alkalmas.
- `handleHeadersFinished`: A *HTTP* fejrészek teljes elküldése után kerül meghívásra.
- `handleRequestFinished`: Magának a *HTTP*-kérésnek a befejezésekor kerül meghívásra.

A *Twisted* programozók úgy illeszthetik sorba az eseményeket, hogy az adott protokoll kezelése folyamán felmerülő műveletekhez és állapotokhoz külön eljárásokat írnak. Adott kérés elindításakor megadhatjuk, hogy a kérés kezelésének egyes lépéseiben milyen eseményekre kerüljön sor. Korábbi példánkban egy kapcsolat létrejötte után a `self.outgoingConnectionMade` eljárást hívtuk meg. Vizsgáljuk meg ezt az eljárást. (*4. kódrészlet*)
Megjegyezném, hogy az `outgoing_proxy` a távoli kiszolgálóval a böngészőprogram nevében létrehozott kapcsolatot képviseli. A *HTTP*-kérést az `outgoing_proxy.write` eljárással küldjük el. A `self.outgoingFirstlineReceived` eljárás meghívását akkorra ütemezzük, amikor választ kapunk a távoli kiszolgálótól, a `self.outgoingHeadersReceived` eljárás pedig akkor jut szerephez, amikor a távoli kiszolgáló az összes *HTTP* fejlécet elküldte. A záró hívás a `self.handleOutgoingRequestFinished`, erre akkor kerül sor, amikor a távoli kiszolgáló teljes egészében elküldte a kimenő *HTTP*-kérésünkre adott válaszát.
Bár az `outgoingConnectionMade` eljárás visszatér, mielőtt bármi is történne, mi sorba állítjuk a jövőbe ütemezett eseményeket. Akár az is előfordulhat, hogy miközben adott kapcsolaton válaszra várunk, ugyanazon a szálon belül további tíz kérést nyitunk meg és zárunk le. A kapcsolatokkal összefüggő adatok mindegyike protokollosztályok példányadataként tárolódik. A `Factoryk` életre hívják a protokollpéldányokat, a protokollpéldányok tárolják a kapcsolatállapotokat, a `deferred` objektumok pedig a jövőbeli adatokat párosítják az eseménykezelőkkel.
A kirakójátékot a `reactor` teszi teljessé, mely elvégzi a foglalatok lekérdezésének piszkos munkáját.
Ez a *Twisted* alapjául szolgáló eszközkészlet.

Összeáll a kép

Barkácsolás céljára bárki letöltheti az eddigiekben tárgyalt proxykiszolgáló egészen pontosan 606 soros kódját. Nem tagadom, a vállalati intranetünket nem tenném mögé, itthon viszont már egy hete használom a nemkívánatos sütik és képek eldobására, illetve egy bizonyos cég oldalának elérését is letiltottam. Amikor elkezdtem a *Twistedet* használni, könnyedén rá tudtam hangolódni az aszinkron programozás gondolatvilágára; az általam kívánt adatáramlás és az események összerendelése már nehezebben ment, a legnehezebb pedig mindezt elmagyarázni más valakinek. Senki ne ijedjen meg! Mi a *Zotonál* gyakorlatilag nulla *Twisted*-ismerettel indultunk, mégis kevesebb mint egy év alatt fel tudtunk építeni egy rendkívül sokoldalú, magas fokon bővíthető, fürtözött fényképkezelő és -tároló alkalmazást. Ráadásul én voltam az egyetlen, aki teljes időben a kiszolgálóval foglalkozott.
Természetes, hogy a *Twisted* nem felel meg mindenki igényeinek. Kiterjedtsége, bár sokoldalúságának alapja, riasztó lehet. Aki egyszerű, *Python* alapú, aszinkron csevegő kiszolgálót keres, ismerkedjen meg a *Medusa*-val. A *Twistedhez* hasonlóan a *Medusa* is *Factorykba* (ezek a *Dispatcherek*) és csevegő osztályokba szervezi az aszinkron programozást.

Linux Journal 2005. március, 131. szám

Ken Kinder jelenleg egy fürtözött Twisted kiszolgálót fejleszt a Zoto számára az Oklahoma állambeli Oklahoma városában. Szeret túrázni, síelni, fényképezni – és persze Linuxot nyúzni. Szülővárosa a Colorado államban található Boulder.

KAPCSOLÓDÓ CÍMEK

A cikkhez tartozó források elérhetősége:
➲ www.linuxjournal.com/article/7963

SDL – Multimédiás programozói könyvtár (1. rész)

Nem tudom ki hogyan van vele, de számomra a linuxos átállás után az első hiányérzetem akkor támadt, mikor valamilyen grafikus alkalmazás fejlesztésébe szerettem volna kezdeni. Sokáig nézelődtem az eszközök között, míg végül kikötöttem az SDL mellett. Ez a programozói könyvtár szinte gyerekjátékká teszi a videomódok, hangszolgáltatás, időzítés programozását Linux alatt. Könnyű használni, népszerű, multiplatform és napjainkban már minden disztribúcióban megtalálható.

Mi is az SDL?

Az SDL a *Simple Directmedia Layer* kifejezés rövidítése. Egy multiplatform programozói könyvtár, elsősorban az egér, billentyűzet, joystic és hangszolgáltatások valamint a video framebuffer kezelésére szolgál. Nagyon jó „barátja" az *OpenGL*-nek is. A rövidítést keverni szokták egy igen komoly kifejezéssel: *Specification and Description Language*. Ez utóbbi egy formális nyelv elnevezése, melyet telekommunikációs rendszerek tervezésére fejlesztettek ki. Maradjunk mi most a jó öreg *Linuxnál* és a multimédiás szolgáltatásoknál, melyek arra várnak, hogy az *SDL* segítségével kiaknázzuk őket.

Támogatottság

Felmerülhet a kérdés, hogy milyen nyelvhez is készült ez a könyvtár. Nagyon jó érzés azt elmondani, hogy szinte minden „divatos" programozási nyelvhez létezik *SDL*. A hivatalos honlapról a következő nyelvekhez tölthetjük le: *Ada, C++, C#, Eiffel, Erlang, Euphoria, Guile, Java, Lisp, Lua, ML, Objective C, Pascal, Perl, PHP, Pike, Pliant, Python, Ruby*. Véleményem szerint ez igen barátságos lista. Ha már a támogatott nyelveknél tartunk, akkor érdemes megemlíteni a platformokat is, melyeken az *SDL* futtatható: *Linux, BeOS, MacOS, FreeBSD, Qnx, Amiga, Atari, Symbian OS, Win32*. Mindet nem szeretném felsorolni, hiszen már ebből is látszik, hogy mennyire széles körben támogatott. Minden szükséges eszköz egy helyen elérhető a projekt honlapján: http://www.libsdl.org. A sok nyelv közül a *C++* segítségével szeretnék majd bemutatni pár apróságot, hogy mire lehetünk képesek az *SDL* használatával.

Az SDL család

Az *SDL* több különböző *alrendszerrel* rendelkezik. Ezek az alrendszerek végül is önmagunkban is programozói könyvtárak: videó, eseménykezelő, audio, cd-rom audio, szálak és időzítő. Lássuk egyenként, hogy mire is képesek ezek a tagok.

Általános szolgáltatások

Az *SDL* könyvtár tartalmaz egy általános, az *SDL* rendszert kezelő szolgáltatás csoportot. Ezekkel lehet inicializálni és lezárni a rendszert, hibakezelést végezni, valamit információkat lekérdezni az *SDL* alrendszereiről.

Videó

A videó szolgáltatás a legbővebb alrendszer az *SDL*-ben. Segítségével bármely videó módot – ha azt a videó hardver támogatja – beállíthatunk 8 bites színmélységtől kezdve. Manipulálhatjuk az adott videó felület gamma, paletta és alpha beállításait és még sok mindent. Az *OpendGL* támogatást is rajta keresztül lehet programozni. Nehéz is lenne itt pár mondatban összefoglalni, hogy milyen sokféle és színes szolgáltatásokat nyújt. Igyekszem majd példákon keresztül megmutatni az alapokat. Kétség kívül ez a legtöbbet használt alrendszer az *SDL*-ben.

Ablakkezelés

Ezekkel a szolgáltatásokkal tudjuk manipulálni az adott videó felület bizonyos tulajdonságait, mint teljes képernyő vagy ablakos mód, ablakfelirat és az ablak ikonja.

Eseménykezelés

Az *SDL* másik lényeges része, melynek segítségével biztosíthatjuk alkalmazásunkban a felhasználói interakciókat. A nagy része *egér-* és *billentyűzetkezelés*, de tartalmaz *joysticra* vonatkozó elemeket is, bár ennek kezelésére az SDL külön szolgáltatás csoporttal rendelkezik.

Audio

Segítségével információkat kaphatunk az adott audio hardverről, *wav* fájlokat tölthetünk a memóriába és játszhatunk le, valamint *konverziókat* eszközölhetünk különböző audio formátumok között (ne értsük félre, csak a wav formátumokon belül).

Szálak

A szálak programozása igen hasznos dolog játékfejlesztések esetén. Itt találhatjuk a *szemaforokat*,

szálazonosítókat kezelő szolgáltatásokat. Szálakat hozhatunk létre illetve törölhetünk, közöttük randevúkat intézhetünk. Nagyon nem fogunk belemenni a cikk során, mert a szálak programozása párhuzamos programozási ismereteket követel meg. Erről pedig hatalmas mennyiséget lehetne kifejteni.

Időzítő

A szálak mellett a másik fontos dolog az *időzítő*. Játékfejlesztések során megint csak alap dolognak számít. A cikksorozat folyamán lesznek példák az alkalmazására.
Nagy vonalakban láthattuk, hogy milyen szolgáltatásokkal kecsegtet az *SDL*. A cikkben sajnos mindenre nem tudok majd kitérni, de remélhetőleg utat nyitok a megismerés felé, azoknak akik még esetleg nem is hallottak az *SDL*-ről, vagy – programozói körökben divatos szakkifejezéssel élve – még csak „szagolgatják” azt.

Inicializálás, előkészületek

Mivel a legtöbb disztribúció alapból tartalmazza az *SDL* könyvtárat, így nagyon nem is térünk ki annak telepítésére. Lássuk, hogy hogyan is néz ki egy egyszerű *SDL*-t felhasználó *C++* program *(1. lista)*.
Minden *SDL* alkalmazásnak be kell építenie az *SDL.h*-t. Ez a fejlécállomány fogja tartalmazni az alrendszerekre történő hivatkozásokat. Csak azokat az alrendszereket használhatjuk melyeket az `SDL_Init` eljárással inicializáltunk. Ez az eljárás `Uint32` típust vár paraméterként és egészet ad vissza. Hiba esetén -1 értékkel tér vissza, ha minden rendben akkor ez az érték 0. Hiba esetén az *SDL* hibaüzenetét az `SDL_GetError` függvény adja vissza. Az inicializálás során adhatjuk meg, hogy mely alrendszereket szeretnénk használni. Ezt különböző flagekkel tehetjük meg. Az elnevezések magukért beszélnek, a felhasználható flagek a következők: `SDL_INIT_TIMER`, `SDL_INIT_AUDIO`, `SDL_INIT_VIDEO`, `SDL_INIT_CDROM`, `SDL_INIT_JOYSTIC`. Ha mindent egyszerre szeretnénk használni, nem fontos beírni az összeset, használjuk inkább az `SDL_INIT_EVERYTHING` jelzőt. Megjegyzem ez nagyon kényelmes dolog, de ha az alkalmazásunk optimális teljesítményére szeretnénk

1. lista Az SDL inicializálása

```
#include <iostream>
#include "SDL.h"

int main()
{
    if (SDL_Init
    ➥(SDL_INIT_VIDEO) < 0)
    {
        std::cerr << "Nem
        ➥sikerült az SDL
        ➥inicializálása!
        ➥Hiba: " <<
        ➥SDL_GetError();
        exit(1);
    }

    /* Ide jöhet az SDL-t
    ➥felhasználható kódunk */

    SDL_Quit();

    return 0;
}
```

törekedni, csak azokat az alrendszereket inicializáljuk, melyeket feltétlen használni fogunk a fejlesztés során. Például: `SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO)`.
Eseménykezelésért felelős alrendszert soha nem kell külön elindítanunk, ha a videót inicializáltuk. A videóval együtt az eseménykezelés is automatikusan elindul. Az *SDL* rendszert a program végén az `SDL_Quit` utasítás kapcsolja ki, minden inicializált alrendszert deaktiválva.
Ha most ezt a kódot „bedobjuk” a fordítónak akkor, bizony nem lesz sikerélményünk. Az *SDL* fejléceket nem fogja látni a fordító alapból. Az SDL csomaghoz tartozik egy hasznos kis program, mely segít konfigurálni a fordítót. Ennek neve: `sdl-config`. Megfelelően paraméterezve megkapjuk azt a kimenetet, melyet a fordítónk hiányol:

```
sdl-config --libs --cflags
```

A program kimenetét – a héjnak hála – játszva át tudjuk adni a fordítónak. Tehát egy *SDL* program (*main.cpp*) fordítása a következő-

2. lista A videó alrendszer inicializálása és az eseménykezelés alapja

```
#include <iostream>
#include "SDL.h"

int main()
{
    SDL_Surface* sdl_surface;
    SDL_Event sdl_event;
    bool main_loop_exit =
    ➥false;

   /* Ide jön az
   ➥inicializálás! */

    sdl_surface =
    ➥SDL_SetVideoMode(640,
    ➥480, 16, SDL_SWSURFACE);

    if (sdl_surface == NULL)
    {
        std::cerr <<
        ➥"SDL_Surface hiba: "
        ➥<< SDL_GetError();
        exit(1);
    }

    while (!main_loop_exit)
    {
      while (SDL_PollEvent
      ➥(&sdl_event))
      {
        switch
        ➥(sdl_event.type)
        {
          case
          ➥SDL_KEYDOWN:
            main_loop_exit =
            ➥true;
                  break;

              default:
                  break;
          }
      }
    }

    SDL_Quit();

    return 0;
}
```

képpen történhet, ha mindent szabványosan szeretnénk:

```
g++ `sdl-config --libs --cflags`
-Wall -pedantic -ansi main.cpp
```

3. lista Egy pixel megjelenítése (16 bites színmélység esetén)

```
void pixel(SDL_Surface*
➥surface, int x, int y,
➥Uint16 color)
{
    Uint16 *p = (Uint16
    ➥*)surface->pixels + y *
    ➥surface->pitch/2 + x;
    *p = color;
}
```

Most, hogy mindent előkészítettünk készítsünk végre egy olyan programot, ami valamilyen szemmel látható módon alkalmazza az ***SDL*** szolgáltatásait. Kezdjük a videó alrendszer használatával és az eseménykezeléssel, mivel ezek a domináns csoportok. Egy általános grafikus SDL alkalmazás a kezdetek kezdetén a *2. listában* látható módon nézhet ki.
Amint láthatjuk már egy alap ***SDL*** program is több sorra rúghat. Nem zárom ki, hogy kevesebb sorból is ki lehet jönni. Ámde lássuk sorban mit is csinál a *2. listában* szereplő kód. Láthatjuk, hogy deklaráltunk egy `SDL_Surface` típusú mutatót.
Ez a mutató fog mutatni egy videó felületre, amit az `SDL_SetVideoMode` eljárással alakíthatunk ki. Ez az eljárás rendre a következő paramétereket várja: képernyő szélessége, magassága, bitek száma pixelenként és flagek. Ha gond van akkor `NULL` mutatóval tér vissza. A paraméterek közül a flagek szorulnak némi magyarázatra. Ezek közül is több létezik.
Az `SDL_SWSURFACE` alapján a rendszer a videó felületet a rendszer memória területén foglalja le, `SDL_HWSURFACE` esetén pedig a videómemóriában.
Ha teljes képernyőn szeretnénk látni az adott felbontást, akkor használjuk az `SDL_FULLSCREEN` flaget. Ezeket a flageket természetesen a vagy („|”) művelettel össze tudjuk kapcsolni, például:

```
SDL_SetVideoMode(640,480,16,SDL
➥_SWSURFACE|SDL_FULLSCREEN)
```

A másik fontos dolog az ***eseménykezelés***. A videószolgáltatás szemléltetése kényelmesebb így, hiszen ha a kódot enélkül futtatnánk, akkor a videófelületből nem láthatnánk semmit sem, csupán egy felvillanó ablakot, vagy teljes képernyős mód esetén egy villanást.
Az ***SDL*** minden eseményt egy eseménysorban tárol. Az ebben a sorban tárolt eseményeket az `SDL_PollEvent` eljárással nyerhetjük ki. Az `SDL_Event` típus egy ***unió*** típus. Innen az esemény típusától függően kinyerhetjük a számunkra értékes adatokat, többek között, hogy milyen az adott esemény(billentyűleütés vagy felengedés, egérgomb, egér mozgatása stb.) .
Példánkban *(2. lista)* az `SDL_KEYDOWN` eseményre hivatkozunk. Használhatjuk még az `SDL_MOUSEMOTION`, `SDL_MOUSEBUTTONDOWN`, `SDL_MOUSEBUTTONUP`, szintén eléggé magukért beszélő elnevezéseket is.
Tehát alapból az eseménykezeléshez deklarálnunk kell egy `SDL_Event` változót, valamit ki kell nyernünk ebbe a változóba az eseménysorból, az éppen soron lévő eseményt az `SDL_PollEvent` eljárással. Innen pedig a típustól függően lekezelhetjük a felhasználó interakcióit. Később még több példát is láthatunk erre. Most főként a kényelem miatt vezettük csak be az eseménykezelést. A programunk most bármely billentyű megnyomására kilép.

Egy színes példa

Térjünk vissza a videószolgáltatáshoz. Most hogy már van egy felületünk a soron következő dolog, hogy meg is tudjunk jeleníteni valamit. Lássunk hogyan nézhet ki egy pixelt megjelenítő rutin (*3. lista*).
A `p` mutatóban az `x` és `y` koordinátákból kiszámolt pixel memóriában elfoglalt helyét kapjuk meg. Ez most a 16 bites színmélység speciális esete. Más bit értékek esetén ügyeljünk a mutatók típusára! A `surface->pixels` tag mutat a tényleges videófelületre, ahová a színértékeket írhatjuk be. Ehhez adjuk hozzá a klasszikus módon kiszámított pixel helyét. A szín (`color`) érték kiszámításához az `SDL_MapRGB` függvényt hívjuk segítségül. Ezzel az eljárással keverhetünk ki a piros, zöld,kék értékekből, az adott felület típusától függően színértékeket. Például állítsuk elő az `sdl_surface` felülethez illő sárga színértéket:

```
Uint16 sarga = SDL_MapRGB
➥(sdl_surface->format,
➥0xff,0xff,0x00)
```

■ *1. ábra* Egy egyszerű színátmenet SDL segítségével

Így a ***(10,10)*** koordinátában már ki tudunk rajzolni egy sárga képpontot:

```
pixel(sdl_surface,10,10,sarga)
```

Azonban egy videófelületre nem lehet csak úgy „firkálni”! A felületet le kell zárnunk, addig amíg a módosítás történik. Ez több szálú programozás esetén hasznos dolog, hiszen gondoljuk csak el mi történne ha egyszerre több programszál egy időben kezdene kirajzolni egy adott felületre. Az ***SDL_LockSurface*** eljárás segít nekünk. Lássunk egy konkrét példát, mely a már ismertetett ***pixel*** eljárást alkalmazza *(4. lista)* és a következő képet fogja kirajzolni: *(1. ábra)*.
Amint látjuk alkalmaztunk egy `SDL_UpdateRect` nevű eljárást. Ennek segítségével aktivizáljuk az adott felületen a változtatásokat és jelenítjük meg a frissített felületet. Paraméterként a felület nevét, majd a frissítendő négyzet koordinátáit várja (bal felső sarok, jobb alsó sarok).
Hogy látványosan elnevezzük az első komolyabb ***SDL*** programunkat, ismerjük meg az ablakkezelő rendszer egyik funkcióját:

```
SDL_WM_SetCaption.
```

Ezen eljárás segítségével képesek vagyunk módosítani az adott alkalmazás ablakának feliratát, valamint ikonnevét. Például:

```
SDL_WM_SetCaption("SDL
➥Színátmenet","")
```

Ezt a sort az `sdl_surface` inicializálása után érdemes beszúrni a fenti példában.
Remélhetőleg érdekes információkkal szolgált az olvasnivaló az ***SDL*** programozói könyvtárról. Még korántsem láttunk mindent! A következő részben még tárgyalunk néhány videó funkciót és eseménykezelést. Rajzolunk majd az egér segítségével és példát láthatunk majd egy ***BMP*** fájl megjelenítésére is. Később használni fogjuk az audio, cd-rom, timer alrendszerek funkcióit is.
Addig is érdemes ellátogatni a ➲ ***http://www.libsdl.org*** oldalra bővebb információkért és „lapozgatni” a ➲ ***http://www.libsdl.org/cgi/docwiki.cgi*** oldalakat. Itt többek között utána lehet nézni az inicializációs flageknek valamint a videómódok beállításánál használt flageknek is. Az események típusai is részletesen megtalálhatóak, melyekből természetesen a következő számban egy párral meg is fogunk ismerkedni.

4. Lista Színátmenet megjelenítése

```
#include <iostream>
#include "SDL.h"

void pixel(SDL_Surface* surface, int x, int y, Uint16
➥color){...}

int main()
{
    SDL_Surface* sdl_surface;
    SDL_Event sdl_event;
    bool main_loop_exit = false;
    Uint16 color;

...
     /* Itt inicializálunk */

    sdl_surface = SDL_SetVideoMode(255, 255, 16,
     ➥SDL_SWSURFACE);

    if (sdl_surface == NULL) { ... }

   // Lefoglaljuk az írásjogot a felületre.
    SDL_LockSurface(sdl_surface);

    for(Uint16 i=0;i<255;i++)
    {
        for(Uint16 j=0;j<255;j++)
        {
            // Színkeverés és kirajzolás.
            color = SDL_MapRGB(sdl_surface->format, i,
            ➥j,255-j);
            pixel(sdl_surface,i,j,color);
        }
    }
   // A felület frissítése
    SDL_UpdateRect(sdl_surface, 0,0,255,255);

   // Elengedjük a felületet.
    SDL_UnlockSurface(sdl_surface);

    while (!main_loop_exit) {...}

    ...

    return 0;
}
```

Radics Péter
(peter.radics@gmail.com)
Az ELTE-n tanulok programtervező matematikus szakon. Hobbim a kosárlabda, autóvezetés, web-design, programozás. Főleg webes alkalmazások fejlesztése érdekel. 4 éve megrögzött Linux felhasználó vagyok.

Párhuzamos programok fejlesztése PVM könyvtárral (1. rész)

Bevezetés a PVM-be

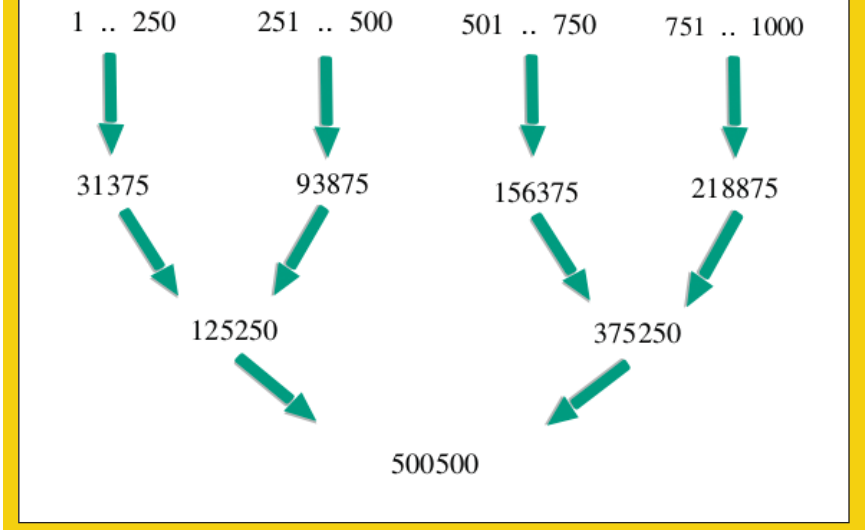
Egyszerű használata és robusztussága miatt párhuzamos programok fejlesztéshez a PVM könyvtár kezdők és profik számára egyaránt az egyik legjobb jelenleg fellelhető eszköz.

Párhuzamos architektúra és a párhuzamos programozás

Minden egyéb hardverismeretet mellőzve tegyük fel, hogy nem egy számítógéppel (processzorral) rendelkezünk, hanem többel. Legyen ezek száma *N*. Tegyük fel azt is, hogy ezek a számítógépek valamilyen összeköttetésben vannak egymással, legyen az akár hálózati vagy közvetlen kapcsolat (például többmagos processzorok, osztott memória), vagy esetleg valami egészen más. Természetesen szükség lesz még a megfelelő szoftverre is számítógépeink munkába fogásához. Egy ilyen hardver- és szoftverrendszert együtt párhuzamos architektúrának nevezünk. Párhuzamos architektúra valójában többféle létezik, de ebben a cikkben mi kizárólag hálózatba kötött számítógépekkel, úgynevezett klaszterekkel (cluster) fogunk foglalkozni.
A párhuzamos architektúrákra fejlesztett programokat párhuzamos programoknak, míg programírás illetve programtervezés folyamatát párhuzamos programozásnak nevezzük.
A példák kipróbálásához nem kell, hogy legyen otthon egy klaszterünk, mivel mindegyik működik egy processzoros, hálózatba nem kötött gépen is. Ám a dolog igazi előnyét akkor láthatjuk, ha igazi klaszteren futtatunk.

Ha egy tyúk egy nap alatt...

Kezdjük egy nagyon egyszerű példával. Összegezni szeretnénk az egész számokat 1-től M-ig. A tudománytörténetből ismert, hogy Gauss (1777-1885) már gyermekkorában kitalálta ehhez azt az algoritmust, amivel akár fejben is kiszámolhatja bárki az eredményt.
Mi most mégis essünk neki nyers erővel és adjuk össze a számokat N processzoron párhuzamosan. Tegyük fel, hogy egy összeadás egy processzoron 1 másodpercig tart. Legyen M=1000, a processzorok száma (N) pedig 4.
A számításhoz egy processzorral M-1=1000-1 azaz 999 másodpercre volna szükségünk, tehát ez lesz a viszonyítási alap. Osszuk most fel a számokat 4 csoportra: 1..250, 251..500, 501..750, 751..1000.



1. ábra Számok összeadásának egy lehetséges módja 4 processzoron

Ha minden processzorunkkal egy ilyen csoport összegét számoljuk ki, és a processzorok párhuzamosan dolgoznak, akkor a csoportösszegek kiszámítása 249 másodpercig tart.
A négy csoportösszeget összeadni a 4 processzoron (de egyen is) további 3 másodpercig tart, így a feladat teljes megoldása 252 másodpercig tartott. Ezt a fajta feladatmegoldást szemlélteti *1. ábránk*.
A lényeg ebben az esetben az volt, hogy a megoldandó feladatot felosztottuk részfeladatokra és az egyes részfeladatokra, és azok megoldását rábíztuk processzorainkra, amik

eztán a végső megoldást egymással összehangolva számolták ki jelentős időt takarítva meg ezzel. A megoldás 4 processzoron hozzávetőleg negyedannyi időt vett igénybe mint egy processzoron, hatékonyságunk tehát négyszeres a hagyományos módszerhez képest. Szinte minden párhuzamos algoritmus lényege ez. Az elvégzendő lépések felosztása, hogy egymástól függetlenül, több processzoron futhassanak.

A PVM telepítése

A *PVM* (*Parallel Virtual Machine*; *párhuzamos virtuális gép*) egy programkönyvtár, amellyel párhuzamos programokat írhatunk. Számos platformon elérhető (többek között *Linux*, *Unix* és *Windows* rendszereken), elterjedt, nagyon hatékony és viszonylag könnyen használható. Mi természetesen *Linux* alatt fogjuk kipróbálni, ám a programok más rendszereken minimális változtatással (vagy akár változtatás nélkül) szintén lefordíthatók.
A *PVM* telepítését az olvasóra bízom, erről itt csak annyit jegyeznék meg, hogy a *Debian* illetve *Ubuntu* felhasználóknak igen egyszerű dolguk van, hiszen csak ki kell adniuk az

```
apt-get install pvm
apt-get install pvm-dev
```

parancsokat.
A forráskódot a *PVM*-et honlapjáról (➲ *http://www.csm.ornl.gov/pvm/*) tölthetjük le.
Telepítés után a következő lépés a futtatási környezet elkészítése. Ez mindössze néhány könyvtár létrehozását jelenti:

```
mkdir ~/pvm3/
mkdir ~/pvm3/bin
mkdir ~/pvm3/bin/LINUX
```

Ez a PVM helyes működésének szükséges feltétele, mivel a rendszer párhuzamos programjainkat ezen az elérési úton fogja majd keresni. A gépek közti kommunikációt megvalósító PVM démont a `pvm` parancs kiadásával indíthatjuk el. Ha minden jól ment, akkor a következőt láthatjuk:

```
pvm>
```

1. Lista A master folyamat forrása

```
/* master.c */
#include <stdio.h>
#include <stdlib.h>
#include <pvm3.h>

#include "tags.h"

/* az inditando slave folyamatok szama */
#defineNUM_SLAVES      8

int main ()
{
    intmytid, slave_tids[NUM_SLAVES];
    char       message[30];
    int        ;
    int        result;


    /* sajat tid lekerdezese, belepes a PVM-be */
    mytid = pvm_mytid();

    /* - az inditando allomany neve
       - a paranccsori parameterek
       - pvmnek info, hogy milyen modon futtassa
       - ide megadhato a gepnev ill. az architektura
       - hogy hany darab induljon
       - a tomb, ahova a tid -ek megerkeznek majd
    */
    result = pvm_spawn("slave1", (char **)0, PvmTaskDefault, "",
    ➥NUM_SLAVES, slave_tids);

    if (result != NUM_SLAVES) {
  fprintf(stderr, "HIBA: A futtatas nem lehetseges.
  ➥Keves bevonhato processzor\n");
  pvm_exit();
  exit(-1);
    }

    for (i = 0; i < NUM_SLAVES; i++) {
  /* az uzenetkuldes inicializalasa */
  pvm_initsend (PvmDataDefault); /* ez a default kuldesi mod */

  /* kuldendo uzenetet (ami egy egesz) becsomagolasa */
  pvm_pkint(&slave_tids[i], 1, 1);

  /* elkuldes MSG_DATA tag-gel */
  pvm_send(slave_tids[i], MSG_DATA);
    }


    for (i = 0; i < NUM_SLAVES; i++) {
  /* uzenet fogadasa az i. slave folyamattol */

/* -1 jelentese a feltetlen partatlansag, vagyis barkitol
➥johet uzenet */
```

1. Lista folytatás

```
  pvm_recv (-1, MSG_DATA);

  /* kicsomagolas */
  pvm_upkstr(message);

  /* kiiras */
  printf ("%s \n",message);
    }

    pvm_exit ();

    /* kilepes */
    return 0;
}
```

Adjuk ki a `conf` parancsot. Ekkor a konzolon a következőket láthatjuk:

```
pvm> conf
conf
1 host, 1 data format
 HOST   DTID   ARCH   SPEED   DSIG
 Lorien  40000  LINUX  1000
 ➥0x00408841
pvm>
```

A `conf` parancs hatására a *PVM* démon kiírta, hogy jelenleg milyen gépek tagjai a klaszterünknek. A fenti információ azt jelenti, hogy egyelőre csak egy gépünk van, ám a kezdéshez ez is elegendő lesz.

Első PVM programunk

Ha egy gépre írunk párhuzamos programot, akkor a *PVM* démon szimulálja a többi gépet méghozzá úgy, hogy az összes indított folyamat ezen az egy gépen fog futni.
Így a párhuzamosságot csak imitáljuk, időt nem nyerünk, de a teszteléshez ez egyelőre elég lesz.
Komolyabb programok tesztelése így csalóka lehet, hacsak nem írtunk bombabiztos programot. Egy párhuzamos program nyomkövetése – ha egyáltalán lehetséges – sokkal nehezebb mint egy hagyományos szekvenciális társáé.
Első programunk feladata az lesz, hogy bebizonyítsa párhuzamosságát, méghozzá úgy, hogy minden processzor (a klaszter egyes gépei) kiír majd a képernyőre egy rövid kis üzenetet. Valójában nem maguk a processzorok írnak a képernyőre, a példa a *PVM* folyamatainak üzenetküldési mechanizmusát demonstrálja.
Az implementáció két C nyelvű program – mondhatnánk a párhuzamos programunk egy-egy folyamatának is –, egy úgynevezett „master” és egy „slave”, magyarul mester és szolga, de talán szebb, ha úgy fogalmazunk, hogy szerver és kiszolgáló. Ez utóbbi elnevezés nem a hétköznapi értelemben vett szerver-kiszolgáló kapcsolatot jelenti. Jelen esetben a master feladata, hogy elindít adott számú slave folyamatot, a slave folyamatok pedig üzenetet küldenek a masternek, jelezvén, hogy ők futnak. Ezzel tulajdonképpen „bizonyítják” a párhuzamos futást.
Egy fontos fejléc fájlunk van, a ***pvm3.h*** ami a *PVM* könyvár függvényeinek deklarációit és előre definiált konstansait tartalmazza. A ***tags.h*** saját fejléc-fájl, itt konstansként definiáljuk a programjaink által küldött üzenetek típusait (itt csak egyetlen üzenettípust használok és legtöbbször ez elég is, később azonban szükség lehet az üzenetek megkülönböztetésére).
Lényeges változóink a `mytid` (egész) és `slave_tids` (egészek tömbje). Minden *PVM* programnak (folyamatnak) van egy egyedi azonosítója, úgynevezett `tid`-je. Ez az azonosító tulajdonképpen olyan mint az operációs rendszer futó folyamatainak azonosítója, azaz a `PID`, ám ez a *PVM* futtatókörnyezeten belüli azonosításra szolgál és nincsen kapcsolatban az esetleges operációs rendszerbeli `PID`-del.
A `tid` lekérdezésére szolgál a `pvm_mytid()` függvény. Ez a függvény egy egész számot ad vissza, ami a futó folyamatunk `tid`-je.
Minden egyes *PVM* program elején meg kell hívni ezt a függvényt, ezzel jelezzük a *PVM* számára programunk elindítását.
A `pvm_spawn()` függvény meghívásával újabb folyamatokat indítunk. Ezek lesznek az egyes slave-ek, akiktől majd üzeneteket fogadunk. Paramétereinek pontos magyarázatát, mint ahogy az összes *PVM* függvényét, man oldalának átböngészésével ismerhetjük meg. Dióhéjban a `pvm_spawn()` függvény paraméterei a következők:

- Az indítandó állomány neve (string)
- Az indítandó állomány parancssori paraméterei (string)
- Információ a futtatás módjáról (ez most számunkra kevésbé lényeges)
- Gépnév és architektúra (számunkra most és később sem lényeges)
- Az indítandó folyamatok száma
- Egy egész-tömb címe, ahová az elindult folyamatok `tid`-jei kerülnek

A függvény visszatérési értéke egy egész, ami a sikeresen indított folyamatok számát adja meg. Ezt összehasonlítottuk a `NUM_SLAVES` konstanssal amiben az indítani kívánt folyamatok számát definiáltuk. Ha nem sikerült az igényelt számú folyamatot elindítani, akkor kilépünk. Kilépéskor kötelezően meghívandó függvény a `pvm_exit()`.
Miután elindultak slave folyamatok, először egyenként üzenetet küldünk nekik, majd üzenetet fogadunk tőlük és azt kiírjuk a konzolra. Az egyes slave folyamatoknak elküldött üzenet a saját `tid`-jük, fogadott üzenet pedig egy karakterlánc lesz.
Az üzenetküldés folyamata három részre tagolódik:

- előkészítés
- a küldendő üzenet becsomagolása
- küldés

Az előkészítést a pvm_initsend() függvény végzi. Paramétere most legyen PvmDataDefault. Ez egy beépített konstans, jellemzően szinte mindig ezt használjuk adatküldésnél.
Az elküldendő adat becsomagolását a pvm_pkint() függvény végzi.
Ez a függvény csak int (azaz egész) típusú adatok becsomagolására használható, de a PVM lehetőséget nyújt más típusú adatok csomagolásához és átküldéséhez is. Az első paraméter a küldendő szám (számok) címe, második a darabszáma, a harmadik pedig a lépésközt adja meg (ezt hagyjuk most 1-nek).
Miután az adatokat előkészítettük, már csak el kell küldeni őket. A küldés *PVM* függvénye a `pvm_send()`, paraméterei a fogadó folyamat `tid`-je (a címzett) és az üzenet típusa, melyről már korábban szóltunk a `tags.h` fejlécfájl kapcsán. Az első paraméter nyilvánvaló, a második már némi magyarázatra szorul. Értéke gyakorlatilag teljesen ránk van bízva, leginkább magunk biztosítására használható.

2. Lista A slave folyamat

```
/* slave.c */
#include <stdio.h>
#include <stdlib.h>
#include <pvm3.h>

#include "tags.h"

int main ()
{
    int  mytid;
    int  parent_tid;
    char message[30];
    int  number;


    /* a sajat tid
    ➥lekerdezese, belepes
    ➥a PVM-be */
    mytid = pvm_mytid();

    /* a szulo folyamat
    ➥tid-jenek lekerdezese */
    parent_tid = pvm_parent();

    /* az uzenetkuldes
    ➥elokeszitese */
    pvm_initsend
    ➥(PvmDataDefault);

    /* uzenetfogadas
    ➥a szulotol */
    pvm_recv(parent_tid,
    ➥MSG_DATA);

    /* tudjuk, hogy szamrol
    ➥van szo, kicsomagolas */
    pvm_upkint(&number, 1, 1);

    /* ha az uzenet helyesen
    ➥jott meg, azaz
    ➥megegyezik a sajat
    ➥tid-del */
    if (number == mytid) {

     /* akkor kuldjunk
     ➥stringet a master-nak */
  sprintf(message, "Hello from
  ➥%x", number);
  pvm_pkstr(message);
  pvm_send(parent_tid,
MSG_DATA);
    }

    pvm_exit ();
    exit (0);
}
```

3. Lista A tags.h

```
#ifndef TAGS_H
#define TAGS_H

#define MSG_DATA  1000

#endif /* TAGS_H */
```

Itt felhasználjuk a definiált `MSG_DATA` konstanst az üzenetek típusának megadására.
Adatokat elküldtük slave folyamatoknak, melyek valahogyan feldolgozzák azokat (jelen esetben nem tesznek semmi érdekeset). Következő feladatunk a visszaérkező üzenetek fogadása és kiírása a konzolra. Ez a küldéssel analóg módon történik, de némileg egyszerűbben. Az üzenetfogadás folyamata:

- várakozás megfelelő üzenetre és fogadás
- az üzenet kicsomagolása

A `pvm_recv()` függvénnyel addig várunk, míg megfelelő adat nem érkezik. Első paramétere a fogadás módja, jelen esetben jöhet az üzenet bárkitől, semmilyen sorrendet nem határoztunk meg (azaz értéke –1), a második a várt adat azonosítója, most `MSG_DATA`.
Példánkban vissza nem egészeket várunk, hanem karakterláncokat (string).
Ha megjött az üzenet (ami egy karakerlánc), kicsomagoljuk a `pvm_upkstr()` függvény segítségével. Természetesen más típusú adatokat is fogadhatunk, csak a példa miatt tárgyaljuk a stringet. Egyetlen paramétere a fogadott adat számára fenntartott memóriatömb címe.
Az üzenet megérkezése után kiírjuk a konzolra. Ha minden üzenetet fogadtunk és kiírtunk, akkor a program kilép.
Nézzük a slave folyamat megvalósítását.
Itt már nincs sok újdonság. Ami új, az a `pvm_parent()` függvény, mely annak a folyamatnak a `tid`-jét adja vissza, ami ezt a folyamatot létrehozta (spawn), vagyis most a master folyamatunk `tid`-jét.
Ez majd az üzenetküldéshez kell.
Másik új függvény a `pvm_pkstr()`, ami egy karakterlánc (string) becsomagolását végzi.
A teljesség kedvéért a fent említett ***tags.h*** forrását a listában olvashatjuk.
Programunk fordítását az alábbi parancsok kiadásával végezzük:

```
gcc -o ~/pvm3/bin/LINUX/master1
➥master.c -Wall -lpvm3
gcc -o ~/pvm3/bin/LINUX/slave1
➥slave.c -Wall -lpvm3
```

Ha már korábban elindítottuk a ***PVM*** démont, akkor lépjünk ki belőle a ***PVM*** konzolban kiadott `quit` paranccsal. Ennek hatására a ***PVM*** démon futása nem áll meg, a háttérben továbbra is fut.
Lépjünk be a ***~/pvm3/bin/LINUX/*** könyvtárba és futtassuk le a kapott master1 állományt.
A futtatás eredményeképpen valami hasonló kell, hogy kapjunk:

```
bha@Lorien:~/pvm3/bin/LINUX$
➥./master1
Hello from 40003
Hello from 40004
Hello from 40005
Hello from 40006
Hello from 40007
Hello from 40008
Hello from 4000a
Hello from 40009
bha@Lorien:~/pvm3/bin/LINUX$
```

Az első programunk, ami végül is semmi hasznosat nem művel lefutott és bebizonyította párhuzamos működését.
Legközelebb klasztert építünk majd, hogy programunk tényleg több gépen fusson és megismerkedünk egy igazi problémával és annak igazi megoldásával is.

Bánki Horváth András
(bha@elte.hu)
Végzős programtervező matematikus hallgató vagyok az ELTE-n. Minden érdekel ami az informatikával kapcsolatos. 1997 óta vagyok aktív Linux felhasználó. Ha nem dolgozom, legszívesebben a barátnőmmel és a barátaimmal vagyok.

GRAMPS
a családfa

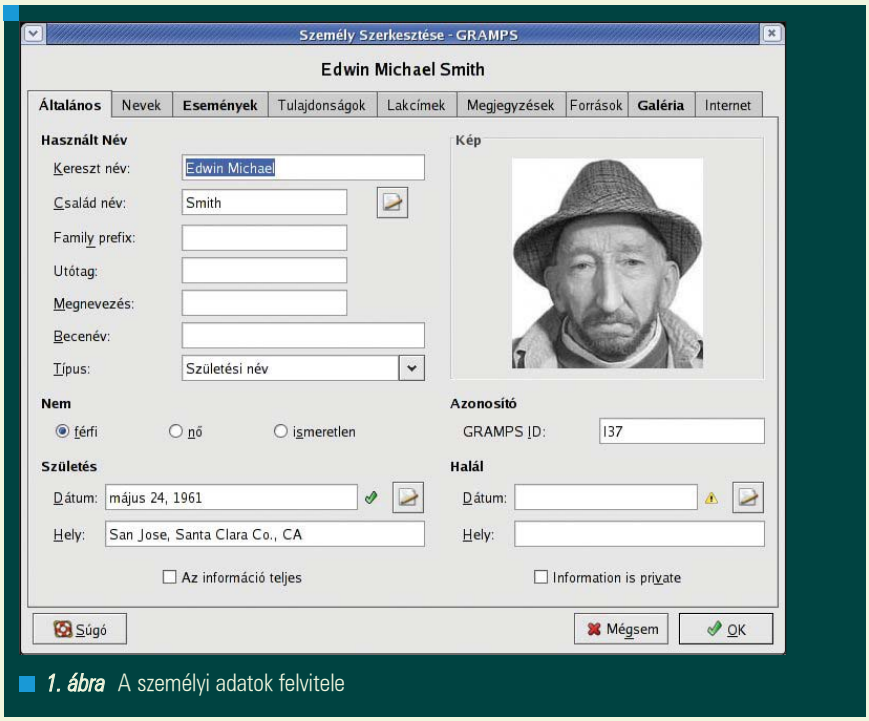
Körülbelül 9-10 éves koromban elhatároztam, hogy ameddig csak tudom, felkutatom az őseimet és felrajzolok egy családfát. Kifaggattam hát a legidősebb rokonaimat és mindent papírra vetettem. Még ma is megvan a kezdetleges változat. Sajnos a papír megsárgult és sok helyen már nehezen olvashatóak az adatok...

A modern kor megoldása

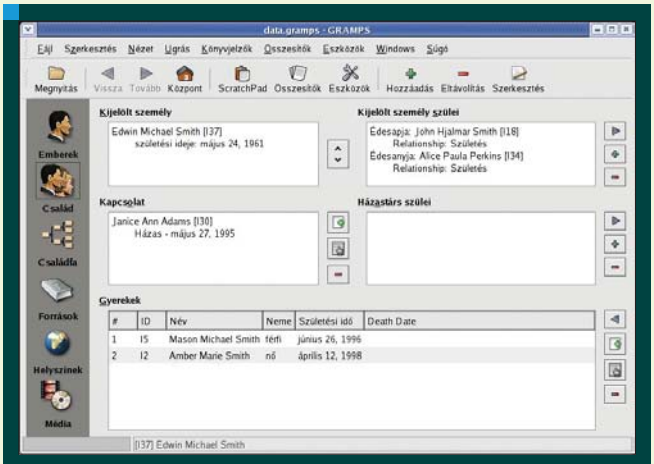
2005 decemberében akadt a kezembe a *GRAMPS* (*Genealogical Research and Analysis Management Programming System* – vagyis leszármazást kutató és analizáló programcsomag). *Linuxon, BSD-n, Solarison* és *Mac OSX*-en fut. Csupán három csomag és azok függőségei szükségesek: *python* (2.3), *gnome-python* (minimum 2.6.0), illetve *pygtk* (legalább 2.5.0). A pythonos megvalósítás tapasztalataim szerint némileg visszafogja a programot, de egy mai (legalább 1.5 GHz-es) számítógépnek nem jelent gondot a futtatása.
A program forráskódból is telepíthető, de *Debian, Fedora, Suse* és *Mandriva* esetén előrefordított csomaggal is dolgozhatunk. Utóbbi ajánlott, bár *Suse* esetén nem triviális a telepítés, érdemes tehát meglátogatni a projekt honlapját.

A program indítása

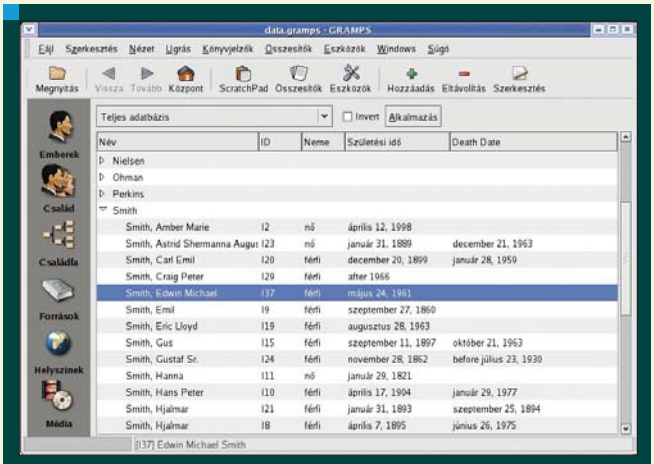
A program elindítása után megkérdezi, hogy már meglévő adatbázis töltünk-e be, vagy újjal kezdünk. Amennyiben új készítésébe kezdünk, *megkérdezi* a családfakutató (jelenesetben pl. a kedves olvasó) *adatait*. A nevet muszáj beírni, a többit nem.



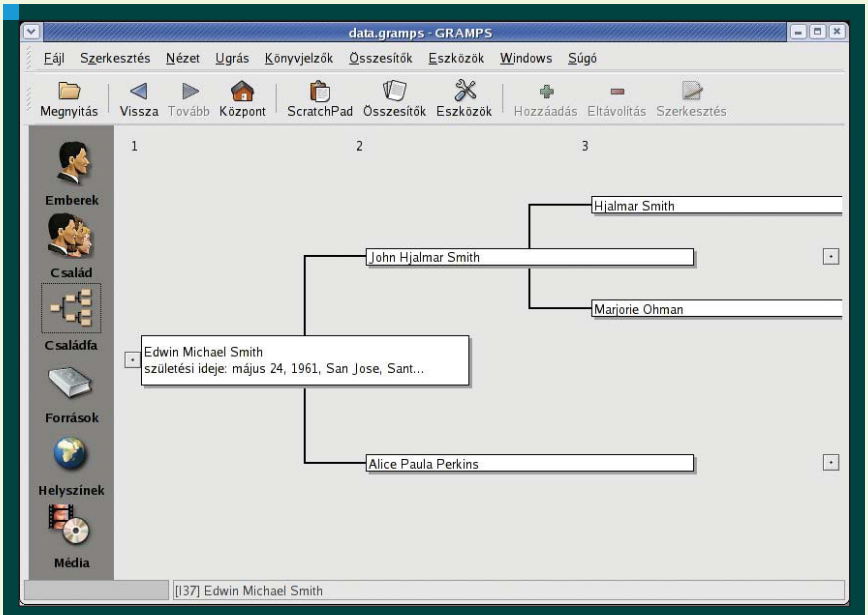
1. ábra A személyi adatok felvitele



2. ábra A családi kapcsolatok rögzítése



3. ábra Személy kiválasztása a családfa rajzoláshoz



4. ábra A „családfa" gomb hatása – közvetlen ősök

Jó hír az angolul nem beszélőknek, hogy ***magyarul*** is ***tud*** a rendszer, igaz néhol hiányosan. A nyelvet a ***LANG*** környezeti változóval állíthatjuk be (`export LANG=hu_HU`).

Növekszik a fa

A programot elsőre nem könnyű átlátni és vannak olyan menük, amiket például én nem is használtam eddig és valószínűleg nem is fogok. No de vágjunk bele. A baloldalon található az *„Emberek"* gomb. Ezzel egy embert lehet fel venni (*1. ábra*). A menü alatti vízszintes eszköztáron a „+" gombbal tudunk felvenni egyedeket, a „-" gombbal pedig törölhetünk a listából. Az adatbázis rengeteg különféle adatot tud tárolni, a ***fülek*** magukért beszélnek.

Indulásnak mondjuk vegyük fel a szüleinket, a testvéreinket és magunkat. Ezután klikk a *„Család"* gombra. Itt lehet a családi kapcsolatokat felvinni. Mármint ki kinek a mije. (*2. ábra*) Házasság esetén a házasságkötés dátuma is bejegyezhető. Érdemes persze ***előbb*** az összes ***embert*** felvinni és csak ***utána*** a ***kapcsolatokat***.
Ha mindenkit felvittünk, akkor érdemes egy mentést nyomni.
Megjegyzés: a képernyőképek nem a saját családfámból származnak, hanem a ***gramps*** mintaadatbázisával készültek, ami a ***Súgó*** menüből érhető el.

Lekérdezések

Ez az, amiért felraktam a ***GRAMPS***-ot. Ha például a *„családfa"* gombra bökünk, akkor az *„Emberek"* gombnál kiválasztott személy (*3. ábra*) közvetlen őseit felrajzolja a rendszer (*4. ábra*). Azonban az ***Összesítők*** menü ***Grafikus összesítők*** menüpontját használva különféle lekérdezéseket el is menthetünk emészthető formában. Ami nekem kellett, az a ***Kapcsolat grafikon*** készítése. A program számtalan formátumba tudja exportálni a grafikonokat (***pdf, ps, svg***, stb.)

Konklúzió

Az újság keretei sajnos nem teszik lehetővé a program mélyebb ismertetését, annál is inkább mert sokak számára a családfakutatás csupán múló hóbort. Én azonban büszkén fogom mutatni a utódaimnak a GRAMPS-al készült, immár maradandó családfát.
Szükség esetén a program fejlesztőit ***IRC***-n (***irc.freenode.net***) ***a #gramps*** csatornán lehet elérni. Sikeres családfakutatást kívánok annak, aki belevág.

Medve Zoltán
(e-medve@e-medve.hu)

2001-ben kezdett „Linuxolni", de már korábban is ismerkedett a szabad szoftverek világával.
Ha éppen nem a gép előtt ül, akkor fotózgat, olvasgat vagy bicajozik.

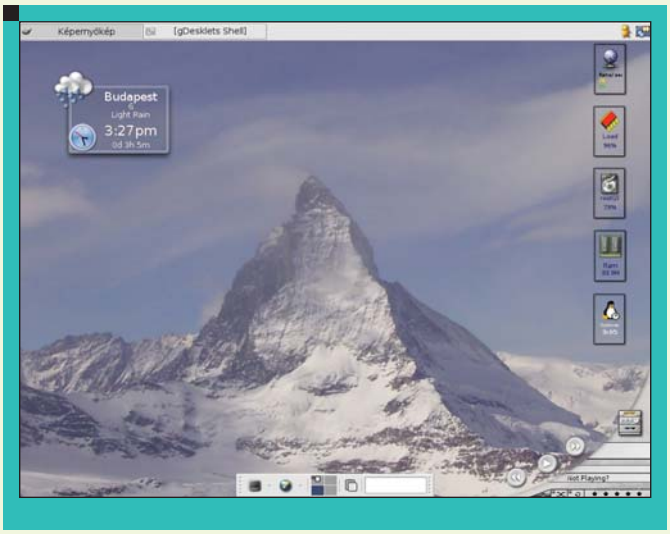
KAPCSOLÓDÓ CÍMEK

GRAMPS projekt honlapja
➲ http://gramps-project.org/

Dobd fel a desktopod – Gdesklets

Aki szereti otthonossá, egyedivé tenni a környezetet amiben munkáját végzi, az operációs rendszerét (munkaasztalát) is fel fogja tudni dobni.

Ez gyakran egy egyedileg kiválasztott háttérben, egy stílusos ikonkészletben valamint a maximálisan testreszabható elrendezésben kimerül. Másfelől aki szereti folyamatosan különböző a hardverekkel kapcsolatos valamint számítógépen kívüli információhoz jutni, nos azok most találkoznak a *gDesklets* programnak köszönhetően. A program funkcióját tekintve kettősnek mondható: egyrészt díszítőelemként feldobja és egyedivé varázsolja az asztal hangulatát, másrészt információt szolgáltat a kiválasztott forráspontokról.
Maga a program a *Gnome* asztali környezethez tartozik, de természetesen működik más felületeken is, ha megvannak az alapjait képző alkalmazások: *Gtk 2.0*-re valamint a *Python*-ra támaszkodik. Amennyiben rendszerünk tartalmazza a *Gnome* asztali környezetet, akkor valószínűleg része a rendszernek a program, de nincs telepítve, akkor is csomagként ott lesz valamelyik telepítő lemezen. Csomagból való telepítés egyszerűen elvégezhető, fel kell tenni a *gdesklets* és *gdesklets-data* csomagokat, melyet rutin művelet lévén most nem részleteznék ki. A program funkcionalitását tekintve megegyezik a *KDE*-s *SuperKarambával*.



De lássuk, hogyan is néz ki ez az egész a gyakorlatban. Sikeres telepítés után elindítjuk a *gdesklets* démont (konzolon a `gdesklets start` utasítással; grafikusan meg az adott disztribúció szolgáltatás/programkezelőjében), aminek hatására, ha minden rendben zajlott, megjelenik egy kirakós darabokat ábrázoló, kékes árnyalatú ikon a rendszertálcán. Ez jelzi, hogy a démon fut.
Ha szeretnénk a rendszer indulást követően automatikusan elindítani a démont és az asztali környezet/ablakkezelő nem teszi lehetővé, akkor a *~/Desktop/Autostart/* könyvtárba hozzunk létre egy futtatható állományt, mely tartalmazza a

```
gdesklets start
```

parancsot. Visszatérve a beállításokhoz: az előbb említett az ikonra történő jobb-kattintással végezhetjük el. A felugró menüből válasszuk az ***Asztali kisalkalmazások kezelése*** menüpontot: itt jön a lényeg, megjelenik egy ***gDesklets Shell*** nevet viselő ablak, ahonnan a különböző kisalkalmazásokat választhatjuk és indíthatjuk el. Ezeket az úgynevezett kisalkalmazásokat helyezhetjük el a munkaasztalon, oda ahova nekünk jól esik: az „odaragad” és mintegy aktív részévé válik a háttérnek. Meglehetősen látványos, valamint hasznos tud lenni egy-egy ilyen kisalkalmazás.
A választék széles, én négy kategóriába sorolom be most őket: hardver (állapot), rendszer (szoftver állapot), külső információ és egyéb.
A program kategória rendszere

ennél részletesebb, valamint lehet benne ábécésorrendben és szerző szerint is keresni.
A hardverrel kapcsolatos adatot szolgáltató funkciók, ilyenek mint a processzor, memória, partíciók és cserélhető médiák állapota, hőmérséklet, ventilátor fordulatszám és még sorolhatnám. Ezek általában az adott eszköz állapotát közlik valamilyen ikon/grafika segítségével, meghatározott időközönként frissítve az információt. Aztán vannak a rendszerrel kapcsolatos kisalkalmazások, mint az óra, rendszerterhelés, uptime, riasztások, naptárak... valamint a különböző programvezérlő alkalmazások (***xmms***, ***toolbar***). Szintén mint a hardveres kijelzők, ezek is a betűméret/színig testreszabhatóak.
Harmadik kategóriába tartozik az összes olyan alkalmazás, ami külső információt szolgáltat, vagyis külső forrás szükséges (azaz élő internetkapcsolat meglétét feltételezi). Ebbe tartozik az email figyelőtől kezdve az időjárás előrejelzőn keresztül az ***RSS***-olvasóig minden. Ezek meghatározott időközönként (szintén változtatható gyakorisággal) lekérdezik a kijelzőhöz tartozó szerverről az információkat.
Egyéb kategóriában azok kapnak helyet, amik sehogyan sem sorolhatók az előző háromba, ilyen a véletlen kép, a nap idézete vagy a halas akvárium. Mindezeken túl még

bővíthető a választék különböző hivatalos és nem hivatalos oldalakról. Érdemes meglátogatni a ➲ ***http://gdesklets.gnomedesktop.org/*** weboldalt is, ahol frissítések is találhatóak.
Természetesen minden ilyen alkalmazás látványos és hasznos, de személy szerint azokat kedvelem, amelyek számomra is hasznos információt szolgáltatnak, mint a ki és bemenő internetforgalom, az újonnan érkezett levelek – gyors reagálás titka, hogy (beállítástól függően) 5 percen belül kézhez kapom a levelet, valamint a kijelzőre mért egyetlen kattintással el is indítja kedvenc levelező programom. Időjárás előrejelzés is hasznos dolog, bár gyakran van pár fok eltérés a helyi hőmérő és a megadott város átlagos hőmérséklete között, de ez nem számottevő.
Ami innen inkább hasznos, az a megjelenített grafika, ami jelzi, hogy éppen süt a nap, látható a hold, szélvihar közeleg esetleg ködös az idő. Óra... rengeteg óra megoldás létezik (megkockáztatom, hogy abból van a legtöbb) az egyszerű digitális órától kezdve a szép analóg zsebórán át egészen az elvontnak mondható, bináris vagy hexadecimális számrendszerben kijelzett pontos időig. Nos szerintem a képek magukért beszélnek, némi kreativitással és kitartással csodálatos dolgokat lehet alkotni. Nem utolsó sorban a haverok is el fognak sárgulni az irigységtől...

Rácz Zoltán
(razoli@linuxforum.hu)

Jelenleg egyetemista az ELTE informatika-matematika tanári szakán. A Linuxszal két éve került kapcsolatba az UHU 1.0 kapcsán. Fél éve egyetlen operációs rendszer van a gépén: egy Debian Sid.

Kettőslátás 1. – Kétpaneles fájlkezelők

Minden felhasználó, aki levelezésnél vagy a webböngészésnél többre kezdi el használni a számítógépét előbb-utóbb szembetalálkozik a fájlkezelés problematikájával. Fájlokat szeretne keresni, másolni, vagy esetleg rendszerezné a digitális fotóit a merevlemezen. Ezek a feladatok valamilyen fájlkezelő programot igényelnek.

Ami minden *Linux* terjesztésben megtalálható az a parancssoros megoldás: ***ls, cp, mv, ln, rm, cat, mkdir, find, locate,*** stb. Ezek azok a parancsok, amelyek minden rendszerben ott vannak, de az átlagos felhasználó vagy sohasem találkozik velük, vagy csak hosszas tanulás után tudja őket jól használni.
A grafikus operációs rendszerek egy paneles fájlkezelőt tartalmaznak alapállapotban, ez a gyakorlatlanabb felhasználók egér orientáltságának tudható be. ***Linuxban*** ez a választott disztribúciótól, ablakkezelőtől függően ***Konqueror*** vagy ***Nautilus*** a legtöbb esetben. Ezek a programok az egérrel vonszolós technika látványosságán és egyszerűségén alapulnak (lehetőség van a billentyűzettel való kezelésre is, de ezt valószínűleg jóval kevesebben használják), mivel ezt az írni-olvasni már tudó gyerek is használatba tudja venni.
A kettő közti megoldásként írta meg évtizedekkel ezelőtt ***John Socha*** a saját kis programját, ami minden két paneles fájlkezelő őse. Ez lett ***Peter Norton*** a cégalapító nevéből adódóan ***Norton Commander***. Ennek a programnak később nagyon sok klónja keletkezett. Ezt a programcsoportot szokták ***OFM (Orthodox File Manager)*** programoknak is nevezni az ortodox szó szabályokhoz ragaszkodó jelentéséből adódóan.
A ***Linux*** világában karakteres felületen a ***Midnight Commander*** nevű program terjedt el általánosan. Grafikus felületen az ablakkezelők eltérő függvénykészlete miatt több programot használnak:

- ***GTK*** – grafikus könyvtárra épülő: ***Gnome Commander***, ***Tux Commander***, ***emelFM2***
- ***Qt*** – grafikus könyvtárra épülő: ***Krusader***
- Grafikus könyvtáraktól független ***X Window*** alkalmazás: ***Northern Captain***.

Más rendszerből érkezők ezek között találhatják meg, mit szeretnének használni a ***Total Commander***, vagy valamelyik klónja helyett.
A következőkben áttekintjük azokat az alapfunkciókat, amelyekkel valamennyi komolyabb fájlkezelő alkalmazás rendelkezik.

Alapfunkciók

Két panel, egy aktív és egy passzív:
Az alapműveletek (másolás, mozgatás, szimbolikus hivatkozás készítése) az aktívból indulnak, célpontjuk pedig a passzív ablakban mutatott hely.
E kettő között általában a Tab billentyűvel lehet váltani.

Parancssor integráció: Ez elsősorban a karakteres megvalósításnál fontos, mivel grafikus felületen a tetszőleges számú terminál ablak megjelenítése miatt kevésbé szükséges. Lehetőség lehet az egyik, vagy mindegyik panel ideiglenes eltűntetésére, hogy a kiadott parancs kimenete látható legyen.

Billentyűparancsok: A gyors használat érdekében a funkció billentyűkre van beállítva néhány parancs a „történelmi" hagyományok szerint, úgymint

- ***F1 – Súgó***
- ***F2 – Felhasználói menü / Terminál***
- ***F3 – Megjelenítés***
- ***F4 – Szerkesztés***
- ***F5 – Másolás***
- ***F6 – Áthelyezés,***
- ***F7 – Új könyvtár***
- ***F8 – Törlés***
- ***F9 –Menü / Átnevezés***
- ***F10 – Kilépés***

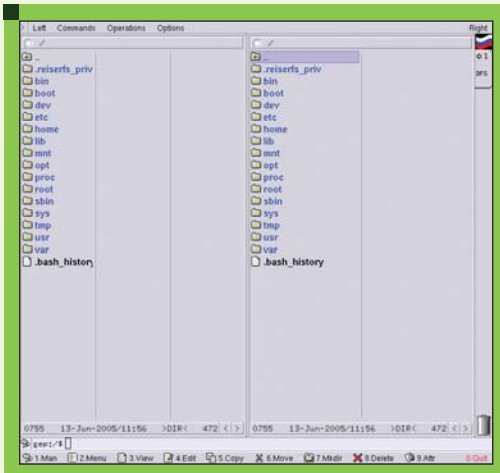
A program alsó sorában az ezekhez a funkciókhoz tartozó billentyűk megjeleníthetők, és egérrel rábökve is használhatók. Ezek az alapműveletek az egérrel vonszolós használattal is elérhetőek. (Az adott program telepítése után érdemes ellenőrizni a beállításokat!)

Fájltípustól függő funkciók: Az állomány megnyitásához, szerkesztéséhez, futtatásához. Ez a ***GNOME*** és a ***KDE*** használata esetén alapállapotban is jól használható, de mindenki egyedi ízlésére szabhatja.

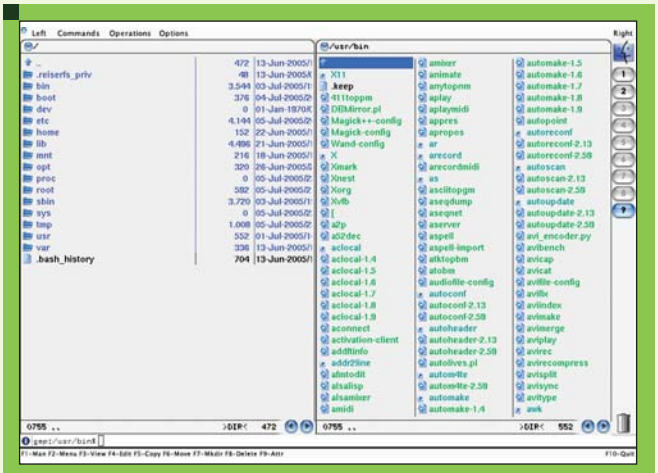
Az előzmények megjegyzése: Minden beírást váró mező esetében (másolás helye, keresett kifejezés, parancssor, stb.)

Virtuális fájlrendszerek kezelése: Olvasni ezek mindegyikét lehet, de írásuk korlátozott egyes esetekben.

- ***FTP***, ***SFTP***, ***Fish***, ***SAMBA***, ***NFS*** és más nem lokális fájlrendszerek
- ***ISO CD / DVD*** lemezképek
- ***rpm***, ***deb*** és hasonló disztribúció specifikus csomagok



1. ábra Ezt látjuk elindítás után



2. ábra Egy kicsit szebb is lehet...

- *tar*, *tar.gz*, *tar.bz2*, *gz*, *bz2*, *arj*, *rar*, *ace*, *zip* és más archívumok
- a keresés eredménye is megjeleníthető egy panel tartalmaként
- „Kisimított" fastruktúra- az aktuális könyvtárat az alkönyvtárak tartalmával együtt egy listában jeleníti meg

Beépített egyszerű szerkesztő és megjelenítő program: Ezek a hagyományos szerkesztők a legegyszerűbb szövegszerkesztési funkciókra képesek, nem hasonlíthatók az irodai programcsomagok szövegszerkesztőihez. Némelyik megvalósítja a fájltípushoz igazodó színes kiemeléseket, a hexadecimális szerkesztést, különböző kódtáblák kezelését, és a ***Windows – Unix – Macintosh*** világban eltérő sorvégek kezelését. Lehetőség van a fájl megtekintés funkciót az egyik panelra helyezni is.

Keresési lehetőség: A fájl neve, dátuma, mérete, tulajdonosa, jogosultságai alapján. A keresési kifejezés lehet egyszerű szöveg és dzsóker karakterek, vagy reguláris kifejezés. Nemcsak a fájlok tulajdonságai, hanem a tartalmukban történő keresésre is mód van. A tartalom szerinti keresésnél oda kell figyelni, hogy ne legyen túl sok eleme az átnézendő listának, mert ez időigényes folyamat. A keresés eredményeként létrejövő lista, akár egy virtuális fájlrendszerré is tehető, így megkönnyítve a további munkát a megtalált fájlokkal.

A könyvtár rendszer fastruktúrájú megjelenítése: Így könnyen követhető, hogy a fájlrendszer mely részén barangolunk éppen, illetve hol találjuk azt a könyvtárat, amit pl. másolásnál célként szeretnénk megadni. Bizonyos programokban az aktuális könyvtárra vonatkozó helyfoglalási adatokat is kaphatunk. A fastruktúrájú panel aktiválása után, azok is jól boldogulhatnak ezekkel a kétpaneles programokkal, akik korábban ***Windows*** alatt ***Explorert*** használtak.

Fontos helyek gyors elérése: Lehetőség van az internet böngészéshez hasonlóan könyvjelzők elhelyezésére is, így akár egy billentyűkombinációval ugorhatunk a múlt heti nyaraláson készült fotók könyvtárába.

Gép-gép közti kapcsolat: Az internet korszak előtti időkben a kétpaneles programok jellemzője volt a párhuzamos vagy soros porton összekötött gépek közötti direkt kapcsolat megvalósítása. A technika fejlődése nyomán manapság inkább a hálózaton, ***TCP/IP*** protokollon keresztüli direkt összeköttetések kezelése a jellemző. A kapcsolat akár titkosított is lehet, megvalósítástól függően.

Jó testreszabhatóság: Ami nem a megjelenítést jelenti elsősorban, hanem a billentyűkombinációkhoz és egérműveletekhez történő parancsok hozzárendelést.

Elsőként nézzünk egy ***X Window*** rendszerre készült megvalósítását a két paneles fájlkezelésnek.

Az X Northern Captain 5.0.4

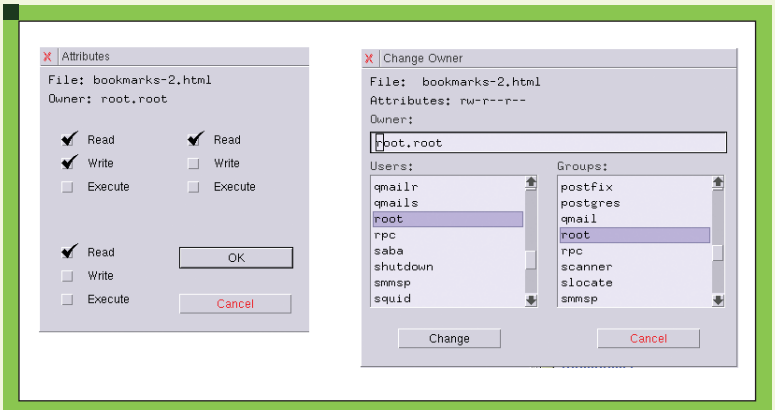
A programot egy szentpétervári programozó fejleszti, valószínűleg innen ered az elnevezése. Ha az éppen használt terjesztésben nem található, akkor a következő címről tölthető le:
➲ ***http://xnc.dubna.su/*** A program sajnos csak részben magyarított. Az elindítása után egy puritán képernyő fogadja a felhasználót, az alapbeállítás szerinti listák csak az állományok nevét jelenítik meg.
A divatos trendeknek megfelelően, még ez a kis programocska is „újrabőrözhető".

Left – Right menü

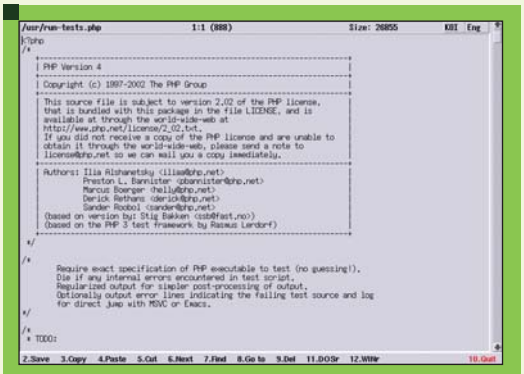
A jobb illetve baloldali beállítások külön szabályozhatóak. A menüből háromféle megjelenítésből választhatunk, de hogy melyik mód milyen oszlopokat tartalmazzon, azt az ***Options – Configuration – Módok*** menüpontban szabályozhatjuk részletesen. A lista elemei sorba rendezhetőek különböző szempontok szerint, de lehet rendezetlen és fordított sorrendű is. Amennyiben például a ***Rövid*** listamódot választjuk 1 vagy akár 4 oszlopban is láthatjuk folytatólagosan. Mindenestül meg lehet jeleníteni a könyvtárat az alkönyvtárak tartalmával együtt egy listában is.

Commands menü

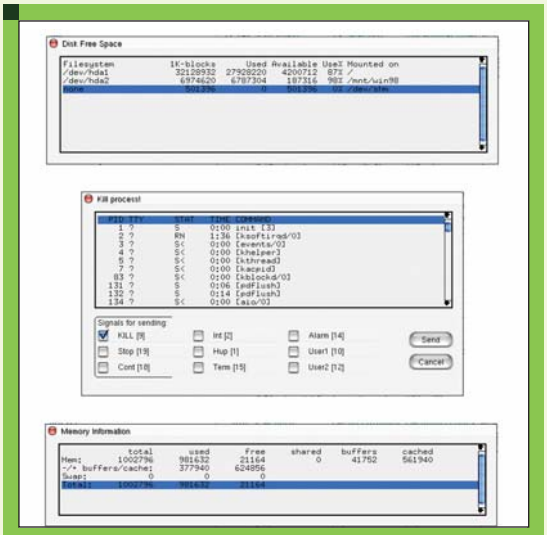
Ebben a menüben találhatók a hagyományos ***Szerkesztés, Másolás, Törlés, Keresés, Könyvtár vagy link létrehozása*** funkciók. Bizonyos esetben hasznos lehet az aktuális könyvtár újramountolása menüpont is. Az állomány



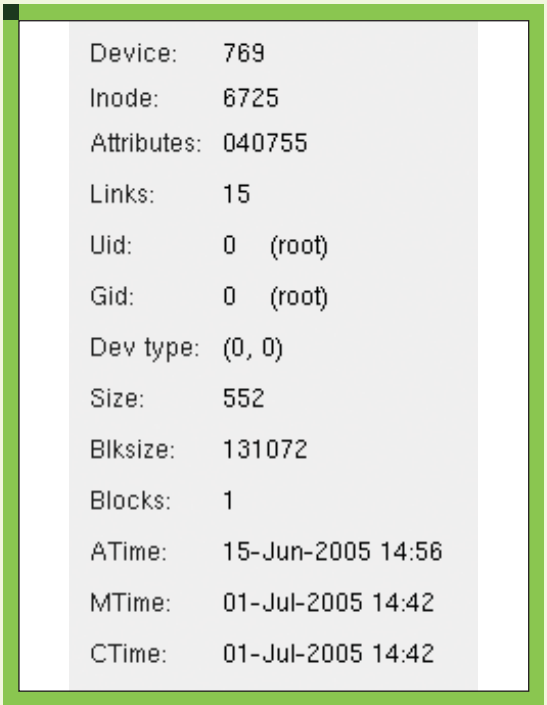
3. ábra Az állomány tulajdonságainak beállítása



6. ábra A beépített megjelenítő program



4. ábra Hasznos információk a rendszerről



5. ábra Minden amit egy fájlról tudni lehet

tulajdonosa, illetve a hozzá tartozó jogosultságok egyszerűen beállíthatók. Létrehozhatunk tömörített archívumokat az aktuális vagy a kijelölt állományokból. Megtaláljuk még ebben a menüben a *df* és *ps* parancsok ablakos megvalósítását. A memória és a swap felhasználásról is kaphatunk adatokat.

Egy külön ablakban megjeleníthető az aktuális fájl minden adata.

Ha *FTP* kapcsolatot szeretnénk használni, akkor sincs szükség újabb program indítására. A biztonságos csatlakozásokra nem, viszont proxy kezelésre fel van készítve. Miután minden szükségest letöltöttünk az összehasonlítás funkcióval ellenőrizhetjük, mi az ami már meg volt.

Operations menü

Ebben a menüben találjuk a panelek cseréje, az aktuális könyvtár újraolvasása, a terminál ablakra váltás funkciókat. Lehetőség van kiválasztó, ill. kiválasztást megszüntető maszkok megadására, ha például csak a *HTML* fájlokkal szeretnénk dolgozni a továbbiakban. Egyes gyakran használt könyvtárakhoz könyvjelzőket rendelhetünk, melyekre a jobb szélen található ikonokra kattintva válthatunk.

Options menü

A menü fájlt mindenki a saját ízlése, és igénye szerint szerkesztheti. Az itt felsorolt parancsok az *F2* gomb lenyomására megjelenő listából indíthatók. Mivel két linuxos biztosan nem ugyanazokat a programot használja minden fájltípus esetében, ezért ki-ki a saját szája íze szerint állítsa be a kiterjesztéstől függően melyik alkalmazás induljon el. Az alapbeállítás legtöbb esetben a programhoz készült megjelenítő programocska.

Amennyiben az adott típushoz még nincs társítva alkalmazás, a már társítottak listája jelenik meg és abból választhatunk. A *Configuration* menüpontban beállítható a program kinézete, mely műveleteknél kérjen megerősítést, és az egérgombokhoz társított műveletek.

Összefoglalás

Noha már körülbelül 1 éve nem jelent meg új verzió, a program az alapfunkciókat jól teljesíti. Amennyiben valakinek problémát okoz a részleges magyarítás, neki nem ajánlom.

Előnyös tulajdonsága, hogy sem a *Qt*, sem a *GTK* eljárás gyűjteményhez nem kötődik. Így elsősorban azoknak ajánlható akik a gépük kis teljesítménye miatt, vagy csak mert szeretik az egyszerűséget nem igényelnek többet. Hiányolom a programból, hogy nem tárolja a különböző adatbekéréseknél, illetve a terminál részben az előzményeket.

Szabolcsi Csaba
(szabolcsi@walla.com)
A Linuxot 1994-ben ismerte meg, jelenleg Gentoo-t használ. Szeret olvasni és főzni.

IskolaPortál – Moodle harmadszor

2004 novemberében és 2005 márciusában Horváth Ernő tollából már napvilágot látott két cikk a Linuxvilágban a legjobb ingyenes e-Learning keretrendszerről, a Moodle-ról. Akkoriban még világviszonylatban mindössze 1843 regisztrált Moodle alapokon nyugvó weboldal volt. Mára ez a szám 75000 fölé emelkedett, 70 nyelven, 138 országból vannak felhasználók. A cikk írásakor az 1.5.3+ a legfrissebb stabil kiadás, de létezik az 1.6-os fejlesztői változat is.

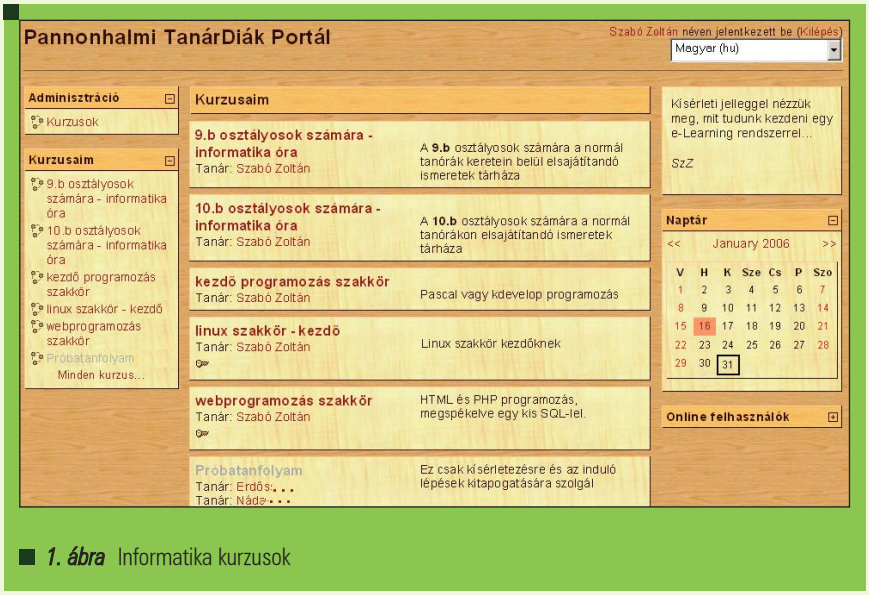
Áttekintés a változásokról és a dokumentációról

A „helyzet komolyságát” jelzi, hogy a ➲ *www.moodle.org*-on kívül van egy ➲ *www.moodle.com* weboldal is. Itt segítséget adnak mindazoknak, akik hajlandóak ezért fizetni. A program fejlesztését koordináló központ Ausztráliában van; a projekt vezetője *Martin Dougiamas*. A *www.moodle.org* oldal köré csoportosult lelkes csapat azonban az esetek túlnyomó többségében (ingyenesen) megválaszolja a felmerülő kérdéseket. Magam is kértem, kaptam már segítséget a fórumokon – érdekes módon többnyire hölgyektől.

A *Moodle* magyarítását *Bozsa István (bozsa@jht.gau.hu)* kezdte el, az *SZTE Távoktatási Központja* részéről *Dr. Fábricz Károly (kfabricz@mail.u-szeged.hu)* folytatta. Hihetetlen mennyiségű munkát jelentett a rengeteg kifejezés, súgószövegek lefordítása – ezt ne felejtsük akkor sem, amikor esetleg helyenként angol részekbe botlunk.

A *www.moodle.org* oldalon található dokumentációban nem túl könnyű megtalálni, amit szeretnénk (legalábbis nekem nem mindig sikerült).

Sok esetben hatékonya(bba)n tudtam használni a saját *Moodle* rendszerünk induló *URL*-je mögé illesztett */doc* alatt megjelenő, baloldalt található *„Other Docs”* részt, ezen belül is a *Súgó* (*Help*) oldalakat. Az on-line oldalakon is vannak olyan eligazító oldalak, „kurzusok”, amelyeket át lehet emelni saját rendszerünkbe, és így off-line módon tanulmányozhatóak.



1. ábra Informatika kurzusok

Hol használjuk?

Meglátásom szerint egy mai normál középiskolában az informatika-közeli tárgyakon és szakkörökön kívül nemigen van olyan szituáció, amikor a klasszikus oktatással versenyre kelhetne egy ilyen – lényegében távoktatásra tervezett – rendszer a maga kurzusaival. Az informatikával azért teszek kivételt, mert egyelőre nincs megfelelő, *Linuxra* és más szabad operációs rendszerre építő tankönyv, amit jó szívvel kézbe lehetne adni a gyerekeknek. Ez az a tantárgy, amelyben a legszórtabb a tanulók tudásszintje, s így jól tud jönni egy hipertext-alapú tananyag, amelyben a gyengébbek is végére érhetnek az alapvető ismereteknek, de az okosabbak sem unatkoznak, mert ízlésüknek megfelelően mélyebbre áshatnak az anyagban (ha a tanár jól előkészítette azt). Megfelelő ügyességgel hatékonyan lehet ötvözni a *Moodle*-alapú és a normál órai stílust. A tanár munkája persze a tanórán lecsökken, de az előkészítés nagyságrendekkel több időt felemészt, mint egy normál óra esetében.

Használat közben kiderült, hogy a *Moodle* közel sem pusztán arra jó, ami a hangzatos „e-Learning” szó asszociációi nyomán ébred bennünk.

■ *2. ábra* Egy „linux szakkör” (mint kurzus) látványa szerkesztés közben

Minden tanár óráin adódhat olyan pillanat, amikor jól jönne, ha tanítványai számára elérhetővé tudna tenni egy-egy szöveget, képet, filmet, hangot, zenét, de ott az órán ez nem megvalósítható valami okból (például nincs nála, vagy nincs kéznél számítógép vagy projektor stb.). Az a szép a ***Moodle***-ban, hogy sok kérdés hatékonyan oldható meg vele ilyen esetekben is.
Kézlegyintve gondolhatná azt a kívülálló, hogy nyilván van az iskolának weblapja, tegyék fel oda a tanárok a dolgaikat. Csakhogy ez ingoványos területre vezet az iskolai weblaprendszer biztonságát illetően, a tanárok informatikai szakértelmének tekintetében, a kialakuló oldal képének kuszaságáról nem is beszélve, valamint az is szempont lehet, hogy nem mindig szeretné a tanár mindenki számára láthatóvá tenni az éppen esedékes anyagot vagy dolgozatkérdéseket.

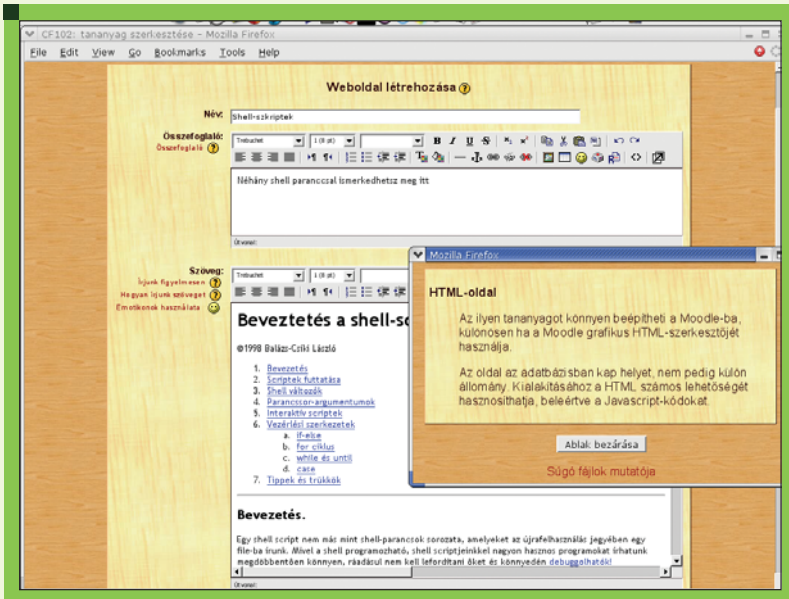
A ***Moodle*** a maga egyszerű eszközeivel lehetővé teszi, hogy a(z erre kapható) tanárok – akár csak egy-két – anyagot felvigyenek, megfelelő struktúrába illesztve és az illetékes diákok számára láthatóvá téve ezeket.
(Ha csak néhány téma kerül felvitelre, akkor nyilván nem a „heti”, hanem a „téma” típus a javasolt formátum az érintett kurzus számára.) A tanárok „e-munkára bírása” nem kifejezetten informatikai feladat, mégis, az iskolákban általában egy személy testesíti meg a technikai hátteret és a „szociális munkást” (aki segít a rászorulóknak vagy a kérdésekkel küszködőknek), sőt az „e-marketingest” is, aki buzdítani szokta a billentyűzettől idegenkedőket. Eme erőfeszítéseket nagyban segíti, ha van bátorságunk egy-két ponton hozzányúlni az alapértelmezett értékekhez, vagy ad abszurdum magukhoz a .php fájlokhoz.
Ezzel sok ember munkáját könnyebbé lehet tenni.

Tapasztalatok az első lépések után

Érdemes bármikor, tanév közben is úgy dönteni, hogy elkezdünk ismerkedni a ***Moodle***-val. Az talán kizárt, hogy valaki augusztusban úgy dönt, hogy szeptembertől teljes sávszélességgel ***Moodle*** az irány. Sok-sok apró felfedezni való van, amit nem érdemes rögtön egy éles ***Moodle***-tanév keretein belül megtenni, mert az igen kínos tud lenni, ha alapvető dolgokkal nincs tisztában a tanár (például hogy jelszót nemcsak a diák tud változtatni, hanem az adminisztrátor is). Érdemes a kurzuskategóriákat tanárok szerint kialakítani, nem témák szerint (amíg nincs túl sok tanár a porondon). Az adminisztrátor tanárt (egyelőre) csak akkor tud kijelölni a ***Felhasználó/Tanár hozzárendelése*** menüpontban (*8. ábra*), ha van már legalább egy kurzusa (nem pedig csak egy neki szánt kurzuskategória).
Ahhoz, hogy egy tanár maga dönthesse el, milyen kurzusokat indít, egyszerűen szerzőnek kell titulálni (az adminisztrátor teheti ezt meg a ***Felhasználó/Szerzők hozzárendelése*** alatt). Talán mondanom sem kell, hogy a fenti tanárkijelölési nehézséget tilos úgy megoldani, hogy a tanár felhasználóinkat rendszergazdának nevezzük ki, hogy mindent megtehessenek – remélhetőleg egy ***Linuxvilág*** olvasó ezen a megközelítésen már túl van. Én úgy orvosoltam ezt a „legyen tanár aki tanár” problémát, hogy az „Egyéb” kurzuskategóriában csináltam egy próbakurzust (amibe mellesleg beletettem egy olyan olvasmányt, ami a tanárok kezdő lépéseit segítheti, és ami innen letölthető: ➲***http://www.osb.hu/z/moo.html***). Ehhez a próbakurzushoz aztán mindenkit, aki ***IRL*** tanár (***„in real life”***, azaz ***„a valóságban”***), ***Moodle***-tanárként rendeltem hozzá.

■ *3. ábra* Kurzuskategóriák szerkesztés közben

■ *4. ábra* Kurzuskategóriák, ahogy a diákok is látják – kurzusokkal együtt



■ *5. ábra* Weboldal szerkesztése a Moodle-ban

■ *6. ábra* Adm./Beáll./Modulok - a legtöbb „tan-egység” alapértelmezései beállíthatóak

Nyúljunk bele

A barkácsolni csak egy kicsit is szeretők számára jó hírem van. A *Moodle* fájljai és adatbázisa remekül átgondolt, nem túlbonyolított rendszert alkotnak. Szemben például az *„eZ Publish”*-sal, ami valamelyest rokon a *Moodle*-val, és amit néhány éve webmesterként volt alkalmam közelről megnézni, egészen meglepő a *.php* fájlok egyszerűsége és az adatbázis struktúrájának áttekinthetősége. Többször jártam úgy, hogy szerettem volna valamit átalakítani, de – nem szánván elég időt a webes felület beállítási lehetőségeinek feltérképezésére – nem találtam, hogy hol változtassam a változtatandót. Ekkor csináltam egy mentást az adatbázisról (dump), abban rákerestem egy-két szóra, és pillanatok alatt megvolt, hogy mit hogyan kell módosítani. (Aztán előbb-utóbb, esetleg egy moodle-fórumon feltett kérdés válaszaként érkező segítség nyomán megtaláltam, hogy hogyan kell azt szépen, webfelületről megoldani.) Hasonlóképpen, néhány jól irányzott `grep -ri` paranccsal a *.php* fájlokban is el lehet igazodni.
Néhány példa: Engem zavart, hogy az új felhasználó kézi létrehozásakor annak neve „changeme”, mert túl sok betűt kell visszatörölni a megváltoztatáshoz, vagy éppen ki kell jelölni az egészet, ami esetleg az előzőleg vágólapra tett felhasználónevet felülírja. Így aztán az *admin/user.php*-ben a

```
$user->username  = "changeme";
```

sort átalakítottam:

```
$user->username  = "n";
```

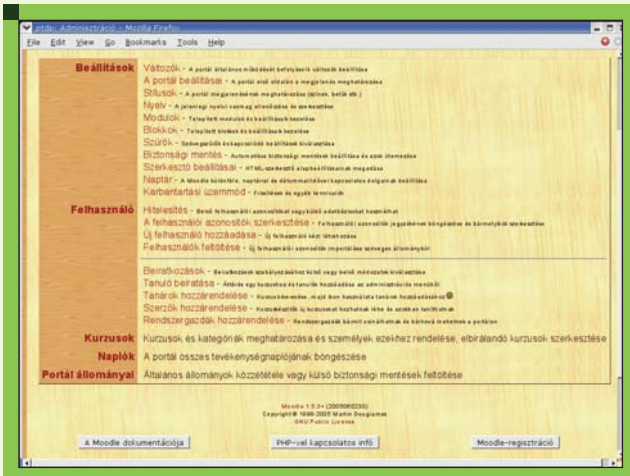
Hasonlóképpen, ugyanebben a fájlban az alapértelmezett üres emailcímet és városnevet is ízlésemnek megfelelően bedrótoztam:

```
$user->email     =
➥"@iskolai.emailcim.hu";
$user->city      =
➥"Pannonhalma";
```

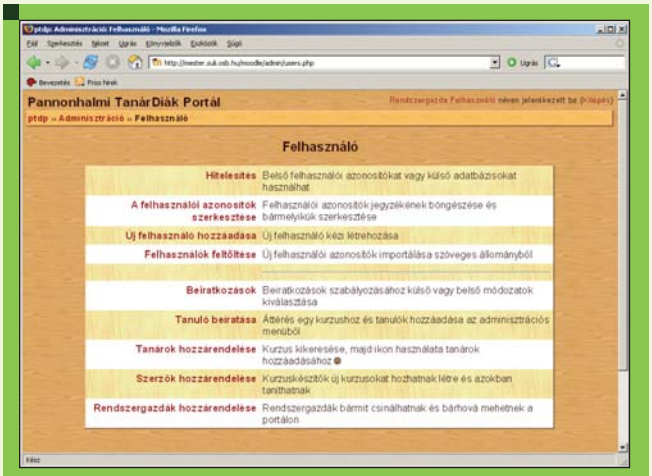
Ez utóbbinak valóban van egy kis idegzetkímélő utóhatása, mert a kifelejtett városnév esetén a személyes adatok megváltoztatását lehetővé tevő űrlap nem enged továbblépni, és ez bosszantó időveszteséget és a rendszer presztízsveszteségének előmozdítását eredményezi.
Egy másik példa a testreszabásra: a tananyagok közül a *Weboldal létrehozása* vagy csatolása esetén alul alapértelmezetten az volt kijelölve, hogy „ugyanabban az ablakban” nyíljon meg, ahonnan rákattintanak. Ezt én nem így szerettem volna, de nem bírtam rájönni, hogy hol és mit kell állítani. Míg aztán úgy jártam el, hogy készítettem egy mentést az adatbázisról (dump), ebben rákerestem a néhány számomra relevánsnak tűnő szóra (például `window`, `popup`, `size`), és meg is találtam a `mdl_config` táblában a szükséges alapértelmezett értékeket. Először a rövidebbik úton haladtam: átállítottam az adatbázisban az értékeket, oly módon, hogy megnéztem, milyen adatok vannak már meg benne:

```
update mdl_config set
➥value='999' where
➥name='resource_popupwidth';
update mdl_config set
➥value='999' where
➥name='resource_popupheight';
update mdl_config set
➥value='checked' where
➥name='resource_popup';
update mdl_config set value=''
➥where name='resource_
➥popupresizable';
update mdl_config set
value='checked' where name=
➥'resource_popupscrollbars';
update mdl_config set value=''
➥where name='resource_
➥popupdirectories';
...
```

Később tudtam meg, hogy az *Adminisztráció/Beállítások/Modulok*



■ *7. ábra* Az adminisztrátor áttekintő oldala. Érdemes tanulmányozni.



■ *8. ábra* Felhasználók adminisztrálása, például tanárok és szerzők hozzárendelése

alatt lehet minden ilyesmit állítani, szép webes felületről.
A diákok számítógéptermében nálunk *SMB* alapon nyugszik a bejelentkezés. Van egy ilyen hitelesítést lehetővé tevő *PHP* függvény is, az `smbauth`: ezt egy-egy intranetes alkalmazásban használtam is, ahol a diákoknak webes felületre kell bejelentkezniük, és nem akartam újabb felhasználói neveket és jelszavakat kiosztani.
Bár sokféle lehetőség van a hitelesítésre a *Moodle*-ban, ilyen változatot (*smbauth*) nem találtam beépítve. Az *auth/pam* könyvtárbeli *lib.php* fájlt szerkesztettem át oly módon, hogy az `if (pam_auth...` helyett `if (smbauth...`-ot írtam. Valószínűleg teljesen kulturáltan, módosítás nélkül, pam_auth-al is megoldható lett volna a kérdés, de ahhoz le kellett volna tölteni a megfelelő `pam_auth` *PHP*-modult, beüzemelni, és lenyelni az a lelkiismereti békát, hogy egy újabb potenciális réssel lett gazdagabb a webszerverünk. Ennél számomra megnyugtatóbb megoldás volt ama néhány betű beleírása a megfelelő *.php* fájlba, olyan *PHP* függvény meghívásával, amit máshol már úgyis használtam.

Felhasználók be és ki

A felhasználók tömeges hozzáadásának kérdése előbb-utóbb felmerül. Vannak olyan autentikációs módszerek a *Moodle*-ban (például az *IMAP*), amivel úgy is be lehet jelentkezni, hogy még nincs bent a felhasználónév az adatbázisban (ilyenkor az első bejelentkezéskor bekerül).

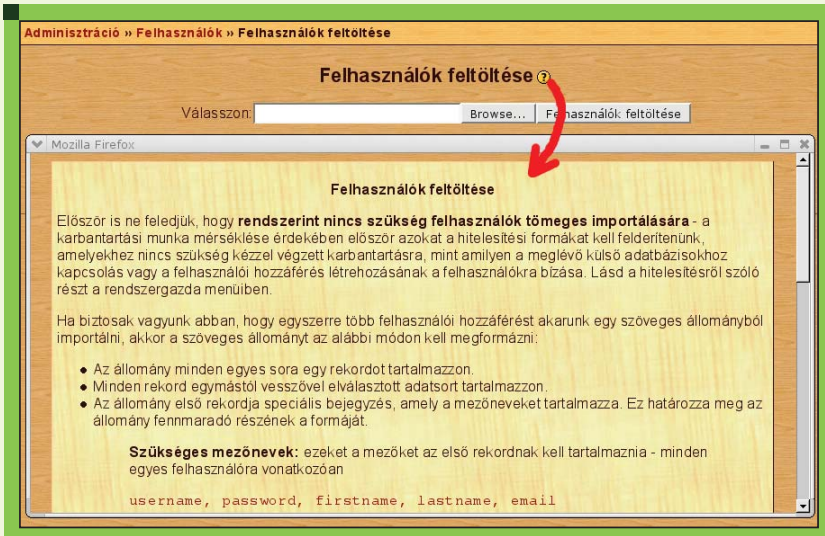
Mivel az *SMB* alapú bejelentkezés nem ilyen, ezért rá kellett szánnom magam, hogy előállítsam azt a fájlt, amit át tudok adni a *Moodle*-nak: „Felhasználók feltöltése” menüpont, *(7. és 8. ábra)*. Vigyázat, e tekintetben eltér az 1.5 és 1.6 verzió, mert a régebbiben a már létező felhasználóneveket egyszerűen nem veszi figyelemben, az újban pedig megváltoztatja (felülírja) a meglevő adatokat a frissen betöltöttekkel.
Ennek a „tömegfájlnak” az előállítására lehet egy *Bash* vagy egy *Perl* szkriptet írni: ki kell mazsolázni a megfelelő szerveren levő */etc/passwd* fáljt. Azonban már úgyis volt egy *PHP* szkriptem, ami hasonló szerepet töltött be, másrészt pedig voltak kivételes „felhasználónevek”, amikkel könnyebb volt így elbánni, inkább ezt használtam fel:

```
$l=file("/etc/passwd");
foreach ($l as $key=>$lsor) {
$sor=explode(":",$lsor);
$nev=explode(" ",$sor[4]);
$login2vnev[$sor[0]]=trim($nev
➥[0]);
$knev=trim($nev[1]."
➥".$nev[2]." ".$nev[3]);
$login2knev[$sor[0]]=
➥($knev=='' ? '-' : $knev);
```

Mint látjuk, kulcsszerepe van egy adott szöveget egy jel mentén szétrobbantó (explode) függvénynek. Ha csak egyszavas a teljes név a */etc/passwd*-ben (például „Vendég”), akkor a keresztnévnek kell adni valami más értéket (pl. ‘-’), hogy meg tudja emészteni a *Moodle*. Majd egy jól célzott `foreach` ciklussal kiírtam a login2(v/k)nev tartalmát, az alábbi eredményt kapva:

```
username, password, lastname,
➥firstname, email, auth
angel,angel,Angyal,Áron,angel@
➥emailcimem.hu,pam
cickany,cickany,
➥Zagyvai,Bernadett,
➥cickany@emailcimem.hu,pam
...
```

Itt a jelszót csak amiatt kell megadni, mert a „tömegfájlt” csak így veszi be a *Moodle* – szerepe nem sok lesz, mert a „pam” úgyis a(z általam smbauth-ra átírt módon) hálózatról ellenőrzi a jelszót. Az emailcímekben a helyi hálózaton használt login nevek szerepelnek.
Még egy apróság: hogy honnan tudtam, hogy milyen mezőket kell/lehet felsorolni? Már említettem, hogy *Moodle* rendszerünk induló *URL*-je mögé illesztett */doc* alatt megjelenő, baloldalt található *Other Docs* részben milyen hasznos oldalak lapulnak. Itt az *Adminisztráció/Felhasználók feltöltése* alatti segítség (ami – adminisztrátorként – az *Adminisztráció/Felhasználók/Felhasználók feltöltése* űrlap címe melletti sárga kérdőjellel #5# is előjön) kifejti a legfontosabb tudnivalókat, és ad egy igényes példát is. Sajnos az nem derül ki belőle, hogy hogyan adhatjuk meg az általunk igényelt autentikációs módot. A *Moodle* gyökérkönyvtára alatti lib/db könyvtárban találjuk az adatbázis-sémákat.



■ *9. ábra* Felhasználók feltöltése – példa egy „helyi súgóra”

Számomra a *postgres7.sql* volt hasznos (a kedves Olvasó számára nagy valószínűséggel a *mysql.php*). Itt először még nem tudhatja a jámbor szemlélő, hogy milyen táblát kell keresnie, de rákeresve a `username` szóra, egyből ráakadunk a *prefix*_user táblára (valódi adatbázisunkban a *prefix* szórészlet helyett más áll, pl. *mdl*). Itt aztán a teljesség igényével lehetett bogarászni, és kezembe is akadt az „auth” mezőnév, amire szükségem volt. (Azt, hogy ennek a mezőnévnek mit adjak értékül, azt úgy tudtam meg, hogy egy felhasználónál az általam választott `pam` autentikációs módot adtam meg a regisztrációkor, majd rákerestem az élő adatbázisban, hogy mi áll az `auth` mezőben. Meglepve láttam, hogy „pam” az értéke. Innen tudtam, hogy nekem is ez kell. Egyébként teljesen logikusak az elnevezések, lehet tudni, hogy *IMAP* esetén `imap` stb.
Ha fel tudja fogni minden érintett diák, hogy a bejelentkezés után csak rá kell kattintania az általa igényelt kurzus nevére, majd „kurzustagként” kell magát elfogadtatnia, akkor semmi teendőnk nincs a beiratkozás tekintetében. (Ha megadtunk beiratkozási kódot, akkor ezt persze célszerű az érintettekkel tudatni.)
Ha a diákok egy része annyira nagy tudású, hogy ez az igenlő kattintás nem várható el tőle (vagy egyszerűen csak kedveskedni akar a tanár nekik azzal, hogy ne kerüljenek ilyen döntéshelyzetbe, hogy meg kelljen válaszolniuk egy *„Kurzustag vagy?!”* kérdést), akkor az adminisztrációs blokkban a *Diákok* menüponttal (ahol is a lap címe: *Diákok beíratása*) kézileg is be lehet jelölni a szükséges diákokat (vagy eltörölni a szükségteleneket).
Ez a kijelölési tudomány aztán hasznos lesz év végén, mert a kiíratás is hasonlóképp történhet. Ha már nagyon sok a diák, akkor sok időt spórolhatunk egy kis hackeléssel.
Az a gond, hogy a fent említett *Diákok beíratása* részben (legalábbis az 1.5.3+ verzióban) keresztnév szerint rendeződnek a diákok.
Ezt hiába próbáljuk úgy orvosolni, hogy a megfelelő helyen *Vezetéknév, keresztnév* sorrendet adunk meg általunk áhított névformátumnak, attól még a rendezés a keresztnév szerint fog történni. Kézbe kell vennünk a *course/student.php* fájlt, aminek a 110. sora környékén ezt látjuk:

```
get_course_students($course
➥->id, "u.firstname ASC,
➥u.lastname ASC"
```

Ehelyett ezt a részt így célszerű átírnunk:

```
get_course_students($course
➥->id, "u.lastname ASC,
➥u.firstname ASC"
```

S hasonlóképp a jobb oldali szövegblokkot is ugyanígy írjuk át a 130. sor környékén.
Ha esetleg más szempont szerint szeretnénk rendezni a nebulókat (például osztály vagy `timemodified` szerint), akkor megint csak az előbb említett *prefix_user* tábla siet a segítségünkre, hogy megtaláljuk, milyen mezőnevek alapján történő rendezéssel úszhatnánk meg az egyesével klikkelgetés tantaloszi kínjait.
A biztonság kedvéért rákerestem, melyik .php fájlban van a `get_course_students` deklarálása, és meg is találtam a *lib/datalib.php*-ben. Erről a témáról bővebben itt olvashatnak a *www.moodle.org* fórumain belül: *General problems –> Easier way to unenroll students from a course.*

Frissen, ropogósan

Időnként frissítést is el kell követnünk a rendszeren. Egyszer magam is tettem ilyet, de annyira jellegtelen volt a műveletsor, hogy még csak emlékfoszlányt sem hagyott bennem (szemben néhány emlékezetes operációs rendszer frissítéssel). Nyilván előtte célszerű elmenteni, ami menthető – itt utalnék *Horváth Ernő* 2004. novemberi *Moodle*-cikkére, amelyben egy remek archiváló szkriptet mutatott be (*Linuxvilág #46/62. o.*).
A ➲ *http://moodle.org/doc/?frame=upgrade.html* alapján a frissítés könnyedén megvalósítható. Ezt követően még a ➲ *http://download.moodle.org/download.php/lang/hu.zip* nyelvi modult is leszedtem, és betettem a *lang/* könyvtár alá. Minden ment remekül. A *Moodle* rendszerében a kurzusok elmenthetők és újra felhasználhatóak. Nagy előrelépés lenne informatikaoktatásunkban, ha lenne néhány ember és megfelelő fórum arra, hogy a jól bevált *Moodle*-tananyagainkat egymás rendelkezésére bocsássuk. Jó volna egy ilyen tankönyv is, ami a *Moodle*-vel összehangolva vezetné diákjainkat az informatika rögös, de szép birodalmának útjain.

Szabó Zoltán
Három gyermekével és feleségével Pannonhalmán él. Tíz éve kísérletezik a Linuxszal. Matematikát és informatikát tanít, diákotthonban keseríti a rábízottak életét. Szívügye a PHP és a PostgreSQL.
(szz@freemail.hu)

A LaTeX néhány alkalmazási lehetősége esszéírásban

Ebben a részben előbb az emacs és a LaTeX alapjaival foglalkozom, majd a dokumentum szerkezete, tagolásával következik. Ezután részletesebben ismertetem, hogy miként lehet több nyelvet használni egy dokumentumban. Végül a bibliográfia kezelését mutatom be. Eközben kitérek az emacs vagy az auctex kapcsolódó részeire is. Célom néhány jellemző lehetőség ismertetése, rövid kódrészletekkel, példákkal.

A cikkben feltételezem az *emacs*, az *auctex* és a *teTeX*, az egyik elterjedt linuxos *TeX* rendszer meglétét, és azt, hogy ezek jól vannak beállítva. Ezt általában a *Linux* disztribúciók csomagkezelője elvégzi. A külön telepítendő csomagokat megemlítem, a csomag nevek *debian*-ra vonatkoznak.

Emacs

A *LaTeX* használatához kell egy szövegszerkesztő. Bármelyik, egyszerű szövegfájlt előállítani képes program megfelel. Természetesen a kényelem, szolgáltatások tekintetében jelentős különbségek vannak köztük.
A sorozat célja többek közt az egyik kiemelkedő szövegszerkesztő, az *emacs* képességeinek bemutatása. Léteznek elterjedt rendszerek, melyek célja a *LaTeX*-helés folyamatának közelítése a *WYSIWYG* világhoz, ezekkel ebben a sorozatban nem foglalkozom.
A továbbiakban az *emacs*-et és a hozzá tartozó *auctex* csomagot fogom használni szövegszerkesztőként, fejlesztői környezetként a *LaTeX*-hez. Az *emacs* és az *auctex* napjaink *Linux* disztribúcióinak standard része, és *Windowsra* is telepíthető. Különösebb ismeretek, képességek és erőfeszítés nélkül, bárminemű leírás vagy súgó elolvasása előtt elkezdhetünk az *emacs*-ben gépelni.

Ekkor úgy használhatjuk, mint bármely más, karakteralapú szövegszerkesztőt, ilyen például a *midnight*, vagy a *norton commander* szerkesztője. A *LaTeX*-hez kapcsolódó menüket és a szöveg kiszínezését így is élvezhetjük. Az *emacs*-et terminálból, vagy az ablakkezelőnk start menüjéből indíthatjuk el.
A gépelés során a programmal a párbeszédet a felső menüsoron, a *minibufferen* és az eszközkészleten (toolbar) keresztül folytatjuk.

A megszokott szövegszerkesztői funkciókat többek közt a felső menüsor `File`, `Edit` és `Buffers` pontjaiban érhetjük el, az utolsó a megnyitott fájlok közti váltásra szolgál. Az *emacs*-el való ismerkedés kezdetekor talán ez a leghatékonyabb.
A *minibuffer* speciális fogalom, az *emacs* ablak legalsó sorát jelenti. Billentyűzetről alapvetően itt adhatjuk ki a parancsokat. A `M-x` kombinációval érhetjük el, és a `C-g`-vel léphetünk ki belőle.

1. ábra Az emacs látható, görög beviteli módba váltva, help-el együtt

A *Tab* használható parancs és fájlnév kiegészítésre, vagy a lehetséges opciók felsorolására. A `C-` kezdetű parancsokat is nyomon követhetjük a *minibufferben*.

Emacs konvenciók

Az *emacs* egérrel is kezelhető, azonban a billentyűzet sok esetben kényelmesebb, hatékonyabb.
A legfontosabb billentyűkombinációk megismerését segíti az *emacs*-el együtt terjesztett *refcard.ps* fájl.
Ebben az emlékeztetőben ugyanazokat a konvenciókat használják, mint a részletes leírásokban. Itt az `M` a *meta* billentyűt jelöli, amely az elterjedt billentyűzeteken általában az *Alt*, de ha ez nem működik, akkor az *Esc* szinte mindenhol használható `meta`-ként. A `C` a *Ctrl* jelölése. A kötőjellel összekötött billentyűket egyszerre kell lenyomni.

Beállítások

Az *emacs* fentebb említett képességei közül kiemelek néhányat, amik megkönnyítik a *LaTeX* forrás fájlban az eligazodást és összefogják a *LaTeX* használatának lépéseit. Néhány hasznos beállítás az *options* menüből:

- `syntax highligthing`, ez a szöveg kulcsszavainak színezését kapcsolja be, így a különböző `latex` parancsok és ezek opciói színesek lesznek
- `word wrap in text modes`, a hosszú sorok automatikus törése
- `paren match highligthing`, a zárójelek párosítását és az itt elkövetett hibákat jelzi
- `active region highligthing`, a kijelölt terület színezése
- `save options`, itt menthetjük el a beállításainkat, ez létrehozza a *~/.emacs* fájlt, ami az *emacs* indulásakor végrehajtandó parancsokat tartalmazza

A *command* menüben, amit az *auctex* jelenít meg, a `texing options`-ön belül a `sourcevspecials` pontot érdemes bekapcsolni, ez kétirányú kapcsolatot teremt a *dvi* és a forrásfájl között. Azt a bekezdést, amelyikben a kurzor áll, bekeretezéssel kiemeli az *xdvi*, valamint az *xdvi* ablakban kiadott `C-bal egér gomb` paranccsal az *emacs*-ben, a mutatott helyre ugorhatunk.
Az *emacs* a fájlok típusát többek közt a kiterjesztés alapján ismeri fel, és ez alapján kapcsol az adott fájl kezelését jelentősen megkönnyítő módba. Ekkor, ha engedélyeztük a színezést, a fájl szintaxisának megfelelően kiszínezi a kulcsszavakat, az utasítások paramétereit, és a felső menüsorban új pontok jelennek meg, amik esetünkben a *LaTeX*-re jellemző parancsokat tartalmazzák. Ezek a leglátványosabb változások, de van még néhány. A *TeX* fájlok kezelésére van az *emacs*-ben egy mód, de a külön csomagként letölthető *auctex* jelentős fejlesztés ehhez képest.

MULE

A természetes nyelvek kezelését az *emacs*-en belül nagyrészt a *mule* (multilingual environment) csomag végzi. Itt többek közt beállíthatjuk a szöveg kódolását, a beviteli módot és a nyelvi környezetet, és ezekről információt is kérhetünk. A `toggle input method` menüponttal engedélyezhetjük a különleges beviteli módok használatát, amit a `select input method` ponttal választhatunk ki.
A *Tab* itt is használható a lehetséges választások felsorolására. A *postfix mód* azt jelenti, hogy a leütött betűk után egy módosító billentyűvel érhetjük el a kívánt betűt. Ezalatt az *emacs* segítségül a *minibufferben* kiírja a lehetséges folytatásokat. Sok nyelvnél az *emacs* átdefiniálja a billentyűzetkiosztást.
A dokumentum mentésekor használt kódolást, a `set coding system` pont `for saving this buffer` alpontjával állíthatjuk be. Csak magyar nyelvű szöveghez jó a *latin2* kódolás, európai nyelvekhez a megfelelő *latin* kódolás használható. Ha több európai, vagy nem európai nyelvet használunk, akkor az `utf-8-unix` kódolást válasszuk, ezt a *LaTeX* az `ucs` és `inputenc` csomagjai értelmezik.

1. táblázat

Karakter	Jelentése	Beillesztése a dokumentumba
#	kötelező paraméterek száma	\#
$	matematikai mód kezdete és vége,	\$
%	megjegyzés kezdete, a sor végéig tart	\%
&	táblázatokban oszlop elválasztó	\&
~	nem törhető szóköz	\verb+~+
_	alsó index jelölése	_
^	felső index jelölése	\verb+^+-
\	parancs kezdete	$\backslash$
{ és }	kötelező paraméterek jelölése	\{ és \}


Vázlat

Vecsei Balázs
vecseib@math.bme.hu

2006. január 2.

Tartalomjegyzék

1. Bevezetés 1
1.1. Előszó 1

1. Bevezetés

1.1. Előszó

2. ábra A váz LaTeX-hel lefordítva

1. Lista Hello world LaTeX-ben

```
\documentclass{article}
\begin{document}
hello world
\end{document}
```

2. Lista cím, szerző és dátum megadása és tagolás

```
1: \documentclass
   ➥[a4paper,11pt]{article}
2: \usepackage{ucs}
3: \usepackage[utf8x]
   ➥{inputenc}
4: \usepackage[magyar]{babel}
5: \author{Vecsei Balázs\\
   ➥vecseib@math.bme.hu}
6: \title{Vázlat}
7: \begin{document}
8: \maketitle
9: \tableofcontents
10: \section{Bevezetés}
11: \subsection{Előszó}
12: \end{document}
```

LaTeX alapok

Ebben a fejezetben a ***LaTeX*** néhány alapvető jellemzőjét mutatom be, melyek a következő kódrészletek megértését segítik. A ***LaTeX*** dokumentumokban együtt van jelen a szöveges tartalom, és a `latex`-nek szóló parancsok, ezért fontos, hogy a kettőt könnyen megkülönböztethessük.
A parancsoknak formailag két fajtája van, mindkettő \ jellel kezdődik.
Az egyik lehetséges folytatás a parancs neve és az opcionális meg a kötelező paraméterek. Az opcionálisak szögletes, míg a kötelezők kapcsos zárójelben vannak, például `\usepackage{magyar}[babel]`. Másik formája a parancsoknak a \-t követő pontosan egy nem betű karakter, például \%. Parancsok argumentumán belül állhatnak további parancsok, de nem mindegyik. A pontos szabályokat nem részletezem.

Különleges jelentésű karakterek LaTeX-ben

Van néhány karakter melynek speciális jelentése van. Ha ilyen karaktereket akarunk megjeleníteni, parancsot kell írni a forrás fájlba. (*1. táblázat*)

Szóközök, újsorok

A szóközök, újsorok kezelése ***LaTeX***-ben alapvetően eltér a ***WYSIWYG*** szövegszerkesztőktől. A forrás fájlban levő, tetszőlegesen sok egymást követő szóközből egy lesz a fordítás után. Egy vagy több üres sorból meg új bekezdés lesz. Sortörés vagy lapdobás kikényszerítése ellentétes a ***LaTeX*** elveivel, de néha, általában a dokumentum végső formázásakor, szükséges. Sortörést a \\ vagy a `\newline` parancsokkal érhetünk el. Új lapot a `\newpage` paranccsal kezdhetünk.

A dokumentum tagolása

Néhány bekezdésnél hosszabb dokumentumok tartalmi tagolására a következő parancsok szolgálnak: a `book` osztályban használható `\chapter{}` és a `\part{}`, továbbá az `article` osztályban is megtalálható `\section{}`, `\subsubsection{}`, `\paragraph{}`, `\subparagraph{}`. A formai jellemzőket ezek, és a használt nyelv alapján a ***LaTeX*** határozza meg.

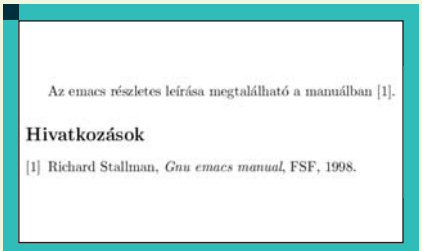
Hello world, LaTeX-ben

Ezt már le lehet fordítani és megjeleníteni az ***emacs command*** menüjéből a következőkkel:

- `latex`
- `quick view`, ez megjeleníti a keletkezett ***dvi*** fájlt az ***xdvi*** nézegetővel anélkül, hogy megmutatná a parancssort, ha a `source special` be van kapcsolva, akkor az első alkalommal megkérdezi, hogy indítson-e ***emacs*** szervert, erre válaszoljunk igent
- `view`, ez is megjelenítő, csak itt szerkeszthetjük a parancssort

3. ábra A magyar és az angol fejezetcímek közti különbségek, felül a magyar

4. ábra A dátum magyarul, angolul, görögül és oroszul



5. ábra A két bibliográfiával kapcsolatos kód eredménye

3. Lista nyelvi környezetek

```
\documentclass[a4paper,11pt]{
➥article}
\usepackage{ucs}
\usepackage[utf8x]{inputenc}
\usepackage[magyar,english,
➥greek,russian]{babel}
\begin{document}
\selectlanguage{magyar}
\section{Bevezetés}
magyarul: \today\\
\selectlanguage{english}
\section{Bevezetés}
\foreignlanguage{magyar}
➥{angolul:} \today\\
\selectlanguage{greek}
\foreignlanguage{magyar}
➥{görögül:} \today\\
\selectlanguage{russian}
\foreignlanguage{magyar}
➥{oroszul:} \today\\
\end{document}
```

4. Lista egy elem a bib_demo.bib fájlból

```
@Book{emacs_man,
  author = {Richard Stallman},
  ALTeditor = {},
  title = {GNU Emacs Manual},
  publisher =   {FSF},
  year =        {1998},
  OPTkey =      {},
  OPTvolume =   {},
  OPTnumber =   {},
  OPTseries =   {},
  OPTaddress =  {},
  OPTedition =  {},
  OPTmonth =    {aug},
  note = {http://www.gnu.org/
  ➥manual/emacs-20.3/
  ➥emacs.html},
  OPTannote =   {}
}
```

5. Lista Bibliográfia készítése

```
\documentclass[a4paper,11pt]
➥{article}
\usepackage{ucs}
\usepackage[utf8x]{inputenc}
\usepackage[magyar]{babel}
\begin{document}
Az emacs részletes leírása
➥megtalálható a manuálban
\cite{emacs_man}.
\bibliographystyle{amsplain}
\bibliography{bib_demo}
\end{document}
```

Egy LaTeX dokumentum váza

Elterjedt módszer, hogy a *LaTeX*-hel írt munkáinkat egy tartalmilag üres, az általunk használt alapvető *LaTeX* utasításokat tartalmazó fájl módosításával kezdjük. A következőkben magyarázatokkal együtt megadok egy ilyen üres fájlt, ami elég a *LaTeX*-hel való munka tényleges elkezdéséhez. Ezt *emacs*-be begépelve a preambulum értelemszerű módosítását követően az első `\section{}` után elkezdhetjük írni az esszét. A sorszámozás nem része a forrásfájlnak.

1. Az első sorban megadjuk, hogy milyen dokumentumot készítünk és ennek opcióit. Az egyik legáltalánosabban használt típus a cikk. Néhány további dokumentumosztály: könyv, önéletrajz, levél és fóliák, kinyomtatáshoz és projektorhoz.

2. Az *ucs* csomag betöltése, leírása a `latex-ucs-doc` csomagban található `languages.ps` és `ucs.ps` fájlokban található

3. Az *UTF-8* kódolás használatát teszi lehetővé, az `inputenc` csomaggal.

4. A `babel` csomag betöltése, ez végzi a *LaTeX* magyarítását, illetve egyéb nyelvekhez illesztését.

5. Szerző megadása

6. Cím megadása

7. Dokumentum kezdete, a preambulum vége

8. Címoldal készítése, a pontos kinézet a választott dokumentumosztálytól függ, például `article` osztályban nincs külön címlap, míg `book` osztályban van.

9. Tartalomjegyzék készítése

10. Fejezet nyitása

11. Alfejezet nyitása

12. A dokumentum befejezése

Több nyelv használata

Az *emacs*-ben beírt többnyelvű, *UTF-8* kódolású, szöveget a *LaTeX* a `latex-ucs` csomag segítségével tudja feldolgozni. A `babel` csomag segítségével a nyelvnek megfelelő lesz többek közt a fejezetcímek kezelése, például angolban a sorszám után nincs pont, míg magyarban van és nagyobb a helykihagyás, mint magyarban. A `\usepackage{babel}` parancs opcionális paraméterében adjuk meg a használt nyelveket. Ezek közül a `\selectlanguage{}` utasítással válthatunk. A `\foreignlanguage {magyar}{görögül:}` sor a görög vagy orosz nyelvű szövegben teszi lehetővé magyar szavak beillesztését. A `\today` parancs a dátumot írja be a szövegbe.

Bibliográfia

Hagyományos módon a fájl végére írjuk a bibliográfiát, amit kézzel szerkesztünk. Ez, azon túl, ellentétes a *LaTeX* elveivel, amely szerint nekünk csak a tartalmi részeket kell megadni, kényelmetlen is. Használjuk inkább a ***bibtex*** programot, mely a *LaTeX* rendszerben a bibliográfiát kezeli. Ehhez egy külön, ***.bib*** kiterjesztésű fájlt használ, aminek megnyitásakor az ***emacs bibtex*** módba kerül. Itt az `entry-types` menüből kiválaszthatjuk a kívánt bibliográfiai elem típusát ezután az *emacs* megadja a kötelezően és az alternatívan vagy opcionálisan kitölthető mezőket. Az `ALT`-al jelölt elemek közül legalább egyet meg kell adni, míg az `OPT`-al kezdődőek opcionálisak, ha kitöltjük ezeket a rekordokat a jelzőket ki kell törölni a `.bib` fájlból.

Itt egy könyvről van szó, melyre a szövegben `emacs_man` néven hivatkozhatunk, a következő módon: `\cite[15]{emacs_man}` az opcionális paraméterben az oldalszámot adhatjuk meg. A dokumentumba a `\bibliography{}` paranccsal, melynek paramétere a `.bib` fájl neve, kiterjesztés nélkül, illeszthetjük be a bibliográfiát, aminek a stílusát a `\bibliographystyle{}` utasítás paramétere határozza meg. A legtöbb folyóirat közzéteszi az általa megkövetelt stílust leíró fájlt.

A bibtex használata

A cikkre visszaváltva a ***command*** menüben ki kell adni a következő parancsokat: `latex`, `bibtex`, `latex`, `latex`, ebben a sorrendben. A *LaTeX* rendszerre jellemző a programok többszöri futtatása, amire a *LaTeX* által generált részeket tartalmazó dokumentumok esetén általában szükség van.

Bibliográfia készítésekor a `latex` első futtatása alatt kigyűjti a hivatkozásokat a forrásfájlból. Ezt követően a `bibtex` elkészíti a bibliográfiából a dokumentumosztálynak, nyelvnek és az egyéb opcióknak megfelelő formátumú bibliográfiát, a `latex` második futtatásakor már megvan a kész bibliográfia és az is, hogy a hivatkozásokat hova kell tenni a dokumentumban. A harmadik futtatásra azért van szükség, mert a hivatkozások beszúrása után elcsúszhatnak az oldalszámok.

Összefoglalás

Röviden bemutattam az *emacs* szövegszerkesztőt, a *LaTeX* lehetőségeit, működését, az ugyanazon dokumentumban különböző betűkészletű nyelvek használatát és a *LaTeX* bibliográfiakészítő szolgáltatását. A bonyolultabb formázások, és a további lehetőségek elsajátításához a gyakorlás, egymás közötti megbeszélés, melynek egyik fóruma a ***texnh@yahoogroups.com*** magyar nyelvű levelezőlista, valamint a magyar és angol nyelvű nyomtatott és számítógépes irodalom ad lehetőséget.

Vecsei Balázs
(vecseib@math.bme.hu)
A BME TTK-n végzett matematikusként. 1998 óta foglalkozik Linuxszal és LaTeX-hel. Szabadidejében sokat túrázik Magyarországon és külföldön is, kedvence a Börzsöny.

KDE alkalmazások (3. rész)
Tömeges átnevezés II.

A KRename program haladó használata nem csak a füleket jelenti, mint azt a Beállítások ablak sugallja. A haladó mód ennél sokkal mélyebb tudással rendelkezik, amely egyesítheti a konzol tudását a grafikus felület könnyedségével.

Fájlok fül

Az előző rész tapasztalatai alapján az első fülön minden gond nélkül tovább kell jutnunk, hiszen itt adhatjuk meg azon a fájlok listáját, amelyeket át szeretnénk nevezni (*1. ábra*).

Cél fül

A második fül tartalma önmagáért beszél: az átnevezésen túl képesek vagyunk átmásolni vagy áthelyezni az állományainkat, illetve akár linket készíthetünk róluk (*2. ábra*).

Modulok fül

A harmadik fül hordozza a legtöbb tudást, ugyanis minden egyes fájlon a bekapcsolt modulok műveleteket végezhetnek. Mindegyik modulnak megvan a saját specialitása, így használatuk egyszerű (*3. ábra*).

Figyeljünk arra, hogy minden modul egy A ***modul használata*** jelölőnégyzettel kezdődik, amelyek az alapértelmezés szerint üresek, be kell őket jelölni a modul használatához.

Parancs végrehajtása

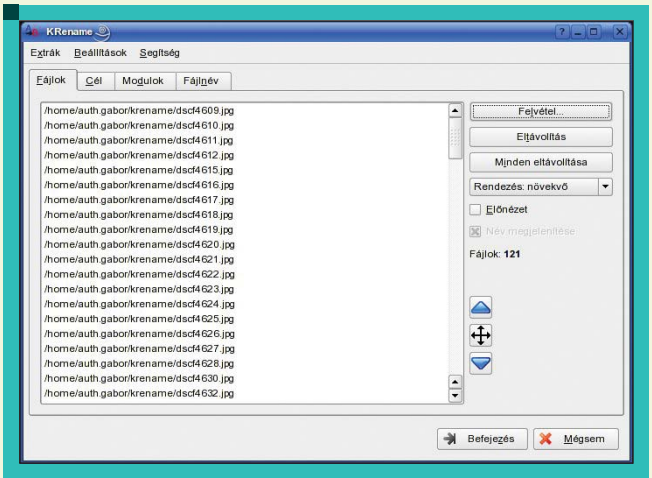
Ez a modul akkor hajtódik végre, ha a fájl átnevezése már megtörtént, s egy megadott parancsot fog végrehajtani.
A parancssor megadása után lehetőségünk van az a ***Felvétel*** gomb megnyomásával a listába elmenteni, illetve a lista elemére kattintva azt újra felhasználni. A felesleges parancsokat az ***Eltávolítás*** gomb segítségével ki tudjuk törölni. A példában minden egyes átnevezett állományt össze fogunk tömöríteni a GZIP program segítségével.

Jogosultságok

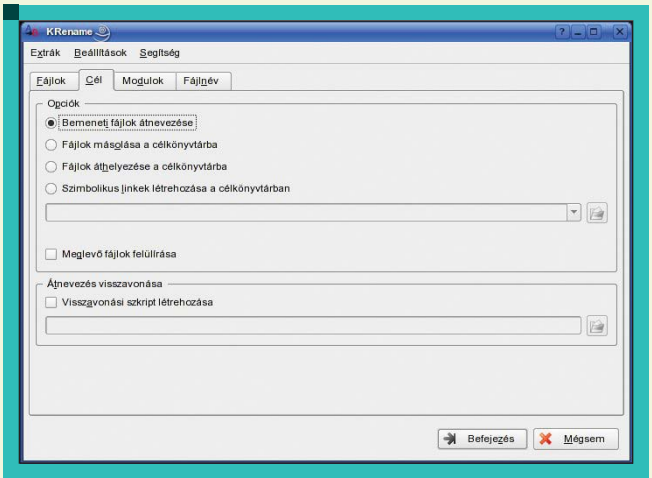
Ez a modul az aktuális állomány jogosultságait változtatja meg, akár a tulajdonost és csoportot is tudunk váltani, ha jogunk van erre (rendszergazdai jogokat igényel).
Mind a ***Hozzáférési jogosultságok***, mind a ***Tulajdonos*** csak akkor fog érvényesülni, ha a hozzájuk tartozó jelölőnégyzetet bejelöljük.
A működése egyszerű és könnyen átlátható, nehezen tudunk hibát elkövetni.

Dátum és idő

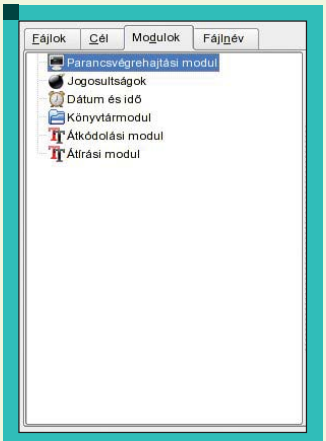
Az átnevezett állomány dátumát és idejét tudjuk változtatni ezzel a modullal, legyen ez a létrehozási, a módosítási vagy a hozzáférési idő. A dátumot a KDE szokásos dátumválasztó paneljében tudjuk beállítani,



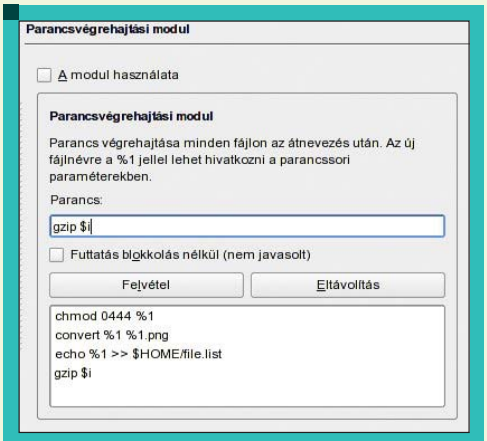
1. ábra Fájlok hozzáadása



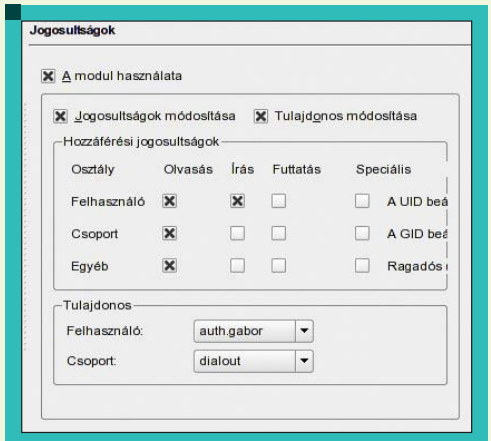
2. ábra A célkönyvtár meghatározása



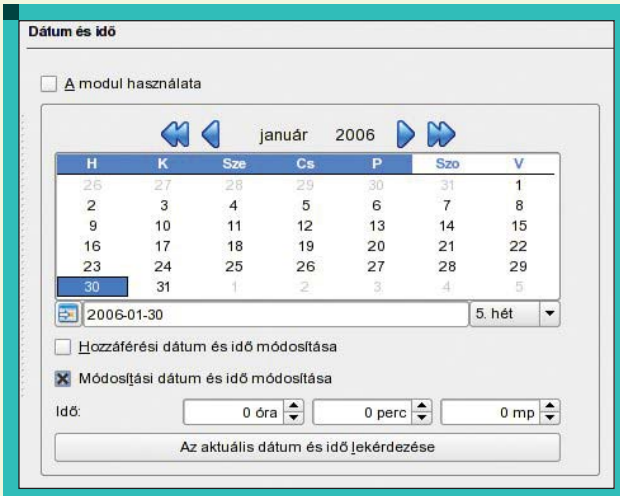
3. ábra Modulok



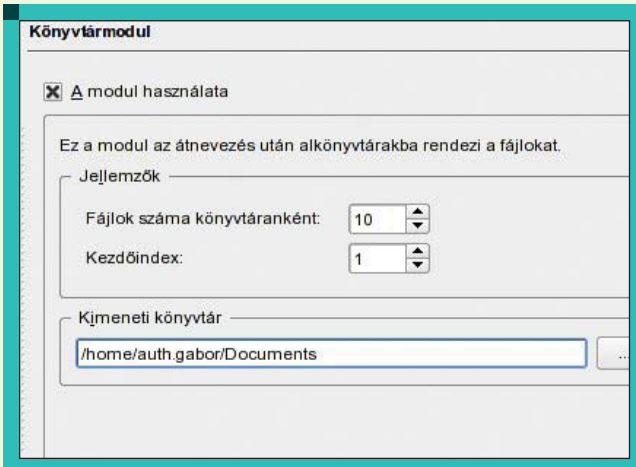
4. ábra Parancs végrehajtása



5. ábra Jogosultságok



6. ábra Dátum és idő



7. ábra Könyvtármodul

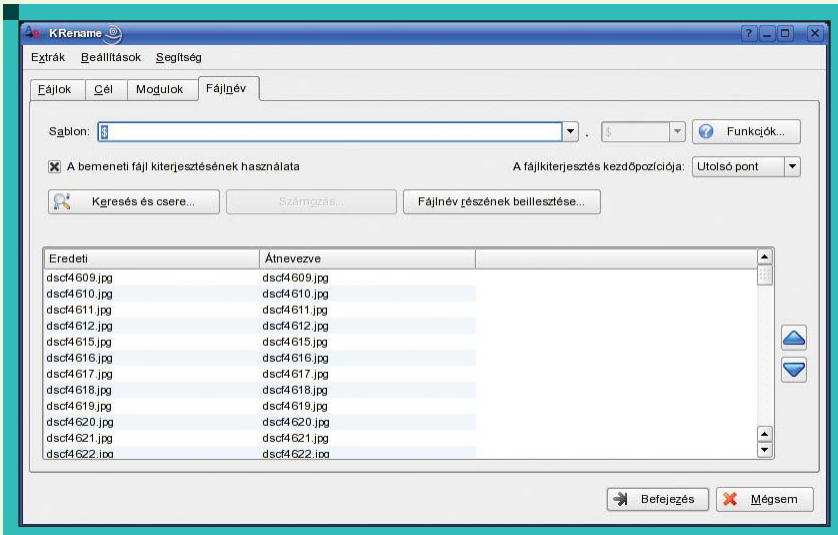
az időt pedig egyszerűen csak be kell írnunk. Ha bejelöljük a módosítási vagy az elérési időt is, akkor a modul azt is beállítja.

Könyvtármodul

Az átnevezett állományokat a *KRename* képes más-más könyvtárakba tenni, jelenleg csak darabszám szerint, így sok fájlt tartalmazó könyvtárat át tudunk szortírozni több könyvtárba. A darabszámot és az alkönyvtárak kezdő számozását kell megadnunk, illetve kiválasztani a célkönyvtárat.

Egyéb modulok

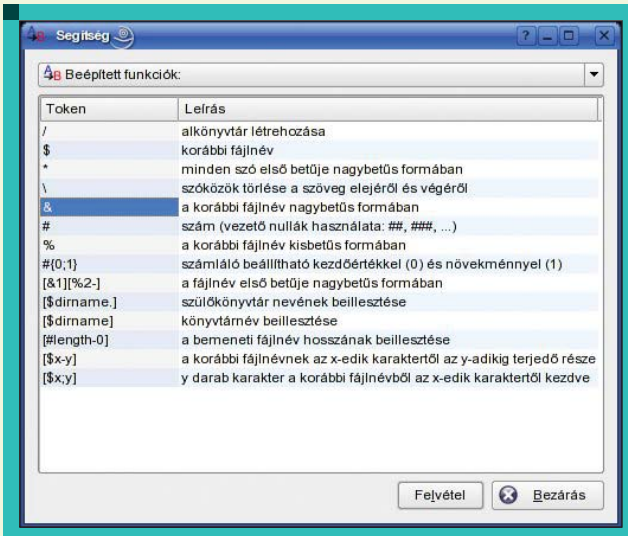
A *KRename* fejlesztése során egyre több modult ismerhetünk meg, ilyen például az *Átkódolási modul* és az *Átírási modul* is. Ezen utóbbiak még kezdetleges állapotban vannak, erre a vaskos *„THIS PLUGIN IS UNTESTED AND MIGHT CAUSE LOSS OF DATA!”* (vagyis: ez a modul nincs tesztelve, használata adatvesztést okozhat!) szöveg is figyelmezteti a gyanútlan felhasználót.



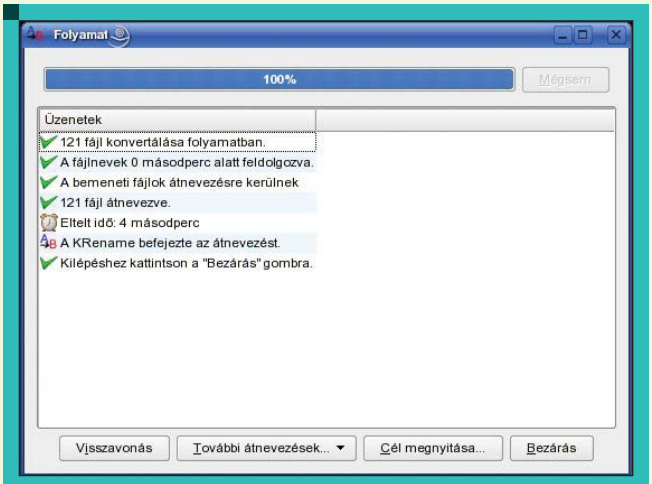
8. ábra A Fájlnév fül

Végső fázis, avagy a Fájlnév fül

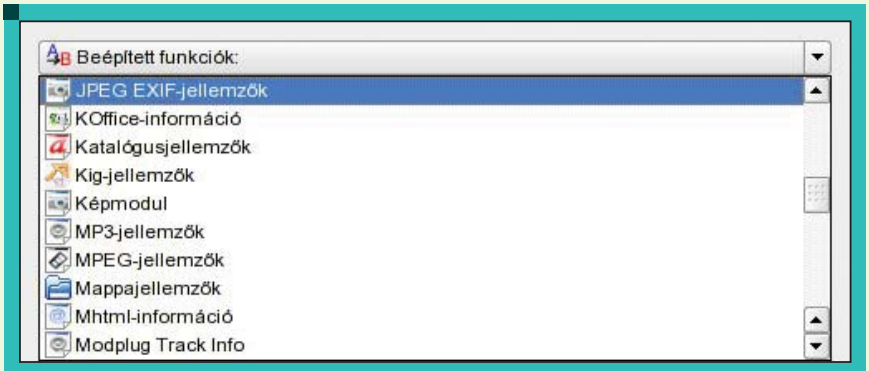
Minden ezen a negyedik fülön dől el, ugyanis itt tudjuk az új fájlneveket létrehozni (*8. ábra*). A $ jelenti az



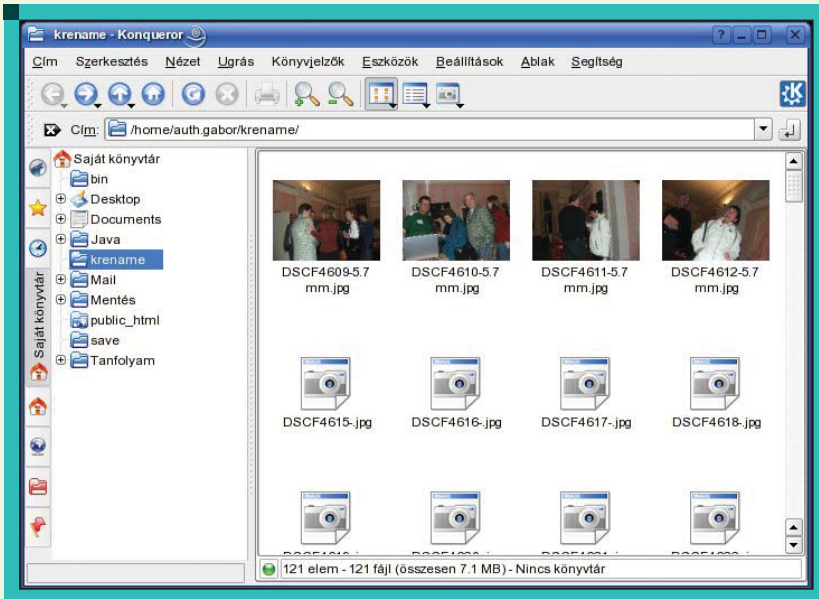
9. ábra Funkciók



11. ábra Folyamatjelző



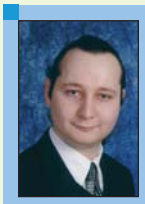
10. ábra JPEG EXIF



12. ábra A végeredmény

állomány nevét illetve kiterjesztését, ezt kitörölve vagy felhasználva új nevet tudunk készíteni. A ***Funkciók*** gombra kattintva megtekinthetjük a program saját névkezelő függvényeit, amelyekkel az állományaink nevét befolyásolni tudjuk (*9. ábra*), illetve a bővített tulajdonságokat, amelyeket egy-egy állománytípusból ki tud nyerni a ***KRename***. Mivel példánkban ***JPEG*** állományokkal dolgozunk, így a ***JPEG EXIF*** adatokat (*10. ábra*) tudjuk felhasználni, a [jpgFocal length] az objektíven beállított gyújtótávolságot tárolja, ezt hozzá tudjuk adni a képeinkhez.

Ha kiéltük az átnevezési kreativitásainkat, akkor a ***Befejezés*** gombra kattintva a program elvégzi a fájlokon beállított műveleteket (*11. ábra*), amelyet a folyamatjelzőn nyomon tudunk követni. A ***Cél megnyitása...*** gombon kattintva a ***Konqueror*** megnyitja azt a mappát, amely munkánk eredményét tartalmazza (*12. ábra*).



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

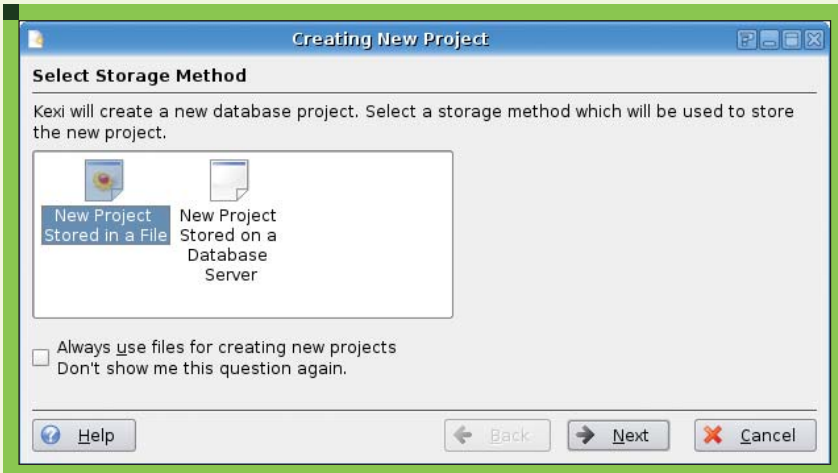
KDE projekt oldala
➲ http://www.kde.org

KOffice – irodai alkalmazás-család KDE alapokon (2. rész)

Kiegészítő eszközök

A KOffice a KDE grafikus asztali környezetbe illeszkedő irodai alkalmazás-gyűjtemény. Bemutatásának első részében a szövegszerkesztésre, táblázat-kezelésre és bemutatókészítésre összpontosítottunk. A második részben a munkát segítő kiegészítő alkalmazásokra koncentrálok: adatbázisok, rajz/grafika-készítés, képmanipulálás, folyamatábra-készítés.

Egy irodai alkalmazás-csomag alapjai (szöveg, táblázatok és bemutatók) mellett számos olyan kiegészítő alkalmazás lehet hasznos, amely a dokumentumok létrehozásában, szerkesztésében, látványosabbá tételében, vagy az adatok vizualizációjában ad segítséget. A *KOffice* alkalmazás-csomagban számos olyan elemet találunk, amely ezekben a feladatokban segítségünkre lehet. Az itt röviden bemutatásra kerülő *Kexi* adatbázisok létrehozásában és kezelésében lehet segítségünkre, a *Kugar* jelentések készítésében, a *Karbon* rajzok és ábrák készítésében, a *Krita* képek szerkesztésében és manipulálásában, míg a *Kivio* folyamatábrák és diagramok készítésében.



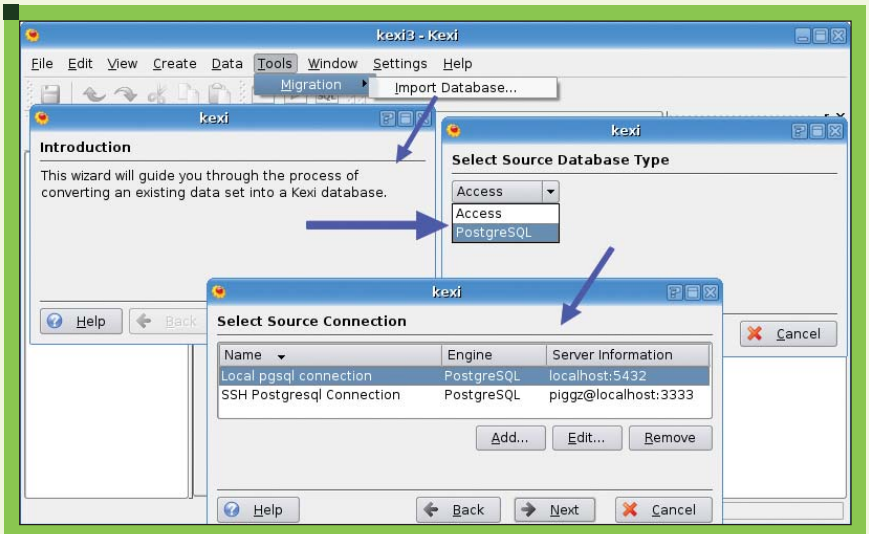
1. ábra Új adatbázis létrehozása kexi adatbázis-fájlban vagy távoli adatbázis-szerveren lehetséges

Adatbázisok – Kexi

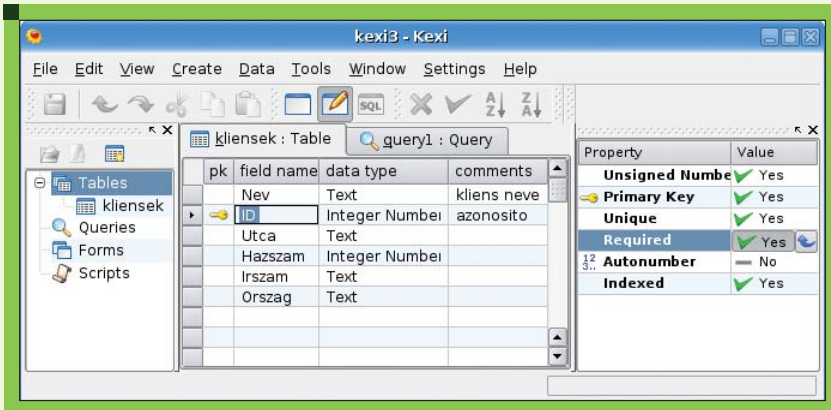
Különféle adatok tárolására és kezelésére az emberek általában táblázatkezelőket használnak, ami egy viszonylag könnyen kezelhető megoldás kis mennyiségű, ritkán bővülőm nem túl bonyolult összefüggéseket tartalmazó adathalmazok esetén. Ellenkező esetben gyorsan felmerülhet az igény olyan adattárolási módszerekre, amelyek megkönnyítik az adatkezelést. Nagyvállalatok esetén ilyenkor nagy tudású adatbázis-szervereket vetnek be, de átlagos irodai alkalmazások esetén ezek több szempontból sem megfelelőek. Azonban az irodai alkalmazások szintjén is találhatunk megoldást, olyan alkalmazásokban, mint pl. a *Microsoft Windows* világában igen elterjedt *Microsoft Access*. A *Kexi* az *Access*, és ehhez hasonló alkalmazások körébe beszálló versenytárs, hasonló funkcionalitást kínálva, *Linux* és *Windows* operációs rendszereken (itt a linuxos változattal foglalkozunk). Kezdjük az elején. Új *Kexi* adatbázis létrehozásakor két választásunk van: olyan adatbázist hozunk létre, amelynek teljes tartalma és összes funkciója egyetlen fájlban tárolódik (*.kexi* kiterjesztésű állományokban), vagy olyan adatbázist hozunk létre amely egy távoli (vagy éppen lokális) *MySQL* vagy *PostgreSQL* adatbázis-szerveren tárolja az adatokat (*1. ábra*).

Ha már létező adatainkkal szeretnénk *Kexi*-ben adatbáziskezelést végezni, *Kexi*-ben a *Tools->Migration->Import Database* opciót választva *PostgreSQL*, *MySQL* adatokat vagy *Access MDB* fájlokat tudunk importálni (*2. ábra*). Az *Access* importáláshoz külön kell telepítenünk az úgynevezett *MDB Driver*-t, amelyet a *Kexi* oldaláról tudunk elérni, *keximdb* néven.

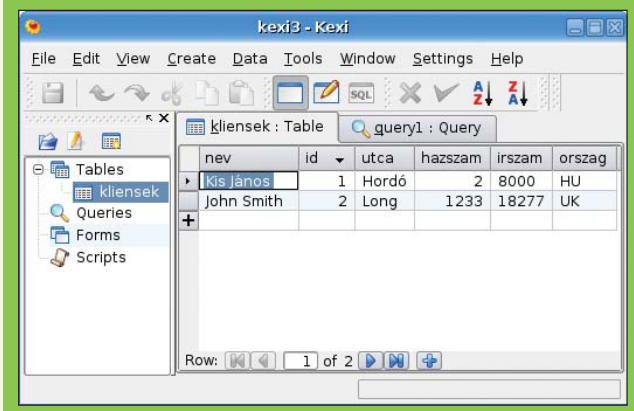
Új adatbázis létrehozását vagy importálás után a alkalmazás ablakának bal



2. ábra Adatok importálása



3/a. ábra Táblázat létrehozása Design nézetben



3/b. ábra A 3/a ábra táblájának Data nézete, adatok feltöltése

oldalán található úgynevezett *Project Navigator* panelen férhetünk hozzá az adatbázis egyes elemeihez, a táblákhoz, a lekérdezésekhez, a létrehozott kérdőívekhez (*form*-ok) és az adatbázisban tárolt szkriptekhez.
A *Kexi* három megjelenítési/szerkesztési módot tesz lehetővé, a *View* menüből vagy az eszköztárról elérhető *Data*, *Design* és *Text* módokat, amelyeket minden elem szerkesztésekor használhatunk. Az alkalmazás többféle többablakos megjelenítést támogat, amelyeket a *Windows* menü *MDI Mode* pontjában állíthatunk be, mindenki kiválaszthatja azt a módot ahogyan a legkönnyebb számára a több ablak egyidejű kezelése.

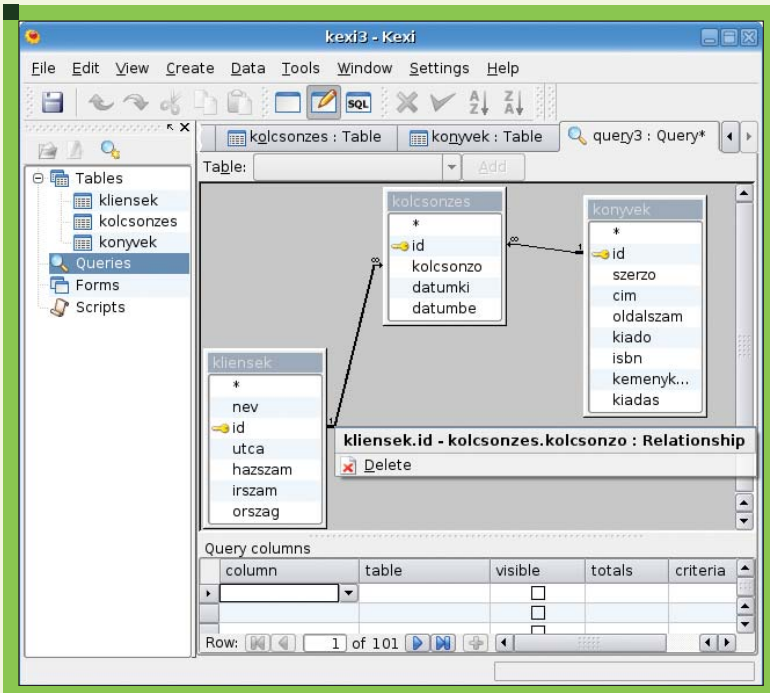
Táblák létrehozásakor a *Design* nézetben állíthatjuk be a táblák szerkezetét, adhatunk nevet neki, stb. (*3/a ábra*).
A *Data* nézetben pedig feltölthetjük adatokkal (*3/b ábra*).
Ha létrehoztunk (vagy importáltunk) táblákat, a *Project Navigator*-ban a *Queries*-t választva lekérdezéseket készíthetünk. *Design* megjelenítési módban ekkor könnyen vizualizálható tábla-összefüggéseket (relációkat) készíthetünk (*4. ábra*), hasonlóan a *Microsoft Access*-ben található megoldáshoz. *Text* (szöveges) módra váltva *SQL* parancsokkal írhatjuk be szabadon a lekérdezéseinket, amelyeket ellenőrizhetünk is és hiba esetén a hibáról leírást is kapunk segítségképpen (*5. ábra*), majd *Data* nézetbe váltva láthatjuk a lekérdezésre adott választ.
Kexi-ben *Form*-okat is készíthetünk. Például a *6. ábrán* a következő *SQL* kérés (amelyet *Query*-ként hozhatunk létre) eredményét jelenítjük meg:

```
SELECT kliensek.id,kliensek.nev,
konyvek.cim FROM kliensek,
kolcsonzes,konyvek where
kolcsonzes.kolcsonzo=kliensek.id
```

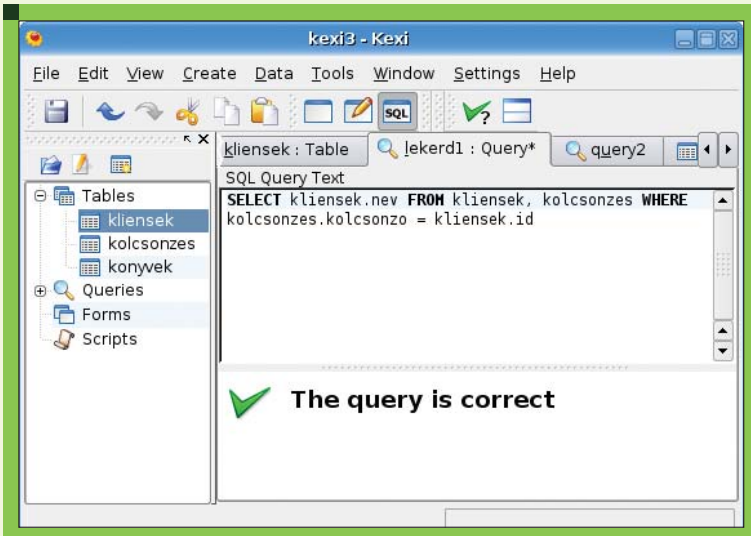
(vagyis az összes kölcsönzés esetén a kölcsönzéshez tartozó kliens és a könyv címe) úgy, hogy a lekérdezése eredményeinek azonosítóit (jelen esetben id, nev, cím) adjuk meg *Data Source*-ként (adatforrás) a form-ra kihelyezett szövegdobozoknak. A *Form*-ot *Design* nézetben kell létrehoznunk és *Data View* nézetre váltva láthatjuk az adatokat (*6. ábra*).
Sajnos a *Kexi*-nek vannak hiányosságai is, pl. a jelenlegi verziókban nincs lehetőség táblák exportálására, *Form*-ok használatakor viszonylag gyakran tapasztalhatjuk, hogy az alkalmazás egyszerűen kilép, ami néha eléggé kellemetlen lehet. Viszont érdemes szemmel tartani a *Kexi* projekt oldalát, mert a következő verzióban számos új funkciót vezetnek majd be.

Jelentések/report-ok készítése – Kugar

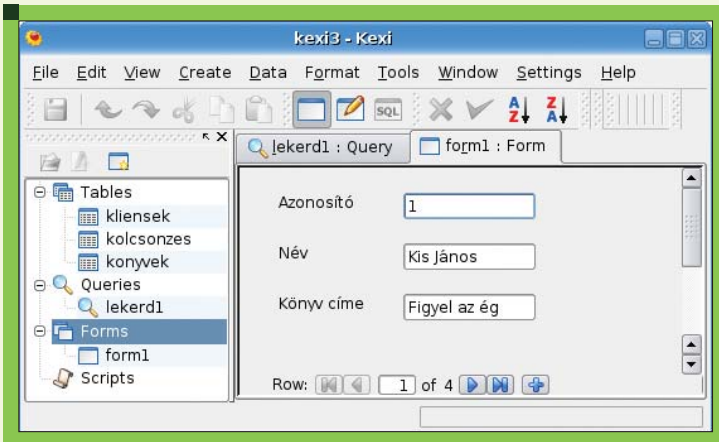
A *Kugar* egy *report*-készítésre használható, nagyon egyszerű, mondhatni „fapados” alkalmazás, viszont funkcióját megfelelően ellátni képes *KOffice* alkalmazás. A *Kugar* két részből áll, egy elrendezés-tervező, amivel



4. ábra Lekérdezések: Design megjelenítési mód, táblák relációinak vizualizálása



5. ábra Lekérdezések: SQL kérések beírása és ellenőrzése



6. ábra Lekérdezés eredménye megjelenítve egy Form-on

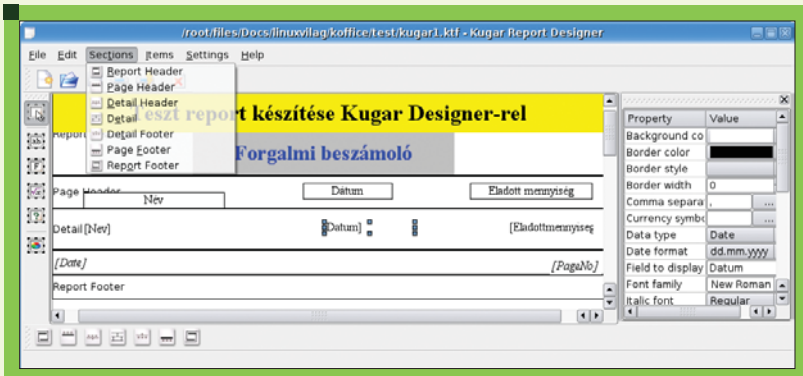
könnyen és gyorsan leírhatjuk, hogy a megjelenítendő adataink hol, hogyan, milyen formátumban jelenjenek meg a ***report***-ban, illetve a második rész maga a megjelenítő, amely a megtervezett elrendezést és egy adatfájlt beolvasva megjeleníti a ***report***-ot.

A ***Kugar Designer***-ben a jelentés elrendezését tervezhetjük meg könnyedén úgy, hogy egyrészt meghatározhatjuk az oldal felosztását, fő szekcióit, másrészt a szekciókon belül elemeket helyezhetünk el, amelyek szöveget, dátumot, oldalszámot, illetve magukat az adatokat fogják tartalmazni. Fontos tudnunk, hogy magukat az adatokat nem kell beillesztenünk, hanem csak a később megjelenő adatok egy példányának formátumát és helyét kell meghatároznunk, a többi úgymond „megy magától". Mintha ***XSLT*** transzformációkat írnánk az adatainkhoz, de itt nem kell parancsokat és kulcsszavakat ismernünk ill. használnunk, mert mindent vizuális eszközökkel tervezhetünk meg.

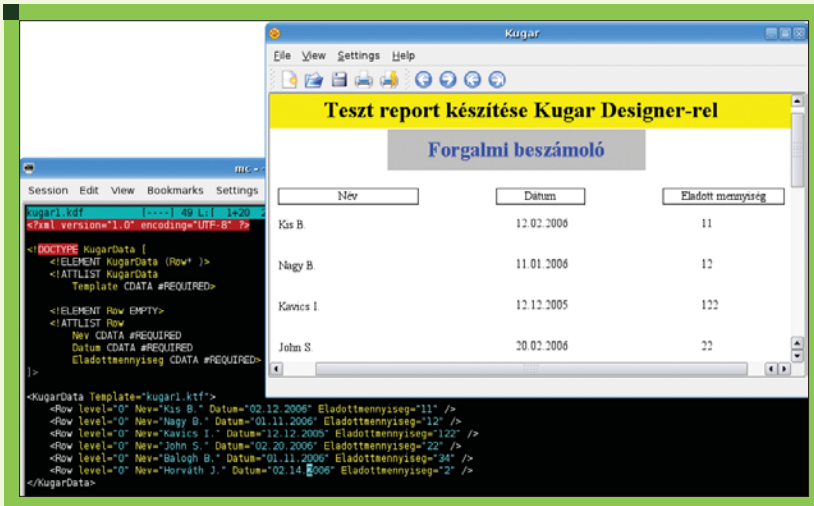
A *7. ábra* egy egyszerű példát mutat ***Kugar Designer***-ben készülő jelentésre. Az oldal fejlécre (***Report Header***, ami csak az első oldalon jelenik meg és a ***Page Header*** ami minden oldalon meg fog jelenni), a tartalmi rész (***Detail*** szekció, ez fogja az adatokat tartalmazni), és a lábjegyzetek (ide oldalszámot és dátumot illesztettünk). Az egyes szekciókba az ***Items*** menü elemei közül választható elemeket illeszthetünk, ezek a ***Designer*** ablakában a baloldalon látható függőleges eszköztáron is megtalálhatók. Az elemeket lehelyezve, ezekre kattintva a jobboldali paraméter-ablakban állíthatjuk be megjelenési és tartalmi tulajdonságait. Ezek az elemek lehetnek ***label*** (egyszerű szöveg, ami nem változik), ***field*** (olyan elemek, amelyek a megjelenítendő adatbejegyzéseket írják le, például a *7. ábrán* a ***Név***, ***Dátum*** és ***Eladott mennyiség*** oszlop-fejlécek), ***calculated field*** (aminek az értéke más mezők értékéből lesz számítva, lehet összeg, átlag, szórás, stb.), illetve ***special field*** (ami lehet dátum vagy oldalszám). A ***Designer***-rel létrehozott elrendezés-fájl kiterjesztése *.ktf* lesz (azaz ***Kugar Template*** File).

A megjelenítés formátumának megtervezése után az adatokra is szükségünk lesz. A ***Kugar XML*** adatfájlokat tud beolvasni és felhasználni, amit

vagy kézzel (sok adatnál kivitelezhetetlen) kell megszerkesztenünk, vagy más programmal kell előállítanunk, vagy írnunk egy ***script***-et vagy ***plugin***-t ami képes ezt egy már meglévő adathalmazból előállítani. Az adatfájl neve egyezzen meg a ***Designer***-ből lementett elrendezés-fájl nevével, kiterjesztése pedig ***.kdf*** legyen (azaz ***Kugar Data File***). Ezután nyissuk meg ***Kugar***-ban a ***.kdf*** fájlt. A *8. ábra* példája a *7. ábra* elrendezését felhasználva, a *8. ábra* bal oldalának adatait jeleníti meg.



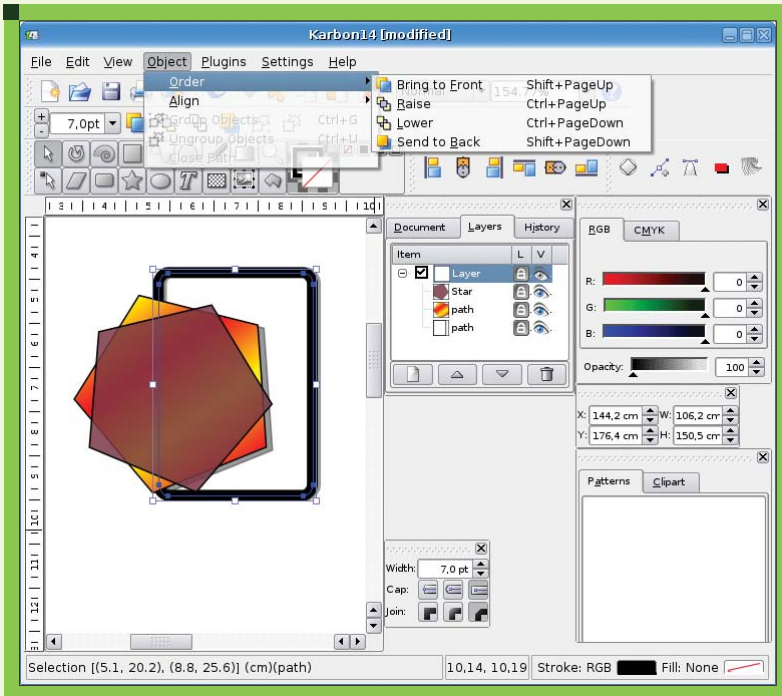
7. ábra Kugar Designer-ben tervezzük meg az adatok kinézetét, elrendezését a report oldalán belül



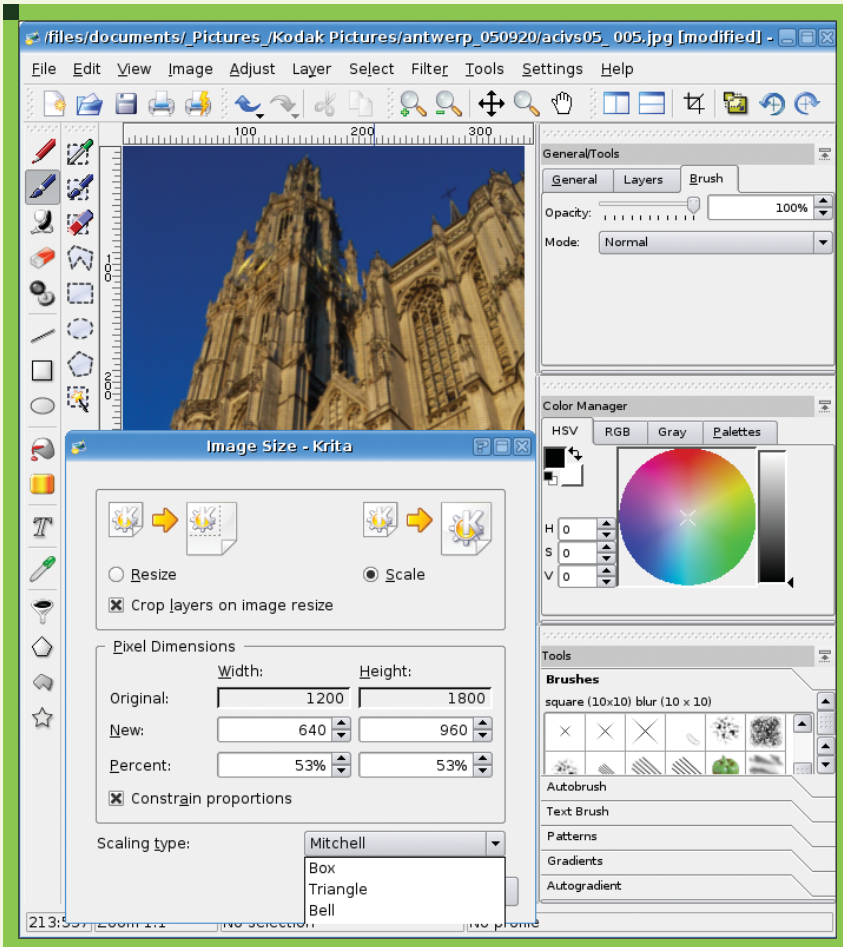
8. ábra A megtervezett elrendezés és az adat-fájl [bal] adataiból a Kugar elkészíti a jelentés végső változatát (jobb)

Rajz és grafika (Karbon), képmanipulálás (Krita)

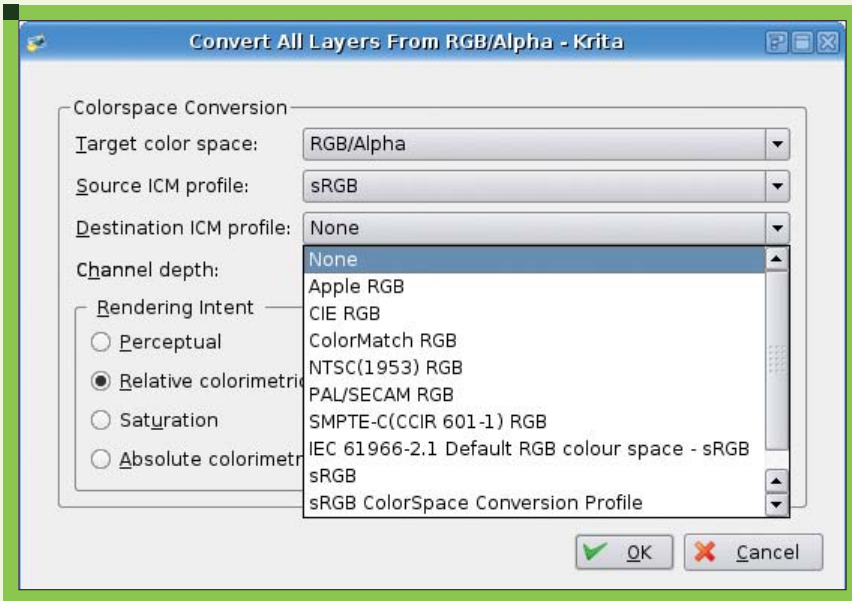
A ***Karbon*** és a ***Krita*** két olyan eszköz, amelyekkel egyrészt vektorgrafikus szerkesztővel különféle rajzokat és illusztrációkat készíthetünk (Karbon), másrészt képeket módosíthatunk, manipulálhatunk, rajzolhatunk (***Krita***).
A ***Karbon*** egy vektorgrafikus rajzolóprogram (*9. ábra*). Helyének megtalálása a hasonló alkalmazások között nem kis feladat, erre nem is vállalkozunk. Mindazonáltal funkcióit tekintve kissé az ***Inkscape*** – szintén nyílt forrású vektorgrafikus alkalmazás – tudását még nem éri el, de nagyon közel jár és tekintve a rohamos fejlődést, elképzelhető hogy utoléri azt, esetleg tovább is fejlődik. A ***Karbon*** számos eszközt tartalmaz, amivel a rajzolás könnyebbé válik, geometriai alakzatok, vonalak, poligonok, szabad vonalrajzok, kezeli a ***layer***-ek et, importálhatunk raszter-képeket. Az elkészített rajzokat elmenthetjük SVG vektorgrafikus formátumba, a ***Karbon*** saját ***XML*** alapú formátumába, ***OASIS OpenDocument, EPS, Adobe Illustrator és Photoshop, Krita, Gimp, PNG, GIF, JPEG, TIFF,*** stb. formátumokba.
A ***Krita*** egy egyszerű és gyors képszerkesztő alkalmazás (*10. ábra*), valahol egy ***KolourPaint***-szerű rajzolóprogram és egy ***Gimp***-szerű képszerkesztő alkalmazás között: az előbbinél többet, az utóbbinál kevesebbet kínál. A ***Krita*** kezeli ***layer***-eket, képes ***ICM*** színprofilokat kezelni (*11. ábra*), de állíthatunk kontrasztot, gamma-korrekciót végezhetünk, színeket állíthatunk csatornánként (utóbbiakat az ***Adjust*** menüben). A fentiek mellett lehetőségünk van néhány egyszerű alakzat rajzolására is, amely estenként nagyon fontos lehet, pl. ha szöveges megjegyzéseket szeretnénk képekre elhelyezni.



9. ábra A Karbon vektorgrafikus rajzolóprogram működés közben



10. ábra A Krita főablaka, képméretezés dialógusa



11. ábra ICM profilok, Image->Mode->Convert Image Type menüpont

Sajnos szűrőkből (*Filter* menü) nem nagyon válogathatunk, gyakorlatilag csak elmosás, egy élesítés, élszűrés, invertálás és egy-két egyszerű effektus áll rendelkezésre.

Gyakorlottak saját 3x3-as konvolúciós szűréseket írhatnak be kézzel (*Filter->Custom Convolution*), de ezzel végére is értünk a szűréseknek.

Elmondhatjuk, hogy a *Krita* nem lesz a haladó képszerkesztők kedvence, és funkciói sem nevezhetők éppenséggel sokrétűnek vagy gazdagnak, mindazonáltal az alapvető funkciókat tartalmazza, és ha gyorsan kis módosításokat kell végrehajtanunk egy-két dokumentumba beillesztendő képen, akkor sebessége és egyszerűsége miatt jó választásnak bizonyulhat. Fagyásokkal, instabilitással szinte soha nem találkozunk.

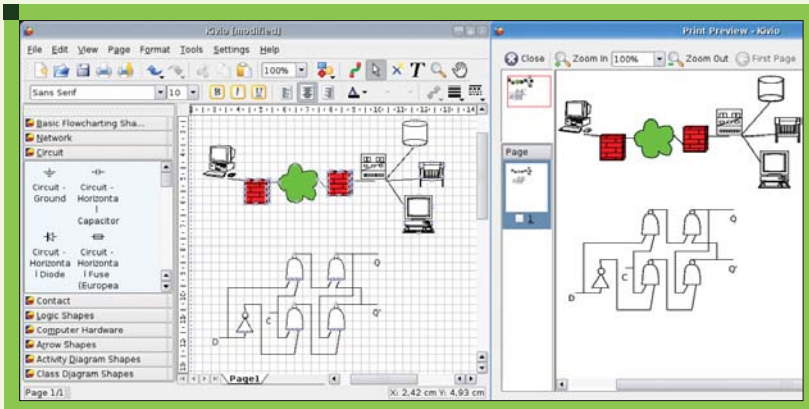
Folyamatábrák – Kivio

A *Kivio* egy, a *Windows* világából ismerhető *Smartdraw*-hoz vagy *Microsoft Visio*-hoz hasonló folyamatábra készítő alkalmazás, amely gyakorta elengedhetetlen eszköz egy-egy dokumentum vagy bemutató készítésekor, ha mondjuk egy áramkör rajzát, egy számítógépes hálózat topológiáját, egy *UML* folyamatábrát, stb. kell készítenünk. Ezekre és egyéb hasonló célokra a *Kivio* egy adhat *KDE* alat, illetve *Linux* alatt megoldást.

A *12. ábrán* látható a *Kivio* működés közben. A *Kivio* főablakának baloldalán találhatók az úgynevezett *stencil*-ek, azok a minta-halmazok, amelyeket használhatunk a diagram vagy folyamatábra elkészítéséhez. Az egyes elemeket egérrel át kell húznunk a rajzlapra, annyiszor, ahány darabra szükségünk van az adott ábrához. Az egyes elemeket a *Tools* menü *Straight/Polyline Connector* elemével lehetséges (az eszköztáron is elérhető).

A lehelyezett elemek tulajdonságait úgy állíthatjuk, hogy egérrel kiválasztjuk, jobbklikkre megjelenő menüből pedig a *Format Text/Stencil/Arrowhead* opciókat választjuk (ezek a szöveg, az elem ill. az összekötővonalak nyílvégződéseit módosítják).

Noha az elérhető *stencil*-ek eléggé sokrétűek, a kínálat jelenleg eléggé elmarad a *Windows*-os *Smartdraw*-tól, vagy *Visio*-tól. A beépített *stencil*-eken kívül sajnos viszonylag kevés ingyenesen elérhetőt találhatunk az Interneten, és sajnos *stencil*-ek készítéséhez is pénzbe kerülő megoldást találhatunk. Ezen kívül meg kell említeni, hogy a *Kivio* viszonylag korai fejlesztési státusza



12. ábra Kivio ablaka, baloldalon a használható stencil-halmazok, a jobboldali ablakrészletben az oldal nyomtatási képe

miatt jelenleg számos, egyébként természetesnek vett funkció hiányzik, például a lehelyezett objektumokat tudjuk méretezni, de nem tudjuk elforgatni.
Mindezek mellett, ***Linux*** alatt jelenleg a ***Koffice***-hez tartozó ***Kivio*** szolgáltatja az egyetlen alternatívát folyamatábrák készítéséhez, és tekintve a projekt fiatal életkorát, nem is teszi ezt olyan rosszul. Stabil, könnyen használható, remélhetőleg rohamosan fog továbbfejlődni.

Végszó

Gyorstalpaló ***KOffice***-t bemutató sorozatunkban a ***KDE***-s irodai alkalmazás-gyűjtemény minden elemére próbáltam némi időt szentelni és a legfontosabb tulajdonságait, képességeit bemutatni. Vannak közöttük olyanok, amelyek majd minden funkciójukban használható alternatívát nyújthatnak más hasonló irodai alkalmazás-csomagok hasonló alkalmazásaira, és vannak olyanok is, amelyekre még fejlődés és tetemes funkcionalitás-bővítés vár. Mindazonáltal remélhetőleg sikerült megmutatnunk, hogy a ***KOffice*** a ***linuxos*** irodai alkalmazások világának egyik legsokoldalúbb és ígéretes tagja, amelyet érdemes kipróbálni és használni, akár hétköznapi otthoni, akár irodai alkalmazásban.

Kovács Levente
(leventek@gmail.com)

26 éves informatikus- és villamosmérnök. Évek óta használ különféle Linux disztribúciókat. Fontosnak tartja a nyílt forrású szoftverek és fejlesztés előnyeinek megismertetését az emberekkel.

KAPCSOLÓDÓ CÍMEK

- www.koffice.org
- www.koffice.org/kexi
- www.koffice.org/kugar
- www.koffice.org/karbon
- www.koffice.org/krita
- www.koffice.org/kivio

Munin

Kiszolgáló terhelésének megfigyelése vizuális típusoknak

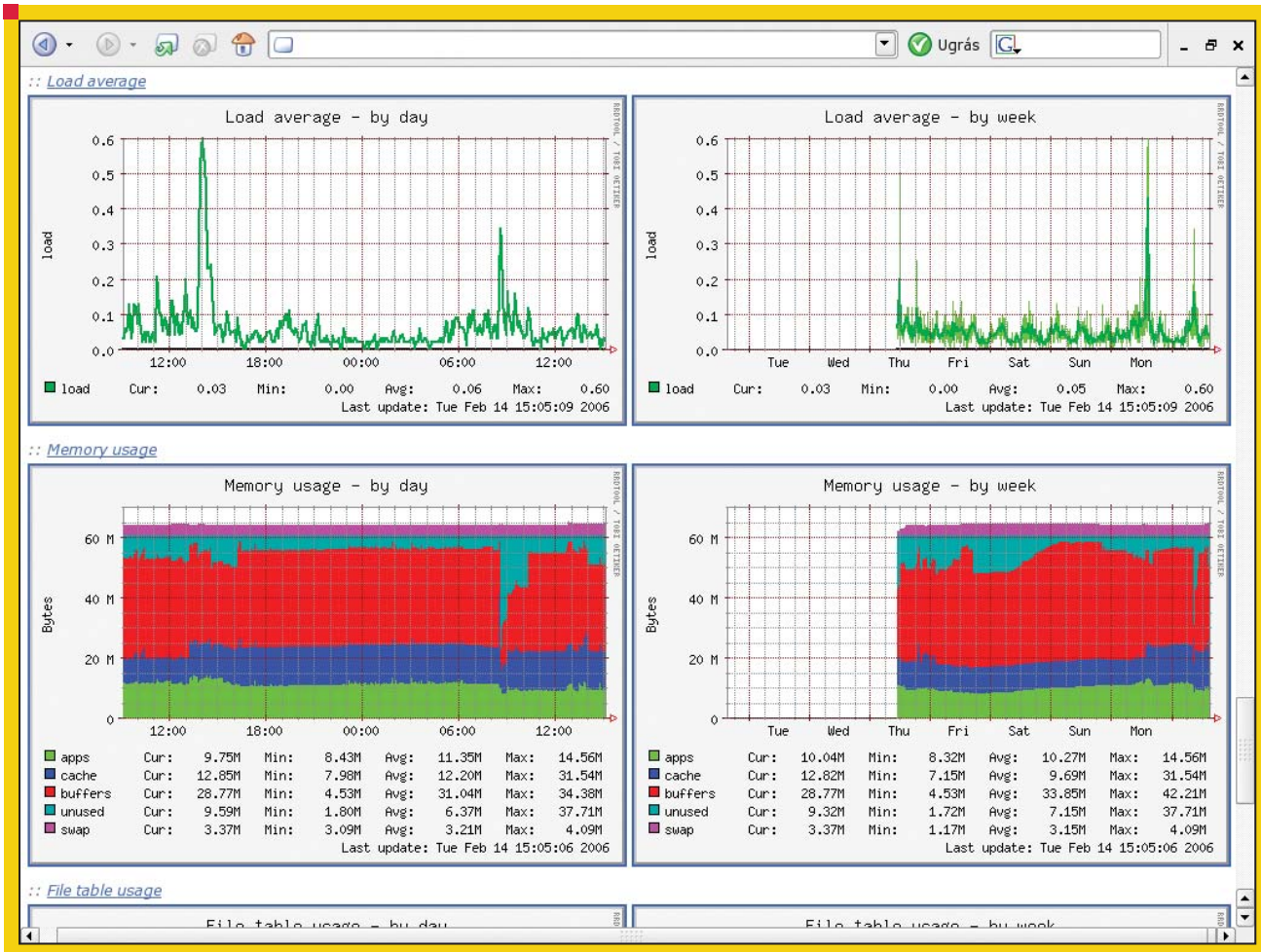
Az RRDTool nagyon hasznos eszköz arra, hogy rendszerünk minden egyes rezdüléséről értesüljünk. Ugyanakkor ha ugyanezt több szerverrel, vagy netán egy egész szerver parkkal szeretnénk megtenni, akkor valamilyen hatékonyabb eszközt kell keresnünk, természetesen RRDTool-lal kapcsolatos ismereteinknek most is hasznát vehetjük.

Amikor valamilyen okok miatt szervereink túl lassúnak bizonyulnak, a rendszergazda általában a következő eszközökhöz nyúl: `top`, `vmstat`, `mtop`.
Az esetek többségében ezekkel az eszközökkel jól meghatározható a probléma forrása. Ha nem voltunk előrelátóak, akkor ezután szokott jönni a rendszerbeállítások optimalizálása. Illetve ha már tovább nem tudjuk optimalizálni a rendszerünket végső megoldásként a vasat szoktuk bővíteni, lecserélni.
Jó lenne elkerülni ugyanakkor azt, hogy az esetlegesen bekövetkező kiszolgáló túlterheltségről ne csak az utolsó pillanatban értesüljünk, mikor már a szerencsétlen vas nem bírja tovább és képtelen kiszolgálni a sok beérkező kérést. Ugyanakkor az sem túl célszerű, hogy a legújabb erőművet állítjuk be, és a processzor terhelése sohasem haladja meg az 5%-ot.

Az ***RRDTool*** kitűnő eszköz erre, ugyanakkor egy grafikon elkészítése nagyon sokáig tart és ha tényleg minden egyes részletet szeretnénk monitorozni, akkor nagyon sok időnkbe telik, mire elkészítjük a megfelelő grafikonokat.
Kellene egy olyan eszközt találni, amely már magában foglalja a legfontosabb kiszolgáló paraméterek monitorozásának képességét. Általában pedig több kiszolgálónk van, ezért az is egy szempont lehet, hogy szeretnénk az összesről egy helyen tárolni és megjeleníteni az információt. Ebben nyújt nekünk segítséget a ***munin***.
A ***munin*** egy kliens-szerver felépítésű monitorozóeszköz, mely az ***RRDTool*** segítségével előállított grafikonokat webes felületen keresztül teszi elérhetővé számunkra.
A kiszolgáló periodikusan begyűjti a kliensekről az információt és ezeket webes felületen keresztül megjeleníti. A ***munin*** eleve tartalmaz rengeteg bővítményt (plugin), melyek segítségével gépeink legfontosabb paramétereit tudjuk megjeleníteni, de lehetőségünk van magunknak is ilyen bővítmények írására, amennyiben egyéb információkat is szeretnénk tudni a rendszerünkről.
Mielőtt belekezdenénk a saját ***munin*** kiszolgálónk telepítésébe, tekintsük meg a ➲ ***http://munin.ping.uio.no/*** oldalon található szerverek statisztikáit. Láthatjuk, hogy a ***muninnal*** szinte minden mérhető és grafikusan megjeleníthető, a ***CPU***-kihasználtságtól kezdve például a levélkiszolgáló ***spam/ham*** arányáig.

Telepítés

A ***munin*** kis erőforrásigényű eszköz, így felesleges lenne külön kiszolgálóra telepíteni, erre a célra tökéletesen megfelel egy már működő webszerver. A ***munin*** programcsomag két komponensből épül fel, az egyik a ***munin***, amely maga a ***munin*** kiszolgáló, ez az alkalmazás fogja periodikusan lekérdezni a többi számítógépet, tárolni az egyes számítógépek kihasználtságát, és az ebből készült grafikonokat megjeleníteni webes felületen keresztül. A ***munin***-t célszerű csak



1. ábra Egy kisebb levelezőszerver átlagos terheltsége és memóriafoglalása

egy gépre feltelepíteni. A másik komponens a ***munin-node*** melyet fel kell telepítenünk az összes számítógépre, amelyről szeretnénk a statisztikákat megjeleníteni.

Legelső lépésként tehát telepítsük fel a ***munin*** és a ***munin-node*** csomagot a ***munin*** kiszolgálóra. A telepítés lépéseit nem részletezném, de természetesen csomagként és forrásból is lehetőségünk van ezt megtenni. Az elkészült grafikonokat a http://localhost/munin alatt rövidesen már el is érhetjük amennyiben szerencsénk van.

Ha minden stimmel, akkor már létezik az oldal, csak meg kell várnunk, amíg az első adatok megjelennek a grafikonokon. Ha csomagból telepítünk, akkor valószínűleg minden működni fog, de ha 5 perces várakozás után mégsem történne semmi, akkor induljunk el hibát keresni.

A konfigurációs fájlokat a későbbiek folyamán majd amúgy is módosítanunk kell, amikor új ***munin-node***-okat veszünk fel.

A kiszolgáló beállítása

A */etc/munin/munin.conf* fájl tartalmazza a kiszolgáló beállításait, és a */etc/munin/munin-node.conf* tartalmazza a csomópontok beállításait. A fájlban található egy kis segítség, de a fenti beállítások például ***Debian*** alatt megfelelőek. Számunkra ebből a `htmldir` fontos, hiszen itt kell megadnunk azt, hogy a ***munin*** hova tegye a generált weboldalakat, ha ez nem stimmel, akkor javítsuk ki, majd indítsuk újra a ***munin-t*** a

```
/etc/init.d/munin restart
```

parancs segítségével.

Ha még mindig nem látjuk a grafikonokat, akkor nézzük meg, hogy a ***munin-node.conf*** fájlban engedélyezett-e az, hogy a ***munin-node***-hoz a ***localhost*** csatlakozhasson. Ez alapértelmezett esetben szintén így van. Ha mégsem lenne ott a megfelelő bejegyzés, akkor adjuk meg azt:

1. kódrészlet Egy minimális munin.conf állomány

```
dbdir /var/munin
htmldir /var/www/munin
logdir /var/log/munin
rundir /var/run/munin

[localhost.localdomain]
        address 127.0.0.1
        use_node_name yes
```

```
allow ^127\.0\.0\.1$
```

Most már remélhetőleg minden működik és idővel az *1. ábrához* hasonló grafikonokat fogunk látni.

A csomópontok beállítása

Minden egyes csomópontra telepítenünk kell a ***munin-node*** csomagot ahhoz, hogy az adott kiszolgálót

monitorozni tudjuk. Lehetőleg próbáljunk meg azonos verziószámú csomagokat telepíteni az összes kiszolgálóra, mert nekem például az egyik csomóponton 1.2.4-1-es verziószámú ***munin-node*** volt, míg a ***munin*** csak 1.2.3-1-es volt, és így az egyik grafikont a szoftver hibásan rajzolta ki. Konkrétan a cpu kihasználtság grafikon y tengelye volt 0-1.0 között megjelenítve, holott az adatok 0%-100% között változtak. Megjegyzem, hogy a grafikon melletti értékek ettől függetlenül jók voltak.
Tehát a ***munin-node*** csomag telepítése után nincs is más dolgunk, mint a ***munin-node.conf*** fájlban beállítani a ***munin*** kiszolgáló ***IP*** címét az `allow` parancs után.
Itt lehetőségünk van akármennyi ***IP*** címet is megadni (mindegyiket új sorban az `allow` után), illetve megadhatunk egész tartományokat is. Majd indítsuk újra a ***munin-node*** programot.
Természetesen ez még nem elég, hiszen a ***munin*** kiszolgálónak is meg kell adni, hogy melyik ***munin-node***-okhoz csatlakozhat.

Tehát a kiszolgálón lévőn ***munin.conf*** fájlt ki kell bővítenünk, hogy ne csak a `[localhost.localdomain]` bejegyzést tartalmazza, ennek kitöltése nyilvánvaló, csak az ***IP*** címeket kell értelemszerűen megadni.

Tetreszabás

Már az első tesztek alatt találhatunk olyan grafikonokat, melyek teljesen érdektelenek lehetnek számunkra, illetve minél több grafikon van, annál kevésbé látjuk át a rendszer tényleg fontos paramétereit. Például egy ***MySQL*** kiszolgálónál kevésbé fontos adat lehet az ***Ethernet*** csatolófelület adatforgalma, mint mondjuk a ***MySQL*** lekérdezések paraméterei, mivel a szűk keresztmetszetet várhatóan a ***MySQL*** sebessége fogja jelenteni.
Az egyes grafikonokat a különböző bővítmények (pluginek) készítik el. A ***munin*** elég sok ilyent tartalmaz, amelyek ***Debian*** alatt a ***/usr/share/munin/plugins*** könyvtárban találhatók. Bővítményeket aktiválni és deaktiválni úgy tudunk, hogy létrehozunk, illetve törlünk egy szimbolikus linket a ***/etc/munin/plugins*** könyvtárból. A bővítményt tartalmazó fájl neve alapján nem mindig nyilvánvaló, hogy az milyen funkciót tölt be, de szerencsére ezt a fájl fejlécében megtalálhatjuk. A változások érvénybelépéséhez már csak ***munin-node*** újraindítása szükséges.

Mindenkinek jó monitorozást kívánok!

Horváth Ernő
ernohorvath@gmail.com
24 éves, műszaki informatikus. Három évvel ezelőtt ismerkedett meg komolyabban a Linux rendszerekkel és emellett érdeklődik még a robotika és a biztonságtechnika iránt is. Ha lenne szabadideje sokat kirándulna, biciklizne és filmeket nézne.

KAPCSOLÓDÓ CÍMEK

A projekt weboldala:
➲ http://munin.projects.linpro.no/

DJBDNS – Élet a Bind-en túl

Az Internet infrastruktúrájának kritikus eleme a DNS szolgáltatás, amelyet jellemzően az ISC BIND alkalmazásával valósítanak meg. Nem mindenki elégedett azonban ezzel.

A ***djbdns*** talán az első olyan alkalmazás, amely pénzgaranciát tartalmaz. Az alkalmazás írója, ***D.J. Bernstein***, 500 dolláros díjat ajánlott fel annak, aki kihasználható biztonsági hibát talál benne. Úgy tudom, a pénz még mindig nála van. A ***bind*** egy nagy, monolitikus alkalmazás (minden funkciót egy alkalmazás végez), ebből következően túl bonyolult (***Bernstein*** 300,000 kódsorról beszél; összehasonlításként a ***Solaris*** forráskódja hosszú ideje stabilan 7 millió sor), a konfigurációs fájl formátuma nehézkes, egyetlen pont hiánya nagy kavarodás okozója lehet, és a biztonsága sem győzött meg minden kételkedőt. Bernstein egy írásában (➲ ***http://cr.yp.to/djbdns/blurb/unbind.html***) éles kritikával illeti a ***bind***-et, a jelenleg domináns ***DNS*** implementációt, ill. annak hibáit. Tervezési hibákat említ, szerinte a ***BIND 9*** gyakrabban összeomlik, mint a ***BIND 8***, a ***BIND 9*** módosításnaplójában (***Changelog***) kereken 672 hiba javítását találta, ami szerinte nem fér össze egy robusztusnak mondott rendszerrel.
Ha az olvasó egy egyszerűen használható, moduláris, nagy teljesítményű és biztonságos ***DNS*** implementációt szeretne, ne keressen tovább: a ***djbdns*** mindent tud, ami ma egy ***DNS*** kiszolgálótól elvárható.
Az alábbiakban bemutatom, hogyan lehet egy fiktív vállalatnál 2 ***DNS*** kiszolgálót építeni, amely nem csak a cég tartományát kezeli, de névfeloldást is biztosít a belső felhasználóknak (*1. ábra*). Legyen a cég tartományának neve ***xxxx.hu***, az ***1.2.3.4*** és ***1.2.3.5*** ***IP***-címeken szolgálja ki az ún. autoritatív kéréseket, amelyek közül legyen az ***1.2.3.4*** az elsődleges kiszolgáló, a cég belső felhasználóinak pedig a ***10.1.1.4*** és ***10.1.1.5*** címeken biztosít névfeloldást. Üzembiztonsági okok miatt a másik kiszolgálót célszerű egy másik hálózaton (például ***co-location***) elhelyezni, így a másodlagos kiszolgáló ***IP***-címe más lenne (és kellene egy újabb gép a cég hálózatán belül, amelyik a belső felhasználóknak névfeloldást biztosít), de ezzel most a példában nem foglalkozom, a telepítés szempontjából mindegy.
Telepítsük először a ***daemontools*** (➲ ***http://cr.yp.to/daemontools/install.html***) ill. az ***ucspi-tcp*** (➲ ***http://cr.yp.to/ucspi-tcp/install.html***) csomagot (ezeken az oldalakon részletesen le van írva a telepítésük), majd töltsük le a ***djbdns*** legutolsó verzióját (➲ ***http://cr.yp.to/djbdns/install.html***), és hajtsuk végre az alábbi parancsokat:

```
tar zxvf djbdns-1.05.tar.gz
cd djbdns-1.05
echo gcc -O2 -include /usr/
➥include/errno.h > conf-cc
make
su -c 'make setup check'
```

Hozzunk létre három felhasználót a ***djbdns*** programjainak futtatásához:

```
groupadd dnslog
groupadd dnscache
groupadd tinydns
useradd -g dnslog -s /bin/false
➥dnslog
useradd -g dnscache -s /bin/
➥false dnscache
useradd -g tinydns -s /bin/
➥false tinydns
usermod -L dnslog
usermod -L dnscache
usermod -L tinydns
```

Konfiguráljuk először a névfeloldást:

```
dnscache-conf dnscache dnslog
➥/opt/dnscache 10.1.1.4
ln -s /opt/dnscache /service
touch /service/dnscache/root/
➥ip/10.1.1
```

És készen is vagyunk az egyik gépen! A

```
svstat /service/dnscache
```

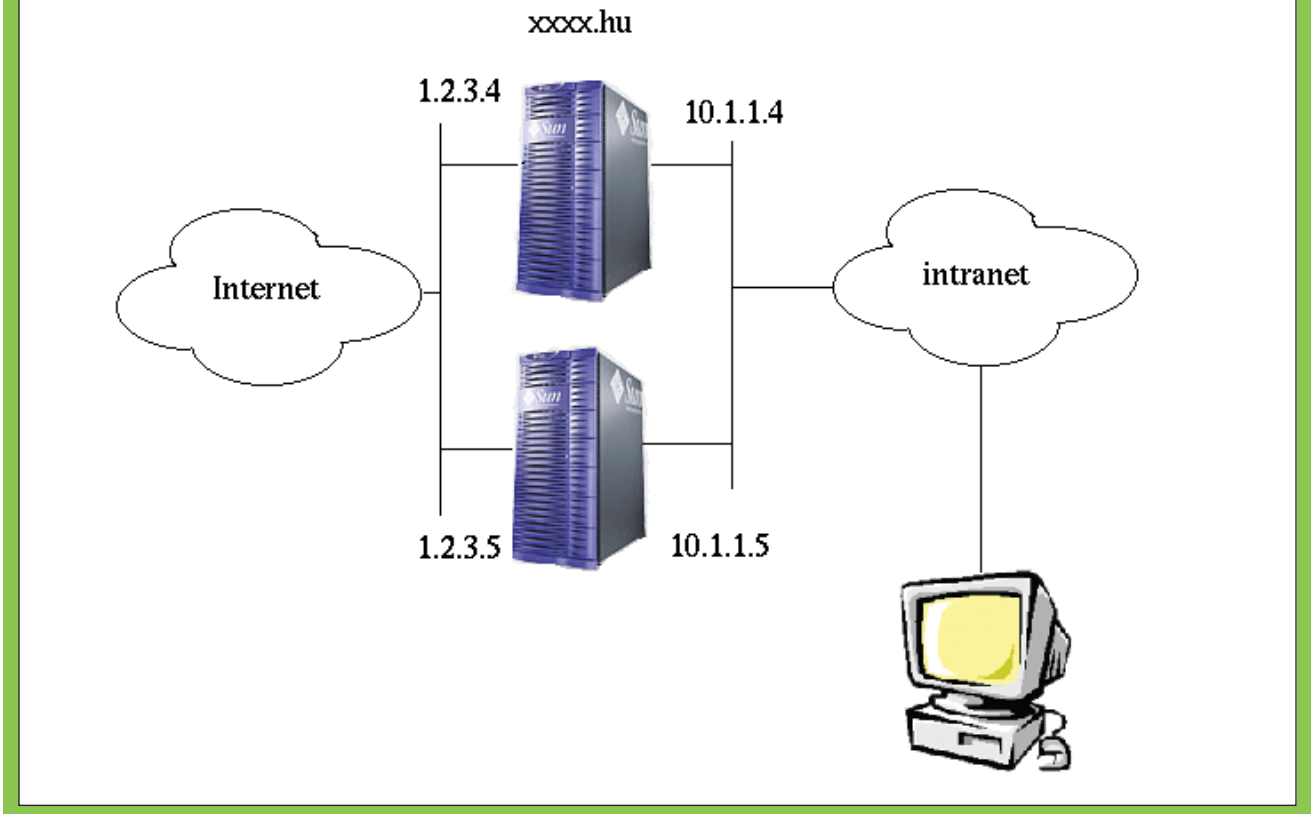
parancs kiadásával meggyőződhetünk, hogy a `dnscache` valóban fut, az én gépemen ezt a választ kaptam:

```
/service/dnscache: up (pid 237)
➥1556402 seconds
```

Az utolsó sor pedig azt jelenti, hogy a ***10.1.1.0/24*** hálózat gépei használhatják a gépen futó `dnscache`-t. Igény szerint több címtartományt illetve gépet is felsorolhatunk.
Opcionálisan hangolhatjuk a `dnscache`-t, például beállíthatjuk a gyorsítótár (cache) méretét 4 MB-ra:

```
echo 4096000 > /service/
➥dnscache/env/CACHESIZE
```

A `dnscache` az ügyfelektől induló lekérdezések kiszolgálásakor először mindig a gyökér kiszolgálókat kérdezi meg – amelyek a */service/dnscache/root/servers/@* fájlban vannak felsorolva – és a kapott úton megy végig



1. ábra

addig a *DNS* kiszolgálóig, amely tudja a választ a megadott kérésre. Így elkerülhető az úgynevezett *cache-poisoning* (gyorsítótár szennyezés), amikor valaki érvénytelen adatok tárolására bírja rá a kiszolgálónkat (➲ *http://en.wikipedia.org/wiki/DNS_cache_poisoning*). A részeredményeket gyorsítótárba teszi, így nem jár be egy utat többször feleslegesen. Csak rekurzív kéréseket fogad el. Következő lépésben indítsuk el az autoritatív névszerver szolgáltatást, hogy le lehessen kérdezni az *xxxx.hu* tartomány gépeinek címét!

```
tinydns-conf tinydns dnslog
➥/opt/tinydns 1.2.3.4
ln -s /opt/tinydns /service
```

Fontos! Nem telepíthetjük a `dnscache` és a `tinydns` kiszolgálót ugyanarra az *IP*-címre, mert mindkettő az *udp/53*-as portot használja. A nehezén már túl is vagyunk. Az előbbihez hasonlóan lekérdezhetjük az alkalmazásunk állapotát. A másik gépen is végezzük el ezeket a beállításokat a megfelelő *IP*-címekkel. Következő lépésben hozzuk létre a cégünk tartományát, és adjunk hozzá néhány *DNS* rekordot (`RR`). Ezt megtehetjük a *djbdns* segédprogramjaival vagy egy szövegszerkesztő segítségével (például *vi*).

```
cd /service/tinydns/root
vi data
make
```

Amint az látható, a *data* fájl egyszerű szöveges állomány. Nem nehéz felismerni a példában szereplő rekordok típusát. Az 1. sorban a zóna *SOA* rekordját láthatjuk, és egy `Z` jel vezeti be.
A pont (`.`) *NS* rekordokat jelöl, míg az = A rekordokat. Tehát a 2-5. sorokban regisztráltuk a tartományunk *DNS* kiszolgálóit. A 6-7. sorokban a levelezést tettük lehetővé: készítettünk egy *MX* rekordot 20-as preferenciával, illetve egy A rekordot a levelező kiszolgálónknak. Végül a cégünk honlapjait tettük elérhetővé, a 8. sorban egy A rekord, a 9-ben pedig egy álnév (alias) szerepel. Végül egy *TXT* rekordot helyeztem el a zónában. A *data* fájl formátumáról bővebben a ➲ *http://cr.yp.to/djbdns/tinydns-data.html* oldalon található információ.

A *tinydns* azonban nem szövegfájlból dolgozik, hanem *CDB* formátumú adatállományból. A szükséges konverziót a `make` paranccsal végezhetjük el. A *djbdns* sokkal hibatűrőbb, mint a *bind*. Ha a *bind* konfigurációs állományában csak egy hibát is vétünk, az jó időre elfoglaltságot biztosít, amíg azt ki nem javítjuk.
A *djbdns* azonban egyszerűen figyelmen kívül hagyja a hibás rekordot, és megy tovább az élet. A *bind*-del ellentétben nem szükséges `HUP` jelzést küldeni vagy más módon értesíteni a kiszolgálót, hogy megváltozott a zóna adata, automatikusan az új adatokkal dolgozik.
A zónánk adatait szinkronizálni szükséges az elsődleges és a másodlagos kiszolgálónk között.
A *bind* ezt jellemzően inkrementális (*IXFR*) vagy teljes (*AFXR*) zónatranszferrel oldja meg. A *djbdns* ezt a feladatot sokkal elegánsabban végzi el, nem találja fel újra a kereket, hanem egy külső eszközzel másolja át a platform független *data.cdb* fájlt a másik gépre: `rsync`-kel vagy egyszerűen `scp`-vel (aligha van adatátvitelre jobb megoldás ennél). Ehhez

1. Lista A data fájl tartalma

```
Zxxxx.hu:ns1.xxxx.hu:hostmast
➥er.xxxx.hu:2005012001:28800:
➥7200:604800:10800::
.xxxx.hu:ns1.xxxx.hu:ns1.xxxx
➥.hu:259200
.xxxx.hu:ns2.xxxx.hu:ns2.xxxx
➥.hu:259200
=ns1.xxxx.hu:1.2.3.4:86400
=ns2.xxxx.hu:1.2.3.5:86400
@xxxx.hu::mx1.xxxx.hu.:20:
➥86400
=mx1.xxxx.hu:1.2.3.6:86400
=www.xxxx.hu:1.2.3.7:86400
+www2.xxxx.hu:1.2.3.7:86400
'xxxx.hu:tinydns really
➥rocks:86400
```

készítsünk el egy jelszó nélküli *SSH* kulcsot, és módosítsuk a *Makefile*-t így:

```
data.cdb: data
  /usr/local/bin/tinydns-data
  scp -i /home/sj/.ssh/
  ➥ssh_kulcs data.cdb
1.2.3.5:/service/tinydns/root
```

Ezután amikor csak aktualizáljuk a `make` paranccsal az elsődleges kiszolgáló adatait, az automatikusan átkerül a másodlagosra. Feladat megoldva, cégünk felhasználóinak névfeloldást végzünk, az egész internet számára pedig kiszolgáljuk az *xxxx.hu* tartományba intézett kéréseket.
Tegyük fel, hogy a cégünknek van egy belső, nem regisztrált zónája is, például *sales.aa*, amelyeket szintén az *1.2.3.4* és *1.2.3.5* gépeken akarunk tartani. Hogyan tudathatjuk a névfeloldást végző gépekkel, hogy ezt a zónát hol keressék? Hozzuk létre a */service/dnscache/root/servers/aa* fájlt, amelybe beírjuk az adott tartomány kiszolgálóit (soronként egyet).
Ha valaki nem olyan szerencsés, hogy a nulláról kezdheti egy *DNS* kiszolgáló építését, annak sem reménytelen a helyzete: a *djbdns* hasznos eszközökkel segíti a migrációt. Vegyük azt az esetet, amikor van egy bind8/9 kiszolgálónk, amelynek az adatait *tinydns*-re akarjuk költöztetni. Tegyük fel, hogy van két tartományunk: *domain1.hu* és *domain2.hu*. Állítsuk be a *bind* kiszolgálón, hogy engedélyezze számunkra az *AXFR* zóna transzfert az *1.2.3.3:53/tcp*-n:

```
options {
    request-ixfr no;
    allow-transfer { 1.2.3.4; };
};
```

```
for i in `domain1.hu
➥domain2.hu`; do
/usr/local/bin/tcpclient
➥1.2.3.3 53 \
/usr/local/bin/axfr-get $i
➥file1 file1.tmp; cat file1 >>
➥data; rm -f file1; done
```

A parancs lefuttatása után megkapjuk a *data* fájlt, amit a `make` paranccsal „fordíthatunk le”, és készen vagyunk.
A naplózással kapcsolatos dolgok a */service/tinydns/log* könyvtárban vannak. A *run* fájl egy héjprogram, ami a naplózó alkalmazást futtatja. Módosítsuk ezt az alábbi módon:

```
#!/bin/sh
exec setuidgid dnslog multilog t
➥2048000 ./main
```

Ennek hatására a `multilog` program 2 MB-onként lerotálja a naplófájlt, így az nem fog a végtelenségig hízni. A modularitás előnye, hogy a naplózást minden *djbdns* programnál (sőt *Bernstein* összes programjánál) a `multilog` végzi, így ugyanezt beállíthatjuk a *dnscache*-nél vagy akár a *qmail*-nél is.
A konkrét naplófájl a */service/tinydns/log/main/current* fájlban található.

```
@4000000043cbb73f05ee267c
➥0a010102:802a:3bce + 000f
➥acts.hu
```

Ez a bejegyzés az *acts.hu MX* rekordjának lekérdezése során keletkezett. Az időbélyeg úgynevezett *TAI64N* formátumban (➲ *http://cr.yp.to/daemontools/tai64n.html*) van az elején. Emlékszik még valaki a 2000. év körüli dátum-mizériára? A *UNIX* 4 byte-os időbélyeg formátuma 2038-ban fog túlcsordulni (de addigra már biztosan 8, 16 esetleg 32 byte-on tárolja a dátumokat a jövő 256-bites processzora). Ezzel ellenben a *TAI64N* formátum már a jelenlegi architektúrákon is több milliárd évet képes ábrázolni. Ezután a kliens *IP*-címe (10.1.1.2) és *UDP* kapuja következik, a *DNS* kérés azonosítója, majd az eredmény (+: ok), végül pedig a lekérdezés típusa (`000f: MX`) és a keresett tartomány név (a *DNS* csomag úgynevezett *DNAME* része). A *djbdns* naplóformátumáról bővebb információ a ➲ *http://dqd.com/~mayoff/notes/djbdns/tinydns-log.html* oldalon található.
Nem olyan régen történt, hogy a *Verisign* egy helyettesítő (*wildcard*) A rekordot helyezett el a *.com* és *.net* tartományokban, amely a *64.94.110.11 IP*-címre mutatott (➲ *http://tinydns.org/*). Ennek hatására minden olyan kérés, amely nem létező tartományra irányult, a *Verisign* gépének címét kapta vissza. Ezért ezt semlegesítendő *Russ Nelson* készített a *djbdns*-hez egy úgynevezett *Verisign*-foltot (➲ *http://tinydns.org/djbdns-1.05-ignoreip.patch*). Aztán kiderült, hogy több más regisztrátor is folytat(ott?) ilyen megkérdőjelezhető gyakorlatot, ezért *Nelson* írt még egy foltot, amivel több címet is figyelmen kívül lehet hagyni (➲ *http://tinydns.org/djbdns-1.05-ignoreip.patch2*). Egy másik lehetőség lehet a védekezésre az, ha a problémás regisztrátorok gyökér névszervereit töröljük a *dnscache* listájából (*/service/dnscache/root/servers/@*), csak aztán maradjon néhány (➲ *http://homepages.tesco.net./~J.deBoynePollard/FGA/verisign-internet-coup.html*).
Tegyük fel, hogy az olvasó még mindig *bind*-et futtat, és ugyanazon az *IP*-címen szolgálja ki a hozzá tartozó tartományokat és az ügyfelek névfeloldás kéréseit. Legyen adott egy támadó, aki több adattal árasztja el a gyorsítótárat, mint amit a *bind* kezelni képes. Ekkor szétszakad nem csak a kimenő, de a bejövő web forgalom is a *bind* zavara miatt (➲ *http://cr.yp.to/djbdns/separation.html*). Nagyban megkönnyíti hát az életet, ha szétválasztjuk a névfeloldást és az autoritatív névszerver szolgáltatást, ahogyan azt az *ISC* is javasolja. De amíg ez korántsem magától értetődő a *bind*-nél, addig a *djbdns* kikényszeríti ezt.
Nagyobb mennyiségű adatnál szükség lehet arra, hogy az egyes zónák adatait külön fájlokban, esetleg adatbázisban tároljuk. A *http://tinydns.org/* oldalon találunk olyan programokat,

1. táblázat *A djbdns csomag segédprogramjai*

`dnsip`	Egy FQDN tartománynevet old fel
`dnsname`	Egy IP-címhez tartozó tartománynevet ad vissza
`dnsname`	Egy IP-címhez tartozó tartománynevet ad vissza
`dnsmx`	Egy tartomány MX kiszolgálóit és azok preferenciáit mutatja meg
`dnstxt`	Megkeresi a tartománynév TXT rekordját
`dnstrace`	Végigkövethetjük vele, mely DNS kiszolgálók ismerhetik az adott rekordot
`dnsq`	Nem rekurzív lekérdezést küld a megadott kiszolgálónak (tinydns tesztelésére)
`dnsqr`	Rekurzív lekérdezést küld a /etc/resolv.conf-ban megadott kiszolgálók felé (dnscache tesztelésére)

amelyekkel kényelmesen, egy böngészőből kezelhetjük a ***tinydns*** adatait, némelyiket kifejezetten többfelhasználós, elosztott környezetbe szánt az írójuk, így tartalmaznak például felhasználó kezelést és különböző jogosultság szinteket.
Bizonyos esetekben szükség lehet arra is, hogy egyes gépek számára zónatranszfert biztosítsunk. Az ***axfrdns*** szolgál erre a célra, és a ***tinydns data.cdb*** állományából dolgozik. Természetesen meg lehet és kell adni azt, hogy mely gépek ***AXFR*** kéréseit szolgálja ki.
A feketelistákat gyakran alkalmazzák a levelezés kapcsán. Az ***rbldns*** kifejezetten erre a célra készült. Abban különbözik a ***tinydns***-től, hogy kizárólag olyan ***DNS*** lekérdezésekre válaszol, amelyek vagy A vagy ***TXT*** rekordra irányulnak, illetve az adatállománya is egyszerűbb formátumú.
A ***tinydns-rrd*** (➲ ***http://develooper.com/code/tinydns-rrd/***) alkalmazás az ***rrdtool*** segítségével képes grafikonokat készíteni a ***djbdns*** terheléséről, amelyek jól jöhetnek egy esetleges hibaelhárítás során is.
A ***djbdns*** csomagban különböző segédprogramok is vannak, amelyek a `host`, `nslookup` és `dig` parancsokat helyettesítik (*1. Táblázat*).
A ➲ ***http://www.fefe.de/dns/*** oldalon található folt segítségével felokosítható a ***djbdns***, hogy kezelni tudja az ***IPv6*** címeket is.
Egy ***DNS*** kiszolgálóval szemben követelmény, hogy könnyen használható, nagy teljesítményű, hibatűrő és biztonságos legyen. Szinte észrevétlenül kell működnie, hogy észre se vegyük a jelenlétét. A ***djbdns*** egy ilyen ***DNS*** implementáció, nekem bevált, ajánlom mindenkinek.

Sütő János
(jsuto@freemail.hu)

1997 óta használ Slackware Linux-ot. Szabadidejében a postfix clapf nevű vírus- és spamszűrőjét polírozza.

Operációs rendszerek egyenrangú közlésen alapuló védelme

Az internet növekedésével és elterjedésével a számítógépes biztonsági kérdések egyre inkább előtérbe kerültek. Manapság az egyik legnagyobb veszélyt számítógépeinkre a hálózati támadások jelentik. A cikkben bemutatott program éppen a hálózatot próbálja meg arra használni, hogy a kiszolgálók a támadásoknak minél hatékonyabban ellenállhassanak. A működés során a programot futtató számítógépek egy P2P hálózatba szerveződnek, amelyen keresztül megosztják az érzékelt betörési kísérletekről gyűjtött tapasztalatokat, így növelve az összes résztvevő biztonságát.

A szoftvert *Komondor*nak neveztük el, hiszen feladatkörében sok minden hasonló a házőrzésben híresen kiváló kutyafajtára. Léteztek már korábban is a hálózaton egymással kapcsolatot tartó biztonsági programok. Az itt bemutatott szoftver újdonsága az, hogy az egyes gazdagépeken futó példányok az Interneten egy *egyenrangú* (*peer-to-peer, P2P*) hálózatot hoznak létre. A szerveződés önműködő, felhasználói beavatkozást nem igényel. Ez a hálózati modell nagy stabilitást biztosít, amelyre az egyes egyedek között a tapasztalatok gyors, megbízható átadása miatt van szükség. A rendszer felépítéséből adódóan a hálózati hibák és főként a támadások miatt labilis hálózaton is működőképes marad.

Fontos megjegyezni, hogy a *Komondor* elfedni hivatott az egyes egyedek operációs rendszereinek, szolgáltatásainak biztonsági réseit, nem pedig kijavítani. Ehhez nincsen szüksége az adott biztonsági rés ismeretére. Előzetesen képes lehet bizonyos védelmet kialakítani, de csak akkor, ha valahol egy helyen történt már betolakodási kísérlet. Nem az adott rést foltozza be, hanem a konkrét támadót akadályozza meg a további tényekedésben. Ha az adott rés ismert, érdemes inkább annak közvetlen kijavításával foglalkozni.

A program első változata *Linux* rendszerre épül, *C* programozási nyelven íródott. A bemutatása után becslést adunk egy ilyen átfedő általános használhatóságára, bemutatjuk az előnyeit és a korlátait is.

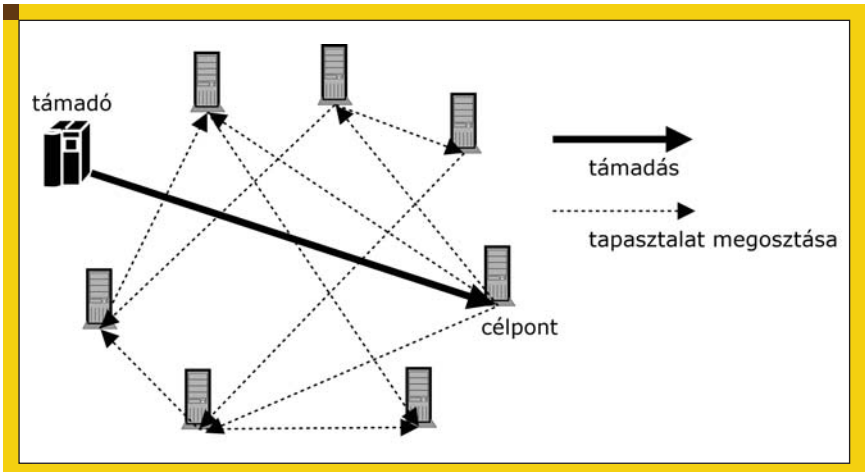
A biztonsági rések és a támadások természete

Ha a támadó egy adott számítógép *működését zavarni* szeretné, vagy *adatokat* szeretne megszerezni róla, akkor a célba vett kiszolgáló hálózati címe általa előre ismert. Ebben az esetben nyitott hálózati *kapukat* (port) keres szolgáltatások után kutatva, annak reményében, hogy biztonsági rést talál. Gyakori a *kapupásztázás* (port scan), amely a számítógépen futó alkalmazások hálózatról való megcímzéséhez szükséges kapuk teljes tartományának végigpróbálását jelenti. Ennek célja az, hogy találjon valamelyik kapun egy hibás alkalmazást, amelyet azután felhasználhat a rendszerbe való bejutáshoz. Kifejezetten erre a célra tervezett programok is léteznek. Ezek nem feltétlenül rossz szándékból íródtak, hanem saját rendszerünk biztonságának tesztelésére is alkalmasak. Ilyen alkalmazás a jól ismert *Nmap*. Biztonsági résnek számítanak a hanyag felhasználók *gyenge jelszavai* is. Ezt használja ki az ún. szótár módszer, amely arra a tényre épül, hogy a gyenge jelszavak köznevek vagy gyakori tulajdonnevek. Ezek viszonylag kis számú próbálkozással kitalálhatóak. Hibás szolgáltatáson keresztül *erőforrások* után is kutathat a támadó. Jellemzőbb ekkor az, hogy a pásztázás egy adott *IP* szám tartomány ellen irányul. A kapuszám ilyenkor kötött: például a 25-ös port, *SMTP* kiszolgáló keresése levélszemét küldés céljából. A fenti támadások bár különböző módszerekre épülnek, mégis közös vonásuk, hogy a támadó több kísérletet tesz céljának elérésére. A *Komondor* ezt használja ki: ha ugyanis valamelyik *Komondor* egyed betolakodást észlel, és ezt a tényt megosztja a többiekkel, akkor azok már felkészülve képesek várni *ugyanazt a támadót*, aki módszereinél fogva (pásztázás) nagy valószínűséggel előbb-utóbb meg is érkezik.

A kiszolgálók biztonsága

Egy számítógép soha nem lehet teljesen biztonságos. A szakirodalom meghatározza a *megfelelően képzett támadó* fogalmát, egy olyan elméleti személyt, aki szaktudásánál fogva képes felderíteni bármilyen biztonsági rést. Tudjuk, hogy nem hozhatunk létre tökéletesen hibamentes rendszert. Ennek ellenére számítanunk

kell rá, hogy bármely hibát lehetséges megtalálni, akár szisztematikusan, akár véletlenül.
Az átlagfelhasználónak sokat nem kell tennie a rendszere biztonságossá tételéhez. Értékes adatok tárolása esetén azonban mindenképpen foglalkoznunk kell ezekkel a kérdésekkel. Fontos megjegyezni, hogy minél biztonságosabb egy rendszer, annál kisebb a használhatósága a szigorú feltételek miatt. A biztonság és a használhatóság között kompromisszumot kell létrehoznunk. Egyszerű példa erre a hálózati forgalom korlátozása.



1. ábra Támadás a Komondor rendszer egyik tagja ellen

A betolakodás-észlelés fogalma

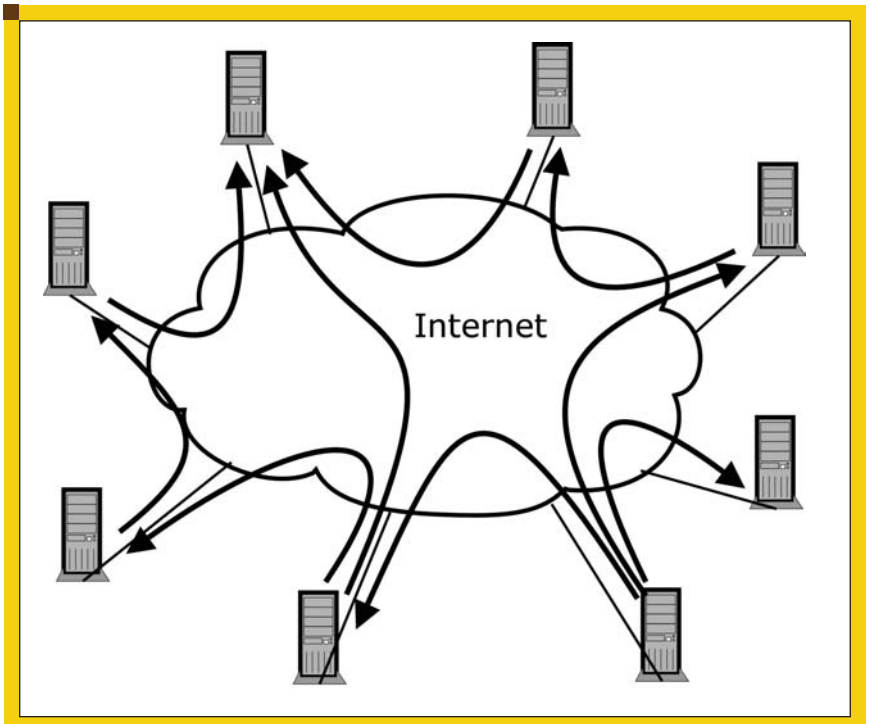
A betolakodás-észlelő technikáknak két fő kategóriája van: a ***rendellenesség-észlelés*** (***anomaly detection***) és ***visszaélés-észlelés*** (***misuse detection***).
A ***rendellenesség-észlelés*** a felhasználók, illetve az alkalmazások elvárt viselkedésének modelljeit felhasználva az ettől való eltéréseket problémaként értelmezi. A rendellenesség-észlelési rendszerek fő előnye, hogy előzetesen tudják észlelni a támadásokat. Azzal, hogy meghatározzák, hogy mi a normális, ennek bármilyen megsértését azonosítani tudják függetlenül attól, hogy ez része-e a ***fenyegetési modell***nek (***threat model***) vagy nem. A módszer hátránya ugyanakkor a gyakori a téves riasztás és az, hogy az időben gyorsan változó rendszerekre nehéz az eljárást betanítani.
A ***visszaélés-észlelési*** rendszerek lényegében azt határozzák meg, hogy mi a rossz. Támadás leírásokat, más néven ***aláírás***okat (***signature***) tartalmaznak, melyeket összemérnek a felülvizsgálás adatfolyamával, keresve az ismert támadások tanújelét. A visszaélés-észlelési rendszerek előnye, hogy a felülvizsgálati adatok elemzésére irányul és ritkán vezet téves észleléshez. Ugyanakkor a módszer hátránya, hogy csak az ismert támadásokat tudja észlelni, amelyeknek van egy meghatározott aláírása. Ha egy új támadást felderítenek, azt a fejlesztőknek modellezni kell és hozzáadni az aláírás-adatbázishoz.
Fontos megemlíteni, hogy nem minden támadás jár együtt önműködően észlelhető, arra utaló jelekkel, egy rendszer feljogosított felhasználója általi visszaélés például nagyon nehezen észlelhető. Mégis a betolakodás-észlelés első módja a felhasználói tevékenység megfigyelése volt. Ekkor a szokatlan viselkedésekre figyeltek fel. Ilyen például ha egy felhasználó szabadságon van, s mégis helyileg be van jelentkezve. Az ilyen észlelés hátránya, hogy alkalmi jellegű és nem méretezhető bonyolult rendszerekre.
A betolakodás-észlelés fejlődése terén a következő lépés a operációs rendszer naplófájljainak figyelése, főként ***UNIX*** alapú rendszereknél. Több biztonsági segédprogram is erre épül, köztük az ismert ***Swatch*** (***Simple WATCHer for logfiles***), amelyről már esett szó a ***Linuxvilág 2001. augusztusi*** számában is. Az alkalmazásnak rendszernapló fájlok neveit adhatjuk meg, illetve azokhoz tartozó sorokat, részleteket. Ha az adott fájlban hibaüzenetnek megfelelő szövegrész jelenik meg (például ***file not found***), a ***Swatch*** előre meghatározott műveletet hajt végre: adott programot indít el, e-mailben értesíti a rendszergazdát stb.
Vállalati méretű rendszerek védelemre alkalmasak az összetett szerkezetű, ***hálózati betolakodás-észlelő rendszerek*** (***Network Intrusion Detection System, NIDS***). Ezek közül a kereskedelemben kapható ***RealSecure***, míg a ***Snort*** nyílt forráskódú. A ***Snort*** egy szabályleíró nyelvre épül, amely az adatminták, a hálózati protokollok és a rendellenességek vizsgálatát, illetve ezek kezelését is támogatja. Elsősorban a hálózati forgalom megfigyelésére (szonda) alkalmas, jól konfigurálható rendszert valósít meg; a szabályok (aláírások) adatbázisát az Internetről folyamatosan tudja frissíteni. A ***Snort*** fejlesztői és a felhasználói közössége által létrehozott új szabályok így azonnal beépülhetnek a szoftver által használt adatbázisba.

Válasz a betolakodásra

Egy probléma észlelésekor a betolakodás-észlelő rendszer által végrehajtott tevékenység sokféle alakot ölthet.
A legközönségesebb egy riasztás létrehozása, amely leírja az észlelt betolakodást. De a válasz lehet támadóbb is, mint amilyen egy rendszer adminisztrátor felhívása, egy sziréna megszólaltatása vagy éppen egy ellentámadás beindítása.
Az ellentámadás magába foglalhatja egy útválasztó újraalakítását úgy, hogy zárolja a támadó címét vagy még meg is támadja a tettest. Természetesen a támadó válasz veszélyes is lehet, mivel lehet, hogy ártatlan áldozat ellen indul. Erre példa az, hogy ha egy támadó a hálózatot ***átejtéses forgalom***mal (***spoofed traffic***) terheli meg. Az ilyen forgalom egy adott címről származóként jelenik meg, de ténylegesen máshol hozták létre. Ha a betolakodás-észlelő rendszer a támadás észlelése után újraalakítja a hálózati útválasztókat úgy, hogy zárolják a tettetett címről érkező forgalmat, akkor ez hatásában egy ***szolgáltatásmegtagadásos*** (***Denial of Service, DoS***) támadás végrehajtása lesz a vétlen hellyel szemben.

A Komondor részei

Különböző kiszolgálókon a programnak egyforma példányai futnak és figyelik a hálózaton észlelt esetleges betolakodási kísérleteket.



2. ábra Szabályozott elárasztás a Gnutella átfedőben

Ha egy ***Komondor*** egyed felismer egy, az általa felügyelt rendszer ellen irányuló támadást, akkor

- az általuk felügyelt rendszer védelmét erősíti, ami jellegzetesen a tűzfal szigorítását jelenti;
- a hálózaton keresztül értesíti a többi szoftveregyedet.

Az egyedek így egymást védelmezik. A védelem szempontjából lényegtelen, hogy egy adott támadó rosszindulatú ténykedését melyik egyed érzékelte először. Ha a támadási kísérletet egyszer rögzítette valamelyik egyed, akkor a többi a tapasztalat megosztása révén már felkészülhet (lásd az *1. ábrát*). A rendszer feladatai két részre oszthatóak, helyi és hálózati feladatkörre.

Tapasztalatok szerzése

A ***Komondor*** jelenleg működő, ***Linux*** rendszeren működő változata a ***rendszernapló fájlok vizsgálatával*** gyűjt tapasztalatot. Ez tulajdonképpen bármilyen betolakodási forma felderítésére alkalmas lehet, ugyanis a fent említett programok (például ***Snort***) is képesek ilyet vezetni. A naplókban többféle hibaüzenet jelenhet meg, amelyek támadásra utalnak.

Ilyen lehet bejelentkezési kísérlet nem létező felhasználói néven, vagy több egymás utáni kísérlet nem létező fájl letöltésére a ***HTTP*** kiszolgálón. Íme például egy tetten ért ***SSH*** féreg:

```
Sep 30 05:19:46 horka
➥sshd[6244]: Failed password
➥for illegal user munka from
➥::ffff:192.188.242.112 port
➥1391 ssh2
Sep 30 05:19:46 horka
➥sshd[6246]: Illegal user
➥munkaugy from
➥::ffff:192.188.242.112
Sep 30 05:19:46 horka
➥sshd[6250]: Illegal
➥user juhasz from
➥::ffff:192.188.242.112
```

A ***Komondor*** több naplófájlt is képes egyidejűleg követni. Az egyes naplófájlokhoz különböző kereső karakterláncokat adhatunk meg, amelyek a fentiekhez hasonló hibákra utalnak. Az egyes karakterláncok tetszőleges hosszúságú úgynevezett ***szabályos kifejezések*** (***regular expression***) lehetnek, amelyekhez két további idő paraméter is tartozik. Egyik, hogy az adott karaketrlánc milyen gyakori ismétlődése jelent támadást, ez másodperc nagyságrendű. (Például adott címről három másodpercen belül két bejelentkezési kísérlet nem lehet egy igazi felhasználó, aki csak a jelszavát elgépelte.) Másik, hogy a letiltás, a ***Komondor*** által a támadóval szemben életbe léptetett biztonsági szabály mennyi ideig legyen érvényes. Bármilyen megkötésről van is szó, nem lehet örök érvényű. Például ha a tiltás alapjául a támadó *IP* számát vesszük, annak a tulajdonosa változhat dinamikus *IP* szám kiosztás esetén. Ez inkább óra, vagy nap nagyságrendű időintervallum, bár támadó pásztázás esetén egy néhány perces blokkolás már eredményt hoz.

A Komondor által nyújtott védelem

Felmerült a kérdés, hogy a ***Komondor*** hol avatkozzon be a saját gazdagépe működésébe, a betolakodások elkerülése végett. Alkalmazási szinten történő beavatkozáshoz az egyes szolgáltatást nyújtó programok módosítására lehetne szükség. A szolgálat forráscím alapján történő megtagadását ugyanis nem mindegyik alkalmazás támogatja. Gondot jelent az is, hogy nagyobb méretű programok (például az ***Apache***) lassan indulnak, és a módosított beállításaikat is lassan olvassák így be. Az első változat ezért inkább a tűzfalon keresztül védi meg a rendszert, eldobásra kijelölve a támadó *IP* számról érkező csomagokat. A statikus csomagszűrés ezután a megadott ideig a műveleti rendszer feladata. Az idő leteltével a ***Komondor*** törli a tűzfalból a tiltó szabályt. A program így egy olyan speciális ***adaptív tűzfalként*** viselkedik, amely a tapasztalatait az alkalmazási szinten, illetve az alkalmazási szint felett gyűjti. A ***Linux*** rendszerben az ***iptables*** paranccsal hangolhatjuk a rendszermagba épített tűzfalat, amely egy statikus csomagszűrő, a hangolását szoftverünk biztosítja.

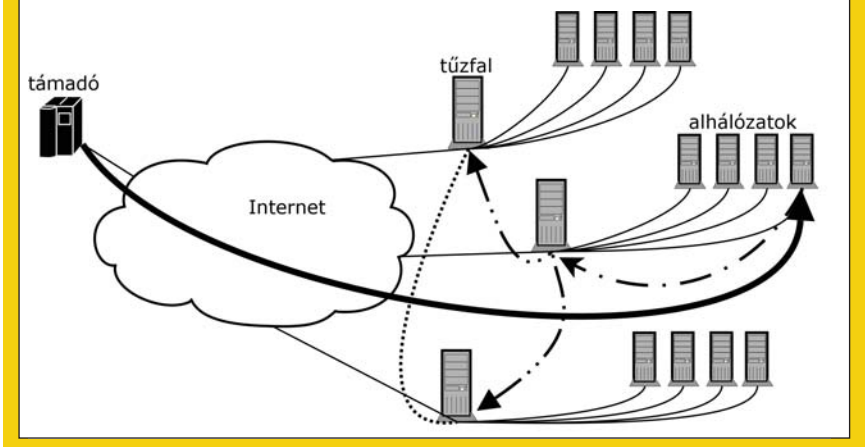
A hálózati modell kiválasztása

A szerzett tapasztalatokról a rendszer adatbázist vezet, és a többi egyeddel megosztja azokat. Az adatbázis megosztásának sebessége nagyban függ az alkalmazott hálózat felépítésétől. A választás két szempont miatt is a ***P2P*** topológiára esett. Egyrészt ez a tapasztalatok gyorsabb megosztását teszi lehetővé nagyobb számú csomópont esetén. Másrészt, egy ilyen hálózat jobban eltűri a csomópontok

megbízhatatlanságát, míg egy központosított (ügyfél / kiszolgáló) felépítésű rendszer a központ kiesése esetén azonnal működésképtelenné válik.
A megosztás gyors és megbízható kivitelezéséhez a *P2P* hálózatok közül is a *Gnutella* alkalmazásban is használt felépítés tűnt a legalkalmasabbnak.
A *Gnutella* a csomópontokat *serventnek* nevezi, mivel azok kiszolgálók és ügyfelek is egyben (*server+client*). Az új csomópont a hálózatba belépéskor egy úgynevezett *horgonyhoz* (*anchor*) kapcsolódik, amelytől megkapja az éppen működő csomópontok hálózati címét. Ezek után megpróbál kapcsolódni ezekhez az egyenrangúakhoz egy *ping* üzenettel; a kapcsolódási pontokat kínáló csomópontok *pong* üzenettel válaszolnak neki. A keresés úgy történik, hogy az egyenrangú szabályozottan elárasztja a kérdezését hét csomópont felé (2. *ábra*). Az ilyen kérést fogadó csomópont, függetlenül attól, hogy ő maga talált-e egyezést, ugyancsak hét másik csomópontnak küldi tovább. A mi programunkban a tapasztalatok továbbítása történik ehhez hasonló módon.
Az egyenrangúak legalább három, de legfeljebb öt másik egyeddel tartanak kapcsolatot. Ezek a paraméterek a programban egyszerűen beállíthatóak, segítségükkel vezérelhető a hálózat sűrűsége. A sűrűbb hálózat gyorsabb tapasztalatcserét és nagyobb megbízhatóságot biztosít, de nagyobb forgalmat is eredményez, ugyanis az egyedek nem fa-gráfba rendeződnek, így egy tapasztalat adatai több úton is érkezhetnek.
A rendszer indítása után az egyedek nem várják meg, hogy találjanak három megfelelő csatlakozási pontot az átfedőhöz. A rendszernapló fájlok vizsgálatát minél hamarabb el kell kezdeni, hogy a támadások észlelése elkezdődhessen. Ha az egyednek még nem sikerült csatlakoznia az átfedő egyetlen tagjához sem, a tapasztalatait akkor is rögzíti az adatbázisában. Legalább egy kapcsolat felépülése után a tapasztalatok már közvetíthetőek a többi csomópont felé.

A Komondor hatékonysága

A *Komondor* rendszer *hatékonysága* elsődlegesen az azt felépítő egyedek változatosságán múlik. A biztonsági



3. ábra Alhálózatok védelme Komondor hangolt tűzfalakkal

réseket kihasználó támadások szoftver-, illetve verzióspecifikusak.
A különböző szolgáltatásokat nyújtó hálózati programok (például *sshd*, web kiszolgáló) más-más verziói, fajtái lehetnek telepítve az egyes egyedeken. Az észlelt és remélhetőleg kivédett támadás után a *Komondor* segítségével a sebezhető kiszolgálók is megvédhetőek lehetnek. Ez három gyakran előforduló helyzetben következhet be:

- a kiszolgálók ugyanannak a szoftvernek különböző változatait futtatják (például *Apache 2.0.54* és *Apache 2.0.30*).
- a kiszolgálók ugyanazt a szolgáltatást különböző programokkal valósítják meg (webkiszolgálóként használható például az *Apache* vagy a *Zeus* is);
- a kiszolgálók más műveleti rendszerekre épülnek. (például *Linux* és *Windows*)

Az első esetben, amikor a kiszolgálók ugyanannak a programnak eltérő változatait futtatják, a sebezhetőségi pontok az egyes gazdagépeknél hasonlóak lehetnek, ugyanis a hibák azok felderítéséig a programokban rendszerint verzióról verzióra öröklődnek. A második és harmadik esetben viszont az egyes szolgáltatások sebezhetőségi pontjai általában diszjunktak, ami miatt az egyes biztonsági rések elfedése hatékonyabb lehet.
A *Komondor* hatásfokát korlátozza az *IP* címek nagy száma. A jelenleg használt protokoll, az *IPV4* esetén a címtartomány 2^{32} méretű, *IPV6* esetén ennél is jóval nagyobb (128 bites hálózati címek). Annak valószínűsége, hogy egymástól távol lévő *IP* számú gazdagépeket rövid időn belül ugyanaz a támadó vesz célba, rendkívül alacsony. Ahogyan korábban már említésre került, gyakori viszont az egymáshoz közeli címek rövid időn belül, sorrendben történő vizsgálata, amikor a támadó egy adott tartományt pásztáz végig célpontok után kutatva. Érdemes emiatt a *Komondort* használó rendszergazdának úgy szerveznie a hálózatot, hogy a tartomány első gazdagépe, a fenti példában a *194.143.224.1* című, különösen biztonságos legyen (kevés, szokatlan azonosítóval rendelkező felhasználó, bonyolult jelszavak, kevés szolgáltatás stb.). A tartomány támadó célú átvizsgálása ennél a gépnél kezdődik, és a védelem így a lehető leghamarabb reagálhat. A támadó az alhálózati tartomány átvizsgálásakor kétféleképpen járhat el:

- Felderítheti, hogy mely gépeken fut az általa támadni kívánt szolgáltatás, utána próbálva kihasználni a hibákat.
- Fordítva, a hibák vizsgálatát (például a belépési kísérlet gyenge jelszavakkal) az *IP* számok szerint haladva sorban elvégzi.

Mind a két támadói magatartás eseté egy *Komondor* egyednek valós esélye van a támadó hálózati címének „szétkürtölésére”, mivel hasonló *IP*

címek esetén valószínűleg helyi hálózaton dolgozik a többi *Komondor* egyeddel, vagyis közöttük az összeköttetés fizikai szinten gyorsabb, mint a távoli támadó és az egyes helyi hálózatbeli gazdagépek között.

Biztonság és robusztusság

A program legnagyobb hátránya, hogy az átfedőbe sajnos könnyen rosszakaratú egyed épülhet. Egy ilyen a legnagyobb kárt hamis tapasztalatok megosztásával tud okozni, ezek ugyanis *szolgálatmegtagadás*t (*DoS*) okoznak. A biztonság és hatékonyság között a megszokott módon csak kompromisszum lehetséges. Megoldás lehet ez ellen az átfedőbe csatlakozó egyedek felhasználói jogainak vizsgálata (*ident vizsgálat*). Ehhez felhasználhatjuk azt a tényt, hogy a *Komondor* csak rendszergazda jogosultságokkal futhat az egyes egyedek gazdagépein, különben nem tudna naplófájlokat vizsgálni és a hálózatba beavatkozni. Vagyis felhasználói jogosultságokkal rendelkező rosszakaratú egyén nem tudná futtatni a *Komondor* szoftvert. Ha azonban egy *Komondor* mégis felhasználói jogokkal fut egy gazdagépen, akkor az valószínűleg hamis tapasztalatokat hirdet, vagy más módon zavarja a rendszer integritását. Sajnos az *ident* szolgáltatás általában csak *UNIX* típusú rendszereken ismert. *Windows* platformon rendszerint meghamisított *ident* kiszolgáló fut valamilyen külső elvárás miatt.

A rendszer kritikus pontja a horgonypont. Ennek kiesése esetén új egyed nem képes kapcsolódni az átfedőhöz. A már kapcsolódott egyedek között a tapasztalatcsere megmarad a *P2P* elvnek köszönhetően.

A továbbfejlesztés irányai

Ebben a fejezetben kerül bemutatásra a *Komondor* rendszerének lehetséges továbbfejlesztési irányai, melyeket az előbbiekben alkalmazott szempontok szerint értékelünk.

Az *alhálózatok védelméhez* megfontolandó a *Komondor* átfedő hálózatának részben központosított kialakítása, hasonlóan a *FastTrack* hálózati modellhez. A *FastTrack* eredeti célja ugyan a méretezhetőség javítása volt, azonban a következőkben ismertetésre kerülő megfontolások alapján szintén a részben központosított átfedő látszik célszerűnek. Tudjuk, hogy hálózatok, géptermek kialakításakor megszokott módszer, hogy az egyes hálózati szelvények munkaállomásaihoz közös tűzfal tartozik. Mivel ilyenkor a munkaállomások általában erre támaszkodnak, önmaguk nem látnak el tűzfal feladatkört. A védelmet nyújtó *Komondor*nak mindenképpen a tűzfal számítógépén kell futnia. A munkaállomásokon futó *Komondor* egyedeknek a feladata ekkor a betolakodás-észlelésre korlátozódik. A megszerzett tapasztalatokat, amelyekről adatbázist sem szükséges vezetniük, azokat mindegyik a saját tűzfalán futó felsőrangújához továbbítja. Természetesen a felsőrangúak is észlelhetnek betolakodási kísérletet.

A *P2P* elv ebben az esetben ugyanúgy alkalmazható, mint eddig. A tűzfalak között jön létre ekkor a *P2P* összeköttetés, amely lehet teljes gráf is, azaz mindegyik mindegyikhez, mivel a forgalom még egy adott tapasztalat száz másik gép felé történő megosztásakor is kilobájt nagyságrendű csupán. A már ismertetett címtartomány pásztázás miatt pedig itt szomszédos alhálózatokat védhetnénk hatásosan, ami úgyszintén kis számú felsőrangút feltételez.

Ez a fajta védelem igen hatásos lehet, azonban a hálózati/szállítási réteg közötti beavatkozásra korlátozódik. A munkaállomások *Komondor* egyedei egy egyszerűbb programot futtatnak, mint a felsőrangúak. A *P2P* hálózatba szerveződés helyett ezek a hálózati átjáró (gateway) címét kérdeznék le az operációs rendszertől, amely értelemszerűen megegyezik a tűzfal címével, ahol a felsőrangú egyed fut. Az építménynek akkor is van értelme, ha az egyes tűzfal mögötti gazdagépek *közös hálózati címen osztoznak*.

Ekkor a *hálózati címfordítás*t (*Network Address Translation, NAT*) végző útválasztó a kapcsolat kapuszáma szerint más-más belső ponthoz továbbít csomagokat. Ezek a csomagok eredetileg hozzá érkeznek, de a szolgáltatásokat érő támadásokat csak az azokat megvalósító gazdagépek képesek felderíteni. Ennek az oka például az, hogy az útválasztónak nincs arról tudomása, hogy egy adott felhasználói név egy belső gépen létezik-e.

A betolakodás-észlelés további módszerei

A *Komondor* megvalósított változata bővíthető további tapasztalatszerző módszerekkel. Betolakodási kísérletek észleléséhez lehetséges például egy olyan program modul létrehozása, amely egy web *kiszolgálót imitál*, azaz tényleges szolgáltatásokat nem nyújt, csak a bejövő *HTTP* kérések vizsgálatával próbál gyanús jeleket gyűjteni. A támadó számára megtévesztő lehet, ha az igazi web kiszolgálót egy szokatlan hálózati kapura (port) telepítjük, például a 8080-asra, míg a szokásos 80-as kapun a *Komondor* imitált web kiszolgálója fut. Egy későbbi windowsos változatban erre a célra felhasználható lehetne az ott egyébként nem használt 22-es (*SSH*) vagy 23-as (*telnet*) kapu. Ezt a modult nem szükséges közvetlenül a *Komondor*ba építeni, futhat külön folyamatként is, naplófájlt vezetve.

Az előbbiekben ismertetett saját tapasztalatszerzési eljárás mellett számítani lehet arra, hogy a *Snort* közösség egy mindig naprakész felderítő adatbázissal rendelkezik, így a fejlesztés során a hangsúlyt a saját tapasztalatszerzés helyett a *Komondor* egyedek P2P hálózatba szerveződésének javítására érdemes fektetni.

Czirkos Zoltán

Jelenleg diplomatervező a Budapesti Műszaki Egyetem Elektronikus Eszközök Tanszékén. Kutatási területe az operációs rendszerek betörésvédelme és a P2P kommunikáció. 2005-ben a Tudományos Diákköri Konferencián II. helyezést ért el. Kedvencei a boszorkányos és a rózsaszín párducos filmek.

KAPCSOLÓDÓ CÍMEK

Komondor tesztváltozat:
➲ http://jutas.eet.bme.hu/

Snort:
➲ http://www.snort.org

RealSecure:
➲ http://www.iss.net

Swatch:
➲ http://swatch.sourceforge.net

Nmap:
➲ http://www.insecure.org

Gnutella:
➲ http://www.gnutella.org

Hollywood parancssorból

mencoder

A „legjobb linuxos videólejátszó” és számos más cím sokszoros győztese, az MPlayer nevű lejátszóprogram csak fele a csomagnak, ami az mplayerhq.hu címről letölthető. A másik, talán még nagyobb tudású eszköz a mencoder, mely hihetetlenül sokrétű, de parancssoros felülete sokakat elriaszt a behatóbb tanulmányozástól. Ezen szeretne ez a cikk segíteni.

Kedvcsinálóként nézzük meg, mire is használható a ***mencoder***:

- Hibás indexű fájlok javítása
- Filmek összefűzése, vágása, átméretezése
- Különféle videóformátumok közötti konverziók
- *DVD*-rip készítése
- Tv-adások felvétele
- Különálló képekből film létrehozása
- Rengeteg szűrő (például zajcsökkentés, logó eltávolítása) alkalmazásának lehetősége
- és még rengeteg egyéb funkció...

Áttekintés

A ***mencoder*** használatához szükséges egy vagy több forrásfájl, ami(ke)t felhasználva/átalakítva hozunk létre egy célfájlt. A forrásfájl – mely nem feltétlenül jelent a szó szoros értelmében vett fájlt, hanem lehet a szabványos bemenetre csővezeték segítségével érkező adatfolyam vagy akár a tv-tuner kártya is – alapvetően két fontos folyamot tartalmaz: az audio és a video részt.

Mindkettőről el kell mondanunk a ***mencodernek***, hogy mit szeretnék vele csinálni. Ez jelentheti valamilyen audio/videótömörítő (***codec***) alkalmazását vagy akár közönséges másolást. Codec használatakor rengeteg paramétert megadhatunk, melyek azonban nagyrészt elhagyhatók, ilyenkor az alapértelmezett beállításokat veszi figyelembe a program. Egy általános `mencoder` parancssor a következőképpen néz ki:

```
mencoder -oac <audio codec>
➥-ovc <video codec> [<codec
➥paraméterek>] <bemeneti
➥fájl(ok)> -o <kimeneti fájl>
```

A használható audio- és videocodecek listáját lekérhetjük a

```
mencoder -oac help
```

illetve a

```
mencoder -ovc help
```

parancsokkal.

Egyszerű feladatok másolással

Index javítása

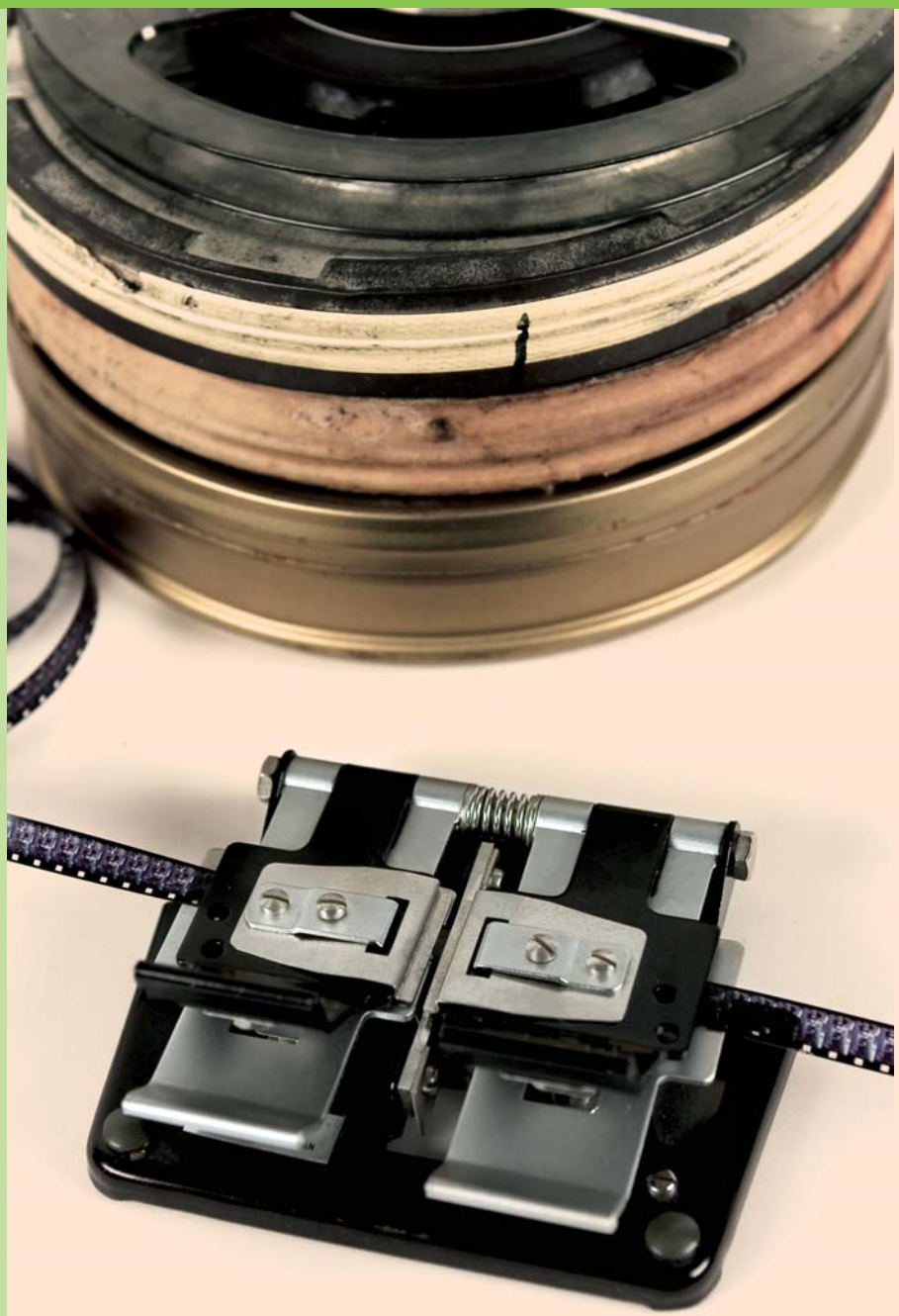
Tegyük fel, hogy (természetesen legálisan...) letöltöttünk az internetről egy filmet, ami azonban hibás indexet tartalmaz (például azért, mert nem sikerült teljesen letölteni), vagy a *CD*, amin tároltuk megsérült, és így nem tudunk benne előre-hátra tekerni. A ***mencoder*** segítségével egyszerűen átmásoljuk a hibás fájl videó- és audióanyagát egy új, jól működőbe, amihez újrageneráljuk az indextáblát:

```
mencoder -oac copy -ovc copy
➥-idx rossz.avi -o jo.avi
```

Ebből már ki is derül, hogy ha nem akarjuk a fájlt újratömöríteni, csak másolni, a `copy` kulcsszót kell a codec neve helyett írnunk.

Több fájl összefűzése

Kihasználva azt, hogy a `mencoder` több fájlt is képes forrásként használni, egyszerűen átmásoljuk őket egy újba:



```
mencoder -oac copy -ovc copy
➥film1.avi film2.avi -o
➥osszefuzve.avi
```

Esetleg szükség lehet a `-noidx` és/vagy `-forceidx` kapcsolókra, hogy a keletkező fájl indexe ne legyen hibás. Az összefűzés csak azonos felbontású és azonos codec-kel tömörített fájlok esetén működik!

Vágás

Nem csak egy fájl elejétől a végéig dolgozhatunk a `mencoder`-rel, hanem megadott kezdőponttól kiindulva adott hosszúságú részt is használhatunk. A kezdőpontot a `-ss` kapcsolóval, a hosszt pedig a `-endpos` segítségével adhatjuk meg. Például a forrásfájl ötödik percétől másfél perc hosszúságú rész kivágása:

```
mencoder -ss 5:00 -endpos 1:30
➥-oac copy -ovc copy
➥forras.avi -o vagott.avi
```

Átalakítás más formátumba

Természetesen a `mencoder` nem csak másolni képes, hanem rengetegféle formátumba tömöríteni is tud.
A videóanyagra leggyakrabban valamilyen Mpeg4 formátumot szoktunk használni (ilyenek például az ismert *XviD*, *DivX* codec-ek). A `mencoder` által favorizált codec az *MPlayer* forrásában is megtalálható *libavcodec*, a példákban is ezt fogjuk használni.
A parancssor nagyon hasonlít a másoláskor használtakra, de kibővítjük még más kapcsolókkal is, és természetesen nem a `copy` paramétert adjuk meg a `-oac` és `-ovc` kapcsolóknak.
Kezdjünk egy egyszerű konvertálással, ahogy egy bármilyen fájlból (például egy tunerkártyáról előzőleg felvett, kevéssé tömörített anyagból) készítünk *MPEG-4* kép- és *MP3* hangtömörítéssel rendelkező *avi* fájlt:

```
mencoder -oac lavc -ovc lavc
➥-lavcopts vcodec=mpeg4:
➥acodec=mp3 forras.avi -o
➥cel.avi
```

Audio- és videócodecnek egyaránt a már említett *libavcodec*-et (a `mencoder` rövidítése szerint `lavc`) választottuk. A *libavcodec*-nek paramétereket a `-lavcopts` kapcsoló után, egymástól kettősponttal elválasztva adhatunk át. A fenti példában csak kettő ilyet láthatunk: a keletkező fájl video (`vcodec`) és audio (`acodec`) formátumát adtuk meg. Nagyon sok paramétert sorolhatunk még fel, de ha nem tesszük, az alapértelmezett értékeket veszik fel.
Ha a legjobb képminőségre törekszünk, érdemes még néhány paraméterrel megtoldani a parancssort (ezek magyarázatát lásd a keretes részben):

```
mencoder -oac lavc -ovc lavc
➥-lavcopts vcodec=mpeg4:
➥acodec=mp3 :vbitrate=1800:
➥abitrate=128:mbd=2:trell:
➥v4mv:qpel forras.avi -o
➥cel.avi
```

Erre még rátehetünk egy (sőt, akár kettő!) lapáttal, ha dupla időt szánunk a tömörítésre, és két menetben tesszük ezt. Ezt egyszerűen elérhetjük, csak adjuk a `lavcopts` részhez a `vpass=1` paramétert (ez lesz az első menethez tartozó parancssor), majd miután a kódolás elkészül, az egyest írjuk át kettesre és futtassuk ismét.
Az első menetben készül egy `divx2pass.log` nevű fájl, amit a második menetben a codec felhasznál hogy jobb minőséget érhessen el, ezt a második menet után letörölhetjük.
Egy kétmenetes tömörítés parancssorai tehát:

```
mencoder -oac lavc -ovc lavc
➥-lavcopts vcodec=mpeg4:
```

```
➥acodec=mp3 :vbitrate=1800:
➥abitrate=128:mbd=2:trell:
➥v4mv:qpel:vpass=1  forras.avi
➥-o  cel.avi
mencoder -oac  lavc  -ovc  lavc
➥-lavcopts vcodec=mpeg4:
➥acodec=mp3 :vbitrate=1800:
➥abitrate=128:mbd=2:trell:
➥v4mv:qpel:vpass=2  forras.avi
➥-o  cel.avi
```

A forrás megadása

Kedvenc ***mencoderünk*** nem csak fájlokat képes beolvasni, használhatunk *DVD*-t, tv-tunert vagy akár különálló képeket is film létrehozásához.

DVD

A `dvd://` „protokollt” és a tömörítendő sáv számát kell megadnunk. Az első sáv tömörítése *AVI*-ba például így fest:

```
mencoder -oac lavc -ovc lavc
➥dvd://1 -o cel.avi
```

Ha több *DVD*-olvasónk is van, a `-dvd-device` kapcsolóval jelölhetjük ki a megfelelőt:

```
-dvd-device /dev/hdc
```

TV-tuner

A bemeneti fájl helyett a `tv://` formával jelezhetjük hogy a tunerkártyáról fog érkezni a tömörítendő anyag. Ilyenkor mindenképp adjunk meg néhány fontos paramétert, például a csatorna számát vagy frekvenciáját, amiről fel szeretnénk venni, illetve szükség lehet a meghajtó (`driver`) megadására (rendszerint `v4l` vagy `v4l2`), és többnyire előre megadjuk azt is, hogy mekkora felbontású legyen a film. Egy „mencoder-nyelvű” példa:

```
mencoder tv:// -tv driver=v4l2:
➥channel=21:width=384:height=
➥288 -oac lavc -ovc lavc -o
➥tv.avi
```

A `channel` paraméterrel adhatjuk meg a használandó csatorna kódját. Ehelyett állhat a csatorna frekvenciája is, a következő formában:

```
freq=583.250
```

Ha 384x288-nál nagyobb felbontásban szeretnénk rögzíteni a képanyagot, ***deinterlacert*** kell használnunk, ellenkező esetben mozgás közben „fésűs” lesz a kép. Ehhez írjuk a parancssorhoz a következő varázsszavakat:

```
-vf pp=fd
```

A részletes magyarázatot a szűrők leírásánál adjuk meg.

Különálló képek

Készíthetünk filmet több állóképből is, amihez természetesen hanganyag is illeszthető. Így például hangulatos „kisfilmet” állíthatunk össze családi fényképekből. Íme egy példa:

```
mencoder mf://*.jpg -mf
➥type=jpeg:fps=1:w=640:h=480
➥-ovc lavc -oac copy -
➥audiofile zene.mp3  -o
➥kesz.avi
```

Az `mf://` protokollal jelezzük hogy képfájlokat használunk forrásként, a két `/` után írjuk a fájlmaszkot vagy egymástól vesszővel elválasztva a fájlneveket. A `-mf` kapcsoló paraméterei között szerepeltethetjük a képfájlok típusát (`type`), a kimeneti videófájl szélességét (`w`) és magasságát (`h`) valamint a másodpercenkénti képkockák számát (`fps`). Ha a `-mf` után átadott méretezési paraméterekre nem reagál a `mencoder`, használhatjuk az átméretező szűrőt (lásd később).

Szűrők

A `mencoder` döbbenetes mennyiségű szűrő használatával képes átalakítani a forrásként használt videóanyagot. A szűrők használatához a `-vf` kapcsolót kell megadnunk, paraméterként pedig a használandó szűrő nevét, illetve szükség esetén a szűrőnek átadott opciókat (ha több is van, egymástól kettősponttal elválasztva). Mivel a szűrők átalakításokat végeznek a filmen, nem elég csak másolni a képanyagot, mindenképp valamilyen codecet kell használnunk. Nézzük át a legfontosabb illetve leggyakrabban használt szűrőket!

Vágás

Most más vágásról van szó, mint pár bekezdéssel feljebb amikor a film egy részét használtuk csak fel. *DVD*-ről származó filmnél vagy néha tv-felvételeknél előfordul a kép alsó és felső részén széles fekete csík, amikre egyrészt nem vagyunk kíváncsiak, másrészt a tömörítés hatásfokát és a kódolt videó minőségét is rontja. A megoldás az, hogy a képnek csak azt a részét használjuk, amin a számunkra értékes képanyag van. A `mencoder` szűrői között a `crop`-ra lesz szükségünk, melynek megadhatjuk a használni kívánt képrészlet szélességét, magasságát és a bal felső sarkát. Ha nem akarunk tippelgetéssel eltölteni rengeteg időt, használhatjuk az `mplayer`-t a `cropdetect` szűrővel:

```
mplayer -vf cropdetect film.avi
```

Ennek eredményeképpen az `mplayer` a terminálra írja a használandó paramétereket, így azokat egyszerűen csak be kell másolnunk a `mencoder` parancssorába.

1. táblázat *A DVD illetve VCD készítésekor használható felbontások*

	Felbontás	Bitráta [kbit/s]
VCD	352x288	1152
SVCD	480x576	2500
DVD	720x576 704x576 352x576 352x288	max. 9800

2. táblázat *A lavcopts után megadható paraméterek*

	VCD	SVCD	DVD
keyint	25	25	25
vrc_buf_size	327	917	1835
vrc_minrate	1152	elhagyható	elhagyható
vrc_maxrate	1152	2500	9800
vbitrate	1152	max. 2500	max. 9800

Egy példa:

```
mencoder -oac lavc -ovc lavc
➥-vf crop=464:272:8:34
➥film.avi -o vagott.avi
```

Átméretezés

DVD-ről történő tömörítéskor a torzult képarány miatt célszerű átméretezni a képet. Tunerkártyáról digitalizálva többnyire nem 768x576-os felbontást használunk, hanem kisebbre vesszük a képméretet. Mindkét esetben a `scale` filterrel lesz szükség, aminek a kívánt szélességet és magasságot kell átadnunk:

```
mencoder -oac lavc -ovc lavc
➥-vf scale=640:480  film.avi
➥-o atmeretezett.avi
```

Deinterlacing

Ha *DVD* vagy tv-tuner a forrásunk, szükség lehet a váltottsoros formátum (interlacing) miatt megjelenő „fésűhatás” eltüntetésére. A `mencoder` több úgynevezett ***deinterlacer szűrőt*** is tartalmaz, melyek használatához a `pp` nevű szűrőnek kell átadnunk a ***deinterlacer*** típusát. itt most egyet említünk meg: az `fd`-t (mivel hatalmas hitviták tudnak kirobbanni a „legjobb” ***deinterlacer*** keresése közben, ettől most eltekintünk, de biztatjuk a kedves Olvasót a lehetőségek kipróbálására és az eredmény összehasonlítására – az alkalmazható paramétereket megtaláljuk a mencoder man oldalán).

Egy példa:

```
mencoder tv:// -tv driver=
➥v4l2:channel=21:width=768:
➥height=576 -oac lavc -ovc
➥lavc -vf pp=fd -o
➥felvetel.avi
```

Zajszűrés

Leginkább tv-felvételeknél vagy *VHS*-ről digitalizált anyagnál lesz szükségünk zajszűrő használatára hogy a képen megjelenő zavart mérsékeljük vagy eltüntessük. A zajszűrő használat talán nem szükséges túl hosszan ecsetelni: jobb képminőség, jobb tömöríthetőség. Azonban nem árt tisztában lenni az esetleges hátrányokkal is: a képélesség csökken, kissé ‘elmosódott’ lesz a kép, a tömörítési idő jelentősen megnő. Mindazonáltal a paraméterek helyes megválasztásával igazán szép eredményeket érhetünk el. A legjobb minőségű zajszűrő a mencoderben a `hqdn3d` nevű, ennek egy valamivel gyengébb eredményt produkáló de érezhetően gyorsabb változata a `denoise3d` nevet viseli. Mindegyikük három paramétert vár, ezeknek az alapértelmezettől (4:3:6) eltérő megadásával szabályozhatjuk a szűrő erősségét (nagyobb szám erősebb zajszűrést jelent). Íme egy példa:

```
mencoder -oac copy -ovc lavc
➥-vf hqdn3d=3:2:4 zajos.avi
➥-o szurt.avi
```

Logó eltávolítása

Szintén tv-felvételeknél lehet elsősorban hasznos a képernyő minden sarkát elfoglaló többnyire kevéssé esztétikus logók eltüntetése. Természetesen erre tökéletes megoldás nincs, de bizonyos esetekben (főleg az átlátszatlan, téglalap alakú logóknál) sokkal jobb lehet a kozmetikázás utáni végeredmény mint a logókkal kidekorált felvétel. A `mencoder` erre a feladatra a `delogo` nevű szűrőt kínálja, melynek meg kell adnunk a logó által a filmből elfoglalt terület bal felső pontjának koordinátáit és a vízszintes illetve függőleges méreteit, valamint egy értéket arra vonatkozóan, hogy a kijelölt terület körül hány pixel vastagságú sávot vegyen figyelembe a logó eltávolításánál – a szűrő ugyanis úgy működik, hogy a logó körüli területekből próbál következtetni a letakart részek tartalmára. Mivel egyszerű megoldás nincs a logó pozíciójának automatikus megállapítására, használhatjuk (sőt: használnunk kell) azt a funkciót, hogy az utolsó paramétert -1-nek választva egy zöld téglalap lesz látható a koordináták által kijelölt terület körül, így néhány próbálgatásból egész szépen megtalálhatjuk a kívánt pozíciót.
Egy példa a *TV2* logó eltávolítására 480x352-es felbontású felvételről:

```
mencoder -oac copy -ovc lavc
➥-vf delogo=392:32:50:30:
➥6 felvetel.avi -o
➥logo_nelkul.avi
```

VCD/DVD készítése

VCD/*SVCD*/*DVD* formátumú filmek készítése önmagában is megérne egy cikket, így itt csak nagyon rövid ízelítőt adunk a témából. Az első amit tisztáznunk kell, hogy a `mencoder` csak a *MPEG* formátumú filmfájlt fogja létrehozni, nem egy *CD* képfájlt vagy *DVD*-re írható könyvtárszerkezetet, erre más programokat kell igénybe vennünk.
A létrehozandó fájlra nagyon szigorúak a megkötések. *DVD* készítésekor mindössze 4 felbontás közül választhatunk (ráadásul ebből a gyakorlatban inkább csak az egyik használatos), *VCD* és *SVCD*-re szánt anyagnál pedig csak egyféle felbontást használhatunk (*1. táblázat*).
Ráadásul a képarány sem lehet akármilyen: 4:3 arányt használjunk, ahogy a *TV* is, egyedül a *DVD* formátum engedi meg 16/9-es képarány használatát

3. táblázat *Néhány fontosabb/gyakran használt libavcodec paraméter*

`acodec`	A hanganyag formátuma. Alapértelmezés szerint `mp2`.
`abitrate`	A hanganyag bitrátája kbit/sec-ban, alapértelmezésben 224.
`vcodec`	A videó formátuma, ha nem adjuk meg, MPEG-4-et használ a libavcodec.
`vbitrate`	A videó bitrátája, alapértelmezés szerint 800 kbit/sec.
`mbd`	A tömörítés minőségét befolyásoló kapcsoló, alapértelmezett értéke 0; nagyobb értékkel (1 vagy 2 (legjobb) választható) a képminőség jobb lesz, de a tömörítés sebessége csökken.
`v4mv`	Több mozgásvektor használata; a képminőséget javítja, de a tömörítés sebességét csökkenti. Néhány asztali mpeg4-et ismerő (hipermarket-terminológiával élve "DivX-es") lejátszó nem szereti, ha ilyen eszközön is meg szeretnénk nézni a filmet, legyünk óvatosak! Használatához nem kell értéket megadnunk, csak felsorolni a paraméterek között. (például `-lavcopts vcodec=mpeg4:v4mv:mbd=2`)
`qpel`	Javított mozgásbecslés; használata nem csak a tömörítést hanem a lejátszást is lassítja valamennyire. A v4mv-nél leírtak erre a paraméterre is érvényesek!
`trell`	A képminőséget befolyásoló paraméter, a többi ilyenhez hasonlóan lassítja a tömörítés sebességét.
`vpass`	Többmenetes tömörítés esetén kell megadni, az első menetben egyes, a másodikban kettes értékkel

720x576-os felbontás mellett.
Ezt a `lavcopts`-nak átadott `aspect=4/3` vagy `aspect=16/9` paraméterekkel állíthatjuk be.
Az elfogadott hangformátumokból sem nagy a választék: *VCD*-nél 224, *SVCD*-nél 384 kbit/sec bitrátájú *MP2* 48000 Hz-es mintavételezési frekvenciával, *DVD*-n *MP2*, *AC3* vagy esetleg „sima" *PCM* 44100 Hz-cel. A frekvencia helyes beállításához használjuk a

```
-srate <frekvencia>
```

és

```
-af lavcresample=<frekvencia>
```

kapcsolókat.
Egy eddig nem használt kapcsoló a `-of`, mely a kimeneti fájl úgynevezett konténer formátumát adja meg, ebből nekünk a `-of mpeg` paraméterre lesz szükségünk, valamint a `-mpegopts` kapcsolóra `format=xvcd`, `xsvcd` vagy `xdvd` paraméterrel.
Talán a legnagyobb falat azonban a `lavcopts` kapcsoló helyes paraméterezése. Az audio formátumokról már esett szó (az `acodec` és `abitrate` paraméterekre figyeljünk!). A használandó videócodec *VCD* esetén `mpeg1video`, *SVCD*-nél és *DVD*-nél `mpeg2video`. A problémák elkerülése érdekében a *2. táblázatban* összefoglaltam a a `lavcopts`-nak megadható paramétereket.
A már megismert minőségjavító kapcsolók közül a `trell` és az `mbd=2` használata ajánlott, a `qpel`-t és a `v4mv`-t azonban ne használjuk *(S)VCD* és *DVD* készítésekor!
Lássunk egy példát DVD készítésére:

```
mencoder -oac lavc -ovc lavc
➥-of mpeg -mpegopts format=dvd
➥-vf scale=720x576 -srate
➥48000 -af  lavcresample=48000
➥-lavcopts vcodec=mpeg2video:
➥keyint=25:vrc_buf_size=1835:
➥vrc_maxrate=9800:vbitrate=
➥4500:acodec=ac3:abitrate=
➥192:aspect=16/9 -ofps 25
➥film.avi -o dvd.mpg
```

Ha valami nem működik...

A `mencoder` fejlesztése során több változás is történt a program működésében illetve paraméterezésében. Ha valamelyik kapcsolót nem ismeri fel a `mencoder`, töltsük le a legújabb változatot a ➲ *http://www.mplayerhq.hu* weboldalról. Addig is a legfontosabb dolgok amik gondot okozhatnak egy régebbi verzió használatakor:

A `-vf` helyett a régi verziók a `-vop` kapcsolót használták
a `dvd://` protokoll régebben `-dvd`-ként szerepelt (hasonlóan változott a `tv://` használata is)

Abból is adódhat azonban problémánk, ha az új `mencoder`-t használjuk, ugyanis a *FourCC*-nek nevezett négykarakteres azonosítókód, ami a lejátszóprogram számára ad információt a használandó videócodecről, alapértelmezés szerint *FMP4*-re változott, amit azonban sok lejátszó (ez lényegében az `mplayer`-en kívül az összes többire vonatkozik...) nem tud értelmezni, ezért nem tudja lejátszani. A megoldás az, hogy explicit módon megadjuk a használandó kódot. Ez lehet *DIVX*, *DX50* vagy *XVID* a leggyakrabban használt *libavcodec* esetében. Az `mplayer` dokumentációja az utolsót ajánlja. A gyakorlatban ez a `-ffourcc XVID` paraméter megadását jelenti.
Ezen cikk leginkább csak kedvcsináló, gyors áttekintés a `mencoder` adta lehetőségekről, aki szeretne komolyabban foglalkozni vele, annak mindenképp ajánlott a forráscsomagban és a `mencoder` honlapján egyaránt fellelhető dokumentáció tanulmányozása! Egy másik – főleg kezdőknek szóló – segítség a ➲ *http://bokorn.uw.hu/linux/dmencoder* címen található parancssor-generátor, mely a leggyakrabban használt kapcsolókból állít össze az igényeinknek megfelelő parancsokat; így nemcsak pillanatok alatt állíthatjuk elő a szükséges parancssort, hanem a tanulmányozásával jobban meg is ismerhetjük az egyes kapcsolók, paraméterek használatát.

Bokor Norbert
(doc@coder.hu)
Egy autóipari cégnél informatikus, emellett Győrbe, a Széchenyi Egyetemre jár. A számítógép mellett imádja a társasjátékokat. Most éppen gitározni tanul.


Linuxvilág
www.linuxvilag.hu

Taktikai küzdelem – valós idejű stratégia Linux alatt

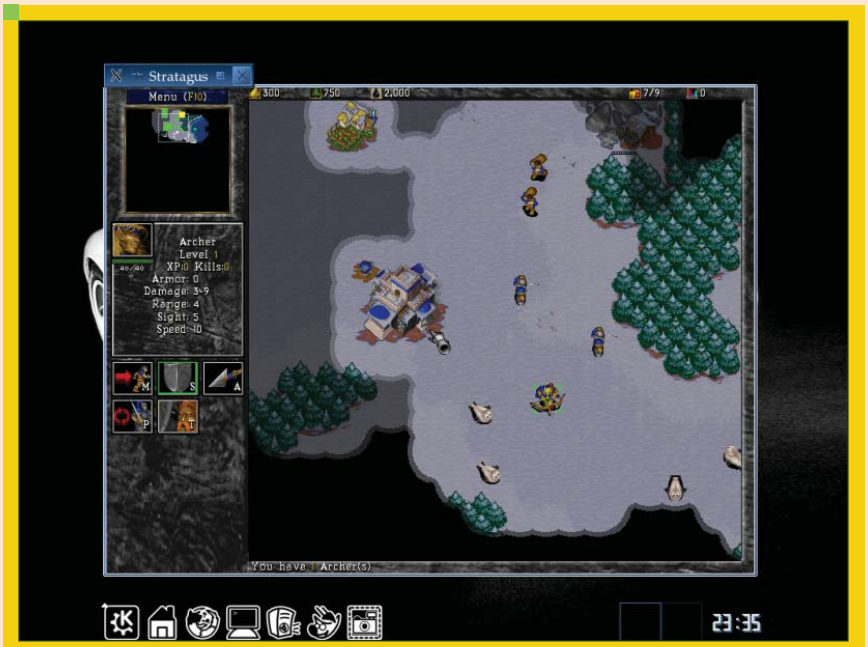
A belső nézetű, lövöldözős játékstílus mellett leginkább a stratégiai vonal képes tömegeket csalni a számítógép elé. Írásunkban három igen kedvelt játék kerül terítékre.

WarCraft

Kezdjük elsőként a *WarCraft2* reinkarnációjával! A régmúlt időket idéző, nagynevű *RTS* (*Real Time Strategy*) a *Blizzard* csapatának munkája, mely csapat sikerprogramjai által páratlan sikereket könyvelhet el a *PC*-s játékipar területén. Első projektjük (*Lost Vikings*) óta jellemzően izometrikus nézetű kaland / szerepjáték hibrideket, illetve valós idejű stratégiákat adnak ki kezeik közül. Linuxos támogatással nem szolgálnak, csupán *Win32* platformra fejlesztenek, így játékaik kizárólag *Wine* / *Cedega* / *CrossOver* wrapperek segítségével futnak *Linux* alatt.

Egyik projektjük azonban „kilóg a sorból": a *WarCraft* sorozat második része gyakorlatilag natívan is életre hívható *Linux* alatt. A program két módszerrel lenne feléleszthető, viszont a *Blizzard* az egyik utat bezárta, mivel sértve érezte saját érdekeit. Ez a bizonyos problémás megoldás a *FreeCraft*, amely kimondva, kimondatlanul a *WarCraft2* adatainak (képek, hangok, játékszabályok) felhasználására volt kihegyezve. Így a második lehetőséget ismertetem, mely által egy kiforrott játékmotorra feszítjük rá az eredeti játék adatcsomagjai.

A világháló ➲ *http://www.stratagus.org* címén elérhető *Stratagus* játékmotor a szabadon elérhető kódok egyik iskolapéldáját testesíti meg. Jó néhány próbálkozás szíveként dobog, jellemzően *2D* izometrikus nézetű stratégiai játékok alapját adva. A játékmotorhoz



1. ábra A Stratagus / Wargus megoldás, KDE asztalon

tartozó adatállományokat kötött szabályok szerint kell felépítenünk kezdve a könyvtárstruktúrától egészen a fájlnevekig. A *WarCraft2* adatainak ilyen irányú felhasználásához előbb fel kell nyitnunk azokat: a *Wargus* projekt rendelkezik erre a célra szolgáló kóddal (linkje az említett oldalon található), mely segítségével az eredeti *Warcraft2* játéklemezről kinyert adatokat könnyen a szükséges formára alakíthatjuk.

Csupán a *readme* állomány szerint kell eljárnunk: a *Wargus* forrás tarballját kicsomagoljuk valahová, majd lefuttatjuk rootként a benne lévő `make` állományt (ami által létrejön egy *wartool* file). A *Win32* játéklemez (*Battle Net Edition* verzió nem megfelelő!) befűzése után a diszken lévő */DATA* mappát (kisbetűsre alakítva) a kibontott *Wargus* útjára kell másolnunk, majd annak `build.sh` szkriptjét le kell futtatnunk. Némi várakozás után erősen megváltozik az említett könyvtár tartalma, melyet ebben az állapotában egy tetszőleges útra kicsomagolt *Stratagus* motor binárisa mellé kell mozgatnunk.

Ezután a `stratagus` állományt indítva a szabad játékmotor – saját szabályait alkalmazva – a *WarCraft2* képi világát

2. ábra A Homeworld csodaszép világa

kelti életre. Futtatás idejére már nem szükséges a *Win32* játéklemez jelenléte hacsak nem szeretné valaki hallgatni a zenekari audiosávokat. Természetesen az engine elérhető forrásként is, így akinek nem tetszik a bináris megoldás, az le is fordíthatja magának a kódot (`./configure, make depend, make`) – viszont az említett telepítési mód kellőképpen egyszerű és hibalehetőségektől mentes. A játék kellemesen nehéz, kicsi hardverigényű, grafikája „2D retro art" jellegű. Annak ellenére, hogy nem a *Blizzard* hivatalos megvalósításáról van szó, mindenképpen linuxos gépünkön a helye!

Homeworld

Második „versenyzőnk" a *Homeworld* című alkotás, mely az 1999-es év díjazott játéka. A ➲ *http://www.relic.com* címen elérhető *Relic Entertainment* űrstratégiája igen érdekes történettel áll elő: távoli jövőnkben az emberiség felfedez egy lezuhant űrhajót, melyről kiderül: sok ezer éve szenvedett hajótörést, a túlélők alapította civilizációnk bölcsőjét pedig mindössze a roncs jelentette. A most felfedezett technikát felhasználva az emberiség eddigi történelme legnagyobb vállalkozására készül: a hajótest mellett talált térkép szerint útra kelnek eredeti otthonukba. Mindezt egy olyan produktív anyahajóval, mely sok millió társunkat szállítja.

Így minden fejlesztést, termelést, hipertér ugrást az űrben sikló otthonunkról kell majd irányítanunk...

A játék igen komoly függőség kialakítására képes: élvezetes űrcsatáit látványos *OpenGL* leképezéssel rendereli, a zenei aláfestés nem különben fenomenális. A programkód megjelenése után néhány évvel a fejlesztő közkinccsé tette a forrásállományokat, ami alapján egy maroknyi fanatikus átírhatta ezt a remeket *Linux* alá.

Életre keltéséhez sajnos mindenképpen szükségünk lesz a *Homeworld* windowsos telepítőlemezére.

A *Win32* változatot valahova fel kell telepítenünk egy rövid időre. Erre megfelelő lehet egy *Windowst* futtató gép, de akár egy *Linuxon* futó *Wine* / *Cedega* wrapper is. A sériaszámot beütve, telepítési (esetleg fake) útra került játékot azonnal frissítenünk kell 1.05 verzióra (a folt a *Relic* honlapján található).

Ezek után a ➲ *http://www.thereisnospork.com/projects/homeworld/* címről letöltött linuxos bináris kódot tartalmazó *homeworld-sdl.verzió.tar.bz2* fájlt kicsomagoljuk egy tetszőleges könyvtárba, ahová az imént telepített *Win32* játékverzióból a *Homeworld.big, Update.big, devstats.dat* állományokat át kell mozgatnunk.

Ha készen vagyunk, a telepített *Win32* változat törölhető. A linuxos játék az általunk kreált könyvtárban fellelhető `homeworld` parancs segítségével indítható. A játék *Win32* telepítőlemezét a futtatás idejére be kell fűznünk. Nagyon fontos, hogy a csatolási pontnak a */mnt/cdrom*-nak kell lennie.

A projekt *alfa 0.3* állapotú, ennek ellenére mentes a jelentősebb hibáktól. Egy fontos dologra mindenképpen figyelnünk kell: első indításkor, a menüket használva a megjelenítés típusát *OpenGL* alapúra kell állítani,

3. ábra A Glest működés közben

ezért a működő *GLX/DRI* kapocs itt alapfeltétel. A kód a *Simple DirectMedia Layer* könyvtárakat használja, így *SDL* környezetünket ajánlott naprakészen tartani.
A linuxos játék tulajdonképpen nem különböztethető meg a jóval kiforrottabb *Windows* verziótól ami nem rossz ajánlólevél. A játék menetét tekintve nem a véletlen műve az említett kitüntetés, a *Homeworld* ugyanis hemzseg a jó ötletektől. Személy szerint kifejezetten imádom nézni, ahogyan térugrás előtt minden vadászom, az összes technikám „libasorban" visszatér a vezérgépre.

A Glest projekt

A *Glest* projektet hagytam utoljára, holott talán a legérdekesebb linuxos képességű *RTS*: a ➲ *http://www.glest.org* címen elérhető szoftver többszörösen díjnyertes, szabadon elérhető alkotás, mely a *WarCraft* sorozat hagyományaira próbál építeni. Teljesen *3D* képi világú, összetett modellekkel dolgozó játék, ötletes újításokkal fűszerezve. Igényesen felépített világban kell legyőznünk ellenfeleinket, a műfajra jellemző stílusban figyelve a termelés, fejlesztés és hadi tevékenységünk arányaira.
„Birodalmunk" alapját itt is a termelő munkások fogják jelenteni, harcosainkat pedig a lovagoktól kezdve egészen a sárkányokig bezárólag választhatjuk/fejleszthetjük attól függően, hogy technika vagy mágia vonalon szeretnénk küzdeni. A programban olyan apróságokra is figyeltek a készítők, mint a napszaknak megfelelően alakuló fény- és árnyviszonyok, nem beszélve a választható évszakról, ami által ködös, havas tájon „oszthatjuk" ellenségeinket.
A programkód natív linuxos formában is elérhető. Kész binárisként, *Loki* – alapú telepítőbe ágyazva a ➲*http://liflg.org* címen keresendő, *glest.verzió.run* néven – mely archív a *Glest* binárisát és adatcsomagját foglalja magában. Rootként indítva az említett állományt, a játék alapértelmezésben a */usr/local/games* útra kerül, egy *glest* elnevezésű kötéssel az */usr/local/bin* mappában így az egyszerű felhasználóként kiadott `glest` paranccsal indítható.
Futtatásához erős gépre lesz szükségünk, működő *GLX/DRI* hajtotta grafikus kártyával. (Érdekesség, hogy a *Glest* ugyan elérhető forrásként is – viszont komoly függőségei miatt nagyon nehéz fordítani, továbbá a kód nem teljes egészében szabad. A *Loki Installers for Linux Gamers* csapatának hála, a bináris nagyon egyszerűen üzembe állítható a leírt módon.) Ez a program a stílus szabad megvalósítású ágának koronázatlan királyává nőtte ki magát, igen rövid idő alatt. Hazai rajongói oldala ➲*http://glest.atw.hu* oldalon érhető el, a magyar nyelvi fájl is itt található. Őszintén remélem, hogy sikerült a kedvére tennem mindazoknak, akiket komolyabban érdekel a stratégiai műfaj, és Linux alatt is szívesen kipróbálnák rátermettségüket. Kellemes időtöltést kívánok!

Kovács Zsolt (kovi@linuxforum.hu)

Quake fanatikus. Négy éve a debreceni linuxosok egyike. Töretlenül hisz a Slackware terjesztésben.


Linuxvilág
www.linuxvilag.hu

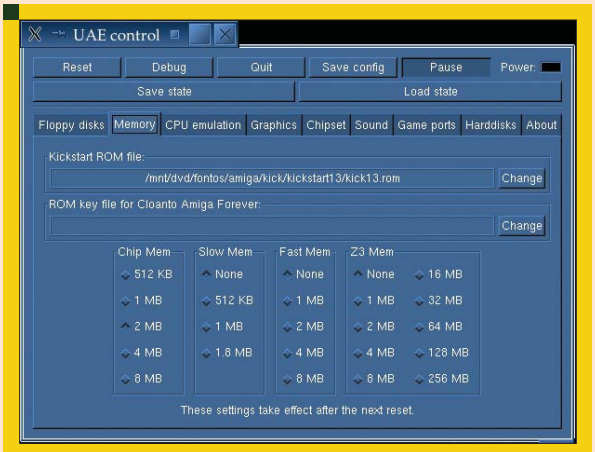
Az örökifjú Amiga 500

Ebben a cikkben a legendás Commodore Amiga 500 számítógépet fogjuk felélesztени, mégpedig Linux platformon.

Az „újkori számítástechnika” történelmében kitüntetett szerepet betöltő ***Commodore Amiga*** számítógépcsalád a kilencvenes évek elején rendkívül komoly piacot tudhatott magáénak. Története azzal kezdődött, amikor az eredetileg az ***Atari*** kötelékébe tartozó ***Jay Miner*** elhagyta munkaadóját és egy saját, minden konkurenst elsöprő gép építésébe kezdett. (Egyébként sokak szerint ***Miner*** azért választotta az ***Amiga*** nevet, mert az ***Atari*** előtt akart szerepelni a szakmai listákon.) Bár ***Miner*** kézzel fogható eredményeket produkált, rövidesen igen nehéz anyagi helyzetbe került. Ironikus módon leginkább az ***Atari*** mutatott komoly érdeklődést a felvásárlást illetően, de a ***Commodore International*** nagyobb vételárat kínált, így végül az ő kezükbe került a zseniális projekt. A ***Commodore*** irányítása alatt ***Amiga*** számítógépek egész sorozata látott napvilágot. A 16 bites „csodamasinák” (elsősorban az ***500-as*** és ***1200-as*** szériák) a szoftverfejlesztők és egyszerű vásárlók körében egyaránt népszerűek voltak, mivel képességeikhez mérten korrekt összegért lehetett hozzájuk jutni, paramétereik pedig messze megelőzték az adott kor fejlettségi szintjét. Az ***Amiga 500*** egy mai, ***PC***-hez szokott felhasználó számára leginkább egy vaskos billentyűzetnek tűnik. (A szépség ugyebár szubjektív dolog. Szerintem kifejezetten formás gépről van szó, amit gyűjteményem egyik nagybecsű darabjaként őrzök).

1. ábra Íme, az Amiga 500

A számítógép beépített floppy meghajtója, ***Motorola 68k*** processzora, 512 Kbyte ***ChipRAM***-ja, tisztességes hangkezelése saját idejében igen komoly kereskedelmi potenciált jelentett. Az ***Amigákhoz*** a ***Commodore*** gyártottak külön monitort is, ám a felhasználók többsége ***TV***-modulátor segítségével egyszerű televíziókat használt megjelenítőként – jelentős összeget takarítva meg ezzel. A kilencvenes évek elején már tömegesen jelentek meg a gép képességeit végletekig kihasználó játékprogramok, az igen jó színvonalat képviselő szoftverek között pedig már fellelhető néhány – számunkra igen régi – ***PC***-s ismerős is. Talán akadnak az olvasók között, akik emlékeznek a ***Settlers***, ***Prince of Persia***, ***Lotus***, esetleg a ***Street Rod*** első részeire. Hála a lelkes gyűjtőknek, ez a hihetetlenül sok program ma is elérhető, javarészt az emulátorok számára megfelelő ***Amiga Disk File*** (***ADF***) formátumban.



2. ábra Az emulátor GTK alapú felülete

A 16 bites ***Amiga 500*** emulálására ***Linux*** alatt több megoldás is létezik. Ezek közül most az ***Ultimate Amiga Emulatort*** szeretném bemutatni. Kiforrott program, hosszú évek óta szolgálja felhasználóit, a ➲ ***http://www.freiburg.linux.de/~uae*** címen található honlapjáról pedig szabadon letölthető a forrása illetve binárisa egyaránt. Hangolhatóságának köszönhetően több ***Amiga*** szériát is képes emulálni. Lefordítani a szokásos

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

parancsokkal lehet. Könnyen használható forráskódok esetén ritkán szoktam ajánlani az előre fordított bináris verzió használatát. Mivel azonban ennél a programnál semmilyen különbség nincs a két megoldás hatékonysága között, ha valaki nem szeretné az idejét pocsékolni, nyugodtan használhatja a kész binárist is (mindössze a ***GTK*** függőségekre kell figyelni).
A program felhasználóként kiadott

```
uae
```

parancsra indul. Használatához szükség lesz a megfelelő ****.adf*** formátumú lemezképek mellett a jogvédett

3. ábra Így néz ki az „öreg" Lotus2

4. ábra A Settlers világa

kickstart.rom állományra is. Utóbbi tartalmazza a gép beégetett, valódi többfeladatos operációs rendszerének rutinjait. Az eredeti ***Amiga 500***-hoz tartozó ***KickStart***-ot már elég nehéz fellelni, a modulárisabb felépítésű ***1.3*** verzió beszerzésére viszont nagyobb esélyünk van. Ha bármilyen ***Amiga*** állományt keresünk, az első három hely, ahova érdemes ellátogatni a következő: ➲ ***http://back2roots.org,*** ➲ ***http://amiga.emucamp.com***, valamint a ➲ ***http://www.marksplace.f9.co.uk***.
Az emulátor első indításakor érezhetően lassú, ezek után viszont nem lehet okunk panaszra. Beállítását egy életszerű példán keresztül fogom bemutatni. Nézzük tehát, hogyan lehet életet lehelni egyik régi kedvencembe, a ***Lotus2*** játékba!
Az ***UAE*** kezelőfelületén lévő ***Memory*** fülön válasszuk az esetünkben használandó ***kickstart13.rom*** állományt, majd az eredeti ***Amiga*** konfigurációt erősebbre állítva a ***Chip Mem*** értéket 2 Mbyte, ***Slow Mem*** értéket 512 Kyte, a ***Fast Mem*** értéket 2 Mbyte méretre érdemes rögzítenünk. Ezek után a ***CPU Emulation*** fülön található a ***68k*** megoldást lehetőség szerint változtassuk ***68k20***-ra. A listában elérhető processzorok némelyike matematikai társprocesszorban különbözik a többitől, az ***FPU*** viszont számunkra most lényegtelen. A ***CPU Speed*** kapcsolót tegyük 7 MHz állásba. Grafikai finombeállítással most ne foglalkozzunk, mindössze az eredeti ***Original ChipSet*** beállítást változtassuk ***Full ECS***-re, a ***sprite*** ütközés kezelését pedig tegyük ***Sprites Only*** állásba.
Hang természetesen állítható ***normal***, ***stereo***, ***16 bit*** értékre is, ne feledjük azonban, hogy az eredeti gép csak 8 bites hangkeltéssel rendelkezett!
Game Ports fülön ízlés szerint kell beállítani a vezérlést – én egyes vezérlőként mindig a billentyűzetem numerikus részét adom meg, másodikként pedig a kurzorbillentyűket. Az ***UAE*** egyébként botkormányt is kezel!
A ***HDD*** kezelés utánzása ***Amiga 500*** esetében lényegtelen, ez eredetileg a 600-as sorozat rendszerébe került be lényeges fejlesztésként. (Ennek ellenére merevlemez használható volt az említett régebbi szérián is, de csak speciális illesztőkártyával.)
Álljunk vissza a ***Floppy*** fülre, ***Insert*** gombbal válasszuk ki az autóverseny lemezének lenyomatát, majd „engedjük fel" a ***Pause*** gombot. Eredményképpen új grafikus ablak bukkan fel, amelyben először a ***KickStart*** képe, majd a játék betöltője (***loader)*** fogad minket. Rövid idő múlva a ***Lotus2***-be feledkezhetünk...
A programok betöltésére nem ritkán várnunk kell akár egy egész percet is. Ha valamit elrontottunk, a kívánt ****.adf*** állomány nem töltődik be. Ilyenkor a felbukkanó ablak alsó részén látható, lemezegységet szimbolizáló fények hibát jeleznek. Teljesen félresikerült beállításra, hibás lemezképre a híres „***Guru Meditation***" üzenetet kapjuk: ez volt a számítógép egyik jellemző – de meglehetősen ritka – hibaüzenete.
A paramétereket futás közben is meg lehet változtatni, de ezzel a lehetőséggel bánjuk óvatosan: röptében legfeljebb a kontrollbillentyűket állítgassuk.
Ha új programot szeretnénk betölteni, csupán a ***Reset***, majd ***Pause*** gombot kell megnyomni a kezelőfelületen, és már jöhet is a „virtuális lemezcsere". Többlemezes programnál a diszkek cseréje a ***Pause*** funkció használatával lehetséges, a megállított emulátorban egyszerűen válasszuk ki az új állományt, majd engedjük tovább a futást.
A más beállításokat igénylő, későbbi és komolyabb programok (például a ***Shogo*** amigás változata, egyéb ***scene*** munkák) konfigurációs környezete a fentebb említett ***back2roots*** oldalon le van írva, egészen az ***AGA*** lapkakészletes gépeket utánzó beállításokig. Jelentős terjedelmük miatt ezeket az információkat itt most lehetetlen leírnom.
Amellett, hogy az emulált ***Amiga*** platform összes programjával együtt digitális örökségünk megbecsülendő része, bármilyen régi program, bármely játék az ***UAE*** segítségével a fiatalabb és „tapasztaltabb" korosztálynak egyaránt több órás szórakozást ígér.
Mivel a cikk írásakor a projekt honlapján linkelt ***FTP*** elérhetőségek szüneteltek, így az érdeklődők számára feltöltöttem egy linuxos ***UAE 0.8.21*** verziójú binárist tartalmazó ***RPM*** csomagot ide: ➲ ***http://kovi.uw.hu/lvilag2006***. Használatához természetesen szüksége lesz az olvasónak ***KickStart ROM*** lenyomatra.

Kovács Zsolt (kovi@linuxforum.hu)

Quake fanatikus. Négy éve a debreceni linuxosok egyike. Töretlenül hisz a Slackware terjesztésben.