Beköszöntő

Linuxvilág

Augusztusi számunk témalistáját elnézve azt hiszem ez lesz az egyik legszínesebb ***Linuxvilág*** szám, amióta a „nagy váltás” csaknem egy évvel ezelőtt megtörtént. Lesz szó zenéről, grafikáról, játékokról, párhuzamos programozásról, webfejlesztésről, hibakeresési módszerekről, titkosításról, hálózatkezelésről, irodai programról, kiadványszerkesztésről. Lássuk mindezt kicsit részletesebben.
A ***Fejlesztői sarokban Karai Csaba*** a memóriafoglalás hibáinak felderítéséről, ***Szabó Zoltán*** a ***PHP*** programok hibakeresésről, ***Radics Péter*** pedig az ***SDL*** könyvtárról ír. A hardverfejlesztők, klönösen pedig a processzorok tervezői körülbelül tíz évvel ezelőtt kezdték el előbb csak csendben, aztán egyre hangosabban mondogatni, hogy a félvezetőkből kezd „kifogyni a szufla”. Az órajel egy bizonyos határon túl már nem emelhető, és bár az utasításkészletek ügyes kialakításával még lehet kicsit húzni-halasztani a dolgot, az igazi megoldást tulajdonképpen a párhuzamos programok és a szuperszámítógépek jelentik. Tíz éve a szakértők úgy gondolták, hogy a lassulás első jelei valamikor 2005-ben mutatkoznak majd meg, és talán nem is tévedtek nagyot. Egy másik, újabb keletű elemzés szerint a közönséges (értsd: boltban kapható) alkatrészekből megépíthető mini-szuperszámítógépek néhány éven belül tömegcikknek számítanak majd. Az igazi probléma tehát mostantól nem a sebesség, hanem a párhuzamosítás lesz. Mindez különös aktualitást kölcsönöz ***Bánki Horváth András PVM***-ről szóló cikkének.
Egyfajta ellenpontként az alkalmazásokat bemutató rovatban ebben a hónapban viszonylag nagy hely jutott a ***KDE***-alkalmazásoknak, jelezvén, hogy a ***Linux*** asztali rendszerként is tökéletesen megállja a helyét.
Az augusztusról a legtöbb embernek a hullócsillagok jutnak eszébe, így gyakorlatilag kihagyhatatlan volt a csillagászat mint cikktéma. A megoldást ***Kovács Zsolt*** szállította, aki a ***Celestia*** nevű programot mutatja be.
A rendszerek üzemeltetőinek szóló rovatban ezúttal betörőt fogunk (***Horváth Ernő***; ***Tripwire***), gépet klónozunk (***Auth Gábor***; ***G4U***), a ***GnuPG*** segítségével pedig minden, a kezünk ügyébe eső dolgot titkosítunk.
Végezetül megint jöhet némi művészet és szórakozás. ***Szalai András Blendert*** bemutató sorozata az utolsó előtti részéhez érkezik, ***Fábián Attila*** pedig megmutatja, hogyan lehet ezt a rendszert kissé felpörgetni. ***Juhász Attila*** ezúttal mindenféle csavaros dolgokat művel a ***Gimppel***, ***Kovács Zsolt*** pedig kottából próbál játszani, aztán meg azon nosztalgiázik, milyen szép is volt az, amikor a számítógépet még ***C64***-nek hívták.

Mindenkinek jó szórakozást, kellemes időtöltést kíván a Linuxvilág stábja!

Hírek

Pepper Pad 3

Augusztustól már kapható a *Linux*ot futtató és *AMD Geode* processzorral szerelt *Pepper Pad 3*, amely kisebb, gyorsabb és olcsóbb elődjeinél. A *Pepper Pad 2*-eshez képest elég jelentős a változás, gyorsabb a *WiFi* és az *USB* kapcsolat, valamint kapott *VGA* felbontású *kamerát*. A készülék lelke egy 533 Mhz-en futó *AMD* processzor. A méretek is impozánsak: 29 centiméter széles és 15 centiméter magas, a tömege pedig alig egy kilogramm. A megjelenítésért egy 800x480 képpontos, 7 hüvelykes érintőképernyő felel. Az alapmodell 256 megabájt memóriát és 20 gigabájtos merevlemezt tartalmaz. Az ára előrendelés esetén 700 dollár körül alakul.

➲ http://www.linuxdevices.com/articles/AT5638626152.html

VoIP már a Firefoxban is

Már most is joggal állítható, hogy *svájci bicska* a *Mozilla Firefox* megfelelő kiegészítők (plugin) telepítése után. Ezt csak tetézi, hogy az *Abbeynet Labs* kiadott egy újabb kiegészítőt, amellyel „*SIP*" kompatibilis *internetes telefonklienssé* varázsolhatjuk a kedvenc böngészőnket. A „*SIP*" technológiáról a *Linuxvilág 2005 novemberi* számában olvashat részletesebben az olvasó.

➲ http://labs.abbeyphone.com/firefox/abbeyphone-ff.xpi

Linuxos mobiltelefonok, egyesüljetek!

Négy nagy mobiltelefongyártó és két szolgáltató (*Motorola, NEC, NTT DoCoMo, Panasonic, Samsung, Vodafone*) együttműködési megállapodást írtak alá, melynek értelmében létrehoznak egy közös, nyílt, *Linux*-alapú telefonplatformot.

➲ http://www.linuxdevices.com/news/NS2710208789.html

GSM és WiFi telefon

Neuf, a francia *GSM* és *Wifi* szolgáltató bemutatta legújabb duál módú telefonját, amely egy *Wistron NeWeb's GW1*. A telefon *2.6.10*-es *Linux* kernelt futtat, a grafikus felületet pedig a *Qtopia 2.2* adja. Egy feltöltéssel 200 óra a készenléti és 4 óra a beszélgetési idő. Az *LCD*-je a jelenlegi telefonokhoz képest elég jónak mondható, hiszen 176x220 képpont és képpontonként 262 ezer színt tud megjeleníteni. Beépített programok között találunk *POP3*-at és *IMAP4*-et tudó levelező klienst, *mp3* lejátszót, *Opera 8*-as böngészőt, illetve pár alapprogramot: számológépet, órát, diktafont. A tudása ellenére a tervezőknek sikerült ehhez mérten kis telefont alkotni, hisz méretei: 106 x 45 x 19 mm és a tömege is kevesebb 10 dekánál, igaz nem sokkal marad alatta. Az ára hálózatfüggetlenül 200 euró körül várható.

➲ http://www.linuxdevices.com/news/NS9996556326.html

Mini-ITX méretű számítógépek multimédia célokra

A *VIA* már szállítja az 1 vagy 1.3 gigahertzes processzorral szerelt, mintegy 16 watt energiaigényű *Epia CN* alaplapját, amelyet elsősorban multimédiás célokra ajánlanak. Az alaplapra szerelt *C7*-es processzor az *Intel Celeron M* processzor energiaigényének csupán a felét használja. A kis energiaigény ellenére minden szabványos csatlakozó megtalálható rajta. A korábbi *Epia EN*-el szemben *nem kapott* azonban *Firewire* csatlakozót, illetve hálózatból sem kezeli a gigabites sebességűt. Továbbá lemaradt róla a digitális audió csatlakozó. Memóriából maximum 1 gigabájt helyezhető bele, merevlemezből viszont akár *4 ATA* és *2 SATA* is csatlakoztatható. Az alaplap jelenleg *Linuxot*, *Windows 2000*/*XP*-t, vagy *Windows CE*-t futtathat. Jó hír annak, aki *Linux*ot szeretne rajta futtatni: az integrált *VIA UniChrome Pro* grafikus kártya bináris meghajtóprogramja *Linux* alatt *MPEG2* lejátszást is gyorsít. Az ára 225 dollár körül várható 1.3 gigahertzes processzor esetén.

➲ http://www.linuxdevices.com/news/NS4357200751.html

Taiwan a Linux mellett

A tajvani kormányzat döntése értelmében minden újonnan eladott *PC*-nek tudnia kell *Linux*ot futtatni. A döntéssel a kormányzat a nyílt forrású szoftverek terjedését szeretné elérni.

Titkosított merevlemezek

A *Seagate* a *tajvani Computex* kiállításon bemutatta legújabb, *Momentus* elnevezésű merevlemez családját, melyben egy beépített chip külső program nélkül végez 1*28 bites AES* titkosítást a merevlemez teljes területén. A család minden tagja *8 megabájt* gyorstárral szerelt, és *60, 80, illetve 160 gigabájtos* kapacitást nyújt. A híradásban arról már nincs szó, hogy a titkosítás feloldására hivatott jelszót mikor kell begépelni (talán a gép indulításakor?). A merevlemezek várhatóan *2007 elejétől* kerülnek a boltokba.

Linuxé már a Niagara is

A *2.6.17*-es kernelverziótól kezdve támogatja a *Linux* a *Sun Niagara* processzorát. A támogatás egyszerre 8 magot képes kezelni, magonként 4 szállal. A legelső disztribúció, amely ennek köszönhetően támogatja a *Niagarát*, az *Ubuntu Drapper Drake*.

Táblázatkezelő Google módra

A *Google* szemmel láthatóan a *Microsoft* babérjaira tör, hiszen elkészítette saját táblázatkezelőjét, mely képes *CSV* és *XLS* formátumú állományokkal dolgozni. Az alkalmazás hálózatos mivolta miatt azonban több biztonságtechnikai szakember ítéli el, hiszen *biztonsági kockázatot rejt* magában.

➲ http://slashdot.org/article.pl?sid=06/06/06/1226236

IP TV Linuxszal

A *Visioneering* piacra dobta *Sonata* névvel *set top box*-át, amely nem más, mint átalakító normál TV-khez *IP TV* adások vételére.
A rendszer lelke egy *Texas Instruments* gyártmányú *DaVinci* kódnevű processzor. Az eszköz érdekessége, hogy *Linux*ot futtat, továbbá *vezetéknélküli* kivitelben is kapható. Az eszköz kezeli a *PAL* és az *NTSC* szabványt is. Az ára mennyiségtől függően 100 és 170 dollár között alakul.

➲ http://www.linuxdevices.com/news/NS4123198942.html

Újabb, Linuxszal készült animációs film

A *Pixar* legújabb filmje – a *Verdák* (az angol címe: *Cars*) – is *Linux* alatt készült. Noha mind a munkaállomások, mind a szerverek jelentős számítási kapacitással bírtak, mégis *egy-egy képkocka 8 óra alatt készült el*. Mindez annak tudható be, hogy sugárkövetéses eljárást (*raytracing*) használtak (sugárkövetésen alapul például a *Povray* is).

Korábbi animációs filmjük, A *hihetetlen család* (az angol címe: *The Incredibles*) ehhez képest sokkal hamarabb készült el, hiszen a modellezés valósághűsége az évek során tökéletesedett, közelített a valóshoz, ennek tudható be a megnövekedett számítási igény is.
A *Pixar* három szoftvert használ filmjeihez. A *Marionette*-t (*Linux*on, *Solaris*on vagy *Irix*-en futhat) az animáláshoz és a megvilágításhoz, a *Ringmaster*-t az időzítésekhez, illetve a mozgásokhoz, végül pedig a *RenderMan*-t, amelyel létrehozták a fotórealisztikus filmkockákat.
Sajnos azonban – hacsak nem dolgozunk a *Pixar*nál – nem használhatjuk ezeket a programokat, hiszen zárt forráskódúak.

➲ http://www.linux-watch.com/news/NS6281055297.html

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

Paul Mundt benyújtott egy olyan foltot, amely eltávolítja a magból a *RelayFS* minden részét és azokat egy olyan általános *API*-ba helyezi, amely valamennyi fájlrendszer számára elérhető. Annak ellenére tehát, hogy a *RelayFS* egy egészen speciális szolgáltatásnak indult – jelesül az volt a feladata, hogy nagy sebességű adatátvitelnél gyorsan tudjon adatokat mozgatni a felhasználói tér és a mag között – mostanra általános, széles körben alkalmazható rendszerelemmé nőtte ki magát. Mivel a *RelayFS* már elég régóta része volt a magnak, az eltávolítása okozott néhány belső ellentmondást is. Ennek ellenére *Andrew Morton* azt ígérte a fejlesztőknek, hogy nyugodtan haladhatnak tovább ezen az úton egészen addig, amíg a magba beépülő kód nem függ közvetlenül a *RelayFS* kódjától. Mindez összefoglalva azt jelenti, hogy lesz egy olyan bizonytalan, átmeneti időszak, amikor a felhasználói alkalmazásoknak nem szabad vakon a *RelayFS* szolgáltatásaira támaszkodniuk.

Az *Intel* bejelentett egy új, nyílt forrású fejlesztési projektet, amelynek célja az általa gyártott *PRO/Wireless 3945ABG Network Conncetion* nevű *mini-PCI express* csatlakozóval szerelt adapter támogatásának biztosítása (*IPW3945*). A fejlesztés ugyanakkor nem teljesen nyílt forrású, hiszen van egy olyan, csak bináris formában elérhető része is, amely azoknak az országoknak a hatályos rendelkezésihez igazítja az eszközök működését, ahol azokat forgalmazzák. A bejelentésben *James Ketrenos* hangsúlyozta, hogy az *Intel* a korábbiakhoz képest jelentősen könnyített a lincencelés feltételein. Jelen esetben ez azt jelenti, hogy a csak bináris formában hozzáférhető csomagra ugyanazok a feltételek vonatkoznak, mint a szintén csak binárisan elérhető beégetett programra (firmware). James szerint a feltételek ráadásul sokkal könnyebben értelmezhetőek, és könnyebb továbbadni a fejlesztett kódot. Amint az tulajdonképpen várható is, az egész részleges nyitottság meglehetősen ellentmondásos fejlesztési módszer. A bináris démont nyilván root jogosultságokkal kell futtatni, ami azt jelenti, hogy ha bármilyen hiba van benne, az nagyon komoly biztonsági kockázatot jelenthet. Ráadásul az *FCC* vonatkozó szabályainak értelmezése sem egyértelmű. Egyesek – például *Alan Cox* – szerint a szervezet által megfogalmazott követelményrendszerből nem következik, hogy a meghajtót csak bináris formában adhatja ki a gyártó, csupán arról kell gondoskodnia, hogy lehallgatásbiztos legyen az átvitel. Függetlenül azonban minden ezzel kapcsolatos vitától – amely amúgy várhatóan a jövőben is folytatódik majd – az *Intelt* mindenképpen elismerés illeti, amiért legalább részben nyílt forrásúvá tette a fejlesztést, illetve amiért láthatólag igyekezett átdolgozni a licencelési feltételeket.

Bernhard Rosenkraenzer saját korábbi fejlesztésével egyesítve *dvdrecord* projekt néven új ágat nyitott a *cdrtools* projekt fejlesztésében és kiadta annak 0.3.1-es változatát, amely számos újítást, például a 2.6-os rendszermag támogatását is tartalmazza. Az új szoftver emellett immár úgy támogatja a *DVD-R* és *DVD-RW* lemezek írását, hogy ahhoz kizárólag szabad szoftvereket használ, illetve letisztult benne a *make* rendszer is. Az eset kapcsán óriási vita bontakozott ki, amelynek végkicsengést valahogy úgy lehetne összegezni, hogy sokak szerint *Bernhard* munkája volt az egyetlen érdemleges előrelépés az elmúlt időben ezen a területen. Később a társalgás inkább a jövőbeni tervek irányába terelődött, és egyesek arról érdeklődtek, mikor jelenik meg a csomagban az olyan formátumok támogatása mint a *DVD RAM*, a *DVD+R*, a *DVD+RW* vagy a *DVD+DL*. Az ilyen és ehhez hasonló kérdésekre persze egyelőre nincs válasz, hiszen a projekt hosszú távú jövője nem tisztázott. Általában is elmondható, hogy az ilyen váltások jövője kezdetben eléggé megjósolhatatlan, tehát a leglényegesebb eredménynek pillanatnyilag azt kell tekintenünk, hogy *Bernhard* lépése nem ütközött különösebb

ellenállásba. Azzal kapcsolatban azonban, hogy a mostantól általa fejlesztett rendszer hová fog eljutni, egyelőre korai volna találgatásokba bocsátkozni.

Szeredi Miklós elkészítette a ***mountlo*** nevű segédeszközt, amivel a felhasználói térben lehet visszacsatolt (***loopback***) eszközön keresztül fájlrendszereket becsatolni. Egészen eddig ha be akartunk csatolni egy a merevlemezen tárolt lemezképet, azt csak a rendszermag saját ***loopback*** szolgáltatásával tehettük meg. Az új eszköz ezzel szemben a ***FUSE*** (***Filesystem in USErspace***) rendszerre támaszkodik, amivel az egész szolgáltatást a felhasználói térbe, vagyis a rendszermagon kívülre helyezhetjük. Maga a fejlesztő amúgy az egész projektet inkább amolyan játéknak tekinti, amivel az volt a célja, hogy valamilyen értelmes dologra kezdje használni a ***FUSE*** szolgáltatásit, és kitapasztalja, mihez is lehet azokat hatékonyan felhasználni.

A mindig szemfüles ***Christopher Hellwig*** rámutatott, hogy a jelek szerint pillanatnyilag a ***gdth*** meghajtó az egyetlen olyan rendszer, ami a várhatóan a 2.6.17-es magban megszűnő ***scsi_request*** interfészre támaszkodik. Mivel a ***gdth*** fenntartásáért felelős fejlesztők egyelőre nem reagáltak az ezzel kapcsolatos megkeresésre, elképzelhető, hogy a 2.6.17-es kernelben ez a rendszer a ***„BROKEN”*** jelzőt kapja majd.

Achim Leubner ugyan bejelentette, hogy hajlandó teszteléseket végezni, de ezzel korántsem oldódott meg az a kérdés, hogy tulajdonképpen ki is a fenntartó. A szokásos eljárás amúgy az, hogy az árván maradt kódok villámgyorsan kikerülnek a rendszermagból.

Greg Kroah-Hartman nekilátott az ***ABI*** (***Application Binary Interface***) stabilitási szintekkel kapcsolatos dokumentáció elkészítésének.

Ha megváltozik egy bináris interfész, valamennyi olyan a felhasználói térben futó bináris hibás lesz, ami erre támaszkodik. És mivel egy-egy interfésznek megszámlálhatatlanul sok felhasználója lehet, a kernel ***ABI***-k módosítása nem számít szalonképes ötletnek. Ugyanakkor a felületek változását néha az élet követeli meg. Ilyesmi a múltba is előfordult már, és teljesen bizonyos, hogy a jövőben is elő fog. ***Greg*** szerint éppen ezért nem az a kérdés, hogy egy bináris felület változhat-e vagy sem, hanem hogy melyek legyenek azok a szempontok, amelyek alapján mérlegelni lehet a hasznokat és a károkat. Az ő ötlete szerint a felületek dokumentációjában világosan fel kell tüntetni, hogy a fejlesztők milyen módosításokat terveznek velük kapcsolatban a jövőben. Ezt tudva az alkalmazások fejlesztőinek már elég idejük lehet arra, hogy felkészüljenek a váltásra.

Amint az várható volt, a felvetés heves vitákat eredményezett. Számos magas rangú kernelfejlesztő az ***ABI***-t amolyan szent tehénnek tekinti, amit soha nem szabadna módosítani.

Maga ***Linus Torvalds*** a kritikák ellenére úgy véli, hogy ***Greg*** alapvetően jó irányba indult el, mert a változtatások egyszerűen elkerülhetetlenek, így a kernelfejlesztőknek mindent el kell követniük annak érdekében, hogy megkönnyítsék a többiek életét.

Linux Journal 147. szám

Zack Brown

A Linux mint játékplatform – múlt, jelen, jövő

A Linux története az első rendszermag megjelenésétől kezdve példaértékű. A fejlesztői közösség lelkesedésének hála, a nagy számú disztribúció közül némelyik ma már igen komoly alternatívát nyújt a mindenkori piacvezető termékek ellenében.

■ Az elmúlt években alkalmazott rendszermodell már számtalan helyen bizonyított, többek között az otthoni szegmensben is, ahol leginkább multimédiás és szórakozási céllal, esetleg kisebb irodai munka erejéig ülünk képernyőnk elé. Ebben a régióban érezhetően nyomja a képzeletbeli mérleget a játékok piaca is, hiszen sokak szerint „nem is igazi rendszer" az, mely alatt nem lehet egy kiadósat játszani. És valóban. Ugyan, melyikünk ne szeretné néha egy jó játékprogrammal agyonütni az időt? Játékok tekintetében azonban a ***Linux*** kétségtelenül elmarad a ***Windows*** mögött, ami persze nem jelenti azt, hogy nem lenne sok népszerű kereskedelmi játéknak linuxos változata is.

A közelmúlt

Nézzük röviden a „száraz" tényeket! A ***Linuxot*** kezdetben nem lehetett befogni otthoni használatra, vagy multimédiás platformnak, így játékprogramok alaprétegeként sem szolgált. A dolog mögött számos tényezőt búvt meg, kezdve a kezdetben kis létszámú felhasználói tábortól egészen a megfelleő programozási felületek hiányáig.
Mindez azért nem tekinthető igazán a negatívumnak, mert a ***Linux eredetileg egyáltalán*** nem szórakozási céllal született. Ennek megfelelően sokáig csak a „komolyabb" projektek profitáltak a stabilitásából és az áttekinthető szoftvermodellből.
Volt persze néhány grafikus felületet nem igénylő „játékprogram" is, de a „könnyed kikapcsolódás" lehetősége valójában a hatékony ***X*** kiszolgáló megjelenésével vált kézzel foghatóvá. Mint mindenütt, itt is akadt néhány a szabályokat erősítő kivétel, mint például a ***Doom***. Az ezt fejlesztő ***id Software a*** játék ***MS-DOS***-ra szánt változata (1993) után két évvel kiadott egy linuxos binárist is, a csapat pedig jelenleg is azok közé tartozik, akik komolyan gondolják a szabad szoftver ilyetén támogatását.
Az út innen egyenesen vezetett a komolyabb műfajok felé, ami előtt a falat valójában azok a programozási felületek segítették áttörni, melyek által a fejlesztők (már munkájuk kezdetén) komolyan fontolóra vehették a független kivitelezést és használhatóságot. Ahogyan telt az idő, az ***SDL*** keresztplatformos környezete, az ***OpenAL*** és ***OpenGL*** programozási felületek (sok más rendszerkomponenssel együtt) oly mértékben váltak használhatóvá, ami már hosszú ideje megkérdőjelezhetetlenül stabil és komoly alapot biztosított a fejlődésnek. Ezekre alapozva, a születendő játékprogramok ***Linux*** alatt közel ugyanazt a teljesítményt, látványvilágot és fizikát képesek elérni, melyet a ***Windows*** korszerű verziói alatt elérhetnek.

Kitaposott út

Linux felületen a „modern" natív játékok között több csoportot vélhetünk felfedezni. Vannak a platformtól független kereskedelmi megvalósítások, vannak aztán a – sok esetben – idegen kezek által átültetett kereskedelmi projektek, de akadnak olyan szabad játékok is, amelyek (általában ***GPL***) forráskódja gyakorlatilag az összes, tényleges felhasználói táborral rendelkező operációs rendszerre felkészíthető és azon lefordítható.
A legjellemzőbb független kereskedelmi játékokat mind a mai napig az előzőekben már említett ***id Software*** fejlesztői adják ki kezeik közül. Rugalmas kódjaiknak köszönhetően valójában a konkurenciát is belekényszerítve az említett függetlenség követésébe. A „***Linux***-képes" kereskedelmi játékok nagyágyúit nem hiszem, hogy be kellene mutatnom, elég lesz csupán a ***Doom***/***Quake*** sorozatra, az ***Unreal Tournament*** ágra, vagy a ***Neverwinter Nights*** családra gondolni: mára már nem kell megerőltetnie magát annak, aki ebbe a kategóriába tartozó népszerű játékot szeretne említeni.
Ezek a programok jellemzően ***OpenGL API***-n keresztül programozzák a grafikai hardvert, ahol gyakran az ***SDL*** könyvtárrendszert is segítségül hívják. Hangszolgáltatásuk ***ALSA***/***OSS*** meghajtóra épülve sokszor ***OpenAL*** illetve ***FMod*** könyvtárakra támaszkodik, mely leképezési módszerek többek között ***Linux***, és ***Win32*** operációs rendszereken egyaránt elérhetőek. Így a fejlesztőknek effektíve könnyű dolga akad az eltérő rendszerekre fordított binárisok felépítését illetően, miközben arról sem szabad megfeledkeznünk, hogy igen hatékonyan kihasználható, jól

dokumentált és „masszív” alapokról beszélünk, melyek már számos helyen bizonyították: van élet a *DirectX*-en túl.

A második csoport

Ez a csoport kitüntetett figyelmet érdemel, mivel az a „porter”, aki már egy elkészült, *Win32/DirectX*-re támaszkodó kódot szeretne átültetni *Linuxra,* jellemzően olyan komoly feladat elé néz, melynek kivitelezése elképesztő (és sokszor „önzetlen”, anyagiaktól mentes) munkával jár. Ezen a területen feltétel nélküli tisztelettel tekinthetünk a *Scott Draeker* vezette, néhai *Loki Entertainment* munkásságára, hiszen ők voltak azok az úttörők, akik a *DirectX* környezetre fejlesztett játékok magját tömegesen írták át a szabad rendszerre, jellemzően *OpenGL* és *SDL* programozási felületekre / rétegekre támaszkodva.
Közel húsz, igen komoly potenciállal rendelkező átültetés fűződik nevükhöz. Nem sorolom őket, de aki kíváncsi a listára, az a még elérhető (és sajnos négy éve inaktív) ➲ *http://www.lokigames.com* oldalon minden témába vágó információt megtalál. A csapat hiánya mind a mai napig érezhető, bár tevékenységük természetesen maradandó és számunkra felbecsülhetetlen értékű: az olyan nagyágyúk, mint a *HOMM3* vagy éppen a *Descent3* a *Loki* nélkül nem léteznének natív, linuxos verzióban. (Szintén figyelemre méltó az általuk megálmodott, jellemzően **.run* formájú telepítő alkalmazás, mely mindmáig használatban van, egyéb projektek által.) A *Loki* szerepét ma leginkább a nem kevésbé szakértő *Icculus* csapata tölti be, több – kevesebb sikerrel, eltérő (nem kereskedelmi) szemlélettel.
A ➲ *http://icculus.org* oldalon hosszú listán keresztül követhető nyomon a párhuzamosan foltozott kódok állapota, nem ritkán bináris formában és *Loki*-alapú telepítőbe drótozott elérhetőséggel. Természetesen azokról a „kisebb” porterekről se feledkezzünk meg, akik néhány fős csapatukkal vállalkoznak a nehéz feladatra. Jó példa erre a *Relic Homeworld*-jét átdolgozó csoport, hiszen ennek linuxos változata is meggyőzően sikerült az idegen kezek által. Ennél a „műfajnál” a legfőbb buktatót valójában maga az eredeti fejlesztő jelentheti: amennyiben bármilyen okból kizárólag a piacvezető operációs rendszert támogatja, teljesen érthetően közel sem biztos, hogy programjának fő kódját belátható időn belül külsős csapatra bízná – nem beszélve arról a nehézségről, ha ez a kód velejétől *DirectX* specifikus. Szerencsére bőven akad életképes ellenpélda, a kereskedelmi szemlélettel dolgozó „porterek” (*Loki*, *Hyperion*)anyagain felül példaként említhető az előbb említett *Homeworld*, de akár a *FreeSpace2*-t is megnevezhetem (mely az *Icculus* egyik projektjeként változott át). Utóbbi esetekben köszönettel tartozunk a fejlesztőknek és kiadóknak is, a forráskód önzetlen megnyitásának ügyén. Rendkívül érdekes azt a folyamatot megfigyelni, ahogyan az átírt játékok lassan, de biztosan szaporodnak: bár a *Loki* bő négy évvel ezelőtti bezárása nem volt túlzottan jó előjel, a jelen történései úgy gondolom bíztatóak.
A reményteljes folyamat kimenetele nagyon sok dologtól függ, többek között természetesen attól is, hogy az elkövetkező néhány évben a szabad rendszerek az asztali szegmensben milyen részesedésre lesznek képesek szert tenni. Az sem mellékes, hogy a *Win32/DirectX* rendszerkörnyezet, és ennek marketing munkálatai milyen eséllyel lesznek képesek a programozók zömét továbbra is kizárólagosságra késztetni (sajnos ez a lehetőség még mindig valós veszélyforrásként lebeg előttem).

A teljes szabadság

A harmadik kategória cikkem írásakor (múltjához mérten) már egy másik „dimenziót” kóstolgat: az olyan szabadon elérhető projektek, mint a *Glest*, a *Racer*, vagy a *FlightGear* az idő múlásával mind komolyabb alternatívát fognak jelenteni a játékok között, mely megoldások jelenléte (előbb vagy utóbb) akár a két előző csoport ellenében is érezhető lesz. Ebben a csoportban a kisebb, egyszerűbb próbálkozásokon keresztül jó néhány olyan elképzelés is fellelhető a világhálón, melyeket az előbb példaként említettem, és amelyek állapotukat tekintve már most sokkal többet ígérnek, mint egy átlagos platformfüggetlen „másolat”. Mivel eme régió megvalósításainak programkódja gyakran mindenki által elérhető, így nem ritkán a publikálást követően akár *OS X*-en futó bináris verziók is felbukkannak néhány nap elteltével, jellemzően szakértő „kezek” által alakítva.

Mit hozhat a jövő?

Nem győzöm hangsúlyozni, a rejtély kulcsa a platformfüggetlenség. A cikk írása előtt néhány nappal töltöttem le a néhai *Quake2*-ből készített *Jake2*-t, melyben fő változásként az egész játék *Java* értelmezőn fut – így gyakorlatilag minden rendszeren működik, és mindenütt ugyanazt nyújtja. Mindezt a *Quake2* nyitott forráskódjából készítette egy német csapat, olyan önzetlen munkát végezve, ami által már nehéz olyan asztali operációs rendszert mondani, ahol e népszerű játék módosított kódja ne futna (holott az *id Software* gyermeke már eleve multiplatformos kód, bár nem ennyire széles használhatósággal).
A jövő remélhetőleg a fejlesztők ez irányú felismeréséről fog szólni, mivel független kódjaikkal saját piacukat bővíthetik tovább, ami minden bizonnyal mérhető profittal (és szakmai elismeréssel) kecsegtet. Márpedig a *Linux* (vagy éppenséggel a *BeOS*, *OS X*) az otthoni, asztali gépek piacán mind gyakrabban köszön vissza. Mindezen felül rendkívül érdekes megfigyelni a játékprogramok és az asztali *PC* szegmens közötti kölcsönhatást is: a *Linux* (és az egyéb alternatívák) mind erősebb terjedése nagyobb platformfüggetlenséget szül, miközben a független játékok megkérdőjelezhetetlenül segítik a rendszer asztali elterjedését. Személy szerint bizakodó vagyok: habár az alagút végét esélyem sincs meglátni, de részemről bizalmat szavazok annak a közösségnek, mely szabadon elérhető forráskódú játékokat készít nekünk, vagy éppen a mind hatékonyabb függetlenség modelljén dolgozik.

Kovács Zsolt (kovi@linuxforum.hu)

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

Pingvinkiállítás – A fogpiszkálótól a vibrátorig

Azt mondják, egy átlagos, egészséges, kiegyensúlyozott felnőtt reggel fél nyolckor felkel és pocsékul érzi magát. Hogy egy átlagos felnőtt pingvinrajongó miképpen ellensúlyozza e kedélyállapotot, vagy éppen mik születnek e kedélyállapot nyomán – álljon itt néhány példa és fotó szemléltetésképpen.

Lássuk, milyen felnőtt személyek tartozhatnak a pingvinekért rajongók táborába: Itt van például a vegetáriánusok egy csoportja, kik az állatokat szeretik, a növényeket utálják, így madármásolatot fogyasztanak marcipánba öntve, tortát helyezve annak ülepe alá.

(Kevésbé állatbarát, katonás típusok homlokon szúrt pingvinkatonákat csemegéznek.) Aki „számít”, minimum herendit tart e szárnyasfajtából asztalán; megszállott La Marche de L'Empereur-nézők pedig vitrinbe zárva csodálják nap mint nap mama- vagy papapingvint és kitömött sarját. Mivel az emberek ízlését alábecsülni nem lehet, hamisítványként vagy nagyon ügyes eredetiként születhettek a különböző anyagú és színű ízék, melyek funkciója valószínűleg hol az elrémülés, hol a harsány kacaj

kikényszerítése. E kettő ötvöződik a szemléltetett pingvinvázában… Noha sokan úgy vélik, jobb egy pillanatig gyávának lenni, mint halottnak életünk végéig, az extrémebb sport- és madárrajongók merészen emelkednek az égbe nagymadár-léghajón, majd csőrrel lefelé ereszkednek-zuhannak alá sárkányrepülővel. Az elemek sorában következzék a víz: bájos kis szökőkút formálható csőrből köpködő kőpingvinből, s a H_2O-t csinos kis ötujjas kifordított (pingvin)-bundakesztyűvel súrolhatjuk szét bőrünkön. Joan Rivers amerikai színésznő azt állítja, férje szeretkezés előtt fájdalomcsillapítót vesz be – talán hasonló áldozatok és fent említett rajongások nyomán készülhetett el ama pingvin formájú szexuális segédeszköz, mely többféle extrával bír, mint például vízállóság (lehet, hogy szökőkúthoz passzítva?), zselés, illatos, forgós-

fémgolyós kivitel, felcsatolható bevitel stb. (Némelyek szerint e madárvibrátor a brit nyuszifüles változatot szorította ki, melyet „a köz biztonsága érdekében” tiltottak be sérülések miatt…) Feltehetően a legjobb ötletek viccnek indulnak (szerintem többnyire azok is maradnak), és sorolhatnám még a pingvin-remekeket, de bevallom őszintén, a legtöbbet látván nem találtam szavakat…

Halusz Léna

Magas rendelkezésre állás a Novell SUSE Linux Enterprise Serverrel

Az informatikai rendszerek nem tervezett leállása jelentős veszteséget okozhat. A veszteségek nem feltétlenül csak többletkiadást jelentenek, akár több órányi többletmunkát, vagy éppen elveszett üzleti lehetőségeket is eredményezhetnek. Csak azok a vállalatok és intézmények tudhatják biztonságban informatikai rendszerüket, ahol a folyamatos, zökkenőmentes rendelkezésre állás megoldott. Ebben nyújt segítséget ügyfeleinek a *Novell*, hiszen a *Novell SUSE Linux Enterprise Server 9* kiemelten magas rendelkezésre állási (*High Availability, HA*) szolgáltatásokat nyújt, és a közeljövőben piacra kerülő *SUSE Linux Enterprise 10*-ben a cluster szolgáltatások is továbbfejlesztésre kerülnek majd. A *HA clusterek* célja, hogy több számítógép és periféria olyan módon kapcsolódjon össze, hogy azok egyetlen rendszert alkossanak, amely akkor is működőképes marad, ha valamely komponens meghibásodik.

A *HA*-szolgáltatásokat a *SUSE Linux Enterprise Server 9* szerveren futó speciális alkalmazások valósítják meg. Ezek a programok folyamatosan kommunikálnak a hálózaton található másik, redundáns szerverrel (csomóponttal), és ha a csomópont elérhetetlenné válik, áthelyezik („átterhelik") a rajta futó folyamatokat. A *HA*-szolgáltatások segítségével gyorsan, egyszerűen állíthatók be olyan rendszerek, amelyek automatikusan biztosítják a szolgáltatások redundanciáját a hardver meghibásodása esetére.

A HA-szolgáltatások lelkét a viszonylag egyszerűen beállítható *heartbeat* csomag képezi. A továbbiakban áttekintjük a HA-szolgáltatások egyes előnyeit, valamint azt, hogyan kell beállítani és kipróbálni őket. Először is üzembe helyezünk egy redundáns csomópontot, amely képes átvenni a weboldalak kiszolgálását, ha a fő csomópont leállna.

A hardver beállítása

A ***Linux HA-szolgáltatások legegyszerűbb formájában két csomópont figyeli egymást, és ha az aktív elérhetetlenné válik, a passzív átveszi a szerepét. Az egymás állapotának figyelése általában redundáns kapcsolatokon keresztül történik, hogy ha éppen az elsődleges hálózat hibásodna meg, akkor a szolgáltatások ne kezdjék átvenni egymás szerepét feleslegesen.***

A hálózati csatolók beállításakor, amennyiben ez egyáltalán lehetséges, szét kell választani a munkához és a csomópontok közötti kommunikációhoz használt hálózatot. Ezzel biztosítható, hogy a hálózati problémák (például egy kapcsológond vagy egy szokatlanul magas hálózati terhelés) ne aktiválják a szolgáltatás-átterhelést. A meghibásodási pontok számának csökkentéséhez érdemes a két csomópontot elkülönített, dedikált hálózaton elhelyezni. A legjobb megoldás egy *Ethernet*-keresztkábel és egy pár hálózati kártya használata. Állítsa be a hálózati címeket egy pont-pont hálózatnak megfelelő módon.

A szolgáltatások előkészítése

A *HA*-támogatású szolgáltatások a *HA*-inicializációs folyamat részeként kerülnek indításra, nem a szokásos rendszerinicializáláskor. A dupla elindítás kiküszöbölésére, és ami még fontosabb, annak érdekében, hogy ne induljanak el, amíg nincs rájuk szükség, vegye ki a szolgáltatásokat a normál rendszerindításból. Az ***insserv -d apache2*** parancs kiveszi az *Apache*-it (a webszervert) a normál rendszerinicializáció folyamatából. Az *Apache*-ot ehelyett az alábbiakban beállított HA-szolgáltatások fogják elindítani.

A Heartbeat csomagok telepítése

A *HA*-szolgáltatások három *heartbeat* csomagban állnak rendelkezésre (***heartbeat***, ***heartbeat-stonith***, ***heartbeat-pils***). Ezek biztosítják az alap *HA*-funkcionalitást. Ha a *YaST* (*Yet Another Setup Tool*, telepítő- és konfigurációs eszköz)használatával akarja beállítani a *HA*-t, akkor használja a ***yast2-heartbeat*** csomagot, mivel ebben benne van a *HA* beállításához szükséges *YaST*-modul.

A HA-szolgáltatások beállítása

A *HA*-szolgáltatások a *SUSE Linux Enterprise Server 9 YaST* grafikus felhasználói felületén keresztül állíthatók be; ez létrehozza és módosítja a konfigurációs fájlokat az */etc/ha.d* könyvtárban. A kívánt csomagok telepítése után indítsa el a *YaST*-ot és válassza ki a *Magas rendelkezésre állás* modult a *Rendszer* részben. A ***yast2 heartbeat*** paranccsal közvetlenül is elindítható a modul.

A HA-szolgáltatásokat úgy szokás beállítani, hogy elinduljanak a rendszer betöltésekor. Ez ugyanis probléma esetén lehetővé teszi az automatikus helyreállítást, és – opcionálisan – a redundáns szolgáltatások fölötti irányítás átvételét. Ha a HA-szolgáltatásokat még nem indítottuk el, akkor a két csomópont között nem történik üzenettovábbítás és a beállított szolgáltatások nem indulnak el. Ha megadta a *HA*-clusterben használt gépeket, adja meg, a kommunikáció módját (*heartbeat*). Jelen esetben kétféle módszert használunk: az egyik a */dev/ttyS0*

Kis költség, nagy eredmények

A *HA* a *Distributed Replicated Block Devices* (*DRBD*, elosztott replikált blokkeszközök) szolgáltatással együtt is használható, ami lehetővé teszi a fájlrendszer hálózaton keresztüli tükrözését, illetve redundáns fájlrendszerek és szolgáltatások használatát. A *MySQL* vagy *PostgreSQL* fürtözési megoldásaival párosítva a HA kiemelkedő rendelkezésre állású, cluster alapú *MySQL* és *PostgreSQL* adatbázisok használatára is módot ad.
E megoldások révén a *Linux* rendszerek rendkívül magas rendelkezésre állású adatbázis-, fájl- és egyéb szolgáltatásokat biztosíthatnak, meglehetősen alacsony áron. A *SUSE Linux Enterprise Server 9* és a hozzá tartozó *Linux High Availability* szolgáltatások használatával különösebb költségek nélkül rendszerbe állhatnak a *HA*-megoldások, amelyek egy rendszerhiba esetén mindenképpen kifizetődnek.

(soros port egy nullmodem-kábellel), a másik a hálózat. Bár további módszerek is felvehetők, ez a kettő a legtöbb konfigurációhoz elegendő.
Azért, hogy a *heartbeat*-üzeneteket ne lehessen meghamisítani vagy módosítani, használható egy hitelesítési kulcs. Kulcs nélkül az üzenetek egy egyszerű ellenőrzőösszeggel kerülnek elküldésre, amellyel ellenőrizhető az érintetlenségük, de nem rendelkeznek speciális azonosító információval.
Esetünkben a kommunikációs közeg egy biztonságos hálózat, így elég egy egyszerű ellenőrzőösszeg, ami kevésbé terheli le az erőforrásokat, de ha a *HA*-kommunikáció nem biztonságos hálózaton keresztül történik, akkor fontos egy titkos (szimmetrikus) kulcs használata. Ez a kulcs mindkét csomóponton azonos, és biztosítja, hogy csak érvényes *HA*-üzenetek kerüljenek elfogadásra.
A legbiztonságosabb, de a legtöbb erőforrást igénylő mechanizmus az *SHA1* algoritmus használata. Az ezzel az algoritmussal megadott információkat az */etc/ha.d/authkeys* fájl tárolja. (Ez a fájl mindig átmásolható a készenléti rendszerre egy *SSH*-kapcsolaton keresztül.)

Erőforrások beállítása

A legáltalánosabb *HA*-alkalmazások esetén meg kell adni az átterhelési *IP*-címet és egy átterhelési szolgáltatást. Az átterhelési *IP*-cím arra vonatkozik, hogy ha az egyik rendszer elérhetetlenné válik, akkor a partnerrendszer átveszi annak *IP*-címét. A *HA*-rendszer automatikusan kiküld egy *ARP*-csomagot, amelyben tájékoztatja az útválasztókat és a kapcsolókat, hogy az *IP*-címnek új tulajdonosa van. Több szolgáltatás esetén több *IP*-cím is használható; a *HA* automatikusan aktiválja az *IP*-címet, amint a szolgáltatás átterhelésre kerül.
Erőforrások hozzáadásakor rendelje hozzá mindegyik erőforrást egy elsődleges csomóponthoz – ahhoz, amelyen annak alapesetben el kell indulnia.
A *HA*-rendszer először az elsődleges csomóponton fogja elindítani a szolgáltatásokat. Ha egy szolgáltatás elsődleges csomópontja kiesik, akkor a partnercsomópont indítja el.
Amikor egy rendszer elérhetetlenné válik, az erőforrások a listában szereplő sorrendben indulnak el. Ez biztosítja, hogy a szolgáltatások indulásakor a függőségek, szükség esetén teljesüljenek. A jelen példában az új *IP*-címet az *Apache* elindítása előtt hozzá kell rendelni, másképp az Apache nem tudja elvégezni a megfelelő *IP*-címek hozzárendelését.
Az erőforrások beállításakor meg kell adni a *HA*-rendszernek, hogy hiba után hogyan építse fel szolgáltatásait.

A STONITH eszköz beállítása

A szolgáltatások átterhelésekor az átvevő rendszernek meg kell bizonyosodnia arról, hogy a másik fél valóban kiesett. Ennek biztosításához a *HA* kommunikálhat egy *STONITH* (*Shoot-The-Other-Node-In-The-Head*, szabad fordításban kb. „lődd tarkón a másikat") nevű eszközzel.
A *STONITH* eszköz lehetővé teszi, hogy a másik csomópont valóban lekapcsoljon, általában úgy, hogy egy szoftvervezérelt kapcsolóval megszakítja annak tápellátását. A *HA* rendszer a *STONITH* eszközök széles választékát támogatja, amelyek közül több *SNMP* protokoll használatával vezérelhető.
Ha megosztott eszközöket – például megosztott lemezt – használ, győződjön meg róla, hogy mielőtt a passzív csomópont aktívvá válik, az aktív leállt; ellenkező esetben az adatok megsérülhetnek, amikor mindkét rendszer ugyanazokat próbálja írni és olvasni.
Számos olyan eszköz létezik, amely *STONITH* műveleteket biztosít a *HA*-rendszerekhez. Ezek általában többaljzatos tápkapcsolók, amelyeket a hálózaton keresztül *SNMP*-vel vezérelnek.

HA-szolgáltatások indítása és ellenőrzése

A HA-szolgáltatások az *rcheartbeat start* parancsfájllal indíthatók és az *rcheartbeat stop* parancsfájllal állíthatók le. Ezeket először a szolgáltatások elsődleges csomópontján indítják el, majd az átterhelési csomóponton.
Ha elindultak az átterhelési csomóponton és a kiszolgáló nem kap *heartbeat*-információkat, akkor az átterhelési csomópont át fogja venni a szolgáltatásokat.
Elindulás után a */var/log/ha-log* naplófájlban találhatók információk a HA-szolgáltatások állapotáról; ne felejtsen el egy erre a naplóra vonatkozó konfigurációs fájlt hozzáadni a *logrotate* (naplóváltás) beállításaihoz! A *HA*-szolgáltatások az *insserv heartbeat* paranccsal helyezhetők el a rendszer betöltő parancssorozatában; ezt általában a HA-rendszer beállításakor szokás elvégezni.
Ha a *HA*-szolgáltatások már futnak, akkor a két csomópont a beállított csatorná(ko)n kommunikál. Mindkét csomópont meghatározza a másik állapotát, és annak alapján fogja elindítani a szolgáltatásokat. Az elsődleges csomópont naplófájljából látszik, hogy ha az elsődleges csomópont aktiválta a *HA*-szolgáltatáshoz kiosztott *IP*-címeket, majd elindította az *apache2* szolgáltatást. A második csomópont naplófájlja azt mutatja, hogy a csomópont készenlétbe került. Ebben az esetben tovább folytatja az elsődleges csomópont heartbeat üzeneteinek és állapotának figyelését, hogy ellenőrizze, elérhető-e.

Speciális HA-funkciók

A *Linux HA*-funkciói messze túlmutatnak az itt tárgyalt egyszerű átterhelési forgatókönyvön. A *Heartbeat* sokféle más művelet elvégzésére is beállítható, amellyel elérhető, hogy a hálózati szolgáltatás mindig megbízhatóan működjön.

Valgrind

A szoftverek minőségellenőre

– Mester, fagy a program!
– Valóban? Nem értem, hisz nálam működött!

A C, C++ fejlesztőknek gondolom fájdalmasan ismerősek a fenti sorok. Ugyan honnan is tudnánk, hogy hibás a program, ha nálunk rendesen működik? A megoldás: Valgrind.

A *Valgrind* egy program, mely képes megmutatni a C és C++ programok tipikus memória kezelési hibáit (puffertúlírás, döntés hozás ismeretlen tartalmú (inicializálatlan) memóriacella alapján, átlapoló `memcpy`, összekevert `malloc/delete`,...). Használatával és a hibák javításával, jó esélyünk van arra, hogy a programunk nem csak nálunk, hanem más felhasználók gépén is működni fog.
Mint a *Krusader* (kétpaneles fájlkezelő) projekt fejlesztője, igen sokszor találkozom olyan szituációkkal, hogy egyik gépen megy valami, a másikon pedig fagy. A legutolsó ilyen hiba volt a legfájdalmasabb számunkra, amikor kiadtuk az 1.60-as „stabil" változatot. Kezdetben mindenki elégedett volt, aztán egyik napról a másikra egy csomó levelet kaptunk, hogy a program fagy. A legrosszabb az volt, hogy még csak el sem indult, hanem már a bejelentkező ablak megjelenítése előtt elszállt. Mi történhetett? Egyik fejlesztőtársam végül megfejtette a rejtélyt: egy puffertúlírási hiba miatt kizárólag *GCC 3.x* alatt működött a program. Amikor viszont frissen kijött a *GCC 4.0*, az alatta fordított *Krusader* fagyott.
Még a bejelentkező ablakig sem jutott el, így teljesen használhatatlan lett. Szerencsére a felhasználói táborunk igen türelmes volt és sokan még arra is hajlandóak voltak, hogy lefordítsák az instabil változatot (amiben a hiba már javításra került). Persze ilyen lépést csak egyszer lehet meglépni. Amennyiben rendszeressé válnak a fagyások, előbb-utóbb elpártolnak tőlünk a felhasználók. Ez a hiba ráadásul olyan triviális volt, hogy a *Valgrind* azonnal jelezte volna...
A történtek után nekiugrasztottam a *Valgrindet* a *Krusadernek* és még 7-8(!) memóriahibát kijavítottam, ami nálam nem fagyott (remélem máshol sem), de fagyhatott volna. Néhol engem is meglepett, hogy a legstabilabbnak tűnő részeken is talált valamit. Azóta csönd és béke van a projekt körül és egyenlőre nincs komoly panasz rá. Ezután elhatároztam, hogy minden egyes stabil kiadás előtt legalább egyszer tesztelem a *Krusadert Valgrinddel* is, mert megéri.

A Valgrindről

A projekt neve a skandináv mitológiából ered. *Valgrind* a főbejárata *Valhallának* (a kiválasztott halottak termének). A bejáraton túl egy farkas áll őrt, mögötte pedig egy vaddisznó fej van, melyen hatalmas sas ül.
A sas szemével kilenc világ messzi tájait láthatja. Csak azok kelhetnek át *Valgrind* kapuján, akiket az őrök érdemesnek ítélnek meg. A többiek kívül maradnak. Ennyit a mitológiáról.
Maga a *Valgrind* program egy szintetikus x86-os processzor. A tesztelt program utasításait nem a mikroprocesszor értelmezi, hanem a *Valgrind*, mely eközben elvégzi a szükséges cím és memória ellenőrzéseket.
Úgy működik, mint egy emulátor. Ez persze teljesítménycsökkenést eredményez, ezért készüljünk fel arra, hogy a programjaink legalább 10-szer lassabban fognak futni és több, mint kétszer annyi memóriát zabálnak majd, mint nélküle. Mindenképpen megér egy erős, nagy memóriájú gépet használni tesztelésre.
A *Valgrindet* szinte az összes nagy linuxos projekt használta már, vagy használja: *Firefox, OpenOffice, StarOffice, AbiWord, Opera, KDE, GNOME, Qt, libstdc++, MySQL, PostgreSQL, Perl, Python, PHP, Samba, RenderMan, Nasa Mars Lander software, SAS, The GIMP, Ogg Vorbis, Unreal Tournament, Medal of Honour, RenderMan,...*
A *Valgrind* nagyon sok eszközt biztosít a szoftverek tesztelésére, cikkem viszont kizárólag a memóriaellenőrző komponensével (*memcheck*) foglalkozik.

1. Lista – Példa memória túlolvasásra (test.c)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5     /* 10 byte memória
➥foglalása */
6     char * mem = (char *)
➥malloc( 10 );
7     /* túlolvasás */
8     printf( "Char: %d\n",
➥mem[ 10 ] );
9  }
```

Tesztelés Valgrinddel

A ***Valgrind a*** program futása közben igen sok dolgot tesztel:

- Inicializálatlan memória használata
- Írás, olvasás felszabadított memóriaterületről
- Lefoglalt memória túlírása, túlolvasása
- A verem (stack) helytelen írása, olvasása
- Memória-szivárgás
- Összekevert `malloc/new/new []`, `free/delete/delete []`
- Átlapoló forrás és cél a `memcpy` függvényben

A tesztelés menete is igen egyszerű: lefordítjuk a programot és ***Valgrinddel*** futtatjuk. Mivel grafikus felület nincs, ezért parancssorból kell használni. Próbaként gépeljük be a ***test.c*** programot *(1. lista)*, mely puffer túlolvasási hibát tartalmaz, majd fordítsuk le és futtassuk ***Valgrinddel.***

```
gcc -g -O test.c -o test
valgrind --tool=memcheck test
```

Nem mindegy, hogy hogyan fordítunk. A `-g` opció `DEBUG` módú fordítást jelent. Fontos, mert a ***Valgrind*** csak így képes a hibás sor számát kiírni. A `-O` opcióval pedig az optimizációt kapcsoljuk ki. Az optimizáció azért problematikus, mert ha inline-ként fordít a fordító egy függvényt és hiba van benne, a problémás sor számát lehetetlen lesz kiírni. `O2` és magasabb optimizációk esetén GCC-vel néha hamis hibaüzenetet is kaphatunk (`conditional jump or move uninitialized depends on uninitialized value(s)`). A projekt írói nem javították ki a hibát, mert sokkal lassabb végrehajtást eredményezne, meg egyébként sincs értelme `O2`-ben ***Valgrindezni***.

A `–tool=memcheck` szintén fontos opció, mert ez állítja be a memóriaellenőrzést. Nélküle nem biztos, hogy megbízható eredményt kapnánk.

A futtatás eredményét a *2. lista* tartalmazza. Amennyiben ettől eltérő jellegűt kapunk, úgy elképzelhető, hogy valami rosszul működik a rendszerben. Ilyenkor töltsük le a legfrissebb ***Valgrindet***, fordítsuk le és használjuk azt.

Vegyük egy kicsit alaposabban szemügyre a végeredményt!

2. Lista – A Valgrindes futtatás végeredménye

```
==17199== Memcheck, a memory error detector.
==17199== Copyright (C) 2002-2005, and GNU GPL'd, by Julian
➥Seward et al.
==17199== Using LibVEX rev 1575, a library for dynamic binary
➥translation.
==17199== Copyright (C) 2004-2005, and GNU GPL'd, by OpenWorks
➥LLP.
==17199== Using valgrind-3.1.1, a dynamic binary
➥instrumentation framework.
==17199== Copyright (C) 2000-2005, and GNU GPL'd, by Julian
➥Seward et al.
==17199== For more details, rerun with: -v
==17199==
==17199== Invalid read of size 1
==17199==    at 0x8048402: main (test.c:8)
==17199==  Address 0x415D032 is 0 bytes after a block of size
➥10 alloc'd
==17199==    at 0x401A451: malloc (vg_replace_malloc.c:149)
==17199==    by 0x80483FE: main (test.c:6)
Char: 0
==17199==
==17199== ERROR SUMMARY: 1 errors from 1 contexts (suppressed:
➥13 from 1)
==17199== malloc/free: in use at exit: 10 bytes in 1 blocks.
==17199== malloc/free: 1 allocs, 0 frees, 10 bytes allocated.
==17199== For counts of detected errors, rerun with: -v
==17199== searching for pointers to 1 not-freed blocks.
==17199== checked 57,568 bytes.
==17199==
==17199== LEAK SUMMARY:
==17199==    definitely lost: 10 bytes in 1 blocks.
==17199==      possibly lost: 0 bytes in 0 blocks.
==17199==    still reachable: 0 bytes in 0 blocks.
==17199==         suppressed: 0 bytes in 0 blocks.
==17199== Use --leak-check=full to see details of leaked
➥memory.
```

A ***Valgrind*** alapértelmezés mellett a sztenderd kimenetre írja az üzeneteit. Az „==17199==" a futtatott folyamat száma (***PID***-je). Egy hibát talált a futtatás során, és 10 byte-nyi memóriát elfolyattunk, azaz nem szabadítottunk fel (leak). A tesztprogram 8. sorára ***Invalid read of size 1*** (érvénytelen 1 byte-os olvasás) üzenetet ad, ami jogos, hiszen 1 bájttal valóban túlindexeltük a puffert.

Még két tesztprogramot érdemes átnézni, az egyik a nem inicializált memória felderítését ismerteti *(3. Lista)*, ***a*** másik a felszabadított memória olvasásáról szól *(4. lista)*. Ezek a leggyakoribb ***C*** és ***C++*** hibák.

A Valgrind működése – A- és V-bitek

Ahhoz, hogy megtudjuk, mire alkalmas a program, elengedhetetlen megismerni a működését (az A- és V-biteket), ha pedig tudjuk hogyan működik, a korlátait is megismerhetjük.

A-bit (Valid-address, érvényes cím)

A *Valgrind* minden memóriacellához rendel egy A bitet, mely megmondja hogy a cella tartalma jogosan írható és olvasható-e. Ez nem mond információt a cella tartalmáról (inicializált-e), csak arról, hogy elvileg lehetséges-e írni bele és olvasni.
A cellatartalommal a V bitek foglalkoznak (lásd lejjebb).
Minden esetben, amikor a program egy memóriacellához fordul, a *Valgrind* ellenőrzi, hogy a hozzá tartozó A-bit érvényes-e. Ha nem, hibát jelez. Az írás-olvasás műveletek nem befolyásolják az A-bitek állapotát.

Hogyan állítódnak be az A-bitek?

- Amikor a program elindul, az összes globális adat A-bitje érvényes lesz.
- `malloc`/`new` után a lefoglalt terület A-bitjei érvényesek lesznek (csak azok) és `free`/`delete` esetén érvénytelenné válnak.
- Amikor a veremmutató (stack pointer) föl és le mozog, az A-bitek átállítódnak. A szabály az, hogy a veremmutató feletti A-bitek érvényesek, alatta érvénytelenek. Ennek megfelelően, amikor egy függvény befejeződik, az általa használt memóriaterület érvénytelenné válik, mert a veremmutató feljebb lép. Ez igencsak pontatlan kezelése a dolognak, mert nyugodtan felülírhatnánk mondjuk például a függvény visszatérési címét és még hibát sem kapnánk
- Rendszerhívásoknál a bitek megfelelően állítódnak be (például `mmap`).
- Opcionálisan a programunk is tud szólni a *Valgrindnek*, hogy változtassa át a biteket.

V-bit (Valid-value, érvényes érték)

A szintetikus *Valgrind* processzor szinte teljesen ugyanúgy működik, mint az eredeti, egy dolgot kivéve: minden egyes tárolt és feldolgozott bithez hozzárendel egy másik bitet, hogy a bit tartalma érvényes-e, vagy inicializálatlan. Ennek köszönhetően minden egyes bájthoz a memóriában 8 V-bit és 1 A-bit van rendelve, tehát kicsit több mint duplája lesz a memóriafogyasztás. A processzor regisztereihez is rendel V-biteket. Ez azért fontos, mert csak akkor jelez hibát a program, ha a V bitek alapján feltételes elágazáshoz ér. Addig mindenféle műveletet lehet velük végezni. Az indok

3. Lista – példa nem inicializált memória használatára (test2.c)

```
 1  #include <stdio.h>
 2
 3  int main() {
 4    /* 10 byte inicializálatlan memória */
 5    char buf[ 10 ];
 6    /* 5. elem betöltése egy másik változóba */
 7    int num = buf[ 5 ];
 8    /* hiba: elágazás ismeretlen tartalom alapján */
 9    if( num == 0 )
10      printf( "Az 5. elem tartalma 0.\n" );
11    else
12      printf( "Az 5. elem tartalma nem 0.\n" );
13  }

Eredménye:
...
==7904== Conditional jump or move depends on uninitialised
➥value(s)
==7904==     at 0x80483CC: main (test2.c:9)
...
```

4. Lista – példa felszabadított memória olvasására (test3.c)

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  int main() {
 5    /* 10 byte puffer lefoglalása */
 6    char *buf = malloc( 10 );
 7    /* kezdőérték a 7. elemnek */
 8    buf[ 7 ] = 23;
 9    /* felszabadítjuk a puffert */
10    free( buf );
11    /* hiba: kiolvassuk a tartalmát */
12    if( buf[ 7 ] == 23 )
13      printf( "Rendben.\n" );
14  }
...
==7972== Invalid read of size 1
==7972==     at 0x8048441: main (test3.c:12)
==7972==  Address 0x415D02F is 7 bytes inside a block of size
➥10 free'd
==7972==     at 0x401AF78: free (vg_replace_malloc.c:235)
==7972==     by 0x804843D: main (test3.c:10)
...
```

5. Lista – Példa a Valgrind hibáira (hiba.c)

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  char buf2[ 10 ];
 5  int  var = 0;
 6
 7  int main() {
 8    /* visszatérési cím olvasása (-1. byte) */
 9    char buf[ 10 ];
10    /* negatív irányba túlolvasás */
11    buf[ -1 ] = 10;
12    printf( "Buf[-1]: %d\n", buf[ -1 ] );
13    /* globális területen felülírok egy tömböt */
14    buf2[ 10 ] = 20;
15    /* a Valgrind egyikért sem szól! */
16  }

Eredmény:
nincs hiba
```

egyszerű: a jól megírt programokban is vannak olyan esetek, hogy inicializálatlan területeket olvasunk és írunk. Képzeljünk el egy struktúrát, amely 2 `int`-ből és 1 `char`-ból áll és a `char` középen van. Mivel a fordító az `int`-eket gyakran 4-8 bájtra igazítja, a középső 1 bájtos `char` után 3-7 bájt üresen maradhat. A struktúra `memcpy`-val történő másolása esetén az üres területek is másolásra kerülnek. Baj csak akkor van, ha a középső üres bájtok tartalma alapján döntést hozunk.

A V-bitek ellenőrzése 3 helyen történik:

- Memóriára hivatkozás esetén
- Feltételes elágazásoknál (`if`)
- Rendszerhívásoknál (például `getcwd`)

Ezekben az esetekben, ha az érték nincs inicializálva, hibát kapunk, majd a problémát okozó V-bitek érvényesre állítódnak, hogy többször ne jelezze ugyanazt a hibát.

A Valgrind hibái

A ***Valgrind*** sajnos korán sem tökéletes. Ahol hibát jelez, akkor ott hiba is van, de ha nem jelez semmit, az még nem jelenti azt, hogy a tesztelt alkalmazás jól működik. A ***hiba.c*** tesztprogram *(5. lista)* szemlélteti azokat a szituációkat, melyeket a ***Valgrind*** rendre elnéz. A legnagyobb hiányossága, hogy tömb indexeléseket nem ellenőrzi rendesen. Ez komoly probléma. Más kereskedelmi programok

6. Lista – Példa felszabadított memória olvasására (test4.c)

```
valgrind --tool=memcheck --leak-check=full test4

 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  char * notLeaked = 0;
 5  char * possiblyLeaked = 0;
 6
 7  int main() {
 8    char * pointer;
 9    /* 10 byte lefoglalása és eltárolása */
10    /* nem veszett el, mert mutat rá
➥pointer */
11    notLeaked = (char *)malloc( 10 );
12    /* másik 10 byte lefoglalása */
13    /* mutató csak a terület közepére mutat,
14      tehát valószínűleg elveszett */
15    possiblyLeaked = (char *)malloc( 10 ) + 5;
16    /* 10 byte lefoglalása és felszabadítása */
17    pointer = (char *)malloc( 10 );
18    free( pointer );
19    /* 10 byte lefoglalása, semmi nem mutat rá
➥*/
20    malloc( 10 );
21    /* 10 byte lefoglalása, csak lokális
➥változó
22       mutat rá, ami kilépéskor eltűnik.
➥A memóriában
23       viszont szemétként az értéke megmarad
24       (érvénytelen A-bittel). A Valgrind nem
➥jelez
25       hibát, tehát itt rosszul működik! */
26    pointer = (char *)malloc( 10 );
27  }

==7709== ERROR SUMMARY: 0 errors from 0 contexts
➥(suppressed: 13 from 1)
==7709== malloc/free: in use at exit: 40 bytes
➥in 4 blocks.
==7709== malloc/free: 5 allocs, 1 frees, 50
➥bytes allocated.
==7709== For counts of detected errors, rerun
➥with: -v
==7709== searching for pointers to 4 not-freed
➥blocks.
==7709== checked 57,568 bytes.
==7709==
==7709== 10 bytes in 1 blocks are definitely
➥lost in loss record 2 of 4
```

6. Lista – folytatás

```
==7709==    at 0x401A451: malloc
➥(vg_replace_malloc.c:149)
==7709==    by 0x8048437: main (test4.c:20)
==7709==
==7709==
==7709== 10 bytes in 1 blocks are possibly lost
➥in loss record 3 of 4
==7709==    at 0x401A451: malloc
➥(vg_replace_malloc.c:149)
==7709==    by 0x804840F: main (test4.c:15)
==7709==
==7709== LEAK SUMMARY:
==7709==    definitely lost: 10 bytes in 1
➥blocks.
==7709==      possibly lost: 10 bytes in 1
➥blocks.
==7709==    still reachable: 20 bytes in 2
➥blocks.
==7709==         suppressed: 0 bytes in 0
➥blocks.
==7709== Reachable blocks (those to which
➥a pointer was found) are not shown.
==7709== To see them, rerun with:
➥--show-reachable=yes
```

úgy csinálják, hogy fordítási időben extra sorokat raknak a programba indexek ellenőrzésére. Ez sajnos nagyon hiányzik a ***Valgrindből***, de még enélkül is rengeteg hiba felderítésére alkalmas. Ha van pár ezer eurónk megvásárolhatunk olyan termékeket, melyek sokkal jobb eredményt adnak, de amennyiben nincs erre lehetőség, meg kell elégedni ezzel a tudással. A munkahelyemen van ***Rational Purify*** licenszünk, így módomban állt a ***Valgrindet*** a ***Purify***-jal összehasonlítani. A meglepő az volt, hogy néhol a ***Valgrind***, néhol a ***Purify*** adott jobb eredményt, de nem mondhatnám egyértelműen, hogy a kereskedelmi ***Purify*** (ami szintén jó szoftver), jobban teljesített volna.
Másik hibája a ***Valgrindnek***, hogy kizárólag ***x86***-os 32 bites ***Linuxon*** fut, sem ***Windows***, sem ***Solaris***, sem egyéb operációs rendszereken nem megy.
A ***Valgrind*** gyorsan fejlődő projekt. Gyakran jelennek meg új verziók és a személyes véleményem az, hogy továbbra is fejleszteni fogják, mivel egyre több nyílt forrású projektnek lesz szüksége minőség-ellenőrzésre. Ebben a tekintetben pedig jelenleg egyedülálló. Remélhetőleg a következő verziókban már alaposabban fog tesztelni.

Memóriahasználat ellenőrzése

Miután a programunk összes fagyást okozó hibáját eltüntettük, tovább is javíthatunk a minőségen. Mert ugye jó az, ha megbízhatóan fut, de vajon miért is van 150 MB memóriára szüksége? A memóriahasználat helyrerakása persze már kevésbé fontos, mint a fagyások kijavítása. Nem minden programnak van szüksége komoly memória-ellenőrzésre.
Ha például egy ***ICQ*** kliens minden üzenetnél elpazarolna 1k-t, akkor is 1000 üzenet kellene ahhoz, hogy 1 megát elszivárogtasson. 1 mega pedig a mai gépek mellett nem jelentős memória. Szerver alkalmazások esetén már más a helyzet. Ott 10 bájt elherdálása is végzetes lehet, amennyiben ez sokszor (több milliószor!) történik. Általánosságban azokat a részeket kell leellenőriznünk, amelyeken a program rengetegszer végigfut.
A ***Valgrind*** sokat segíthet az eltűnt memória felkutatásában, mert pontosan megmondja, hogy hol foglalódott le és hogyan.
Háromféle memória-szivárgást különböztet meg:

- Biztosan elveszett (***definitely lost***), akkor ha a memória nem lett felszabadítva és rá mutató pointert sem talált sehol a memóriában.
- Valószínűleg elveszett (***possibly lost***), ha ugyan talál mutatót a memóriablokkra, csak az nem a blokk elejére, hanem máshova mutat
- Még elérhető (***still reachable***), ha talál a memóriában mutatót az adott blokk elejére

Amennyiben szeretnénk megtudni, hogy hol tűnt el a memória, akkor indítás előtt adjuk meg a `--leak-check=full` opciót is.
Példát a memória-szivárgásra a *6. lista* mutat.

Végszó

A ***Valgrind*** szerintem nagyon jól program és ha lehetőség van rá, érdemes használnunk. Bár minden előforduló hibát nem mutat meg, de azért nagyon sok helyen figyelmeztet, ha baj van. A hibák javításával szoftvereink minőségét jelentősen megnövelhetjük. Sajnos vannak olyan hibák is – nem is kevés – amelyeket nem mi okozunk, hanem más rendszerkomponens (***KDE***, ***X***, ***QT***,...). Ezekkel a hibákkal nem sok mindent tudunk kezdeni, legfeljebb örülhetünk, ha nem fagy miatta a program. A lényeges az, hogy amit mi csinálunk, azt ellenőrizzük le és javítsuk, ha rossz.
A ***Valgrind*** rengeteg más hasznos funkcióval is rendelkezik (heap ellenőrzés, holtpont vizsgálat, processzor cache elemzés,...), melyeket hely hiányában nem fejtettem ki. Akit komolyabban érdekel, elolvashatja a 184 oldalas dokumentációját. Egyszer legalább érdemes átfutni rajta, mert nagyon sok érdekes apró részletet elárul. Sok sikert kívánok a ***Valgrind*** használatához és még több jó minőségű szoftvert ***Linux*** alá!

Karai Csaba
(cskarai@freemail.hu)

Informatikus vagyok egy mobiltelefonokkal foglalkozó vállalatnál, szabadidőmet legszívesebben feleségemmel töltöm, de szeretek focizni, táncolni, biciklizni és görkorizni is.

SDL – Multimédiás programozói könyvtár (2. rész)

Az előző részben próbáltunk átfogó képet kapni az SDL rendszerről, valamint belekezdtünk a video funkciók igen gazdag családjának használatába. A mostani cikkben még mindig a képi megjelenítéssel fogunk foglalkozni. Megnézzük, hogyan is lehet képeket betölteni és menteni SDL használatával, de fel fog bukkanni pár új eseménykezeléssel kapcsolatos újdonság is.

Amit szeretnénk

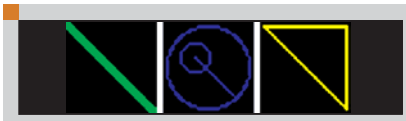
Lássuk mit is szeretnénk ez alkalommal alkotni. Készítsünk egy olyan *SDL* alkalmazást, melyben az egér segítségével rajzolni tudunk, lehetőleg többféle ecset segítségével, lehessen törölni a rajzfelületet, valamit elmenti annak tartalmát. Oldjuk meg azt is, hogy az alkalmazás tetszés szerint teljes képernyőn illetve ablakban fusson.

Ahogyan szeretnénk

Maradunk a *C++* és *SDL* ötvözeténél. Az előző számban már láthattuk, hogyan is kell az *SDL* rendszert inicializálni, ezért erre már nem térünk ki különösebben. Mindenképpen szükségünk lesz eseménykezelésre is, amiből szintén kaptunk már egy nagyon kicsi ízelítőt. Most az egérkezelés fog előtérbe kerülni, de nem felejtjük el a billentyűzetet sem.

Előkészületek

Mindenek előtt készítsük el az ecseteket, valamilyen rajzprogram segítségével. A kép legyen 32x32 képpont, fekete háttérrel, és rajzoljunk rá valamit, ami szerintünk megfelel majd az ecsetünk mintázatának. Lényeges, hogy az ábrát *BMP* formában mentsük (*1. ábra*).



1. ábra Az ecsetek

Kezdhetjük

Most, hogy minden forrásunk megvan a programozáshoz, kezdjük el a munkánk nagyobbik és komolyabb részét. Először is szükségünk lesz egy rajzfelületre, ez esetben 800x600-as felbontás már igen kényelmes méretnek bizonyul. Szükség lesz még ezenkívül még egy `SDL_Surface`-re, amire az ecseteket fogjuk tölteni. Rajzoláskor ezt a felületet fogjuk „átbillenteni” a rajzfelületünkre, ott ahol éppen az egerünk jár. Mivel a rajzunkat úgy készítettük, hogy a háttér fekete, ezért erre állítunk be egy színkulcsot, ezen a felületen, ahol az aktuális ecset ábrája van. Ez azért lesz jó mert mi csak az ábrával szeretnénk rajzolni, a háttér számunkra felesleges, tehát a fekete szín a mi esetünkben átlátszó kell hogy legyen. Ezt valósítjuk meg az `SDL_SetColorKey` függvénnyel. Első paramétere, annak a felületnek a mutatója melyre a kulcsot szeretnénk „ráhúzni”. Második paraméterként flageket vár.
A mostani flagek:
`SDL_SRCCOLORKEY,SDL_RLEACCEL`.
Az első tájékoztatja az *SDL*-t, hogy ez a felület forrásként fog szolgálni az átbillentés (egyik felület másolása a másikra) során. A második flag, pedig a billentés gyorsításáért felelős. Harmadszor pedig azt a színt adjuk meg melyet nem szeretnénk látni az `SDL_MapRGB` függvénnyel (erről már volt szó az előző cikkben). A kódunk tehát az *1. Listában* látható módon alakul. A `brush1,brush2,brush3` változók tárolják az egyes ecsetekhez tartozó elérési utakat. A `saveto` változó pedig annak a fájlnak a nevét melybe, majd menteni fogjuk a rajzfelületet. Kezdetnek töltsük be a `brush_surface`-re az

1. Lista – Kezdjük a rajzoló alkalmazást – felületek, inicializálás

```
#include <iostream>
#include "SDL.h"
int main()
{
    SDL_Surface *sdl_surface;
    SDL_Surface
    ➥*brush_surface;
    SDL_Event sdl_event;
    bool main_loop_exit =
    ➥false;
    char *saveto =
    ➥"sdl_surface.bmp";
    char *brush1  =
    ➥"brush1.bmp";
    char *brush2  =
    ➥"brush2.bmp";
    char *brush3  =
    ➥"brush3.bmp";

   // inicializációs rész
   brush_surface =
   ➥SDL_LoadBMP(brush1);
    SDL_SetColorKey
    ➥(brush_surface,
    ➥SDL_SRCCOLORKEY |
    ➥SDL_RLEACCEL,
    ➥SDL_MapRGB
    ➥(brush_surface->
    ➥format,0,0,0));
...
```

2. lista – Eseménykezelő ciklusunk

```
...
 while (!main_loop_exit)
 {
   while (SDL_PollEvent
   ➥(&sdl_event))
       {
           switch
           ➥(sdl_event.type)
           {
       case SDL_KEYDOWN:
       ...
       case SDL_MOUSEBUTTON:
       ...
       case SDL_MOUSEMOTION:
       ...
       default: break;
     }
   }
 }
```

3. lista – Billentyűk azonosítása

```
...
if (sdl_event.key.keysym.sym
➥== SDLK_1)
                    {
          brush_surface =
          ➥SDL_LoadBMP
          ➥(brush1);
          SDL_SetColorKey
(brush_surface,
➥SDL_SRCCOLORKEY |
➥SDL_RLEACCEL,
➥SDL_MapRGB(brush_surface->
➥format,0,0,0));
                    }
if (sdl_event.key.keysym.sym
➥== SDLK_2)
                    {
    brush_surface =
    ➥SDL_LoadBMP(brush2);
    ...
                    }
...
```

4. lista – Egérmozgás

```
...
case SDL_MOUSEMOTION:
       x =
       ➥sdl_event.button.x;
       y =
       ➥sdl_event.button.y;
       if (SDL_GetMouseState
       ➥(NULL,NULL) &
       ➥SDL_BUTTON(1))
       {
          paint(x, y,
          ➥brush_surface,
          ➥sdl_surface);
           SDL_UpdateRect
           ➥(sdl_surface, x,
           ➥y,32,32);
        }
        break;
...
```


`brush1.bmp`-t. Az *SDL* a *BMP* képek betöltésére rendelkezik egy külön `SDL_LoadBMP` nevű eljárással, mentésére pedig analóg módon az `SDL_SaveBMP` használható.

Események

Az alkalmazást végül is inicializáltuk. Most a nagy eseménykezelő ciklus fog következni, mely hasonló az elő cikkben szerepelt példához, csak most egy kicsit bővíteni fogjuk. Oldjuk meg azt, hogy az alkalmazásunk csak az *ESC* gomb megnyomására lépjen ki. Most a *2. Listában* látható módon így fog alakulni a ciklusunk.
Az `SDL_KEYDOWN` ágon, egyértelműen a billentyűzetet fogjuk kezelni, az `SDL_MOUSEBUTTON` ág az egérgombokat fogja figyelni, amíg az `SDL_MOUSEMOTION` az egér mozgására fog koncentrálni. Lássuk hát, hogyan fog kilépni a program *ESC* billentyűre. A ciklusunk feltételét a `main_loop_exit` változó képviseli. Elég ennyi a megoldáshoz az `SDL_KEYDOWN` ágon:

```
main_loop_exit =
➥(sdl_event.key.keysym.sym ==
➥SDLK_ESCAPE)
```

A lenyomott billentyűket az `esemeny.key.keysym.sym` változó alatt érhetjük el. Minden billentyűnek van egy konstansa az *SDL*-ben, például `SDLK_ESCAPE`, `SDLK_a`, `SDL_b`, `SDL_1`,... Az `SDL_Key` címszó alatt megtalálható az ide vonatkozó táblázat a dokumentációban. A teljes képernyő és ablakos mód közötti váltás is ezen fog alapulni:

```
if (sdl_event.key.keysym.sym
➥== SDLK_f)
➥SDL_WM_ToggleFullScreen
➥(sdl_surface)
```

Az `SDL_WM_ToggleFullScreen` eljárás az adott felületre váltogatja a teljes képernyős és ablakos módokat. Legyen például az 1-es billentyű az *brush1.bmp*-t használó ecsetünk és így rendre a többi (*3. Lista*). Elvárjuk azt is, hogy a képernyő letörölhető legyen. Ezt mondjuk társítsuk a *D* billentyűhöz. Az `SDL_FillRect` eljárás segítségével az első paraméterként megadott felületet a következő három paraméterben megadott RGB értékekkel tölthetjük fel. Töröljük hát a „rajzlapunkat”:

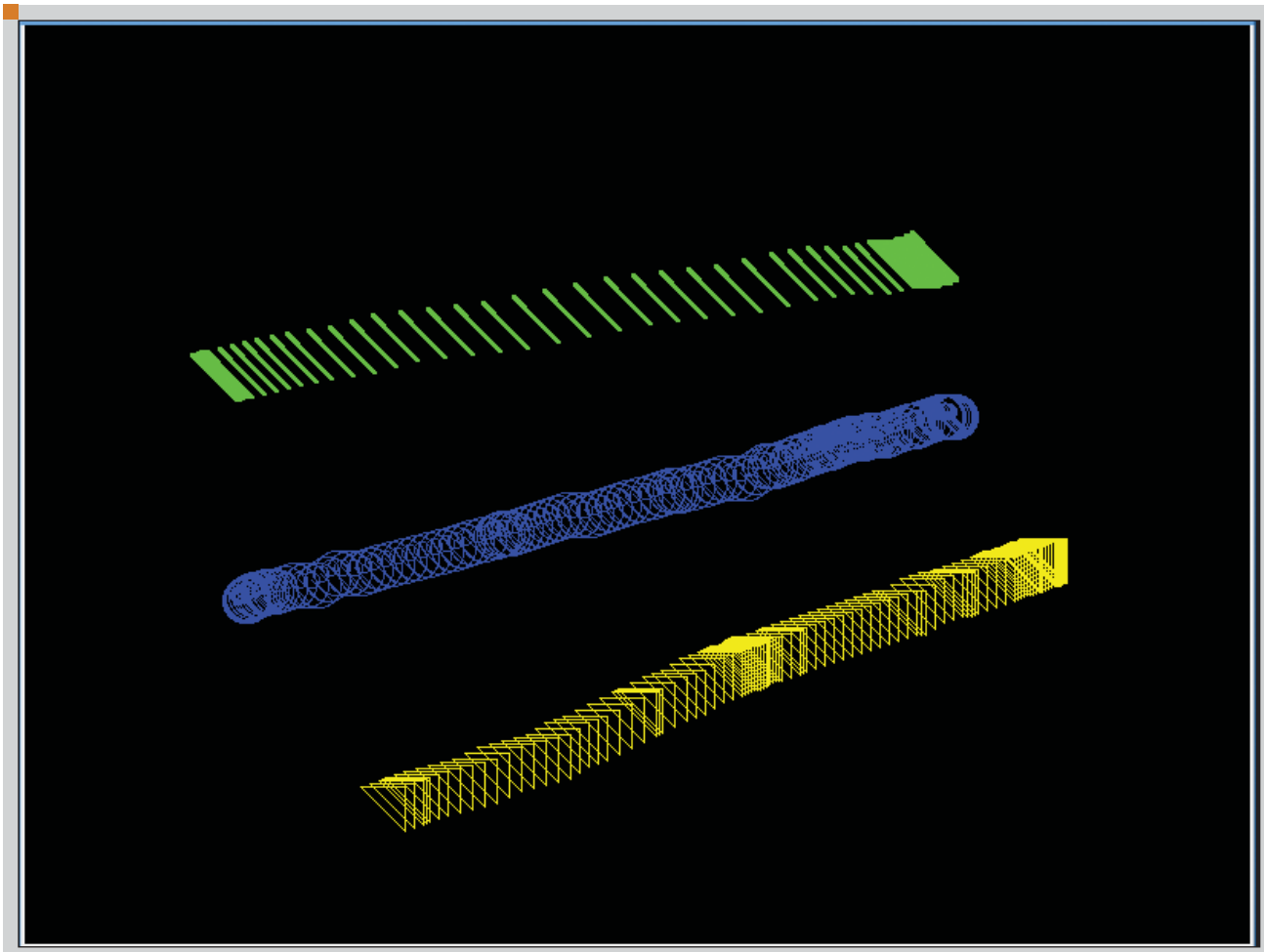
```
SDL_FillRect(sdl_surface,0,0,0)
```

Ne felejtsük el a felületet frissíteni az `SDL_UpdateRect`-el!
Maradt még a kép elmentése. Alkalmazzuk erre az *S* gombot. Az `SDL_SaveBMP` egy `SDL_Surface`-t és egy fájlnevet vár paraméterként, amibe a felület tartamát lementheti:

```
SDL_SaveBMP(sdl_surface,saveto)
```

Egér

Térjünk most át az egérre. Az `SDL_MOUSEMOTION` esemény aktiválódik, ha az egeret megmozdítjuk. Közben már csak figyelnünk kell, hogy le van-e nyomva a bal egérgomb vagy sem. Rajzolunk ha igen (*4. Lista*). Hogy hol is jár éppen az egér azt az `sdl_event.button.x`, `sdl_event.button.y` változók fogják nekünk megmondani. A feltételben vizsgáljuk, hogy le van-e nyomva a bal egérgomb ha igen, akkor rajzolunk. A `paint` eljárás fogja az ecsetet tartalmazó felületet átbillenteni a „rajzlapra” (*5. Lista*).
Ami új az az `SDL_BlitSurface`. Az első két paraméter egy felület és egy `SDL_Rect` típus. Az `SDL_Rect` egy négyszögterületet tartalmaz, *x,y* a koordináták, w a szélesség, h a magasság. Ha éppen ez `NULL` mutató, akkor



2. ábra Az egyes ecsetek mintái

az egész felület másolásra kerül. A második két paraméter is egy felület-rect páros. Ez már a célfelület és a cél terület a felületen. Itt a `NULL` mutató a `rect` érték helyén, azt eredményezi, hogy a forrás felület a célfelület 0,0 koordinátájában fog megjelenni.

5. Lista – Az ecset kirajzolása megadott pozícióba

```
void paint(Sint16 x,Sint16 y,
➥SDL_Surface
➥*src,SDL_Surface *dst)
{
    SDL_Rect *dst_rect = new
    ➥SDL_Rect;
    dst_rect->x = x;
    dst_rect->y = y;
    SDL_BlitSurface
    ➥(src,NULL,dst,dst_rect);
}
```

Mi most csak az ***x,y*** koordinátákat adtuk meg, mivel az `SDL_BlitSurface` csak ezeket veszi figyelembe a struktúrából. Tehát az ecsetet tartalmazó kép a rajzlap x,y koordinátájára fog kerülni. A koordinátákat pedig már az egér eseménykezelője fogja átadni. A `paint` után természetesen frissíteni kell a „rajzlapot”. Rendben is volnánk, de ez a kezelő csak akkor rajzol, ha mozgatjuk az egeret. Mi van ha csak egy kattintásra szeretnénk rajzolni. Erre fogjuk alkalmazni az `SDL_MOUSEBUTTON` ágat. Ide ugyanazt a kódot írjuk, amit az `SDL_MOUSEMOTION` ághoz, csak a feltételt elhagyjuk.
Ha minden rendben ment, most van egy alkalmazásunk, mely megfelel a cikk elején említett elvárásoknak. Megismerkedtünk közelebbről a billentyűzet és az egér eseménykezelésével, valamit láttuk, hogyan kell betölteni illetve menteni egy ***BMP*** képet. Alkalmaztuk az `SDL_BlitSurface`-t, ami alapvető minden grafikus ***SDL*** alkalmazásban, segítségével másolhatunk egy felületet a másikra. Láttunk példát színkulcshasználatára, mellyel átlátszóvá tehetünk egy bizonyos színt egy felületen.
A következő részben az ***audio*** eszközökkel fogunk ismerkedni, valamit a ***CD-ROM*** kezelésbe is fogunk kicsit vájkálni, valamit hátra van még az ***időzítő*** funkciók használata. Remélhetőleg hasznos információkkal szolgált a sorozat ezen része is mindazoknak, akik most kezdenek érdeklődni az ***SDL*** iránt. Jövő hónapban folytatjuk!

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.

PHP nyomkövetők (1. rész)

„Kössük össze a Föld számítógépeit egy hálózattá... A Föld forgásától a vezetékeken hamarosan megindul a szoftverek áramlása” (Eric S. Raymond: A katedrális és a bazár). Sorozatunkban a PHP nyelvű programok hibajavításáról, teljesítményelemzéséről, gyenge pontjainak felderítéséről lesz szó, valamint különböző alkalmazásokról, amik ezt megkönnyíthetik.

A *PHP* nyelv egyik fő előnye és sikerének kulcsa (valamint sebezhetőségének gyökere), hogy szabad volta és egyszerűsége miatt amatőrök is nekiállhatnak egy-egy szkript megírásának. Így voltam ezzel magam is. Komoly motivációt szolgáltat e nyelv megtanulásához, hogy az ingyenesen elhelyezhető/felhasználható webes programok gyümölcseit sokan élvezhetik (lásd ***Web 2.0, Linuxvilág #63.***). A pozitív visszajelzés a szabad szoftver „fizetőeszköze”. Szerencsés, ha elég idő áll rendelkezésre: a programok nyugodt megtervezését ugyanis semmi nem pótolja; ha segítőkész emberek és megfelelő feladatok vannak a (virtuális) környezetben, a kezdőkből idővel óhatatlanul haladók lesznek egy-egy témában – azzal együtt, hogy az autodidakta módszereknek megvannak a maguk hátrányai, maradnak bőven fehér foltok a módszerek terén.

Hogyan és milyen alkalmazások segítségével lehet elmélyedni, hibát keresni *PHP* programokban, azaz hogyan lehet „debuggolni” őket? A ***debug*** szótő valószínűleg a számítógépek (h)őskorszakáig vezet vissza, amikor egy hatalmas termet betöltő gépezet elektroncsövei között kellett hibá(sa)t keresni, és az egyik ilyen hiba oka egy odakerült bogár (***bug***) volt. A kényelmes nyomkövetés azonban nemcsak a szó szoros értelmében vett hibák felkutatására jó, hanem egy-egy „újrahasznosított” (például *PECL* vagy *PEAR*) program hatékonyabb, gyorsabb megismerésére is. Ebben az esetben (és a fejlesztés bizonyos szituációiban) lehet létjogosultsága annak, hogy a webszerver a helyi gépünkön legyen. Általában azonban olyan helyzeteket fogunk vizsgálni, amikor a webszerver egy távoli gépen helyezkedik el, ahová csak biztonságos (***SSH***) kapcsolaton át van lehetőségünk belépni, sőt, ahonnan esetleg a rendszergazda le is tiltotta a belső hálózatunk felé irányuló *SSH*-kapcsolatok nyitásának lehetőségét is. A nyomkövetők másik fontos szolgáltatása általában a teljesítményelemzés, azaz a szűk keresztmetszetek, időigényes részletek keresése, a *„profiler”* mérések végzése.

A „házi” hibakeresés módszerei

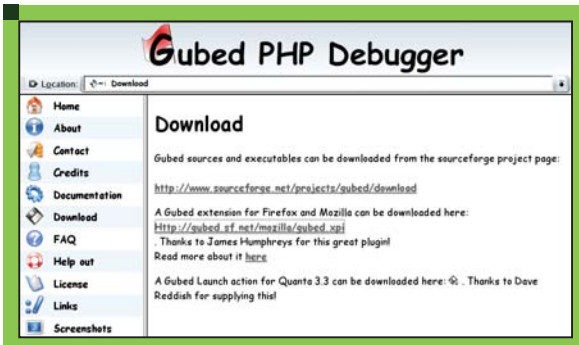
Legtöbbünknek ilyesfélék jutnak eszébe, amikor valami kisebb(nek hitt) hiba merül fel egy *PHP* programban: tegyünk be egy `echo 'itt tartunk';` részletet, hogy eljut-e odáig az interpreter. Írassunk ki egy-egy változót, például a `print_r($tomb);` vagy a `var_dump($egyeb);` segítségével. Ez utóbbi az értékek mellett típusokat is mutat, bár pont emiatt zsúfoltabb, kellemetlenebb is a látvány. Ha nem akarunk állandóan „oldal forrását” kérni, akkor célszerű odatenni ezen parancsok elé és után egy `<pre>` és `</pre>` *HTML*-kulcsszót. Ezek a kézi megoldások gyorsak, ha tudjuk, hol kell keresni a hibát, verziófüggetlenek, és nem igényelnek rendszergazdai jogokat.

Azonban ezeknek hátrányai is vannak: átfogó vizsgálathoz sok helyre kell beszúrni a fenti parancsokat, ami veszélyes lehet, ha elfelejtjük kiszedni őket, és a „kikommentezés” (megjegyzéssé változtatás) zavarja a kód képét. (Bár ez előny is lehet, mert a távoli jövőben is láthatjuk majd, hogy itt már kerestünk hibát korábban.) Nemigen lehet futás közben próbálkozni (pédául egy-egy ciklusbeli vagy bemeneti) változó megváltoztatásával. Ami talán a legnagyobb hátrány, hogy ezekkel a módszerekkel meghiúsulhat a *HTTP*-fejléc küldése, ami átirányításkor (redirect) vagy munkamenetek, illetve „sütik” (cookie-k) szervezésénél jelenthet gondot.

„Célszerszámok” a PHP-n belül

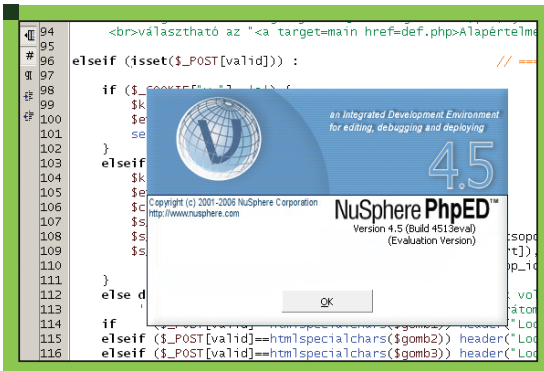
Körmönfontabb hibakeresésre is lehetőség van PHP függvények segítségével: `error_log()`, `set_error_handler()`, `user_error()`, `trigger_error()`, `assert()` stb. Ez utóbbi különösen jól használható. Érzékenysége (hogy megálljon-e a program egy-egy hibára és milyenre) jól skálázható a ***php.ini*** konfigurációs fájlban; meghálálja a megismerésére szánt időt. Célszerű a ***php.ini*** fájlt úgy beállítani, hogy fejlesztés, tesztelés közben kapjunk hírt mindenről (dönthetünk: képernyőre vagy naplófájlba?); az éles rendszer azonban legyen a lehető legcsöndesebb a hibákat illetően (és a hibákat feltétlenül naplófájl fogadja). Tegye mindezt természetesen úgy, hogy maga a *PHP* kód ugyanaz maradhasson, amikor a tesztkörnyezetből átkerül az éles környezetbe.

A *PHP*-t maximális hibajelentési szinten használva – az `error_reporting=E_ALL` beállításával – figyelmeztetést kapunk mindenről, ami gondot



1. ábra Gubed

2. ábra Quanta+



3. ábra Nusphere::PhpED

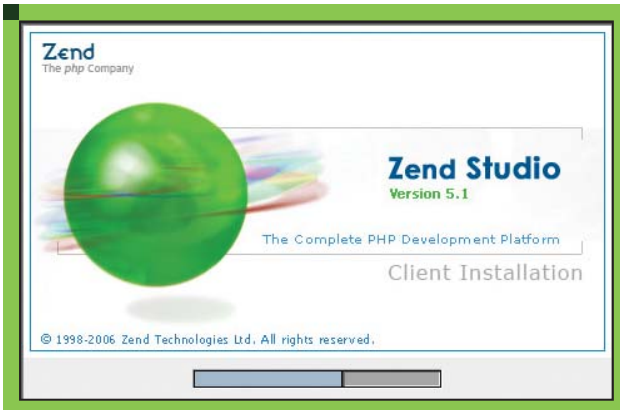
4. ábra ActiveState::Komodo

5. ábra Xored::TruStudio

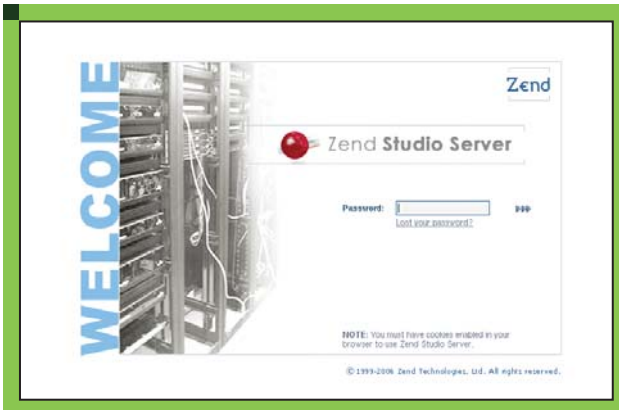
okozhat, például az előzetes érték nélküli, definiálatlan változókról, amikor is hibás adatokkal dolgozhat a program. Ez a sajátosság akár programból is állítható az `error_reporting()` függvénnyel – azonban veszélyes lehet, ha így felejtjük. Ha a program már kész bevezetésére, akkor az `E_NONE`-t használva teljesen leszigetelhető a kód a további vizsgálatoktól.

Miért is jó nyomkövetőt használni?

Az interaktív nyomkövetők esetén kényelmesen kézben tartható a futtatás. Parancsról parancsra léptethetünk, a felesleges blokkokat átugorhatjuk, töréspontokat tudunk elhelyezni a kódban, akár feltételhez kötve őket. A program belső állapotváltozásainak lépésenkénti nyomon követése szinte csak így lehetséges. Könnyen inicializálható a bemenet (`$_GET`, `$_POST`, `$argv` változók), működés közben módosítható a változókörnyezet, a hibaüzenetek és naplók a kimenettől elkülönülnek. Az érintetlen kód gyümölcse az eredeti funkcionalitás – ezt sehogy máshogy nem lehet elérni „házi módszerekkel”. Hátrányként jelentkeznek az előkészítés nyűgös teendői: a webszerveren telepíteni kell valamit – általában webszerver-modul(oka)t, *.so* fájl(ok) formájában (rendszergazdaként), mint a *PhpED* és az *Xdebug* esetében, vagy egy kisebb *PHP*-programrendszert például a *Gubed* esetén (ennek előnye, hogy ehhez nem kell rendszergazdának lenni, és nem változik meg a *PHP*-értelmező a nyomkövetés miatt). Kliens alkalmazás is kell a helyi gépen. Az az ideális eset, ha ez a kliensprogram ugyanaz, mint a fejlesztőkörnyezet maga (például a *Komodo* vagy a *Quanta*), de lehetnek előnyei puritánabb kliensprogramoknak vagy akár parancssori klienseknek is. Egy kényelmes *PHP* fejlesztőkörnyezettől azt várjuk, hogy lehessen benne kódot szerkeszteni (szintaxis-színezéssel, navigálással, kódkiegészítéssel, esetleg az összetartozó egységek takarásával/kibontásával), lehessen nyomkövetni (helyi vagy távoli gépen), legyen benne teljesítményelemző (profiler) és kódelemző alkalmazás, lehessen fel- és letölteni a távoli webszerverre/ről fájlokat (*scp*-vel), tudjon kapcsolatot tartani egyéb alkalmazásokkal, mint amilyenek például a *CSS* vagy *HTML* érvényesítők, a *PHPdocumentor* stb. A kód lefedettségét, azaz a futás közben érintett (és nem érintett!) részek feltérképezését (*code coverage*) hasznos lehet látni nagyobb programoknál. Úri huncutság, de segíthet időnként az incidens vezérelt („*just in time*”, *JIT*)



6. ábra Zend Studio Kliens



7. ábra A Zend Studio Szerver webfelületen konfigurálható

nyomkövetés, azaz: csak akkor nyílik meg a megfelelő kódrészlet a fejlesztőkörnyezetben, ha valami hiba lépett föl a futás során. Előnyös lehet még a munkamenet alapú (*per session*) hibakeresés. Nem nagy baj az sem, ha a legszükségesebb (*PHP*, *HTML*, *SQL*) kézikönyvek egy kattintással előjönnek, valamint az oktatóanyagok is hasznosnak bizonyulnak a kezdők számára.

A futtatás közben történő kódmódosítás még a jövő zenéje. Az is az igazsághoz tartozik, hogy nyomkövetés közben jóval lassabb a futás – ez azonban többnyire nem okoz gondot, és ezen valamennyit lehet segíteni jól elhelyezett (feltételes) töréspontokkal.

A hibák kijavításához kell némi gyakorlat, de ez viszonylag hamar megszerezhető. Érdemes először minden releváns hibát megszüntetni, és csak utána rágódni a kisebbeken. Arra is ráérez az ember egy idő után, hogy mekkora lépésekben érdemes a kódot hizlalni, hogy az új részletek hibái még átlátható mennyiségűek legyenek. Jó önismerettel megfelelő középutat találhatunk, hogy ne pazaroljuk az időnket a túl gyakori kipróbálgatással, de ne állítsunk elő akkora programot sem, aminek a hibái már átláthatatlan mértéket öltenek, és a kijavításukhoz szükséges idő igencsak megnő. Ilyenkor a részletes hibajelentés nagyon sokat ér.

Hasznos, ha az érvényes és hibás tesztadatokból teszthalmazokat tudunk létrehozni, és az ezekkel való tesztelést megfelelően dokumentáljuk. (Nagyobb programok dokumentálatlan tesztelése többnyire egyenértékű azzal, mintha semmit sem csináltunk volna.)

Áttekintés a nyomkövetőkről

A *PHP3* még tartalmazott hálózat-alapú nyomkövető-támogatást, de a *PHP4* és *PHP5* már nem. Ehelyett külső nyomkövetők használata ajánlott:

- A legegyszerűbb a „debuG" tükörszavaként elnevezett *Gubed* (például a *Quanta Plus* fejlesztőkörnyezet által felajánlott integrált kliensprogrammal) ➲ gubed.mccabe.nu, ➲ quanta.sourceforge.net
- jó lehetőségeket kínál a *DBG*, de a rá épülő legjobb fejlesztőkörnyezet, a *NuSphere::PhpED* sajnos a *Microsoft Windows*-os verzióban (és ehhez tartozó funkcionalitásban) jóval előrébb tart, mint a Linuxos változatban – és ráadásul időkorlátos ➲ dd.cron.ru/dbg
- a teljesítményelemzésben jár élen az *Advanced PHP Debugger* (*APD*) ➲ pecl.php.net/package/apd
- az (időkorlátos) *ActiveState::Komodo* fejlesztőkörnyezetet használhatjuk kliensprogramként az *Xdebug*-hoz, mely működését tekintve a PHP 3-hoz tartozó hibakereső utódjának tekinthető; az *Xdebug* honlapja mintaszerűen egyszerű és igényes. ➲ www.xdebug.org, ➲ www.activestate.com/Products/Komodo
- szintén *Xdebug*-ot (és *Eclipse*-et) használ java alapon a *Xored::TruStudio*; sajnos licenc után érdeklődik néhány percenként, és csak CGI interpreterrel ajánlja fel a nyomkövetést (azaz, amennyire láttam, nem tud távoli szerveren nyomkövetni) ➲ www.xored.com/trustudio/download
- a legjobban kidolgozott és a legszélesebb igényeknek is megfelel az időkorlát nélküli, *„Personal Licence"*-cel használható, java alapú *Zend Studio* és a hozzá tartozó (Apache-modulként működő, *.so* fájlokat tartalmazó) szerver ➲ www.zend.com/store/products/zend-studio

A ➲ www.php-editors.com értékelése szerint maximális pontot kapott a *Nusphere::PhpED*, az *ActiveState::Komodo* és a *Zend Studio*. Saját tapasztalatom is e hárommal kapcsolatban volt a legjobb, pontosabban az utóbbi kettővel.

A következő részekben a fent felsoroltakat fogjuk részletesen megvizsgálni.

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)

KAPCSOLÓDÓ CÍMEK

Írásom közben felhasználtam Papp Győző előadását:
➲ phpconf.hu/2004/media/eloadasok/chemotox.pdf

➲ hu.php.net/assert-options

➲ www.sitepoint.com/article/bug-detection-php-assertions

Párhuzamos programok fejlesztése PVM könyvtárral (2. rész) Szegény ember szuperszámítógépe

Folytassuk ott ahol legutóbb abbahagytuk és építsük meg életünk első PVM klaszterét. Ha ezzel megvagyunk gyorsan számoljunk is rajta valami érdekeset...

Egy *PVM* klaszter építéséhez a legegyszerűbb recept a következő:

- Vegyünk néhány hálózatba kötött linuxos gépet
- Telepítsünk rájuk *OpenSSH*-t és *PVM*-et
- Hozzunk létre rajtuk egy adott azonosítóval rendelkező felhasználót
- A létrehozott felhasználó számára engedélyezzük a jelszómentes *SSH*-t.

Előkészítés

A *PVM* telepítése a szokásos módon történhet, erről az előző cikkben már volt szó, ismétlésképpen a lépések egy *Debian* alapú rendszerben a következők:

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

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

Ha minden jól ment, a `pvm` parancs kiadásával ekkor el kell indulnia a *pvm* démonnak.

```
pvm>
```

A `conf` parancs kiadásával megtekinthető a „klaszter” jelenlegi konfigurációja:

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

A kapott listában a klaszterben szereplő gépek nevei láthatók. Új gép hozzáadása az add paranccsal történik, ahhoz azonban, hogy ezt könnyen és gyorsan megtehessük szükség van néhány kisebb beállításra a gépeken.

Jelszó nélküli SSH

Új gép hozzáadásánál a *PVM* démon megpróbál *SSH*-n keresztül csatlakozni a célgéphez és azon egy újabb *PVM* démont indítani. Ha a célgépet a felhasználó nem tudja jelszó nélkül elérni, akkor fárasztó gépelésre van szükség, ami ugyan biztonságos belépést biztosít, a munkát azonban nagyban megnehezítheti.
Az *OpenSSH* lehetőséget nyújt kulcs alapú azonosításra is. Ekkor adott gép adott felhasználója publikus és privát *RSA* vagy *DSA* kulcsokat generál, majd a publikus kulcsot elérhetővé teszi egy másik gép (ezentúl célgép) számára. Ha a felhasználó ezután *SSH*-n keresztül bejelentkezési kísérletet tesz a célgépre, akkor a publikus kulcs átküldésekor a célgép *SSH* szervere kódolt üzenetet vált a felhasználó gépével, felismeri a bejelentkező felhasználót és sikeres üzenetváltással azonosítja. A kulcshoz jelszó rendelhető, ez azonban nem kötelező, sőt *PVM* klaszter építésekor kifejezetten hasznos lehet ha nincsen jelszó.

A kulcs alapján történő azonosításhoz lássunk egy példát két gép között. Legyen az egyik gép ***cluster01.xyz.hu*** a másik pedig ***cluster02.xyz.hu***.
A kulcsgeneráláshoz ki kell választani azt a gépet ahonnan jelszó nélkül szeretnénk belépni a másikra, és a gép terminálján ki kell adni a következő parancsot:

```
ssh-keygen -t dsa -f .ssh/
➥id_dsa
```

Az `ssh-keygen` végzi a kulcsgenerálást, paraméterben a *DSA* rendszerű kulcs generálását kértük a *.ssh* könyvtárba. A parancs végrehajtása közben jelszót kér, ilyenkor jelszó nélküli belépéshez egyszerűen tovább kell lépni az *Enter* gomb lenyomásával. Az eredmény két fájl, az egyik a publikus (megosztandó) kulcsot a másik pedig a privát kulcsot tartalmazza.
Ha a kulcsgenerálást a ***cluster01.xyz.hu*** gépen végeztük akkor következő parancsokkal a kulcsot megoszthatjuk a másik géppel (***cluster02.xyz.hu***):

```
scp .ssh/id_dsa.pub
➥cluster02.xyz.hu:.ssh
ssh cluster02.xyz.hu
(itt még kér jelszót)
cat .ssh/id_dsa.pub >>
➥authorized_keys2
chmod 640 authorized_keys2
rm .ssh/id_dsa.pub
```

Ezután a ***cluster01.xyz.hu*** gépről belépve az *SSH* már nem kér jelszót. A *PVM*

1. Lista A master forráskódja

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

/* az inditando slave folyamatok szama
(a pontossag) */
#define NUM_SLAVES 32

int main ()
{
    int mytid, slave_tids[NUM_SLAVES + 2];
    int i;
    int num_slaves;
    double x;
    double partial_sum;
    double result;


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

    /* slave folyamatok inditasa */
    num_slaves = pvm_spawn("slave1", (char **)0,
➥PvmTaskDefault, "", NUM_SLAVES, &slave_tids[1]);

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

    /* a csatorna elso es utolso eleme a master */
    slave_tids[0] = slave_tids[NUM_SLAVES + 1] =
➥mytid;

    for (i = 1; i < NUM_SLAVES+1; i++) {
        /* az uzenetkuldes inicializalasa */
        pvm_initsend(PvmDataDefault);
        /* az elozo processzor tid-je */
        pvm_pkint(&slave_tids[i - 1], 1, 1);
        /* elkuldes */
        pvm_send(slave_tids[i], 0);

        pvm_initsend(PvmDataDefault);
        /* a kovetkezo processzor tid-je */
        pvm_pkint(&slave_tids[i + 1], 1, 1);
        /* elkuldes */
        pvm_send(slave_tids[i], 0);

        pvm_initsend(PvmDataDefault);
        /* index a szamitashoz */
        pvm_pkint(&i, 1, 1);
        /* elkuldes */
        pvm_send(slave_tids[i], 0);
    }

    x = 5;
    partial_sum = 0;

    /* x bedobasa a csatornaba */
    pvm_initsend(PvmDataDefault);
    pvm_pkdouble(&x, 1, 1);
    pvm_send(slave_tids[1], 0);

    /* a kezdo reszosszeg bedobasa a csatornaba*/
    pvm_initsend(PvmDataDefault);
    pvm_pkdouble(&partial_sum, 1, 1);
    pvm_send(slave_tids[1], 0);

    /* x kiolvasasa a csatornarol */
    pvm_recv(slave_tids[NUM_SLAVES], -1);
    pvm_upkdouble(&x, 1, 1);

    /* az eredmeny kiolvasasa a csatornarol */
    pvm_recv(slave_tids[NUM_SLAVES], -1);
    pvm_upkdouble(&result, 1, 1);

    /* kiiras */
    printf("exp(%f) = %f \n", x, result);

    /* slave taszkok "leallitasa" */
    for (i = 1; i < NUM_SLAVES+1; i++) {
        pvm_kill(slave_tids[i]);
    }

    /* kilepes */
    pvm_exit ();
    return 0;
}
```

klaszter felépítése most már gyerekjáték. Válasszunk ki egy központi gépet, ahol a programokat elindítjuk majd. Ha ezen a gépen publikus kulcsot generálunk és azt megosztjuk a leendő klaszter többi gépével akkor a *PVM* démon gond nélkül fel tudja építeni a kapcsolatot jelszó kérése nélkül. A központi gépen ezután hozzunk létre egy szöveges fájlt (például *hosts.txt*), melynek tartalma a klaszter többi gépeinek neve, valahogy így:

```
cluster02.xyz.hu
cluster03.xyz.hu
...
cluster10.xyz.hu
```

Most indítsuk el a *PVM* démont a *host* fájllal és a prompt megjelenése után adjuk ki a `conf` parancsot:

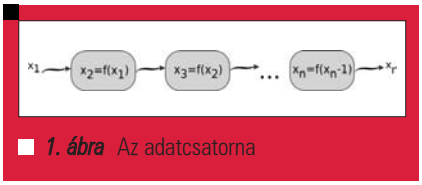
```
pvm hosts.txt
pvm>conf
10 hosts, 1 data format
        HOST    DTID    ARCH SPEED DSIG
cluster01  40000  LINUX 1000
                              ➥0x00408841
cluster02  80000  LINUX 1000
                              ➥0x00408841
. . .
cluster10  280000 LINUX 1000
                              ➥0x00408841
pvm>
```

A klaszter ezzel bevetésre kész, kihasználni azonban már nem ilyen

könnyű. Most megnézünk egy egyszerű programot, ami egy általánosan elfogadott párhuzamos tervezési mintára épül és könnyen átírható komolyabb feladatok megoldásához.

Az adatcsatorna

Az adatcsatorna az egyik legkönnyebben megérthető és alkalmazható párhuzamos tervezési minta. Olyan számításokra használható, ahol az eredmény egymáshoz nagyon hasonló függvények kompozíciójával kapjuk. A párhuzamosítást a kompozíciós lépések szétdarabolásával lehet elérni. Az *1. ábra* egy egyszerű adatcsatornát ábrázol. Az ábrán látható dobozok a csatorna elemei és egyben a kiszámítás egyes lépéseit szemléltetik. Jó látható, hogy az x_1 bemenethez a csatorna végén az x_n kimenet tartozik. Az x_n kiszámításához az *f* függvényt kellett alkalmazni (n-1)-szer. Általában ez annyival bonyolultabb, hogy nem egyetlen függvény építi fel a csatornát, hanem pontosan (n-1) darab. Ha az adatcsatorna egyes elemei folyamatok és mindegyik egy külön processzoron fut, akkor a csatorna első elemébe bedobva egy kezdőértéket, nem kell kivárni a végeredmény kiszámítását, elegendő az első elem



1. ábra Az adatcsatorna

felszabadulására várni, ugyanis amint az továbbadta részeredményét ismét képes bemenetet fogadni. Így egy m lépésben kiszámítható függvény esetében n elemre szekvenciális esetben (n*m) lépést kellene végrehajtani, míg párhuzamos esetben mindössze (n+m)-et. A gyorsulás tehát igen nagy.

2. Lista – A slave kódja

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

int main ()
{
    int mytid;
    int parent_tid;
    int index;
    double partial_sum;
    int last;
    int next;
    double x;
    double factab[170];
    int i;

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

    /* faktorialis prekalkulacio */
    factab[0] = 1;

    for (i = 1; i < 171; i++) {
        factab[i] = factab[i - 1] * i;
    }

    /* uzenetfogadas a szulotol */
    pvm_recv(parent_tid, -1);
    /* a csatorna elozo eleme */
    pvm_upkint(&last, 1, 1);


    /* uzenetfogadas a szulotol */
    pvm_recv(parent_tid, -1);
    /* a csatorna kovetkezo eleme */
    pvm_upkint(&next, 1, 1);

    /* uzenetfogadas a szulotol */
    pvm_recv(parent_tid, -1);
    /* index */
    pvm_upkint(&index, 1, 1);


    /* fogadas -> szamolas -> kuldes */
    while (1) {
        /* a csatorna elozo elemetol */
        /* x */
        pvm_recv(last, -1);
        pvm_upkdouble(&x, 1, 1);

        /* a reszosszeg */
        pvm_recv(last, -1);
        pvm_upkdouble(&partial_sum, 1, 1);

        /* szamolas */
        partial_sum += pow(x, index-1) /
➥factab[index-1];

        /* x tovabbkuldese */
        pvm_initsend(PvmDataDefault);
        pvm_pkdouble(&x, 1, 1);
        pvm_send(next, 0);

        /* a reszosszeg tovabbkuldese */
        pvm_initsend(PvmDataDefault);
        pvm_pkdouble(&partial_sum, 1, 1);
        pvm_send(next, 0);
    }

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

Példaként nézzünk meg egy adatcsatorna megvalósítást *PVM*-ben. A csatorna az exp függvényt fogja kiszámítani adott pontosságig. Az exp függvény az e (Euler-féle szám: 2,718281...) x-edik hatványa. Kiszámítására létezik egy végtelen összeg alak: $x^0/0! + x^1/1! + \ldots + x^n/n! + \ldots$ amiből az ***exp(x)*** adott pontosságú értéke egy részletösszeg. Ennyi matek után az olvasó akár meg is ijedhet, a feladat megoldása azonban nagyon egyszerű. Most is két programunk van, egy master és egy slave.
A fenti program felépíti a csatornát és bedobja az elején az 5-ös számot, majd (miután elkészült) kiveszi az eredményt a csatorna végéről.
Az egyszerűség kedvéért most erre az egyetlen számra nézzük meg a működést, azonban komolyabb munkára is fogható a program, ha bemenetét például egy fájlból veszi és az nem egyetlen szám.
A csatorna felépítését egyszerű láncolással lehet megoldani, amihez az egyes elemeknek el kell küldeni az előző és a következő elem azonosítóját. Az azonosítón kívül az egyes csatornaelemek megkapják még a kiszámításhoz az indexüket is. Ebből az indexből határozza majd meg az adott elem, hogy a részletösszeg hányadik tagját alkotja.
Hogy mindez világosabbá váljék, lássuk a csatornát felépítő slave kódját (*2. Lista*).
A slave program miután megkapta a sorban előtte és mögötte álló elemet azonosítóját és saját indexét, vár az eddig kiszámolt eredményre és a bemenetre. E kettőre lesz szüksége ahhoz, hogy a részletösszeg következő tagját kiszámolja és továbbküldje.
A programok használatba vételéhez le kell fordítani őket, itt nincs semmi új, emlékeztetőül a fordítás menete:

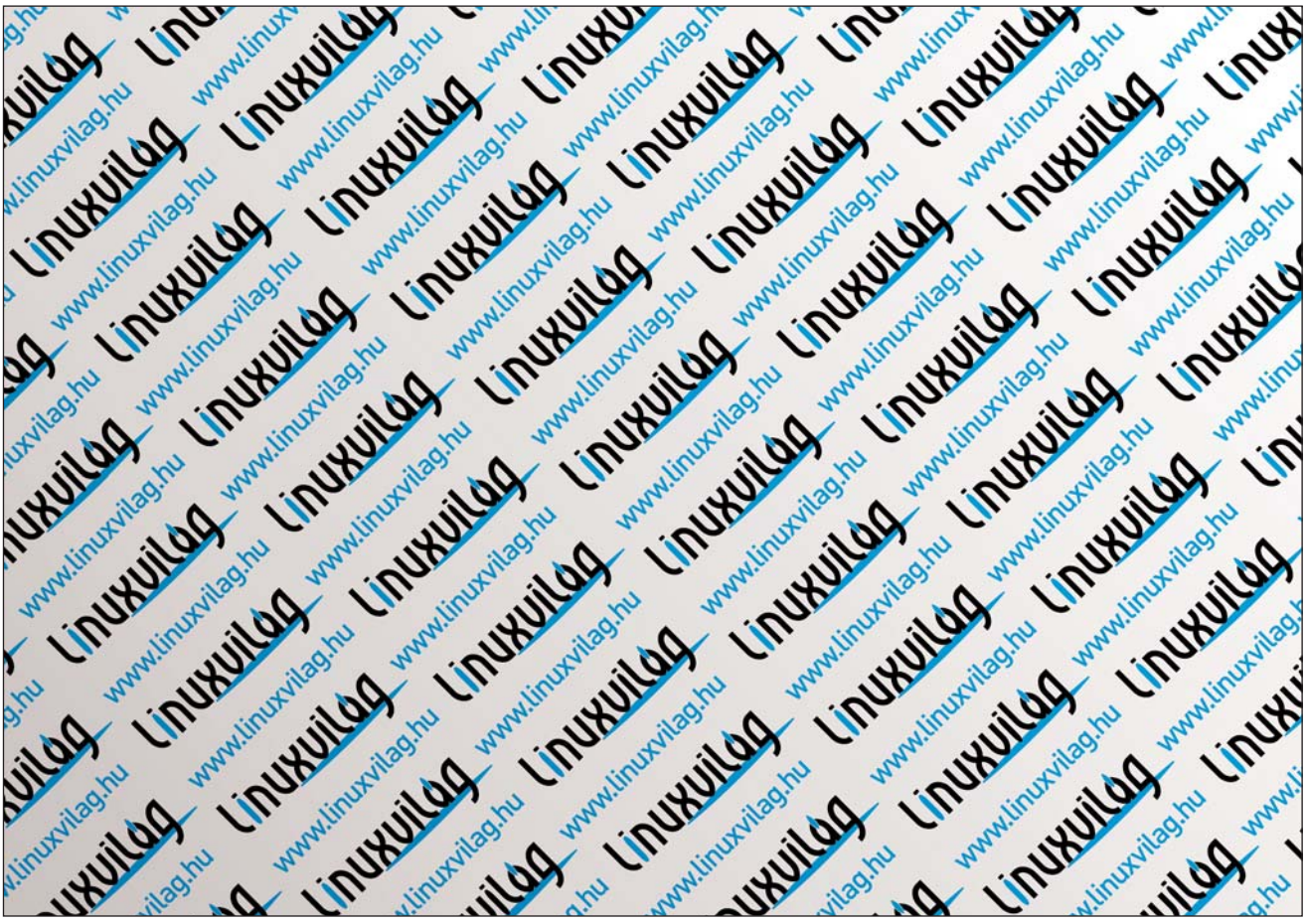
```
gcc master.c -o master1 -lpvm3
gcc slave.c -o slave1 -lpvm3
➥-lm
```

Az elkészült állományokat (***master1*** és ***slave1***) másoljuk a *~/pvm3/bin/LINUX* könyvtárba, majd a ***slave1*** programot osszuk meg a klaszter többi gépével. A megosztás *SSH*-n keresztül *SCP*-vel történhet és könnyen automatizálható, hiszen a többi gép már jelszó nélkül is elérhető.
Például:

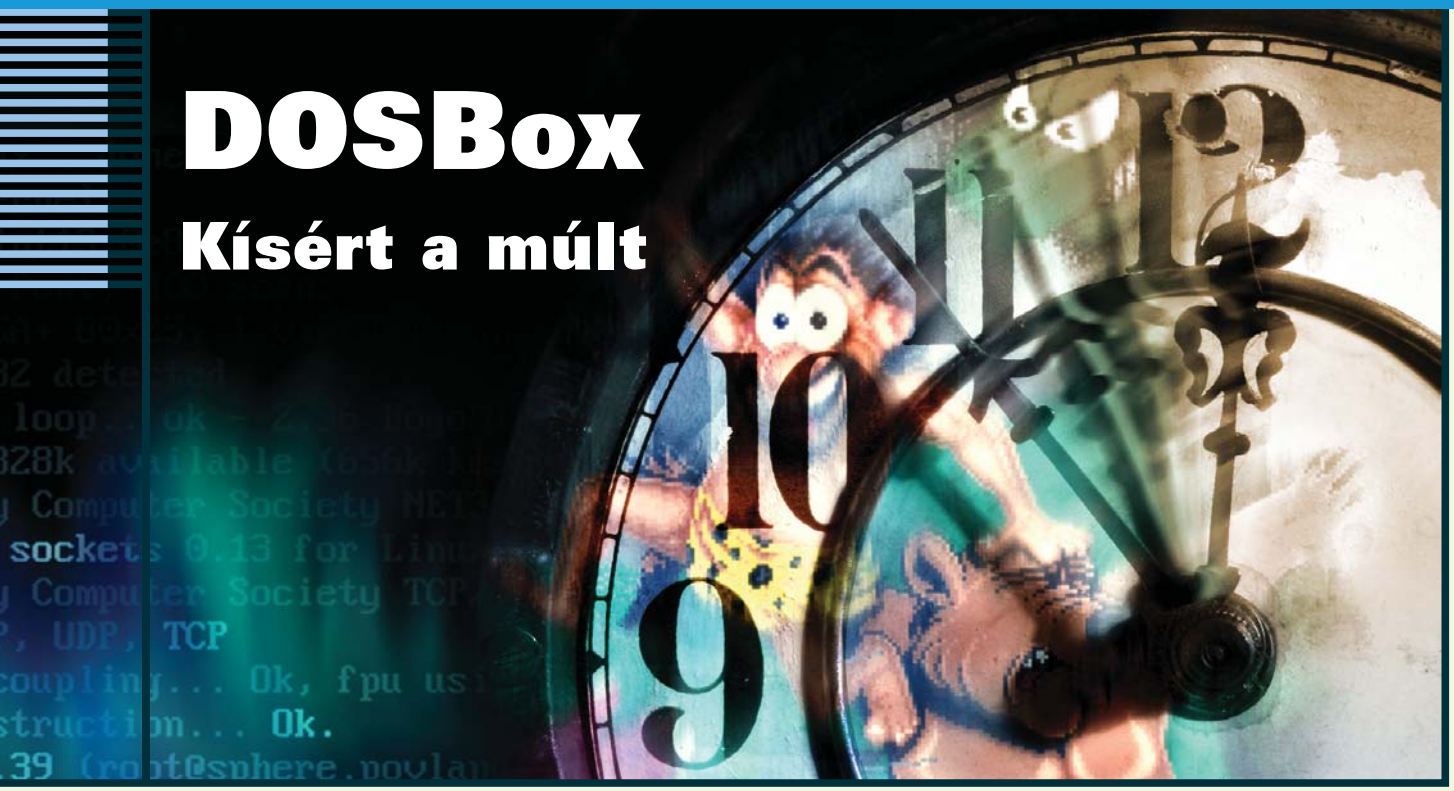
```
scp slave1 cluster02.xyz.hu:
➥pvm3/bin/LINUX
```

Ha a megosztást az összes gépre elvégeztük, akkor a ***master1*** program elindítható. Fontos megjegyezni, hogy ha az elindított program a képernyőre is ír, akkor nem szabad *PVM*-en belülről `spawn` paranccsal indítani, mert ekkor a kimenetet nem látjuk. Az indításhoz lépjünk ki a *PVM*-ből, mely a háttérben tovább fut, és adjuk ki a master1 parancsot. Ha minden jól működik, az eredmény nem más mint ***exp(5)*** egy egészen pontos közelítése.

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.



DOSBox
Kísért a múlt



Manapság szó szerint kísért a múlt. Előretörnek az egyszer már jól bevált dolgok, divat lett a retro. A DOSBox szintén segítségünkre lehet felidézni a múltat. Ez a program mint látni fogjuk, több mint egy DOS emulátor. Szép emlékeket ápolni jó dolog, ismerjük meg a DOSBox-ot!

Első hallásra, talán az elnevezés miatt, inkább gondolnánk egy ablakkezelőre, hiszen több ablakkezelő nevében megtalálható a ***box*** kifejezés. Akik még nem hallottak volna a ***DOSBox*** nevű ***emulátorról***, és a régi szép időkben még keményen használták a ***DOS*** nevű operációs rendszert, azok remélem sok örömüket fogják lelni ebben az alkalmazásban. Akik pedig kimaradtak ebből a korszakból és nem ismerősek számukra azon kifejezések, mint ***DOS-Navigator, Norton Commander, edit, autoexec.bat, config.sys, command.com, qbasic, PreHistorik, Prince*** és még sorolhatnám, azoknak is ajánlatos megismerkedni a programmal. Azonban, mint említésre került, nem csak ***DOS*** programok futtatására alkalmas a ***DOSBox***. Hivatalosan ez egy ***x86 PC emulátor***. Még javában fejlesztés alatt áll, de a fejlesztők célja egy komplett ***x86 PC emulátor*** elkészítése. Természetesen a legtöbb platformra ***portolva*** van. Jelenleg már támogatja bootolható lemezképek betöltését is. Alapból, ha nem alkalmazunk semmilyen különleges kapcsolót, vagy opciót akkor a jó öreg ***DOS*** indul el a ***DOSBox*** ablakában *(1. ábra)*. Így varázsol ez az emulátor linuxos gépünkből is egy igazi időgépet.

Gép a gépben

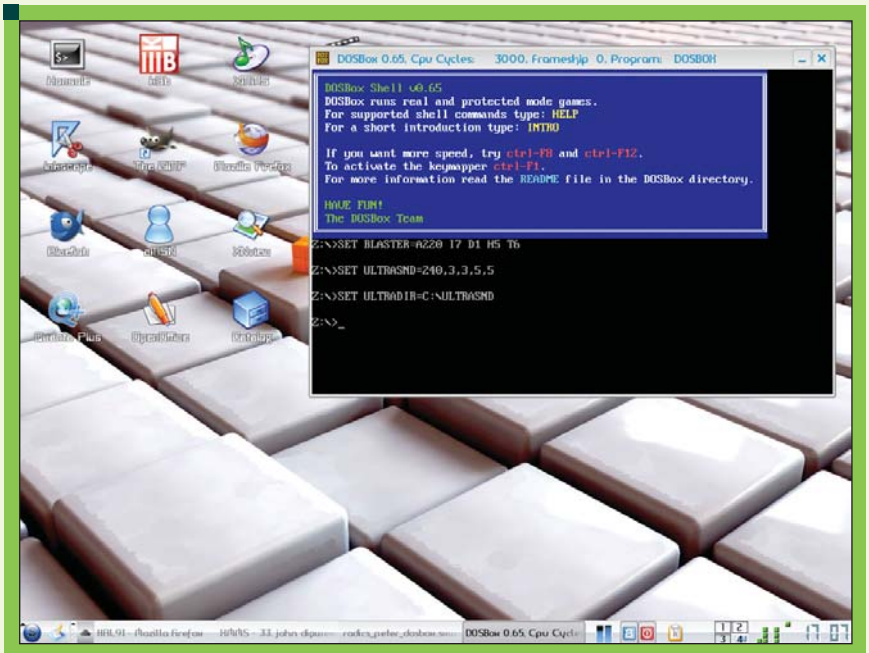
A ***DOSBox*** igen sok szolgáltatással rendelkezik már, amit általában egy ***PC***-től elvárunk. Mint említettük megbirkózik a bootolható lemezképekkel, van modem és ***IPX*** emulációja, támogat számos hangkártyát, külső könyvtárakat lehet az emulált gépbe csatlakoztatni, több videokártyát tud utánozni, csak hogy egy párat említsünk a lehetőségek közül.

Beállítás

Hogy ne kelljen mindent mindig beállítgatni indítás után, a parancssorból ügyesen konfigurálhatjuk a programot (lásd az *1. Táblázatot*). A ***DOSBox DOS promptja*** alapból minden olyan utasítást ismer, amit még a jó öreg ***command.com*** is tudott. Pár dologgal azért ez az utasításkészlet ***kiegészült***, hiszen emulátorról van szó. A `mount` parancs segítségével külső, azaz a valódi gépünkön lévő könyvtárakat illetve meghajtókat csatolhatunk be az emulátor fájlrendszerébe. Például :

```
mount c /home/jozsi/dos
```

A `c` itt a meghajtó betűjele. A `mem` parancs megjeleníti a rendelkezésünkre álló memória méretét. A `config` az aktuális beállításokat tudja kimenteni, hogy később ne kelljen megint beállítgatni a dolgokat. Ezt a konfigurációs állományt a parancssorból, az emulátor indításakor kényelmesen betölthetjük. A `rescan` parancs akkor lehet hasznos, ha egy, az emulátor fájlrendszerébe bekapcsolt könyvtár vagy meghajtó tartalma megváltozik. Ekkor ezen parancs újraolvassa a becsatolt helyeket. Az `imgmount` segítségével lemezképek becsatolását vihetjük végbe. Az `ipx` parancs az ***IPX protokol*** emulációját bonyolítja le. Ezt az opciót először a konfigurációs állományban kell engedélyezni, majd a protokoll beállításait az emulátoron belül, ezen parancs segítségével lehet elvégezni.



1. ábra Indul az időgép

1. táblázat *A DOSBox parancssori opciói*

`-fullscreen`	Alapból teljes képernyőn indítja az emulátort
`-startmapper`	Induláskor átállíthatjuk a billentyűzetet
`-noautoexec`	Kihagyja a konfigurációs fájl [autoexec] alatti lépéseit
`-c` utasítás	Induláskor végrehajtja az utasítást
`-conf` fájl	Betölti a fájlból a konfigurációs beállításokat
`-lang` fájl	Betölti a fájlból a nyelvi elemeket
`-exit`	Kilép a programból ha a benne futtatott DOS alkalmazás sikeresen véget ért
`-machine` típus	A típusban adhatjuk meg, hogy milyen típusú PC-t emuláljon a program (hercules, pcjr, cga, tandy, vga); alapból a vga opció van bekapcsolva
`-version`	Megjeleníti az aktuális verziót

A beépített parancsokkal kapcsolatban a *DOSBox* a `help` parancs segítségével tud több információt szolgáltatni. A konfigurációs állomány helye, alapesetben a *home* könyvtárunk gyökere (*~/.dosboxrc*), de parancssorból könnyen betölthetünk máshonnan is beállításokat.
A *DOSBox* tulajdonságait futási időben is tudjuk módosítani. A beállítások megváltoztatására különböző billentyűkombinációk nyújtanak lehetőséget *(2. Táblázat)*.

A sebesség növelésénél figyeljünk oda! Ha túl sokat emelünk a sebesség értékén, akkor az ellenkezőjét érhetjük el. Az emuláció lelassulhat. Hogy mely értéktől következik ez be, az gépfüggő.

Indulhat az időgép

Már mindent tudunk, ami az emulátor üzemeltetéséhez szükséges. Lássunk hát példát az alkalmazására. Ha valakinek nem áll rendelkezésére régi *DOS* alkalmazás, de szeretné kipróbálni az *emulátort,* látogasson el

2. táblázat *A DOSBox billentyűkombinációi*

ALT+ENTER	Az ablakos és teljes képernyős mód között válthatunk
ALT+BREAK	Leállítja az emulációt
CTRL+F1	Elindítja a key-mappert, melynek segítségével a billentyűzet kiosztását módosíthatjuk
CTRL+ALT+F5	Segítségével AVI-ba rögzíthetjük az emulátor „képernyőjén" megjelenő eseményeket, a felvétel elindítása és megállítása is ezzel a kombinációval történik
CTRL+F5	Képernyő mentése egy PNG állományba
CTRL+F6	A hangkimenet mentése WAV fájlba
CTRL+ALT+F7	Az OPL parancsok rögzítése
CTRL+ALT+F8	A MIDI parancsok rögzítése
CTRL+F7	Növeli a frame kihagyások számát
CTRL+F8	Csökkenti a frame kihagyások számát
CTRL+F9	Kilép a DOSBoxból
CTRL+F10	Elengedi az egeret az emulátorból
CTRL+F11	Lassítja az emulációt
CTRL+F12	Gyorsítja az emulációt
ALT+F12	Letiltja a sebességkorlátozást, azaz maximalizálja az emuláció sebességét

a ➲ *retrogames.intranet.hu* oldalra. Itt temérdek régi *DOS* játékot találhatunk, természetesen jogtisztán letölthetőek, mivel a gyártók már ingyenessé tették őket, ahogyan eljárt felettük az idő. Készítsünk az öreg alkalmazásoknak a *home* könyvtárunkban egy külön könyvtárat, például */home/jozsi/dos*. Ide pakoljuk

2. ábra PreHistorik: Vajon van olyan, aki nem emlékszik?

a kipróbálni kívánt alkalmazásokat. Parancssorból indíthatjuk ezután az emulátort így is:

```
dosbox -c 'mount c
/home/jozsi/dos'
```

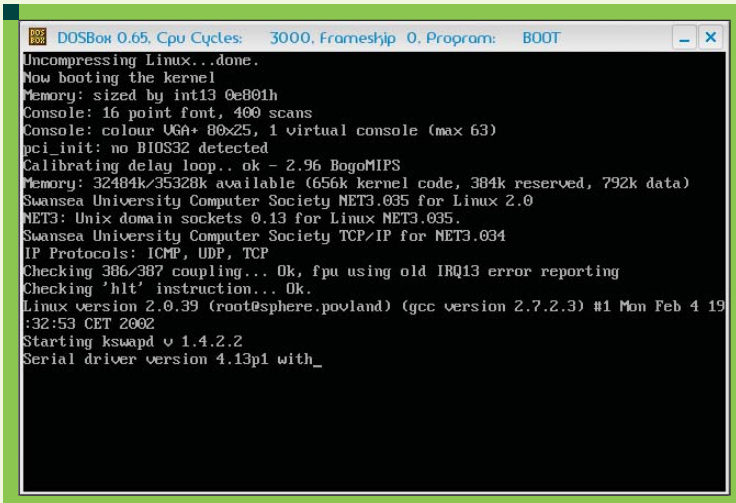
Ezek után már csak a *„C:"* jelű meghajtóra kell váltanunk és indíthatjuk is a *„retro"* alkalmazást *(2. ábra)*.

Nem csak DOS

A ***DOSBox*** segítségével, mint látni fogjuk, igen könnyen indíthatunk el bootolható lemezképeket is. A mi esetünkben a ***HAL91 Linux*** disztribúció egy képét fogjuk bootolni *(3. ábra)*. Indítsuk a ***DOSBox***-ot majd csatlakoztassuk azt a könyvtárat, ahol a bootolandó kép található. Ezek után váltsunk arra a könyvtárra az emulátorban ahol a kép található, vagy adjuk meg a `boot` parancsnál annak teljes elérési útját:

```
boot hal91.img
```

Remélhetőleg sikerült bemutatnom egy hasznos alkalmazást, és sikerült ezúton régi, kellemes érzések



3. ábra Linux bootol Linuxon

felidézéséhez segítenem pár embert . Ez az emulátor igen szépen fejlődik, érdemes látogatni a honlapját: ➲ ***dosbox.sourceforge.net***. Régi ***DOS***-os alkalmazásokra pedig könnyen rá lehet lelni az ***Interneten***. Érdemes ellátogatni a ➲ ***www.opus.co.tt/dave/*** lapra is. Sok hasznos ***DOS*** alkalmazást rejt ez a lap. Kellemes és hasznos időtöltést kívánok a ***DOSBox***-hoz!

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.

A Latex használata (3. rész)
Felsorolások, táblázatok és ábrák

A sorozatnak ebben a részében a szövegformázás további eszközeiről lesz szó. Néhány példát mutatok felsorolások, táblázatok és ábrák beillesztésére. Az ábrákkal kapcsolatban kitérek olyan rajzolóprogramokra, melyek képesek hatékonyan együttműködni LaTeX-hel. Most sem célom a technikai részletek alapos tárgyalása.

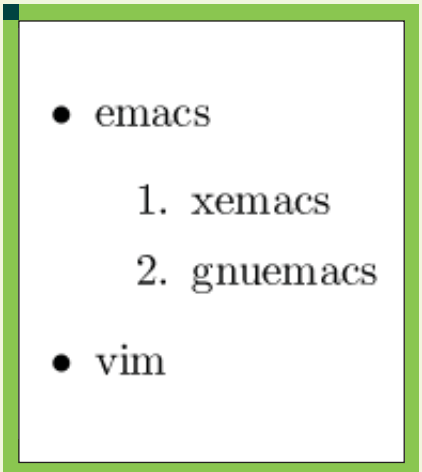
LaTeX-ben többfajta felsorolás van, számozott vagy számozás nélküli és leíró. Számozás nélküli listát a `\begin{itemize} \end{itemize}` környezettel írhatunk, ahol az egyes elemeket az `\item` jelzi. Számozott listák esetén az `itemize` helyett az `enumerate` környezetet alkalmazzuk, az elemeket itt is az `\item` jelöli, ennek most van opcionális paramétere, amivel az elem előtt álló számot állíthatjuk be. Leíró listákat a `description` környezettel készíthetünk. Itt megadunk egy címszót, amit félkövéren szed a *LaTeX*, és utána a leírását. Ekkor az `\item` opcionális paraméterében van az a kifejezés, aminek megadjuk a leírását. *Emacs*-et és *auctex*-et használva a `LaTeX` menüpont `Insert Environment` alpontjában az előbbi környezeteket választva szúrhatunk be felsorolásokat a forrás fájlba. Leíró listák esetén az *auctex* a minibufferben, ami az *emacs* ablak legalsó sora, rákérdez az `\item` paraméterére. A különböző típusú felsorolások négy szintig egymásba ágyazhatók.
Az elemek előtt álló szimbólumok vagy számok megváltoztathatók. A `\renewcommand{}{}` már definiált *LaTeX* parancsok átdefiniálására való, az első paramétere az a parancs amit megváltoztatunk, a második az új definíció.

Hatása a következő újradefiniálásig tart. Számozott listáknál az első szinten arab számok az alapértelmezettek, további szinteken római számok és betűk lesznek. Az első szinten a következő paranccsal állíthatjuk be számozás stílusát: `\renewcommand{\labelenumi}{\Alph{enumi}}` További szinteken a `\labelenumii`, `\labelenumiii` és a `\labelenumiv` parancsokat kell átdefiniálni.

Lehetséges értékeik:

- `\arabic` arab számok
- `\alph` kis latin betűk
- `\Alph` nagy latin betűk
- `\roman` kis római számok
- `\Roman` nagy római számok

A felsoroló listáknál használt szimbólum is átállítható a `\renewcommand{\labelitemi}{$\star$}` paranccsal, ahol a használható szimbólumok a *symbols.dvi* fájlban találhatók. Debian alatt ez a fájl a */usr/share/doc/texmf/latex/general* könyvtárban van.
Előbbihez hasonlóan, a további szinteken a `\labelitemii`, `\labelitemiii` és a `\labelitemiv` definiálja szimbólumokat.



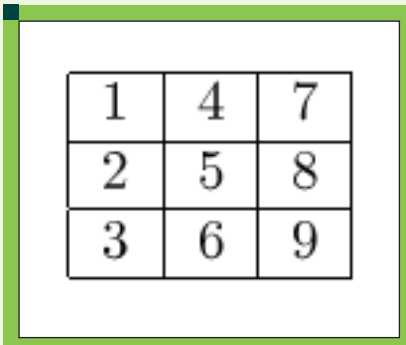
1. ábra Felsorolásba ágyazott számozott lista

1. Lista Felsorolás

```
\documentclass{article}
\begin{document}
\begin{itemize}
\item emacs
\item
  \begin{enumerate}
  \item xemacs
  \item gnuemacs
  \end{enumerate}
\item vim
\end{itemize}
\end{document}
```

2. Lista Táblázat

```
\documentclass{article}
\begin{document}
\begin{tabular}{|l|l|l|}
\hline
1 & 4 & 7 \\\hline
2 & 5 & 8 \\\hline
3 & 6 & 9 \\\hline
\end{tabular}
\end{document}
```



2. ábra Háromszor hármas, teljesen bekeretezett táblázat, amiben az oszlopok balra igazítottak

3. Lista PostScript fájl beillesztése

```
\documentclass[a4paper,12pt]
➥{article}
\usepackage{graphics}
\begin{document}
\resizebox{70mm}{!}{
  \rotatebox{90}{
    \includegraphics{3_
    ➥Balazs.eps}
  }
}\\
\end{document}
```

Táblázatok közül csak a legegyszerűbbekkel foglalkozom. Ugyan szinte tetszőleges bonyolultságú táblázat elkészíthető *LaTeX*-ben, de ehhez nagy tapasztalat és több, egymással részben inkompatibilis csomag betöltése szükséges. Fontos, hogy a táblázatok geometriai jellemzőivel nem kell törődnünk, azokat a *LaTeX* meghatározza. Lehetőségünk van ezeket is megadni, amivel azonban csak különleges esetekben érdemes élni.
A *LaTeX* táblázatok a `\begin{tabular} \end{tabular}` környezetben találhatók. A kötelező paraméterben adjuk meg az oszlopok számát és igazítását. Itt az oszlopokat az `lcr` betűk valamelyike jelzi, utalva az igazításra, `l`: balra, `c`: középre, `r`: jobbra. Az oszlopok között függőleges vonalakat húzhatunk a `|`-al. A táblázat sorait `\\` választja el, míg a soron belül a cellákat az &. Vízszintes vonalakat `\hline`-al illeszthetünk a táblázatba.
Az esetleges üres cellákat is jelölni kell, az oszlopok számánál eggyel kevesebb & jelnek kell lennie a táblázat soraiban.
Ezek egyszerű táblázatok esetén ez megegyezhetnek az *emacs* soraival, bonyolultabb táblázatoknál ez általában nincs így.
Ábrákat több, lényegesen különböző módon illeszthetünk *LaTeX* dokumentumainkba. A csak vonalakból, körívekből álló ábrákat elkészíthetjük *LaTeX* utasításokkal, ez azonban elég nehézkes. Másik lehetőség, hogy rajzolóprogramot használunk és az állítja elő a *LaTeX* kódot. Ilyen a ***kig*** vagy az ***xfig***.
A *gnuplot* is hatékonyan használható *LaTeX* ábrák előállítására, különösen formulával megadott függvények vagy mérési adatok, grafikonok esetén. *Gnuplot*-ban élesen kettéválik az ábrák elkészítése és megjelenítése. Előbb szöveges módban, nem interaktívan leírjuk a grafikont majd külön lépésben kirajzoltatjuk képernyőre vagy fájlba. Az első lépést segíti az *emacs* *gnuplot* módja.
A *LaTeX* kód kézzel vagy programmal történő előállításán kívül egy további lehetőség *postscript* fájlok beillesztése. Ehhez a `graphics` csomagot kell betölteni. Ezután a következő kódrészlettel szúrhatunk be *eps* fájlokat.
Belülről kifele fogom elemezni a fenti kódot. Ebben szerepel a *TeX* egyik belső, felhasználók elöl általában elrejtett fogalma, a doboz. Segítségével meglehetősen bonyolult formázásokat elérhetünk, amit részletesebben nem vizsgálok.
Az `\includegraphics{}` utasítással adjuk meg az *eps* fájl nevét.
A `\rotatebox{}{}` elforgatja az első paraméterben adott fokkal a másodikban szereplő dobozt, ami esetünkben a *postscript* fájl.
A `\resizebox{}{}{}` az első két paraméternek megfelelően átméretezi a harmadikban lévő dobozt. Az elsőben a vízszintes, a másodikban függőleges méret van. A `!` hatására a paraméter értékét a másik, számokkal megadott alapján úgy választja meg a *LaTeX*, hogy az ábra köré rajzolt téglalap oldalainak aránya ne változzon. Az ábrák beillesztése szerintem azon ritka esetek egyike, amikor érdemes kézzel megadni a geometriai méreteket mm-ben. *PostScript* ábrák használatakor a `pdflatex` nem működik, a *PDF* formátumot több lépésben kell előállítani. A konvertálás a `dvipdf`-el végezhető el, ami nem érhető el *emacs*-ből.
Mint általában a *TeX*-ben, felsorolások, táblázatok vagy grafikák esetén is, a gyakran előforduló, egyszerű feladatok könnyen megoldhatók. Azonban különösen jellemző hogy a bonyolultabbak megoldásához több, egymást részben kizáró, *LaTeX* csomagot és külső programokat kell használni. Valamint, hogy nem elkerülhető a *LaTeX* belső felépítésének megismerése. Fontos, azonban, hogy már a *LaTeX* által biztosított eszközök alapértelmezett beállításaival is általában elérhetjük céljainkat, esetleg kis kompromisszumok árán.
A sorozat további következő részében a hosszú, több szerzős dokumentumok szerkesztéséről lesz szó.
Ehhez a *LaTeX* és az *emacs* kényelmes eszközöket kínál. Nyomon követhetők a változtatások, visszatérhetünk régi változatokhoz, azokból részleteket emelhetünk át.

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 (5. rész)
KDirStat – Könyvtárak statisztikái

Mindig szerettem volna tudni, hogy az 50 GBájt kapacitású /home könyvtáramban miért fogy el a hely pillanatok alatt. Visszagondolva a régi „szép” időkre – amikor 20 MBájt volt a teljes merevlemezem mérete, azon sem volt túl sok szabad hely. Mintha Murphy egyszer mondta volna, hogy „ami hely megtölthető, az meg is telik”. De mivel?

A fenti problémát egy rutinos *UNIX* felhasználó a du parancs segítségével oldaná meg, amelyet megfelelően felparaméterezve hamar kideríthető, hogy melyik alkönyvtár terjeszkedett el galád módon – helyet nem kímélve – a merevlemez felületén. Kevésbé rutinos felhasználóknak a ***KDirStat*** programot javaslom, amely minderre grafikus felületen és egy szép ábrát mellékelve képes.

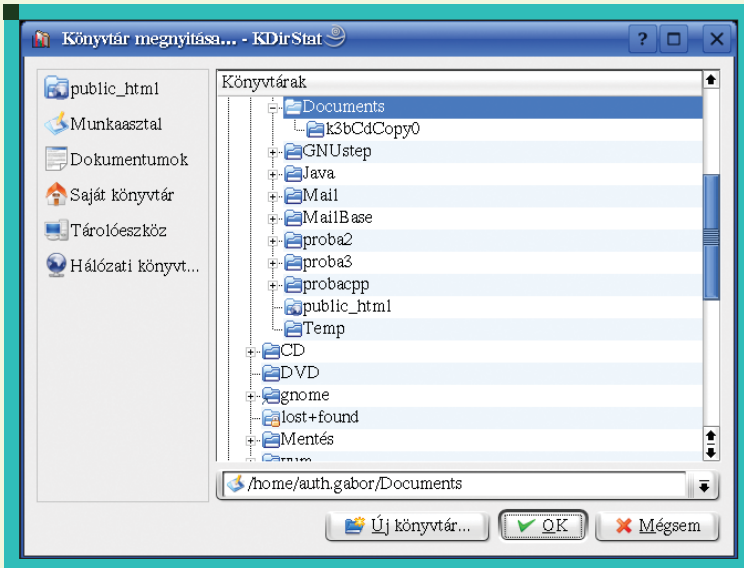
Ez a program nem része a KDE terjesztésnek, viszont a legtöbb disztribúció tartalmazza a feltelepíthető programok között. Ha ennek ellenére nem találnánk meg, akkor a ➲ http://kdirstat.sourceforge.net/ oldalon is megtalálhatjuk.

A feltelepült programot általában a ***Rendszer -> Fájlrendszer -> KDirStat*** menüpontban találjuk, ha mégsem, akkor parancsként a `kdirstat` nevet kell beírnunk.

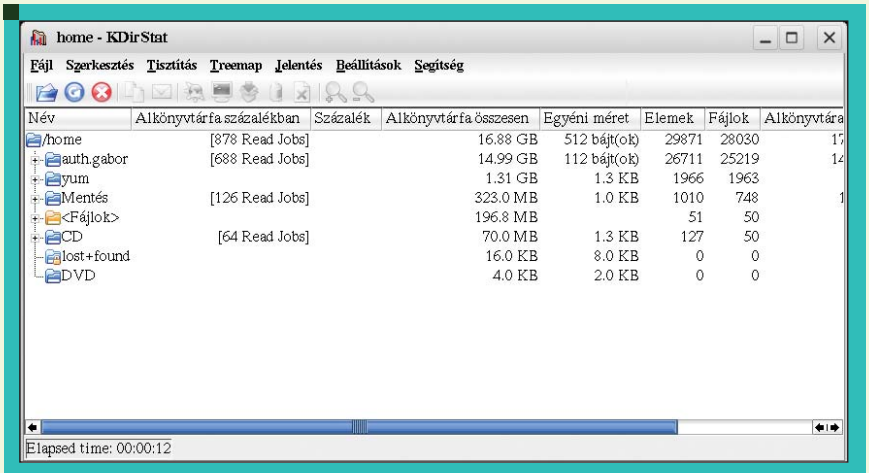
Első dolgunk azon könyvtár kiválasztása, amelytől kezdve a foglalt helyet fel szeretnénk deríteni (*1. ábra*).

Ez alapesetben a ***Documents*** mappa, amely helyett kiválaszthatjuk például a ***/home*** könyvtárat, ha erről szeretnénk információkat kapni.

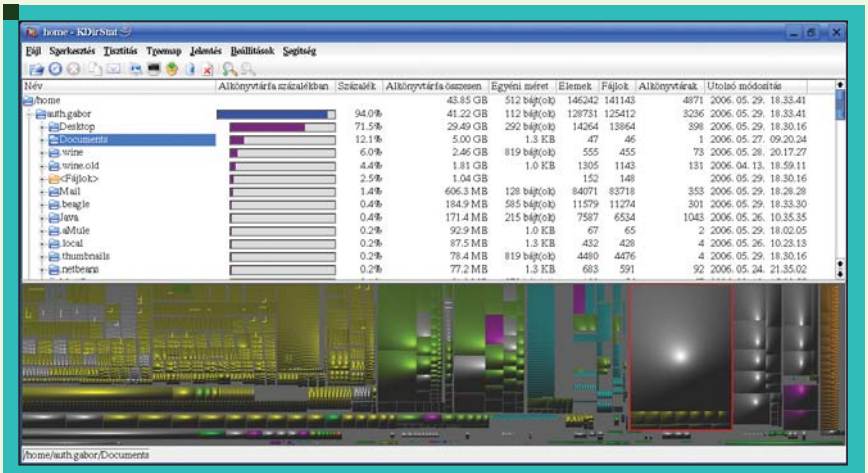
Az ***OK*** gombra kattintva a program nekigyürkőzik, és állományról-állományra, illetve alkönyvtárról-alkönyvtárra járva felderíti a megadott könyvtár alatti elfoglalt helyeket (*2. ábra*). Ez a fájlok méretétől és számától függően akár több perc is lehet.



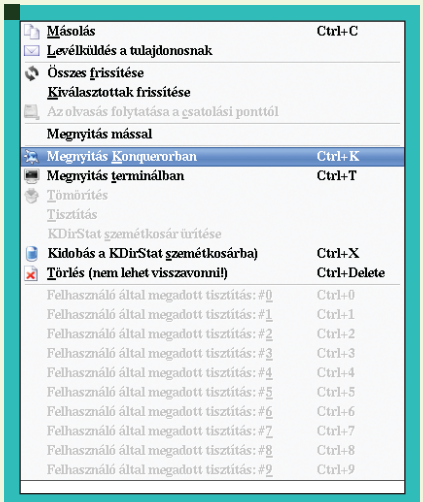
1. ábra A gyökérkönyvtár kiválasztása



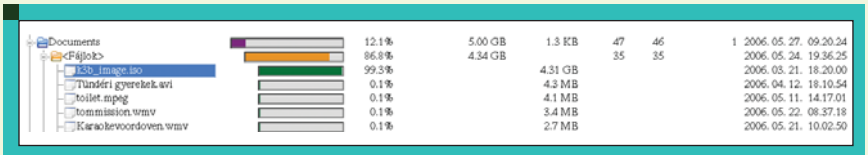
2. ábra Keresés



3. ábra A keresés eredménye



5. ábra Mivel nyissuk meg?

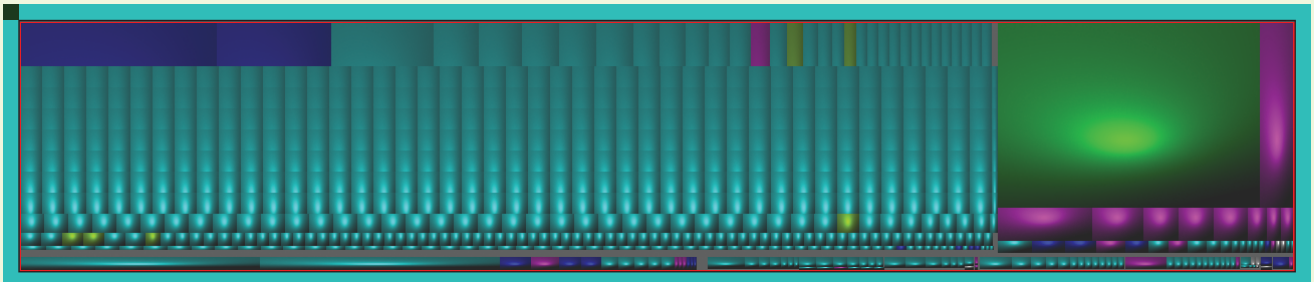


4. ábra Micsoda feledékenység!

A keresés végén kapunk a kiválasztott könyvtár alatti alkönyvtárakról egy elfoglalt hely szerint csökkenő sorrendbe rendezett listát az alkönyvtárakról és állományokról. A program ezen része teljesen azonosan működik, mint a

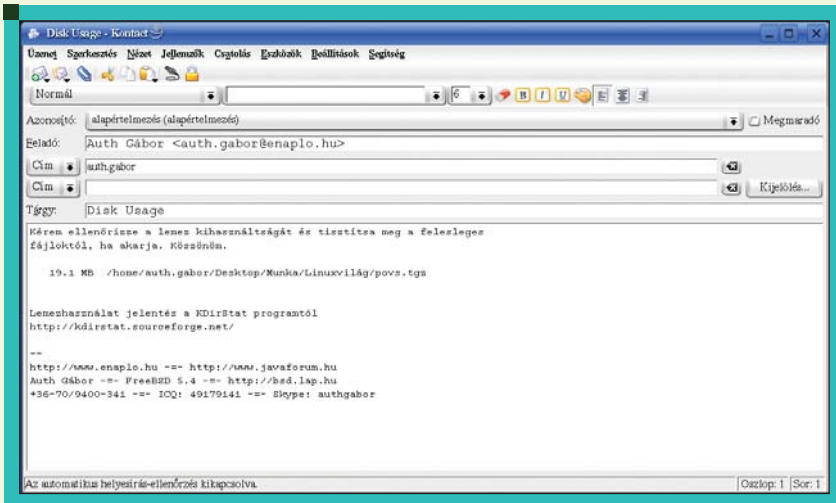
```
du -max-depth=1 /home
```

program kimenete, a lényeg az alább található grafikán van (*3. ábra*). Rögtön leolvasható a listáról, hogy a 43 GBájt foglalt területből 41 GBájtot én használok el olyan dolgokra, amelyeket már régen elfelejtettem.
Első pillanatokban egy kusza összevisszaságot látunk a program alsó részén lévő grafikán, amelynek egy-egy egybefüggő téglalapja egy-egy állományt reprezentál. Ha kiválasztunk egy ilyen nagyobb állományt, vagy könyvtárat, akkor láthatjuk, hogy azt vörös vonallal veszi körül a program. Szinte vonzza az egeremet az a nagy egybefüggő terület, ami a jobb szél felé található és a hely tizedét biztos elfoglalja. A szép nagy területre kattintva a program kibontja a hozzá tartozó könyvtárakat és megmutatja (*4. ábra*), hogy egy március vége óta változatlan ***DVD ISO*** okoz közel 4.3 GBájtnyi felesleges foglaltságot.
Ha jobb gombbal kattintunk a kérdéses állományon, akkor különféle műveleteket tudunk végezni (*5. ábra*). A legtöbb állományt meg tudjuk nyitni a Konqueror segítségével, amely majd meghívja az állomány kezeléséhez szükséges programot, ha nem képes azt önmaga kezelni. ***ISO*** állományokhoz a ***K3b*** program van rendelve, amellyel képesek leszünk a kiválasztott ***ISO***-t ***CD***/***DVD***-re írni. (***ISO*** állományt a ***Kiso*** programmal is, amelyről egy későbbi cikkben lesz több szó.)
Ha eldöntöttük, hogy nem kell a kérdéses állomány, akkor az *5. ábrán* látható menüből egy szimpla egérkattintással ki tudjuk törölni.
Ha nem találunk túl nagy méretű állományokat, csak sok kicsit és még több aprót, akkor érdemes a program felső részén lévő fa struktúrában megkeresni azokat az állományokat, amelyekre nem biztos, hogy szükségünk van.
Ha belemászunk ebbe a fa struktúrába, akkor a program ikonsorában aktív lesz a két nagyítót formázó ikon, amelyek közül egy egyikkel az alsó grafikába szintenként bele tudunk nagyítani, a másikkal pedig egy szintet feljebb lépni (*6. ábra*).



6. ábra Nagyítás

Ha belenagyítunk egy ilyen kiválasztott könyvtárba, akkor láthatunk olyan nagyobb állományokat, amelyek a teljes foglaltsághoz képest elenyésző méretűek, viszont magában a könyvtárban gyanúsan feleslegesnek tűnnek. A *6. ábrán* jobbra látható szép nagy zöldes terület például a ***PovRay*** cikkekhez tartozó képeket tartalmazta egy állományba csomagolva, már le is töröltem. A program egyik előnye, hogy a különféle állomány típusokat különféle színnel jelöli, az előbbiekben kitörölt fájl egy ***.tar.gz*** volt, a sok ki apró fájl képeket tartalmaz, a néhány lilás pedig ***OpenOffice.org*** dokumentumokat.
Nagyobb rendszereken használható a program levélküldés funkciója, amelyre rákattintva a kiválasztott fájl tulajdonosának tudunk egy sablonlevelet küldeni (*7. ábra*), hogy legyen kedves a megadott állományt eltávolítani, mert sok helyet foglal.
A program remekül illeszkedik a ***KDE*** többi programja közé, remélhetőleg nemsokára bekerül a belső körbe, vagyis az „alaprendszer” része lesz, így különösebb nehézségek nélkül megtalálhatjuk és használhatjuk bármilyen telepített rendszeren.



7. ábra Levélküldés

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

Klik – linuxos alkalmazások egy kattintásra

Okozott már gondot egy Linux-os alkalmazás telepítése? Történt már olyan, hogy egy alkalmazás csomagjának telepítése nem történt zökkenőmentesen? És olyan, hogy egy alkalmazás nem a megszokott helyre települt? Jó lett volna ugyanazt a csomagot más disztribúciókra is telepíteni? A Klik talán egy napon minderre megoldást nyújthat.

A *Linux* alapú operációs rendszerekkel (disztribúciókkal) ismerkedő felhasználók előbb-utóbb eljutnak egy pontig, ahol már elkerülhetetlen az adott disztribúció csomagkezelési rendszerének ismerete. Kezdő felhasználók számára új alkalmazások telepítése esetenként egy félelmekkel teli szürke terület, ahová nem szívesen vándorolnak, attól félve, hogy valami kárt okozhatnak egy sikertelen telepítéssel.
A legtöbb mai *Linux* disztribúció csomagkezelési rendszere megbízhatóan működik, de gondok mindig adódhatnak. Például ha egy csomag telepítése közben történik egy áramszünet vagy valami más okból megszakad a telepítés, ez okozhat egy átlagos felhasználónak néhány álmatlan órát.
Egy másik lehetséges helyzet, amely már nem csak kezdő vagy átlagos felhasználókat érinthet, ha például a felhasználó szeretne frissíteni egy adott alkalmazást egy újabb verzióra, de szívesen kipróbálná mielőtt élesben is lecserélné a régit az új változatra. Vagy tegyük fel szívesen megismerkednénk egy komplex alkalmazással (például egy másik grafikus asztali környezet) úgy, hogy ne okozzon túl nagy erőfeszítést a kipróbálás utáni eltávolítás.
Az előbbiekben felvetett és még jó néhány hasonló problémára adhat megoldást a *Klik*.

Ismerkedés

A *Klik* nem csak egy újabb csomagformátum a csomagkezelők- és formátumok tengerében. A *Klik* lényege az *AppDir* paradigma: egy alkalmazás – egy könyvtár, az alkalmazás könyvtára tartalmaz mindent, ami a futtatásához szükséges. A *Klik* lehetőséget ad arra, hogy a kiválasztott alkalmazást egyszerűen letölthessük és használhassuk úgy, hogy az alkalmazás minden eleme a saját könyvtárába kerül (valójában csupán egyetlen fájlba, de erről később). Ez azt jelenti, hogy a rendszerünk más részeibe nem avatkozik bele, nem másol *library*-kat sem más állományokat sehová. Egyedül az alkalmazások beállításai kerülnek tárolásra a rendszerünkön, ill. a hozzá tartozó adatok, például egy levelezőalkalmazás esetén a levelezőszerverek beállításai és levelesláda.
Túl szépnek hangzik, hogy igaz legyen? Részben igen, ugyanis a *Klik* folyamatos fejlesztés alatt áll, és még számos funkció megvalósításra vár. Elöljáróban le kell szögeznünk annyit, hogy a fejlesztők álláspontja szerint a *Klik*-nek nem célja, hogy teljes értékű csomagkezelő legyen, nem arra szánják, hogy egy teljes rendszer összes alkalmazását telepítsük vele. Az elsődleges cél az, hogy könnyen legyen lehetőségünk egyes alkalmazások telepítésére, ha tesztelni akarunk egy-egy új vagy frissebb változatot, vagy hogy gyorsan hozzáférhessünk olyan alkalmazásokhoz, amelyeket esetleg a használt disztribúció (még) nem tartalmaz. Még egy fontos pont, hogy a *Klik*-en keresztül történő alkalmazás-telepítés teljesen disztribúciófüggetlen, és csak egy Web-böngészőre van szükség hozzá.
Ezek után térjünk rá a *Klik* telepítésére és használatára, majd a bemutatás után megpróbáljuk összefoglalni előnyeit-hátrányait.

A Klik telepítése

Léteznek olyan *Linux* disztribúciók, amelyek alapértelmezésben is támogatják a *Klik* használatát. Ilyen például a *Knoppix*, ami egy *Debian* alapokon nyugvó disztribúció.
Ilyen disztribúciók esetén az alábbi telepítésre nincs szükség.
A *Klik*-et a honlapjáról érhetjük el. Ugyanitt hasznos dokumentációt is találunk a *Klik*-ről és használatáról (dokumentáció, *Wiki*). A telepítés egyszerű, egy telepítőszkript segítségével történik, ahogy azt a honlap főoldalán *(1. ábra)* meg is találjuk:

```
wget klik.atekon.de/client/
➥install -O -| sh
```

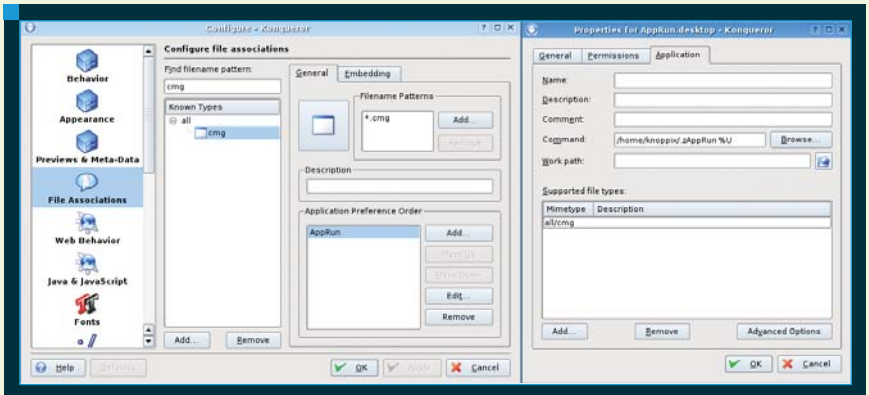
A lényeg, hogy le kell töltenünk a ➲ *klik.atekon.de/client/install* telepítőszkriptet és el kell indítanunk, az előbbi *wget*-es megoldás ezt teszi egy lépésben.
A telepítés során egy *.zAppRun* nevű szkript kerül a felhasználó könyvtárába, ami a későbbiekben letöltött *Klik*-es alkalmazások futtatásáért lesz felelős.
Még egy dolog történik a telepítéskor, mégpedig az, hogy a Web-böngészőkbe bejegyzésre kerül a *klik://* protokoll (az alkalmazások letöltésénél van rá szükség) ill. a *Klik*-en letöltött alkalmazás-fájlok (*.cmg* kiterjesztésűek) *MIME*-típusának bejegyzése (a 2. *ábra* ezt

Konqueror böngésző esetén illusztrálja). Mindez automatikus és gyors.
Ezek után készen állunk az alkalmazások telepítésére.

Alkalmazások használata Klik-el

A fejezet címében szándékos a „telepítés" helyett a „használat" szó. Mégpedig azért, mert amint azt látni fogjuk, a ***Klik***-es alkalmazások használatához nem igazán van szükség megszokott telepítési lépésekre.
Alkalmazásokat a ***Klik*** honlapjáról érhetünk el. Amint azt az *1. ábra* is mutatja már a főoldalon láthatók a főbb kategóriák. Jelenleg több ezer alkalmazás érhető el ***Klik***-en keresztül. Lássuk az alkalmazások használatát egy konkrét példán keresztül.
Tegyük fel, hogy nincs a jelenlegi ***Linuxunkon*** egy használható szövegszerkesztő, és sürgősen szükségünk lenne egyre, mondjuk az ***AbiWord***-re. A ***Klik*** honlapján keressük meg és válasszuk ki az ***Editors*** kategóriát *(3. ábra)*, azon belül pedig az ***AbiWord***-öt *(4. ábra)*. Az alkalmazás leírása tartalmazza azt, hogy milyen elemekből áll össze, és az oldal alján található a letöltési/telepítési link. Erre kattintva egy dialógus fogad ***(5. ábra)*** ami megerősítést kér, hogy valóban le szeretnénk tölteni az adott alkalmazást.
Jelenleg csak a ***Klik*** honlapjáról érhetők el hivatalos ***Klik***-alkalmazások. Ennek elsődleges oka a biztonság fenntartása és a csomagok minőségének megőrzése.
Miután beleegyeztünk a letöltésbe *(6. ábra)* a háttérben megtörténik az alkalmazás letöltése és a letöltés végén az alkalmazás el is indul *(7. ábra)*.
És most néhány fontos kérdés: hová kerül a letöltött alkalmazás, milyen formában, hogyan érhető el, hogyan távolítható el? Vegyük sorra ezeket.
Klik-es letöltéskor valójában egy darab ***.cmg*** kiterjesztésű fájl keletkezik az asztalunkon (***desktop***, *8. ábra*), amit természetesen bárhová áthelyezhetünk. Ez egy ***image*** fájl, ami minden könyvtárat és fájlt tartalmaz amire az alkalmazásnak szüksége van a futáskor. Amikor ezt az alkalmazást elindítjuk, valójában a ***/tmp/app*** időleges könyvtárba ***loop*** eszközként csatolódik fel, ami az alkalmazás bezárásakor eltűnik. Ebből az időlegesen felcsatolt könyvtárból indul el maga az alkalmazás. Az alkalmazás ***.cmg*** fájljára kattintva ez mind transzparensen, a felhasználó számára láthatatlanul történik. Ha kézzel szeretnénk elindítani az adott alkalmazást, akkor a ***home*** könyvtárunkban található ***.zAppRun*** szkriptnek az alkalmazás ***.cmg*** fájlját paraméterként megadva tehetjük.
Az alkalmazás eltávolítása nagyon egyszerű művelet: lépjünk ki az alkalmazásból és töröljük le az alkalmazás ***.cmg*** fájlját.
Az előbbiekből kiderül a ***Klik*** egy másik nagy előnye, mégpedig az, hogy minden felhasználó használhat ***Klik***-es alkalmazásokat (ha például nincs ***root*** jogosultsága és nincs telepítve az adott alkalmazás), mert minden alkalmazás csupán egy darab ***.cmg*** fájlból áll, amit a felhasználó a saját könyvtárába letölthet, futtathat és törölhet.
Az előbbi letöltési/telepítési módszer mellett más lehetőségünk is van.
Ha például tudjuk a használni kívánt alkalmazás nevét, de nem tudjuk melyik kategóriában van, vagy csak nincs időnk végigböngészni, akkor a ***Klik*** főoldalán található keresőt is használhatjuk *(9. ábra)*. Az alkalmazás

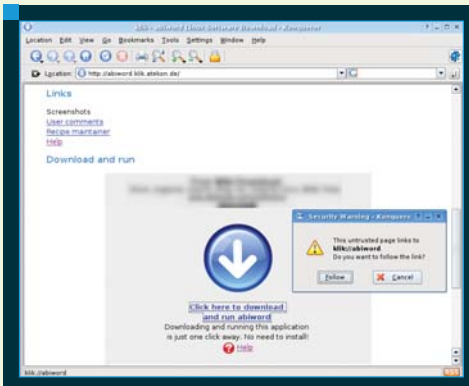
1. ábra A Klik honlapjának főoldala, az alkalmazás-kategóriák egy részével



2. ábra A Klik-alkalmazások .cmg fájljainak hozzárendelése az őket futtató zAppRun szkripthez (automatikusan történik)

3. ábra Az Editors (szerkesztők) kategória

4. ábra Az AbiWord alkalmazás információs lapja és letöltés (reklám elmosva)



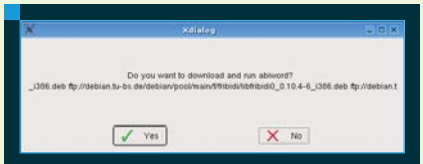
5. ábra Letöltés helyének megerősítése

megtalálása után a letöltés menete az előbbi ***AbiWord***-ös példával azonos. Még egy lehetőség adott a ***Klik***-es alkalmazások letöltéséhez, amelyik az eddigiek közül a legpraktikusabb, feltéve, hogy tudjuk az alkalmazás nevét. A *10. ábra* illusztrálja, ahogy ***Konqueror*** böngészőben a ***klik:// abiword URL***-t beírva automatikusan elindul az alkalmazás letöltése. Letöltés után minden ugyanúgy történik, mint az előző példákban.

Mivel az alap-filozófia a ***Klik*** esetén az, hogy az alkalmazás futtatásához szükséges minden hozzávaló benne legyen az alkalmazás ***.cmg*** fájljában, egy újabb nagy előnyre is tudunk példát adni. Tegyük fel, hogy a felhasználó ***KDE*** grafikus asztali környezetet használ és semmilyen ***Gnome*** környezetet használó alkalmazás nincs telepítve, sem pedig a ***Gnome***-os alkalmazások által használt ***GTK*** könyvtárak. Tegyük fel ekkor, hogy a felhasználó gyorsan szeretne egy ***Gnome***-os alkalmazást futtatni aztán törölni, de nem szeretné ehhez tucatnyi egyéb könyvtárat telepíteni, mert túl sok időt venne igénybe ezek telepítése majd eltávolítása. Ilyen esetben nyugodtan használható a ***Klik*** – csak letölti az alkalmazást az előbbi módszerek valamelyikével egy darab ***.cmg*** fájlban, futtatja, majd törli a fájlt. Lehet ez a *7. ábra* ***AbiWord***-je, vagy a ***C/C++*** fejlesztői környezet ***Anjuta*** *(11. ábra)*, vagy bármi más.

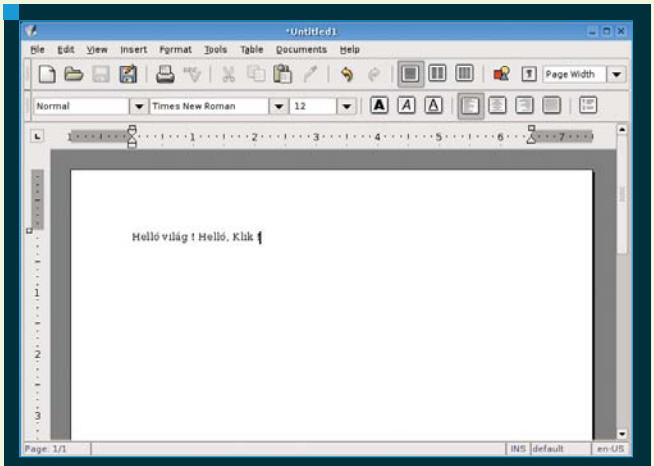
Előnyök, hátrányok

Kezdjük először a hátrányokkal, amelyek többsége igazság szerint abból adódik, hogy ***Linuxok*** csomagkezelőihez próbáljuk hasonlítani a ***Klik***-et. Minthogy a ***Klik*** nem teljes rendszerek csomagkezelésére lett kitalálva, és jelenleg ez nem is cél, ezért nem tudunk egy teljes rendszert ***Klik***-alapokon telepíteni és működtetni. Hátrány – a szokványos telepítésekkel szemben természetesen –, hogy mivel minden alkalmazás a saját ***.cmg*** fájljában tartalmazza a szükséges könyvtárakat, sok ***Klik***-es alkalmazás telepítése érezhetően sok lemezterületet igényelhet. Esetenként hátrány lehet

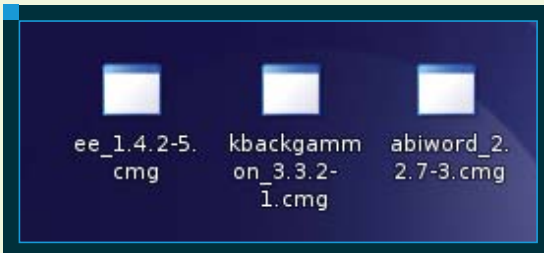


6. ábra Az AbiWord csomagok letöltésének megerősítése

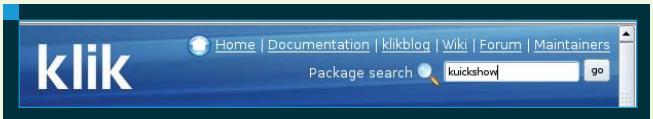
az is, hogy szükségünk van egy viszonylag gyors internetkapcsolatra, hogy ne tartson túl sokáig egy új alkalmazás letöltése. Hátrány lehet, hogy csak ***i386*** architektúrát támogatnak, tehát csak erre fordított csomagok érhetők el jelenleg (az összes 32 bites ***Intel***-kompatibilis architektúra). Ami még egy lényeges hátrány lehet az az, hogy mivel az egyes alkalmazások ***.cmg*** fájlja az alkalmazás indításakor felcsatolódik a ***/tmp*** könyvtárban, egyszerre alapesetben csak nyolc alkalmazás futtatható ily módon. Ez a határ a ***loop device***-ként felcsatolható állományok számára vonatkozó rendszerszintű határszám (ezt a ***Linux*** kernelben lehet átállítani).



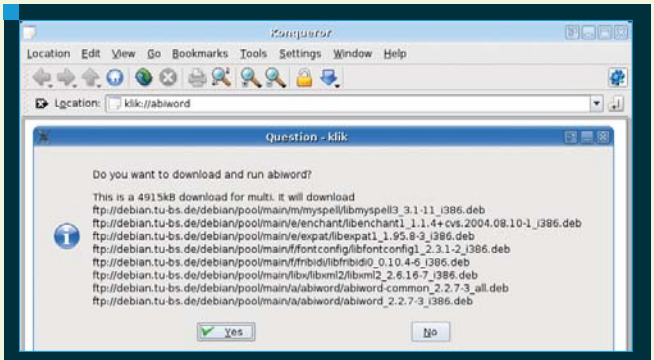
7. ábra Letöltés után rögtön indul is az alkalmazás



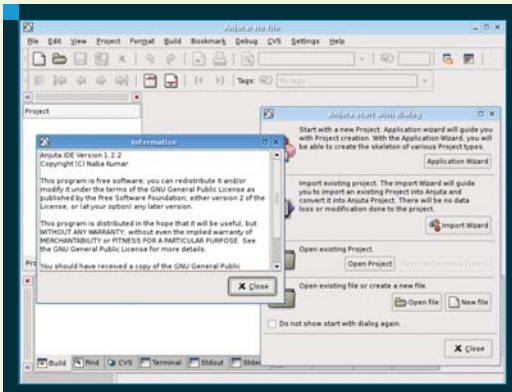
8. ábra Letöltött .cmg fájlok: mindegyik egy-egy teljes alkalmazás



9. ábra Alkalmazás keresése a Klik főoldalán



10. ábra Egyszerű Klik alkalmazás-telepítés Konqueror-ból. A csomag-lista a .cmg fájl összeállításához használt csomagokat mutatja.



11. ábra Klik-ről letöltött és elindult Anjuta C/C++ fejlesztői környezet

Ezek után térjünk át az előnyökre. A *Klik*-et bármilyen jogosultságú felhasználó használhatja, mivel a rendszer többi részéhez nincs szükség hozzáférésre. A letöltött *.cmg* fájlok hordozhatók, tehát bárhol tudjuk őket futtatni, ahol van telepített *Klik* támogatás. Fontos szempont, hogy csupán webböngészőre van szükségünk az alkalmazások letöltéséhez. Nagy előny, hogy ha például *Gnome* környezetet használunk és szeretnénk egy rövid pillantást vetni a legfrissebb *KDE* környezetre, akkor *Klik*-en letölthetjük, kipróbálhatjuk és letörölhetjük.
A lényeg az, hogy a rendszerünk megváltoztatása nélkül futtathatunk alkalmazásokat. Ez kezdő és haladó felhasználóknak egyaránt jelenthet jó hírt, akár alkalmazások teszteléséről van szó, akár olyan alkalmazás gyors eléréséről, amelyik (jelenleg) nem elérhető rendszerünk csomagformátumában, vagy csak időlegesen van rá szükségünk. Fejlesztők számára is jó eszköz lehet arra, hogy *.cmg*-be csomagolt változatot készítve saját alkalmazásaikból könnyen adjanak lehetőséget a felhasználó-tömegnek az alkalmazás gyors kipróbálására és tesztelésére.

Végszó

Mindent figyelembe véve a *Klik* nem a csomagkezelők Szent Grálja. Ennek ellenére, vagy inkább ezzel együtt egy olyan eszközt ad a fejlesztők/csomagkészítők, az alkalmazás-tesztelők ill. az átlagos felhasználók kezébe, amellyel nagyon sok esetben megkönnyítheti az életünket. Természetesen nem is egy világrengető csoda, még csak nem is új ötlet: csupán egy jó - és ami fontosabb – könnyen használható, működő megoldás, amelyre sokan és régóta vártak. A linuxos közösségben megjelenésekor eléggé nagy figyelmet keltett, éppen a felsorolni próbált előnyei miatt.
Reméljük, hogy ezzel a rövid bemutatóval felkeltettük az olvasó érdeklődését egy „próbaútra”. Próbaútra, ami nem csak magának a *Klik*-nek a kipróbálására szól, hiszen ez csupán egy eszköz, hanem a *Klik*-en keresztül elérhető több ezer alkalmazás használatára.

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

- http://klik.atekon.de/
- http://klik.atekon.de/docs
- http://klik.atekon.de/wiki

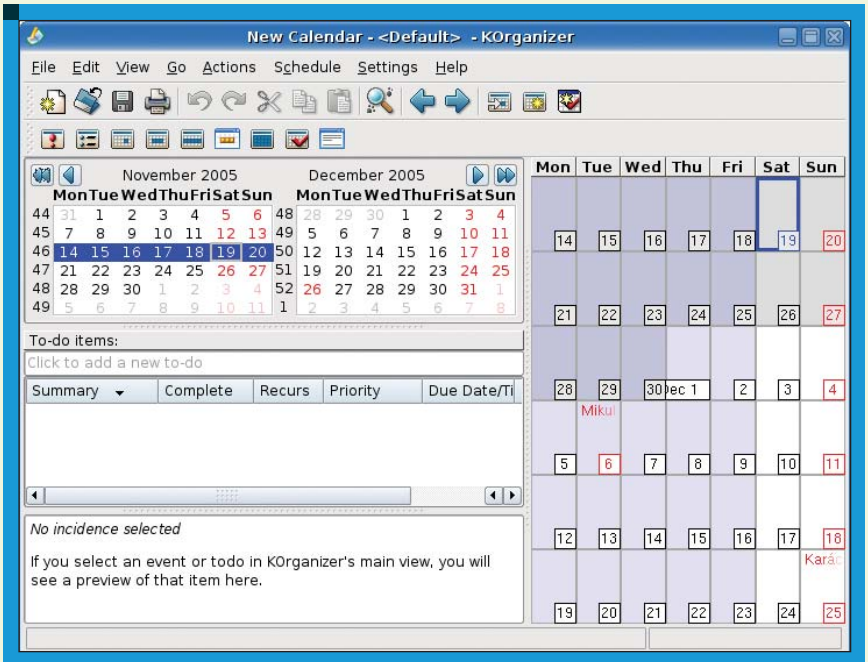
Korganizer – Segítség a szervezéshez

A KOrganizer alapvetően egy KDE asztali grafikus felület alá írt sokoldalú digitális határidőnapló-alkalmazás. Újabban a szintén KDE-s Kontact PIM – „personal information manager”, személyi-információ kezelő – központ szerves része, de mint különálló alkalmazás is elérhető és használható.

A *Kontactot* bemutató előző írásom után ebben a cikkben áttekintem a *KOrganizer*, mint különálló határidőnapló alkalmazás minden elemét, és megpróbálok minél teljesebb képet adni erről az alkalmazásról. Célom az, hogy a napló- és határidőnapló alkalmazások tengerében bemutassak egy olyan alternatívát, amely napi használatra alkalmasan minden fontosabb funkciót tartalmaz, könnyen megtanulható és használható felületbe ültetve, ingyenesen használható.

Ismerkedés

Ebben az írásban a *KOrganizer 3.4.2* verzióját mutatom be, mivel jelenleg a legtöbb *Linux* disztribúció ezt tartalmazza. Minden olyan disztribúcióban, amely tartalmazza a *KDE* felületet, a *KOrganizer* is elérhető és telepíthető. A csomag neve minden esetben *„korganizer-verzió.formátum”* alakú, ahol a *„verzió”* a verziószámot, a *„formátum”* a disztribúció csomagformátumát jelöli (*deb*, *rpm*, stb.). Noha a *KOrganizer* alkalmazás elérhető, telepíthető és használható mint különálló egység, a *KDE 3.4.x* sorozatában a *Kontact PIM* alkalmazás részeként is elérhető. Itt most a különálló *KOrganizer* alkalmazásra koncentrálunk.
Telepítés után a *KOrganizer* egy üres határidőnapló- és tennivaló-listával fogad, *1. ábra*.
A *KOrganizer* egy gyorsan kezelhető és könnyen átlátható felületet ad feladatok, találkozók, tennivaló-listák, határidős feladatok, stb. bejegyzésére és kezelésére. A felület baloldalán négy elemet találunk:



1. ábra A KOrganizer első képernyője, üresen

- *Hónap-naptárak*: tetszőleges hónapot kikereshetünk, egy vagy több napot kijelölhetünk, amelyekhez aztán feladatokat rendelhetünk.
- *Tennivalók listája*: bejegyzett tennivalók listája, teljesítési állapottal, határidőkkel, prioritás- és kategória-megjelöléssel.
- Az aktuálisan kijelölt bejegyzett feladat részletes leírása.
- Az aktuálisan kezelt és megjelenített lokális és távoli határidőnaplók szerkeszthető listája.

A felület jobboldalán napos, háromnapos, munkanapos, hetes vagy hónapos megjelenítési módban a kiválasztott időszak órái/napjai láthatók. Az aktuális nézet formátumát a *View (Nézet)* menü menüpontjaival ill. az alkalmazás eszköztárának elemeivel választhatjuk ki. Feladatokat, találkozókat, stb. óra-perc pontossággal tudunk definiálni és megjeleníteni is.
Összességében a *KOrganizer* a következő feladatokra ad megoldást:

- A felhasználók által szabadon definiált kategóriák szerinti típusú feladatok bevitele a határidőnaplóba

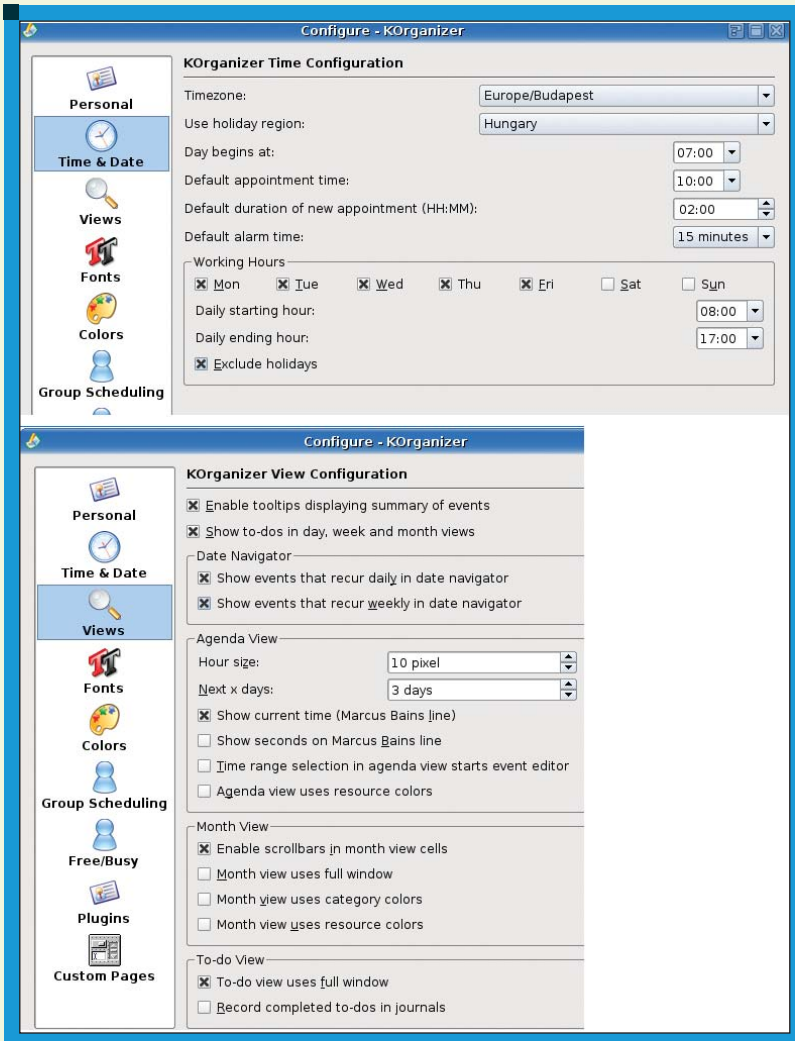
(alapesetben pl. találkozó, születésnap, üzleti találkozó, konferencia, szabadság, telefon, stb.).

- A bevitt feladatokhoz értesítéseket rendelhetünk, szöveges és/vagy hangos formában, pontosan megadható időzítéssel.
- Nyilvántarthatunk úgynevezett. tennivaló-listákat (***To-do List***), amelyek olyan feladatokat jelölnek amelyeket egy adott határidőre teljesíteni kell; a feladatok mellett vizuálisan nyomon követhető teljesítési mérték kijelzést tesz lehetővé.
- Személyes napló (***Journal***) vezetése: a ***View*** menüből vagy az eszköztárból válthatunk erre a nézetre, amiben minden kiválasztott naphoz szöveges napló-bejegyzéseket készíthetünk; ezek könnyen visszakereshetők, olvashatók, szerkeszthetők.
- Lokális és távoli határidőnaplók kezelése, külön, vagy ugyanazon a naptáron belül; ezen kívül lehetőségünk van csatlakozni ***Microsoft Exchange 2000***, ***Novell Groupwise***, ***SuSE OpenExchange***, ***eGroupware*** vagy ***OpenGroupware*** szerverekhez ezek szolgáltatásainak eléréséhez.

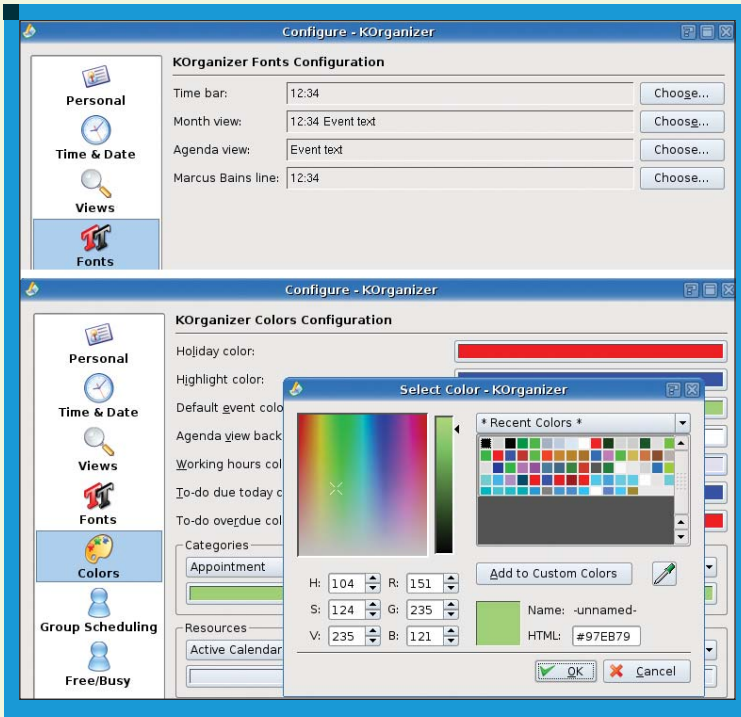
Beállítások

A ***Settings->Configure KOrganizer*** (***Beállítások->KOrganizer beállításai***) menüpontban számos olyan paramétert állíthatunk be, amelyek egyrészt a ***KOrganizer*** kinézetét, másrészt a kezelhetőségét befolyásolják. A *2. ábra* a megjelenítés idő-paramétereinek ill. a találkozók megjelenítésének paramétereit mutatja, míg a *3. ábra* a megjelenítések betűtípusait ill. a különféle találkozó- és feladat-kategóriák megjelenítésénél használandó színek beállítását szemlélteti. Ezek a beállítási lehetőségek, noha elsőre talán nem tűnnek létfontosságúnak, sok feladat és találkozó bevitelekor, ill. nagy mennyiségű adat vizuális megjelenítésekor különleges értelmet nyernek. Ahhoz, hogy egy zsúfolt napi, heti vagy havi határidőnapló gyorsan áttekinthető és könnyen értelmezhető legyen fontos a feladatok színekkel történő konzisztens elkülönítése.

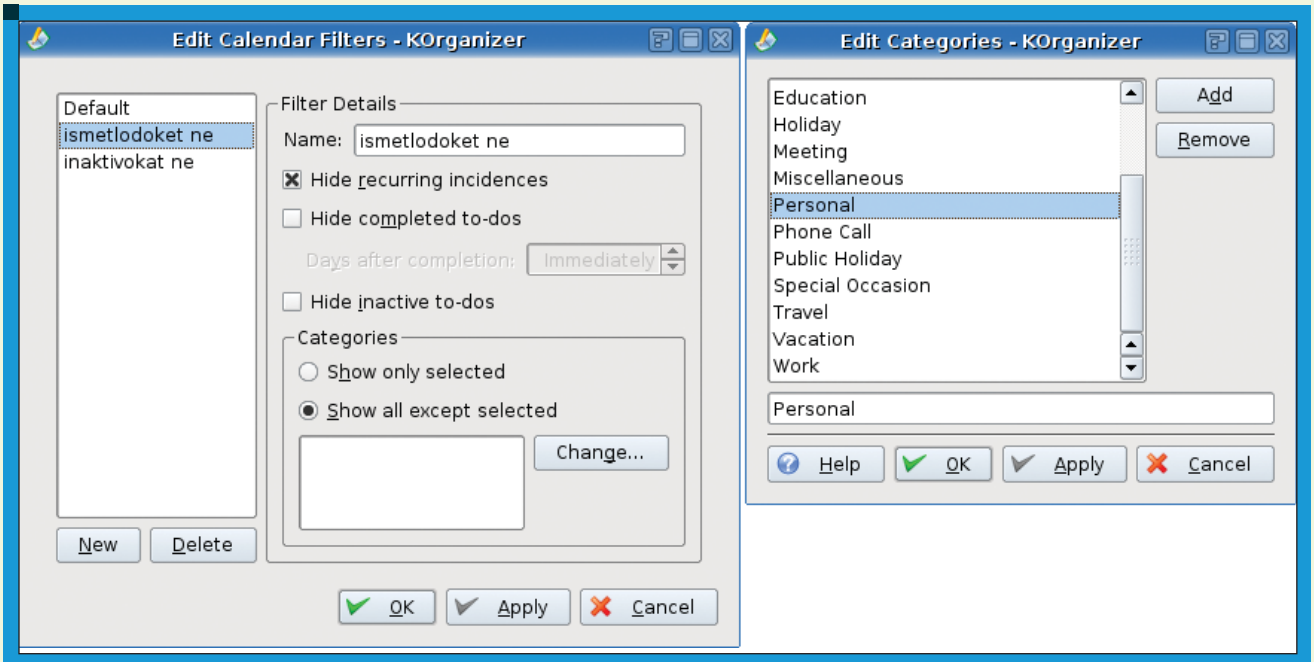
A ***KOrganizer*** tartalmaz egy alapértelmezett feladat/találkozó kategórialistát és a hozzájuk rendelt színeket, de ezeket viszonylag rövid használat után



2. ábra Idő-paraméterek és megjelenítési paraméterek beállítása



3. ábra Megjelenítési betűtípusok és a találkozó-kategóriák színeinek beállítása



4. ábra Szűrések és kategóriák szerkesztése

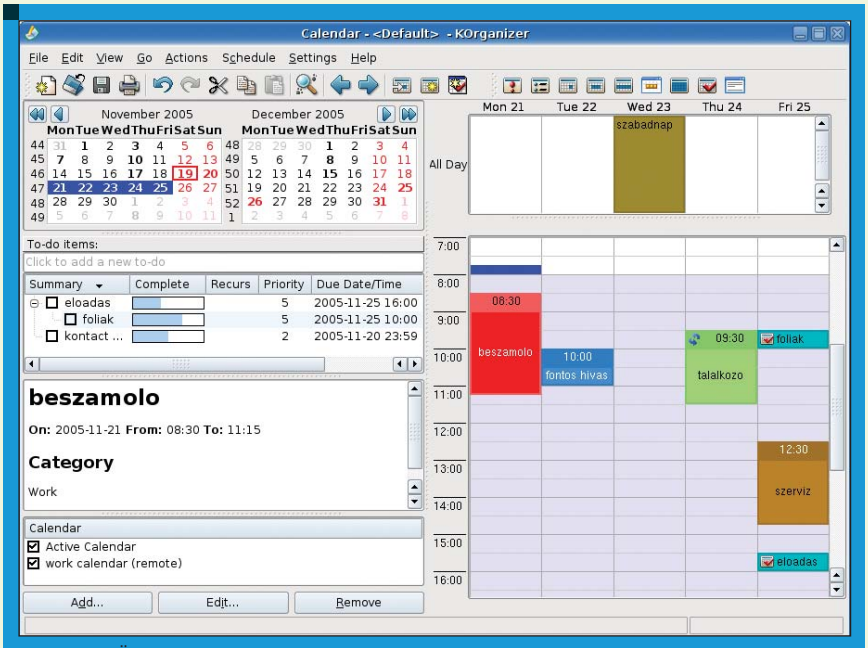
már szinte biztosan személyre kell szabnunk. Ez egyaránt jelenti a nekünk megfelelő kategóriák kialakítását ill. ezekhez egyedi színek hozzárendelését. Egy jó példa a vizuális elkülönítés hasznosságára az, hogy például ha egy feladat több lépésből áll amelyek több különböző időpontban történnek, akkor ezeket ugyanazzal a színnel jelölve egy hosszabb időintervallum áttekintésekor is könnyen követhető reprezentációt kapunk. A ***Settings (Beállítások)*** menü ***Edit filters (Szűrők szerkesztése)*** arra ad lehetőséget, hogy szűrhessük az egyes feladatok és események megjelenítését. A *4. ábrán* látható szűrő például elrejti a naptárról az ismétlődő eseményeket. Az ***Edit categories (Kategóriák szerkesztése)*** ad lehetőséget arra hogy saját céljainknak megfelelő egyéni kategóriákat hozhassunk létre, az események és feladatok saját igényeink szerinti kategorizálásához (*4. ábra* jobboldala).

Találkozók/események kezelése

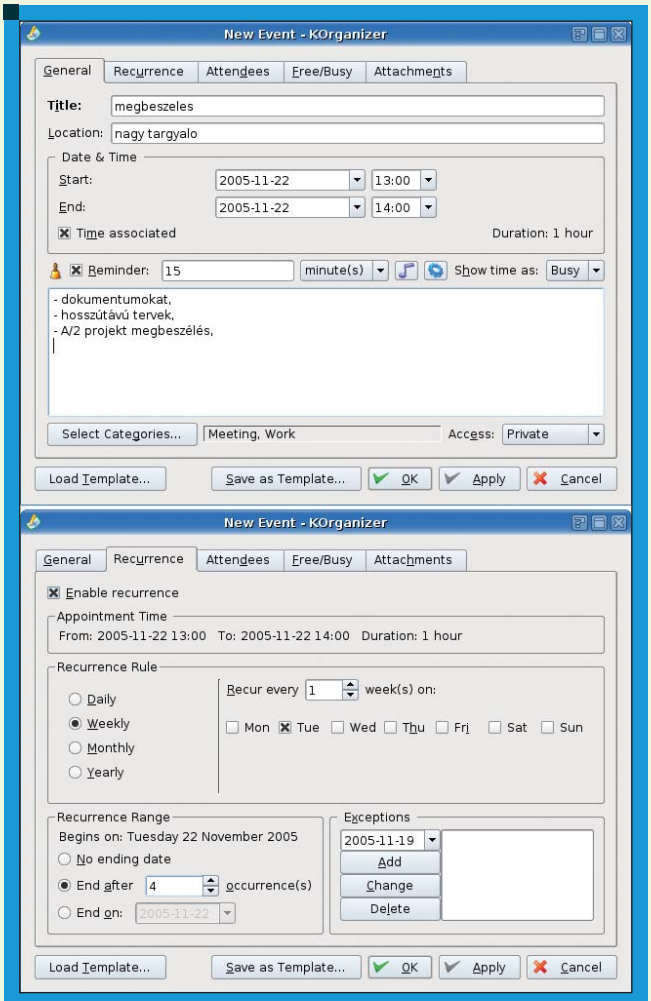
A *5. ábra* egy ötnapos hétfő-péntek munkahét néhány feladatát mutatja naponként, óra-szintű lebontásban. Új találkozó/esemény felvételéhez az ***Actions->New event*** menüpontot használhatjuk, vagy a naptár-nézetben egy adott idő-intervallumot kijelölve majd jobb-klikket nyomva a ***New event*** menüpontot kiválasztva. Az új találkozó/esemény bevitelekor szabad kezet kapunk az időpont, a téma, a hely megjelölésében. Ha nem egyszeri, hanem adott szabály szerint ismétlődő eseményről van szó, akkor az új eseményt megadó dialógusban a második fülre kattintva az ismétlődési adatokat is megadhatjuk (*6. ábra*). Ezen kívül a többi füllel összeállíthatjuk a résztvevők listáját, akiket e-mailben értesíthetünk, ill. fájlokat is csatolhatunk az eseményhez, amelyeket akár el is küldhetünk a résztvevőknek. Természetesen előzetes figyelmeztetést is kérhetünk a találkozóról, jelen esetben tizenöt perccel előtte kapunk egy szöveges figyelmeztetést. A *7. ábra* az előző ábrán létrehozott találkozó kivonatát mutatja a ***KOrganizer*** felületén.

Feladatok kezelése

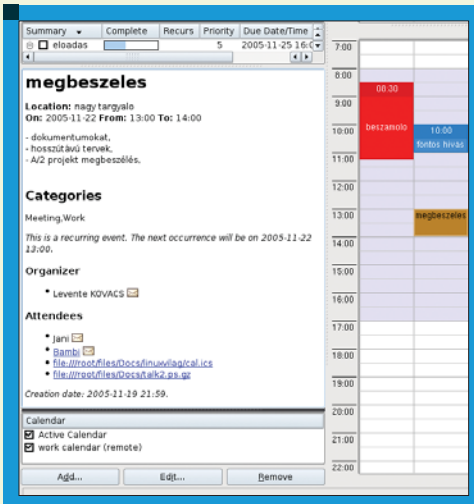
A ***KOrganizer*** lehetőséget ad úgynevezett feladat-listák (***To-do items***) nyilvántartására. A feladatok abban különböznek az eseményektől/találkozóktól,



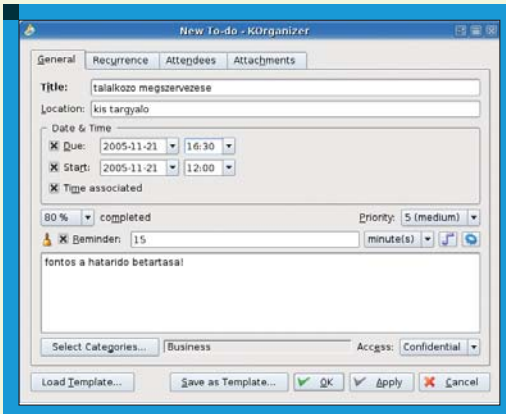
5. ábra Öt munkanap egyes feladatai órás lebontásban



6. ábra Új esemény megadása



7. ábra Létrehozott találkozó részletei

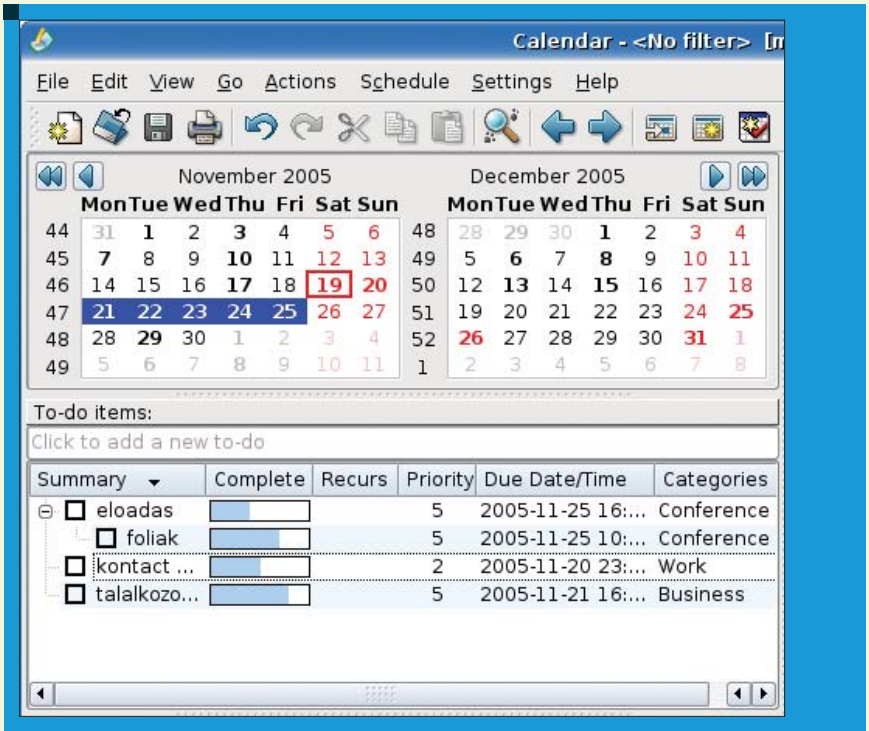


8. ábra Feladat („To-do") létrehozása

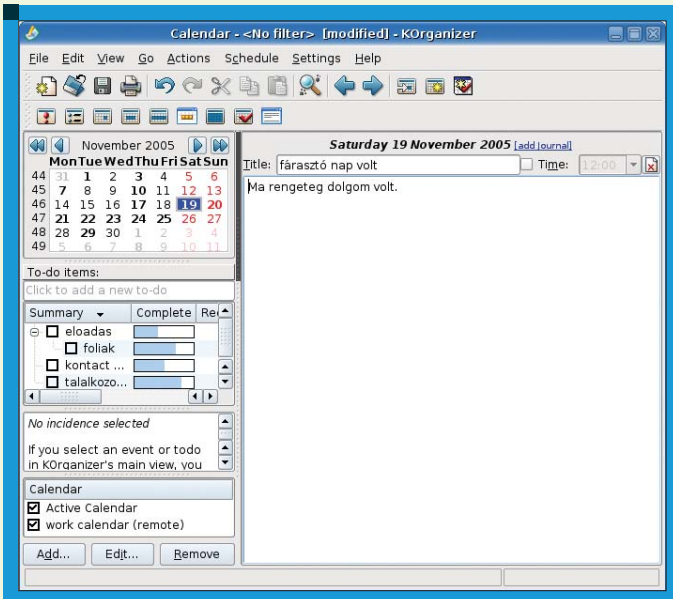
hogy míg azok adott időpontban bekövetkező eseményeket rögzítenek, addig a feladatok egy olyan folyamatot írnak le, amely valamikor elkezdődött, valamilyen mértékű teljesítési szinten áll jelenleg és van egy várható időpont, amikorra várjuk a teljesítést. Ennek a folyamatnak a vizuális rögzítését teszi lehetővé a feladat-lista, amelynek kinézetére példa látható a *5. ábrán* a ***To-do items*** panelen. Feladatok hozzáadása az események hozzáadásához nagyon hasonló módon történik (*8. ábra*). Az aktuális feladatok a feladatlistában követhetők nyomon (***To-do items***) (*9. ábra*).

Naplóbejegyzések

A ***View- >Journal (Nézet->Napló)*** menüpont, vagy az eszköztár ***Journal*** elemével válthatunk a napló nézetre. Ekkor a ***KOrganizer*** bal felső sarkában a naptáron kiválasztott tetszőleges naphoz szerkeszthetünk szöveges naplóbejegyzést (*10. ábra*).



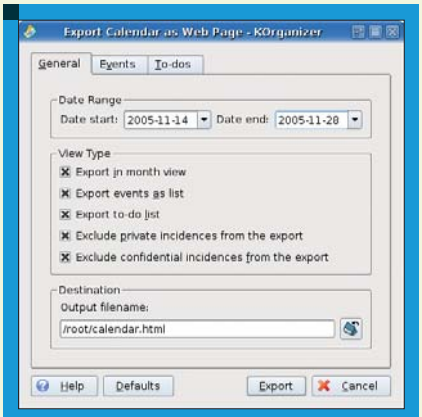
9. ábra Feladatok listája („To-do items")



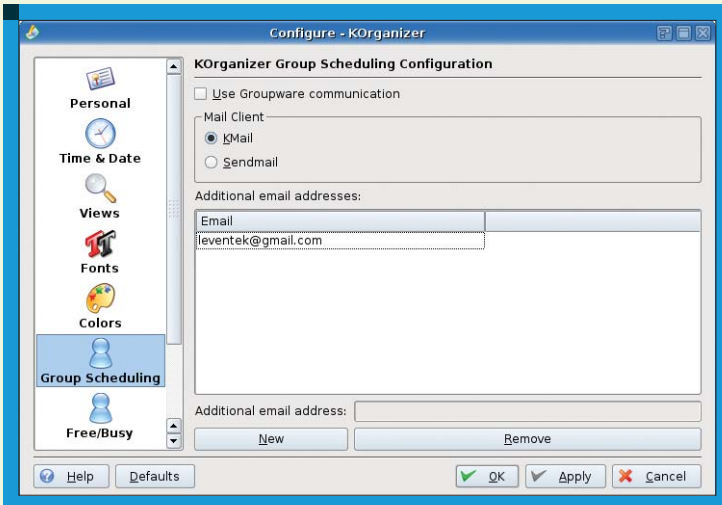
10. ábra Naplóbejegyzések

csomaggal együtt kerül telepítésre) amelynek az az előnye, hogy a *KOrganizer* elindítása nélkül is értesít minket a közeledő határidőkről (az előzetes értesítési beállítások szerint). A *KOrganizerhez* természetesen a weben is elérhető sok-sok hasznos dokumentáció.

Ezzel az írással az volt a célunk, hogy bemutassuk az olvasónak a *KOrganizerben* rejlő lehetőségeket, hogy megpróbáljuk felkelteni az olvasó érdeklődését, hogy kipróbálja ezt az egyszerű, de sokoldalú, és nyílt forrású határidőnapló-alkalmazást. Ha a kipróbálás után esetleg egy olyan alkalmazást ismernek meg a *KOrganizerben*, amit érdemesnek találnak a továbbiakban is használni, az már csak hab a szabad forrású szoftverek és a *KDE* fejlesztők tortáján.



11. ábra Adatok exportálása



12. ábra Adott a csoportos időzítések/értesítések lehetősége

A naplóbejegyzések is az aktuális *calendar* fájlban (*.ics*) tárolódnak, legyen az lokális vagy távoli, mint szöveges információ.

Exportálás, értesítések

A szerkesztett teljes határidőnaplót elmenthetjük *ics* naplófájlokba, vagy a *File->Export* menüpontban *iCal*, *vCal* vagy *HTML* formátumban tudunk exportálni (*11. ábra*). Fontos tulajdonság az, hogy például egy találkozóról mindazok értesítést kapjanak, akiknek fontos a részvétel (akik meg voltak hívva, például a *7. ábrán*) és a *KOrganizerben* természetesen erre is van lehetőség. A *12. ábra* mutatja a *Settings->Configure KOrganizer* beállítás-dialógus úgynevezett *Group Scheduling (Csoportbeállítások)* beállításait. Ez a lehetőség arra szolgál, hogy létrehozott többrésztvevős esemény/találkozó esetén a résztvevők értesítést kaphassanak (*Korganizerben* alapesetben a *KMail* levelezővel történik az értesítés küldése). *Groupware* kapcsolatra is lehetőség van, az írás elején említett csoportmunka szerverek (*groupware server*) esetén.

Végszó

A előbbiekben említett funkciók és tulajdonságok mellett még egy fontosat meg kell említenünk. A *KOrganizerhez* tartozik egy a *KDE*-s panelen helyet foglaló külső kisalkalmazás (*applet*) (*KOrganizer Alarm Daemon*, az alkalmazás neve *korgac* és a *korganizer*

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

- http://korganizer.kde.org/
- http://docs.kde.org/stable/en/kdepim/korganizer/

Celestia – nézz az ég felé

A csillagászat (mint nagy hagyományokkal rendelkező tudomány) egy szűk szakértői kör „kedvtelése”, holott a tiszta, nyáresti égboltra tekintve hihetetlennek tűnik, hogy ekkora terület kevés hivatásos kutatóval rendelkezik. Amatőr csillagászok ennek ellenében szép számmal akadnak a távcsövek mögött, aki pedig csupán „passzívan” érdeklődik a téma iránt, minden bizonnyal betér néha egy könyvesboltba, ahol talán megvásárol egy csillagászattal foglalkozó könyvet.

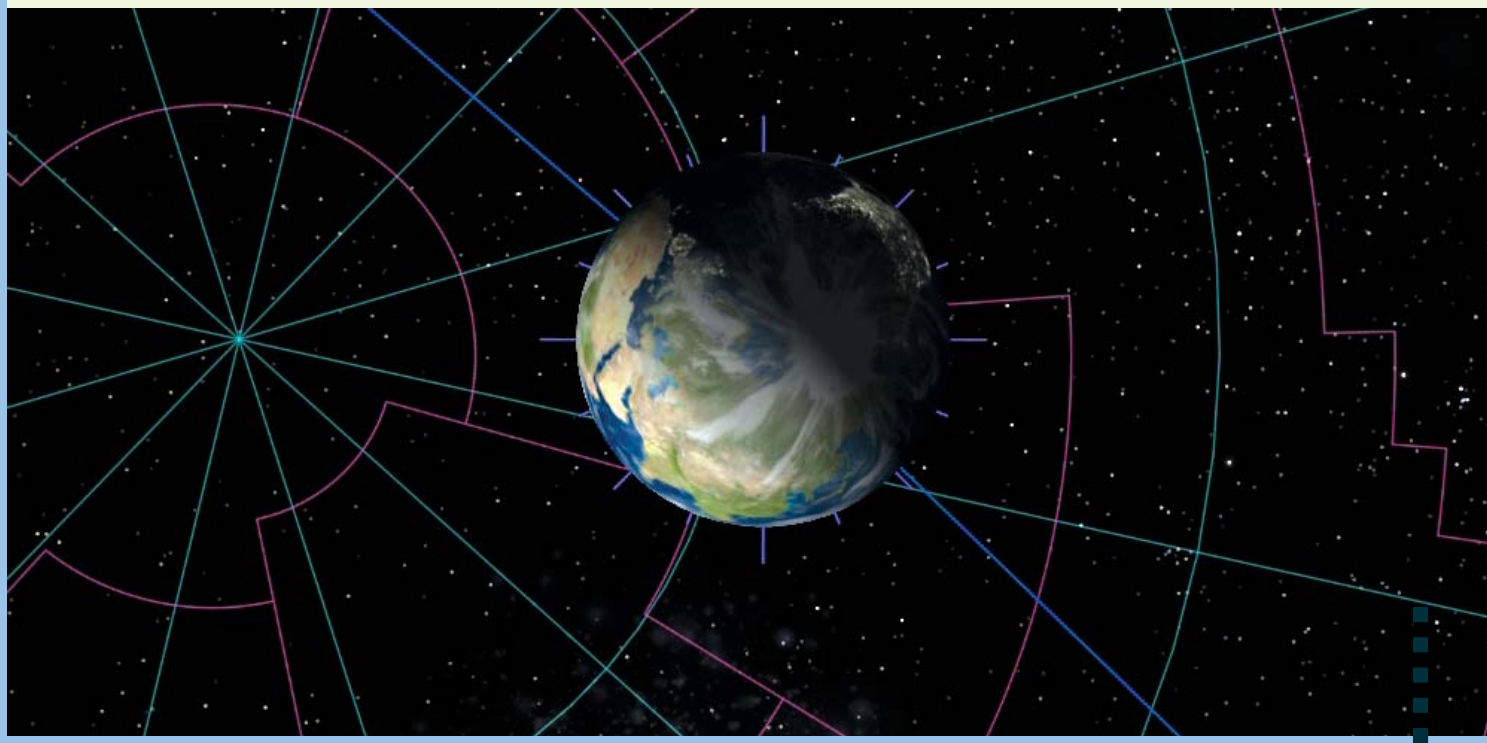
Nos, mindhárom csoportnak szeretnék kedvére tenni: egy olyan alkalmazást fogok röviden bemutatni, ami által a számítógép előtt ülve lehet megtapasztalni a tudomány eme területének egy igen erőteljes kidolgozását.

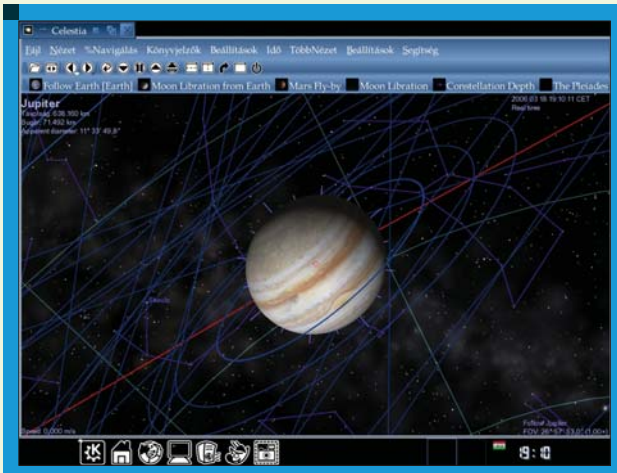
A Celestia

A ***Celestia*** nagy múltú projekt, ebből eredően kiforrott és könnyen használható. Aki még nem látta ezt a „gyöngyszemet”, nyugodtan merjen egy nagyot gondolni: látatlanban állítom, hogy senki elvárásaira sem fog rácáfolni. Nézzük hát, miről is van szó!

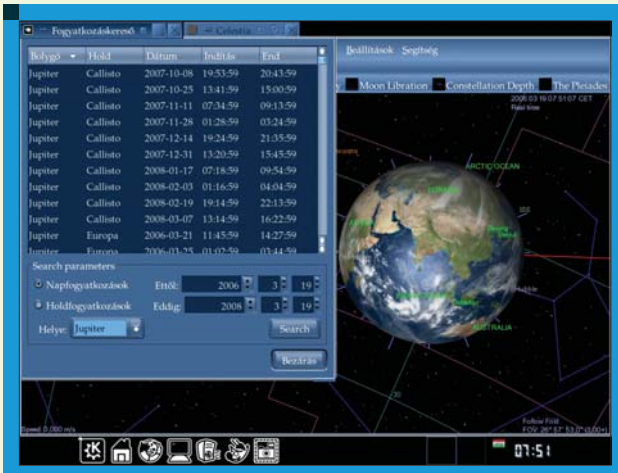
Az említett program egy olyan élethű „galaxis-szimulátor”, ahol az univerzum rádióteleszkópok és optikai csövek által feltérképezett része szabadon beutazható, ***Földünktől*** igen nagy ***Csillagászati Egységnyi*** távolságokra is. A modellezett világ aprólékos és hiteles: azok a virtuális égitestek, melyek eredetijének felszínéről kutatóink képi információval rendelkeznek, ott a modellekre valósághű textúrákat feszítettek, miközben a leképezett világ időtényezője is nagy pontosságú, így az időnek megfelelő fényviszonyokat figyelhetünk meg bármely felületen.

Fontos, hogy a ***Celestia*** belső tere szabad nézőpontból csodálható, miközben kérésünkre képes a bolygók és holdak pályáit, elfordulását is követni. Mindemellett akár extra kéréseknek is eleget tesz: tudni szeretnéd, mikor kell az égre nézned ***2007***-ben, ha holdfogyatkozást szeretnél látni? Esetleg a ***Vénuszon***, az elkövetkező harminc évben hol lesz napfogyatkozás? Nem probléma, mindezt másodpercek alatt megtudhatod, mivel a projekt menüpontjai akár ezekre a kérdésekre is választ adó kalkulátorokkal rendelkeznek. Kíváncsi vagy a ***Naprendszer*** mozgó és részletes

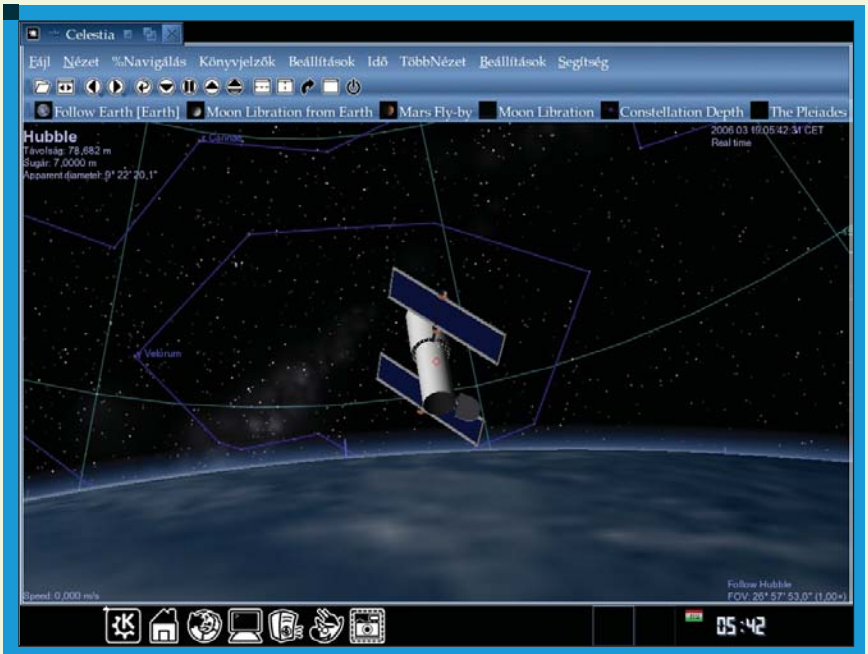




1. ábra A Celestia működés közben



2. ábra A fogyatkozások kalkulátora



3. ábra Íme, a Hubble teleszkóp

keringési képére, miközben a háttér csillagaiból megrajzolódó ***konstellációkat*** is szeretnéd látni? Hidd el, ez sem nagy kívánság.

Telepítés, beállítás

A ***Celestia*** szabad forráskódja a ➲ ***http://www.shatters.net/celestia*** címen érhető el. Mindez egy nagyjából ***30MByte*** terjedelmű ***tarball***-t jelent (melynek jelentős része textúra), amit a megszokottak szerint lehet felépíteni: a kibontott archívban kiadott

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

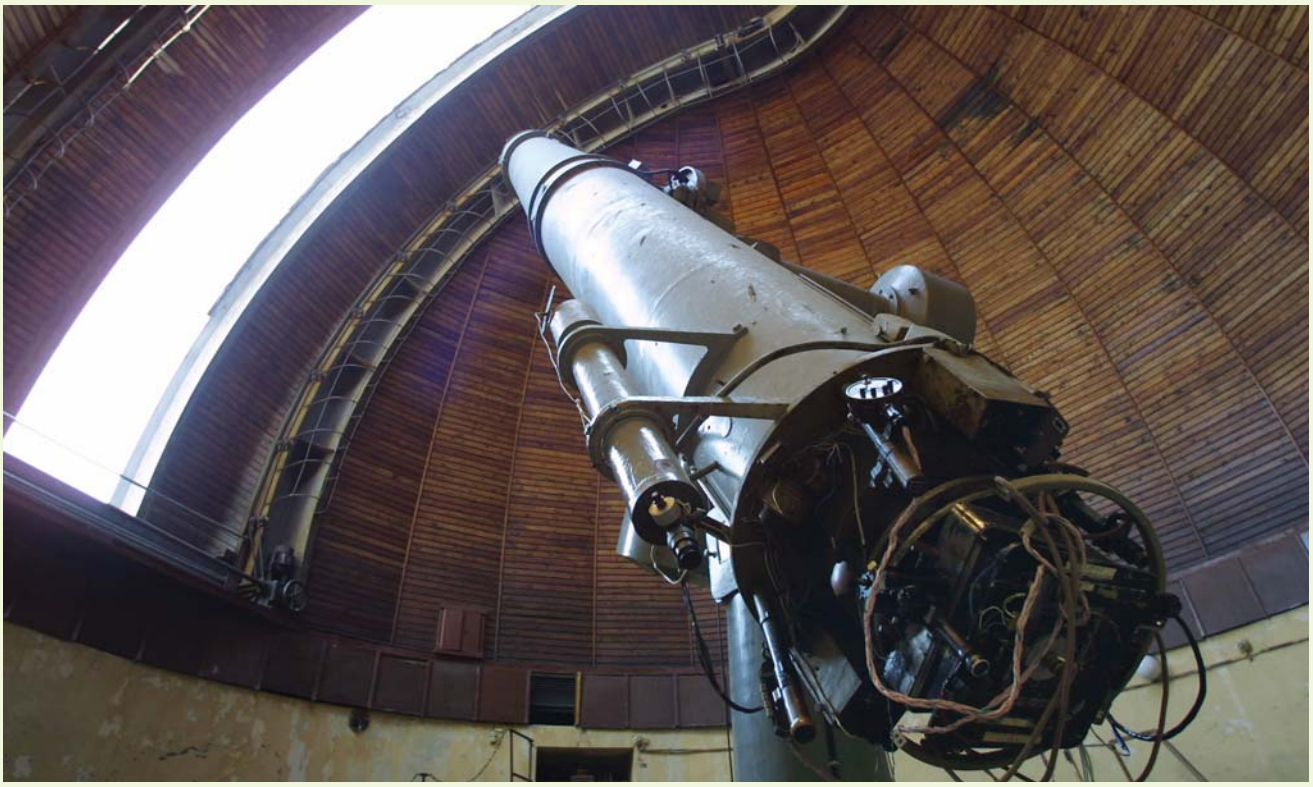
parancsok által hívhatjuk életre a kódot. Mivel megjelenítése ***OpenGL*** alapú, így grafikus hardverünk ***3D*** meghajtói jelentik az alapkövetelményt, a menük megjelenítését pedig a ***gtk***, ***glut*** esetleg a ***KDE***, ***Gnome*** bázis könyvtáraira bízhatjuk.
A menüszerkezetre és ablakozásra vonatkozó „szabályokat” még a konfigurálás során rögzítenünk kell a `configure` szkript kapcsolóival (`--with-gtk`, `--with-glut`, `--with-kde`, `--with-gnome`)
A képeken látható ablakozást és „menüzést” (mivel kezdetektől ***KDE***-párti vagyok) a jól bevált ***KDE*** felületre bíztam. A program alapértelmezés szerint ***/usr/local/share/*** útra települ (kötéssel az ***/usr/local/bin/*** mappában), indítani felhasználóként a

```
celestia
```

paranccsal lehet. Hardverigénye szerény: ***1 GHz*** processzorra támaszkodva, ***256 MByte RAM*** mellett már kompromisszumoktól mentesen fut, feltéve, hogy a megjelenítésért felelős eszköz ***GLX*** illetve ***DRI*** kapcsa hibátlanul működik.

Finomhangolás és használat

Miután kiadtuk a `celestia` parancsot, a program felépíti menüit, majd a ***Földre*** pozicionál (alapesetben ez jelenti az ***„Otthon”*** területét). Érdemes rögtön megtenni a szükséges beállításokat, ezen a téren természetesen a ***„Beállítások”*** menüpont érdemel kitüntetett figyelmet. Elsőként az ***„OpenGL útvonal”*** pontot vegyük szemügyre: grafikus kártyánk képességeihez mérten szabjuk meg a textúrázás módját! Gyakorlatilag minden ***OpenGL*** „képes” ***3D*** gyorsító képes a ***multitextúrázás*** műveletére, tehát ezt érdemes bekapcsolni, a komolyabb kártyák tulajdonosai pedig azonnal kérhetik a népszerű ***API 2.0*** verziójának aktiválását. Ezek után az ***„Objektumok és címkék”*** listáiból válogatva szükséges meghatároznunk, mely modellek legyenek feliratozva, mely keringési pályák legyenek megjelölve, mely üstökösök kerüljenek megmutatásra. Megközelítőleg harminc hasonló esetre szabhatunk paramétereket, ennek ellenére egyszerű dolgunk akad: a program nagy része lokalizálva van, így érthetően szól



hozzánk, ha a grafikus környezetünket is ebben az állapotában találja. Következő lépésben az *„Idő”* menü alatt állítsuk be a pontos időt, majd nézzük meg a *„Navigáció”* menüpontot, ahol várhatóan a legtöbb időt fogjuk eltölteni. Kiemelt fontosságú lehetőségeink: *„Otthon”*, *„Ugrás Szélességre / Hosszúságra”*, *„Ugrás a felszínre”*, *„Égi objektumok”*, *„Fogyatkozáskereső”*. Az első lehetőség az alapértelmezett (*Föld*) helyre pozicionál, a második az általunk megadott ***koordinátákra***, a harmadik pedig az éppen középpontban lévő égitest felszínére.
Az *„Égi objektumok”* felirat mögött találhatjuk a program belső böngészőjét. Valójában ez egy hosszú lista, melyen ha egy égitest holdakkal, társbolygókkal rendelkezik, akkor egy apró *(+)* jellel van megjelölve. Nézzünk egy szép, életszerű példát: kattintsunk a lista egy ismerős elemére jobb egérgombbal!
A felbukkanó helyi menü *„Ugrás”* lehetőségét használva a kiválasztott helyre „utazhatunk”. Itt az egerünk bal gombjával vonszolva tudjuk mozgatni a képet, a jobb gombbal húzva fordulhatunk el a modell körül, a két gombbal egyszerre vonszolva pedig csavarhatjuk a nézőpontot (az egérgörgőt használva beállítható a kívánt távolság is). Keressünk egy olyan nézőpontot, ahol egyszerre látszik a bolygó és ennek egy holdja is, lehetőleg úgy, hogy a kísérő legyen közelebb felénk. Kattintsunk a bolygóra bal gombbal, majd ezután jobbal: a felajánlott lehetőségek közül válasszuk a *„Követést”*, így a hold viszonyított mozgása (és esetleges forgása) látványosan megmutatkozik előttünk. A helyi menük mindenütt elérhetők, így a bal egérgombbal kiválasztott objektummal (akár egy távoli csillaggal) sok látványos dolgot megtehetünk, akár annak pályájára szinkronizálva is, így érdemes kísérletezni: lehetőségek tárháza áll rendelkezésre.
A *„Fogyatkozáskereső”* szerepét már az előzőekben leírtam, ennek bemutatása szükségtelen, olyannyira egyértelműen használható A fogyatkozási találatokon szintén előhívhatjuk a helyi menüt, ahol az *„Ugrást”* használva a fogyatkozás pontos földrajzi helyére repülünk: ekkor csupán módosítanunk kell a látószöget, majd az *„Idő”* kívánt értékre állításával megtekinthető a jelenség, annak teljes pompájában (az idő egyébként a valóstól gyorsabbra és lassabbra egyaránt állítható, ez a forgó objektumoknál látványos lehet).
A nézeteket tekintve, akár osztott képernyőn is megcsodálhatjuk a leképezett világot, ami a kölcsönhatások megfigyelésében játszhat komoly szerepet. A képernyőről egyébként könnyedén készíthető mentés, a *„Fájl”* menü megfelelő pontjának használatával, ezen felül némely ***build*** verzió rendelkezik videó rögzítési lehetőséggel is.

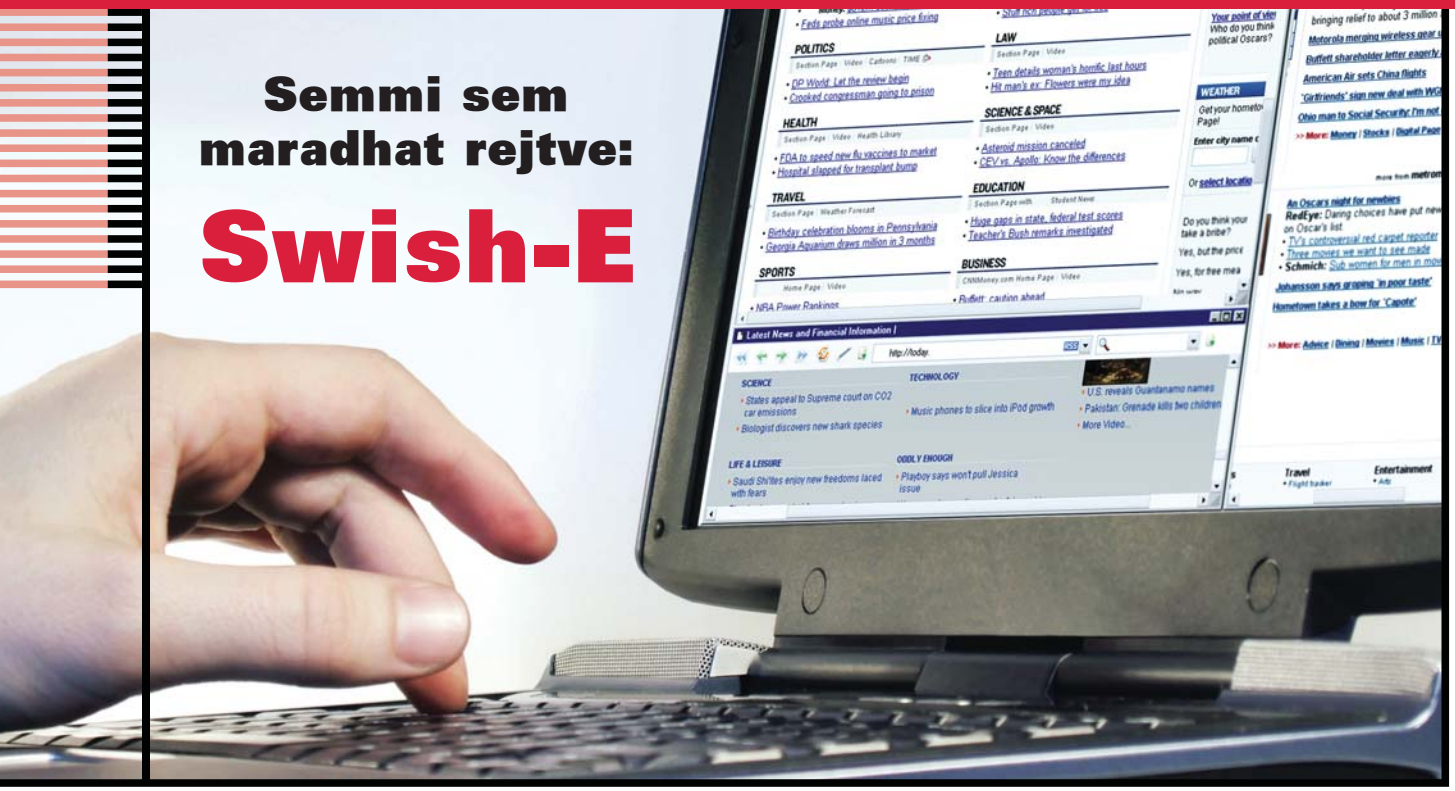
Zárszó

A szabad programok között a ***Celestia*** ugyanúgy tölti be kategóriájának „csúcsalkalmazás” szerepét, mint ahogyan a ***GIMP*** grafikai téren dominál. Az igazsághoz tartozik, hogy e röviden bemutatott program nem rendelkezik mérhető konkurenciával, azt azonban hozzá kell még tennem, hogy komolyan „fel kellene kötnie a gatyáját” annak, aki ennél használhatóbban és pontosabban szeretné a végtelen űrt modellezni. A ***Celestia*** platformtól független alkalmazás, így (többek között) ***Win32*** felületre is készítettek belőle binárist. Aki telepítés nélkül szeretné mozgás közben látni ezt a „világot”, annak készítettem egy rövid videót a program demó üzemmódjáról amely a ➲ ***http://kovi.uw.hu/lvilag2006/*** címen található.
Tartalmas kikapcsolódást kívánok mindenkinek!

Kovács Zsolt (kovi@linuxforum.hu)

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

Semmi sem maradhat rejtve: Swish-E



A swish-e egy egyszerűen használható, rugalmas alkalmazás, amellyel web oldalakat és egyéb fájlokat indexelhetünk. Nem csak szöveges állományokat, de elektronikus leveleket, PDF-, HTML-, XML-, doc-, ppt- és xls fájlokat is képes indexelni, sőt bármit, ami XML/HTML/text formátumba konvertálható...

A *swish-e* lehetőségeinek teljes listája a ➲ *http://www.swish-e.org/docs/readme.html#key_features* weblapon találhatjuk meg. Kipróbálásához első lépésben telepítsük az alkalmazást. Töltsük le a legutolsó verziót a ➲ *http://swish-e.org/* webhelyről, majd adjuk ki az alábbi utasításokat:

```
tar zxvf latest.tar.gz
cd swish-e-2.4.3
./configure
make
su -c 'make install'
```

Ha web oldalakat is akarunk indexelni, telepítsük a *HTML-Tagset*, *HTML-Parser*, *Compress-Zlib*, *Crypt-SSLeay* (*https://* oldalak eléréséhez) és *libwww-perl* modulokat a CPAN-ról (http://www.cpan.org/). Ezek azért szükségesek, mert a *swish-e* a *swishspider Perl* alkalmazást használja web indexelésre.
Első példaként indexeljük a saját web oldalunkat! Először azonban készítsük el a *swish-e* konfigurációs állományát. A paraméterek teljes listája a ➲ *http://www.swish-e.org/docs/swish-config.html* címen található.

Adjuk ki a

```
swish-e -S http -c swish-e.conf
➥-i http://www.mydomain.hu/
```

1. Lista A swish-e konfigurációs állománya

```
# ezek a karakterek alkotják a szavakat, a magyar karaktereket is érdemes felsorolni

WordCharacters abcdefghijklmno
➥pqrstuvwxyz0123456789.-
BeginCharacters abcdefghijkl
➥mnopqrstuvwxyz0123456789
EndCharacters abcdefghijk
➥lmnopqrstuvwxyz0123456789

IgnoreFirstChar .-
IgnoreLastChar  .-

# mennyi információt írjon ki a terminálra indexelés közben
IndexReport 2

# kifejezések kereséséhez használja a swish-e BumpPositionCounter
➥Characters |.

# maximum 10 lépés mélységben keres
MaxDepth 10

# 2 http kérés között 0 másodpercet vár
Delay 0

# az aktuális  könyvtárba teszi a spider az ideiglenes állományokat
TmpDir .

# ebben a könyvtárban van a swishspider program
SpiderDirectory ./src
```

2. Lista Így indexel

```
Indexing Data Source: "HTTP-
➥Crawler"
Indexing
➥"http://www.mydomain.hu/"
retrieving
➥http://www.mydomain.hu/
(0)...
retrieving
➥http://www.mydomain.hu/
➥howitworks.html (1)...
retrieving
➥http://www.mydomain.hu/
➥install.html (1)...

.... itt még sok html oldal
következik ....

Removing very common words...
no words removed.
Writing main index...
Sorting words ...
Sorting 2,773 words
alphabetically
Writing header ...
Writing index entries ...
  Writing word text: Complete
  Writing word hash: Complete
  Writing word data: Complete
2,773 unique words indexed.
4 properties sorted.
20 files indexed.  97,767
➥total bytes.  13,055 total
➥words.
Elapsed time: 00:00:08 CPU
➥time: 00:00:00
Indexing done!
```

utasítást. Erre a *2. Listában* olvasható üzenetek jelennek meg, jelezvén, hogy az elemzés elkezdődött.
Ennek hatására az aktuális könyvtárban létrejött két fájl ***index.swish-e*** és ***index.swish-e.prop*** néven. A futtatás során (az ***IndexReport*** változótól függően) több dolgot is kiír, például az éppen indexelt címeket, hogy éppen az indexet írja, a szavakat rendezi, hogy hány egyedi szót talált, hány webcímet nézett végig, azok mérete mekkora, illetve hogy mindez meddig tartott.
Azokat a webhelyeket amelyekben szerepel a `training` szó, az alábbi parancs kiadásával kereshetjük meg:

```
swish-e -c swish-e.conf -w
➥"training"

# SWISH format: 2.4.3
# Search words: training
# Removed stopwords:
# Number of hits: 10
# Search time: 0.003 seconds
# Run time: 0.028 seconds
1000 http://www.mydomain.hu/
➥training.html "Training the
➥token database" 5002
410 http://www.mydomain.hu/
➥install.html "Installation"
➥5750
410 http://www.mydomain.hu/
➥config.html "clapf.conf"
17127
205 http://www.mydomain.hu/
➥questions.html "Questions"
➥6144
205 http://www.mydomain.hu/
➥"clapf" 3256
.
```

A hashmarkkal kezdődő sorok tartalmazzák a ***swish-e*** verzió számát, a keresett kifejezést, a találatok számát és a futás illetve keresés idejét. A találatok egyes sorai négy részből állnak: találati relevancia (minél nagyobb ez a szám, annál fontosabb, relevánsabb az adott dokumentum), a webhely, majd annak címe, végül a dokumentum mérete byte-ban. A keresés eredményét egy pont (`.`) zárja.
A ***swish-e*** csak szöveges állományok indexelésére képes. Segédprogramok (szűrők) segítségével azonban képes akár a ***.doc***, ***.pdf***, stb. állományokat szövegessé alakítani, majd indexelni. Adjuk hozzá az alábbi sort a konfigurációs állományhoz, majd indexeljük újra a web oldalunkat! Az újabb futtatáskor az indexelt webhelyek között a ***PDF*** állományok is meg fognak jelenni, illetve a találatok között is, ha megfelel a keresési feltételeknek:

```
FileFilter    .pdf
➥/usr/local/bin/pdftotext
➥"'%p' -"
```

Nem csak web oldalakat, de a számítógépünkön tárolt állományokat is indexelhetjük. Ehhez módosítsuk az *1. Listában* szereplő konfigurációs állományt úgy, hogy elhagyjuk a következő direktívákat: `MaxDepth`, `Delay` és `SpiderDirectory`, majd indexeljünk!

```
swish-e -c swish-e2.conf -S fs
➥-i swish-e-2.4.3/src/test
```

Keressük meg azokat az állományokat, amelyekben szerepel a `This is just` kifejezés:

```
swish-e -c swish-e2.conf -w
➥'This is just'

1000 /home/sj/temp/swish-e-
➥2.4.3/tests/test_xml.html "If
➥you are seeing this, ..." 159
778 /home/sj/temp/
➥swish-e-2.4.3/tests/test.xml
➥"test.xml" 126
565 /home/sj/temp/swish-e-
➥2.4.3/tests/test.txt
➥"test.txt" 55
```

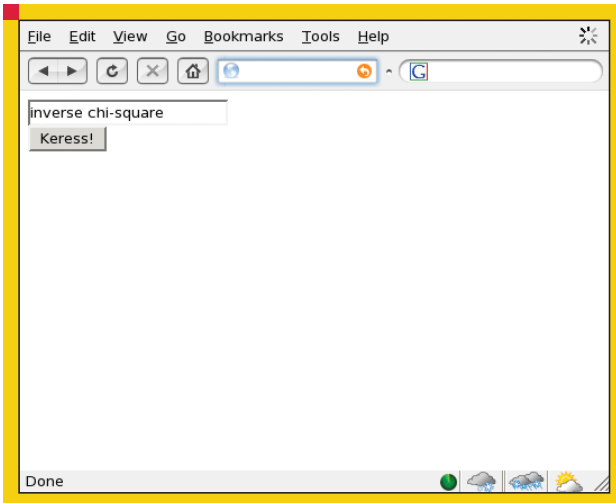
Ez a módszer nem jó, mert az első kettő találat nem tartalmazza a szóban forgó kifejezést. A kereső logikai VAGY kapcsolatba hozta a „This”, „is” és „just” szavakat. Próbáljuk újra:

```
swish-e -c swish-e2.conf
➥-w '"This is just"'

1000 /home/sj/temp/swish-e-
➥2.4.3/tests/test.txt
➥"test.txt" 55
```

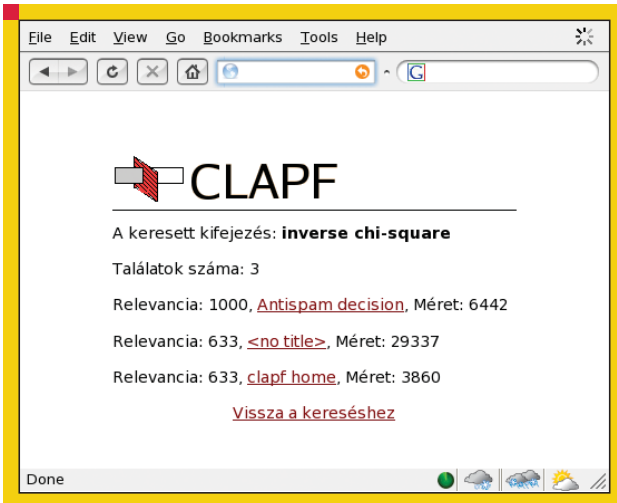
Ez már jó. A különbség csak annyi, hogy a `This is just` kifejezés még külön kettős idézőjelek közé lett téve, és így a ***swish-e*** már mint kifejezést használta.
Ha az Olvasónak van egy web oldala, amihez keresőt szeretne, akkor mindössze annyit kell tennie, hogy ír egy ***CGI*** programot, amely csövön (pipe) keresztül lefuttatja például az iménti parancsot, majd annak kimenetét valamilyen kellemes formába öntve visszaadja a böngészőnek. Erre csak abban az esetben biztatom az Olvasót, ha rendkívül körültekintően írja meg a ***CGI*** input szűrését, hogy ne lehessen érvénytelen adattal a gépünk biztonságát veszélyeztetni.
Sokkal biztonságosabb, ha a ***swish-e*** biztosította ***Perl*** vagy ***C API***-t használjuk, amelyek segítségével ***Perl*** vagy ***C*** nyelven fejleszthetünk olyan alkalmazást, amely képes a ***swish-e*** rendszert használni, majd a tőle kapott eredményt megjeleníteni. Nézzünk meg egy egyszerű programot (az ***src/libtest.c*** alapján) a ***C API*** használatára!

3. Lista Egy a swish-e-t használó program

```
/*
 * fordítás: gcc -O2 -Wall -o kereso kereso.c
 * -lswish-e
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <swish-e.h>

static void display_results(SW_HANDLE
➥swish_handle, SW_RESULTS results){
   SW_RESULT result;

   while((result = SwishNextResult(results))){

      printf("Rank: %ld, Title: %s, Path: %s,
      ➥Size: %ld\n",
         SwishResultPropertyULong(result,
         ➥"swishrank"),
         SwishResultPropertyStr(result,
         ➥"swishtitle"),
         SwishResultPropertyStr(result,
         ➥"swishdocpath"),
         SwishResultPropertyULong(result,
         ➥"swishdocsize"));
   }
}

int main(int argc, char **argv){
   SW_HANDLE swish_handle=NULL;
   SW_RESULTS res=NULL;

   swish_handle = SwishInit("index.swish-e");
   if(SwishError(swish_handle))
      SwishAbortLastError(swish_handle);

   if(argc < 2){
      printf("hasznalat: kereso kulcsszo\n");
      exit(0);
   }

   res = SwishQuery( swish_handle, argv[1]);
   if(SwishError(swish_handle))
   ➥SwishAbortLastError(swish_handle);

   printf("Talalatok szama: %d\n",
   ➥SwishHits(res));

   if(res)
      display_results(swish_handle, res);

   Free_Results_Object(res);
   SwishClose(swish_handle);

   return 0;
}
```



1. ábra Ilyen egyszerű űrlappal működik



2. ábra A keresés eredménye a böngészőben

Ez a program megkeresi az argumentumként megadott szót vagy kifejezést az *index.swish-e* állományban, kiírja a találatok számát, illetve az egyes találatokhoz tartozó relevanciát, a dokumentum címét, webhelyét és méretét.

Tegyük fel, hogy az olvasó egy web kiszolgálón akar keresés funkciót készíteni. Az előbbi kód kiegészítésével gyerekjáték az egész, mindössze arra van szükség, hogy átadjuk a keresett kifejezést egy *HTML* űrlap segítségével, dekódoljuk a *CGI* adatot, majd valami dizájnolt körítéssel kiírjuk a keresés eredményét. Ha az olvasó meg akarja spórolni ezt az ujjgyakorlatot, akkor használja a kibővített programomat, amely egy tetszőleges *HTML* sablon állományba illeszti be az eredményt, ill. sok találat esetén lapozni

4. Lista A swish-e „protokollja”

```
Path-Name: keddi_tv_musor
Content-Length: 18
Last-Mtime: 1143554804
Document-Type: HTML*

Ez itt az adat...
```

tud közöttük. A program a (***http://dev.acts.hu/swish_e_kereso-0.1.tar.gz***) címről tölthető le. Az ***1.*** és *2. ábrán* az említett program látható működés közben.
Web oldalak indexelése során a ***swish-e*** nem veszi figyelembe az idegen oldalakra mutató hivatkozásokat, szigorúan csak a megadott web helyen belül marad. Egyszer azonban úgy kellett egy népszerű oldalt indexelnem, hogy bizonyos megadott idegen hivatkozásokat is követnem kellett. Erre azonban nem képes a ***swishspider***, a ***swish-e*** gyári modulja. Mi sem mutatja azonban jobban a ***swish-e*** rugalmasságát, hogy magunk is készíthetünk saját ***spider*** alkalmazást. Így kiindulásként vettem a ***swishspider Perl*** programot, és ez alapján megírtam a saját pókomat, ami úgy indexel egy web oldalt, ahogy én fütyülök. A ***swish-e*** készítői szerint megfelelő pókkal bármit indexelhetünk, nem csak a fájlrendszert vagy a honlapunkat, de akár egy ***MySQL*** adatbázist, a ***Thunderbird*** postaládánkat, az ***MP3*** gyűjteményünket éppúgy, mint a jövő heti tv-műsort. Ehhez mindössze egy olyan program szükséges, amely – az előző példákat tekintve – lekérdezi az adatbázist, feldolgozza postafiókunkat vagy megszerzi valahonnan a tv-műsort, és azt átadja a ***swish-e*** programnak.
Ha elkészültünk saját pókunkkal (`szuperpok.pl`), futtassuk a ***swish-e***-t az alábbi módon:

```
swish-e -c 3.conf -S prog -i
➥szuperpok.pl
```

A külső program paraméterezése:

```
# ezzel a direktívával lehet
paramétereket megadni a külső
spider programnak, pl.
SwishProgParameters
mysql://localhost/
```

A külső segédprogram a ***HTTP*** protokollhoz hasonló módon kommunikál a ***swish-e*** alkalmazással, az alábbi *4. Lista* önmagáért beszél.
A ***swish-e*** egy rendkívül sokoldalú indexelő alkalmazás, amely bármit képes feldolgozni, amit (akár különféle segédprogramokkal is) szöveges formában át lehet neki adni. A cikkben leírt példák mind olyan feladatok nyomán keletkeztek, amelyeket valahogyan meg kellett oldanom. A ***swish-e***-nél alkalmasabb eszközt aligha találhattam volna.

Sütő János
(jsuto@freemail.hu)

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

GnuPG tippek trükkök

Kevesen tudják, de a GnuPG a levelek titkosításánál és visszafejtésénél valójában sokkal többre is képes...

Különösen a kezdő felhasználók gyakran érzik úgy, hogy a kriptográfia határozottan érdekes és hasznos téma ugyan, de egyben riasztóan összetett is. Eleve számos különböző szoftvercsomag létezik, aztán ott vannak a jelszavak, a kulcsok, a kulcskarikák, a tanúsítványok és digitális ujjlenyomatok. Az egész egy nagy zűrzavar. Azt már kevesebben sejtik, hogy nincs szükségünk az összes fura nevű dologra ahhoz, hogy használni tudjuk a titkosítási lehetőségeket. A *GnuPG*-vel gyakorlatilag minden különösebb előismeret nélkül is végezhetünk titkosítást, sőt könnyen lehet, hogy még a telepítésével se kell fáradnunk, mert a legtöbb terjesztésnek alapértelmezésként is része.

A GnuPG és az OpenPGP

A *GnuPG* az *OpenPGP* nyílt forrású (*GNU Project*) megvalósítása, amit *GNU Privacy Guard* néven is ismernek. A *GnuPG* egy meglehetősen kifinomult, nyilvános kulcsokon alapuló titkosítási rendszer, amelynek több mint 70 parancssori kapcsolója van, sőt rendelkezik egy belső parancssorral és egy menüvezérelt környezettel is. Számos különböző operációs rendszerre lefordítható és eleve számos rendszerhez létezik bináris csomagja is. Ezek letöltéséhez látogassunk el a *GnuPG* hivatalos weblapjára (lásd az elektronikus forrásokat). Akár a többi *GNU* szoftvert, ezt a csomagot is szabadon használhatjuk, hiszen ugyanúgy a *GNU General Public License* vonatkozik rá.
Az *RFC 2440*-ben rögzített *OpenPGP* szabvány *Phil Zimmermann* 1991-ben közzétett *Pretty Good Privacy* rendszere alapján készült. Ennek a leírásnak számos más, kereskedelmi termék titkosítási rendszer is megfelel, vagyis nyugodtan nevezhetjük általánosan elfogadottnak. Ami azt illeti, a jelenleg használatban levő titkosítási rendszerek között az *OpenPGP* rendszerek a leggyakoribbak.

Vágjunk bele!

Először ismerkedjünk meg a *GnuPG* néhány olyan szolgáltatásával, amelyekhez nincs szükség jelszóra. Ez után majd kitalálunk egy kiváló jelszót, és titkosítunk is vele valamit. Elöljáróban érdemes megjegyezni, hogy bár magát a rendszert *GnuPG*-nek hívják, az elindításához szükséges parancs a `gpg`.
A következő paranccsal győződjünk meg róla, hogy a *GnuPG* telepítve van rendszerünkön, illetve hogy végrehajtható állományának helyes szerepel az alapértelmezett útvonalak között:

```
gpg --version
```

Valami ilyesminek kell megjelennie a képernyőn:

```
gpg (GnuPG) 1.4.1
Copyright (C) 2005 Free
➥Software Foundation, Inc.
This program comes with
➥ABSOLUTELY NO WARRANTY.
This is free software, and you
➥are welcome to
redistribute it under certain
➥conditions.
See the file COPYING for
➥details.
Home: ~/.gnupg
Supported algorithms:
Pubkey: RSA, RSA-E, RSA-S,
➥ELG-E, DSA
Cipher: 3DES, CAST5, BLOWFISH,
➥AES, AES192, AES256, TWOFISH
Hash: MD5, SHA1, RIPEMD160,
➥SHA256, SHA384, SHA512
Compression: Uncompressed, ZIP,
➥ZLIB, BZIP2
```

A változatszám, a dátum és egyes más részletek természetesen eltérhetnek, de a kimenet nagyjából ilyen. Az ebben a cikkben bemutatott példáknak elvileg működniük kell a *GnuPG* legfrissebb, és minden jövőben megjelenő változatával is. Nos, akkor adjuk ki a következő parancsot:

```
gpg /dev/null
```

Erre a következő kimenetet kapjuk:

```
gpg: /home/you/.gnupg:
➥directory created
gpg: new configuration file
➥`/home/you/.gnupg/gpg.conf'
➥created
gpg: WARNING: options in
➥`/home/you/.gnupg/gpg.conf'
➥are not yet active during
➥this run
gpg: keyring `/home/you/.gnupg/
➥secring.gpg' created
gpg: keyring `/home/you/.gnupg/
➥pubring.gpg' created
gpg: processing message failed:
➥eof
```

A figyelmeztetés és ez a rengeteg tájékoztató adat teljesen normális, ha a *GnuPG*-t először futtatjuk. Ha valakinek mégsem ez jelenne meg a képernyőjén, annak sem kell semmitől tartania. Egyszerűen csak arról van szó, hogy a jelek szerint korábban már futtatta ezt a parancsot, és az létrehozta a *.gnupg* könyvtárat.

Bináris fájlok kódolása

A legtöbb e-mail program támogatja a csatolt fájlok küldését, bár a csak parancssorból működtethetők, mint például a `/bin/mail` történetesen nem. Ez néha nem is különösebben nagy baj, mert lehetnek olyan esetek, amikor célszerű minden adatot a levél törzsébe tenni. Csakhogy a leveleket továbbító hálózat nem képes megbirkózni bináris fájlokkal, így azokat előbb *ASCII* adatokká kell átalakítani. (Ha nem így teszünk, garantáltan hibás lesz az átvitel.) Talán sokan próbálták már a `uuencode` programot, és sokan szembesültek az összetettségével, vagy azzal, hogy néha egyszerűen nem működik. Ráadásul nincs is minden rendszerben parancssorból működtethető *MIME* kódoló. Ilyenkor jöhet jól a *GnuPG*, amelynek – kissé talán meglepő módon – van ilyen szolgáltatása. Ráadásul ez a funkció működését tekintve nagyon hasonló a *MIME* kódolókhoz, de a használata teljesen egyszerű és problémamentes. Ha *ASCII* adatokká akarunk alakítani egy fájlt, gépeljük a következőt:

```
$ gpg --enarmor < filename.bin
➥> filename.txt
```

A kódolt tartalmat a következő paranccsal lehet kibontani:

```
$ gpg --dearmor < filename.txt
➥> filename.bin
```

Figyelem! Bár a *GnuPG* alapvetően egy titkosításra szolgáló program, az *OpenPGP*-vel előállított *ASCII* fájlok semmiféle biztonsági védelemmel nem rendelkeznek. Ezen a helyzeten persze könnyen változtathatunk, amint az hamarosan ki is fog derülni.

Ellenőrzőösszegek hatékonyabban

Mit tegyünk, ha azt gyanítjuk, hogy egy épp most kapott vagy letöltött bináris fájl hibás. Ilyenkor általában a `sum` vagy `cksum` programokat szokás használni az átvitel előtt és után is. A kimenetek egyszerű összehasonlításával viszonylag nagy biztonsággal eldönthető, hogy sikeres volt-e az átvitel. Ugyanakkor sajnos ezeknek a programoknak három különböző, egymással nem kompatibilis változata létezik, sőt az is megeshet, hogy ugyanaz a változat különböző összeget számol ki ugyanabból a bemenetből különböző gépeken futtatva. Ezt utóbbi jelenséget egyébként az eltérő bájtsorrend (endianness) szokta okozni. És ami a legrosszabba az egészben az az, hogy bizonyos eseteben a fenti két program nem is képes kimutatni a hibát. A `sum` és a `cksum` programok kimenete egyaránt mindössze 32 bites, ami egyszerűen túl kevés a megbízhatósághoz. Simán előfordulhat, hogy bár egyezik a változatszám és nincs eltérés a két végpontok működő architektúrák között sem, mégis különböző bemenetekre ugyanazt a kimenetet kapjuk. Az *SSH v1* rendszer elleni „népszerű” *CRC-32* kompenzációs támadást éppen ez a probléma teszi lehetővé.
Minderre az első lehetséges megoldás az `md5sum` parancs használata, amelynek azonban szintén megvannak a maga problémái. A legbosszantóbb talán az, hogy a különböző változatok mind egy kicsit máshogy formázzák a kimenetet, máshogy adják meg a fájl nevét, vagy más módon jelenítik meg a hexadecimális értékeket. Ezek sajnos nem könnyítik meg az automatikus hibaszűrést, hiszen a `diff` parancs ezektől az eltérésektől akkor sem tud simán lefutni, ha amúgy nincs semmi gond. A dolgot csak tetézi, hogy az `md5sum` által használt *MD5* hasítóalgoritmusnak is vannak ismert sebezhetőségi pontjai. És akkor még nem is említettük azt a triviális lehetőséget, hogy a rendszergazda egyszerűen elfelejtette telepíteni a programot.
A *GnuPG*-vel az összes fent említett probléma megoldható, hiszen ez operációs rendszertől és változatszámtól függetlenül mindig ugyanazt a kimenetet adja ugyanarra a bemenetre. A *GnuPG* ráadásul támogatja az újabb és ezért biztonságosabb algoritmusokat is:

```
$ gpg --print-md sha1 filename
filename: E83A 42B9 BC84 31A6
➥6450  99BE 50B6 341A 35D3
➥DCEB
```

Megadhatjuk egyszerre több fájl nevét is:

```
$ gpg --print-md sha1 *.txt
test.txt: E0D6 3F44 4253 CED5
➥9205  4047 4AA6 4E0F FD0F
➥130D
test2.txt: 32AC 34F9 B7AF 1972
➥C015  E5EE 456E 89BD CC3C
➥7246
```

Ha valamiért mégis az *MD5* algoritmust szeretnénk használni, az se gond, mert a program ezt is ismeri:

```
$ gpg --print-md md5 filename
filename: 26 E9 85 5F 8A D6 A5
➥90  6F EA 12 12 83 C7 29 C4
```

A *GnuPG* újabb változatai támogatják az olyan újabb, kiemelten biztonságos hasítóalgoritmusokat is mint a *SHA-512*:

```
$ gpg --print-md sha512
➥filename
filename: FC37410D 9336DD60
➥22AEB6A2 A42E82F1 2EA3470D
➥4982E958 B35C14A0
          CF381CD2 3C4CBA35
          ➥BE5F11CB 05505ED2
          ➥DBF1C7A0 397EFF75
          ➥007FAEBB
          30B43B30 6514990D
```

Apropó a fenti kimeneteket és a `--print-md` példákat egész egyszerűen ellenőrizhetjük saját gépünkön is: Hozzunk létre egy egysoros fájlt, ami a `The Linux Journal` szöveget tartalmazza, és ezt adjuk meg bemenetként a *GnuPG*-nek.
A hash értékeknek pontosan meg kell egyezniük a bemutatottakkal.

Gyors és egyszerű titkosítás

Aki titkosítani akar egy fájlt, de nem tudja hogy fogjon hozzá, annak valószínűleg jól jön a következő, *GnuPG*-re alapozott gyorstalpaló:

```
$ gpg -c test.txt
Enter passphrase:
Repeat passphrase:
```

Kódolásnál a *GnuPG* kétszer bekér egy jelszót, pont ugyanúgy, mint amikor a bejelentkezési jelszavunkat átállítjuk. Az új, kódolt tartalmat egy ugyanolyan nevű fájl fogja tartalmazni, amelynek azonban immár *.gpg* kiterjesztése lesz. Az eredeti fájl érintetlen marad.
A `-c` kapcsoló a szokványos (conventional) titkosítást jelöli, amit más néven szimmetrikus titkosításnak is szoktak nevezni. A *GnuPG* alapértelmezett titkosítási módszere a nyilvános kulcsú architektúra használata, de mi egyelőre nem állítottunk elő egyetlen kulcspárt sem, így most kénytelenek leszünk az egyszerűbb módszert alkalmazni.

1. táblázat *A különböző szerkezetű és hosszúságú jelszavak erőssége (összehasonlítási alap a feltöréshez szükséges becsült idő)*

Típus	Hosszúság	Bitek száma	Bitek teljes száma	A feltöréshez szükséges idő
Egyetlen szó bármilyen nyelven	8 karakter	24	24	Másodpercek
Véletlenszerűen kiválasztott karaktersorozat (csak egyféle betű)	8 karakter	4.7	37	Percek
Véletlenszerűen kiválasztott karaktersorozat (csak egyféle betű)	16 karakter	4.7	75	Évtizedek
base64 [A-Za-z0-9+/=]	10 karakter	6	60	Hónapok
base64 [A-Za-z0-9+/=]	20 karakter	6	120	Törhetetlen?
Teljesen véletlenszerű nyomtatható karakterek	6 karakter	6.5	40	Percek
Teljesen véletlenszerű nyomtatható karakterek	8 karakter	6.5	52	Órák
Completely random printable	12 karakter	6.5	78	Évtizedek
Teljesen véletlenszerű nyomtatható karakterek	15 karakter	6.5	97	Évszázadok
Teljesen véletlenszerű nyomtatható karakterek	20 karakter	6.5	130	Törhetetlen?
Diceware jelszó	2 szó	12.9	26	Másodpercek
Diceware jelszó	4 szó	12.9	51	órák
Diceware jelszó	6 szó	12.9	78	Évtizedek
Diceware jelszó	8 szó	12.9	120	Törhetetlen?

Ez a titkosítási módszer egyébként akkor a leghasznosabb, ha csak mi magunk akarjuk visszafejteni a kérdéses tartalmat, de nem bízunk abban, hogy az a hely, ahol azt tároljuk, biztonságos. A könnyen elveszíthető vagy ellopható tárolóeszközök tartalmát például célszerű szimmetrikus kódolással titkosítani. Szintén hasznos lehet ez a védelem a telephelyen kívül tartott biztonsági másolatok védelmére.
A kódolt fájl visszafejtéséhez adjuk ki a következő parancsot:

```
$ gpg filename.gpg
```

A *GnuPG* automatikus detektálja, hogy szimmetrikus kódolással titkosított tartalomról van szó, és magától rákérdez a jelszóra. A visszafejtett adatokat egy ugyanolyan nevű fájlba írja de levágja a *.gpg* kiterjesztést. Akárcsak a kódolásnál, az eredeti fájl most is érintetlen marad. Ha a kimenetet egy eltérő nevű fájlba, vagy más helyre szeretnénk irányítani, használjuk a szabványos átirányítást ugyanúgy, mint a `--dearmor` kapcsoló esetében.

Fontos megjegyezni, hogy ilyenkor mind a bemenetet, mind a kimenetet átirányítással kell megadnunk, ellenkező esetben *GnuPG* összezavarodik:

```
$ gpg < filename.gpg >
➥filename.txt
```

Ha azt akarjuk, hogy a titkosított anyaghoz más is hozzáférjen, el kell árulnunk neki a kódolás során használt jelszót anélkül, hogy az kiszivárogna. Ennek a legegyszerűbb és legbiztonságosabb módja természetesen a személyes átadás. Erre most nyugodtan mondhatja valaki azt is, hogy ha már úgyis találkozunk az illetővel, akkor ezzel az erővel átadhatjuk neki magát a tartalmat is mindenféle titkosítás nélkül: Ne felejtsük el azonban, hogy ugyanazt a jelszót több alkalommal is használhatjuk, tehát a dolog nem föltétlen értelmetlen. Ugyanakkor a titkosításhoz használt jelszavakat időnként ugyanúgy le kell cserélni, mint a bejelentkezési kódot. Ezen kívül alapszabály, hogy soha nem használjuk ugyanazt a jelszót különböző emberekkel való kapcsolattartásra, hacsak nem akarjuk, hogy közülük bárki bármilyen általunk küldött tartalomhoz hozzáférhessen.
Van itt még egy talán nem lényegtelen megjegyzés. A következő figyelmeztető üzenet megjelenése teljesen normális, ha a *GnuPG*-t jelszavas (szimmetrikus) titkosításra használjuk:

```
gpg: WARNING: message was not
➥integrity protected
```

Ha nem akarjuk többé látni, használjunk nyilvános kulcsú titkosítást.

Jelszavak

A jelszó olyan titok, amely segít más dolgokat titokban tartani. Ebből következőleg a *GnuPG* teljes működési folyamatának egyik leglényegesebb az alkalmazott jelszó. Sajnos ez egyben azt is jelenti, hogy a rendszer legsebezhetőbb pontja is maga a jelszó. Ennek pedig egyszerűen az az oka, hogy igazán jó jelszavakat egyrészt nehéz előállítani, másrészt ha egy jelszó valóban jó, azt nehéz fejben tartani. Erősen javasolt a *Diceware* használata,

2. táblázat *A GnuPG e cikkben említett parancsainak rövid összefoglalása*

A kapcsoló rövid formája	Hosszú forma	Leírás
	`--version`	A változatszám és a támogatott algoritmusok kiíratása
	`--help`	Súgó
`-a`	`--armor`	ASCII kódolás bekapcsolása titkosítás közben
	`--enarmor`	Bináris bemenet ASCII kimenetté való átkódolása
	`--dearmor`	ASCII bemenet bináris kimenetté való visszakódolása
	`--print-md HASH`	Kivonat készítése az üzenetből a megadott hash alapján
`-c`	`--symmetric`	Hagyományos, szimmetrikus kulcson alapuló titkosítás jelszóval
`-o`	`--output`	A kimeneti fájl nevének megadása. A szabványos kimenetet - jelöli.

de ha ez valakinek nem tetszik, érdemes megnézni az elektronikus források között említett ***Wikipedia*** cikket, vagy rákeresni a interneten erős jelszavakat előállító weboldalakra. Függetlenül attól, hogy melyik módszer használata mellett döntünk, az alapvető irányelv azonos: a hosszabb jelszó jobb (lásd az *1. Táblázatot*). Látható, hogy az *1. Táblázat* adatai több nagyságrendet fognak át. Ennek alapvetően az az oka, hogy a törhetőség kérdésének két, nagyjából azonos jelentőségű összetevője van, amelyekkel szabadon lehet játszani: az idő és a pénz. A számítási teljesítmény – amint az közismert – egyre olcsóbb, így a titkosítások feltöréséhez szükséges idő egyre rövidül. Ami a költségeket illeti, bizonyos esetekben a dolog akár ingyen is megoldható, a felső határ azonban a „csillagos ég”. Mindettől függetlenül általánosságban kijelenthetjük, hogy ha valaki elfelejti a *GnuPG* jelszavát, az azzal kódolt adatokról feltehetőleg örökre lemondhat. A *GnuPG*-hez sem ismert hátsó kapu, sem a jelszó visszanyerésére szolgáló egyszerű módszer nem létezik. Ha valaki mégis belevág, a visszafejtéshez szükséges idő attól függ, mennyire jó jelszót választottunk. Egy igazán jó 20 karakterből álló jelszó feltörése úgy néhány milliárd évig tart, nemcsak a jelenleg elérhető gépeken, de valószínűleg a jövőben megjelenőkön is.

Jelszó előállítása

Van egy egyszerű trükk arra, hogy magával a *GnuPG*-vel állítsunk elő biztonságos jelszót. Ez persze nem lesz egy könnyen fejben tartható, vagy könnyen begépelhető darab, viszont garantáltan nagyon biztonságos lesz. Először egy 16 véletlenszerű bájtból álló sorozatot állítunk elő, majd ezt – ismét a *GnuPG*-t használva – base64 kódolásnak vetjük alá. Végül a `sed` segítségével levágjuk a kimenet fejlécét. Az így keletkezett karaktersorozatot nyugodtan használhatjuk jelszó gyanánt:

```
gpg --gen-random 1 16 | gpg
➥--enarmor | sed -n 5p
```

Kódolt tar fájlok előállítása

A *tar* archivumok tömörítésére a szokásos *gzip* helyett a *GnuPG*-t is használhatjuk. A végeredmény ebben az esetben egy körülbelül ugyanakkora fájl lesz, de a tartalom immár titkos. Apropó senkinek nem ajánlom, hogy titkosított fájlok tömörítésével töltse az idejét, az ilyen adatok ugyanis rendszerint tömöríthetetlenek. Ennek – erősen leegyszerűsítve – az az oka, hogy a tömörítés és a titkosítás matematikailag egymáshoz meglehetősen közel álló dolgok. Éppen ezért a legtöbb titkosító rendszer – és ez alól a *GnuPG* sem kivétel – a titkosítás előtt automatikusan tömöríti a kódolandó adatokat. Ráadásul ez valamelyest a biztonságot is növeli.

Archívum titkosításakor a rendszer ugyanúgy be fog kérni egy jelszót, mint amikor egy közönséges fájlt titkosítunk:

```
tar -cf - these files here |
➥gpg -c > these-files-here.tgp
```

A visszafejtéskor természetesen ugyanezt a jelszót kell majd megadnunk:

```
gpg < these-files-here.tgp  |
➥tar -xvf -
```

A GnuPG automatizálása

Ha a *GnuPG*-t egy szkripten belül akarjuk használni, és nem szeretnénk, ha minden egyes futtatásnál bekérné a jelszót, akkor rögzíthetjük azt egy szövegfájlban is (példánkban `passphrase.txt`), aminek a nevét adjuk meg paraméterként:

```
$ cat passphrase.txt | gpg
➥--passphrase-fd 0 -c <
➥filename.txt > filename.gpg
```

A visszafejtés majdnem ugyanígy megy, csak a `-c` kapcsolót kell használnunk, és értelemszerűen át kell írni a fájlneveket:

```
$ cat passphrase.txt | gpg
➥--passphrase-fd 0 <
➥filename.gpg > filename.txt
```

Ha a titkosított fájl e-mailben akarjuk elküldeni valakinek, esetleg egy a telephelyen kívüli címre, akkor érdemes a `-a` kapcsolót is használni, amivel bekapcsoljuk az ***ASCII*** kódolást. A végeredmény pontosan ugyanaz lesz, mint mikor korábban az `--enarmor` kapcsolót használtuk, de a tartalom most titkosítva is lesz. Kellemes mellékhatásként a fájlméret is kisebb lesz mint a `uuencode`-ot vagy ***MIME*** kódolást használva, hiszen a *GnuPG* – mint már említettem – automatikusan tömörít, mielőtt elvégezné a titkosítást. A dolog betetőzéseként a levélküldést magából a szkriptből is megoldhatjuk. Figyeljük meg, hogy ilyenkor szükség van a `-o` kapcsolóra, ami a *GnuPG*-t a szabványos kimenet használatára kényszeríti:

```
$ cat passphrase.txt | gpg
➥--passphrase-fd 0 -ac -o
➥- filename.txt | mail
➥user@example.com
```

Apropó a jelszónak egy szövegfájlban való elhelyezése meglehetősen veszélyes dolog. Bárki, aki képes ezt a fájlt megszerezni, vissza tudja fejteni az összes, ezzel titkosított anyagunkat. Mi több, az illető akár új, titkosított fájlokat is létrehozhat, amelyek megkülönböztethetetlenek lesznek a sajátjainktól. Mielőtt tehát alkalmazzuk az itt bemutatott módszert, győződjünk meg róla, hogy a jelszót tartalmazó fájl, illetve az egész gép kellően biztonságos helyen van.
A dolgok automatizálása eleve azzal jár, hogy a folyamatból kizárjuk az emberi tényezőt, így az imént említett biztonsági problémát igen nehéz kiküszöbölni. Persze a *GnuPG*-nek még erre is van megoldása, hiszen használhatunk nyilvános kulcsú titkosítást.

GnuPG gondok

Megeshet, hogy egyszer csak kapunk valakitől egy *OpenPGP*-vel titkosított fájlt, de nincs hozzá kulcskarika, így első közelítésben fogalmunk sincs, mit kellene vele kezdeni. Lehet, hogy aki küldte nekünk, úgy gondolta, hogy számunkra a dolog egyértelmű lesz, de tévedett, mert mégsem az. Aztán az sem kizárt, hogy a küldő se tudja, mivel mit is kellene tenni ahhoz hogy mi legyen.
Ha a fájlnak *.pgp* vagy *.gpg* a kiterjesztése, megpróbálhatjuk visszafejteni a *GnuPG*-vel. Belenézhetünk egy szövegszerkesztő segítségével is, és ellenőrizhetjük, hogy a tartalma hasonlít-e a következőre:

```
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.2.5
➥(GNU/Linux)
jA0EAwMCwg21r1fAW+5gyS0KR/bkeI8
➥qPwwQo/NOaFL2LMXEYZEV9E7PBLjj
➥Gm7Y
DGG4QnWD5HSNOvdaqXg=
=j5Jy
-----END PGP MESSAGE-----
```

Ha igen, akkor egy *ASCII* kódolású *PGP*-vel titkosított fájlunk van.
A fenti példában a titkosított tartalom ugyanaz, mint amit a `--print-md` kapcsoló bemutatásakor használtunk, és a jelszó is azonos az ott használttal. Abból is számos hasznos dolgot leszűrhetünk, ha egyszerűen csak lefuttatjuk a *GnuPG*-t egy ismeretlen bemeneti fájllal. Ha jelszót kér a program, akkor például biztos, hogy titkosított tartalomról van szó, tehát vagy megszerezzük, vagy kitaláljuk azt, különben semmit nem tudunk vele kezdeni:

```
gpg unknown_file
```

Ha a kapott anyag nem is *OpenPGP* fájl, csak úgy néz ki, a következő üzenetet kapjuk:

```
gpg: no valid OpenPGP data
➥found.
gpg: processing message failed:
➥eof
```

Az ettől eltérő üzenet arra utalhat, hogy a fájlt egy olyan nyilvános kulccsal titkosították, aminek nem rendelkezünk a titkos párjával. Aztán az is előfordulhat, hogy a fájl egyszerűen sérült. Az egyik leggyakoribb hiba az, amikor bináris tartalmat küldenek kódolás nélkül e-mailben, vagy *FTP*-n *ASCII* módban töltik le azt.
A *GnuPG*-nek egy speciális diagnosztikai funkciója is van, ami segít a hibák kiszűrésében. Egy *OpenPGP* üzenet belül csomagokra oszlik, amelyekről a `--list-packtets` kapcsolóval információt is kérhetünk:

```
gpg --list-packets
➥unknown_file.gpg
```

A szabványos információ mellett ez a kapcsoló kiíratja a titkosításhoz használt nyilvános kulcs teljes azonosítóját (már amennyiben így titkosították a fájlt), valamint a titkosító algoritmus nevét is. Előfordulhat, hogy az anyagot a *PGP 2.x* nyilvános változatával titkosították (ezt szokás hagyományos kulcsnak is hívni). Sajnos a *PGP 2.x*. Nem felel meg mindenben az *OpenPGP* szabványnak, így a *GnuPG* nem tudja visszakódolni az így titkosított üzeneteket. A *PGP* legtöbb megvalósítása – különösen az utóbbi években készítettek – megfelel az *OpenPGP* szabványnak, így ha ilyesmivel találkozunk, általában az is elegendő, ha megkérjük a küldő felet, hogy mentse a tartalmat *OpenPGP*-vel kompatibilis formában, és küldje el újra.
Az *OpenPGP* szabvány ezen kívül többféle titkosító algoritmust támogat, amelyek közül egyesek esetleg nincsenek megvalósítva a *PGP* bizonyos változataiban. Hogy saját rendszerünkkel milyen algoritmusok használhatók, azt a

```
gpg --version
```

paranccsal jeleníthetjük meg.
A csomagformátummal és az algoritmusok belső számozásával kapcsolatos részleteket megtaláljuk az *RFC 2440*-es dokumentumban.
A *GnuPG* legtöbb kapcsolójának csak hosszú formája létezik, de néhol használhatunk rövid, egy betűs alakot is. Ezzel akadhat némi probléma is, ugyanis az eredeti PGP program egybetűs parancsai nem mindig ugyanazt jelentik, mint a *GnuPG*-nél.
A `-v` például az egyik helyen a `--verbose`, míg a másikon a `--version` rövidítése.

Hogyan tovább?

A *GnuPG* általános funkciói több helyen is egészen jól össze vannak foglalva. A hozzáférhető leírások közül az egyik legjobb a *GnuPG MiniHOWTO*, amit *Brenni de Winter* írt, és megtalálható a *GnuPG* webhelyén. Ez és még számos más dokumentum is bemutatja, miként használhatjuk a *GnuPG* általános, nyilvános kulcsokon alapuló szolgáltatásit.
A *GnuPG* levelezési listája szintén kiváló információforrás lehet, sőt a *GnuPG* webhelyén a teljes archívumát is megtaláljuk. Ezen a listán amúgy *Werner Koch*, a *GnuPG* vezető fejlesztője is gyakran publikál érdekes dolgokat.

Linux Journal 2006. 143. szám

Tony Stieber
UNIX rendszerekre, kriptológiára és fizikai biztonságra szakosodott információbiztonsági szakértő. 1999 óta foglalkozik Linuxszal, a UNIX-szal pedig 1987-ben találkozott. Számítógépet már 1980 előtt is látott. Ezzel együtt nem igazán tudja, mit hoz majd a következő évtized.

KAPCSOLÓDÓ CÍMEK

A cikkhez tartozó elektronikus források a következő helyen találhatók:
➲ www.linuxjournal.com/article/8743

Ellenség a (tűz)falakon belül

A biztonsággal nem lehet elég sokat foglalkozni. Ez természetesen nem azt jelenti, hogy tűzfalunkat a nap 24 órájában kellene még tovább finomítani, mert tökéletes valószínűleg akkor sem lesz. Előbb-utóbb valakinek úgyis sikerül „fogást találni" rajta , viszont ha már megtörtént a baj, akkor célszerű lenne azt a lehető legrövidebb idő alatt felismerni.

Pontosan ezt a feladatot vállalja magára a ***Tripwire***, mely tulajdonképpen nem más, mint egy fájlváltozás figyelő alkalmazás. Egy átlagos ***Linux*** rendszerben több ezer, vagy akár több tízezer fájl is lehet, ezért nem várható el, hogy ezeket állandóan szemmel tartsuk. Általában pedig már csak akkor kezdünk el hibát keresni, ha valami nem működik megfelelően, hiszen kedvenc operációs rendszerünk stabilitása miatt szerencsére nem túl gyakran kell ezt megtennünk. Pedig az ártó szándékkal gépünket uralni vágyó betörők nagy részének nem az érdeke, hogy rögtön hatalmas károkat okozzanak nekünk.
Jól bevált gyakorlat, hogy egy sikeres betörés után a feltört rendszerben egy ún. „backdoor" (hátsó kapu, hátsó ajtó) programot helyeznek el, mely segítségével később már nem kell újra feltörni a rendszert, hanem a hátrahagyott „bejáraton" keresztül a betörő észrevétlenül, akár hónapokon keresztül is visszalátogathat hozzánk. Mindezt anélkül is megteheti, hogy ez a tudomásunkra jutna.
Akármilyen furcsa is a nálunk ki-be járkáló betörő sokkal kártékonyabb lehet, mintha a támadást rögtön felfedeztük volna. Arról nem is beszélve, hogy először lehet, hogy csak a tűzfalunk esik áldozatul, és csak idővel utána a többi számítógép.

A Tripwire bemutatása

Sok behatolásfigyelő alkalmazás közül most behatóbban a ***Tripwire*** használatával ismerkedünk meg, mely alapvetően a rendszer fájljaiban történt gyanús változásokat figyeli. A ***Tripwire*** az eltárolt szabályok és fájladatbázis alapján átfésüli a fájlrendszerünket, eltárolja és megjeleníti a detektált változásokat. A szabályok segítségével könnyen a testreszabhatjuk azt, hogy mely fájlokon milyen típusú ellenőrzést végezzen az alkalmazás. Ajánlatos a fájlokról az adatbázist még azelőtt elkészíteni, mielőtt az adott számítógépet a hálózatra kötnénk.
Ha végképp szeretnénk biztonságban tudni az adatbázist és az alkalmazást, akkor írjuk ki egy csak olvasható médiára, például egy ***CD***-re vagy pedig tároljuk egy megbízhatónak ítélt távoli számítógépen. Ellenkező esetben a támadónak lehetősége van módosítani a ***Tripwire*** adatbázist, illetve magát az alkalmazást is. Szerencsére ezt azért nem annyira egyszerű megtenni, mert a ***Tripwire*** saját jelszavas védelemmel rendelkezik, illetve az általa generált kimeneti fájlokat digitálisan

1. Lista A Tripwire beüzemelése

```
DIR=/etc/tripwire
SITE_KEY=$DIR/site.key
LOCAL_KEY=$DIR/$(HOSTNAME)-local.key

#központi és helyi kulcs generálása
twadmin -- generate-keys -- site-keyfile $SITE_KEY
twadmin -- generate-keys -- local-keyfile $LOCAL_KEY

#a konfigurációs fájl és szabálygyűjteményt aláírása
twadmin -- create-cfgfile -- cfgfile $DIR/tw.cfg -- site-
➥keyfile $SITE_KEY $DIR/twcfg.txt
twadmin -- create-polfile -- cfgfile $DIR/tw.cfg -- site-
➥keyfile $SITE_KEY $DIR/twpol.txt

#jogosultságok beállítása
cd $DIR
chown root:root $SITE_KEY $LOCAL_KEY tw.cfg tw.pol
chmod 640 $SITE_KEY $LOCAL_KEY tw.cfg tw.pol
```

2. Lista Példa a szabályok kialakítására

```
(
        rulename="Elso kritikus szabalyom",
        severity=100
)
{
        /etc/sudoers            ->$(SEC_CRIT) ;
        /etc/shadow             ->$(SEC_CRIT) ;
        /etc/passwd             ->$(SEC_CRIT) ;
}
```

aláírja. Ezért a fenti megoldások megvalósítását most nem részletezem. Periodikusan futtatva a ***Tripwire-t*** állandó képet kapunk arról, hogy rendszerünkben milyen fájlok változtak meg, természetesen minden nagyobb általunk generált fájl változás (például biztonsági frissítés) után célszerű a kezdeti adatbázist újra frissíteni.

A Tripwire telepítése

Ajánlatos a ***Tripwire*** honlapjáról beszereznünk a legfrissebb stabil verziót (ez a cikk írásának pillanatában a 2.4.0.1). Majd pedig ezt telepíteni forrásból, itt meg kell jegyeznem, hogy a ***Tripwire-nek*** van egy szabad és egy kereskedelmi verziója, ebben a cikkben csak a szabad forráskódú verziót ismertetem. A szokásos módon telepítsük az alkalmazást, majd pedig futtassuk le az ***install.sh*** szkriptet, amennyiben ez nem állna rendelkezésünkre, akkor a következő parancsok segítségével ugyanazt a hatást tudjuk elérni. Először létre kell hoznunk a központi és helyi kulcsot, majd a kulcsok segítségével alá kell írnunk a konfigurációs fájlt és a szabálygyűjteményt. A nem kódolt formátumú konfigurációs állományokat pedig távolítsuk el. Futtassuk le az *1. Listában* látható kódot.
Most már kiadhatjuk a

```
tripwire -init
```

parancsot, melynek hatására elkészül a ***Tripwire*** adatbázis. Ezt az alkalmazás a helyi kulccsal írja alá. Ne felejtsük el a ***twcfg.txt*** és a ***twpol.txt*** állományokat eltávolítani.

A

```
tripwire -check
```

paranccsal pedig bármikor ellenőrizhetjük a fájlok integritását.

A szabályok testreszabása

A ***tw.cfg*** fájl tartalmazza azokat a szabályokat amely alapján az alkalmazás eldönti, hogy az adott fájl a fájladatbázis részét képezze-e vagy sem. Az alapbeállításként használt konfiguráció általában jó, de természetesen sokkal jobb, ha testreszabjuk, már csak azért is, mert megítélésem szerint túl sok fájlt figyel egyszerre a program az alapbeállítással.
Mivel a ***twcfg.txt*** és a ***twpol.txt*** fájlokat előzőleg töröltük le, ezért vissza kell nyerni őket először ahhoz, hogy módosítani tudjuk tartalmukat. A

```
twadmin -- print-cfgfile >
➥/etc/tripwire/twcfg.txt
```

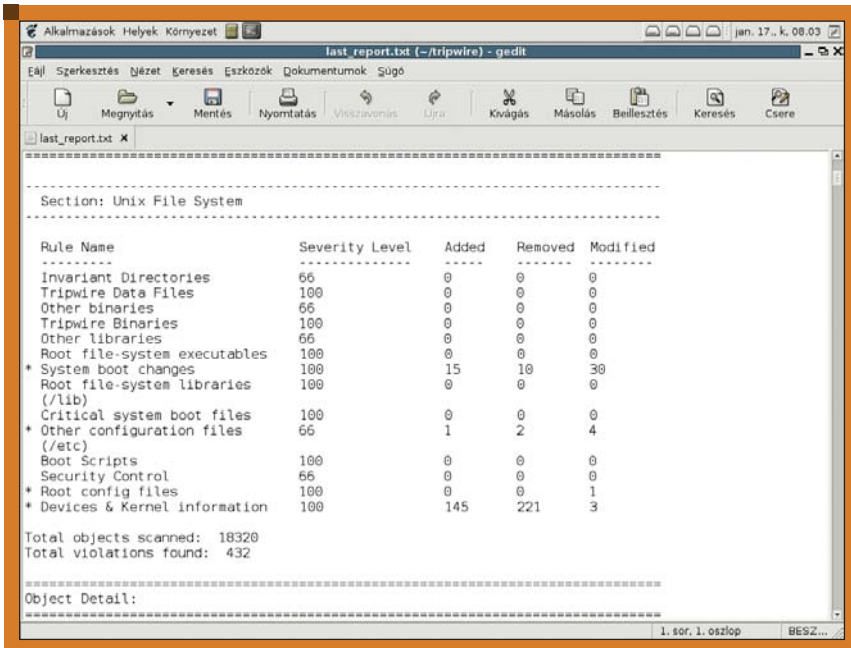
és

```
twadmin -- print-polfile >
➥/etc/tripwire/twpol.txt
```

parancsokkal tudjuk újra számunkra is olvasható formátumra hozni. Beletekintve a ***twcfg.txt*** fájlba láthatjuk, hogy a ***Tripwire*** a fájlokat különféle kategóriák szerint vizsgálja. Gyorsan fussunk végig az egyes kategóriákon.

- `SEC_CRIT`: Olyan fájlok, melyek a rendszer szempontjából kritikusak, és nem változhatnak meg, ilyen például maga a ***Tripwire.***
- `SEC_BIN`: Olyan bináris fájlok, melyeknek nem lenne szabad változni.
- `SEC_CONFIG`: Olyan konfigurációs fájlok, melyek ritkán változnak, de ugyanakkor gyakran használjuk őket.
- `SEC_LOG`: Ide tartoznak azok a fájlok, melyeknek mérete általában folyamatosan nő, de hozzáférési jogaik nem változhatnak.
- `SEC_INVARIANT`: ezek azok a könyvtáraink, melyek hozzáférési jogai nem változhatnak

Természetesen magunk is hozhatunk létre újabb kategóriákat. Az egyes kategóriákat még súlyosságuk szerint is



■ ***1. ábra*** A Tripwire jelentés megtekintése

besoroljuk az alábbi három már előre definiált súlyossági szinttel:

- `SIG_LOW`: Nem kritikus fájlok, melyek esetében általában kicsi a behatolás esélye.
- `SIG_MED`: Nem kritikus fájlok, melyek esetében számottevő a behatolás esélye.
- `SIG_HI`: Kritikus fájlok, melyek a behatolás elsődleges célpontjai lehetnek.

Természetesen újabb súlyossági szinteket is tudunk definiálni. A súlyossági szintek később megkönnyítik a kritikus fájlok gyakoribb ellenőrzését.
A *2. Lista* egy olyan szabályrendszert mutat be, melyhez hasonlóakat mi is könnyen létrehozhatunk.
Az egyes szabályokat a nevük alapján is módunkban áll leellenőriztetni, a fenti példát követve `tripwire -- check -- rule-name "Elso kritikus szabalyom"`. Amennyiben az összes egy bizonyos szintű vagy annál magasabb súlyosságú szabályt akarjuk leellenőrizni, akkor azt a

```
tripwire -- check -- severity
➥40
```

paranccsal tehetjük meg.
Mivel a szabályok bizonyos esetekben fedhetik egymást, ezért az egyes szabályokon belül lehetőségünk van arra, hogy megtiltsunk bizonyos fájlok vagy állományok vizsgálatát. Például az `"Elso kritikus szabalyom"` már leellenőriz 3 állományt a */etc* könyvtárban, ezért célszerű megkeresni azt a szabályt, amely átvizsgálja a teljes */etc* könyvtárat és úgy módosítani, hogy ezeket a fájlokat már később ne ellenőrizze le újra, tehát a kérdéses szabályhoz írjuk hozzá a következőt:

```
!/etc/sudoers
!/etc/passwd
!/etc/shadow
```

A változások ellenőrzése

A *Tripwire* jelentések alapértelmezésben a */var/lib/tripwire/report* könyvtárban találhatóak. A jelentés fájlneve tartalmazza a kiszolgáló nevét és a jelentés elkészítésének időpontját. Mivel az alkalmazás kódoltan és aláírva tárolja el a jelentést, ezért először számunkra is olvasható formátumra kell alakítani. Ezt a

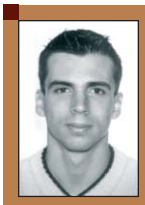
```
twprint -- print-report --
➥twrfile "kiszolgalo-legutolso
➥jelentes.twr"
```

paranccsal tehetjük, mivel a program az alapértelmezett kimenetre dolgozik, ezért inkább irányítsuk át egy fájlba, majd egy megfelelő editor program segítségével tekintsük meg a jelentést.

Az *1. ábrához* hasonló kimenetet fogunk a fájlban találni.
Sajnálatos módon a kimenet elég hosszú, de az elején egy táblázatban vannak összefoglalva a bekövetkezett változások. Természetesen a *Tripwire* beállítható úgy is, hogy minden egyes jelentés alkalmával e-mail formájában értesítsen minket.

Mindennapi használat

Érdekes kérdés ugyanakkor, hogy milyen gyakran futtassuk le az ellenőrzést, hiszen egy-egy terheltebb órában leterhelni a kiszolgálót nem túl szerencsés, ugyanakkor kívánatos volna minél gyakrabban, és nem csak naponta képet kapni a rendszerről.
Mivel a *Tripwire* lehetőséget biztosít arra, hogy ne csak az egész adatbázist, hanem annak bizonyos darabjait ellenőriztessük le egyszerre a rendszerrel. Így például azt a néhány tényleg kritikus fájlt (***index.html***, ***passwd***, ***sudoers***) akár 10 percenként is ellenőrizhetjük, míg a további fájlokat egy kevésbé terhelt napszakban naponta egyszer. Használjuk okosan a lehetőséget a szabály neve vagy az azonos súlyossággal megjelölt szabályok alapján való ellenőrzésre.
Természetesen 100%-os biztonságot még a *Tripwire* használatával sem érünk el, hiszen adódhatnak olyan helyzetek, amikor például frissítjük a rendszert, majd egy rosszindulatú behatoló elvégzi a módosításait bizonyos fájlokon. Utána mi elkészítjük az általunk jónak hitt fájladatbázist, a változásokat természetesen a saját beavatkozásunknak tudjuk be, ugyanakkor azt egy rosszindulatú személy követte el. Ekkor hiába végzünk állandóan ellenőrzést a hibát a *Tripwire* nem fogja jelezni, hiszen egy már eleve hibás fájladatbázissal történik az összehasonlítás.



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.

G4U – Ghost for UNIX

A sok egyforma géppel ellátott vállalatoknál elterjedt eszköz a Symantec cég Ghost nevű programja. Ezen program korábbi verziója a gépek merevlemezéről egy teljes másolatot képes készíteni, amelyet felhasználva később rendszerösszeomlás vagy hibás működés esetén hamar vissza tudjuk állítani az eredeti állapotot. Az újabb kiadások képesek csak a fájlrendszer szintjén végezni ezt a tevékenységet, illetve rögtön hálózatra vagy optikai tárra írni a másolatot. A legújabb kiadás ezt a tudást kiegészíti egy általános biztonsági mentés funkcióval is.

Szegény ember vízzel főz

A *UNIX* világban igazából mindig is megvolt az az eszköz, amelynek az üres helyét a *Windows* rendszerek esetén a *Ghost* betölti: ez a program a `dd`. A nevének eredete homályba vész, legnagyobb esélye a *Disk Dump* rövidítésének van, amely teljes mértékben fedi a program tudását. Szinte mindent tud, amire szükségünk van, képes egy egész merevlemezt kimenteni egy másik állományba, illetve akár egy-egy partíciót is, illetve a lemeznek egy meghatározott részletét is.
A `dd` azonban nem mindenható, sok hiányossága van az újabb *Ghost* verziókkal szemben. Az egyik ilyen hiányosság, hogy nem érti a fájlrendszert, s ezért egy alig megtöltött nagyobb merevlemezt teljes egészében lemásol, holott elég lenne csak azt a részét, ahol hasznos adat van. A `dd` további hiányosságaiért a *UNIX* rendszerekben megtalálható többi program bőségesen kárpótol minket. A tar nevű program remek eszköz arra, hogy egy nagy archív állományba tegyük tömörítve a menteni kívánt állományainkat, illetve akár az egész rendszert egyben le tudjuk menteni, majd visszaállítani. A `dump` és a `restore` páros pedig növekményes mentést képes készíteni az állományainkról.
A legnagyobb hiányossága a `dd` programnak az, hogy a működéséhez futó *UNIX* rendszerre van szükség.

Sok apró próbálkozás van arra, hogy a `dd` köré egy univerzális mentéseket elvégző keretet tegyenek, s az egyik legjobb ilyen próbálkozás a *NetBSD* rendszermagot használó *G4U*, amelyet – mint önálló operációs rendszert – flopira vagy *CD*/*DVD* lemezre írva tudunk indítani.

Honnan, mivel és hogyan

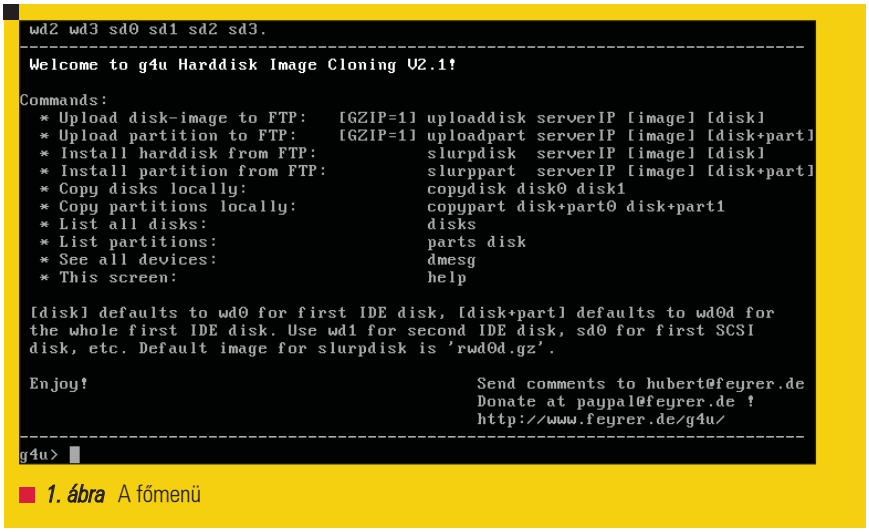
A *BootCD* készítéséhez egyszerűen le kell töltenünk a legújabb *ISO* állományt a ➲ *http://www.feyrer.de/g4u/g4u-2.1.iso* címről, amelyet fel kell írnunk egy üres lemezre. A teljes körű működéshez szükségünk lesz egy combosabb *FTP* tárterületre és egy *DHCP* szerverre is, ezeken kívül 100 vagy 1000 Mbites helyi hálózat sem hátrány.
A *G4U* alapvetően a `dd` programot használja a mentés elkészítésére, azonban ehhez egy könnyen kezelhető felületet is ad. A `dd` kimenetét kétfelé tudjuk irányítani: egy másik merevlemezre vagy egy *FTP* tárterületre. Az első esetben bitről-bitre tudunk teljes lemezt vagy partíciót másolni egyik lemezről a másik lemezre, illetve egyik partícióról a másik partícióra.
Ha *FTP* szervert szeretnénk használni, akkor az adatforgalom várható nagysága és a tárterületet szűkössége okán az átvitelt `gzip` programmal tömöríti a *G4U*. *DHCP* szerverre is csak ez utóbbi esetben van szükségünk.

Üzem közben

A *G4U* elindításához ellenőrizzük, hogy a kiválasztott számítógép a *CD* meghajtót fogja induláshoz először ellenőrizni, majd tegyük bele az elkészített lemezt és indítsuk el a gépet. A *BSD* világban teljesen megszokott módon indul el a rendszermag, a *Linux* felhasználóknak kissé szokatlan lehet az információk sorrendje és kiírási módja. A *G4U* a legújabb vagy az egzotikus hardvereket leszámítva szinte az összes számítógépben eligazodik a számára szükséges eszközök között, hiszen csak egyszerű szöveges módot és a hálózati kártyát kell használnia. A program sikeres elindulása után az *1. ábrán* látható képernyő fogad majd minket, amelyen minden fontos információ megtalálható.

Lemezről lemezre, partícióról partícióra

Iskolai körülmények között az egyik leggyorsabb klónozási módszer, amikor egy feltelepített és készre konfigurált számítógépeket közvetlenül lemezről-lemezre másoljuk, így akár 30-40 MBájt adatátvitelt is el tudunk érni másodpercenként, persze csak ilyen tudású géppel és új merevlemezekkel. Átlagos gépek esetén ez a sebesség lecsökkenhet 10-15 MBájtra is.
A módszernek egyetlen hátránya van, ugyanis szét kell szedni a gépet, amelyet esetleg garanciális zárjeggyel ellátott gépeknél nem tudunk megtenni.



1. ábra A főmenü

A másoláshoz a

```
copydisk wd0 wd1
```

parancsot kell kiadnunk, amelyben a `wd0` és a `wd1` *IDE* lemezek, s a

```
disks
```

parancs fogja kiírni a *G4U* által megtalált lemezek nevét és adatait:

```
wd0 at pciide0 channel 0 drive 0:
wd0: drive supports 16-sector pio
➥transfers, lba addressing
wd0: 6149 MB, 13328 cyl, 15 head,
➥63 sec, 512 bytes/sect x
➥12594960 sectors
wd0: 32-bit data port
wd0: drive supports PIO mode 4,
➥DMA mode 2, Ultra-DMA mode 2
wd0(pciide0:0:0): using PIO
➥mode 4, Ultra-DMA mode 2
➥(using DMA data transfers)
```

Mentés FTP szerverre

Nagyon kényelmes megoldás, hogy a *G4U* képes a mentett adatokat tömörítve feltölteni egy *FTP* szerverre. Ehhez szükséges egy olyan *FTP* szerver, amelyik akár 20-30 Gbyte-os fájlokat képes kezelni. A feltöltés adatátviteli sebességét két tényező befolyásolja: a kliens gép teljesítménye és a hálózat sávszélessége. A gép teljesítménye a tömörítés sebességét nagyban meghatározza, bár képesek vagyunk egy kilenc fokozatú skálán meghatározni a leggyorsabb működés és a legkisebb méret között dönteni, mindegyik fokozat processzorigényes. Egy korszerű P4-3G processzor is csak 5-6 Mbyte adatot képes tömöríteni másodpercenként, egy átlagos 2-3 éves számítógép pedig csak 2-3 Mbyte adatot tud összenyomni. Mégis érdemes a legnagyobb tömörítést választanunk – amely akár kétszer annyi ideig is eltarthat – mivel feltölteni egyszer kell, letölteni általában többször is: a kicsomagolás pedig sokkal gyorsabb!
A feltöltéshez a

```
uploaddisk 192.168.1.1 gep.gz wd0
```

parancsot tudjuk használni, ahol az *IP* cím az *FTP* szerver címe vagy neve, a következő paraméter a feltöltött állomány neve, s végül ezt követi a lemez neve. Figyeljünk oda arra, hogy a program az `install` felhasználónevet használja az *FTP* műveletekhez, így hozzunk létre egy ilyen hozzáférést a szerveren. A feltöltés során a program folyamatosan közli a már felmásolt adatmennyiséget és az átlagsebességet. A lemez méretéből meg tudjuk becsülni a feltöltési folyamat idejét.
A fenti parancs elé írt

```
GZIP=n
```

hatására a megadott szám szerint fog tömöríteni, ahol az 1 jelenti a leggyorsabb és a 9 a legkisebb eredményt, a program alapértelmezésben 5-ös fokozatot használ.

Visszaállítás FTP szerverről

A mentett lemez visszaállítása egy parancsot igényel, amely teljesen hasonló a feltöltéshez:

```
slurpdisk 192.168.1.1 gep.gz wd0
```

A program rákérdez az *FTP* jelszóra, majd elkezdi letölteni és felmásolni a tömörített lemezmásolatot.

Néhány érdekesség

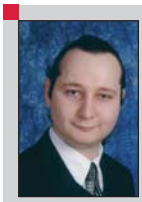
Ha nincs tele a lemez, amit mentenénk, akkor érdemes azt üres állománnyal feltölteni, így az *FTP* szerveren a tömörítés miatt helyet tudunk spórolni. Érdemes a felesleges állományokat kitörölni, így ennyivel kevesebb adatot kell mozgatnunk. A *Windows* gépeket úgy telepítsük, hogy egy kisebb rendszerpartíciót hozzunk létre, amelyre minden program felfér, s az adatokhoz egy másik partíciót hozzunk létre.
Ha nincs *DHCP* szerverünk, akkor tudunk kézzel *IP* címet rendelni a gépben lévő kártyához, azonban ennek a neve nem a (*Linux* alatt) megszokott `eth0`, hanem a gyártóra utal a név.
A kártyákról az

```
ifconfig -a
```

parancs ad tájékoztatást, így például *RealTek* kártyához az

```
ifconfig rl0 1.2.3.4 netmask
➥255.255.255.0
route add default 1.2.3.254
```

parancsokkal tudunk *IP* címet és alapértelmezett átjárót rendelni.
Végül sok sikert kívánok a program használatához, iskolai környezetben – vagyis sok egyforma géppel és vékony pénztálcával – ideális eszköz.



Auth Gábor
(auth.gabor@enaplo.hu)

Egy pécsi középiskolában informatikát és programozást oktat. Tíz éve botlott először a UNIX rendszerekbe, 7 év Linux használat után kapta el a FreeBSD lázat, amiből máig nem tudott kigyógyulni.

KAPCSOLÓDÓ CÍMEK

A cikkben említett fájlok:
➲ http://user.enaplo.hu/~auth.gabor/pov/

Naptárak megosztása

Megosztott naptárak létrehozására irányuló munkánk utolsó lépéseként lehetővé tesszük, hogy a felhasználók feltölthessék naptárjaikat a kiszolgálóra. A feladatra mindjárt kétféle megoldást is ki fogunk dolgozni.

Az elmúlt hónapok során megismerkedtünk az *iCalendar* szabvánnyal, valamint a saját naptáraink létrehozására és a távoli naptárakkal való munkavégzésre alkalmazható módszerekkel.
Messzire jutottunk, de jobban belegondolva valami még hiányzik a szolgáltatások teljes értékű kiterjesztéséhez. Láttuk, hogy saját, helyi naptárakat milyen könnyű létrehozni, ahogy a távoli naptárak lekérdezése sem ördöngösség. A távoli naptárak előállítására, terjesztésére, valamint az események web/adatbázis alapú alkalmazással történő dinamikus előállítására is találtunk megoldást. Arra azonban még nem gondoltunk, hogy az egyéni *Sunbird* felhasználók vajon hogyan oszthatnák meg másokkal saját naptárukat.
Aki dolgozott már legalább középméretű vállalatnál, az tudja, milyen nehéz egy megbeszélést összehozni. Ha mindenki naptárához biztosítani tudnánk a hozzáférést, illetve ütemezni tudnánk a résztvevők számára a megbeszéléseket és találkozókat, akkor programunk értéke jókorát nőne. Ha minden egyes a naptáramban végrehajtott módosítás a többiek számára is látható, akkor könnyebben tudják ütemezni az engem is érintő tevékenységeket. (Vagy éppen tudják, hogy mikor leszek távol, ha el akarnak titkolni valamit előlem.) Sokszor kérdeztem az ügyfeleimtől, hogy miért használnak *Microsoft Exchange*-t levelezőkiszolgálóként, hiszen kiváló nyílt forrású megoldásokat is választhatnának. A válaszokból azt szűrtem le, hogy nem annyira az elektronikus levelezés, sokkal inkább a naptárszolgáltatások miatt kötődnek az *Outlookhoz* és az *Exchange*-hez.
Ebben a hónapban azzal zárjuk le a *Sunbird* és az *iCalendar* megismerését, hogy megvizsgáljuk, hogyan lehet a naptárakat központi helyen közzétenni és másokkal megosztani. A végeredmény talán kevésbé lesz tetszetős, mint a kereskedelmi megoldások, ám biztos vagyok abban, hogy a nyílt forrású világ többi alkalmazásához hasonlóan e téren is gyors fejlődést láthatunk majd, és hamarosan olyan nyílt forrású naptárkiszolgálókat kapunk kézhez, amelyek egyenértékűek lesznek a zárt fejlesztésekkel, sőt, felülmúlják azokat.

Megosztás

Mielőtt nekifognánk a naptárunk megosztásának, pontosan meg kell határoznunk, mit értünk megosztás alatt. Gondolhatunk például arra, hogy a megosztott naptárakat egy központi helyen tároljuk, és egyszerre több naptárprogram is használja őket. Bár a *Sunbirddel* és a többi, az *iCalendar* formátum kezelésére képes programmal (ilyen például az *Evolution*) elméletileg ez is megoldható, általában nem erre van szükség.
Az *iCalendar* világában a megosztott naptár általában egy nyilvános kiszolgálóról letölthető *iCalendar* fájl. Ezt az *iCalendar* fájlt például óránként vagy naponta frissíthetik, hasonlóan az *RSS*-cikkekhez vagy a webnaplókhoz, ám előre nem lehet tudni a frissítés időpontját. Éppen ezért több feltételezéssel is élnünk kell:

- mindenki, akit érdekel az adott naptár, feliratkozott rá
- minden előfizető rendszeresen, naponta legalább egyszer letölti a naptár frissített változatát
- a naptár kezelője a lehető leggyorsabban közzétesz minden változást, és frissíti a nyilvános kiszolgáló tartalmát.

A megosztás tehát nem valós időben történik, hanem arra alapul, hogy minden érintett rendszeresen közzéteszi és letölti a változásokat. A frissítések időpontja között a felhasználók csak az utoljára letöltött, a saját gépükön található *iCalendar* fájl tartalmát látják. Ha például a naptár egy előfizetője csak naponta egyszer tölti le a változásokat, akkor előfordulhat, hogy elmulaszt egy-egy az utolsó pillanatban végrehajtott módosítást. Az, hogy milyen gyakran érdemes letölteni a frissítéseket, az adott szervezet jellegétől, a frissítések terjesztésének fontosságától és a kiszolgáló terhelhetőségétől függ. Nyilvánvaló, hogy száz embernek napi frissítést adni nem ugyanaz, mint tízezer felhasználónak óránkénti frissítést biztosítani.

FTP alapú tárolás

A fájlok internetes közzétételének legegyszerűbb módja a jó öreg *FTP* használata. A saját kiszolgálómon az *FTP* mindeddig gyakorlatilag kihasználatlan volt, nem kis részben biztonsági okokból, ám aki megfelelően

védett géppel rendelkezik, és nem akar *WebDAV*-ot használni (a leírását lásd lejjebb), annak az *FTP* jól használható megoldást kínál a naptárak megosztására.
Az én gépemen *ProFTPd* fut, ez alatt létrehoztam egy új, naptár nevű felhasználót, ennek jelszava ***naptar4atf*** lett. Ha biztosítani akarjuk, hogy távoli bejelentkezésre vagy egyéb nem kívánt célokra ne lehessen használni ezt a felhasználót, akkor az */sbin/nologin* vagy a */bin/false-both* héjat adjuk neki, illetve bármilyen más programot, ami meghívása után egyszerűen kilép, esélyt sem adva a rosszindulatú felhasználóknak a belépésre és a rendszer erőforrásainak igénybe vételére. A baj az, hogy az *FTP*-kiszolgálók csak azokat a felhasználókat engedik bejelentkezni, akik héja szerepel a */etc/shells* fájlban. Emiatt kényes döntést kell hoznunk. A naptár felhasználónak egyrészt nem interaktív héjat kellene adnunk, másrészt biztosítanunk kell neki az *FTP* használatának lehetőségét. A */sbin/nlogin* hozzáadása a */etc/shells* fájlhoz ugyanakkor biztonsági rést nyit a rendszeren. Egy egyszerű megoldás a */sbin/nologin* átmásolása a */sbin/nologin-csak-ftp* név alá, majd az utóbbi hozzáadása a */etc/shells* fájlhoz.
Alapesetben az *FTP*-n keresztül bejelentkező nem névtelen felhasználók saját kezdőkönyvtárukat látják. Alapbeállítás szerint a *ProFTPd* ennél tovább is megy egy lépéssel, és megtiltja a felhasználóknak, hogy kilépjenek a saját kezdőkönyvtárukból. Így biztosak lehetünk abban, hogy ha valaki meg is szerzi a naptárfelhasználónk jelszavát, akkor a legtöbb, amit tehet az, hogy törli vagy módosítja a naptárfájljainkat. Nyilván ennek sem örülnénk, és termelési környezetben lényegesen jobb biztonságot teremthetünk például azzal, hogy mindenkinek külön felhasználónevet és jelszót adunk.
A példa kedvéért azonban most maradjunk annál, hogy egyetlen felhasználóval dolgozunk, elfogadva azt, hogy a biztonság sérülésekor esetleg a naptárfájloktól is búcsút kell vennünk.
Feltételezve, hogy az *FTP* beállításait megadtuk, hogyan tudjuk közzétenni a naptárunkat? *Sunbird* alatt válasszuk ki a közzétenni kívánt naptárat, nálam ennek alapértelmezett neve a *My Calendar*. Megjelenik egy menü, melynek utolsó pontja a *Publish entire calendar (A teljes naptár közzététele)*. Ha rákattintunk, egy kisebb párbeszédpanel jelenik meg, ebben kell megadnunk a naptár közzétételére szolgáló URL-t.
Az nyilvánvaló, hogy az *URL ftp://*-el kezdődik, de hogyan tovább?
A fent megadott felhasználónév és jelszó használatát feltételezve, illetve a *naptar.lerner.co.il* kiszolgálót használva a teljes elérési út a *ftp://naptar:naptar4atf@naptar.lerner.co.il/naptar.ics* lesz.
Mint látható, a felhasználónév és a jelszó közé kettőspontot kell tenni, a jelszót és a kiszolgáló nevét pedig @ jellel kell elválasztani. A kiszolgáló nevét az elmenteni kívánt fájl neve követi. Bár elvileg tetszőleges nevet vagy kiterjesztést használhatunk, a *.ics* kiterjesztés a szabványos, és ennek használatával biztosítható, hogy minden érintett program megfelelően tudja értelmezni a *MIME* típusokat.
Most végezzünk valamilyen módosítást a naptáron. Vajon kézzel kell feltöltenünk a kiszolgálóra, újra végiglépkedve ugyanazon az eljáráson? Nem, szó sincs erről. Kattintsunk a naptár nevére, ekkor megjelenik a már látott menü. A *Publish entire calendar* parancs helyett most az *Edit calendar (Naptár szerkesztése)* parancsot kell választanunk. Ekkor megnyílik egy párbeszédpanel, mely – egyebek mellett – egy szövegmezőt is tartalmaz, ebbe be tudjuk írni a megfelelő *URL*-t, továbbá található itt egy jelölőnégyzet is, amellyel előírhatjuk a naptár minden egyes változást követő közzétételét. Bennem ez a szolgáltatás elég vegyes képet hagyott, bár inkább működött, mint nem, és jó szolgálatot tett a találkozóim különféle rendszerek közötti szinkronizálásában.
A megosztott naptárra a közzétételhez hasonló módon iratkozhatunk fel. Adjuk meg a teljes *URL*-t, ide értve a felhasználónevet és a jelszót is, és minden az *iCalendarnak* megfelelő programnak képesnek kell lennie a letöltésére és a megjelenítésére. Természetesen ennek előfeltétele, hogy az adott program képes legyen a *HTTP* alapú hitelesítés kezelésére.

1. kódlista

```
<VirtualHost 69.55.225.93>
ServerName davnaptar.lerner.co.il
ServerAdmin naptar@lerner.co.il
# Directory and file names not beginning with /
# are relative to ServerRoot
ServerRoot /usr/local/apache/v-sites/davnaptar.lerner.co.il
DocumentRoot www
ErrorLog logs/error-log
CustomLog logs/access-log combined
CustomLog logs/referer-log referer
DAVLockDB  DAVLock
  <Directory
  /usr/local/apache/v-sites/davnaptar.lerner.co.il/www/>
    DAV On
    <Limit PUT POST DELETE PROPFIND PROPPATCH MKCOL
           COPY MOVE LOCK UNLOCK>
      AuthName "A naptár DAV alapú elérése"
      AuthType basic
      AuthUserFile passwd
      Require user naptar
    </Limit>
  </Directory>
</VirtualHost>
```

mod_dav

Az *FTP* kiválóan megfelel bizonyos feladatokra, de alkalmazásának számos hátránya is van. Először is, akiben kellemetlen emlékeket hagytak az *FTP*-kiszolgálókkal kapcsolatos biztonsági problémák, az valószínűleg nem szeretne ilyet futtatni. Lehetséges, hogy teljesítménybeli szempontok miatt minden forgalmat *HTTP* felett szeretnénk bonyolítani, ami mellett további érv az *SSL* titkosítás használatának lehetősége is. Sokféle szempont szól tehát amellett, hogy megpróbálkozzunk egy másik megoldással, és ez a *mod_dav*.

A *DAV (Distributed Authoring and Versioning, elosztott alkotás és változatkezelés)* segítségével nemcsak lekérni és olvasni tudjuk a fájlokat, de létre is hozhatunk ilyeneket a kiszolgálón, illetve módosíthatjuk is a meglévőket. A *DAV* tehát írásra és olvasásra is alkalmas protokollá változtatja a *HTTP*-t. A *DAV* már jó néhány éve létezik, és az *Apache 1.x* és *2.x mod_dav* modulja sem kifejezetten újdonság. Nálam a főkiszolgálón továbbra is *Apache 1.x* fut, ám a *mod_dav* telepítése *Apache 2.x* alatt sem lehet nehezebb.

Az első lépés a *mod_dav* letöltése (lásd az internetes forrásokat). Mivel én az *Apache*-ot *DSO* (megosztott objektum) képességekkel fordítottam le, a *mod_dav* beépítése miatt nem kellett teljesen újrafordítanom. Elég volt megadnom neki, hogy hol találja az *apxs*-t, azt az önműködően előállított *Perl* programot, mely az összes olyan információt biztosítja az *Apache* moduloknak, amely azok az *Apache* forráskódja nélkül történő lefordításához szükséges. A *mod_dav* forráskódjának kibontása után kiadtam a következő parancsot:

```
./configure --with-apxs=/usr/
➥local/apache/bin/apxs
```

Miután végzett, lefordítottam és telepítettem a *mod_dav*-ot:

```
make
make install
```

Ellenőriztem, hogy az *Apache* beállító fájlja, a *httpd.conf* a `make install` által végrehajtott módosítások során nem sérült-e meg. Ez után megadtam egy új nevesített képzetes kiszolgálót az *Apache*-nak, ennek neve *davnaptar.lerner.co.il* lett *(1. kódlista)*. Érdemes átfutni a fenti beállításhalmaz *DAV*-val kapcsolatos részeit.

A `DAVLockDB` a *DAV-zárak* helyét adja meg, ez természetesen a *HTTP*-n keresztül elérhető `DocumentRoot` könyvtáron kívül helyezkedik el. Ez után a kívánt könyvtárra vonatkozóan bekapcsolom a *DAV*-ot, majd a *DAV* alapú elérést a naptár felhasználóra korlátozom, akinek a jelszava egy külső fájlban található. A jelszófájl szintén a webhely gyökérkönyvtárán kívül található, létrehozására és tartalmának frissítésére a *htpasswd* program használható, mely alapesetben a */usr/local/apache/bin* könyvtárban található.

Végül a `<Limit>` rész az esetlegesen veszélyes kérésekre korlátoz. Például a normál *HTTP GET* kérésnél nincs szükség felhasználónév és jelszó megadására. Úgy vélem, a fenti beállítások elég jól használhatók, ha a naptárra való előfizetést bárki számára lehetővé akarjuk tenni, ám a naptárfájl közzétételére és módosítására jogosultak körét már korlátozni akarjuk. Ha a naptárat üzleti célokra használjuk, akkor nyilván az olvasását is korlátozni kell, például úgy, hogy mindenkinek külön felhasználónevet és jelszót adunk.

A közzétételhez ismét nyissuk meg a kívánt naptárhoz tartozó *Publish entire calendar* párbeszédpanelt. Ezúttal egy *HTTP URL*-t használunk, felhasználónév és jelszó megadása nélkül:

```
http://davnaptar.lerner.co.il/
➥calendar.ics.
```

Ezzel közzétesszük a naptárat a webhelyen, ahogy az a kiszolgáló megfelelő könyvtárának tartalmára tekintve is megállapítható. A naptár frissítéseit *WebDAV* használatakor is a korábban látott módon tehetjük elérhetővé.

Végül meg kell említeni, hogy a naptárra feliratkozni a korábbi hónapok során látott módszerekkel lehet. Válasszuk a *File (Fájl)* menü *Subscribe to remote calendar (Feliratkozás távoli naptárra)* parancsát, majd írjuk be a naptárfájl *URL*-jét. A *WebDAV*-nak köszönhetően az írást és az olvasást ugyanazon *URL* használatával végezhetjük el.

Összefoglalás

Bár a nyílt forrású világban még nincs olyan csillogó-villogó naptárkezelő háttérrendszer, mint a *Microsoft Exchange*, léteznek nála rugalmasabb, a legtöbb munkacsoport igényeit kielégítő megoldások.

Természetesen nem titkolhatom el, hogy a *Sunbirdnek* vannak gondjai a közzététel és a feliratkozás terén. Például a saját *Sunbirdömben* magánjellegűként megjelölt találkozók a fájl feltöltésekor is megőrizték ezt a jelölést – aztán (magánjellegűként) ismét megjelentek, amikor egy másik program alól feliratkoztam a naptárra.

A *Sunbird* jelenleg elég lassan tudja csak kezelni a nagyméretű naptárakat; ugyanakkor ezekről a hibákról már a fejlesztők is értesültek, és a következő hónapokban várhatóan javítják őket.

A *Novell Hula* tervezetén belül várhatóan újabb az *iCalendar* fájlok kezelésére képes kiszolgáló is fog készülni. Amióta a *Novell* felvásárolta a *Ximiant* és a *SUSE*-t, a *Hula* a kibővült cég egyik legnagyobb várakozások elé néző újdonsága. Ha a *Hula* valóban képes lesz az *iCalendar* támogatására, akkor kíváncsian várom, vajon a fent ismertetett *FTP* és *WebDAV* alapú megoldásokhoz képest milyen pluszt lesz képes nyújtani. Addig is azonban léteznek működőképes megoldások, melyek nemcsak az én igényeimet voltak képesek kielégíteni, de sok a közös munkában rejlő lehetőségeket kereső csoport elvárásainak is megfelelnek.

Linux Journal 2005., 136. szám

A cikkhez tartozó források elérhetősége:
➲ *www.linuxjournal.com/article/8323*

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

A Blender használata (9. rész)
Karakter-animáció

A 8. részben bemutattuk az IPO görbéket. Ezek alkotják a Blenderben az animáció alapját, a legutóbbi részben leírtakra tehát most is szükség lesz. Az eddigi animációink nagyon egyhangúak voltak, legfeljebb mozgatni tudtuk az objektumokat, azokat nem tudtuk deformálni. Ebben a részben megpróbálom bemutatni a csontvázak létrehozását és a velük történő animációt.

Az elefántok álmodnak

Körülbelül egy éve annak, hogy a ***Blender Foundation***-nél szárnyra kapott egy ötlet, a nyílt forrású programmal illene már egy nyílt forrású animációs filmet létrehozni. Hogy mi lehet egy filmen nyílt a tartalmán kívül, azt mindenki megtekintheti az ***Elephants Dream*** nevű film képében. Hála az ***Orange projektnek*** (így hívták az egy év alatt), nemcsak a film tölthető le többféle felbontásban, de a hozzátartozó összes .blend kiterjesztésű fájl a vázlatokkal, storyboard-al, teljes dokumentációval együtt. Az ***Orange Open Movie Teamnek*** nevezett csapat fél évig éjt nappallá téve dolgozott az animációs filmen, majd miután kipostázták az előre megrendelt több száz ***DVD***-t a teljes projektet ingyenesen elérhetővé tették mindenki számára. Akit tehát érdekel, hogyan készül egy igazi animációs film, és rendelkezik megfelelő sávszélességgel, látogasson el a cikk végén megadott címre. Ha nem is töltjük le a teljes anyagot, a világ első ***Open Movie***-ját akkor is érdemes megnézni. Gratulálunk ***Orange Team***!

Ahhoz, hogy csontvázzal animálhassunk, három dolog feltétlenül szükséges. Legyen valamink, amit animálni tudunk. Érdemes ezzel a kérdéssel foglalkozni először. A ***Blender*** elég engedékeny: egy csontvázzal akár több objektumot is mozgathatunk egyszerre. Ha kész az animálandó karakterünk, létre kell hoznunk neki egy csontvázat. (A ***Blender Armature*** néven hivatkozik rá) Később ezt mozgatva fogjuk animálni karakterünket.
Előtte azonban a csontvázat hozzá kell rendelnünk a mozgatni kívánt objektum(ok)hoz, vagy egy ***mesh vertex group***-jaihoz. Mint később látni fogjuk, ez történhet automatikusan, de megadhatunk saját ***group***-okat is. Ha mindezt megtettük, a ***Pose Mode***-ba váltás maradt az utolsó dolgunk, mielőtt belefeledkeznénk a rongybabánkkal való önfeledt szórakozásba.

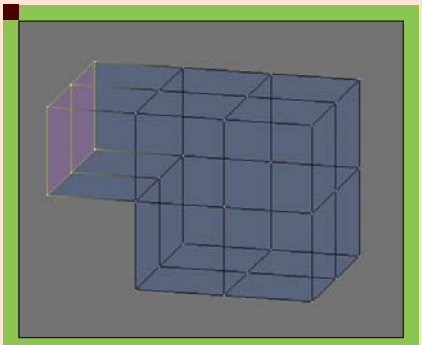
Sensei

Mivel minden valamirevaló rongybaba rendelkezik saját névvel, a miénket ***Senseinek*** fogják hívni. ***Sensei*** nem lesz túl összetett, mivel a hangsúly most az animáción, nem pedig a modellezésen van. Létrehozása senkinek sem okozhat problémát.

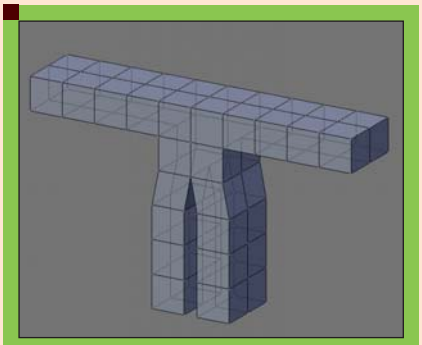
Egy kockából indulunk ki (*1. ábra*), amire rögtön indulásként ráeresztünk egy ***subdivide***-ot (***W -> Subdivide***). A *2. ábrán* látható módon ***extrude***-dal kihúzzuk (***E***), majd az ***S*** és ***X*** billentyűk segítségével vastagságát a felére csökkentjük (*3. ábra*). Ezek után már csak egy ***SubSurf*** hiányzik, és kész is vagyunk. Vagy mégsem? Fej nélkül nem ***Sensei*** a ***Sensei***. Egy kocka hozzáadásával, és egy újabb ***SubSurf***-fel ezt is könnyen orvosolhatjuk. Végül ne felejtsük el a ***CTRL+J*** billentyűkombináció segítségével a fejet végérvényesen a testhez csatolni. ***Sensei*** teljes valójában a *4. ábrán* látható és bár üresnek, és mozdulatlannak tűnik, nemsokára megpróbáljuk életre kelteni.

Armature

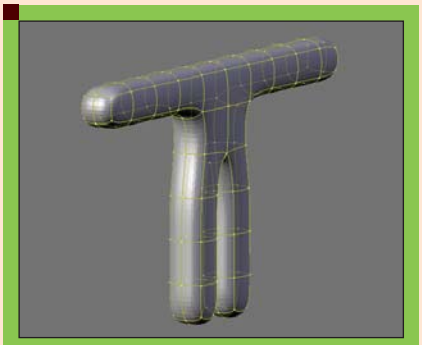
Bár az ***Armature*** szó fegyverzetet jelent, én mégiscsak csontváznak fogom hívni. Ennek oka, hogy a ***Blenderben Armature***-nek hívott objektum egymással összekapcsolt (vagy épp össze nem kapcsolt) „csontok" (***Bone***) halmaza. Ahhoz, hogy ***Sensei*** képes legyen mozogni, csontokra van szüksége. Első lépésként hozzunk létre egy objektumot, ami a csontokat fogja tartalmazni: nyomjuk meg a szóközt, válasszuk ki az ***Add*** menüt, majd azon belül az ***Armature*** opciót (*5. ábra*).
Az így létrehozott csontvázunk jelenleg egyetlenegy csonttal rendelkezik.



1. ábra Sensei készül (1. fázis)



2. ábra Sensei készül (2. fázis)



3. ábra Sensei készül (3. fázis)

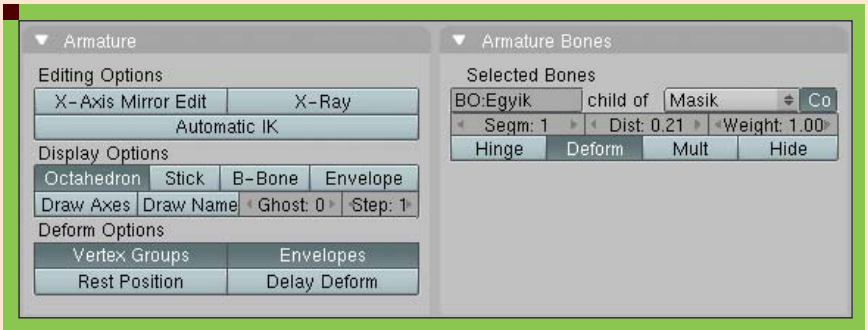
4. ábra Sensei teljes valójában

Object Mode-ban ugyanúgy kezelhetjük, mint a többi objektumot: mozgathatjuk, forgathatjuk, átméretezhetjük, *Edit Mode*-ban csontokat adhatunk hozzá, vehetünk el, stb. tehát kialakíthatjuk a csontvázunkat. Van azonban egy harmadik, *Pose Mode*-nak nevezett mód is. Ha ebben a módban mozgatjuk a csontvázat, a hozzárendelt objektum(ok) a csontvázzal együtt deformálódnak. De ne ugorjunk ennyire előre. Jelöljük ki a csontvázunkat, majd váltsunk *Edit Mode*-ba. Kétféle módon hozhatunk létre új csontot. Kijelölhetünk egy már meglévő csatlakozást (ízületet), majd az *E* (*Extrude*) billentyűvel újat csatolhatunk hozzá, illetve az *Add* menü *Bone* parancsával egy független csontot adhatunk hozzá ugyanahhoz a csontvázhoz. Az előbbi módszerrel tetszőleges fa struktúrát alakíthatunk ki, míg az utóbbival új „fát” kezdhetünk. Az *Armature* illetve *Armature Bones* panelek sokat segíthetnek a csontváz megértésében.

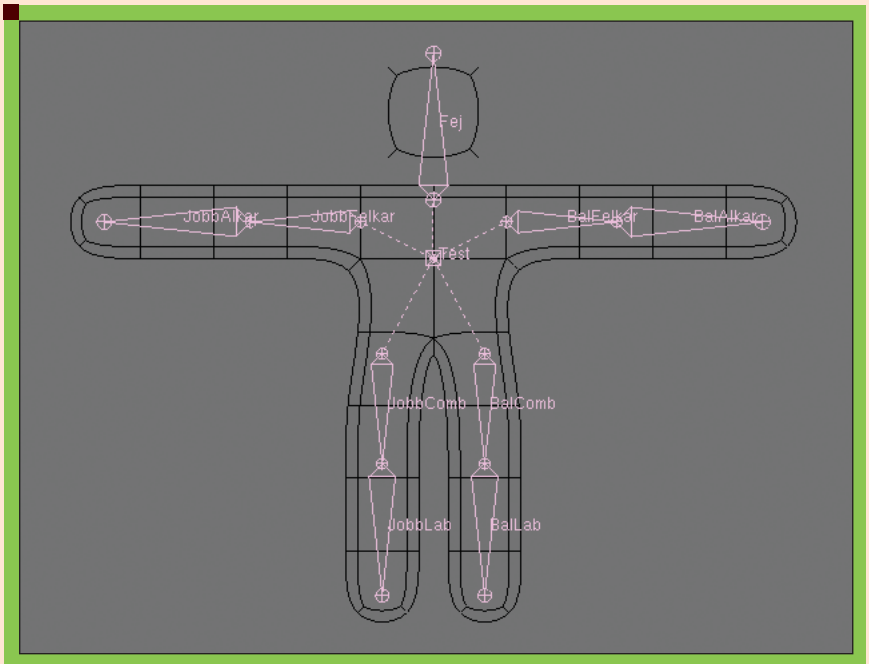
Rögtön kapcsoljuk is be a *DrawNames* opciót, hogy lássuk a csontok neveit. Az egyik (ha nem a legfontosabb) dolog, hogy csontjainknak mindig adjunk beszédes neveket. A későbbiekben nagyon megkönnyítheti a dolgunkat, ha *„Bone.123”* helyett csak *„bal.fül”*-re hivatkozunk. Egy csont nevét (miután a csontot kijelöltük) az *Armature Bones* nevű panelen írhatjuk át.

Rögtön a névtől jobbra megtalálhatjuk a *Parent* mezőt, ahol megadhatjuk a csont szülőjét (tehát azt a csontot, amihez csatlakozik). Ezek ismeretében tetszőlegesen bonyolult csontvázat létrehozhatunk, az egyetlen megkötés a *Blenderben* az, hogy egy csontnak csak 1 szülője lehet. Ebből következik, hogy csak fa struktúrájú csontvázat hozhatunk létre, ahol a „fa” gyökere az a csont, amelynek nincs szülője. Abban viszont nincs megkötés, hogy egy *Armature* objektumon belül hány ilyen fát hozunk létre.

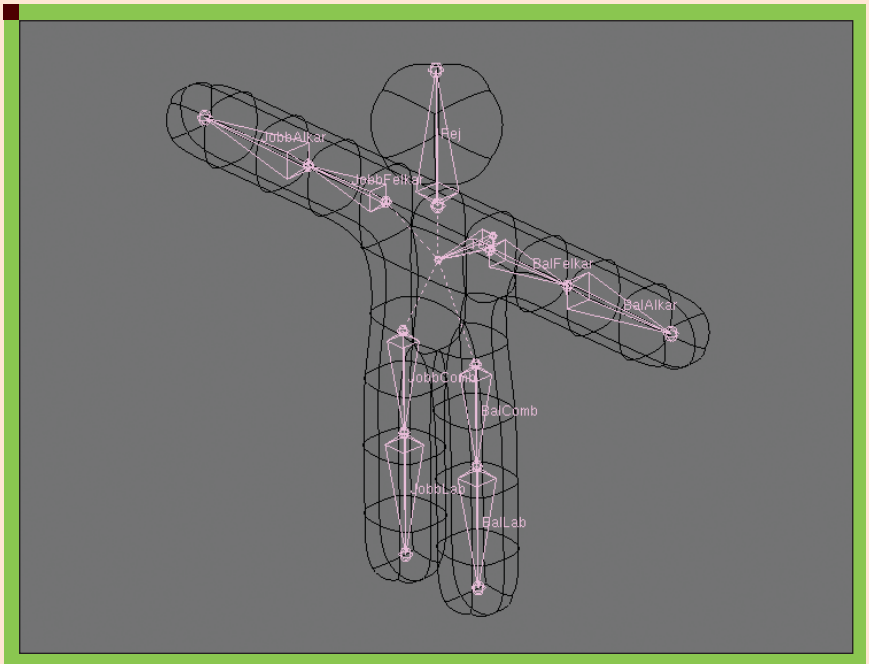
5. ábra Csontváz létrehozása



6. ábra A Csontváz és a csontok beállításai

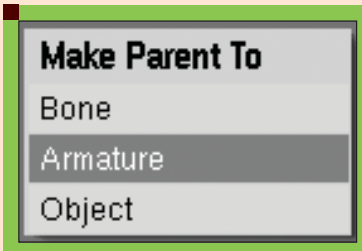


7. ábra Sensei csontváza (1. fázis)

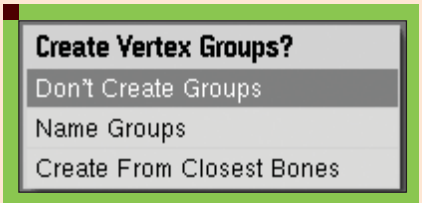


8. ábra Sensei csontváza (2. fázis)

Az ***Armature Bones*** panelen találhatunk egy jelentéktelennek tűnő ***Co*** feliratú gombot. Hogy ne legyen ilyen egyszerű a dolgunk, ez a gomb csak akkor jelenik meg, ha a kijelölt csontnak van szülője. A mi esetünkben a ***Co*** nem a szénmonoxid képlete vagy a kobalt vegyjele akar lenni, hanem a ***Connect*** szócska rövidítése, ami annyit jelent, hogy a gomb bekapcsolt állapotában az adott csont egyik vége csatlakozik a szülőjéhez, míg kikapcsolt állapotban mindkét vége teljesen szabadon mozgatható, csak egy szaggatott vonal jelzi a hovatartozást (*6. ábra*).
Ha elég bátorságot érzünk magunkban, a *7.* és *8. ábra* alapján készítsük el ***Sensei*** csontvázát.
A végletekig leegyszerűsítve egy csont felelős a fej mozgatásáért, kétkét csont a végtagokért és egy az egész testért. Ez utóbbi legyen a végtagok és a fej szülője.



9. ábra Skinning (1. fázis)



10. ábra Skinning (2. fázis)

Skinning

Ahhoz, hogy ***Pose*** módban a csontvázat mozgatva az adott objektum is mozogjon, hozzá kell rendelnünk az objektumot, illetve annak részeit az egyes csontokhoz. A ***Blender*** ezt ***skinningnek*** hívja. Több lehetőségünk is van. Váltsunk ***Pose Mode***-ba, jelöljük ki ***Senseit***, majd a ***SHIFT*** gombot nyomva tartva az egyik csontot. Ha most megnyomjuk a ***CTRL+P*** (***Parent***) gombot, választhatunk a ***Blender*** által felkínált lehetőségek közül (*9. ábra*).

Bone – A kijelölt objektumot teljes egészében az adott csonthoz rendeljük. Több statikus részből álló gépek (például robotok) animálásánál jöhet jól. Nincs megszabva, hogy egy csontvázhoz hány objektumot rendelhetünk.
Object – A ***Blender*** hagyományos szülő/gyermek hozzárendelése. A csontvázat is egyszerű objektumként kezeli, így az nem deformálja ***Senseit***.
Armature – Ez a mód ***Sensei Vertex Groupjait*** rendeli hozzá az egyes csontokhoz. Név alapján összepárosítja a ***Vertex*** csoportokat és a ***Csontokat***.

Ha ez utóbbi módot választjuk (ezt fogjuk), akkor a ***Blender*** további három opciót kínál fel a ***Vertex*** csoportok létrehozására (*10. ábra*).

Don't Create Groups – Nem hoz létre vertex csoportokat. Azokat nekünk kell kézzel létrehozni, vagy más módon megoldani (például ***Envelopes***) a hozzárendelést.

11. ábra Sensei él...

Name Groups – Létrehozza a Vertex csoportokat, azok azonban üresek, nekünk kell „kézzel” feltölteni őket. Egy vertex több csoportba is tartozhat, ilyenkor lehetőség van súlyozásra, megadhatjuk melyik csont mekkora erővel hasson a vertexeire.
Create From Closest Bones – Automatikusan létrehozza a csoportokat, és a vertexeket a legközelebbi csontokhoz rendeli. Bár kétségkívül ez az egyik leggyorsabb és legkényelmesebb megoldás, a végeredményen ez annyira nem látszik. Bár sokat javítottak az algoritmuson a régebbi *Blender* verziókhoz képest, ha automatikus hozzárendelést szeretnénk, mégis inkább az alább leírt *Envelope*-os módszert javaslom.

Envelopes

Az *Armature* panelen lehetőségünk van megváltoztatni a csontok kinézetét (*Display Options*).
Ha itt az *Envelope*-ot választjuk ki, a csontok körül megjelenik egyfajta fehér „erőtér”. Ezek a csontok és a fehér terület kiterjedése csontonként változtathatók az *Armature Bones Panel „Dist”* (*Distance*) gombjával. Egy vertex automatikusan ahhoz a csonthoz tartozik, amelyik erőterébe beleesik. Találhatunk még egy *Envelopes* feliratú gombot (figyeljünk az **s** betűre a szó végén) is, amivel be illetve kikapcsolhatjuk az efféle deformációt. Ugyanezt megtehetjük a *Vertex Group*-os hozzárendeléssel is, de akár kombinálhatjuk is a kettőt.
Általában elmondható hogy a csontvázzal való animáció legkínosabb része a hozzárendelés. *Sensei* esetében (mivel nem egy összetett mesh-ről van szó) javaslom az *Name Groups* opció kiválasztását a fent említett módon (*CTRL+P*, *Armature*, *Name Groups*), majd kézzel hozzárendelni a vertexeket a létrehozott csoportokhoz. Az ízületeknél lévő vertexeket érdemes mindkét csonthoz hozzárendelni.

Pose Mód

Miután megfelelően hozzárendeltük *Sensei* testrészeit a csontokhoz, elkezdhetünk játszadozni vele. Ha *Edit Mode*-ban mozgatjuk a csontokat, azoknak nincs hatása a vertexekre, azonban ha ugyanezt *Pose Mode*-ban tesszük (és mindent megfelelően beállítottunk), *Sensei* bizony életre kel, csontvázához hűen ragaszkodva ő maga is mozogni kezd. A csontokat a *G* (*Grab*), *R* (*Rotate*) és *S* (*Scale*) billentyűkkel majdnem a megszokott módon mozgathatjuk, talán az egyetlen megkötés, hogy egy csont (alapesetben) nem mozdíthatja ez a szülőjét.
Az armature panelen találunk egy *Automatic IK* névre hallgató gombot. Ezt bekapcsolva aktiválhatjuk a *Blender Inverz Kinematikáját*. Ilyenkor a mozgó csontok hatnak a szülőkre is, de csak abban az esetben, ha az *Armature Bones* panelen bekapcsoltuk a *Co* gombot, tehát a csont és szülője között van összeköttetés.

Karakter-animáció

Nincs más hátra, mint előre, animáljunk. A könnyebb kezelhetőség kedvéért váltsunk át az *Animation* ablak elrendezésre. Ehhez használjuk a *CTRL + Jobbra nyíl* gombokat. Mivel a csontokat animáljuk, nem az objektumot, ezért az *IPO Editorban* (remélem, még emlékszünk rá az előző részből) kapcsoljuk át a görbe típusát *Object*-ről *Pose*-ra. Innentől nincs más dolgunk, mint a már megszokott módon – az *I* billentyű megnyomásával – létrehozni, a keyframe-eket, amik között a *Blender* majd interpolál. Ilyen módon minden csontot külön animálhatunk, de használhatjuk a (már szintén bemutatott) *Contraints* panelt, amivel különböző megszorításokat adhatunk az egyes csontoknak, így lehetséges például, hogy egy csont mindig egy adott objektum felé mutasson, vagy egy megadott görbén haladjon végig, stb.
A tizedik (utolsó) részben előreláthatóan a *Blender* részecske és folyadék szimulációjáról, és hasonló finomságokról lesz szó. Megpróbálunk majd tüzet, vizet, tűzijátékot, szökőkutat, vagy épp macskaszőrt renderelni. Addig is jó *Linuxvilág* olvasást.

Szalai András
(sly87@freestart.hu)

Jelenleg középiskolába jár, ahol informatikát tanul. Jövőre érettségizik. Hobbija a programozás és a biztonságtechnika, és a továbbtanulási szándékai is ilyen irányúak.

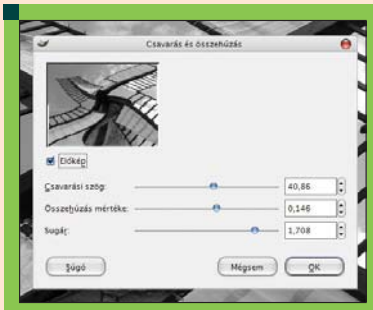
KAPCSOLÓDÓ CÍMEK

Az Orange Project weboldala:
➲ http://www.elephantsdream.org/

Gimp bővítmények (4. rész)
Söndörödik, pöndörödik, bödörödik...

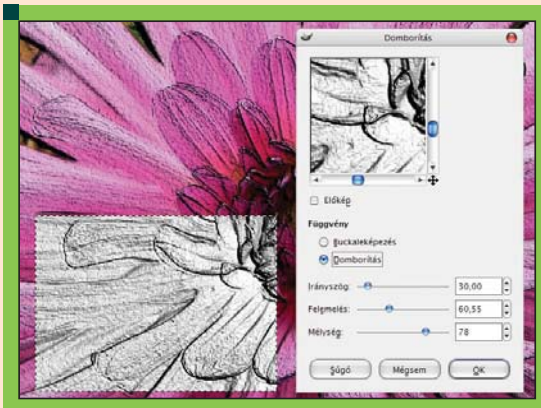
Vidámparkban járva sokan láttunk már mágikus tükröket, amelyek hol nagyobbnak, hol kisebbnek, netalántán szélesebbnek mutattak bennünket a ténylegesnél. A tükrök egészen egyszerűen torzítják a képet. Olyan játék ez, amivel kortól függetlenül mindenki szívesen játszik egy kicsit. Ne maradjunk ki mi sem, hiszen varázstükörre sincs szükségünk, csak indítsuk el a GIMP-et és vegyük sorra a torzító szűrőket!

Domborítás, csavarás, hullámosítás – megannyi érdekesen hangzó lehetőséggel állunk szemben a *[Szűrők>Torzítás]* menüpont tartalmát böngészve. A *Gimp* bővelkedik eszközökben, amelyekkel manipulálhatjuk és újraértelmezhetjük képeinket. Kezdjük az ismerkedést a ***Csavarás és összehúzódás*** fedőnévre hallgató szűrővel(***Whirl and Pinch***)! Válasszunk egy mozgalmas fotót, majd indítsuk el a szűrőt! A barátságos felületen pillanatokon belül csodákat művelhetünk. Sőt, még pilótavizsga sem kell hozzá, hiszen bármit is változtatunk, azonnal frissül az előkép, így egyből láthatjuk mi történt. Az én választásom egy franciaországi szélmalom képére esett, mivel úgy remélem, hogy a vitorlákon szépen látja majd mindenki a csavarást és az összehúzást. Állítsuk be elsőként a sugarat (***Radius***)! Ezzel döntjük el, hogy a kép mekkora részét változtatjuk majd. A sugarat a kép közepétől számítja a program, tehát ha kis értéket adunk meg itt, akkor minden további változtatásunk csak a kép kicsiny középső részét fogja érinteni. (Ha a képünknek nem a közepét szeretnénk változtatni, akkor jelöljük ki a kívánt képrészletet és a kijelölésre alkalmazzuk a szűrőt!) Ha sikerült eldönteni a sugár méretét, akkor jöhetnek a látványos beállítások! Képzeljük úgy el a képünket, mintha egy gumilapra lenne ráfestve. Így egyből érthetővé válik a ***Csavarási szög*** (***Whirl Angle***) és az ***Összehúzás mértéke*** (***Pinch Amount***) is. A szög esetében a negatív érték ellentétes irányt jelent, a negatív összehúzódás viszont nem más, mint nyújtás. A ***Domborítás*** (***Emboss***) szűrő három dimenzióba helyezi a képünket. Legalábbis részben. Megkeresi ugyanis az éleket és átrajzolja őket a beállított fényviszonyoknak megfelelően. Használhatunk két algoritmust is. Az egyik a ***Domborítás*** (***Emboss***) fekete-fehér eredményt ad miközben

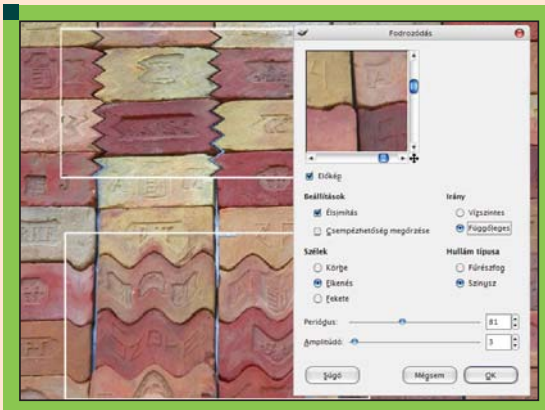


1. ábra A Csavarás és összehúzás szűrő

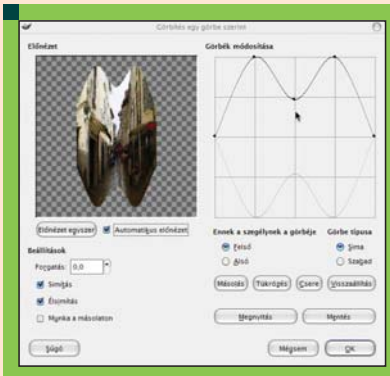
2. ábra A Csavarás és összehúzás szűrő hatása



3. ábra A Domborítás szűrő



4. ábra Hálás eszköz a Fodrozódás szűrő



5. ábra A görbék kezelése

plasztikussá változtatja a képünket, míg a másik algoritmus, a buckaleképzés ennél kifinomultabban használható, hiszen megtartja a kép eredeti színeit.

Képzeljük el, hogy a képünket lefektetjük a szabadban egy asztalra. A nulla fokos irány ekkor a kép bal oldala felé található. Ezt adhatjuk meg az ***Irányszög*** (***Azimuth***) értékével, mely megmondja, hogy honnan jön a fény. Mivel a fényt térbelinek képzeljük, így nem csak iránya van, hanem magassága is. Hasonlóképpen, ahogyan a napnak is. Ezt az elég sután fordított ***Felemelés*** (***Elevation***) paraméterrel állíthatjuk. A ***Mélység*** (***Depth***) pedig a fényforrás távolságát jelenti. Egyszerű, ugye? Az illusztrációnak választott virágon jól látható, hogy milyen plasztikussá tette a szűrő a képet. A fekete-fehér részhez a ***Domborítás*** algoritmusát, míg a kép többi részén a ***Buckaleképzést*** használtam.

Az ***Eltolásra*** (***Shift***) nem vesztegetnék sok időt. Hiszen nem csinál mást, mint beállítástól függően vízszintesen vagy függőlegesen eltolja a képpontokat. Mi az elmozdulás értéket adhatjuk meg. Az egyetlen csavart itt viszi véghez a szűrő, hiszen nem mindig a megadott távolságra mozdítja el a pontokat, hanem kivon belőle egy megfelelően kicsi véletlen számot, hogy a hatás természetesebb legyen...

Hálás eszköz a ***Fodrozódás*** (***Ripple***). Illusztrációnak egy dísztéglákból kirakott falat választottam. A felső bekeretezett részen vízszintes irányú farkasfog alakú, az alsón függőleges szinusz hullámot használtam. Megadhatjuk az irányon kívül a ***Periódust*** és az ***Amplitúdót*** is. Érdemes bekapcsolni az élsimítást, ez által sokkal szebb eredményt kapunk. Hasznos opció a ***Csempézhetőség*** (***Retain Tileability***) megőrzése. Biztosan mindenki találkozott már ismétlődő hátterekkel, ha máshol nem, akkor honlapokon. A csempézhetőség azt jelenti, hogy olyan képet kapunk eredményül, ami ismétlődő mintát képes létrehozni úgy, hogy a határ nem észrevehető. Ha azonban nem választjuk ki ezt a lehetőséget, akkor döntenünk kell arról, hogy a kép szélein mit tegyen a program. Hagyhatjuk feketén a széleken kimaradó pixeleket, vagy elkenhetjük őket. A ***Körbe*** kapcsolót választva a bal oldalon túllógó pixelek átkerülnek a jobb oldalra és így tovább.

Elsőre meglátva a ***Görbítés egy görbe szerint*** (***Curve bend***) szűrőt, rögtön az jutott eszembe, hogy ezzel másodpercek alatt lehet olyan szöveget készíteni mint ami a ***Csillagok háborújában*** van. Nos, arra azért létezik egyszerűbb és jobb megoldás, név szerint a perspektíva eszköz. Viszont nevéhez méltón remekül használhatjuk ezt a szűrőt a képeink torzításához. Mivel szerencsére a gombok magukért beszélnek, így csak röviden összefoglalom a lehetőségeket. Két görbénk van ugyanis. A felsőt fogja követni a képünk teteje, míg az alsó görbéhez pedig a képünk alja lesz hozzáigazítva. Amint bekapcsoljuk az előnézetet, már láthatjuk is a hatást. A görbéket vagy pontjaik megadásával, vagy szabadkézi rajzzal adhatjuk meg.

Ha egy pontot feleslegesen helyeztünk el, akkor csak fogjuk meg és húzzuk jobbra míg el nem tűnik!

Ne felejtsük, hogy éppen azon a görbén dolgozunk, amelyiket kijelöltük a megfelelő kapcsoló kiválasztásával! Ha sikerült szépen kialakítani az egyik görbét, érdemes lemásolnunk. A másolás és a tükrözés gomb az aktív görbét veszi alapul a másik létrehozásához. Így lehet gyorsan és egyszerűen szimmetrikusan torzítani a képünk. A görbéket fájlba menthetjük és később is felhasználhatjuk szükség szerint.

A ***Gimp*** számos eszközéből ismét szemügyre vettünk párat. Próbáltam mindent képekkel illusztrálni, hogy gyorsan fel lehessen mérni a felhasználási lehetőségeket. Legközelebb folytatjuk még a Torzítás menüpontot, hiszen számos remek lehetőség felett átsiklottunk.

Juhász Attila
(rabszolga@goraffe.hu)

Az Információ Technológiai Kar hallgatója a Pázmány Péter Katolikus Egyetemen. Érdeklődik a bioinformatika és a neurális hálózatok iránt. A fotózás és a tánc mellett öt éve foglalkozik webgrafikákkal. A linux terjesztések közül a Gentoo és az Ubuntu áll legközelebb a szívéhez. Fotós oldala a http://people.goraffe.com/attila címen található.

A Blender és a Yafray tuningolása

A legtöbb esetben nincs szükségünk arra, hogy forráskódból telepítsünk egy programot, és nincs is mindig értelme. De mikor lehet szükségünk rá mégis? Ha nagy számításigényű programot futtatunk és szeretnénk a legjobban optimalizált kódot használni, hogy a gépünk lehetőségeit a legjobban kihasználjuk. Vagy ha szeretnénk az adott programból a legfrissebb verziót használni, esetleg nem akarjuk megvárni míg megjelenik a disztribúcióhoz tartózó csomag. A Blender esetében mind a kettő erősen szerepet játszik, ezért személy szerint kizárólak forrásból használom.

A forráskód és a szükséges programok beszerzése

A *Blender* forráskódját sokféleképpen szerezhetjük be. Közvetlenül letölthetjük a *CVS*-ből, vagy a stabil verziót csomagolt formában. Ez vonatkozik a *Yafray*-ra is. Mi az a *CVS*? A program fejlesztésére használt verzió kezelő, változás követő rendszer. A programozok ezen keresztül fejlesztik a programot. Ha ezt használjuk, akkor biztosak lehetünk benne, hogy a program legfrissebb verzióját használjuk. A csapda ott van, hogy a program éppen fejlesztés alatt van, ezért rengetek hiba lehet a programban, sőt az is előfordulhat, hogy nem tudjuk lefordítani. Természetesen nem kell választanunk a kettő között, kipróbálhatjuk mindegyiket akár szimultán is. Érdemes ellátogatni a következő címre ➲ http://www.blender3d.org/cms/The_Release_Cycle.361.0.html ahol részletesen le van írva, hogyan működik a *Blender* fejlesztése.
A mindenkori stabil forrást pedig erről a helyről tudjuk letölteni ➲ http://download.blender.org/source/.
Hozzunk létre egy külön könyvtárat a manipulációnkhoz és tömörítsük ki ebbe a mappába a forráskódokat:

```
$ mkdir blender_cvs_src
$ cd blender_cvs_src
$ tar xzvf blender-2.41.tar.gz
```

vagy *CVS*-t használva:

```
$ cvs -d:pserver:
➥anonymous@cvs.blender.org:/cv
➥sroot/bf-blender login
$ cvs -z3 -d:pserver:
➥anonymous@cvs.blender.org:/cv
➥sroot/bf-blender co blender
```

vagy egy nem hivatalos *CVS* ág esetén (több újdonságot tartalmaz, gyorsabban frissül):

```
$ cvs -d:pserver:
➥anonymous@cvs.blender.org:/cv
➥sroot/tuhopuu login
$ cvs -z3 -d:pserver:
➥anonymous@cvs.blender.org:/cv
➥sroot/tuhopuu co tuhopuu3
```

A *Yafray* esetén hasonló módon:

```
$ cvs -d:pserver:
➥anonymous@cvs.blender.org:/cv
➥sroot/yafray login
$ cvs -z3 -d:pserver:
➥anonymous@cvs.blender.org:/cv
➥sroot/yafray co yafray
```

Illetve a *Yafray* stabil forráskódját töltsük le a ➲ http://www.yafray.org/sec/2/downloads/ címről.
Ha ezzel megvagyunk akkor birtokunkban van a két forráskód. Persze még kellenek az egyéb függőségek is, amiket az *1. táblázat* tartalmaz.
Nagy valószínűséggel ezek már mind fenn vannak a gépünkön, de ha nem akkor telepítsük fel őket. Persze ha csomagból telepítjük őket, akkor tegyük fel a hozzá tartozó *dev* és *lib* csomagokat is ha vannak! Ezenkívül fontos hogy a videokártya meghajtója jól legyen feltelepítve.

1. táblázat *A fordításhoz szükséges függőségek*

Python 2.	http://www.python.org
libjpe	http://www.ijg.org
libpng	http://www.libpng.org/pub/png/
zlib	http://www.gzip.org/zlib/
Scons	http://www.scons.org
CVS	http://ximbiot.com/cvs/cvshome/
Ode	http://opende.sourceforge.net/
Openal	http://www.openal.org/home/
sdl	http://www.libsdl.org/index.php
smpeg	http://www.loki-games.com/development/smpeg.php3
fmod	http://www.fmod.org/
Glut	http://www.opengl.org
Freetype2	http://www.freetype.org/

Ez az *nVidia* esetén azt jelenti, hogy a *64 bites* telepítés során ne telepítsük fel a 32 *bites* kompatibilitási fájlokat, mert zavart okozhatnak. Ha nem szeretnénk a fenti függőségekkel egyenként bajlódni, akkor használjuk a következő parancsok valamelyikét.

Gentoo operációs rendszer esetén:

```
$ emerge -pv blender
```

vagy *Debian* operációs rendszer esetén:

```
$ apt-get install blender
```

Ekkor megtudhatjuk, hogy milyen programok kellenek a telepítéshez, és a *Blendert* illetve a *Yafrayt* kihagyva könnyedén fel tudjuk őket telepíteni, vagy *Blender-el* együtt telepítjük az egészet, majd külön *uninstalláljuk* a *Blendert* és a *Yafrayt*. Szóval most már van egy fordításra kész operációs rendszerünk.

Beállítás és fordítás

Lépjünk be a *Blender* forráskódot tartalmazó mappába, ami a mi példánknál a következő:

```
$ cd ./blender
```

Mi a *Scons* fordítási metódust fogjuk használni, mert ez a leginkább támogatott. A *Scons*-nek a *Sconstruct* nevű fájl mondja meg, hogy milyen paraméterek mellett kell elvégeznie a fordítást. Ebben a fájlban az operációs rendszernek megfelelően egy hivatkozást találunk, ami a további speciális beállításokat tartalmazza. *Linux* esetében ez a fájlt a *./config* mappában *linux2-config.py* néven található meg. Mindkét fájl nagyon beszédes, ezért könnyedén ki lehet igazodni és bátran fel lehet fedezni az egyéb beállításokat is, mert én csak a fordítóra vonatkozókat fogom tárgyalni. Tehát nyissuk meg a *linux2-config.py* fájlt a kedvenc szövegszerkesztőnkkel és az alábbi sorokat módosítsuk a következőkre:

```
111 CFLAGS = ['-w', '-pipe','
➥-fPIC','-funsigned-char', '
➥-ffast-math', '-fno-strict
➥-aliasing', '-fomit-frame-
➥pointer', '-finline-functions']
112 CCFLAGS = ['-w', '-pipe','
➥-fPIC','-funsigned-char', '
➥-ffast-math', '-fno-strict-
➥aliasing', '-fomit-frame-
➥pointer', '-finline-
➥functions']
114 CPPFLAGS = ['-DXP_UNIX']
115 CXXFLAGS = ['-w', '
➥-pipe','-fPIC','-funsigned-
➥char', '-ffast-math', '-fno-
➥strict-aliasing', '-fomit-
➥frame-pointer', '-finline-
➥functions']
116 REL_CFLAGS = ['-march=k8',
➥'-O3']
117 REL_CCFLAGS = ['-march=k8',
➥'-O3']
```

Ha 2.41 vagy ennél korábbi verzióval próbálkozunk, akkor nem lesz *linux2-config.py* fájlunk, sőt *./config* mappánk sem, ezért mindezen beállításokat a *SConstruct* fájlban kell elvégezni a következő módon:

```
79 release_flags = ['-
march=k8', '-O3']
80 debug_flags = ['-O2', '-g']
81 extra_flags = ['-w', '-
pipe','-fPIC','-funsigned-
char', '-ffast-math', '-fno-
strict-aliasing', '-fomit-
frame-pointer', '-finline-
functions']
```

Lássuk, hogy mi mit jelent?
Az *-march=* opcióval lehet megmondani a *gcc*-nek, hogy milyen processzorunk van, milyen mikro-utasításkészlettel fordítsa a programot (*mmx*, *sse*, *see2*, *3dnow*, stb.).
Az egyenlőségjel után kell írni, hogy milyen processzorunk van. Lássunk néhány példát:

AMD64 = k8
Pentium4 = pentium4
AthlonXP = athlon-xp

Ha valakit érdekelnek a további processzortípusokra vonatkozó opciók, akkor látogasson el a ➲ http://www.freehackers.org/gentoo/gccflags/flag_gcc3.html oldalra, ahol további részleteket tudhat meg. Az *-OX* opcióval pedig meg lehet mondani, hogy milyen optimalizációs szinten akarjuk a binárist létrehozni. Lehetséges értékei: *X=0, X=1, X=2, X=3 és X=s*. Az *'s'* érték azt jelenti, hogy nem gyorsaságra igyekszik a fordító, hanem minél kisebb méretre. Értelem szerűen mi a *3*-as értékre pályázunk. A többi opció mind speciális optimalizációt jelent, amire nem térek ki.
Amikor ilyen durva optimalizációs paraméterezéssel dolgozunk, nem kell meglepődnünk azon, ha nem fordul le a program, vagy ha le is fordul tele lesz hibával Ezért mindig le kell tesztelnünk a programot az alap beállításokkal is, hogy biztos nem hozunk-e létre hibás kódot, amiből a későbbiekben problémánk lenne. Ha *CVS*-t használunk, akkor ez fokozottan igaz, mivel itt állandó a fejlesztés, csak a stabil kiadás ideje alatt, vagy kódfagyasztáskor stabil a kód. Ezért érdemes mindenképpen egy optimalizáció nélküli binárist létrehoznunk, azt elmenteni valahova, esetleg gyorsan leteszteltni, hogy amikor hibák jönnek elő könnyen szét tudjuk választani a program hibát az optimalizációból fakadó hibától. Lássunk erre egy példát. Én a következő hibát találtam a *CVS* kód lefordításakor:
Mind a két fordítás ugyan azokkal az optimalizációs paraméterekkel készült, de mégis jól látható a különbség a kettő között. Az *1. ábrán* fekete foltok jelentek meg az üveg majmon, míg a *2. ábrán* szép kék, olyan amilyennek lennie kell. Ilyen és ehhez hasonló apró vagy nagyobb hibákat kell keresnünk a tesztelés során. A fenti *scene* egyébként a hivatalos tesztcsomagban a *render/* mappa alatt található *refract_monkey.blend* néven. Ha készen vagyunk a *linux2-config.py* fájlt szerkesztésével, akkor mentsük el és kezdjük meg a fordítást. Ehhez adjuk ki a következő parancsot a *Blender* főkönyvtárán belül:

```
$ scons
```

1. ábra Blender 2.41 CVS 2006 február

2. ábra Blender 2.41 tar.gz

2. táblázat *A kétféle tesztkörnyezet*

Teszt környezet	CPU	RAM	VGA	Operációs rendszer	Blender forrás
1	AMD64 3500+	1 GB OCZ	ASUS 6600GT	Gentoo 2005.1 64 bit	CVS
2	Celeron 700 MHz	128 SDRAM	Geforce 2 Noname	Ubuntu 5.10 i386	Office

3. táblázat *A teszteredmények*

Beállítások	Teszt környezet	Idő/s		
		-O2 -pipe	-O3 full opti.	Gyorsulás/%
OSA 8	1	51,74	49,52	4,30
640x480	2	151,85	127,91	15,77
OSA 16	1	130,85	125,04	4,40
1024x768	2	759,37	635,31	16,34
OSA 16	1	517,23	495,39	4,20
2048x1536	2	2983,60	2491,88	16,48

Addig amíg a *Blender* fordul, lépjünk be a *Yafray* főkönyvtárába, majd az itt talált *linux-settings.py* nevű konfigurációs fájlt módosítsuk a következőképpen:

```
$ cd ../yafray

11 prefix = args.get
➥('prefix','/usr')
19 flags='-Wall -DHAVE_CONFIG_H
➥-D_PTHREADS'
20 if debug:
21          flags+=' -O3
➥-ffast-math -ggdb'
22 else:
23          flags+=' -O3
➥-march=k8 -w -pipe -fPIC
➥-funsigned-char -ffast-math
➥-fno-strict-aliasing -fomit-
➥frame-pointer -finline-
➥functions'
24 return flags
```

És kezdjük meg a *Yafray* fordítását is:

```
$ scons
$ su
$ scons install
```

Ez szintén elfog tartani egy darabig, addig végezzük el az ügyes bajos dolgainkat. A kedves olvasó bizonyára észrevette, hogy a *Blender-nél* nem adtuk ki a scons install parancsot, csak a *Yafray-nál*. Ez nem véletlen, a *Yafray* esetében ha nem adjuk ki az `install` parancsot is, akkor nem tudjuk elérni a *Blender-ből*! Ha netalántán hibát kapnánk, vagy egyszerűen csak új fordítást szeretnénk csinálni más paraméterekkel, akkor az új fordítás előtt adjuk ki a következő parancsot, hogy tiszta lappal indulhassunk:

```
$ scons clean
```

Ez a parancs a *Yafray* esetében nem működik, helyette kézzel kell csinálni a dolgot:

```
$ rm ./src/Yafraycore/*.os
```

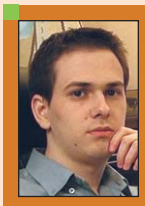
Ha szeretnénk a kész programot csomagolt formában megkapni, hogy könnyebb legyen a szállítása ill. telepítése, akkor adjuk ki a következő parancsot *Blender* esetén:

```
$ scons release
```

Ekkor a végeredményt nem a főkönyvtárban találjuk, hanem az összes szükséges fájlal együtt a *./dist* vagy újabban a *../install/linux2* könyvtáron belül. Így nem kell azon gondolkodnunk, hogy vajon milyen fájlok kellenek a *Blender* binárison kívül. Ha készen vagyunk a *Blender* fordítással, akkor tulajdonképpen futtathatjuk a programot bárhonnan, nem szükséges a $ scons install paranccsal telepíteni. Én csináltam egy *blender/* mappát a *home*/-on belül és ott futtatom. Ilyenkor kényelmesen futtathatunk akár több változatot is.

A tuningolás eredménye

Teszt fájl neve: *refract_monkey.blend*
A szokásos információkon kívül egyértelműen látszik a *3. táblázat* eredményeiből, hogy a sebességnövekedés a felbontás növelésével sem szűnik meg. Továbbá kisebb teljesítményű gépen a sebességnövekedés sokkal jelentősebb. Ez legrosszabb esetben is azt jelenti, hogy ha van egy 30 perces animációnk amit *PAL* szabványnak megfelelően 25 képkocka/másodperces és egy képkocka *renderelése* körülbelül 10 másodperc, akkor az animációnk körülbelül 5 órával előbb lesz kész. Talán a *3D CG*-re igaz a legjobban, hogy nincs az a teljesítmény, amivel az ember megelégedhetne. Remélem sikerült ezzel a módszerrel még egy kevés tartalékot kipréselni az otthoni gépekből.



Fábián Attila
(fabiana@elte.hu)
Az Eötvös Loránd Tudományegyetem Természettudományi karán vagyok vegyészhallgató. 4 éve használok Linuxot. Ha ezek után még mindig marad szabadidőm, akkor 3D grafikával foglalkozok vagy biciklizek.

KAPCSOLÓDÓ CÍMEK

A Blender webhelye:
➲ www.blender.org

A Yafray webhelye:
➲ www.yafray.org

Optimalizációs teszt:
➲ http://mail-index.netbsd.org/current-users/1995/12/11/0000.html

Optimalizációs paraméterek:
➲ http://www.delorie.com/gnu/docs/gcc/gcc_10.html

Tesztfájlok:
➲ http://download.blender.org/demo/test/test240.zip

Saját zenét saját kottából!

E cikkben megpróbálom megismertetni olvasóinkkal a zeneírás legfőbb lehetőségeit, mégpedig szabadon elérhető linuxos programokra támaszkodva.

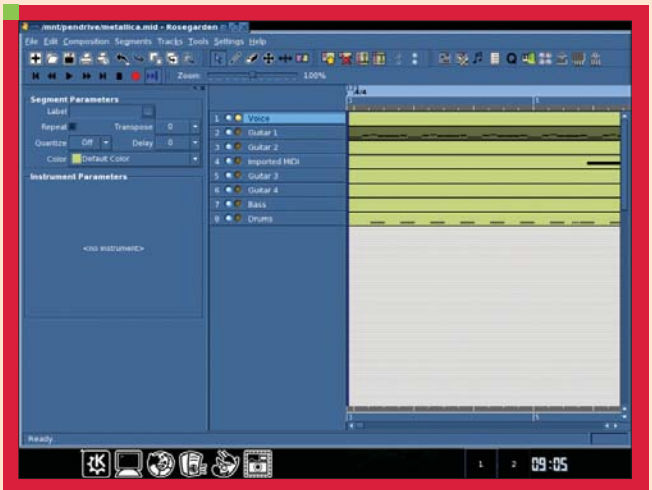
A bevezető után rögtön le kell szögezzem: sajnos én sem vagyok profi... Ha most kifejteném azt, hogy az utóbbi idők zenei „művészete” miképpen torzult sok helyen és rossz irányban, akkor csendesen tudomásul kellene vennem, hogy nem ítélkezhetem, hiszen én sem rendelkezem „muzikális vérrel”. Ilyen irányú képességeim kimerülnek néhány év énekkari szereplésben, a „hőskori” zeneszerkesztő programok aktív használatában, valamint a hazai rock zene feltétlen szeretetében. Ezzel a szerény háttérrel fogok röviden bemutatni egy népszerű *MIDI* alapú kottázót, valamint egy modulszerkesztésre született projektet. A cikk teljes megértéséhez minimális szintű előképzettséget fogok feltételezni, amivel a téma iránt érdeklődő olvasók minden bizonnyal rendelkeznek.

A MIDI leképezés

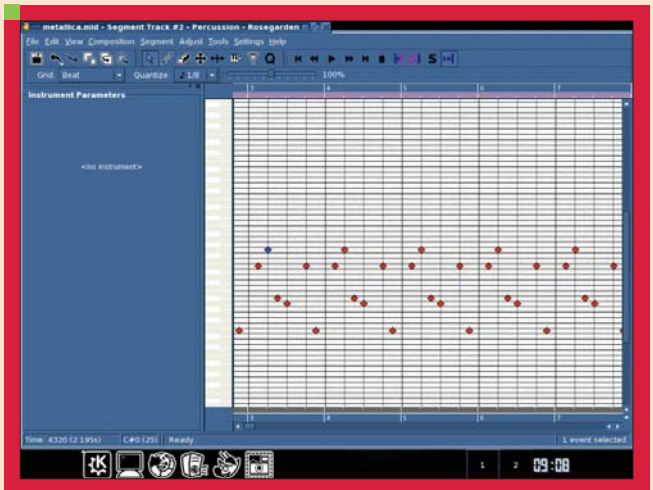
A ***.mid*** formátum a *MIDI* (***Musical Instruments Digital Interface***) szabványának legfőbb fájlkiterjesztése. A szabvány által rögzített zenei leíró állományok minden esetben szakszerű, nyers kottát jelentenek, melyeket a hangkeltő eszközünk saját hangszereiből fog megszólaltatni. Az említett kottákat több sávosra is készíthetjük, viszont ezek lejátszásakor (hivatalosan) annyi csatorna egyidejű szólama megengedett, amennyivel a hangkártyánk képes megbirkózni: ez a paraméter a hangeszköz ***polifónikus*** képessége. A *MIDI* hangkeltés tekintetében két típusú hangkártyát különböztethetünk meg: elsőként azokat, melyek frekvencia modulációs szintézissel, gépiesen képezik le a hangszereiket (két eltérő fázisú, szinuszhullám formájú jel egymásra hatásával), másodsorban pedig azokat a kártyákat, melyek rendelkeznek hullámtáblás szintetizátorral (valódi hangszerek speciális formátumban digitalizált hangkészletével). Számunkra értelemszerűen az utóbbi verzió jelent nagyobb értéket.

A szerkesztés linuxos lehetősége

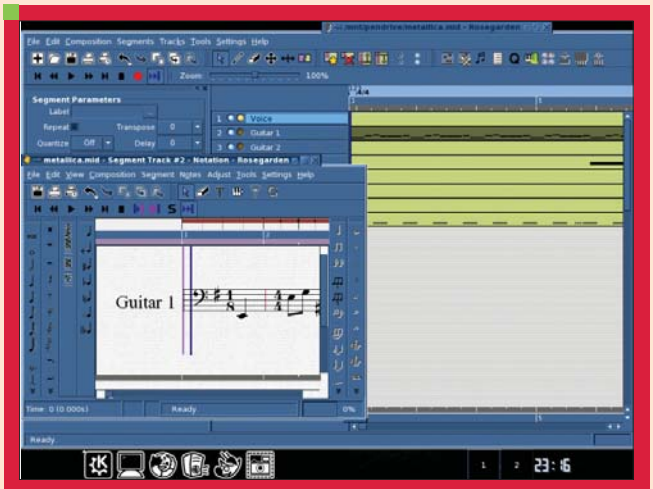
A *Linux* platform számos *MIDI* alapú kottaíró és szerkesztő programmal rendelkezik. Ezek közül választottam ki azt, amely meglátásom szerint a potenciális felhasználó igényeit leginkább szolgálja: a kiszemelt projektet ***Rosegardennek*** hívják. Hivatalos lapja a ➲ ***http://www.sourceforge.net/projects/rosegarden url*** mögött található, a cikk írásakor fellelhető legfrissebb verziót ***R4 v1.2.3*** azonosítóval jelöli fejlesztője, mely gyakorlatilag minden ***„Unix-***szerű” operációs rendszeren életre hívható. Töltsük le a forrás archívot, csomagoljuk ki, és építsük fel ezt a remeket! A fordításhoz szükség lesz a ***scons*** nevű ***compiler*** jelenlétére, mivel ebben az esetben nem használható a megszokott ***make*** parancs.
A ➲ ***http://scons.sourceforge.net*** oldalról beszerezett ***scons*** csomagot annak *README* állománya szerint állítsuk üzembe (az előre definiált ***setup.py*** telepítő szkriptjének segítségével).



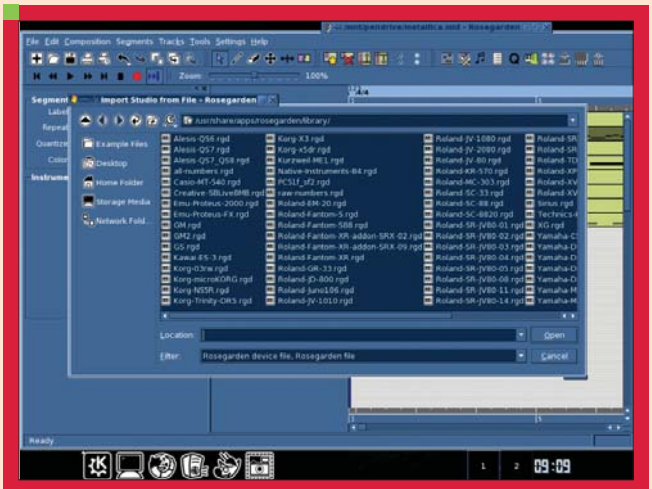
1. ábra Szerkesztésre nyitott, többsávos Metallica kotta



2. ábra A Rosegarden mátrix editora



3. ábra A „valódi" kottaszerkesztő



4. ábra Íme, az importálható stúdiók

Miután a feltelepítettük e fontos, fordításhoz szükséges függést, a ***Rosegarden*** forrásmappájába állva adjuk ki ***root***-ként a

```
scons configure, scons, scons
➥install
```

parancsokat, ami által létrejön az áhított bináris, és az alapértelmezett helyére kerül minden szükséges állomány.
A programot felhasználóként terminálra gépelt

```
rosegarden
```

paranccsal indíthatjuk el. Az ide kapcsolódó képeket mindenképpen érdemes szemügyre venni: a program felhasználói felülete egyszerre barátságos és összetett, kifizetődőnek tűnik időt szakítani megismerésére. Látszólagos bonyolultsága ne tántorítson el senkit sem, hiszen a kezdeti „szárnypróbálgatások" elégségesek az alapszintű ismeretéhez. Rutinszerű használatához azért mindenképpen javallott figyelmesen végigolvasni a honlapon linkelt oktató anyagokat is.
A tudnivalók a ***tutorial*** link mögött találhatóak, ezek „gyors információi" szerint a lerakott mintákon dupla egérklikket használva, a felbukkanó ablakban egységnyi területen belül szerkeszthetünk dallamokat, az eszköztár gombjainak segítségével.
A ***Rosegarden*** természetesen a zenei „trükkök" java részét ismeri: a hajlításokat, a lecsengéseket, a borítékolást „csípőből" kezeli.

Mint az sejthető, a kottázás terén ismernünk kell a zenei jelölőket, ennek ellenére a projekt rendelkezik mátrix editorral is, így a segítségül hívott virtuális zongorájának billentyűihez viszonyítva is lehetséges hangjegyeket „leszúrni". A mellékelt ábrákon ezen felül megtekinthetőek a használható „stúdiók", valamint a kottaszerkesztő is. A ***Rosegarden*** képes importálni és exportálni több formátumba (akár ***Music XML***-be is), viszont képességeinek kompromisszumoktól mentes kiaknázásához természetesen működő szekvenszer eszköz szükséges. A „***File***" menüben található „***Print***" menüponttal valódi kottákat nyomtathatunk, minek hála látványos és hiteles kottafüzettel jelenhetünk meg, bárhová vessen is a sors. Az „öreg" ***MIDI*** létjogosultságát pedig mi sem jelzi jobban, minthogy a példaként megnyitott, világhálóról letöltött ***Metallica*** állomány nem haladja meg a ***20KByte*** méretet, amit csatornánként, hangszerenként szeparálva lehet szerkeszteni. Sőt! Akár az egész hangszerkészletet kicserélhetem „alatta" ***Roland***, vagy éppen ***Yamaha*** stúdióra, ha úgy tetszik.

A zenei modulok

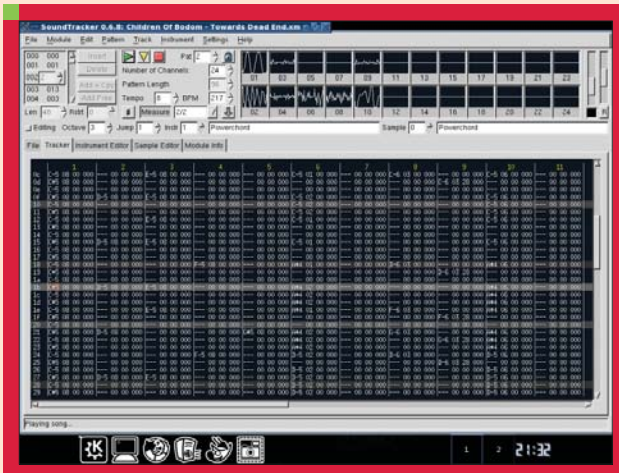
A ***MOD*** formátumok ***MIDI*** állományokkal ellentétben magukban foglalják a skálázandó hangszerek hangmintáit is, valamint egy speciális táblázatban rögzítik le, hogy az adott csatornán, adott időben mely hangszer, milyen programspecifikus effektekkel szólaljon meg.

A legjellemzőbb zenei modulok fájlformái: ****.mod***, ****.s3m***, ****.xm***. A formátum szerkesztési bonyolultsága nagyjából olyan, mint a ***MIDI*** kapcsán említett folyamat, de attól azért több ponton is jelentősen eltér. Erős túlzással talán azt mondhatom, hogy a ***MOD***-ok megalkotása némely (komolyabb) mobiltelefon zeneszerkesztő programjának használatához hasonlít leginkább. A „legjobb" program kiválasztásával kemény fába vágtam a fejszémet, mivel a potenciális jelöltjeim specifikus megoldásokkal, egyéni effektekkel és igen eltérő felhasználói értékkel rendelkeztek.

A SoundTracker

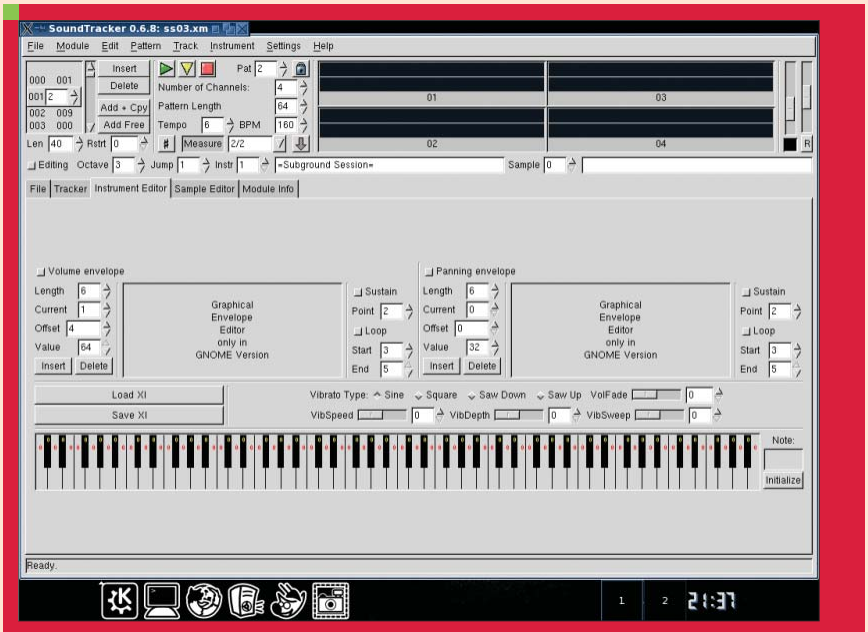
Végül a ***SoundTracker*** programra esett a választásom. A projekt honlapja a ➲ ***http://www.soundtracker.org*** címen található, ahol a forráskód és a kész bináris egyaránt elérhető, a cikk írásakor fellelhető legfrissebb verziót ***v0.6.8*** azonosítóval jelöli fejlesztője. Ebben az esetben is a forráskód felépítésében látom a biztos megoldást: a nem egészen ***900 KByte*** méretű csomag letöltése után, a kibontott archívban adjuk ki ***root***-ként a szokásos `./configure`, `make`, `make install` parancsokat. A ***SoundTracker*** kezelőpanele ***GTK*** könyvtárra alapoz, így érdemes ezt naprakészen tartanunk. A lefordított program felhasználóként terminálra gépelt `soundtracker` paranccsal indítható el. A jól megszokott ***tracker*** programok hagyományait folytatva, átlátható és elegáns felülettel fogadja a leendő művészeket.

5. ábra A SoundTracker felülete



6. ábra Egy Children of Bodom xm modul

A szerkesztő mindössze a **.xm* és **.mod* fáljokat képes kezelni, ezt viszont kétségkívül profi módon teszi. Továbbá néhány kattintás árán képes *XI* típusú hangmintákat használatba venni, de akár egy megnyitott modulból is kiemelhetjük, exportálhatjuk bármely szükséges hangszert. Egy *pattern* területe (a modulszerkesztőkben megszokott) 64 ütemre korlátozódik, melynek időbeli hosszúságát a *tempo* értékkel lehet megszabni. Alapértelmezés szerint 8 sávon alkothatunk, azonban ez a szám tetszés szerint változtatható, akár minden egyes *pattern* esetén. Ebben az esetben is érdemes szemügyre venni a mellékelt képeket: még a laikusoknak is feltűnhet, hogy a végtelenül letisztult felület nagyon leegyszerűsíti az elsőre bonyolultak tűnő munkákat. Amennyiben bármely modulban (akár új zenéről, akár megnyitottról legyen szó) változásokat szeretnénk eszközölni, úgy a panel bal oldalán található *Editing* kapcsolót mindenképpen nyomjuk be, mielőtt az adott ütemen leütnénk bármely hanghoz tartozó billentyűt. A *SoundTracker* (kézen fekvő módon) képes közvetlenül **.wav* formában is renderelni (viszont ez a formátum már nem szerkeszthető tovább).
A kész **.xm* „termék” visszajátszása nem csak e program által lehetséges, hiszen a népszerű *XMMS* lejátszó rengeteg zenei modult (megközelítőleg húsz típust) képes kezelni.



7. ábra A SoundTracker hangminta szerkesztője

Az ehhez szükséges *MikMOD plugin* a ➲ *http://modplug-xmms.sourceforge.net/* címen érhető el, bár a rendszerünkben lévő lejátszó tudása minden valószínűség szerint már „gyárilag” bővítve van a *libmikmod.so* állománnyal.

Összegzés

A számítógéppel támogatott, hobbi jellegű zeneszerkesztés igen nagy hagyományokkal rendelkezik, ennek ellenére ma már szinte csak a *scene* partik (melyek „gyűjtőhelye” a ➲ *http://www.scene.org* címen található) megmérettetéseként tényknek az érintettek. Így őszintén remélem, hogy ez a rövid cikk megtalálja közönségét: nem titkolt célom próbálkozásra buzdítani minden tehetséges (és kevésbé tehetséges) érdeklődőt. Apropó, a *scene* versenyek ügyén majdnem elfelejtettem megemlíteni, mekkora „buli” végighallgatni a sikeresebb pályaműveket! Némelyikük (kategóriájának megfelelően) „süketen” lett komponálva, tehát alkotója a munka során nem hallhatta, mit is csinál valójában, csupán a végeredmény formájában tapasztalhatta meg művét. Hiába, némely ember zenei érzéke hihetetlen és mellbevágó muzikális teljesítményre képes. Minden érdeklődőnek tartalmas kikapcsolódást kívánok!

Kovács Zsolt (kovi@linuxforum.hu)

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

C64 – Három betű, és a „fél világ”

Vajon akad-e bárki az olvasók között, akinek ne csengene ismerősen a Commodore International neve? Kötve hiszem! Ezzel a névvel vonult ugyanis a történelembe a megfizethető árú számítástechnika legjelentősebb úttörője, miközben „oroszlánrészt” vállalt a nyolcvanas évek IT áttörésében. Számtalan sikeres fejlesztés fűződik e névhez, melyek közül néhány konstrukció ma már kultusz-számítógépként pihen a virtuális, és a valós szakmai múzeumok becses darabjai között.

A Commodore rövid története

A cég története egészen ***1968***-ig nyúlik vissza, történetesen ekkor alapították meg ***Kanadában***. Kezdetben elektronikus vezérlőpanelek tervezésével és gyártásával foglalkoztak, csak később nőtték ki magukat a kor meghatározó informatikai vállalatává. Hírnevüket javarészt ***Jack Tramiel*** vezetésével alapozták meg (az említett úriember sorsa példaértékű: a lengyel származású elektromos szerelőt egyéni szervizvállalkozásából szerződtette le a nagy nevű cég, ahol meg sem állt a vezérigazgatói székig).

A hetvenes évek elején kezdtek számológépeket gyártani, majd ***1977***-ben dobták piacra első személyi számítógépüket, ***PET*** néven (érdekes, hogy a gép tervező mérnöke, ***Chuck Pedle*** később a ***Motorola*** hajtóerejeként kamatoztatta tehetségét). A masina szokatlan módon rendelkezett beépített monitorral is, de „fő dobásként” tervezésekor egyaránt szem előtt tartották az otthoni használhatóságot és az alacsony előállítási költséget is.

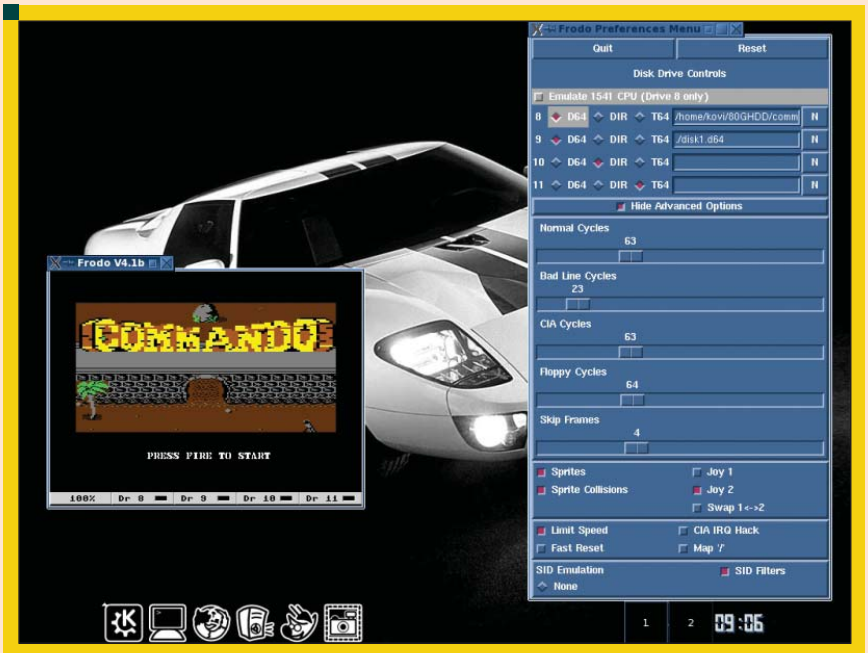
A ***PET***-nek köszönhető műszaki hírnév birtokában kiteljesedő sikertörténet nagy számban szülte a jobbnál jobb gépeket. Sikeres szériái után a cég máig rejtélyes okok miatt megtorpant, és csődközeli helyzetében az amerikai kormány anyagi segítségére szorult. A tőkeinjekciót felhasználva megvásárolta a legendás, ***16 bites AMIGA*** gép terveit, mely ugyan visszaadta az elveszített ***Commodore*** hírnevet, de csodát már az új számítógépek sem tudtak tenni. A belső viszályok miatt időközben menesztett ***Tramiel*** szerepét senki sem tudta hatékonyan betölteni, így a befektetők sorban intettek búcsút, majd a szerencsétlen időben kiteljesedő ***PC***-k térhódítása megadta a „kegyelemdöfést”.

Hiába próbáltak már saját ***PC***-kel operálni, hiába próbáltak a ***16 bites*** gépek terén újabb és jövőbe mutató ***AMIGA*** szériáknak piacot teremteni, az ironikus vég elkerülhetetlen volt: a legendából „csődtömeggé” változott nagynevű vállalat ***1995***-ben eladásra került.

1. ábra C64: a Commodore egyik legnagyobb dobása

A C64-ról dióhéjban

A ***VIC-20*** konstrukcióból fejlesztett ***C64*** a ***Commodore*** nagyágyúja volt: komoly tervezői teljesítmény eredményeként a számítógép igen széles felhasználási területen bizonyíthatta képességeit. A két jellegzetes formában kivitelezett masina gyakorlatilag ugyanolyan gyakran volt fellelhető a cégek gépparkjában, mint



2. ábra A Frodo működése KDE asztalon

a „modern” gondolkodású családok otthoni használatában, így életének derekán a vállalat bevételének kétharmadát biztosította.
Nem mellékesen, saját korában a *C64* rendelkezett a legtöbb játékprogrammal, melyek az alábbi *architektúrára* támaszkodhattak: *6510A CPU* (*1.02 MHz* üzemi órajelen), *64KByte RAM*, *20KByte ROM* (beégetett *BASIC* értelmezővel), *6581 SID* hangcsip. Az említett hardver környezet elérhetővé tette a *320 X 200 X 4bit* képfelbontást (*sprite* kezeléssel), valamint három hang egyidejű generálását is.
A külső port szerepét leginkább a soros kapu jelentette (ez később, a *floppy* meghajtók idején tett jó szolgálatot), de egyaránt rendelkezett kazettás magnó csatlakozási lehetőséggel, *Cartridge* porttal, *Joystick* illesztővel, *RCA* videókimenettel is. Később a fanatikusok munkája által született a géphez *IDE HDD* csatoló is, illetőleg nagy méretű (akár *1-2MByte*) memóriabővítő egység.

Linuxon emulált környezet

Mivel a muzeális *C64* még mai is rendkívül komoly rajongói táborral bír, így a működését utánzó emulátorok nagy számban érhetőek el a világhálón. Emellett a *retro* jellegű játékokat gyűjtő fanatikus csapatokból is akad jó néhány: a nosztalgiázásnak tehát semmilyen akadálya sincs.

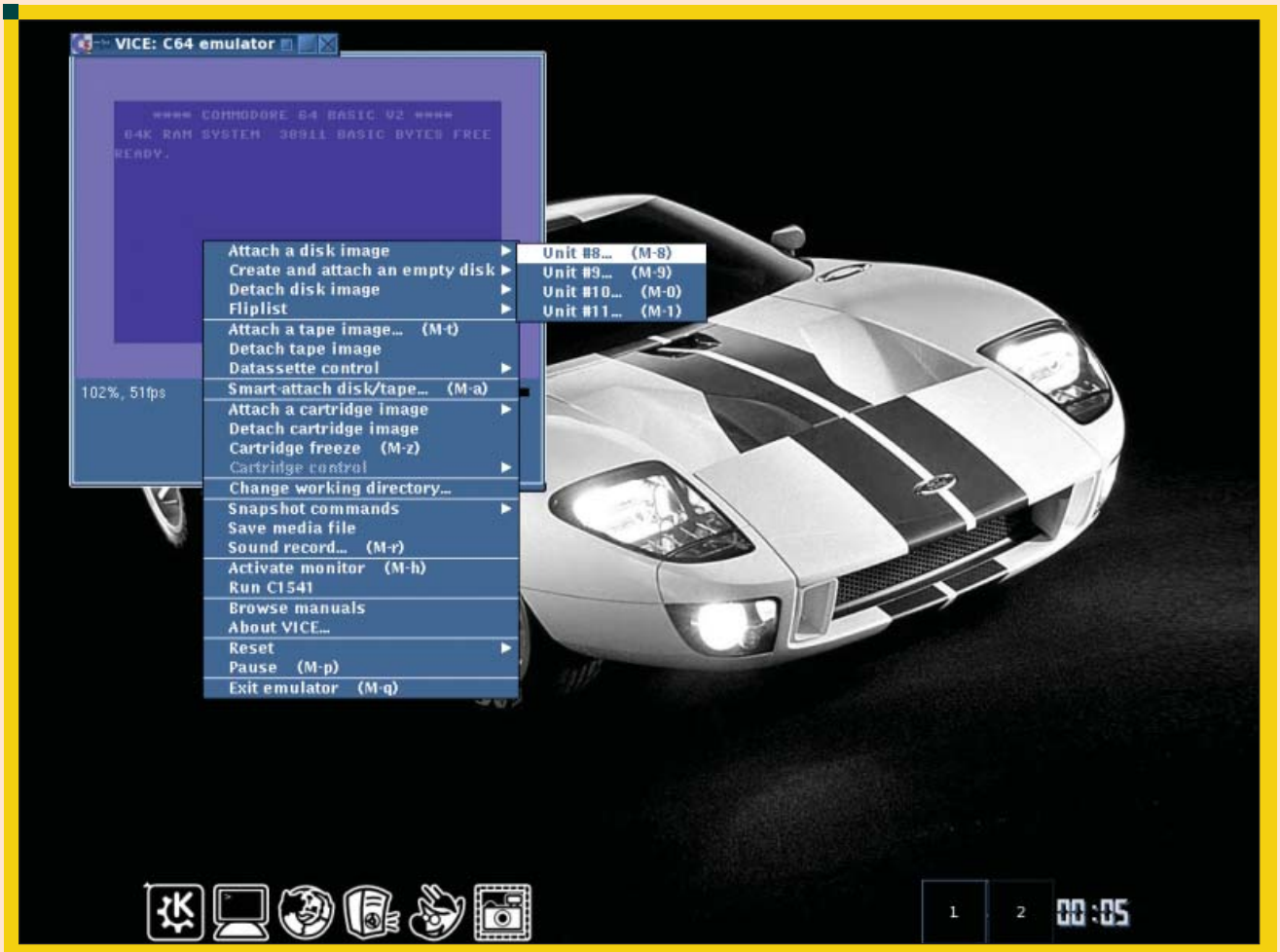
Platformtól független emulátorokat keresve az olvasó valószínűleg össze fog futni a szabad forrású *VICE* projekten keresztül akár kereskedelmi jellegű emulátorokkal is.
Most a két legnagyobb tudású, kiforrott és egyszerűen kezelhető programot fogom bemutatni, melyek gyakorlatilag minden mérhető felhasználói értékű rendszeren életre hívhatóak. Nagy (el)ismertségük nem meglepő, hiszen hatékony és paraméterezhető kódokról van szó, melyek használatát grafikus menük segítik. Természetesen mindkettő projekt hibátlanul kezeli a *SID* processzor által keltendő hangokat, zenéket is.

Frodo

A program *Christian Bauer* nevéhez fűződik, hivatalos lapja a ➲ *http://frodo.cebix.net* címen érhető el, ahol a dokumentációk mellett természetesen a *Frodo GPL* forráskódjához is hozzáférhetünk. Csomagoljuk ki a letöltött forrás *tarball*-t, és lépjünk a kibontott mappa */Src* útjára! Adjuk ki *root*-ként a `./configure, make` parancsokat, minek hatására felépül a végrehajtható bináris fájl, `frodo` néven.
Mivel a forráskód `makefile` állománya (szándékosan) nem értelmezi az `install` paramétert, így az eddig használt forrásmappában lépjünk vissza egy szintet az */Scr* útról. A forráskód gyökérben létrejött futtatható állományokat az ott lévő összes fájl társaságában mozgassuk át egy mindenki által olvasható területre, az új helyre emelt *Frodo* binárist pedig egy megfelelő szimbolikus láncon keresztül tehetjük elérhetővé. (Utóbbi elhelyezésére például az */usr/local/bin* mappa tökéletesen megfelel.)
Ha végeztünk a műveletekkel, a link felhasználói indításával hívható életre a lefordított projekt. A *C64* jellegzetes, kék üdvözlő képernyője mellett felbukkanó grafikus menüben azonnal kapcsoljuk be az „A*dvanced options*” lehetőséget, majd a bővebb panelon a „*Limit speed*” funkciót. Ha elkészültünk, akkor a „*Drive*” szakaszban adjuk meg, mi legyen a virtuális magnónkban (általában **.t64* kiterjesztésű állományok), illetve virtuális lemezmeghajtónkban (általában **.d64* lemezképek).
Kattintsunk vissza az emulátor „munkaablakára”, és töltsük be a kiválasztott programunkat! Virtuális magnóról ezt a `load "*"` utasítással, *floppy* meghajtóról pedig `load "*",8,1` paranccsal tudjuk megtenni (a hajlékonylemez esetén a nyolcas szám az utánzott meghajtó eszközazonosítója!). Ha a kívánt lenyomat betöltődött, adjuk ki a `run` parancsot! Fontos lehet, hogy a *Frodo* indítása előtt érdemes angol billentyűkiosztásra váltani: tapasztalatom szerint „**”* karakter csak ekkor érhető el (mégpedig az *„ú”* billentyűt használva).

VICE

A *VICE* programkódja elsősorban *Andreas Boose* munkáját dícséri.
Ez a projekt gyakorlatilag a *PC* szegmens vezető *C64* emulátora, mely a *Frodo*-hoz hasonlóan platformfüggetlen megvalósítás, *GPL* licenszű forráskódja pedig szintén elérhető honlapján. Látogassunk el a *VICE* csapatának hivatalos honlapjára, a ➲ *http://www.viceteam.org* címre, ahonnan töltsük le a forrás *tarball*-t. Miután kibontottuk az archívot, adjuk ki benne (*root*-ként) a szokásos `./configure`, `make`, `make install` parancsokat, minek hatására felépül és települ az emulátor binárisa.
Virtuális gépünket `x64` paranccsal lehetséges elindítani, amit felhasználóként kell kiadnunk egy *X terminálon. A* felbukkanó kék képernyőn jobb, illetve bal egérgombot használva



3. ábra A VICE felülete

tudjuk előhívni a helyi menüket, melyekben finombeállításokat eszközölhetünk, illetve a lemez/kazetta lenyomatokat tölthetjük be. Ezekről most nem szólnék bővebben, mivel a beállítások alapértéke tökéletesen megfelel a legtöbb ***Linux*** rendszeren, a kívánt lenyomatok pedig igen egyszerűen használatba vehetőek. Érdekesség, hogy a ***VICE*** menüiben található *„About”* részben egy meglehetősen magyarul csengő, ***Bizcó Tibor*** név is olvasható, a fő fejlesztők között megnevezve...

A program további, kellemes meglepetésként képes a későbbi ***C128*** széria utánzására is, valamint akár kizárólag csak ***SID*** emulátorként működni. Előbbi funkciót egy terminálra gépelt `x128`, utóbbi lehetőséget pedig `vsid` paranccsal tudjuk életre hívni.

Hol vannak a programok?

Azért, hogy az emulátorokat önfeledt, nosztalgikus kikapcsolódás céljára tudjuk használni, érdemes beszerezni a hozzá tartozó programok és játékok állományait is. Én mindenképpen egy (nemzetközi szinten is) kiemelkedő hazai oldalt ajánlok erre a célra, mely a ➲ ***http://web.externet.hu/sk/c64 URL*** mögött érhető el: itt gyakorlatilag minden megtalálható, amit egy ***retro*** fanatikus kívánhat.

Összefoglalás

A cikk írásakor rövid ideig értetlenül álltam egy (egyébként ismerős) érzés előtt, mely igen sokszor kerít hatalmába, régi számítógépek kapcsán. Arról van szó, hogy a letölthető, nagy nevű játékprogramok (***Commando***, ***Dizzy***, ***Inter Karate***, ***Last Ninja*** stb.) még mindig megdöbbentenek, holott fiatal koromban „nap mint nap” használtam ***C64***-et: jól ismerem „kívül-belül”, egy példányt féltve őrzök a gyűjteményemben, néha ma is bekapcsolom...

Miért rökönyödök meg mégis, újra és újra? Talán azért, mert ezek a programok még évek múlva is úgy kötnek a képernyő elé, hogy némelyikük mérete még a ***30 Kbyte***-ot sem éri el, ráadásul fejlesztésük során mindegyikük nélkülözte a (ma oly természetes) zenei modulszerkesztők, fejlesztői segédletek, és az ***Internet*** adta lehetőségeket. Ezeknek az apró kódoknak bizony komoly tisztelet jár, hiszen manapság csupán egy rendszerállomány az előbb írt méretnek a többszöröse. Ezt a „slusszpoént” figyelembe véve úgy gondolom, hogy nem hiába uralta a ***C64*** a „fél világot”, a programozók nem hiába alkották meg remekműveiket: a gép még mindig él.

Mindenkinek tartalmas kikapcsolódást, és kellemes nosztalgiázást kívánok.

Kovács Zsolt (kovi@linuxforum.hu)

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