Hírek

Az első 3.5G-s linuxos mobiltelefon

Japánban már kapható a világ első 3.5G-s linuxos mobiltelefonja, melyet a *NEC* mutatott be. A *N902iX* 3.6 megabites sebességre is képes letöltéskor. Az eszköz a megszokott méretekkel bír, azonban tömege 133 gramm, ami a mai készülékek mellett nem mondható túl könnyűnek. A telefon lelke egy Intel XScale processzor.

Kijelzőből kettőt találhatunk, amint az egy összecsukható telefontól manapság elvárható. A külső 65 ezer színt képes megjeleníteni 120x90 képpontos felbontás mellett, míg a belső kijelző 262 ezer színt tud 345x240 képpontos felbontás mellett.

Kamerából is kettőt kapott a készülék. A belsőt elsősorban videótelefonáláshoz ajánlják, hiszen felbontása csupán 0.3 megapixel, míg a külső 4 megapixeles. Ezt képstabilizátorral is ellátták, ami kevés fény mellett bizonyulhat hasznosnak.

A telefon ujjlenyomat, hang vagy arc alapján azonosítja a tulajdonost. *Mini SD* kártyával bővíthető, azonban ennek hiánya azonnal érezhető, hiszen ez esetben csupán 4 megabájt tárhellyel gazdálkodhat a felhasználó.

A készülék készenléti ideje 440-560 óra, a beszélgetési idő egy töltéssel pedig 100-150 perc.

➲ http://www.linuxdevices.com/news/NS3110082615.html

Hálózatbiztonság tesztelése zsebből

A *Miami* székhelyű *Immunity* biztonságtechnikai cég októberben kezdi forgalmazni az *USA*-ban *Silica* névre keresztelt vezetéknélküli eszközét, mely elsősorban *WIFI* és *Bluetooth* kapcsolatokat képes tesztelni – de *USB*-vel vezetékes hálózatra is csatlakoztatható –, és jellemző beállítási hibákat kiszűrni. A szerkezettel több, mint 150 gyakori exploit tesztelhető, hogy azokon keresztül sebezhető-e az adott hálózat. Az eszköz előreláthatóan 3 ezer dollár körüli áron kerül a boltokba.

➲ http://www.eweek.com/article2/0,1895,2003853,00.asp

Vezetékes Skype

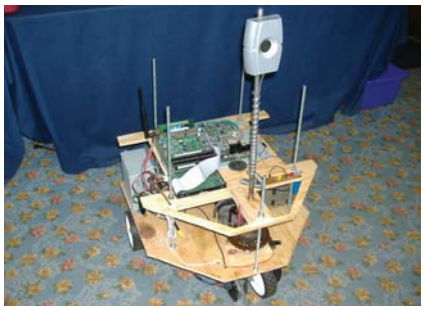
Eddig a *Skype*-ot csak számítógépről vagy *WIFI*-s telefonról lehetett elérni, azonban az új *Philips VOIP 841* lehetővé teszi, hogy ezek nélkül is telefonálhassunk. Az eszközön található egy *RJ-11*-es aljzat a hagyományos telefonhálózathoz és egy *RJ-45*-ös aljzat a szélessávú kapcsolathoz. A készülék *DECT* rendszerű (vezeték nélküli, de nem *WIFI*-s), biztosítva a szabad mozgást bázisállomás körzetében.

A készülék ára várhatóan 150 dollár körüli lesz.

➲ http://www.pcmag.com/article2/0,1895,2010681,00.asp

Linuxos robotok házilag

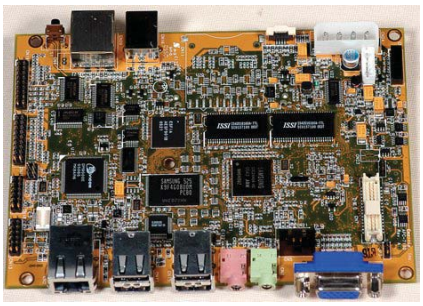
Michael Surran – az amerikai *Greater Houlton Christian Academy* egyik tanára – rendhagyó ötlettel állt elő: építsenek linuxos robotokat. Az első robot egy kutya volt, külseje pedig a népszerű angol science fiction – a *Doctor Who* – alapján készült. A rendelkezésre álló költségvetés minimális volt, így esett a választás a Linuxra és központi egységnek is közönséges asztali számítógépet használtak fel. Természetesen sok más költségtakarékos megoldást is felhasználtak, így például a távolságmérést egy hétköznapi



egérrel oldották, melyet a kerék mellé rögzítettek, így számolták ki annak elmozdulását. A robotok programozását *Python* segítségével végzik a diákok. A projekt elindításának fő célja a robotika népszerűsítése.

➲ http://www.linuxjournal.com/article/9103

Olcsó Linuxos gép ARM processzorral



A taiwani székhelyű ***Embedian*** bemutatta legújabb ***ARM*** alapú számítógépét, az ***EBC-7000***-et. A rendszert mind különálló gépnek, mind egy összetettebb rendszer vezérlésére ajánlják. Az eszköz 400 MHz-es processzora és 128 megabájt memóriája ellenére mindössze 2 wattot fogyaszt.
A kommunikációról egy 100 megabites ***Ethernet*** port és négy ***USB*** kapu gondoskodik, illetve adattárolásra merevlemezt vagy ***Compact Flash*** kártyát használhatunk.

Az eszközt ***Debiannal*** szállítják, és ***X11***, ***Qt***, illetve ***KDE*** támogatás se maradhatott ki. A gyártó minden tőle telhetőt megtesz, hogy az eszköz forráskód szinten kompatibilis legyen az ***x86*** architektúrával, megkönnyítendő az alkalmazások portolását.

➲ http://www.linuxdevices.com/news/NS2120023669.html

Kétmagos 64 bites Intel processzor – már laptopokba is

Kiadta az ***Intel*** legújabb, 64 bites, kétmagos, ***Merom*** kódnevű laptopokba szánt processzorát.
A korábbi, kétmagos ***Yonah*** kódnevű processzorához képest húsz százalékos teljesítménytöbbletet ígér anélkül, hogy több energiát igényelne. Az ***Intel*** jelen lépését az ***AMD*** 64 bites sikereivel indokolta.
A ***Merom*** a meglévő lapkakészletet használja: ***Intel 945 Express*** és ***Intel Pro/wireless 3945ABG***.

➲ http://www.eetimes.com/showArticle.jhtml?articleID=192300808

Három terabyte 80 wattból?

A ***Capricorn Technologies*** legújabb terméke, a ***GB3000***, minden bizonnyal forradalmasítani fogja az adattárolást. A ***GB3000*** nem más, mint egy ***1U*** (44,45 mm) magas rackbe szerelhető eszköz, mely három terabyte tárterületet ad (4 darab 750 gigabájtos merevlemez) mindössze 80 wattos fogyasztás mellett. Az eszközt 1 GHz-es processzor vezérli és maximum 1 gigabájt memória kerülhet a ***VIA*** alaplapra.
Több eszköz akár ***40U*** magas rack szekrénybe is szervezhető, így akár 120 terabyte tároló területünk is lehet mindössze 3,2 KW-os fogyasztás mellett. Az eszközök egymással és a külvilággal 10/100/1000 megabites hálózati csatolón kommunikálnak.

➲ http://www.capricorn-tech.com/gb3000.html

➲ http://www.capricorn-tech.com/tb120.html

Megállapodás a Google Talk és a Skype között

A két óriáscég megállapodott a rendszerük közötti átjárhatóságról, igaz egyelőre csak az írásbeli szolgáltatások esetben (szöveg üzenetek, jelenlét jelzés). A rendszerek teljes összekapcsolása 2007 végén zárul, amikortól a ***Skype*** és a ***Google Talk*** felhasználók közvetlenül is tudják egymást hívni.

➲ http://www.pcmag.com/article2/0,1895,2009389,00.asp

Korlátozott szériájú 16 gigabájtos pendrive

A ***Toshiba*** hamarosan piacra dobja korlátozott mennyiségben 16 gigabájtos pendriveját, mely 8 centi hosszú, 2 centi céles és 8 mm vastag, – ezzel ugyan nem pályázhat a legkisebb címre – azonban tömege mindössze 12 gramm. Az ára egyelőre nem ismert.

➲ http://www.techworld.com/storage/news/index.cfm?newsID=6755

Kézi terminálok

Az *AML* bemutatta két kézi adatterminálját – az *M5900*-at, illetve a strapabíróbb *M5900i*-t –, melyet elsősorban ipari felhasználásra szánnak, mint például: raktárkészletezés, ár ellenőrzés, szállítmányozás, stb.
Az eszközök más eszközökkel (asztali számítógép, pénztárgép) *USB* és *RS-232*-es soros portokon képesek kommunikálni. Az eszközök rendelkeznek vonalkódolvasóval, melynek típusát (lézer, *CCD*) és hatósugarát (közeli, távoli) a megrendelő választhatja meg. Az eszköz működtetésért a 200 MHz-es *ARM* processzor, a 32 megabájt memória és a 16 megabájt flashmemória felel. A 16 megabájtból 10-re telepíthet a felhasználó programokat. A kijelzője fekete-fehér 160x160 képpontos, mely 20x20 karaktert jeleníthet meg. Az adatbevitelt a már említett vonalkód olvasó mellett egy 55 gombos mini billentyűzet segíti.
Az eszközök ára 800 és 1400 dollár között változik a felszereltség függvényében.

➲ http://www.linuxdevices.com/articles/AT5552849613.html

16 gigabájt egyetlen memória modulban

A *Micron Technolgy* kifejlesztette jelenlegi legnagyobb kapacitású memóriamodul, mely 16 gigabájtos kapacitással bír és elsősorban szerverekbe szánják. A 78 nanométeres csíkszélességgel készített memóriamodul *DDR2*-es lesz és várhatóan az év végére kerül a boltokba.

➲ http://www.eetimes.com/showArticle.jhtml?articleID=192500092

Linux az ausztrál Nemzeti Bankban

Az *Ausztrál Nemzeti Bank Linuxra* váltotta *Solarisos* kiszolgálóit.
Az új kiszolgáló griden (10 darab 4 utas *Intel Itanium* szerver) *Red Hat Linux* és *Oracle 10g* fut, mely a 11 terabájt adat biztonságos tárolásáért és napi 30 gigabájt adat feldolgozásáért is felelős. A bank ezzel a költséghatékonyságon kívánt javítani anélkül, hogy a megbízhatóság csökkenne.

➲ http://www.computerworld.com.au/index.php?id=580233128

Arcos médialejátszó Linuxszal

Már kapható a *WIFI* nélküli *Archos 604*, mely a 4.3 hüvelykes (480x272 képpontos felbontás) kijelzőjével, a beépített 30 gigabájtos merevlemezével, illetve *USB 2*-es kapcsolatával ideális szórakoztató eszköz hosszú utakra, míg a tévékimenettel kiváltható akár az otthoni *DivX* lejátszónkat is. Az eszköz *MPEG4*-et, *WMV*-t, *MP3*-at, *WMA*-t tud lejátszani, illetve *JPEG*-et, *BMP*-t, *PNG*-t és *PDF*-et megjeleníteni. (A lista a megfelelő bővítményekkel persze kiterjeszthető.)

A beépített akkumulátornak köszönhetően egy töltéssel 5 órát videózhatunk rajta vagy 16 órán keresztül képes zenét lejátszani. Jelenleg 480 dollárért vagy 380 euróért juthatunk az eszközhöz. A hamarosan piacra kerülő *Archos 604* utódja már *WIFI*-vel és nagyobb felbontású kijelzővel kapható majd, azonban ennek ára még nem ismert.

➲ *http*://www.linuxdevices.com/news/NS7079516466.html

A Kínai Nagy Tűzfal árnyékában

Kínában törvénytelennek számít, ha valaki saját email-kiszolgálót üzemeltet, kivéve persze, ha az illető előbb szerez engedélyt. Nem ez az első olyan lépés, amely Kíne polgárainak szabadságjogait korlátozza. Az országot már most is egy *„Nagy (Tűz)Fal"* köti össze a világgal, amelyen csak cenzúrázott tartalom továbbítható, mind *Kínából*, mind *Kínába*.

➲ http://www.darknet.org.uk/2006/09/china-outlaws-private-e-mail-servers/

Túl öreg is lehet az ember az internethez?

A válasz igen. *Angliában* a *Carphone Warehouse* céggel szeretett volna szerződést kötni egy 75 éves asszony, az ügyintéző azonban arról tájékoztatta, hogy 70 év felett egyedül ezt már nem teheti meg. A cég előírásai szerint ilyen esetben egy fiatalabb családtagjával kell megjelennie, aki – az ügyintéző jelenlétében – elmagyarázza neki a szerződés részleteit. A szellemileg és testileg is fitt asszony tömören hülyeségnek minősítette azt a feltételezést, mely szerint 70 év fölött az emberek már nem érthetik meg egyedül a szerződést.

➲ http://www.dailymail.co.uk/pages/live/articles/news/news.html?in_article_id=403333&in_page_id=1770

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

Annak idején az *ext3* fájlrendszer az után jelent meg, mint az *ext2* utóda, hogy *Linus Torvalds* kijelentette: nem akar több új funkciót, például naplózó fájlrendszert, mert azok csak összekuszálnák a stabil és jól működő rendszert.
A történet most megismétlődni látszik, ugyanis hiszen már jön is az *ext4*, ami az *ext3* trónjára pályázik és *Linus* ezt is ellenzi. Márpedig ez egyértelműen ellentétes az *ext3* fejlesztői csapatának akaratával, ők ugyanis egyáltalán nem szeretnék elveszíteni a felhasználói bázisukat, vagy több különböző irányba fejlődő változatot karbantartani. *Linusnak* ezzel szemben az a véleménye, hogy ezek nem valami komoly érvek, és egyáltalán nem szeretné föláldozni az *ext3* nyújtotta stabilitást és megbízhatóságot pusztán azért, mert a fejlesztőknek ez kényelmesebb. Feltéve, hogy az uraknak végül sikerül elsimítani ezt a vitát, hamarosan lesz nekünk egy szép új *ext4* fájlrendszerünk, ami 1024 petabájtot képes tárolni az *ext3* „szánalmas” 8 terabájtjával szemben, mellesleg pedig támogatja az extentek használatát. Ez utóbbi egy olyan technika, amely automatikusan megakadályozza az adatok töredezését és az ebből következő lassulást.
Az *Intel* útjára indított egy új projektet, amelynek célja a *WiMedia Ultra Wide Band (UWB)* és a vezeték nélküli *USB* szabványok támogatása. Az *UWB* egy olyan drótnélküli technológia, amelynek egészen rövid a hatótávolsága, így gyakorlatilag csak fizikailag egy szobában található eszközöket lehet vele összekapcsolni. Maga a szabvány és a technológia is annyira új, hogy tulajdonképpen még eszközt se nagyon találni a piacon, ami támogatná. Ugyanakkor az *Intel* szeretné elősegíteni a terjedését, és ehhez a *Linux* fejlesztőinek közösségét is segítségül hívta. *Theodora Y. T'so* azon dolgozik, hogy maguk a lemez *inode*-ok a lehető legkevesebb adatot tartalmazzák, a szükséges információ pedig máshova kerüljön a kernelen belül. A dolog mögött az a valós probléma bújik meg, hogy a fájlok rengeteg *inode*-ot használnak, így ha a rendszer sok fájlhoz fér hozzá, akkor az *inode*-ok tárolásához sok *RAM* kell. Ha tehát az *inode*-ok kisebbek lennének, az az egész operációs rendszer gyorsulását jelentené, hiszen kevesebb memóriát igényelne a nyitott fájlok kezelése. Mivel pedig valamennyi fájlrendszer használ inode-okat, ezek hatékonyabbá tétele értelemszerűen valamennyi fájlrendszer teljesítményére jótékonyan hatna.
Az *inode* adatok rövidre nyírásában többen segédkeztek, köztük *Alexander Viro* is. *Linus Torvalds* maga is egyetért a tervezett módosításokkal, *Ted* pedig meglepő gyorsasággal nyújtott be néhány olyan foltot, amelyek a terv egy jelentős részét már meg is valósították. *Sean Estabrooks* elkészített és a vonatkozó dokumentációval együtt közzétett egy olyan segédeszközt, amellyel *Perforce* kódtárakat át lehet emelni a *git* rendszerbe.
Jon Smirl megkezdte azt a várhatóan rendkívül fáradságos munkát, melynek célja az, hogy a *Mozilla* immár hét évre visszatekintő fejlesztésének anyagát *CVS* alól áttegye *git* alá. Aztán hogy ennek a munkának az eredményét akarja-e majd használni a közösség a további fejlesztés során, az még elválik.
Paul Mackerras megvalósította a *gitk*-ban azt a funkciót, amivel megjeleníthetők egy adott jóváhagyás legközelebbi címkéi (ezek általában a hivatalos verziószámok). Ez a szolgáltatás megkönnyíti annak kiderítését, hogy a változatok közül melyik volt előbb, így egyszerűbb lesz dönteni a vonatkozó foltok jóváhagyásáról. Mivel a művelet végrehajtása meglehetősen számításigényes, *Paul* úgy valósította meg, hogy a háttérben fusson, a képernyő pedig automatikusan frissüljön, ha megvan az eredmény. Immár lehetőség van arra, hogy álneveket (alias) használjunk a *git* parancsokkal kapcsolatban. Sokan rájöttek már, hogy hasznos lenne, ha a megfelelő parancssori argumentumok megadásával rövid parancsokat lehetne megadni. Az ilyesmi különösen azoknak jön jól, akik – akárcsak *Linus Torvalds* – naponta dolgoznak komolyabb mennyiségű javítófolttal, hiszen sok időt takaríthatnak meg azzal, ha kialakítanak maguknak egy „kellemes” munkakörnyezetet.
A közelmúltban volt némi zűrzavar azzal kapcsolatban, hogyan is kellene a *git*-nek értelmeznie a foltok *changelog* mezőjét. *Eric W. Biederman* nemrég írt egy olyan foltot, amelynek hatására a *git* a *changelog* bejegyzés bármely pontján található *From* fejléc tartalmát a kérdéses folt szerzőjeként értelmezte. Aztán *Linus* megjegyezte, hogy ez így nem az igazi, mert a *From* mező csak akkor jelenti a tényleges szerzőt, ha a folt legtetején szerepel. És mivel a *git*, meg persze egyetlen más változatkezelő rendszer sem képes kitalálni, mit is jelent egy neki átadott adat. Ahogy *Linus* fogalmazott: ***„Ezek a rendszerek nem arra valók, hogy találgassanak, hanem hogy egy objektum pontos állapotát rögzítsék úgy, ahogy azt a felhasználó kérte. Ebbe pedig nem férnek bele a „kérlek kedves rendszerem viselkedjél szépen” jellegű óhajok, meg a „hivatal tölti ki” feliratú szürke területek”***.
A *git diff* kimenete immár színezhető is, bár az ezt kiváltó parancssori kapcsoló „megformálásán” még vitatkoznak a fejlesztők. Ami pedig az alapértelmezett színkészletet illeti, nos arról *Linus* csak annyit mondott: ***„a legtöbb embert feltehetőleg arra fogja sarkallni, hogy egy kétágú villával kifordítsa helyéből a szemgolyóit”***. Szóval van még javítani a dolgon.

Linux Journal, 150. szám

Zack Brown

SFD2006 – Szabad Szoftverek Világnapja Szegeden

Idén már harmadjára került megrendezésre világszerte a Szabad Szoftverek Világnapja (Software Freedom Day). Így történt ez Szegeden is.

1. ábra A hallgatóság

2. ábra Kéménczy Kálmán előadása az openSUSE projektről

3. ábra Az openSUSE...

Idén szeptember 17-én ünnepelhettük a ***Szabad Szoftverek Világnapját***. Hála a szponzoroknak, idén is ingyenes volt a részvétel, sőt, délelőtti programként külön busszal ellátogathattak az érdeklődők a szegedi ***Informatika Történeti Múzeumba***, amely ***egyedülálló gyűjtemény*** az országban, de akár még Európában is. Déltől változatos előadásokat hallgathatott a közönség a magyar szabad szoftveres szervezetek elmúlt egy évéről, illetve arról, hogy a kisebb-nagyobb cégek hogyan használják és hogyan támogatják az szabad szoftveres fejlesztéseket.

A rendezvény szüneteiben a résztvevők a rendezvény emblémájával ellátott ***ajándéktárgyakat*** vásárolhattak (***poló, cd tartó***), illetve kiosztásra kerültek ***ingyenes Ubuntu CD***-k és ***matricák*** is. A második szünetben került terítékre az ***nyílt forráskódú palacsinta***, azonban az előre fordított ***bináris*** idén is ***népszerűbb*** volt a forráskódnál, melyet idén is mellékeltek a szervezők a binárishoz. A ***tombolán*** idén ***fraktálképekkel*** és szabad szoftver témájú könyvekkel (***Szabad kultúra***, ***Katedrális és bazár***) lettek gazdagabbak a szerencsések.

A rendezvényről videó és hangfelvétel, valamint számos fotó készült, melyek elérési módjáról később a rendezvény honlapján olvashatunk.

4. ábra Pásztor György előadása a nagyméretű hálózatok szabad szoftveres menedzseléséről

Medve Zoltán
(e-medve@e-medve.hu)

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

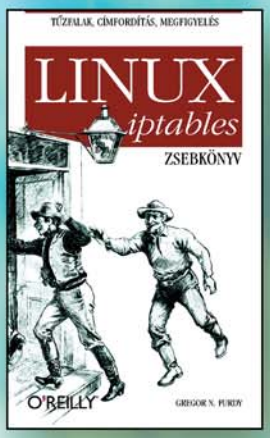
KAPCSOLÓDÓ CÍMEK

Szabad Szoftverek Világnapja (Szeged):
➲ http://www.inf.u-szeged.hu/opensource/sfd/

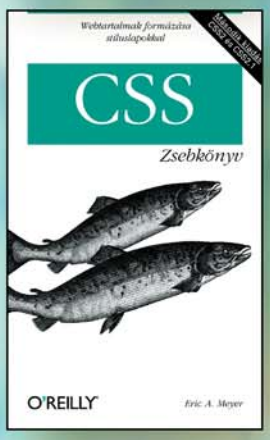
Szabad Szoftverek Világnapja (nemzetközi)
➲ http://softwarefreedomday.org/

Szegedi Informatika Történeti Múzeum:
➲ http://www2.u-szeged.hu/infmuz/

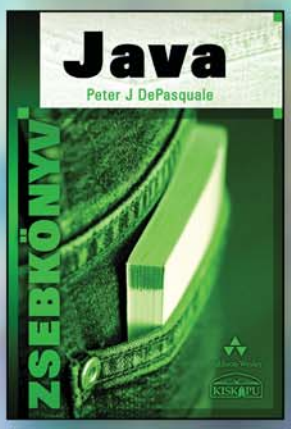
Tudás a zsebben



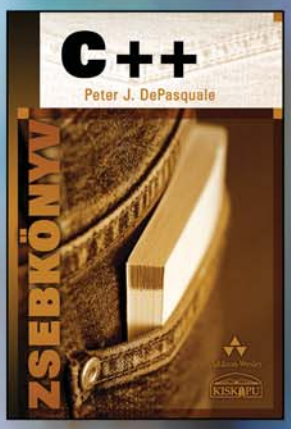
Napjainkban a hálózati biztonság egyike a rendszergazdák számára legfontosabb területeknek. A Linux Netfilter rendszere – vagy ahogy a kezelésére szolgáló parancs után gyakran hívják az iptables – számos dologra képes ezzel kapcsolatban, csak tudnunk kell használni. Ez a zsebkönyv egyrészt általános áttekintést ad a Netfilter rendszer felépítéséről és működéséről, másrészt szerepel benne az összes olyan szintaktikai elem és paraméter, amelyek beállításával a lehető legnagyobb biztonságot érhetjük el.



A HTML kezdeti zűrzavaros fejlődése után az igazi kibontakozást az az egyszerű felismerés hozta meg, hogy célszerű a tartalmat és a megjelenés leírását teljesen különválasztani. Így születtek meg a stíluslapok, illetve a CSS, amivel egészen részletesen szabhatjuk meg, hogy a weblapok egyes elemei hogyan jelenjenek meg a képernyőn. A részletességnek persze ára van: rengeteg paraméter nevét kellene fejben tartanunk, ami a legtöbbünknek természetesen nem nagyon megy. Ez az apró könyv a feledékeny webfejlesztőknek készült.



Bár az objektumközpontú programfejlesztés során a valósághoz sokkal jobban igazodó gondolkodásmódot követhetünk, ez a „logikai kényelem” azonban az ilyen nyelvek összetettségének drámai növekedésével is együtt jár. Ez alól természetesen a Java sem kivétel: számos kulcsszót és felületet kell ismernünk ahhoz, hogy hatékonyan tudjuk használni a nyelv szolgáltatásait. Ez a zsebkönyv tartalmazza a legfontosabb kulcsszavak leírását, rengeteg szintaktikai példát, illetve programozási tippet.



A C++ meglehetősen összetett programozási nyelv. Az alapelvek megértése után ugyan gyorsan tanulható, de elődjével, a C-vel szemben rengeteg kulcsszót, szintaktikai elemet és gyakran használt szerkezetet kell a kezdő fejlesztőnek fejben tartania. Így aztán gyakori, hogy bár az ember pontosan tudja, mit akar megvalósítani, keresgélnie kell az ehhez szükséges nyelvi elemeket. Ezen igyekszik segíteni ez az alig 100 oldalas könyvecske.

Részletes információ és letölthető mintaoldalak:

www.kiskapukiado.hu

KISKAPU
www.kiskapu.hu

Rugalmasan méretezhető számítógép fürtök

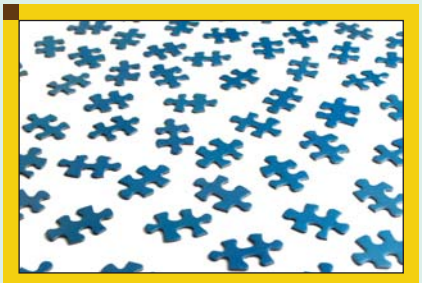
Egyre több vállalkozásnak van szüksége dinamikusan méretezhető számítógép farmra. Az Amazon.com újabb érdekes szolgáltatásával ezt teszi lehetővé.

A technika után érdeklődő világ már egy ideje figyelemmel kíséri az *Amazon.com* életét. Az online kereskedelmi óriás ugyanis számos új koncepciót és szolgáltatást mutatott be, melyekkel új piaci területekre lépett be. Az *Amazon* mérnökeinek elszabadult a fantáziája. Az elsőként bemutatott *Amazon Simple Storage Service* nem más, mint egy hatalmas online tárhely, melyet webszolgáltatásokon keresztül használhatunk programjainkból. Lehetővé teszi, hogy egyszerű parancsokat használva akár 5 Gbyte-os adatokat is eltároljunk ezen a mindig és mindenhonnan elérhető tárhelyen. A legutóbbi szolgáltatásuk *Amazon Elastic Compute Cloud* névre hallgat. Ebben a cikkben erről szeretnék egy rövid áttekintést adni.

Kiszolgálónk méretezése

Szolgáltatás indításánál mindig körültekintően át kell gondolnunk, hogy mekkora forgalmat szeretnénk kiszolgálni. Lehet, hogy csak egy portált szeretnénk üzemeltetni pár ezres látogatói számmal. De mi történik akkor, ha például az oldalunk címe a *digg.com*, vagy a *slashdot.org* címlapjára kerül? Minden bizonnyal gondunk lesz a rengeteg látogató kiszolgálásával. Mi lenne ha egyből két szervert vásárolnánk? Jó ötlet. De várjunk csak! Mi fog történni, ha a látogatói szám visszatér a szokásos értékre? Akkor bizony egy szerver is könnyedén kiszolgálja az oldalt.
Hasonló problémával találkozhatunk naponta. Hasznos lenne egy olyan rendszer, ami lehetővé tenné szerverek csatasorba állítását amikor szükségünk lenne rá, és leállítását olyankor amikor már nem vesszük hasznukat. Pontosan ezt találták ki az *Amazonnál*.

1. ábra Vajon hány kiszolgálóra lesz szükségünk?



2. ábra Számítógépek ameddig a szem ellát

Az *Elastic Compute Cloud* (röviden *EC2*) nevű szolgáltatás megfelel minden kívánalmunknak. Nézzük hogyan! Használatához a következőket kell tennünk:

- Hozzunk létre egy képfájlt, mely tartalmazza a programot amit futtatni szeretnénk, az adatokat és a szükséges beállításokat! Használhatunk előre definiált sablonokat is. Ez a képfájl fog eljövendő szerverünk, szervereink sablonjaként szolgálni.
- Töltsük fel a képfájlt az Amazon tárhelyére!
- Használjuk a rendelkezésre álló webszolgáltatásokat, új szervereket indításához, vagy a meglévők leállításához!

Minden elindított szerver a következő konfigurációnak felel meg: 1.7 GHz *Xeon CPU*, 1.75 GB RAM, 160 GB-os merevlemez, és 250 Mb/s sávszélesség.

Számítógépek ameddig a szem ellát

Az *EC2* segítségével tehát dinamikusan változtatható szerver farmot hozhatunk létre. A megoldás nagyon *rugalmas.* Akkor indítunk új szervert amikor csak akarunk. Ha pedig nincs rá szükségünk akár le is állíthatjuk valamennyit. Mivel az indítás és a leállítás programból szabályozható, így az alkalmazásunk akár *saját magát is skálázhatja*. Kis forgalomnál elegendő lehet pár szervert üzemeltetnünk, míg a forgalom növekedésekor automatikusan több tucat gépre duzzadhat fel a rendszerünk. A rendszer felett pedig mindvégig teljes ellenőrzéssel bírunk, hiszen mindegyik gépre beléphetünk rendszergazdaként.

Mennyibe kerül mindez?

Az Amazon ötletes és szimpatikus árpolitikát folytat. Minden működő szerver után óránként 0.10 dollárt, az 1 Gb sávszélességért 0.20 dollárt kell fizetnünk. Ez azt jelenti, hogy ha egy hónapig folyamatosan futtatunk egy kiszolgálót, az átszámítva 16.000 forintunkba fog kerülni. Ebben természetesen nincsen benne a sávszélesség ára.

Kinek lehet rá szüksége?

Kísérje figyelemmel az *Amazon EC2*-t mindenki, aki a nemzetközi porondon szeretne szolgáltatást indítani. Az interaktív, médiaközpontú és nagy számítási teljesítményt igénylő alkalmazások futtatásához, melyeknél a kihasználtság nagyon ingadozhat, különösen hasznos lehet ez a lehetőség.

Juhász Attila

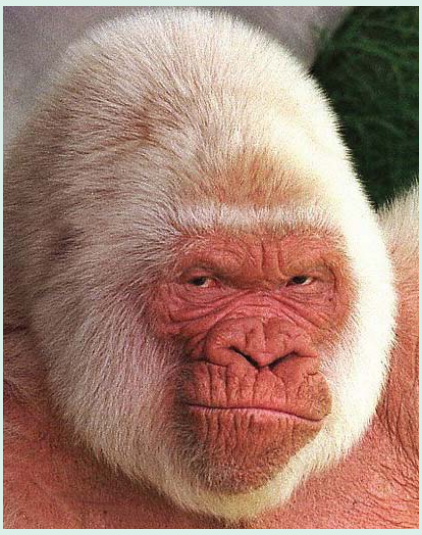

Linuxvilág
www.linuxvilag.hu

A pingvin és a fehér holló

Hol volt, hol nem volt, volt egyszer egy bristoli állatkert és benne kettő darab ivarérett pingvin. Történt pedig e színen és szereplőkkel, hogy elrepült felettük a gólya, s nemi úton érkező hófehér csemetét hagyott maga után.

Mielőtt a pingvinpapa homlokán szarvval gyanúba fogta volna pingvinmamát, hogy eme túl szűzies csemete nem származhatik az ő ágyékából, az állatkert szakemberei védelmükbe vették a megesett madárasszonyt. Noha az albínó pingvin ritka, mint a fehér holló, mégsem egyedülálló jelenség – kettőt is regisztráltak már belőle. A fantáziadús Hóvirág névre keresztelt pigmenthiányos tollas gyermeket, kinek lábai, szeme és arca színe is rózsás árnyalatban játszik, rögvest orvosi vizsgálatoknak vetették alá, s megnyugodva vették tudomásul, hogy bár sápadtabb az átlagosnál, egészsége kifogástalan.

Okos genetikusok nemrégiben beazonosítottak egy gént, mely nagymértékben felelős a bőr- és szőrszín sápadásáért; kísérleteiket egy zebrahal elszíneződésével mérték... Valószínűleg nem további genetikai kísérlet eredménye lett a mellékelt ábrán látható albínó aligátor, mindenesetre az őt megtekintők közül többen állítólag egy másik kutatócsoport kényszerneurózis-tanulmányához jelentkeztek hamarost alanynak. A gyógyászatnál maradva, bizonyára van valami az akupunktúrában, elvégre nem sok beteg sündisznót lát az ember, fehéret meg pláne.



Úgy tartják, ha mindkét szülő hordozza az albínó génváltozatot, akkor is csupán 1:4-hez az esélye festékhiányos utódnak – ezek szerint a kenguruk igen szapora állatok, s még 50 km/h-ás sebességük mellett is jut idejük és módjuk az esélynövelésre. A buenos airesi állatkert még pályázatot is kiírt e világos kedvenceik névadására.

Az ötlet máshol is teret nyerhetett; Hópelyhet, a világ tán egyetlen fehér gorilláját a közelmúltban Barcelonában hősi halottként temették el

37 évnyi kényeztetés után; szegény emberszabású bőrráktól szenvedett. Kétéltűek között is él és hal albínó változat: a zebrahalak mellett elfehéredett csupasz békák próbálnak kimúlni félresikerült álcázási gyakorlatban. A hírek szerint a képen látható egyed a tudományra hagyományozta testét – azonban a tudomány megtámadta a végrendeletet…

Halusz Léna


Linuxvilág
www.linuxvilag.hu

Bill nyugdíjba ment

Kedves Bill!

A minap azt olvastam valamelyik újságban, hogy nyugdíjba mentél. Jó neked! Én még nem értem el azt a kort, hogy ezt megtehessem, de őszintén irigylem azokat, akik félre tudtak tenni egy kisebb összeget ahhoz, hogy akár idő előtt is megkezdhessék a jól megérdemelt pihenést. Na, szóval az van, hogy – amint azt te is tudod – mi még ugyan nem találkoztunk szemtől szemben, miközben pénzt csináltunk az évek során, de ha már itt tartunk, én abban hiszek, hogy az embereknek csak egyszer kell fizetniük egy termékért. Te meg – ha jól hallottam – úgy gondolod, hogy újra és újra fizetniük kell ugyanazért. Na, jól van, spongyát rá. Most már tényleg csak annyit akarok elérni, hogy legalább nyugdíjba a jó úton menjél.

Az első dolog, amiről beszélni szeretnék neked, az a takarékosság. Most, hogy nyugdíjas vagy, nem úgy van ám, hogy a pénz csak úgy dől be a számládra, mint azelőtt. Szóval el kell gondolkodnod azon, hogy hol tudod megfogni a pénzt. Először is javaslom, hogy komolyan gondolkodj el ezen a „Szabad Szoftver" dolgon. Ez ugye arról szól, hogy az ember fogja, oszt' leránt egy jó adag szoftvert az internetről (tudod, ez az a dolog, amiről korábban azt mondtad, hogy ilyen soha nem lesz), majd pedig megvizsgálja, hogy hol tud rajta javítani. Ha nem talál ilyent, akkor sincs semmi gond, nyugodtan használhatja ingyen. Ha pedig kicsit csiszolni kell rajta, akkor a korábbi fejlesztők csak annyit kérnek tőle, hogy amit hozzáírt, azt ő is adja vissza a közösségnek, ingyen és bérmentve. Ugye szerinted sincs ezzel semmi gond ***Bill***?

A másik dolog, amin el kellene töprengened, az a befektetéseid több különböző területen való elhelyezése. Nekem is állandóan ezt mondogatják a tőzsdei tanácsadóim. Azt mondják, hogy amíg nagy a jövedelmem, addig nem gond, ha eljátszadozom egy kicsit a nagy kockázatú, de nagy hozamú részvényekkel, ha viszont nyugdíjba megyek, akkor a stabil, folyamatos bevételre kell koncentrálni. Tudom, hogy a pénzed nagy része abban a vállalatban fekszik, amit alapítottál. Ezzel nincs is semmi gond, csak tudod láttam én már olyant, hogy amint az ember kiteszi valahonnan a lábát, a dolgok egyszerre elkezdenek rosszul menni. Meg általában se jó az összes tojást egy kosárba tenni, mert mi van, ha elejted. Hallottam mostanában egy olyan cégről, amibe szerintem fektethetnél egy kis pénzmagot. Úgy hívják ***Google***. Rengetegen beszélnek róla és szerintem a te „őrült pénzed" egy része is jó helyen lenne ott.

Azt is hallottam, hogy mindenféle jótékonysági ügyekben akarsz részt vállalni. Namost azt ugye tudod, hogy ezek a jótékonysági pasasok rengeteget dolgoznak külföldön, jelesül pedig olyan országokban, ahol – a fene se érti miért – nem beszélnek az emberek angolul. Erre a problémára megint csak azt a ***Szabad Szoftveres*** dolgot tudom neked ajánlani, mert azzal minden ország elkészítheti magának a saját nyelvén beszélő programokat. Ha pedig azt is megtanítod nekik, hogyan csinálhatják ezt a dolgot saját maguk, akkor még több pénzt spórolhatnak meg. „Taníts meg egy embert halászni", ugye te is emlékszel erre a közmondásra ***Bill***?

Ez egyben egy újabb ok arra, hogy a befektetéseidet kicsit jobban szétszórd több hely között. Túl sok pénzed fekszik egyes vállalatokban, ami akadályoz a tisztánlátásban. Például az, hogy olyan sok pénzed van lekötve a saját cégedben arra a téves következtetésre vezethet, hogy a korábban említett országok oktatási, egészségügyi és foglalkoztatási problémáira az egyedül helyes megoldás az, ha a jó öreg vállalatod termékeiből adsz nekik újabb és újabb példányokat. Pedig dehogy! Meg kell tisztítanod az elmédet! Meg kell próbálnod kitörni abból a dobozból, amibe be vagy zárva! Látnod kell, hogy igazán jótékonynak lenni csak a szabad szoftverekkel tudsz.

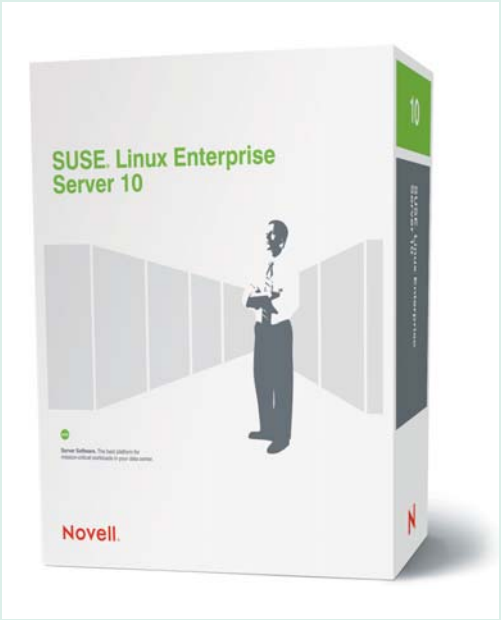
Képzeld el, hogy van ***Kongóban*** egy szegény diák, akinek odaadtad valamelyik termékedet. Emberünk feltelepíti, aztán egyszer csak jön a termékaktiválás. Mégis kit hívjon fel? Hogyan használja a szoftvert, amit adtál neki, ha nincs a kezében a forráskódnak még az a része sem, amivel kiiktathatná ezt az ellenőrzést? Ja, és azt ugye te se gondolod komolyan, hogy egyszer majd fizetni fog a frissítésért?
Találkoztam ***Dél-Afrikában*** diákokkal, akik a digitális fotózást tanulták. Amíg az iskolában voltak, nem is volt semmi gond, de amint hazamentek, és gyakorolni szerettek volna a saját számítógépükön, kénytelenek voltak lopott szoftvert használni. Kalózok voltak. Ezért aztán adtam mindegyiküknek egy-egy példányt a ***Gimp***-ből, és láss csodát, többé már nem voltak kalózok. Ugye, milyen egyszerű a megoldás?
Na, aztán azt is hallottam, hogy a te jó barátod, ***Mr. Buffett*** valami 35 milliárd dollárt adott a te jótékonysági alapodnak. Hát mit mondjak, ez állati rendes volt tőle. Gondolom semmiféle megkötéssel nem élt az ajándék felhasználásával kapcsolatban, szóval nem vagy köteles megvetetni az általad támogatott szervezetekkel a ***CD***-it, vagy odavezényelni őket a koncertjeire. Legyél tehát te is nagyvonalú, és úgy támogasd őket, hogy közben nem feszítesz kezükre-lábukra húrokat. (Érted a poént?! Koncert! Gitárhúrok!)
Szóval próbálj ellenállni a kísértéseknek, és fektess ***Szabad Szoftverbe***, a ***Red Hat***-be, vagy akár a ***Novellbe***. Így tudod a legmesszebbre eljuttatni a támogatásokat.
Na, ha már a bőkezű adományoknál tartunk, kíváncsi vagyok szerinted mekkora lenne az az összeg, amit a ***Szabad Szoftveresek*** összeharácsoltak volna, ha újra és újra megfizettetik a felhasználókkal a szoftvereket ahelyett, hogy egyszerűen csak örömből fejlesztették őket. Szerinted összehoztak volna annyit, mint amennyivel te most jótékonykodni fogsz? Vagy megüberelték volna ***Mr. Buffett*** adományát is? Hát, szerintem van pár guriga a dologban! Ehhez képest valahogy túl sokat írnak a sajtóban a te adományaidról, és nagyon keveset az övékről. Na, de nem gond, amint támogatni kezded a ***Szabad Szoftvereket***, egy csapásra mindent jóváteszel.
Na, ennyit a hírekről, meg a jó tanácsokról. Igazából ennyit akartam csak elmondani neked. Ja, azért azt még had említem meg, hogy a lapok szerint legalább két évbe telik majd, amíg teljesen képes leszel kivonni magad a cég ügyeiből. Apám! Tök szerencse, hogy nem csapott el eddig egy kamion. A vállalatod egyszerűen összenyaklott volna nélküled! Hát, ha ez igaz, akkor szerintem legfőbb ideje volt, hogy vedd a kalapod és továbblépj. Ja, amúgy ez egy újabb érv amellett, hogy jó lesz a pénzedet máshol tartani.
Melindát üdvözlöm, és kérdezd meg tőle, hogy tetszik neki az a pár pingvines fülbevaló, amit küldtem.

Szívélyes üdvözlettel
maddog

PS: Basszus, most mondja a titkárnőm, hogy az ***Warren Buffett*** volt, nem a ***Jimmy***. Na, most már mindegy. Sose találkoztam személyesen ezzel a ***Warrennel***, még csak nem is kártyáztunk egy asztalnál, de szerintem ***Jimmy*** jobban mutatott volna a búcsúpartidon. Ja, azt a viccet a gitárhúrokkal meg elfelejtheted.

Új, Intel technológián alapuló virtualizációs megoldás a Novelltől

A Gartner független piacelemző cég felmérései szerint napjainkban egyre több ügyfél dönt a virtualizációs megoldások mellett hardver eszközeik egyszerűsítésére, rendszerük felügyeletének javítására, vagy a továbbfejlesztett hardverek megnövelt rendelkezésre állása, megbízhatósága és skálázhatósága miatt. A Novell és az Intel most közösen kínálja az iparág első olyan vállalati, Linux alapú virtualizációs megoldását, melynek alapja az Intel Virtualization Technology megoldásra optimalizált Xen technológia.



A *Novell* a múlt hónapban jelentette be az iparág első olyan vállalati *Linux* alapú virtualizációs megoldását, melynek alapja az *Intel Virtualization Technology* megoldásra optimalizált *Xen* technológia. A *Dual-Core Intel Xeon* platformon futó *SUSE Linux Enterprise Server 10* alacsony költségszint mellett nagy teljesítményű virtualizációs megoldást kínál az ügyfelek számára, amely a vendég operációs rendszerek módosítása nélkül képes *Linux* környezetek fogadására. Az *Intel Virtualization Technology Xen* szoftverbe való integrációjával egyidőben a *Novell* bejelentette, hogy a jövőben vállalati szintű támogatást biztosít a virtuális *SUSE Linux Enterprise Server 9* és a *SUSE Linux Enterprise Server 10* termékén futó *Red Hat Enterprise Linux 4* rendszerekhez is. Ezáltal lehetővé teszi a *Red Hat* felhasználók számára, hogy igénybe vegyék a Novell szolgáltatásait és támogatását, miközben a *Red Hat Enterprise Linux* rendszert futtathatják virtualizált környezetükben. *„A Gartner független piacelemző cég felmérései szerint napjainkban egyre több ügyfél dönt a virtualizációs megoldások mellett hardver eszközeik egyszerűsítésére, rendszerük felügyeletének javítására és a továbbfejlesztett hardverek megnövelt rendelkezésre állásának, megbízhatóságának és skálázhatóságának kihasználása érdekében. Mivel a Novell elsőként kínál nagyvállalati Linux rendszerekhez kifejlesztett Xen virtualizációs megoldást, ügyfeleink is elsőként élvezhetik ennek előnyeit”* – nyilatkozta *Hargitai Zsolt*, a *Novell Magyarország* vezető rendszermérnöke. *„A virtualizált Linux platformot – például Novell SUSE Linux Enterprise 10 platformot – futtató felhasználók a már meglévő virtualizációs megoldások árának töredékéért használhatják ki a virtualizáció előnyeit: a költségek megtakarítását és a rugalmasságot”* – tette hozzá *Hargitai*.

A virtualizáció vállalati szintű támogatása

A *Novell* az *Intel Virtualization Technology* megoldást támogató *SUSE Linux Enterprise Server 10* termékén futó *Red Hat Enterprise Linux 4* rendszert a 3. (rendszermérnöki) szintig támogatja. Ennek értelmében a *Novell* műszaki támogatást nyújt a *Xen* virtuális gépeket felügyelő szoftver (*hypervisor*) számára, amennyiben az ügyfél problémába ütközik a *Red Hat Linux* virtuális példányának futtatása közben és a hiba nem reprodukálható natív, vagyis nem virtualizált környezetben. Ha az ügyfél virtuális *SUSE Linux Enterprise Server 9* vagy *10* rendszert futtat, a *Novell* fejlesztői szintű támogatást nyújt a befogadó operációs rendszerhez, a *Xen hypervisor* szoftverhez, valamint a vendég operációs rendszerhez is. Emellett az *Intel* és a *Novell* együttműködik a *hypervisor* szoftverhez és az *Intel Virtualization Technology* hardverhez kapcsolódó összes hiba megoldásában is.

A SUSE Linux Enterprise Server 10

A *SUSE Linux Enterprise Server 10* integráltan és támogatottan tartalmazza a nyílt forrású *Xen* virtualizációs technológiát, ami lehetővé teszi különböző platformokon futó alkalmazások konszolidációját egy szerveren, ezáltal javítva az erőforrások kihasználtságát, növelve a hatékonyságot és csökkentve a költségeket. A *Xen* technológiát a *Novell YaST*-alapú (*Yet Another Setup Tool*, telepítő- és konfigurációs eszköz) menedzsment eszközei is támogatják, leegyszerűsítve a virtualizált megoldások bevezetését a vállalati infrastruktúrákon. A *SUSE Linux Enterprise Server 10*-ben található *Xen 3.0* támogatja az *AMD Pacifica* és az *Intel VT* hardveres virtualizációs technológiáit. A *YaST* fejlődésének köszönhetően egyszerűsödött a telepítés és konfiguráció. A *SUSE Linux Enterprise Server 10* architektúrája teljesen *CIM*-kompatibilis (*Common Information Model*) ezáltal zökkenőmentesen együttműködik más monitoring és menedzsmenteszközökkel is.

A *SUSE Linux Enterprise 10* a *Novell* következő generációs nyílt vállalati platformja, és az első vállalati *Linux*-disztribúció, amely tartalmazza a *Xen* alapú virtualizációt. A *Novell SUSE Linux Enterprise 10* tartalmazza a *SUSE Linux Enterprise Server* és *SUSE Linux Enterprise Desktop* termékeket, így biztonságos és megbízható alapot nyújt a vállalati felhasználóknak az asztali megoldásoktól az adatközpontokig. A platform kiegészül a *Novell* nemrégiben bejelentett online portáljával, a *Novell Customer Centerrel*, amely olyan eszközöket és automatizált szolgáltatásokat kínál, melyek használata leegyszerűsíti a licencelést és a frissítések kezelését, így csökkenti a rendszerfelügyelet költségét.

Az *Intel Virtualization Technology (VT)* hardvertámogatást nyújt a virtualizációs szoftverhez (*Virtual Machine Monitor – VMM*), csökkentve ezáltal a *VMM* beavatkozásának – így a bonyolult, számításigényes szoftverfordítási feladatok – szükségességét. Az *Intel* 2005 novembere óta szállít virtualizációs technológiájával ellátott processzorokat kiszolgálók és munkaállomások számára.

Az *Intel Virtualization Technology* egy olyan többgenerációs fejlesztési terv első állomása, amely egyre nagyobb teljesítményű fejlesztéseket tartalmaz az *Intel* platformjaihoz, és többszintű védelmet biztosít az összes platformerőforrás – processzor, memória és I/O erőforrások – virtuális eléréséhez.

Elérhetőség

A *Novell* október végén próbaprogramot indít számos nagyvállalati ügyfél részvételével. A *Novell Intel* technológiáján alapuló virtualizációs megoldása várhatóan ez év végén lesz elérhető. A *Dual-Core Intel Xeon* platformon *SUSE Linux Enterprise Server 10* virtualizációt futtató kiszolgálók már elérhetők az összes nagyobb számítógépgyártónál. További információ a ➲ http://www.novell.com/intel weboldalon érhető el.

PHP nyomkövetők (4. rész)

Sorozatunk előző részeiben áttekintést adtunk néhány PHP nyomkövetőről, a Gubed/Quanta és a Nusphere::PhpED programokról. Ez alkalommal az Xdebug-ra épülő ActiveState::Komodo és a Xored::TruStudio fejlesztőkörnyezeteket vesszük szemügyre, valamint készítünk egy Firefox-bővítményt a nyomkövetés kényelmesebbé tételéhez.

A DBGp

Maga az ***Xdebug a DBGp*** nyomkövetési protokollra épül (***Common DeBuGger Protocol***). Ennek dokumentációja így határozza meg önmagát: ***„A common debugger protocol for languages and debugger UI communication”***, azaz ***„közös nyomkövetési protokoll nyelvek és nyomkövető felhasználói felület közti kommunikációra”***.

A szerzők: ***Shane Caraveo (shanec@ActiveState.com)*** és ***Derick Rethans (derick@derickrethans.nl)***, aki a 2005-ös PHP Konferencián Magyarországon is járt. Az ***Xdebug*** honlapja mintaszerűen egyszerű és igényes ➲ ***www.xdebug.org***. 2006 júniusában már régóta az xdebug-2.0.0beta5 a legfrissebb verzió. A holland szerző hasznos tippjei a ➲ ***www.derickrethans.nl/errorhandling/talk.html*** oldalon olvashatóak.

Az URL első részletét nézve a blogból arról értesülünk, hogy igen széles látókörű programozóval van dolgunk, akit messze nem csak a bitek állítgatása érdekel – s ez fontos szempont lehet, amikor dönteni akarunk valamelyik nyomkövető vagy nyomkövetési protokoll mellett.

Az utóbbi időben viták lángoltak fel azzal kapcsolatban, hogy mitől jó/jobb egy nyomkövetési protokoll.

Az ***Xdebug*** a ***DBGp***-t használja, ami ***ASCII*** adatforgalmat jelent, ***XML*** formátumban, szemben pl. a ***Zend Studio*** bináris szerkezetű protokolljával. A nyílt forráskódú programok írásának nyilván az előbbi, átláthatóbb szabvány kedvez – a ma használatos sávszélességek mellett általában elhanyagolható az a kis különbség, amivel több adatforgalmat jelent az ***ASCII*** kódolás a binárissal szemben. Ezzel együtt sajnos meg kell hagyni, hogy a ***Zend Studio*** mint eszköz valóban minden igényt kielégít a Linux-kedvelők háza táján is – nem egyszerű versenyre kelni vele. De erről majd egy későbbi részben.

Az ***Xdebug*** támogatja még a ***GDB***-t (***Gnu DeBugger protocol***) és a ***PHP 3 Debugger protocol***-t is.

Xdebug: a webszerver oldalán

Az ***Xdebug*** a webszerver „oldalán” fut, a kliensprogram tőle függetlenül a helyi gépen. (E kettő meg is egyezhet.) Ha éles környezetben szeretnénk forrásból lefordítani az ***Xdebug***-ot, akkor (***dpkg*** alapú csomagkezelő jelenléte esetén) fordítás előtt adjunk ki egy

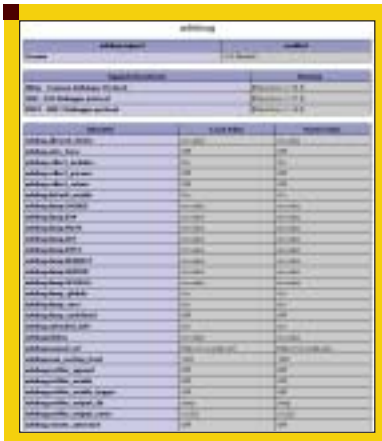
```
apt-get install php-devel
➥automake gcc cpp
```

parancsot, az egész procedúra után pedig majd ezt:

```
apt-get remove  php-devel
➥automake gcc cpp
```

Azaz a fordításhoz szükséges programokat kénytelenek vagyunk feltenni, de utána kár lenne telepítve hagyni őket. A biztonságot szem előtt tartva ne adjunk teret felesleges programok futtatásának a későbbiekben.

Lássuk tehát magát a fordítást. Csomagoljuk ki az ***Xdebug*** programot a `tar -xvzf xdebug-2.x.x.tgz`



1. ábra phpinfo – a lefordított és beüzemelt Xdebug modullal

paranccsal. Váltsunk be (cd-vel) a keletkezett könyvtárba. Bárhová kicsomagolhatjuk a *tar* csomagot, nem kell a *PHP* forrása környékére, hiszen magát a *PHP*-t nem kell újrafordítani (annak ellenére, hogy a forrására szükségünk van, épp a most következő parancshoz). Futtassuk le a

```
phpize
```

parancsot, amely remélhetőleg benne van a *PATH*-unkban. (Ha nincs, adjuk meg abszolút elérési úttal.) Ezután – behelyettesítve a */etc/php.ini* helyére a mi webszerverünk *PHP*-konfigurációs fájlját – jöhet a *make* fájl elkészítése és futtatása:

```
./configure --enable-xdebug
➥-with-php-config=/etc/php.ini
make
```

A kulcsfontosságú lépés, melyben – root-ként – a „végterméket”, az *xdebug.so* modult elérhetővé tesszük a webszerver számára:

```
su -c 'cp modules/xdebug.so
➥/usr/lib/php'
```

Értelemszerűen a */usr/lib/php/* helyett a *php.ini*-ben levő *extension_dir* beállításának tartalma értendő. Az *ActiveState* javaslata szerint a nyomkövetéshez érdemes kikapcsolni a *php.ini*-ben az *output_buffering* hatást egy pontosvessző sor elejére írásával (ezzel megjegyzésbe tesszük), vagy *„off”*-ra állításával. A *php.ini* konfigurációs fájl végére írjuk be az alábbi sorokat (a */usr/lib/php/* helyett itt is az *extension_dir* beállításának tartalma értendő, és az *<idekey>* helyett egy azonosító karaktersorozat) :

```
zend_extension="/usr/lib/php/
➥xdebug.so"
xdebug.remote_enable=1
xdebug.remote_handler=dbgp
xdebug.remote_mode=req
xdebug.remote_host=localhost
xdebug.remote_port=9000
xdebug.extended_info=1
xdebug.idekey=<idekey>
xdebug.profiler_enable=1
xdebug.profiler_output_dir="/tmp"
```

Ezekről a kapcsolókról természetesen bőséges információt nyerhetünk az *Xdebug* honlapjáról közvetlenül, vagy az onnan letölthető dokumentációból (*docs.tar.gz*). A *profiler*-re vonatkozó utolsó két sor csak akkor aktuális, ha profilt is szeretnénk készíttetni, azaz teljesítményt elemeztetni. Miért is ne tennénk? Ez az egyik leghasznosabb lehetőség az *Xdebug*-ban. Ezek után már csak a webszerver újraindítása van hátra. Ha ezek után kiadunk parancssorból egy `php -m` parancsot, akkor a webszerver *PHP* és *Zend* moduljai listázódnak ki – mindkét listában látnunk kell az *Xdebug*-ot. Böngészőnkben hívjunk meg egy ilyen *PHP* oldalt:

```
<?php phpinfo(); ?>
```

Látszania kell az *Xdebug* modulnak (*Zend Technologies with Xdebug v2.0.0beta5, Copyright (c) 2002, 2003, 2004, 2005, by Derick Rethans*) és részleteinek (*1. ábra*)

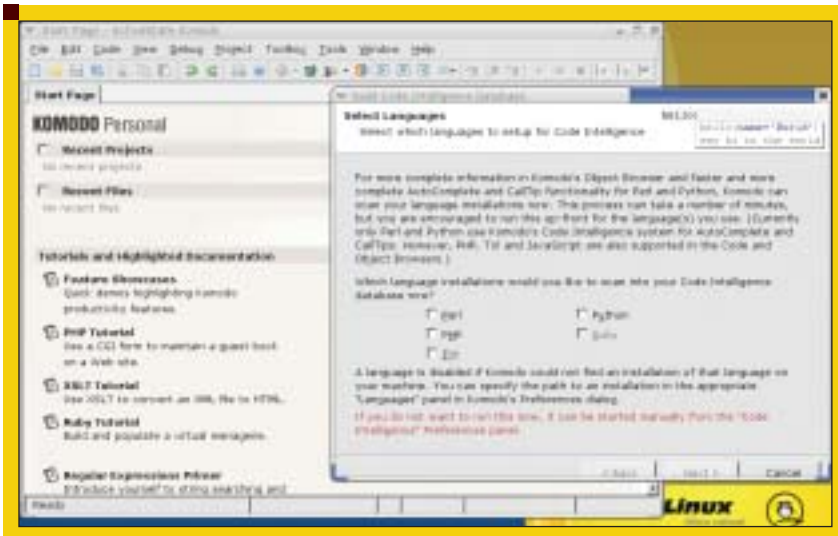
Talán feltűnt a fenti beállítások közt, hogy *localhost*ot adtunk meg *remote_host*-ként, azaz *„távoli gép”*-ként. Ezt a szokásos módon egy *SSH*-alagút nyitásával oldjuk meg – így a legbiztonságosabb:

```
ssh -R 9000:localhost:9000
➥loginnév@gépnév
```

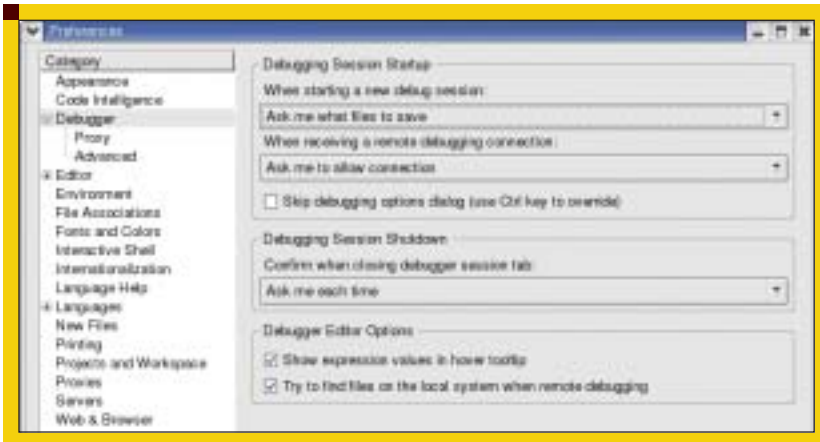
Kliensprogramok Xdebughoz

Többféle kliensprogram is íródott az *Xdebug*-hoz. Magában az *Xdebug* csomagban is van egy *debugclient* nevű parancssori kliens, de ezt elég körülményes használni az *XML* szerkezetek miatt. A szerző fent említett „hasznos tippjei” közt a *Weaverslave* programot nevezi meg egy használható kliensprogram-alternatívaként. Ez Microsoft Windows alá készült, de *wine*-nal is elindul (*2. ábra*).
Az igazán jól használható kliensprogram (sőt, annál több is) az *ActiveState* által kiadott *Komodo*. A 3.5-ös verzióval próbálkoztam 2006 júniusában. Időkorlátos: egy hónapig próbálható ki ingyenesen, de ha lejárt az időszak, akkor lehet kérni, hogy küldjenek újabb licencet – vagy megvásárolható a használati jog 30 dollárért. (A */home/.komodo* törlésével és magának a programnak az újratelepítésével nem lehet meghosszabbítani az időkorlátot, mert a licencfájlba a lejárati dátum van belekódolva.) Angol nyelvű, igazán jól felszerelt fejlesztőkörnyezetet kapunk a kezünkbe, van benne több programnyelvhez is tanfolyam (persze *PHP*-hez is). Alapvetően projekt alapon végzett munkához lett tervezve, de különálló fájlok is szerkeszthetőek.

2. ábra Weaverslave 4 – kliensprogram



3. ábra Komodo – telepítés. Készül a KódIntelligencia Adatbázis



4. ábra A Komodo nyomkövetési beállításai

5. ábra Komodo felhasználói kézikönyv – mindenre kiterjedő eligazítás

A *Komodo* (vagy bármely más fejlesztői) környezet egyik leghasznosabb támogatása az, hogy már begépeléskor jelzi – hullámos aláhúzással – a szintaktikai hibákat. Ez rengeteg időmegtakarítással jár, mert meg sem kísérli az ember a hibás kód kipróbálását.
Az *Edit|Preferences|Debugger|Proxy* menüpont alatt a *„Listen for debug connections on port:”* értékét ugyanarra érdemes állítani, mint amit fent a *php.ini*-ben megadtunk (9000).

Beállításainkat ellenőrizhetjük a *Debug|Listener Status* menüponttal. Ha még nem lenne bekapcsolva, kattintsunk a Debug menü *Listen for Remote Debugger* pontjára (azaz: *Távoli nyomkövető figyelése*).
Következik a webböngésző okosítása. A normál *URL* végére írjunk egy *„?XDEBUG_SESSION_START=<idekey>”* kiegészítést, mint *GET* argumentumot. A dokumentáció szerint az *<idekey>*-nek meg kell egyeznie a *Debug|Listener Status-beli Proxy Key* értékével, azonban azt tapasztaltam, hogy ha helyi gépen futtatjuk a webszervert, akkor ez nem kötelező, sőt, még a *php.ini*-beli *idekey*-értékkel sem kellett, hogy megegyezzen a fenti *GET* argumentum.
Amikor először kap ilyen „megbízást” a böngésző, hogy keresse fel például ezt az oldalt:

```
http://valami.hu/pelda.php?
➥XDEBUG_SESSION_START=joskapista
```

akkor sütivel (*cookie*) rögzíti az *idekey* információt, és a későbbiekben már arra támaszkodik. Ezekről a böngésző által támogatott munkamenetekről bőven olvashatunk a ➲ *www.xdebug.org/docs-debugger.php#browser_session* oldalon.
Ezek után megkezdődik a nyomkövetés a *Komodo*-ban, lehet lépegetni, futtatni az első (akár feltételes) töréspontig stb. Sőt, a *PHP* kódon belül is elhelyezhetünk töréspontot az

```
xdebug_break();
```

használatával (bár ez nem kezdeményez új munkamenetet, és egyébként sem szép belenyúlni a kódba – ez talán inkább a spártaibb kliensprogramoknál segíthet).

A Firefox-bővítmény

Sorozatunk egy korábbi részében vizsgáltuk a *Gubed* nyomkövetőt. Láthattuk, hogy milyen sokat segített egy egészen egyszerű kiegészítés a böngésző menüpontjainál, amiben a nyomkövetést kiváltó *URL*-változtatást lehetett kérni vagy visszavonni.
Úgy döntöttem, hogy ezt a lehetőséget az *Xdebug* számára is meg kell teremteni. Az említett *Gubed*-bővítmény nagyon hasonlóan működik, mint amire

az *Xdebug*-gal való nyomkövetéskor van szükség: kis módosítás az *URL*-en, és már megy is a nyomkövetés.
A ➲ *gubed.sf.net/mozilla/gubed.xpi* helyről letölthető *gubed.xpi*-ből indultam tehát ki. Kíváncsi voltam, mi az a minimális erőráfordítás, amivel már el tudom érni célom.
A kiindulási fájlt átneveztem *xdebug.xpi*-nek. Mivel ez (többszörösen) tömörített fájl, először gyártottam egy kibontó és egy összerakó parancsot. Ezzel lehetett kibontani az *xdebug.xpi*-t:

```
rm -Rf ki; mkdir ki; cd ki
➥;unzip ../xdebug.xpi ;cd
➥chrome; mkdir ki; cd ki;
➥unzip ../gubed.jar
```

Ezzel pedig össze lehet csomagolni *xdebug2.xpi* néven, hogy ne írja felül a kiindulási fájlt:

```
cd ki/chrome/ki; zip -9 -r
➥../gubed.jar *; cd .. ; rm -r
➥ki; cd .. ;zip -9 -r
➥../xdebug2.xpi *
```

A *„ki”* (azaz *kibontott*) könyvtárban végrehatjottam az alábbi (kis-nagybetű érzékeny!) cserét:

```
perl -pi -w -e 's/Gubed/
➥Xdebug/g' `tree -fi`
```

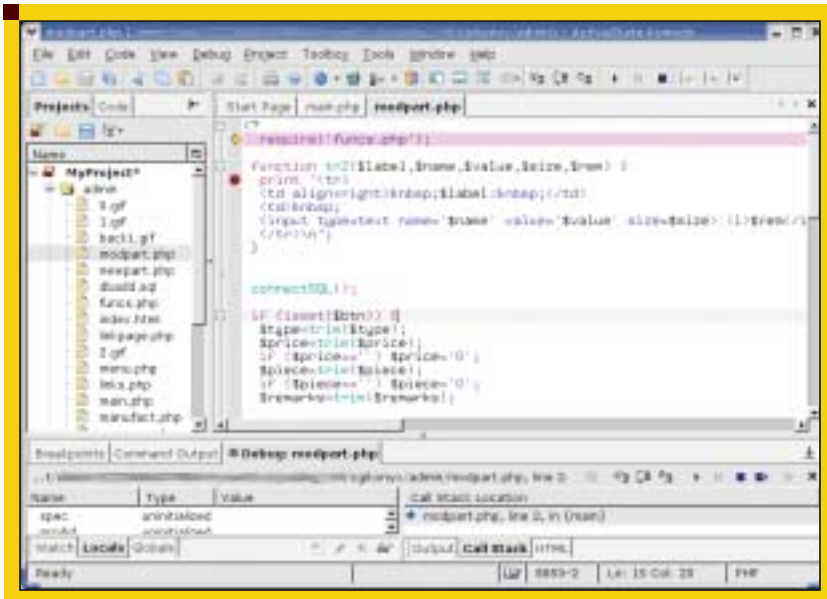
Ezek után már bele lehetett nyúlni a *JavaScript* programocskába és a menürendszerbe (ez utóbbi nem volt létszükséglet, de ha már hozzányúltam, miért ne magyarítsam azt a 4 menüpontot).
A *gubedOverlay.js* fájlban megjegyzéssel (azaz // jellel) láttam el a

```
getURL="?gbdScript=";
```

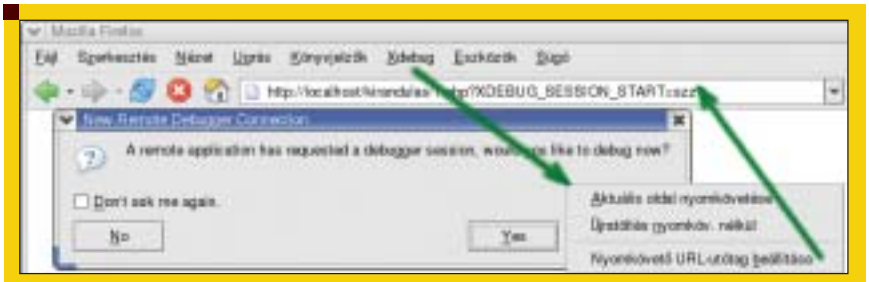
sort, és alatta elhelyeztem a számomra hasznosabb

```
getURL="?XDEBUG_SESSION_START=
➥kulcsom";
```

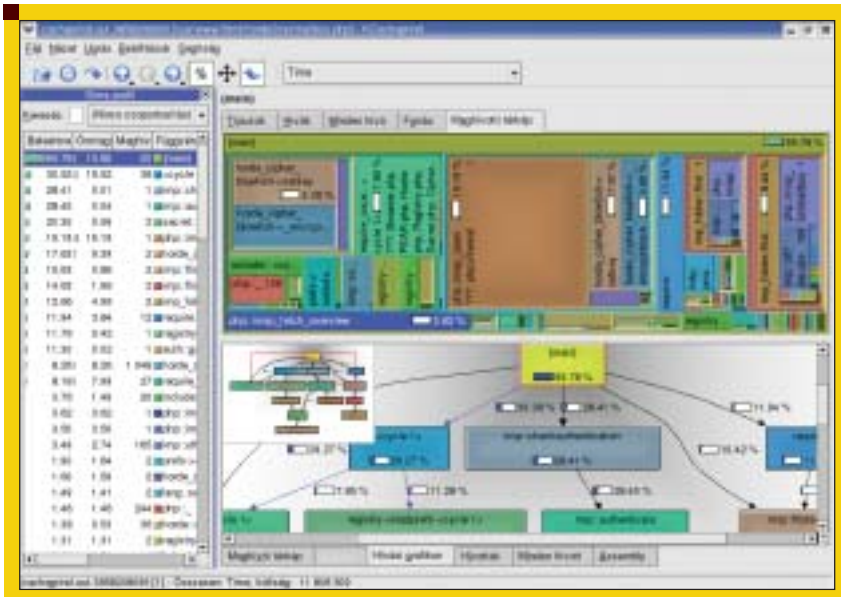
parancsot. Itt a *„kulcsom”* egy olyan *„IdeKey”*, azaz *Fejlesztő-környezetiKulcs*, amivel azonosítani tudja a webszerver, hogy melyik fejlesztő környezet kéri a nyomkövetést. Ez benne van a *php.ini* fájlban is a megfelelő részletnél, mint fentebb említettük. Arra szolgál, hogy lehetővé tegye az azonosítást, ha netán egyszerre többen is szeretnének nyomot követni, és segít kivédeni, hogy illetéktelen kezdjen el munkálkodni. Nem kell lelkiismereti problémát csinálni abból, hogy ezt belehuzalozzuk a kódba, mert van egy beállítási lehetőség



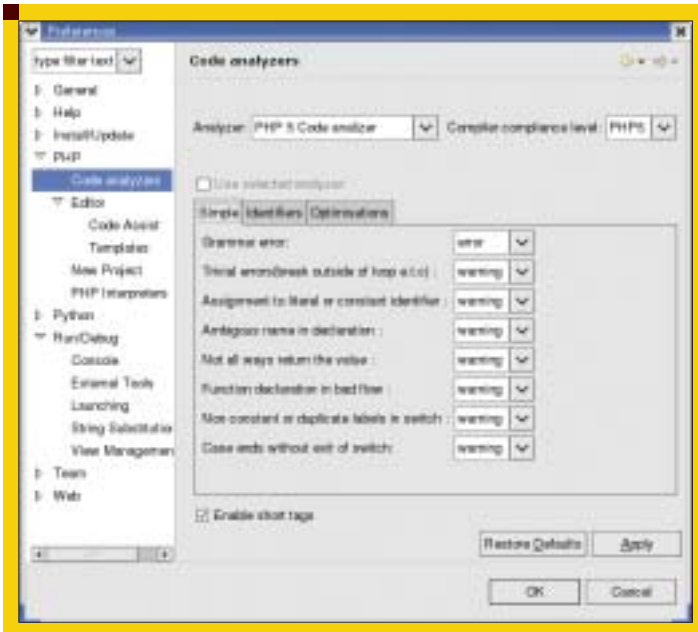
6. ábra A Komodo teljes díszben, beállított töréspontta



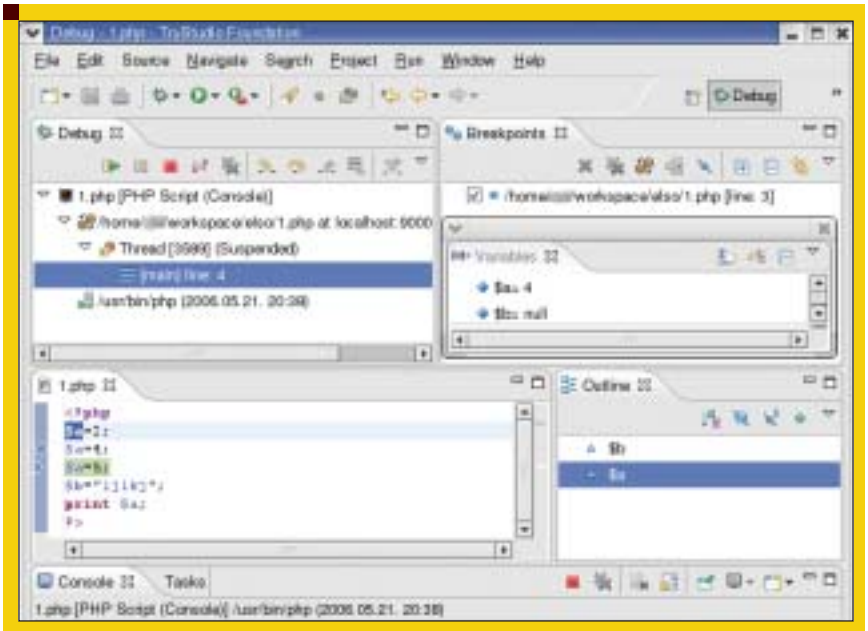
7. ábra A saját gyártású Xdebug bővítmény



8. ábra Teljesítményelemzés az Xdebug kimenetéből, KcacheGrind-del



9. ábra Kódelemző a Xored::TruStudio-ban



10. ábra Teljes díszben a Xored::TruStudio – nyomkövetés töréspontal

a bővítmény menüpontjai közt, ami éppen ennek átírására szolgál. Majd néhány sorral lejjebb:

```
//  var url = scriptURL +
➥getURL + current_url;
  var url = current_url +
  ➥getURL ;
```

Végül egy apróság (ami különös módon nem a menüpontok szövegeinél van, hanem bele lett drótozva a programba) – ezt *iso-8859-2*-es kódolással fogadta el a program:

```
//  var tmp = prompt( "Enter the
➥Script URL: ", scriptURL );
  var tmp = prompt( "írja be
  ➥a nyomkövetési utótagot: ",
  ➥getURL );
```

Lejjebb a *ScriptURL* értékadása helyett:

```
  getURL = tmp;
```

Ezen kívül még a *locale* könyvtár alatti *en-US*-t másoltam át *hu-HU* nevűként, és írtam át azt a néhány menüpontot, ami ott felsorakozott, *UTF-8*-as kódolással. (Vagy ékezet nélkül is lehet, ha biztosra akarunk menni, és nem lényeges az akkurátus ékezetek látványa).

Ezek után összecsomagoltam az *.xpi* fájlt (ami leszedhető a ➲ *www.osb.hu/z/xdebug.xpi* helyről), és húzd-és-ejtsd módszerrel ráhúztam a *Firefox*-ra. Következő indításkor már meg is jelent és használhatóvá vált a megfelelő *Xdebug* menüpont. (Ezt azonban nem lehetett egyszerre használni a *Gubed* menüponttal, az ütköző azonosító- és változónevek miatt; vállalkozó kedvűek kijavíthatják ezt a szépséghibát.)

Megfelelően beállított *Xdebug*-gal teljesítményelemzést is végezhetünk (lásd az `xdebug.profiler...` beállításokat a *php.ini* fájlban). A megadott könyvtárban szépen gyülekeznek a *KCacheGrind*-nek közvetlenül átadható fájlok.

A Xored::TruStudio

Maga a program a ➲ *www.xored.com/trustudio/download* oldalról tölthető le. Az *Eclipse*-re épül, de ezt nem kell (sőt nem szabad) külön telepíteni előtte! Bár nagyon komoly, *Java* alapú fejlesztőkörnyezetként működik, a *Xored::TruStudio* (tapasztalataim szerint) sajnos csak a helyi gépen, *CGI PHP*-környezetben tud futni, és ráadásul időnként egy licenc után érdeklődő (de aztán eltüntethető) dialógusablak zavarja meg jó közérzetünket. Ennek ellenére – különösen a kódelemző része miatt – megér egy próbát, hátha egy hamarosan megjelenő következő verzióban már távoli nyomkövetésre is képes lesz.

Sorozatunk befejező részében a *java* alapú *Zend Studio* kliensprogram és a hozzá tartozó (*Apache*-modulként működő, *.so* fájlokat tartalmazó) szerver kerül terítékre.

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

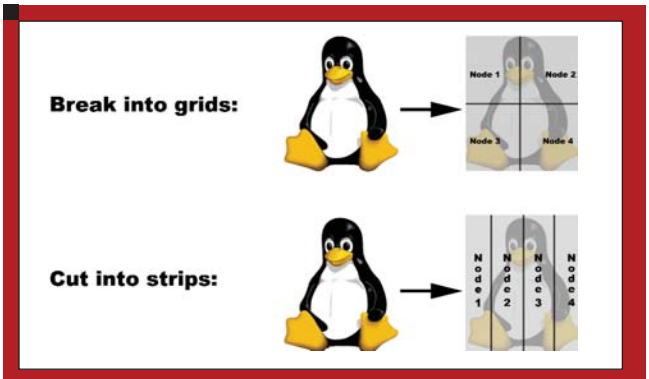
A párhuzamos programozás jelene

Legyen az ember akár tudós, grafikus, zenész vagy filmrendező, könnyen lehet, hogy munkája során egyszer hasznosnak találja majd a mai nagy teljesítményű Beowulf klaszterek képességeit.

Egy tipikus párhuzamos rendszer

Először is szükségünk lesz néhány azonos számítógépre, amelyeken *Linux* fut, és amelyek nagy sebességű *Ethernet* hálózattal vannak összekötve. A legjobb a *Gigabit Ethernet*, hiszen a hálózat sebessége az egyik olyan dolog, ami erősen visszafoghatja egy klaszter teljesítményét. Szintén szükségünk lesz valamilyen elosztott fájlrendszerre, és persze a klasztertechnológiával kapcsolatos könyvtárakra, szoftverekre. A legtöbb klaszterben egyszerűen *NFS*-t használnak elosztott fájlrendszerként, bár tény, hogy létezik néhány ennél egzotikusabb megoldás is. Ilyen például az *IBM GPFS* (*General Parallel Filesystem*) rendszere. Ami a klaszteren történő számítások szoftveres támogatását illeti, itt is van néhány választási lehetőségünk. Napjainkban a de facto szabvány az *MPI* (*Message Passing Interface*), de a *PVM* (*Parallel Virtual Machine*) könyvtár is kiválóan működik. Az elmúlt

Amikor *Donald Becker* az 1990-es évek elején a *NASA*-nál dolgozva felvetette a *Beowulf klaszter* megépítésének lehetőségét, mindörökre megváltoztatta a nagy teljesítményű számítógépek fejlődéstörténetét.
Ez az ötlet ugyanis nem kevesebbet jelentett, mint hogy a korabeli szuperszámítógépek teljesítményét azok árának töredékéért is el lehetett érni. Korábban ha egy szervezet egy ilyen szuperszámítógépet szeretett volna beszerezni, akkor egy több millió dolláros számlára számíthatott.
Egy ugyanekkora teljesítménnyel bíró *Beowulf klasztert* viszont már néhány százezerből is meg lehetett építeni, ami még mindig nem kevés, de az előbbi összeghez viszonyítva szinte baráti ár. Ha vetünk egy pillantást a *TOP500*-as listára (a világ ötszáz leggyorsabb szuperszámítógépe), láthatjuk, hogy ez az egyszerű ötlet mekkora hatással volt a számítástechnika fejlődésére. A *Beowulf klaszterek* két legfontosabb közös tulajdonsága hogy közönséges, vagyis boltban bárki által megvásárolható alkatrészekből állnak, valamint hogy *Linux* fut rajtuk. Elterjedésüknek amúgy volt egy – eredetileg talán nem is sejtett – hatása is: szerte a világon megmozgatták a legjobb programozók fantáziáját.
Ennek ellenére számos ember a mai napig úgy gondolja, hogy a *Beowulf klaszterek* a mindennapi munkára alkalmatlanok, vagy legalábbis nem könnyű megtalálni a helyüket ezen a téren. Ebben persze van némi igazság is. Én például garantáltan nem adnék pénzt a *Quake 4* egy olyan változatáért, ami képes egy ilyen klaszteren futni. Az igazi felhasználások spektrumának egyik végén jelenleg az olyan filmes vállalkozások állnak, mint például a *Pixar*, amelyek a legújabb filmekben alkalmazott digitális trükköket ilyen klasztereken számoltatják ki, a másikon pedig azok a tudósok, akik a magreakcióktól kezdve az emberi genomig megszámlálhatatlanul sok dolgot kutatnak a segítségükkel. Bátran állíthatjuk tehát, hogy a fő felhasználások valóban elég távol esnek a mindennapi élettől. Ugyanakkor jó hír, hogy némi programozási tudással a klasztereket nem csak a tudósok és a hollywoodi stúdiók tudják kihasználni, hanem akár mi, közönséges földi halandók is.
Persze mielőtt párhuzamosítani kezdenénk egy alkalmazást, előbb nem árt elgondolkodni azon, mekkora is lesz a nyereség. Párhuzamos programot az ember általában azért ír, mert a feldolgozandó adatmennyiség nem fér el egyetlen *PC* memóriájában, vagy mert az elvégzendő számítás túlságosan hosszú ideig tartana, ha egyetlen processzor hajtaná végre. Mármost ha egy program párhuzamos változata egy másodperccel rövidebb idő alatt fut le, azért szinte biztosan nem éri meg az algoritmus átírásával tölteni az időnket. Ugyanakkor – amint azt a cikkben bemutatott példával demonstrálni fogom – jócskán akadhatnak olyan helyzetek is, amikor a párhuzamosítás egészen kis munka befektetésével elvégezhető, az eredmény pedig kifejezetten látványos.
Van az algoritmusoknak egy egész csoportja, amelyek bár olyan, első látásra erősen különböző területekről származnak mint a képfeldolgozás vagy a hangjelek átalakítása, ugyanazokkal a módszerekkel ugyanolyan könnyen felbonthatók részfeladatokra. A cikkben ezt a felbontási módszert fogom bemutatni: egy *Tux*-ot ábrázoló képre fogunk alkalmazni egy egyszerű konvolúciós szűrőt.



1. ábra A megoldandó probléma kisebb részekre való felosztása az úgynevezett tartománydekompozíció. Ezt a műveletet általában az itt láthatóhoz hasonlóan szokás végrehajtani.

időszakban meglehetősen sokak figyelme fordult a *MOSIX* és *openMOSIX* megoldások felé, de általánosságban elmondható, hogy ezeket elsősorban nem kifejezetten klaszterekre íródott programok futtatására használják, hanem arra, hogy áthidalják a szakadékot a szekvenciális és párhuzamos világ között. Szintén közös jellemzőjük, hogy működésük során a többszálú programok szálait osztják szét fizikailag elkülönült csomópontok között. Ebben a cikkben a továbbiakban feltételezem, hogy az olvasó rendelkezik egy telepített és működő *MPI* rendszerrel, bár a párhuzamosítás logikája a *PVM* használatakor is teljesen hasonló. Akinek esetleg teljesen új az *MPI*, és még soha nem telepített ilyen rendszert, az olvassa el a *Linux Journal* webhelyén *Stan Blank* és *Roman Zaritski* cikkét a témáról. Ők ketten kiválóan leírták, hogyan kell egy *MPI* rendszert üzembe helyezni.

A program inicializálása

Valamennyi *MPI* program elején kötelezően végre kell hajtanunk néhány olyan rutint, amelyek beállítják a csomópontok közti kommunikációt, és megállapítják minden egyes csomópont rangját (*rank*). A rang egy egész szám, amely a kérdéses gépet egyedileg azonosítja a klaszterben. A számozás nullától indul, és a gépek száma mínusz egyig folytatódik. A nullás rangú csomópont általában a klaszter központja, vagyis ez irányítja az összes többi csomópont munkáját is. Aztán ha a csomópontok mindegyike elvégezte a rá kiszabott feladatot, akkor – szintén kötelezően – végre kell hajtanunk egy záró függvényhívást is, mielőtt a program kilépne. Egy *MPI* program váza tehát a következőképpen fest:

```
#include <mpi.h>
#include <stdlib.h>
int main (void) {
          int myRank, clusterSize;
          int imgHeight, lowerBoundY,
          ➥upperBoundY,
               boxSize;
         // Initialize MPI
        MPI_Init((void *) 0, (void *) 0);
        // Get which node number we are.
        MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
        // Get how many total nodes there are.
        MPI_Comm_size(MPI_COMM_WORLD,
          ➥&clusterSize);
        // boxSize - the amount of the image
        //each node
        //                  will process
        boxSize = imgHeight / clusterSize;
         // lowerBoundY - where each node starts
         //processing.
         lowerBoundY = myRank*boxSize;
        // upperBoundY - where each node stops
        //processing.
        upperBoundY = lowerBoundY + boxSize;
        // Body of program goes here
        // Clean-up and exit:
        MPI_Finalize(MPI_COMM_WORLD);
        return 0;
}
```

Ez a kód valamennyi csomóponton önállóan fut, vagyis a `lowerBoundY` és az `upperBoundY` értéke minden gépen más és más lesz. Ezt a következő szakaszban természetesen ki is fogjuk használni.

A feldolgozandó kép felbontása

Ha egy digitálisra képre egy konvolúciós szűrőt alkalmazunk, a művelet percekig, vagy akár órákig is eltarthat a szűrő bonyolultságától, a kép nagyságától és a számítógép sebességétől függően. Ezen egy klaszter birtokában nyilván úgy segíthetünk, ha a képet kisebb darabokra bontjuk, és ezeket szétosztjuk több számítógép között. Az *1. ábrán* ennek a legegyszerűbb és ezért leggyakrabban alkalmazott módját láthatjuk: a képet csíkokra bontjuk. Ha tehát van egy nagy méretű digitális képünk, akkor *C/C++* nyelv használatát feltételezve a probléma particionálása a következőképpen oldható meg:

```
FILE *imageFile = fopen("image_in.ppm", "rb");
// Safety check.
if (imageFile != NULL) {
   // Read in the header.
   fread(imageHeader, sizeof(char),
                    HEADER_LENGTH, imageFile);
   // fseek puts us at the point in the image
   // that this node will process.
   fseek(imageFile, lowerBoundY*WIDTH*3,
             SEEK_SET);
   // Here is where we read in the colors:
   // i is the current row in the image.
   // j is the current column in the image.
   // k is the color, 0 for red, 1 for blue,
   //    and 2 for green.
   for (i=0; i<boxSize+1; i++) {
      for (j=0; j<WIDTH; j++) {
         for(k=0; k<3; k++) {
           fread(&byte, 1, 1, imageFile);
            pixelIndex = i*WIDTH+j+k;
            origImage[pixelIndex] = byte;
         }
      }
   }
}
fclose(imageFile);
```

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

2. ábra Az élszűrés (edge detect) művelet ezzel a konvolúciós mátrixszal írható le. A piros négyszög az éppen feldolgozás alatt álló pixelnek fele meg, a többi szám pedig a szomszédos pixelek értékének felel meg.

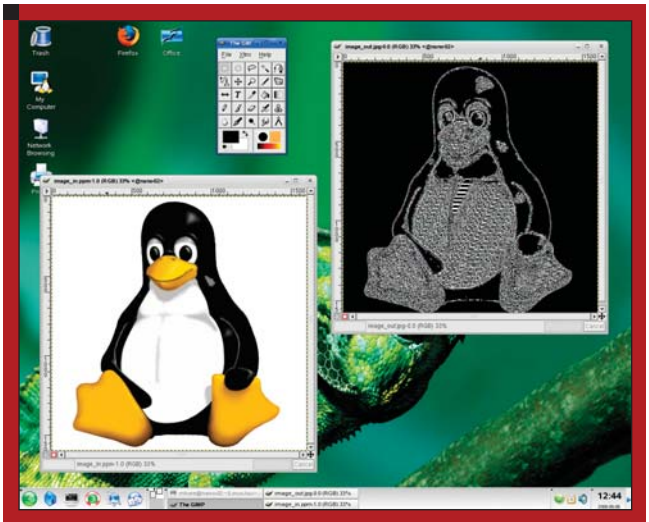
A szűrő alkalmazása

Most, hogy már mindegyik csomópont megkapta a képnek azt a részét, amit neki kell feldolgoznia, nekiláthatunk a tényleges munkának, vagyis a szűrő alkalmazásának. Hogy hogyan is kell végigszámoltatni egy konvolúciós szűrőt, az kiválóan le van írva a *GIMP* dokumentációjában. Ami azt illeti számos olyan képfeldolgozási eljárás van, amelyek tulajdonképpen egy-egy ügyesen kitalált konvolúciós mátrixnak feleltethetők meg. Ilyen például az élesítés (***sharpen***), az elmosás (***blur***), a Gauss-féle elmosás (***Gaussian blur***), az élszűrés (***edge detect***) vagy az élkiemelés (***edge enhance***). A konvolúciós szűrő úgy működik, hogy minden egyes pixel értékét a saját és a szomszédai értékének függvényében változtatja meg. Ebben a cikkben az élszűrés (***edge detect***) szűrőt fogjuk alkalmazni. Az ehhez tartozó mátrixot a *2. ábra* mutatja.

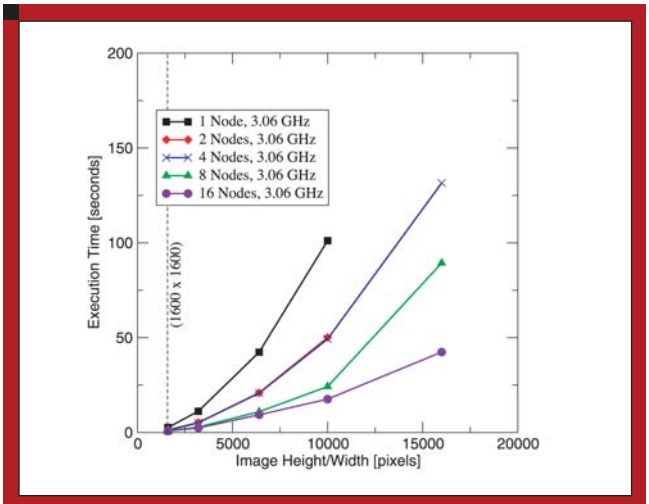
A szűrő alkalmazása jelen esetben azt jelenti, hogy minden egyes pixel értékét megszorozzuk -4-gyel, majd az így kapott értékhez hozzáadjuk a fölötte, alatta, valamint a tőle jobbra és balra levő pixelek értékét. Ez lesz az adott pixel új értéke. Mivel a mátrix sarkaiban csupa nulla van, ezért a hatékonyság növelése érdekében a számítások során nem is vesszük figyelembe valamennyi mátrixelemet, ami szigorú matematikai értelemben csalás ugyan, de esetünkben az eredményen mit sem változtat. Az alábbi kódrészlet a szűrő alkalmazását, míg a *3. ábra* a végeredményül kapott képet mutatja:

```
for (i=0; i<boxSize; i++) {
   for (j=0; j<WIDTH; j++) {
      if (i>0 && i<(HEIGHT-1) &&
          j>0 && j<(WIDTH-1)){
        // Now we apply the filter matrix
        // First to the current pixel.
        pixelIndex = i*WIDTH + j;
        r = origImage[pixelIndex];
        g = origImage[pixelIndex+1];
        b = origImage[pixelIndex+2];
        filter_r = -4*r;
        filter_g = -4*g;
        filter_b = -4*b;
        // Next to the left neighbor.
        pixelIndex = i*WIDTH + j - 1;
        r = origImage[pixelIndex];
        g = origImage[pixelIndex+1];
        b = origImage[pixelIndex+2];
        filter_r += 1*r;
        filter_g += 1*g;
        filter_b += 1*b;
        // Next to the right neighbor.
        pixelIndex = i*WIDTH + j + 1;
        r = origImage[pixelIndex];
        g = origImage[pixelIndex+1];
        b = origImage[pixelIndex+2];
        filter_r += 1*r;
        filter_g += 1*g;
        filter_b += 1*b;
        // The neighbor above.
        pixelIndex = (i-1)*WIDTH + j;
        r = origImage[pixelIndex];
        g = origImage[pixelIndex+1];
        b = origImage[pixelIndex+2];
        filter_r += 1*r;
        filter_g += 1*g;
        filter_b += 1*b;
        // The neighbor below.
        pixelIndex = (i+1)*WIDTH + j;
        r = origImage[pixelIndex];
        g = origImage[pixelIndex+1];
        b = origImage[pixelIndex+2];
        filter_r += 1*r;
        filter_g += 1*g;
        filter_b += 1*b;
      }
      // Record the new pixel.
      pixelIndex = i*WIDTH + j;
      filterImage[pixelIndex]   = filter_r;
      filterImage[pixelIndex+1] = filter_g;
      filterImage[pixelIndex+2] = filter_b;
   }
}
```

A `readImage()` rutinnak természetesen kell legyen egy `writeImage()` megfelelője is, amely lemezre írja a kép részleteit.



3. ábra Balra az eredeti kép látható, míg a jobb oldali ábra az élszűrés eredményét mutatja



4. ábra A működési idő függése a kép méretétől és a klaszter csomópontjainak számától. A kép mérete 1.600x1.600 pixeltől 16.000x16.000 pixelig változott. Így a legnagyobb kép feldolgozásához legalább négy csomópontból álló klaszterre volt szükség.

A kód lefordítása és futtatása

Az *MPI* mindkét, *Linux* alatt elérhető megvalósítása (*LAM* és *MPICH*) tartalmaz olyan szkripteket, amelyek segítségével a felhasználó könnyebben fordíthatja le az általa írt alkalmazást úgy, hogy ahhoz a megfelelő *MPI* könyvtárak is hozzálinkelődjenek. Ezekkel tulajdonképpen a megfelelő kapcsolókat adhatjuk át a *GCC*-nek pont ugyanúgy, ahogy azt általában is tesszük. Az egyes nyelvekhez a következő szkriptek használhatók:

`mpicc` : *C* nyelvű programok
`mpi++` : *C++* programok
`mpif77` : *FORTRAN 77* programok

Az elkészült kódot az `mpirun` parancs segítségével futtathatjuk. Ha tehát egy programot ***parallel.c***-nek hívnak, akkor a fordítását a

```
mpicc -O3 -o parallel parallel.c
```

paranccsal, míg a futtatását a

```
mpirun n0 ./parallel
```

paranccsal végezhetjük. Utóbbiban az `n0` azt jelenti, hogy a programot kizárólag a nullás rangú csomóponton kell futtatni. Ha több csomópont között akarjuk elosztani a munkát, akkor egy nyolc processzoros rendszer esetében az `n0...n7` értékeket használhatjuk. Ha pedig azt akarjuk jelezni a rendszernek, hogy az összes aktuálisan rendelkezésre álló számítási kapacitást ki akarjuk használni, akkor az

```
mpirun C
```

parancsot kell használnunk.

Mekkora többletteljesítményt nyertünk?

Most tehát ott tartunk, hogy néhány egészen egyszerű *MPI* hívással párhuzamosítottunk egy olyan programot, ami egy konvolúciós szűrőt tud alkalmazni egy digitális képre. A lényegesebb kérdés mostantól nyilván az, hogy miért is érte ez meg nekünk? Vajon mennyivel nagyobb teljesítményt nyertünk? A nyereségnek persze többféle mértékegysége lehet attól függően, hogy kinek mi a fontosabb. Nyerhetünk teljesítményt úgy, hogy gyorsabban fut le a programunk, de az is előfordulhat, hogy valakinek nem ez az igazán lényeges szempont, hanem hogy mennyi számítást tud egyszerre elvégeztetni a rendszerével. Ha például van egy 16.000x16.000 pixel méretű digitális képünk, akkor ennek a betöltéséhez egy 768.000.000 elemű tömbre lesz szükségünk. Ez pedig egyszerűen túl nagy. A *GCC*-től csak egy nyájas hibaüzenetet fogunk kapni, hogy nem tud ekkora tömböt létrehozni. (***A probléma máshogyan persze megoldható – a szerk.***) Ha viszont az imént bemutatott módon fölszabdaljuk a képet több kisebb csíkra, az egyes darabok már kezelhetőnek bizonyulnak.
A cikkben bemutatott kódot egy 16 csomópontból álló *Beolwulf* klaszteren teszteltem. Minden csomópontnak ***1 GB*** memóriája és ***3.06 GHz***-es ***Pentium 4*** processzora volt, és valamennyin ***Fedora Core 1*** futott. Az összeköttetést ***Gigabit Ethernet*** biztosította, a csomópontok közti adatcsere pedig egy ***NFS*** partíció segítségével volt megoldva. A kép beolvasásához, feldolgozásához és lemezre való visszaírásához szükséges idő nagyságát a *4. ábrán* láthatjuk.
Amint azt a *4. ábrán* jól látható, a feldolgozás párhuzamosítása már a kisebb méretű képek esetében is jelentős gyorsulást eredményezett, az igazi előny azonban a kifejezetten nagy képeknél látható. Ami azt illeti, a 10.000x10.000 pixelnél nagyobb képek esetében maga a feldolgozás szekvenciális üzemmódban el sem végezhető a korábban említett memóriakorlát miatt, így ezekben az esetekben egy legalább négy csomópontból álló klaszteren kellett a futtatást végezni. Az ábráról azt is leolvashatjuk, hol volt értelme a párhuzamosításnak, és hol nem. Ami azt illeti, 1.600x1.600 pixeltől 3.200x3.200 pixelig nincs különösebben nagy eltérés a futás sebességében. Ezek a képek ráadásul annyira kicsik, hogy a memória sem jelent korlátot, vagyis a párhuzamosítás még ezzel az érvvel sem támasztható alá.
Hogy a kvalitatív értékelésen túl néhány számot is mondjak, egyetlen ***3,06 GHz***-es processzorral szerelt gépen egy 6.400x6.400 pixeles kép beolvasása, feldolgozása és lemezre való visszaírása körülbelül 50 másodpercig tart.
Egy ugyanilyen csomópontokból álló 16 processzoros klaszter ezzel szemben 10 másodperc alatt végez ugyanezzel a művelettel. Mi több, egy 16 processzoros klaszter még a 16.000x16.000 pixeles képpel is hamarabb végez, mint az egyprocesszoros gép a 6,25-szor kisebb képpel.

Lehetséges gyakorlati alkalmazások

Ebben a cikkben a nagy teljesítményű *Beowulf* klaszterek felhasználásának csupán egyetlen lehetőségét tudom bemutatni, ugyanakkor az alkalmazott alapelvek és módszerek mindenütt azonosak. A feldolgozandó adatok szintjén párhuzamosítható alkalmazások mindegyike pontosan úgy működik, mint a most bemutatott képfeldolgozó eljárás: minden csomópont beolvassa az adatok egy részét, feldolgozza azokat, majd vagy visszaküldi az eredményt a központi csomópontnak, vagy maga írja ki azt lemezre. A következőkben felsorolok négy olyan területet, ahol a párhuzamosításra meglátásom szerint nagy jövő vár.

Néhány alapvető és hasznos MPI szubrutin

Az *MPI* könyvtár több mint 200 függvényből áll, és mindegyik hasznos bizonyos helyzetekben. Azért van ilyen sok eleme ennek az interfésznek, mert az *MPI* számos különböző architektúrán képes működni, és így meglehetősen sokféle igénynek kell megfelelnie egyszerre. A következőkben felsorolom közülük azt a néhányat, amelyek a cikkben bemutatotthoz hasonló párhuzamos algoritmusok megvalósítása során hasznosak lehetnek.

`MPI_Init((void*) 0, (void*) 0)` – Inicializálja az *MPI* rendszert.
`MPI_Comm_size(MPI_COMM_WORLD, &cltSize)` – A `cltSize` (egész) változóban visszaadja a klaszter méretét.
`MPI_Comm_rank(MPI_COMM_WORLD, &myRank)` – A `myRank` (egész) változóban visszaadja a kérdéses csomópont rangját.
`MPI_Barrier(MPI_COMM_WORLD)` – Megállítja a végrehajtást mindaddig, amíg a klaszter valamennyi csomópontja el nem jut a kódnak erre a pontjára.
`MPI_Wtime()` – Visszaadja egy önkényes, múltbeli időpillanat óta eltelt időt. A szubrutinok és egyéb folyamatok időzítésére használható.
`MPI_Finalize(MPI_COMM_WORLD)` – Leállít valamennyi *MPI* folyamatot. Ezt a rutint valamennyi programnak meg kell hívnia leállás előtt.

1. ***Képszűrők:*** Amint azt a fenti példa is jól demonstrálta, a párhuzamos feldolgozás kiválóan használható a képkezelő eljárások terén, hiszen nem csak felgyorsítja a folyamatokat, hanem lehetőséget ad kifejezetten nagy képek kezelésére is. Éppen ezért nyilván komoly előrelépést jelentene, ha az olyan képfeldolgozó alkalmazásokhoz, mint amilyen például a *Gimp* elkészülne egy párhuzamos szűrőket tartalmazó csomag.

2. ***Hangfeldolgozás:*** Szűrési módszereket nem csak a képek, hanem a hangfájlok feldolgozásában is használnak, és ezek is jelentős feldolgozási időt igényelnek. Éppen ezért az olyan nyílt forrású alkalmazások, mint az *Audacity* szintén sokat profitálhatnak a párhuzamosítási módszerek alkalmazásából.

3. ***Adatbázis műveletek:*** A kifejezetten nagy mennyiségű adat feldolgozásával járó műveletek párhuzamosíthatók, és ez által felgyorsíthatók úgy, hogy az egyes csomópontok olyan részlekérdezéseket futtatnak, amelyek a szükséges adatoknak csak egy részét adják vissza. Ez után valamennyi csomópont el is végezheti az általa megszerzett adatokon a szükséges műveleteket.

4. ***Biztonsági rendszerek:*** A klaszterek segítségével a rendszergazdák sokkal gyorsabban győződhetnek meg arról, mennyire biztonságos jelszavakat használnak a felhasználók. A */etc/shadow* tartalmának nyers erővel történő visszafejtése köztudottan elég időigényes, ha azonban a feladatot célszerűen szétosztjuk egy klaszter csomópontjai között, sokkal gyorsabban is lefut a teszt. Ezzel aztán nem csak időt nyerünk, hanem a lelkünk nyugalma is visszatérhet, hiszen pontos képet kapunk rendszerünk biztonságáról.

Záró megjegyzések

Őszintén remélem, hogy ezzel a cikkel sikerült rámutatnom, miért gondolom úgy, hogy a párhuzamos programozásnak a mindennapi életben is hely van. Fel is soroltam néhány olyan területet, ahol a párhuzamosítás előnyeit feltehetőleg már a közeljövőben ki fogják használni a fejlesztők és a felhasználók egyaránt. Egyúttal arról is meg vagyok győződve, hogy ilyen potenciális alkalmazási terület még számos akad.

Létezik néhány olyan alapelv, amelyek betartása általában biztosítja azt, hogy a párhuzamos kód hatékonyabban működhessen, mint szekvenciális előde. Az első ilyen ökölszabály a hálózati adatforgalom minimális szinten tartása. A csomópontok közti adatcsere általában sokkal több időt vesz igénybe, mint a magukon a gépeken elvégezhető műveletek. A fenti bemutatott példában a csomópontok között egyáltalán nem volt kommunikáció, hiszen az általuk végrehajtott műveletek logikailag teljesen függetlenek voltak egymástól. Ugyanakkor ez inkább a kivétel, nem a szabály. A legtöbb esetben a csomópontoknak a számítások során is adatokat kell cserélniük. A második szabály a bemenő adatok kezelésére vonatkozik: ha egy csomópontnak adatokat kell beolvasni a lemezről, csak annyit adatot olvastassunk be vele, amennyire ténylegesen szüksége van. Ezzel nyilván hatékonyabban használhatjuk a memóriát és a művelet sebessége is nő. Végezetül legyünk óvatosak olyan esetekben, amikor a csomópontoknak szinkronban kell maradniuk a számítások során, ez ugyanis a klaszterekben nem történik meg automatikusan. Egyes gépek kicsit gyorsabban működnek, míg mások lassabban, vagyis a szinkronizációval kapcsolatban nem élhetünk semmiféle előfeltevéssel. A keretes részben összefoglaltam néhány olyan *MPI* rutint, amelyekkel könnyen megoldhatjuk a csomópontok szinkronizációját.
Azt gondolom, hogy a jövőben a klaszterek egyre nagyobb szerepet játszanak majd mindennapi életünkben is. Egyúttal remélem, hogy ezzel a cikkel sikerült meggyőznöm az olvasót arról, hogy a párhuzamos alkalmazások fejlesztése nem különösebben ördöngös feladat, másrészt pedig az így kapott teljesítménytöbblet messze megér a befektetett munkát, és lehetőséget teremt egészen különleges területeken való alkalmazásra is.
Végezetül szeretném köszönetemet kifejezni *Dr. Mohamed Larajdinak*, aki lehetővé tette, hogy programjaimat a *Memphisi Egyetem Fizika Tanszékének Beowulf* klaszterén teszteljem.

Linux Journal 2006. szeptember, 149. szám

Kapcsolódó anyagok: ➲*www.linuxjournal.com/article/9135*

Michael-Jon Ainsley Hore

Bevezetés a GNU Autoconf rendszer használatába

Bizonyára mindenki ismeri a három parancsot, amellyel egy *.tar.gz fájlból kicsomagolt programot telepíteni lehet: configure, make és make install. E rendszer alapja a GNU Autoconf, amellyel áttekinthetővé tehetjük a fejlesztett programunk forráskódját, és akár egyszerű telepítőkészletet is készíthetünk segítségével.

Bevezetés

Az első, legegyszerűbb programokat parancssorból fordították, a módosítások után kézzel elindítva a fordítót. Később, amikor már több forrás fájlból állítottak össze egyetlen binárist, a fordítási idők lecsökkentéséhez kitalálták a ***Makefile***-okat, amelyekben a programokhoz egy függőségi fa írható le, amely meghatározza, melyik futtatható állományhoz melyik forrásokat kell lefordítani illetve összeszerkeszteni; illetve hogyan, milyen paraméterekkel kell a fordítót indítani. A legnagyobb programoknál persze még ezzel is sok munka van arról nem is beszélve, hogy a programok hordozhatóságára sem ad semmilyen megoldást.
Ma már legtöbbször a ***Makefile***-t sem szokás kézzel írni. Több olyan program is van, amely nagy, több könyvtárból álló forráskódok, összetett programrendszerek kezelését könnyíti meg. Ilyenek a ***Qt*** programok által használt ***qmake***, az ***XFree86 imake*** programja, illetve a ***GNU autoconf***. (Az utóbbi időben az ***X.Org*** is már az ***autoconfot*** használja.)
A ***Linuxot*** használók legtöbbször programok telepítése közben találkoznak az ***autoconf*** rendszerrel. Mindenkinek ismerős a három begépelendő utasítás, a ***configure***, a ***make*** és a ***make install***, amelyekkel az ***Internetről*** forráskódban letöltött programokat installálni tudjuk. Ebben a cikkben azokkal a programokkal foglalkozunk, amelyekkel a ***configure*** parancsfájlt előállíthatjuk.
Az ***autoconf*** rendszernek több haszna is van.

1. lista Helló, világ! GTK+ módra: hello.c

```
#include <gtk/gtk.h>
#include "config.h"

int main(int argc, char *argv[])
{
    GtkWidget *window;

    gtk_init(&argc, &argv);

    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    g_signal_connect(G_OBJECT(window), "delete-event",
    ➥G_CALLBACK(gtk_main_quit), NULL);
    gtk_window_set_title(GTK_WINDOW(window), PACKAGE_STRING);
    gtk_container_set_border_width(GTK_CONTAINER(window),100);

    gtk_container_add(GTK_CONTAINER(window),
    ➥gtk_label_new("Helló, világ!"));
    gtk_widget_show_all(window);
    gtk_main();
    return 0;
}
```

- Segítségével könnyebben biztosítható a programunk hordozhatósága. Egyszerűen használható makrókat ad a programozó kezébe, amelyekkel meghatározható fordítás közben a cél operációs rendszer, illetve a fordításhoz és futtatáshoz szükséges eszközöket, például fordítóprogramokat, könyvtárakat is automatikusan képes megkeresni.
- Egyszerű telepítőt készíthetünk vele a programunkhoz. Ez a felhasználóknak is könnyebbség, ugyanis minden ***autoconffal***

2. lista A configure.in fájl

```
AC_INIT(Hello, 0.1)
AC_CONFIG_SRCDIR([hello.c])
AM_INIT_AUTOMAKE
AM_CONFIG_HEADER(config.h)

AC_PROG_CC
AC_ISC_POSIX
AC_HEADER_STDC

AM_PROG_INSTALL

AM_PATH_GTK_2_0(2.8.0, :,
➥AC_MSG_ERROR(Test for GTK+
➥failed. See the file
➥'INSTALL' for help.))

AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

készített programot ugyanazon a módon lehet telepíteni.
Ha a rendszer ismerős, még egy tapasztalatlanabb felhasználó is egyből tudja, hogyan használja azt. Nem is beszélve az automatizálhatóságról, gondoljunk csak a *Gentoo Portage* felületére vagy a *BSD portsra*.

Helló, világ! program autoconffal

Úgy döntöttem, hogy a cikkben a rendszert inkább egy kész programon keresztül mutatom be, egy *tutorial*-szerű leíráshoz ugyanis minden fájlból három-négy különböző verziót kellene prezentálni, miközben alig lenne köztünk néhány szó különbség.
A példa megszokott „Helló, világ!" kiíratása, mégpedig annak a GTK+ változata. A forráskód az első listán látható. Gépeljük be, és mentsük el *hello.c* néven, illetve írjuk be a *Makefile.am* és a *configure.in* fájlokat is, elhelyezve őket a forráskóddal egy könyvtárban!
Ha ezt a három fájlt megírtuk, indítsuk el a rendszer egyes programjait, amelyek elkészítenek néhány fájlt:

```
aclocal
autoheader
automake --add-missing
autoconf
```

3. lista A Makefile.am fájl

```
LDADD = @GTK_LIBS@
AM_CFLAGS = @GTK_CFLAGS@
➥-Wall

bin_PROGRAMS = hello

hello_SOURCES = hello.c
```

Amikor először elindítjuk, figyeljük meg: az *automake* program jelez, hogy néhány fájl (*AUTHORS*, *README*, *NEWS*, és *ChangeLog*) hiányzik. A fájlok elvileg részei egy *GNU* kompatibilis szoftver terjesztésnek. Ezekben a program leírása található, illetve különböző információk, amelyek a felhasználók számára érdekesek lehetnek: a szerzők nevei és *e-mail* címei, a program újdonságai, változtatásai az előző verziókhoz képest stb. Néhány hasonló fájlt az *automake* az `--add-missing` argumentum hatására automatikusan létrehozott helyettünk: a *COPYING* fájlba másolta például a *GNU GPL*-t. Készítsük el a szükséges hiányzó szövegfájlokat, vagy legalábbis hozzunk létre egyelőre üres fájlokat helyette! Ha megvagyunk, akkor adjuk ki újra az `automake --add-missing` parancsot, és utána folytassuk az `autoconf` beírásával! A folyamat elég sok fájllal gazdagít minket, legtöbbjükkel szerencsére nem kell foglalkoznunk, ugyanis a rendszer automatikusan kezeli őket.
Ha mindennel elkészültünk, már működik is a megszokott *GNU* módon működő szoftver terjesztésünk! Próbáljuk is ki:

```
./configure
make
make install    # rootként
hello
make uninstall    # rootként
make dist
```

Ha a rendszerhez, ahol dolgozunk, van adminisztrátori jogosultságunk, kipróbálhatjuk a szoftver csomagunk telepítését illetve törlését is (`make install` illetve `make uninstall`). Ez alapértelmezés szerint az */usr/local/bin/hello* fájlt hozza létre.
Ha ezt kihagyjuk, akkor indításnál használjuk inkább a `./hello` parancsot!
Lássuk sorban, mit is jelentenek az egyes fájlokban beírt dolgok!

A configure.in fájl

Ez tulajdonképpen egy *Bourne* héjprogram, amelyet az *m4* makrófeldolgozó kezel. Ezért lehetséges a megszokott `if...then...fi` és hasonló szerkezeteket is használni, de ezekre általában nincsen szükség. (Fontos, hogy a szögletes zárójeleket az *m4* előfeldolgozó kezeli, ezért a *test* program nevét minden esetben ki kell írni, vagyis helytelen a `if [ -f fájlnév ]...` utasítás! A helyes működéshez az `if test -f fájlnév...` szükséges.) Ebből lesz a néha több száz kilobájtosra is megnövő *configure* program, amely a forráskód környezetbe beillesztését és a *Makefile* készítését végzi majd a cél környezetben.
Az egyes bonyolult műveleteket a makrókkat végezhetjük el. Ezeket gyűjti össze az *aclocal* nevű program, ezért kellett azt elsőnek elindítani. A makrók feladatai a következőek:

- `AC_INIT`: Elindítja az *autoconf* rendszert. Mindig ez kell legyen az első makró. Megadhatjuk neki a program nevét, verziószámát, és esetleg további argumentumai is lehetnek, pl. egy e-mail cím, ahova a felhasználók hibajelentéseket küldhetnek.

- `AC_CONFIG_SRCDIR`: Ennek a makrónak adjuk meg a forráskód egy fájlját. Most csak egyetlen egy van, a *hello.c*.

- `AM_INIT_AUTOMAKE`: Elindítja az *automake* rendszert.

- `AM_CONFIG_HEADER`: Megadja, hogy melyik fájlban tároljuk a rendszer által létrehozott definíciókat. Erről bővebben később.

- `AC_PROG_CC`: Megkeresi a rendszer C fordítóját. *Linuxon* ez általában a *gcc*.

- `AC_HEADER_STDC`: Ellenőrzi, hogy megvannak-e a szabványos C

header fájlok. (Ilyen a *hello.c*-ben pont nem szerepel, de elég gyakori a makró.)

- `AC_PROG_INSTALL`: Megkeresi a rendszeren az *install* nevű programot. (Ha nem talál ilyet, akkor az *install.sh* héjprogramot használja majd, amelyet az *automake* hozott létre, ugyanis nem mindegyik operációs rendszerben van ilyen.)

- `AM_PATH_GTK_2_0`: Megkeresi a rendszeren található *GTK+* könyvtár *header* fájljait, könyvtárait, és meghatározza, hogy a C fordítónak illetve szerkesztőnek milyen paramétereket kell átadni *GTK+*-os programok készítéséhez.

- `AC_CONFIG_FILES`: Ennek a makrónak kell megadni, hogy a *configure* héjprogram melyik fájlokat hozza létre.

- `AC_OUTPUT`: Létrehozza az előbb megadott fájlokat.

Általában egy programhoz egy *configure.in* fájlt kell írnunk, annak fő könyvtárában. Alkönyvtárankénti *configure.in* fájlokat csak nagy, több részből szerkesztett programcsomagoknál szokás használni. Ha mindenképpen erre van szükség, érdemes az *autoconf* leírását elolvasni.

Autoheader

A második elindított program az *autoheader* volt. A *configure* szkript sok változót összegyűjt a fordítási, futtatási környezetről, amelyeket a C forráskódban `#ifdef` paranccsal, illetve makróként használhatunk. Részben ezt teszi a forráskódot hordozhatóvá, és könnyen kezelhetővé. A definíciókból viszont elég sok lehet, érdemes inkább őket egy külön beleértett (*include)* fájlban tárolni. Korlátozott amúgy a parancssorok hosszúsága is, és könnyen túlléphetnénk azt a C fordítónak átadott sok `-D` paraméterrel.
Az ilyen beleértett fájl készítését támogatja az *autoheader*, illetve *autoconf* `AC_CONFIG_HEADER` makrója. A fájl szokásos neve *config.h*, ezt a forráskódba a `#include "config.h"` sorral illeszthetjük be. Elvileg ehhez is készítenünk kellene egy *config.h.in* bemeneti szövegfájlt, azonban ezt az *autoheader* elvégzi helyettünk. (Az *autoconf* a definíciókat az */usr/share/autoconf/acconfig.h* fájlból másolja. Ha valamilyen makró eltérő bejegyzést igényel, akkor egy *acconfig.h* nevű fájlt kell készítenünk, egy könyvtárban a *configure.in*-nel, és az *autoheader* figyelembe fogja venni.)
Érdemes egyébként fordítás után megvizsgálni a *config.h* fájlt. Láthatjuk például, hogy a program neve, verziószáma és még sok minden más makróként használható.
A *hello.c* programban az ablak nevét egy ilyen alapján állítjuk be a `gtk_window_set_title(...)` függvényhívással. Ezeket a C előfeldolgozó (*preprocessor*) helyettesíti be.

Automake

Az *autoconf* rendszer egyik célja, hogy minden programot ugyanolyan módon kelljen telepíteni. Közismertek a `make` és a `make install` parancsok, ezeken kívül sok kevésbé ismert is van: `make distclean`, `make dist`... Ezek minden *GNU* kompatibilis szoftver terjesztésnek részei.
Az *automake* egy adott *Makefile.am*-ből készíti el a *Makefile.in*-t, amelyből a futtatási környezetben elkészül később az a *Makefile*, amely támogat minden megszokott funkciót. Ez a fájl tulajdonképpen csak néhány változót tartalmaz, amelyeket az *automake* beilleszt egy előre elkészített *Makefileba.* A szintaxis ezért a megszokott: megyjegyzéseket a # karakter után írhatunk, a többsoros listákat pedig a \ karakterrel választhatjuk el.

Nézzük ezt is soronként!

- `LDADD`: ebbe a változóba tehetjük be, hogy a programok szerkesztéséhez (*linking*) milyen paraméterek szükségesek. Például a matematikai könyvtár (*libm*) beépítéséhez: `-lm`. Néhány változót a *configure.in* fájlban megadott makrók is beállítanak, azokat ide beírhatjuk: a *GTK+* használatához pl. `@GTK_LIBS@`.

- `AM_CFLAGS`: a C fordítónak átadandó paraméterek. Ezek hivatkozhatnak például könyvtárakra, ahol az *include* fájlokat keresi; a *GTK+* használatához ez `@GTK_CFLAGS@`, de emögé érdemes írnunk a `-Wall` paramétert is, amely a fordítás figyelmeztetéseit kapcsolja be.

- `bin_PROGRAMS`: azoknak a programoknak a neve, amelyet installálás után az */usr/local/bin*-ben (vagy */usr/bin*-ben) szeretnénk látni.

- `hello_SOURCES`: a hello nevű program előállításához használt forráskódok listája.

Egyéb gyakran használt változók:

- `sbin_PROGRAMS`: a programok, amelyek az sbin-be kerülnek.

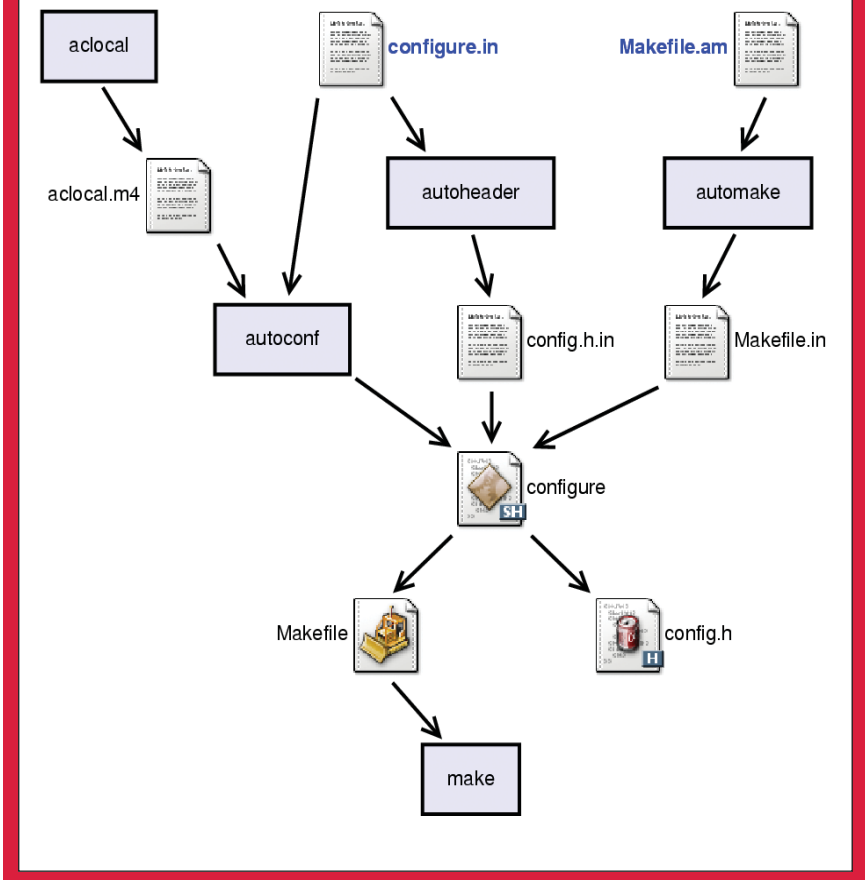
- `man_MANS`: kézikönyv (*manual*) oldalak.

- `noinst_HEADERS`: olyan *header* fájlok, amelyeket csak a fordítás közben kell használni.

- `EXTRA_DIST`: egyéb fájlok, amelyek a terjesztéshez tartoznak, például képek vagy más adatfájlok

Az autoconf rendszer programjai

A bemutatott rendszer elég összetett, érdemes egy ábrán áttekinteni, hogy melyik program mit csinál, és milyen fájlokat használ.
A használathoz tulajdonképpen két fájlt kell megírnunk. Az egyik a bemutatott *Makefile.am*, amely leírja, hogy milyen programokat kell a fordítónak létrehoznia, és hogy az egyes programokat melyik források alapján kell összeszerkesztenie. A fordításhoz szükségesek különböző paraméterek, amelyeket egyszerűbb, néhány soros *Makefile* vagy parancssori fordítás esetén a *pkg-config* programtól kérünk el; ezeket itt változóként kell megadni. A *Makefile.am* írja le azokat az egyéb fájlokat, amelyek a programhoz tartoznak, például képek, kézikönyv oldalak stb. Ebből a fájlból hozza létre az *automake* a *Makefile.in*-t, amely a szoftvercsomag része, és a *configure* héjprogram egyik bemeneti fájlja.



1. ábra Az autoconf rendszer programjai

A másik a *configure.in*. Ebben a szoftverünk által használt könyvtárakat, eszközöket soroljuk fel, illetve ez használható a fordítás folyamatának konfigurálására. Az *autoheader* program ennek alapján hozza létre a *config.h.in*-t, amelyből később egy *C* (vagy más) nyelvű *include* fájl lesz, amely a forráskódunk kezelhetőségét javítja. Az *autoconf* pedig ez alapján hozza létre a *configure* programot, amely már a cél környezetben fog futni.

Eddig tartott a programozó feladata. Ha egy felhasználó letölti, kicsomagolja a programot, akkor a telepítéshez először elindítja a *configure* programot. Ez a meglévő előfeldolgozott fájlok segítségével megvizsgálja a cél környezetet. (Ez akár teljesen más is lehet, mint ahol a programot eredetileg fejlesztették, *FreeBSD*, *MinGW*...) Létrehozza az abban a környezetben érvényes *config.h* fájlt és a *Makefile*-t. Utána indulhat a fordítás és a telepítés.

Az ábra alapján könnyen eldönthető az is, hogy az egyes bemeneti fájlok módosítása esetén melyik programokat kell újra futtatnunk. Szerencsére, ha már van egy elkészített *Makefileunk*, abba az *autoconf* elhelyez olyan szabályokat, amelyek szükség esetén az *automake* vagy *autoconf* programot automatikusan elindítják, így legtöbbször elég csak egy sima *make* parancs.

Terjesztések létrehozása

Van még néhány *Makefile* cél, amelyet eddig nem részleteztem. Az egyik a `make dist`, amely *.tar.gz* fájlba tömöríti a programcsomagunkat, amelyet egyből akár fel is tölthetünk a netre, hogy mások használhassák. A fenti példában a rendszer a *hello-0.1.tar.gz* fájlt hozza létre; a név és a verziószám természetesen megfelel a *configure.in* első sorában megadottnak. Ha ezt kicsomagoljuk, a jól ismert dolgot fogjuk látni, létrejön egy *hello-0.1*/ nevű könyvtár. Innentől pedig már álmunkból felkeltve is tudjuk a teendőt:

```
./configure && make && make
➥install
```

Terjesztés létrehozásakor érdemes egyébként inkább a `make distcheck` parancsot használni, ez ugyanis ki is próbálja a létrejövő fájlt, konkrétan konfigurálja, telepíti egy ideiglenes könyvtárba, aztán letörli. Így rögtön kiderül, ha valamelyik fájl kimaradt volna – főként az adatfájloknál (EXTRA_DIST) szokott ez előfordulni. Működnek egyébként a jól ismert `make clean` és `make distclean` parancsok is, ezekkel a fordítás, illetve a konfigurálás közben keletkezett fájlokat tudjuk törölni.

Autoconf kompatibilis programok

A legtöbb integrált fejlesztői környezet, mint például az *Anjuta*, eleve *autoconfra* épülő projektet hoznak létre. Használható ilyen célra az *autoproject* nevű program is. Ha pedig van egy meglévő forráskódunk, amely még nem használja ezt az eszközt, érdemes kipróbálni az *autoscan* nevű programot, amely egy kezdetleges *configure.in* fájl létrehozásában segít, megvizsgálva a forrást, hogy milyen szokásos nehezen hordozható részek vannak benne.

Czirkos Zoltán

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

KAPCSOLÓDÓ CÍMEK

Autoconf leírás:
➲ http://www.gnu.org/manual/autoconf/

Autoconf bevezető:
➲ http://www.seul.org/docs/autotut/

Autoproject:
➲ http://directory.fsf.org/autoproject.html

Anjuta:
➲ http://anjuta.sourceforge.net/

Ruby on Rails – Webprogramozás könnyedén

A Ruby on Rails-t sokan tekintik az internet programozás egyik zászlóshajójának. Vajon miért övezi ekkora hírverés ezt a mindössze két éves keretrendszert?

A *Ruby on Rails*, mindent tartalmaz egy csomagban, amire adatbázis alapú weboldalak kialakításához szükségünk lehet. Az indulás is egyszerű, hiszen telepítés után mindössze az adatbázis kapcsolat paramétereit kell beállítanunk és már fejleszthetünk is. Lehetővé teszi, hogy a fontos dolgokkal foglalkozzunk fejlesztés közben. A kevésbé fontos és ismétlődő feladatokat pedig nyugodtan rábízhatjuk.

Mindenek alapja a Ruby

Rails-al fejleszteni annyit tesz, mint megérteni és megszeretni a filozófiát ami mögötte áll. A keretrendszer alapját adó *Ruby* nyelv *Japánból* indult hódító útjára.(A ruby angolul rubintot jelent.) A *Ruby* teljesen objektum orientált script nyelv. Szintaktikája nagyon elegáns és egyszerű. Sebessége a *python*-nál kicsit lassabb, de bőven megfelelő weboldalak kiszolgálásához. A legcsodálatosabb jellemvonása, hogy segítségével bámulatosan olvasható kódot írhatunk. Álljon itt pár példa!

```
5.times { print "Hello Linux!" }
# Nem tesz mást, mint 5
alkalommal kiírja, hogy
"Helló Linux".

exit unless
➥"Linuxvilág".include? "nux"
# Kilép, ha a "Linuxvilág" nem
tartalmazza a "nux" szöveget.
```

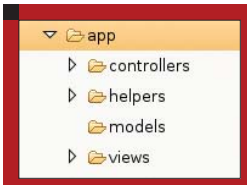
David filozófiája

David Heinemeier Hansson a *Rails* atyja. *David* szerint a bonyolult konfigurációs beállítások, melyek a többi keretrendszerre annyira jellemzőek, csak hátráltatják a munkát. Éppen ezért a *Rails* kerüli ezeket, inkább nagymértékben támaszkodik a konvenciókra. Áthatja a *Rails* működését egy másik fontos elv is, a *DRY* (*„don't repeat yourself"*), ami annyit jelent, hogy „ne ismételd magad hiába"! Ha valamit valahol leírtál, akkor azt soha máshol ne kelljen leírni még egyszer. Ez segít abban, hogy a *Rails*-al öröm legyen a fejlesztés. Eredményül átlátható és könnyen továbbfejleszthető programkódot kapunk.

A keretrendszer

Az elmélet bevezetőn immár túljutottunk, vegyük sorba mit kínál nekünk a *Rails*, mint fejlesztőeszköz! Mint szó volt róla a bevezetőben, mindent tartalmaz, amire egy adatbázis alapú weboldal fejlesztéséhez szükséges lehet. Ez azt jelenti, hogy a segítségével a kérés beérkezésétől kezdve egészen a válasz elküldéséig kontrollálhatjuk a folyamatokat. Kezeli a beérkező kéréseket, segíti az adatbázissal való munkát és rengeteg eszközt ad a kezünkbe a hatékony felhasználói felületeket megtervezéséhez is. Mindezt programozói szemmel nézve bámulatosan hatékonyan és átláthatóan teszi. Egy *Rails-el* készített alkalmazás (weboldal) a *Model-View-Controller* séma alapján épül fel. Vagyis külön tároljuk azokat a kódrészleteket, amelyek az adatokkal foglalkoznak, külön a megjelenítés sablonjait és külön az üzleti logikát. Az üzleti logika az a része a programnak, amely megmondja, hogy ha a felhasználó ezt az címet írta be a böngészőjébe akkor ennek és ennek kell történnie. Vegyük sorra a főbb komponensek működését!

Álljon itt pár olyan weboldal, amerre érdemes körülnézni annak, aki a *Ruby* nyelvet szeretné kicsit közelebbről megismerni:
➲ http://tryruby.hobix.com/
Próbáld ki *Ruby*-t a böngésződdben!
➲ http://poignantguide.net/ruby/
Az egyik legmókásabb programozásról szóló könyv amit életemben láttam.
➲ http://www.ruby-lang.org
A *Ruby* hivatalos oldala.



1. ábra Egy Rails alkalmazás a Model-View-Controller séma alapján épül fel

Az ActiveRecord megszelídíti az adatbázist

Az *ActiveRecord* segítségével játékká egyszerűsödik az adatbázissal való munka. Adatbázis tábláinkat osztályokként kezelhetjük, az adatbázis egy sora pedig egy példánynak felel meg. Ezt hívják szakszóval *Object Relacional Mapper-nek* (*ORM*).
A táblák közötti kapcsolatot rövid, egy soros utasításokkal adhatjuk meg. Ha például minden *user*-hez tartozhat egy cím, akkor azt így írhatjuk:

```
class User < ActiveRecord::Base
  has_one  'address'
end
```

2. ábra Érdeklődj, kezdd el használni, fejlődj és kapcsolódj be a fejlesztésébe! – áll a Rails weboldalán

```
class Address <
➥ActiveRecord::Base
  belongs_to  'user'
end
```

Létrehoztuk tehát az User és az Address osztályokat! Az *ActiveRecord* a konvenciók alapján az *User* osztályhoz az adatbázisban az *users* táblát fogja társítani, az *Address* osztályhoz pedig az *addresses* táblát. Az két osztály közötti kapcsolatot adatbázis szinten az *addresses* táblában található külső kulcs (*foreign key*) oszlop fogja biztosítani. Ezek után már gyerekjáték lesz dolgozni az adatainkkal. Ha például van egy „Attila” nevű felhasználónk, akinek van címe az adatbázisban, azt így módosíthatjuk:

```
user =
➥User.find_by_name('Attila')
#lekérjük az adatbázisból
Attila adatait
user.address = 'itt lakom
➥látod, ez az a hely'
#felülírjuk a kapcsolódó cím
rekordot
```

Ugye milyen egyszerű? Olyan az egész mintha angolul, tőmondatokban írnánk le az utasításainkat. Egyszerűbben már nem is lehetne.

Az ActionController a rendszer lelke

Mi történik akkor amikor egy *Rails* alkalmazás kiszolgál egy oldalt? Ha a felhasználó beírja, hogy „/user/edit/” A *Rails* rögtön az *UserController* osztály *edit* metódusát fogja meghívni. Ha az *edit* metódus nem létezik, akkor pedig megpróbálja megkeresni és a felhasználónak elküldeni a metódushoz tartozó *html* sablont. Ha az *UserController* sem létezik akkor már baj van. Ekkor már egy kövér hibaüzenettel fogunk találkozni. A *Rails*-ben tehát létrehozhatunk osztályokat, melyek a kérések kezelését fogják elvégezni. Ez az a rész, ahol az üzleti logika megbújik.
Az *ActionController* tehát kezeli a bejövő kéréseket, és segít a válasz küldésében is. Ez az a komponens, amelyik eldönti, hogy milyen kérésre mit kell válaszolni, majd pedig a sablonok alapján felépíti a válaszul küldendő oldalt. Itt ugyancsak megfigyelhetőek a *Rails* konvenciói, hiszen a kéréseket lekezelő kódok és a válaszok megjelenítéséért felelős *HTML* sablonok pusztán nevük alapján kapcsolódnak egymáshoz.

És még sok más...

A rendszer két legfontosabb összetevőjét megismertük. Vegyük sorra azokat az eszközöket, melyek segítenek a munka során!
Az egyik például a kódgenerálás. Szinte bármilyen funkciójú kód sablonját legyártathatjuk a keretrendszerrel. Ilyen esetben a *Rails* létrehozza a szükséges fájlokat, bennük pedig elhelyezi azokat az alap kódokat, amelyek segítik az elindulást. Ha például egy új *model*-t szeretnénk létrehozni „User” néven, akkor a következő parancsot kell kiadnunk az alkalmazásunk gyökérkönyvtárában:

```
ruby script/generate model User
```

Ennek hatására a létrejön egy „User” nevű modell, minden hozzá tartozó fájllal együtt. Hasonló módon generálhatunk számos mást is, lényegesen leegyszerűsítve ez által a fejlesztést. A másik érdekes dolog a *scaffold* (magyarul álványzat). Ez az a lehetőség amelybe mindenki beleszeret a kezdetekben. Nevéhez híven megtámogatja a fejlesztést, ugyanis a *scaffold*-dal létrehozhatunk egy alap felületet, melyen keresztül egy *model*-t kezelhetünk. Ez azt jelenti, hogy minden komolyabb programozás nélkül kapunk egy olyan oldalt, amelyen egy adatbázis tábla (ez felel meg egy *model*-nek) tartalmát változtathatjuk. Később, természetesen saját felületet építhetünk az adatok eléréséhez, de a kezdetekben nagyban megkönnyíti a tesztadatok bevitelét.

Összefoglalás

Röviden áttekintettük a *Rails* főbb tulajdonságait. Mivel a legtöbb dolog akkor válik érdekessé amikor munkára fogjuk, így a következő részben, részekben egy alkalmazás fejlesztését fogom bemutatni. Aki nagyon türelmetlen, az látogassa meg a projekt weboldalát és nézze meg a videókat!

A Model-View-Controller séma (MVC)

Az *MVC* egy program tervezési minta. A mintát követő alkalmazásokban szeparálni igyekszünk a különböző funkciókat végző programrészeket. Főleg asztali alkalmazás fejlesztésekor dolgoznak a séma alapján. Az adatokkal dolgozó programrészeket *Model*-nek, a megjelenítés sablonjait *View*-nek, a program lelkét adó, felhasználói interakciót és üzleti logikát rejtő részeket pedig *Controller*-nek nevezzük. Az így megkülönböztetett programrészeket általában különböző könyvtárban is tároljuk. Előnye ennek a tervezési mintának, hogy átlátható, könnyen karbantartható kódot kapunk.

„Rails is the killer app for Ruby.”, vagyis a *Ruby* nyelv számára a *Rails* hozhatja meg a régóta várt ismertséget, írta *Yukihiro Matsumoto*, a nyelv megalkotója. Állítását igazolhatom máris, hiszen a fejlesztői listák tanúsága szerint rengetegen ismerkednek meg a nyelvvel miután felkeltette a kíváncsiságukat a *Rails*. Ez fordítva is igaz, hiszem a *Ruby* nyelv filozófiája nélkül a keretrendszer csak egy lenne a sok közül.

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.

Verziókezelés határok nélkül – A Bazaar és az Olive felület

A verziókezelésről a Linux felhasználók jó része a CVS-re asszociál, érthető módon, hiszen rengeteg nyílt forráskódú projekt tárolja ebben a fejlesztés eredményét. Léteznek azonban kicsit más elgondolás alapján működő rendszerek is. A cikk célja kettős: egyfelől a verziókezelő rendszerek fontosságára akarja felhívni a figyelmet, másrészt pedig egy friss fejlesztéssel is megismerkedünk.

Kik jelentik a célcsoportot?

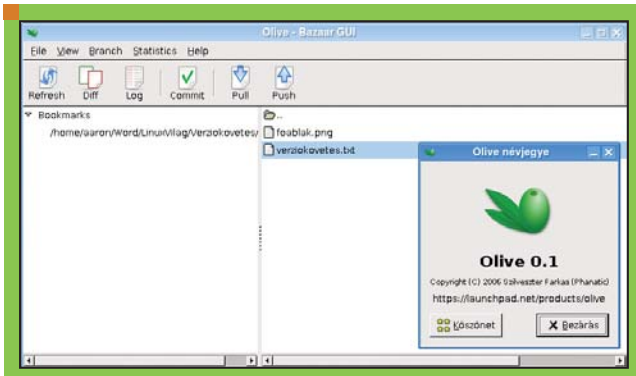
Minden olyan felhasználó, aki termelékeny munkát végez a számítógéppel. Dokumentumokat írunk, digitális képeket retusálunk, prezentációkat készítünk, a rendszer beállítófájljait hangoljuk, programokat fejlesztünk és eközben sok esetben semmilyen verziókezelő rendszert nem használunk. De miért is kellene? Talán velünk is megesett, hogy munkánk során kiderült, hogy tegnap még jó irányba haladtunk, de egy apró módosítás miatt mára sikerült tönkretenni az egészet. Milyen jó lenne előszedni a tegnapi verziót!
Sőt, még jobb lenne, ha azt is láthatnánk, hogy mi az a bizonyos módosítás, amire már nem is emlékszünk.
Ez mindenféle munka során előfordulhat. Ezekre az esetekre (meg még számtalan másra) találták ki a verziókezelést! Egy bizonyos összetettségi szint alatt az életünket csak megnehezíti a verziókezelő rendszer használata, semmint, hogy megkönnyítené. Ezt a szintet magunknak kell megállapítani, hogy mikor ér meg egy kis plusz odafigyelést az, hogy az egyes munkafázisokat pontosan nyomon tudjuk követni és elő tudjuk szedni. Például egy cikk megírásakor nem szoktam verziókezelő rendszert használni. Ezzel a cikkel kivételt teszek a képernyőfotók kedvéért, hogy lássuk, hogyan alakul a szöveg.
Most mindjárt be is rakom az aktuális állapotot a verziókezelő rendszerbe!
Elkészült az első verzió. Hol is tartottunk? Talán az a tévhit él a verziókövetésről, hogy szöveges adatokkal használhatóak csak. Ez valamilyen szintig igaz is, hiszen egy bináris fájlon megnézni, hogy mi változott az egyes változatok között, nem valami vidám mulatság, de a legfontosabb előnyt nem veszítjük el: a rendszer helyettünk gondoskodik arról, hogy bármikor elő tudjuk szedni a megadott állapotot. Ezt persze akár magunk is megtehetjük egy precíz fájlelnevezési konvencióval, de azért az jóval nehézkesebb és több odafigyelést igényel. Ráadásul minden egyes fájlverzióhoz megjegyzést is fűzhetünk, amiből kiderül az éppen aktuális módosítás célja. Talán mostanra már mindenki kedvet kapott a kipróbáláshoz, úgyhogy nézzünk egy konkrét megvalósítást!

Bazaar – elosztott verziókezelés

Ha analógiát akarunk keresni, akkor azt mondhatjuk, hogy a fájlcserélők után a verziókezelő rendszerek is beálltak a sorba: a decentralizálásnak mindenütt vannak előnyei. Gondoljunk csak az internetre, ott is a decentralizálás gondolatával kezdődött a fejlesztés. Mit is ígér egy verziókezelő, ha elosztott? A legnagyobb előnyként azt említi a *Bazaar* leírása, mely a ➲ *http://bazaar-vcs.org* címen érhető el, hogy élő internetkapcsolat nélkül is lehet dolgozni. Értve ezt mind a kliens illetve a szerver kapcsolatára. Hiszen ahogyan mondjuk egy repülőn ülve nem szokott internethozzáférés lenni, úgy a szerver kiesése is okozhatja egy centralizált rendszer működésképtelenségét. A saját gépen lévő lerakatba (*repository*) lehet beküldeni (*commit*) a munka eredményét. További előnyöket is felsorol, melyek leginkább szoftverfejlesztőknek lehetnek hasznosak. Lehet azonban centralizált módon is dolgozni a *Bazaarral*, ekkor nagyon hasonlít a működése a *CVS*-re. Egyébként sem különösebben tér el, már azokban a dolgokban ami a felhasználó felé jelentkezik. Ez jól is van így, aki használt már változatkezelő rendszert, gyorsan megbarátkozik ezzel is.
A *Bazaar* egy karakteres felületű alkalmazás, így a bzr(1) man oldalon részletes leírást találhatunk róla. Én viszont a cikkeket grafikus felületen szoktam megírni és sokan vannak olyan Linux felhasználók, akik csak akkor nyúlnak parancssorhoz, ha másképp nem lehet megoldani a problémát, így most is így teszünk!
A *Summer of Code* keretében elkészült az *Olive* felület a *Bazaarhoz*, így a verziókezelés használata nemcsak hasznos, de egyszerű is!

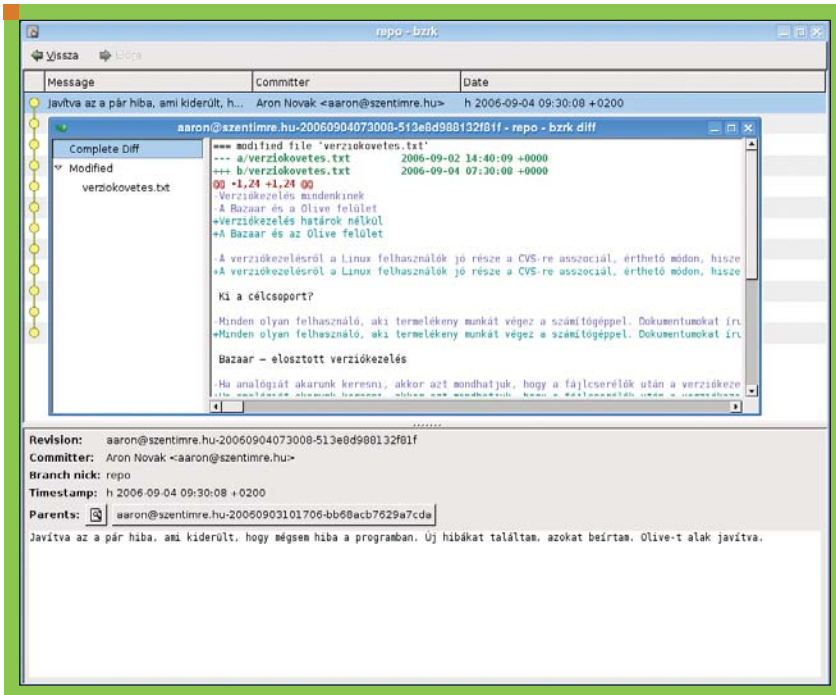
Olive – egy újabb magyar fejlesztés

Tehát grafikus felületen használnánk a Bazaar által felkínált lehetőségeket? A ➲ *http://bazaar-vcs.org/Olive* címről tudjuk letölteni a legfrissebb verzióját *Farkas Szilveszter* fejlesztésével elkészült programnak.



1. ábra Az Olive képernyője

A szoftver ***Pythonban*** íródott és a ***GTK***-t használja. A 0.10.0-es verzió volt a legfrissebb a cikk megírásakor. Ahhoz, hogy az ékezeteink rendesen működjenek, nem árt, ha a rendszerünk ***UTF-8***-as karakterkódolásra van állítva. A csomag letöltése után a `setup.py` paranccsal tudjuk telepíteni a szoftvert. Ezután az `olive-gtk` kiadásával indul az alkalmazás.
Ahhoz, hogy el tudjuk kezdeni használni az ***Olive*** lényegi funkcióit, először létre kell hoznunk egy tárolót. Tipikusan egy tárolóban egy adott munkához tartozó fájlokat tárolunk, például ennél a cikknél a szövegfájl és a képernyőfotók tartoznak ide. Abban a könyvtárban, ahol dolgoztunk vagy dolgozni fogunk, adjuk ki a `bzr init`-nek megfelelő parancsot az ***Olive***-ban: a Branch/Initialize menüvel tehetjük ezt meg. Ez létrehozza azt a struktúrát, amelyre a ***Bazaarnak*** szüksége van a verziók tárolásához (ez az adott könyvtárbeli rejtett ***.bzr*** könyvtárban látható).
Az `init` parancs nem adta a tárolóhoz hozzá önműködően az összes benne lévő fájlt és könyvtárat, ezt nekünk kell megtenni az `add` utasítással. Ezután nincs más hátra, mint nekilátni a munkának. Gépeljünk, kódoljunk, tervezzünk, rajzolgassunk és mikor úgy érezzük, hogy most már készen van egy részfeladat, vagy mondjuk vége a munkanapnak akkor vegyük elő az ***Olive***-ot és adjuk ki a fájlra (fájlokra) a `commit` parancsot. Ezzel mondjuk meg, hogy az aktuális állapotot vegye fel a rendszer egy új verzióba. Ezzel készen is volnánk. Tulajdonképpen a plusz munka, mely szükséges ahhoz, hogy használhassuk a verziókezelést, ennyivel véget is ér. Most lássuk, mit kapunk cserébe!
Két változat között láthatjuk a különbségeket, ez nyilvánvalóan csak szöveges fájloknál használható. A ***Bazaar*** oldala alapján azonban tervezik azt a lehetőséget a fejlesztők, hogy külső programmal bizonyos bináris fájlok közti különbség mégis szemléletes legyen. Most a ***diff*** csak annyit árul el két bináris fájlról, hogy azok különböznek-e vagy sem. Elképzelhető, hogy később például ***OpenOffice.org*** fájlok esetén egy külső programmal láthatóvá válik két beküldött ***odt*** fájl közötti különbség is.
Ezt a ***Statistics/Logs*** menünél a ***Parents*** mellett lévő nagyítóval kérhetjük. Pont azokat a változásokat fogjuk megkapni, amin éppen állunk.
Ha egy régebbi verzióra van szükségünk, akkor azt a ***Branch/Get*** menü alatt tudjuk kérni. Beírjuk a ***Branch*** elérési útvonalát (ez a `bzr inittel` előkészített mappa), kiválasztjuk, hogy hova akarjuk létrehozni az adott verziót, aztán a ***Revision Numberrel*** pedig azt adjuk meg, hogy hányadik módosításig alkalmazzuk a változásokat. Ennyi az egész.
Szoftverfejlesztéshez ennél jóval összetettebb módon is lehet használni a ***Bazaart***, de alapvetően a verziókezelő rendszerek ezeket a szolgáltatásokat nyújtják. Az ***Olive*** pedig segít nekünk, hogy minél kisebb energia ráfordításával tudjunk részesülni mindezen előnyökből.
A ***Bazaart*** jó eséllyel megtaláljuk a disztribúciónk csomagjai között, az ***Olive*** pedig már bekerült az ***Ubuntu Universe*** csomagjai közé, remélhetőleg hamarosan a többi disztribúcióhoz is lesz csomag.



2. ábra A különbségek listája (diff) és a napló, melyet magunk írhatunk az egyes verziókhoz

Novák Áron
(aaron@szentimre.hu)

BME-VIK-es hallgató, műkedvelő rendszergazda. Jelenleg leginkább a NetBeans-szel és mindenféle hordozható eszközzel foglalkozik, legalábbis mindazokkal, amelyeket meg lehet szólaltatni Linux alatt.

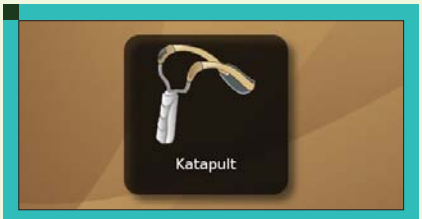
Digitális lőfegyver

Katapult

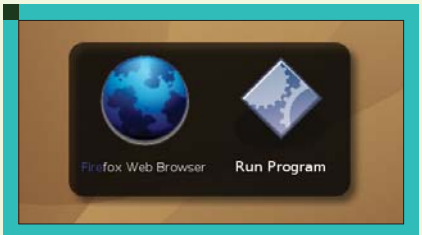
Az ókori rómaiak által használt rettegett fegyver hasznos játékszerré idomult. A Katapult segítségével villámgyorsan indíthatunk programokat, nyithatunk meg mappákat, vagy kereshetjük meg a könyvjelzőink közül az éppen szükségeset.

A *Katapult* az a típusú program, ami nélkül egészen addig lehet élni, amíg nem próbáltuk ki. A mindennapi munka, szórakozás során már észre sem vesszük, hogy mennyi időt töltünk azzal, hogy a menüben programok után kutatunk. Persze kellő rutinnal már gyorsan megtalálhatunk mindent, de maga a menüben történő navigáció is időbe kerül. Rutinosabbak persze indíthatják a programokat parancssorból is, ezáltal egyszerűsítve le a fentebb leírt problémát. De vajon nincs-e valami szebb és ötletesebb megoldás? Hasonlók kérdésekre ad választ a *Katapult*. Ez a program egy univerzális svájci bicska. Segítségével gyorsabban érhetjük el a programjainkat, a könyvtárainkat, a könyvjelzőinket és bármit amit el tudunk képzelni. Telepítéshez használjuk a terjesztésünk csomagkezelőjét! *Debian* alapú rendszeren egy `sudo apt-get install katapult` parancsra van szükségünk. Miután sikeresen telepítettük, indítsuk el!

1. ábra A Katapult logója



2. ábra A Katapult-ot egy „ALT+SPACE” billentyűkombinációval indíthatjuk el



3. ábra Alig gépelünk be pár karaktert, a Katapult már tudja is mire gondoltunk

A használat

A *Katapult* fejlesztői az igazán kézreálló *ALT+SPACE* billentyűkombinációt választották ki a program aktiválására. Ez egy jól bevált ötlet, hiszen ezt a kettő billentyűt tényleg nem lehet eltéveszteni.
Miután leütöttük a billentyűkombinációt, előtűnik a képernyő közepén a program ikonja egy fekete keretben. Ekkor indul a varázslat. Kezdjük el gépelni a kedvenc programunk nevét! Én történetesen a *Firefox*-ot szeretném elindítani, így elkezdem beütni a betűit. Alig írom be, hogy „fire” és a *Katapult* már tudja is mire gondoltam. Ekkor elég megnyomni az *ENTER* billentyűt a kiválasztott program indításához.
A *Katapult* a programjainkhoz hasonlóan ismeri a mappáinkat és az internetes kedvenceinket is.

Tudását pluginekkel bővíthetjük, bár ehhez egy kis *C++* tudásra is szükségünk lesz.
Remélem sokak kedvencévé válik majd ez az igazán hasznos apróság!

Juhász Attila (juhat@goraffe.hu)

Evolution

Linux alá többfajta kiváló levelező-alkalmazás létezik. Elég csak a Mozilla Thunderbird-re vagy a KMail-re gondolnunk. Az Evolution egy GTK-ra épülő, azaz Gnome stíluselemeket felvonultató program, mely könnyen kezelhető, gyors és tetszetős. Ezen felül érdeme még, hogy valamennyi Linux-disztribúció tartalmazza.

Bevezetésként néhány személyes tapasztalatomról írnék, melyek az ***Evolution*** használatával, illetve futtatásával kapcsolatosak. Jómagam 2003 eleje óta használom ezt az alkalmazást és mondhatom, hogy jóban-rosszban kitartottam mellette. Annak idején ***UHU-Linux 1.0 RC2*** volt a gépemen, ami akkoriban a legkönnyebben kezelhető és telepíthető disztribúciónak számított. Volt benne ***MPlayer***, szótárprogram, alapból volt hang, szóval egy kezdőbb felhasználónak bizony sok öröme volt benne. Jól összeválogatott alkalmazások sora növelte ezen terjesztés hitelét. Én is, mint minden kezdőbb felhasználó, nagyon örültem, hogy végre kizárólag ***Linuxot*** használhatok a mindennapi feladatok megoldására, internetezésre és szórakozásra. Korábban ingyenes ***webmail*** szolgáltatást használtam, de gondoltam egyet és kipróbáltam egy tényleges levelezőklienst. Semmilyen tapasztalatom nem volt erről, halvány fogalmam sem volt a ***POP3*** és ***SMTP*** rövidítések jelentéséről, csak derengett valami. Összeszedtem némi információt az internetről és beüzemeltem a levelezőprogramot. Ettől fogva mást sem használtam. Nem kellett hozzá böngésző, nem kellett állandóan felhasználónevet és jelszót megadni, ráadásul egy felszabadultsági érzetet is biztosított. Ez volt az ***Evolution***.

Az internetes levelezésről általában

Amennyiben elektronikus úton szeretnénk levelezni, két megoldás közül választhatunk. Az egyik az úgynevezett ***webmail*** szolgáltatás, ami lehet egyaránt ingyenes (például ***freemail***, ***citromail***, stb.), illetve fizetős. Ezt, hogy „fizetős", többféleképpen is érthetjük. Léteznek az ingyenes mailszolgáltatóknak olyan szolgáltatásai, amelyeket némi összeg fejében vehetünk igénybe. Például gondoljunk a tárhelyre.
Alapból az ingyenes szolgáltatók kevés tárhelyet adnak (körülbelül 20 MB körülit), bár ez sem feltétlenül igaz. Mindenesetre, ha több tárhelyet szeretnénk, akkor fizetni kell. Ugyanígy, a csatolt állományok mérete is lehet pénzfüggő.
A másik csoport a ***webmail*** ellentéte. Ennek igazából nincs is külön neve. Levelezőprogramon keresztül tölthetjük le leveleinket ***POP3***, illetve ***IMAP*** segítségével. Némely ingyenes szolgáltató is nyújt ***POP3*** lehetőséget, amit igénybe vehetünk (manapság már majdnem mindegyik), tehát akár a ***freemailes*** leveleinket is letölthetjük levelezőprogram segítségével.

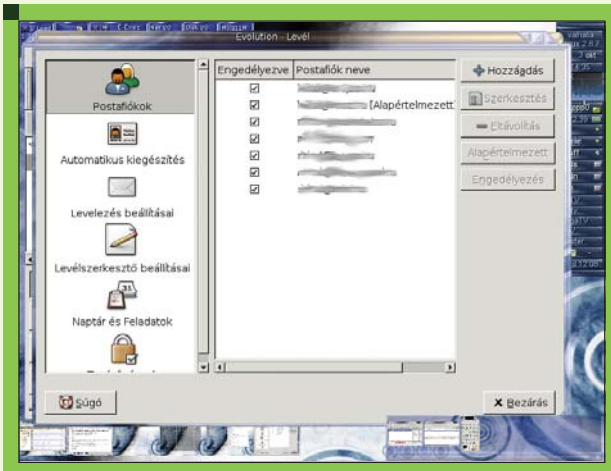
SMTP, POP3, IMAP

Feltételezem, hogy valamennyien találkoztunk már ezen rövidítésekkel, de nem biztos, hogy mindenki tudja, hogy mit is jelentenek. Lássuk akkor, hogy mit takarnak és nagyjából mit jelentenek ezek:

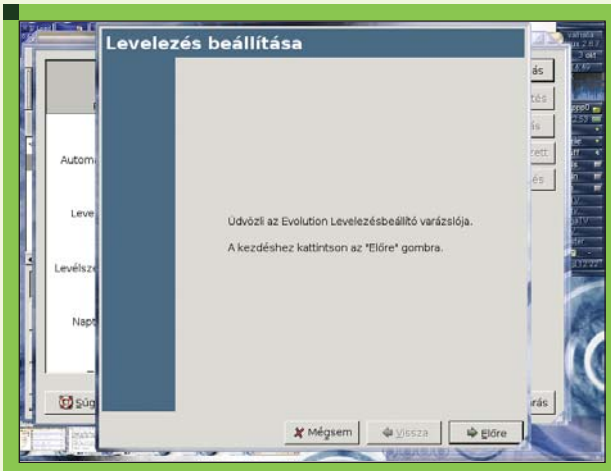
- A ***Simple Mail Transfer Protocol*** (***SMTP***) felel a levelek kézbesítéséért. Az ***Evolution*** konfigurálásánál fontos szerepe lesz. Az ***SMTP*** egy viszonylag egyszerű, szöveg alapú protokoll, ahol egy üzenetnek egy vagy több címzettje is lehet. Könnyen tesztelhetjük az ***SMTP***-t a ***Telnet*** program segítségével. Az ***SMTP*** szolgáltatás a ***TCP*** (***Transmission Control Protocol***) 25-ös portját használja. Ahhoz, hogy meghatározza, hogy az adott ***domain*** névhez melyik ***SMTP*** szerver tartozik, a ***domain*** név ***MX*** (***Mail eXchange***) rekordját használja. Ez a ***domain DNS*** rekordjai között szerepel.
- Az ***IMAP*** (***Internet Message Access Protocol***) és a ***POP3*** (***Post Office Protocol v3***) alkalmazás rétegbeli protokollok, amelyek segítségével a leveleinkhez férhetünk hozzá. A legtöbb modern szerver és kliens is támogatja használatukat. Azért némi különbség van a kettő között: a ***POP3*** egyenesen letölti a leveleinket a szerverről a gépünkre, míg az ***IMAP-ot*** úgy képzeljük el, mintha a levelezőprogramon keresztül bejelentkeznénk a távoli gépre, majd ott olvasnánk üzeneteinket.

Levelezőprogramok

Hogy mi is a levelezőprogram? Egy szoftver, mely képes leveleket fogadni és küldeni az imént említett protokollok segítségével. A ***webmail-lel*** ellentétben ez helyhez kötött, tehát külön kell telepíteni minden gépre. Sok fajta különböző ilyen alkalmazás létezik. A legismertebb a ***microsoftos Outlook Express***. Más ismertebb



1. ábra Az Evolution több e-mail címet is tud egyszerre kezelni



2. ábra E-mail címeinket egyszerűen élesíthetjük

programok is léteznek az ablakos rendszerre, például *PegazusMail*, *Eudora*, stb. *Linux* alatt már szabadabb kezet kapunk a programválasztáshoz, mivel létezik grafikus és konzolos levelezőprogram is kedvenc rendszerünkhöz Parancssoros környezetben a legelterjedtebb a *Pine*, amit magam is több gépen használok.
De említhetném a *Mutt-ot* is, ami szintén egy kiváló karakteres alkalmazás. Grafikus környezetben már szélesebb palettáról választhatunk. Az itteni programokat több kategóriába lehet sorolni. Van a grafikus környezetbe direkten beépített levelezőprogram, mely az ablakozó stílusát tükrözi. Ilyen például *KDE* alatt a *KMail* és *Gnome* alatt az *Evolution*. Van azonban böngészőspecifikus levelezőprogram is *Mozilla-hoz*, *Opera-hoz*, illetve különálló úgynevezett *suite* rendszer, mely több szolgáltatást is magába foglal. Ilyen a *Mozilla Thunderbird* és az *Evolution*. Az utóbbiról szól ez a cikk.

Az Evolution

Az *Evolution* – mint már említettem – a *Gnome* környezet alapértelmezett levelezőprogramja. Ez persze nem jelenti azt, hogy más grafikus ablakkezelő alatt nem lehet használni, mivel *GTK-ra* épül, ami a grafikus *linuxos* programok legelterjedtebb felülete. Jómagam számtalan grafikus felületet használtam és mindenhol az *Evolution* szolgáltatásait vettem igénybe.
A jelenlegi stabil verzió a 2.8.0., ami letölthető az *Evolution* hivatalos honlapjáról. Természetesen ezt csak akkor kell tennünk, ha az általunk használt verziót elavultnak érezzük. Az alkalmazást minden *Linux* disztribúció alapból szállítja. Én jelenleg *Debian Sarge* alatt a 2.0.2 verziószámú *Evolution-t* használom és tökéletesen kielégíti minden ezirányú igényemet. A program elérhető minden olyan terjesztéshez, amely képes futtatni a *Gnome* környezetet.

Telepítés

Ha olyan disztribúciót használunk, amelyik megengedi a minimális *installt* (csak a legszükségesebb fájlok telepítése), akkor az *Evolution-t* telepítenünk kell gépünkre.
Ezt például *Debian GNU/Linux* alatt az

```
apt-get install evolution
```

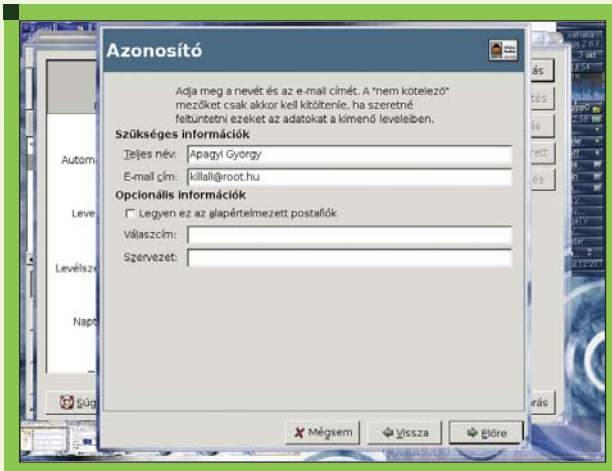
paranccsal tehetjük meg a legegyszerűbben. Természetesen, mint minden *linuxos* alkalmazást, az *Evolution-t* is telepíthetjük forrásból. Ez egy bonyolultabb folyamat, mivel a programnak rengeteg függősége van. A hivatalos honlapon viszont ehhez is kaphatunk segítséget.
Más esetben, a mai, „felhasználóbarátabb" *Linuxoknál* erre nincs szükségünk.

Használat és konfigurálás

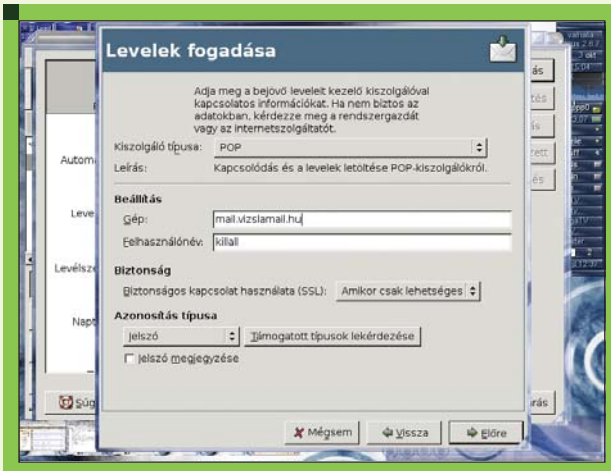
Ezt az alkalmazást végtelenül egyszerű elindítani. Jó esetben megtalálható a grafikus környezetünk menüjében és innen indíthatjuk. Kicsit rosszabb eset, ha ez nem áll fenn. Ekkor – ha grafikus felületünk támogatja az ikonokat –, létrehozhatunk egy ikont az asztalunkra, illetve más, alkalmazásindítási funkciót is használó programmal indíthatjuk (például *GKrellmLaunch*).
Végső esetben bármely grafikus terminálba beírható az

```
evolution
```

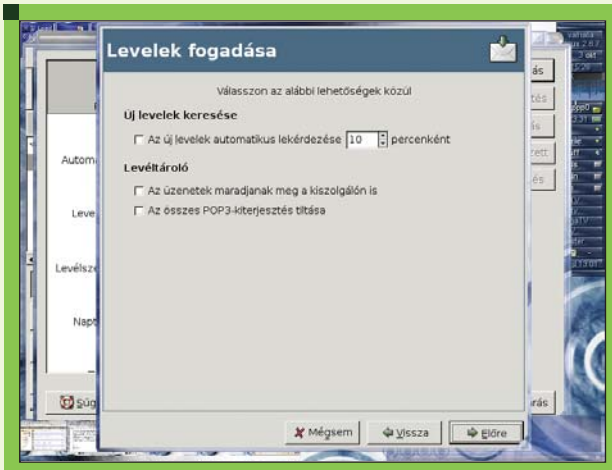
parancs.
Az *Evolution-nek* számtalan funkciója van (például határidőnapló, feladatmenedzser), amiket én nem használok, de a hivatalos dokumentációban megtalálható a használatukhoz szükséges segítség. Ebben a cikkben a legfontosabb funkciót, a levelezést szeretném bemutatni.
Nézzük akkor, hogy hogyan is lehet élesre tölteni programunkat. Először is a levélfiókjaink konfigurálásával kell kezdenünk a műveletet. Mint minden levelezőprogramban, itt is egyszerre több *e-mail* címet használhatunk. Elsőként lépjünk az *„Eszközök/Beállítások"* menübe.
Itt végezhetjük el postafiókjaink beállítását. Ha még sosem használtuk az *Evolution-t*, akkor az ábrával ellentétben egy üres lap fog fogadni bennünket. A *„Hozzáadás"* gombra kattintva vehetünk fel újabb fiókokat, melyben az *„Evolution Levelezésbeállítási Varázslója"* lesz segítségünkre.
Az *„Előre"* gombra kattintva kezdhetjük meg a műveletet. Legelőször meg kell adnunk a saját nevünket, majd a tervezett *e-mail* címünket. Opcionálisan választhatunk válaszcím lehetőséget, illetve megadhatjuk cégünk nevét. Alapértelmezetté is tehetjük a most beállítandó postafiókunkat, azonban ezt később is megtehetjük.



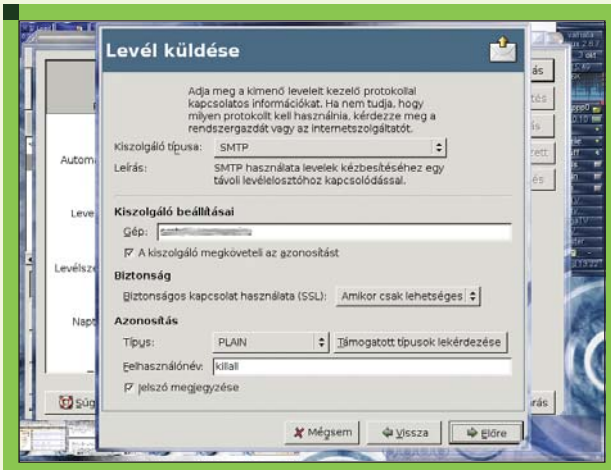
3. ábra Adjuk meg azonosítónk adatait



4. ábra Levélfogadási mód kiválasztása

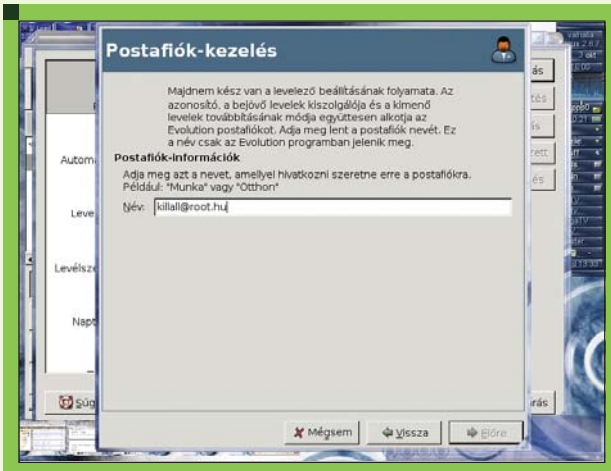


5. ábra Opcionális levélfogadási beállítások

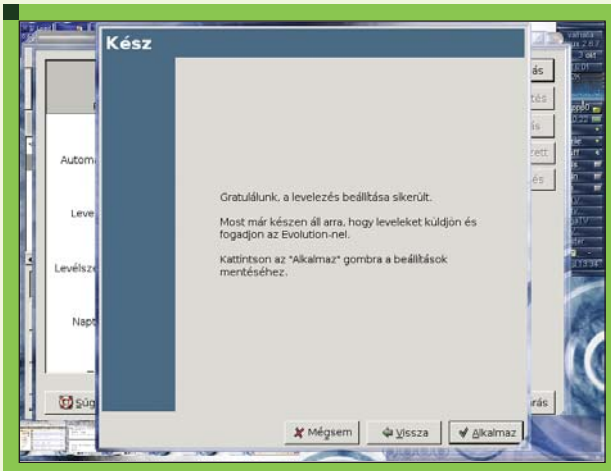


6. ábra Levelek küldésének beállítása

Felhívnám a figyelmet arra, hogy természetesen akárhány címünk is van, alapértelmezett csak egy lehet. Amennyiben végeztünk és tovább lépünk, egy fontos pontra érkezünk. Nevezetesen a levelek fogadásának beállításához. Az ***Evolution*** számtalan fogadási lehetőséget ajánl fel. Mint már korábban említettem, két lehetőség közül kell választanunk leggyakrabban, ***POP3*** vagy ***IMAP***. Ezek közül is a ***POP3*** a gyakoribb, ezt a mailszolgáltatók 99%-a támogatja. Természetesen másfajta opciót is választhatunk, de előbb érdeklődjünk a szolgáltatónknal a támogatott protokollok ügyében.
Az ábrából látszik, hogy ezen a lapon több dolgot is meg kell adnunk, nem csak a protokollt kell kiválasztanunk. Amint a legördülő menüben kiválasztottuk a megfelelő bejegyzést, egyéb mezők fognak megjelenni. Meg kell adnunk a levelek fogadásáért felelős szerver nevét, a felhasználónevünket, valamint opcionálisan választhatjuk a biztonság (***SSL***) mértékét. Ez csak a saját „paranoiánktól" függ. Választhatjuk a jelszó automatikus megjegyzését is. Persze itt is le kell ragadnom egy pillanatra, mivel az ***Evolution*** rengeteg azonosítási módot ajánl fel. Többek között jelszón keresztüli azonosítás, bejelentkezés, ***DIGEST-MD5***, ***CRAM-MD5***. Az egyszerűség kedvéért válasszuk a jelszót a listából, ez mindig beválik!
Fontos funkció még itt a ***„Támogatott típusok lekérdezése"*** gomb. Ezzel alkalmasint ellenőrizhetjük például azt, hogy a ***POP3*** szerver támogatja-e a jelszón keresztüli azonosítást.
Ha minden levélfogadási beállítással végeztünk, akkor léphetünk a következő lapra, ami szintén a levelek fogadásával foglalkozik, azonban más szemszögből nézve.
Az ***Evolution*** képes arra, hogy bizonyos időközönként lekérdezze a ***POP3***-szervert és érdeklődjön, hogy érkezett-e új levelünk. Ehhez természetesen az alkalmazásnak állandóan futnia kell Jómagam egy alternatív megoldást használok az online levélfigyelésre, amire majd a cikk végén fogok kitérni.
Ebben a szekcióban eldönthetjük még, hogy a levelek letöltésével egy időben törlődjenek-e a levelek a kiszolgálóról, illetve a ***POP3*** kiterjesztéseket is tilthatjuk. Az előbbi fontos dolog lehet, ha például sokat utazunk és szeretnénk mindig, mindenhol hozzáférni a leveleinkhez. Természetesen ezt úgy kell érteni, hogy nem tudunk az itthoni gépünkre bejelentkezni. Például, ha ingyenes ***webmailes***, vagy bármilyen ***webmailes*** szolgáltatást használunk és sokat utazunk, akkor ez a funkció nagyon fontos lehet. Tehát ebben az esetben pipáljuk ki a ***checkbox-ot***!

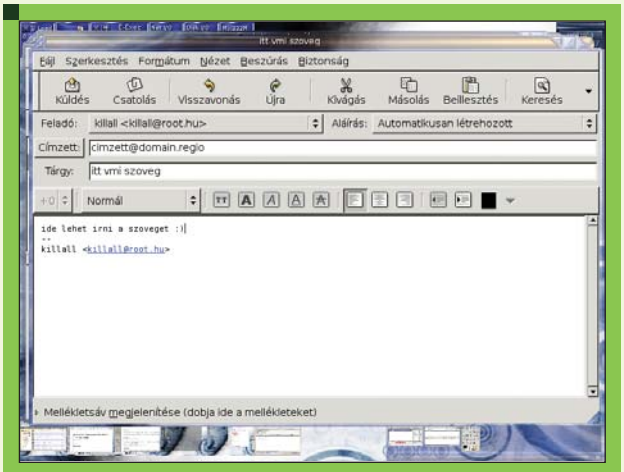


7. ábra A levélfiók nevének megadása



8. ábra Beállításaink sikeresen befejeződtek

A következő lapon, ami szintén létfontosságú, a levelek elküldésének beállításával foglalkozunk.
Ez a legtöbb esetben az *SMTP* protokollon keresztül történik, amiről már volt szó korábban. Ha nem rendelkezünk otthon mailszerverrel, akkor erről kár is többet beszélni. Itt hasonlóképp az előzőhöz, meg kell adnunk az *SMTP* szerver nevét, ami legtöbb esetben az internetszolgáltatónk *SMTP*-szervere. Amennyiben nem tudnánk a pontos címet, látogassunk el internetszolgáltatónk weblapjára vagy hívjuk fel az ügyfélszolgálatot. Nem fognak megharagudni, ez a feladatuk!
Ezután adjuk meg a felhasználónevet, állítsuk be a kívánt biztonsági szintet, valamint az azonosítási lehetőséget. Utóbbinál ismét alkalmunk nyílik a választásra. Legtöbb esetben a *PLAIN STMP*-azonosítás a bevált módszer. Lekérdezhetjük a támogatott típusokat, illetve választhatjuk a jelszó elmentését. Kitérnék arra, hogy már többször írtam a jelszóról, de egyik ábrán sem látható jelszó megadására alkalmas mező. Ez nem véletlen. Itt, a beállításoknál még nem kell megadnunk semmilyen jelszót, majd csak ha mindennel végeztünk és elkezdjük használni az alkalmazást. Az *Evolution* mind a levelek fogadásánál (azaz a levelek letöltésénél), mind a levélküldésnél rá fog kérdezni a jelszóra, amit elég csak egyszer megadnunk. Elérkeztünk a végső lépéshez.



9. ábra Levél írása

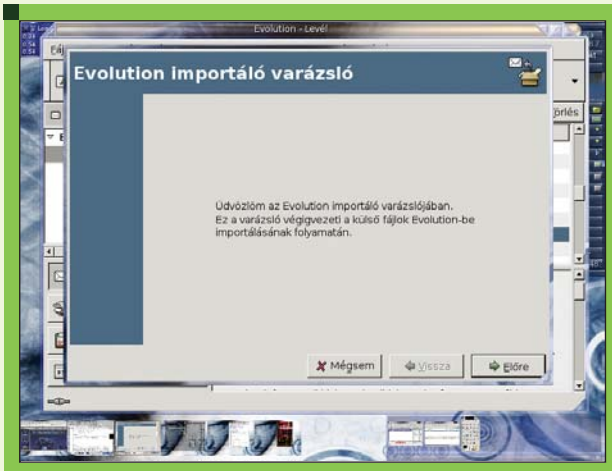
Meg kell adnunk az imént beállított levélfiókunk nevét, mely majd az első ábrán található listában fog szerepelni. Mivel én rengeteg címet használok egyszerre, ezért jobbnak láttam mindig az adott *e-mail* címet megadni névnek. Természetesen bármi lehet a levélfiókunk neve. Ezennel a konfigurálás véget ért.

Levelek írása és küldése

Rájöttem, hogy ebben a cikkben sokat használom a „végtelenül egyszerű" kifejezést. Próbálok megfelelő szinonimákat találni rá, de elég nehéz, mivel az *Evolution* kezelése pofonegyszerű (ez is egy gyakori szó ezen írásomban, nem véletlenül). Az egyszerűség kedvéért ebben a részben maradjunk annyiban, hogy levelet írni is egyszerű az *Evolution-nel*.
Az *„Új"* ikonra kattintva az *Evolution* főablakában megjelenik egy szerkesztőablak, melyben mindent elvégezhetünk. Legelőször meg kell adnunk a címzett *e-mail* címét. Ez kötelező, enélkül nem tudjuk elküldeni a levelünket. Ha mégis véletlenül elfelejtenénk kitölteni, akkor az *Evolution* erre figyelmeztet minket.
A *„Tárgy"* mező kitöltése nem kötelező, rajtunk áll, hogy írunk-e oda valamit.
A levél törzse egyértelmű. Fontos a jobb oldalon található *„Aláírás"* funkcó. Ez is opcionális. Lehet automatikusan létrehozott, vagy saját magunk által szerkesztett. Amennyiben ezt a lehetőséget igénybe vesszük, kreálhatunk egy állandó aláírást, ami minden levelünk aljára odakerül.
A *„Csatolás"* gombbal lehetőségünk nyílik állományokat csatolni levelünkhöz. A levélszolgáltatónktól függ ennek a megengedett mérete, ahogy már korábban is említettem.
Az *Evolution* egy nagyon kellemes szövegszerkesztő lehetőséget is nyújt – enyhe *Word*-érzettel –, amennyiben formázott szöveget szeretnénk küldeni, illetve az elrontott dolgokat visszavonhatjuk.
A levelet a *„Küldés"* gombra kattintva kézbesíthetjük.

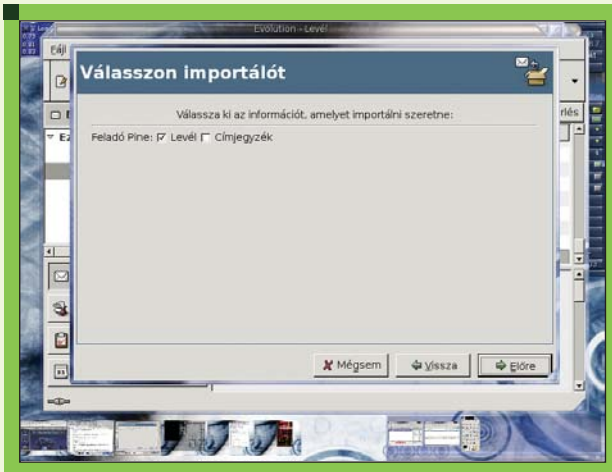
Importálás

Lehetőségünk nyílik külső (más levelezőprogramok által használt) fájlokat az *Evolution-be* importálni egy varázsló segítségével. Ezt a „Fájl/Importálás" menüből érhetjük el.
Két lehetőség közül választhatunk. Az *„Előre"* gombra kattintva megjelennek



10. ábra Fájlok importálásának megkezdése a Varázsló segítségével

11. ábra Mit szeretnénk importálni?



12. ábra Importálandó találatok

13. ábra Mailboxok importálása

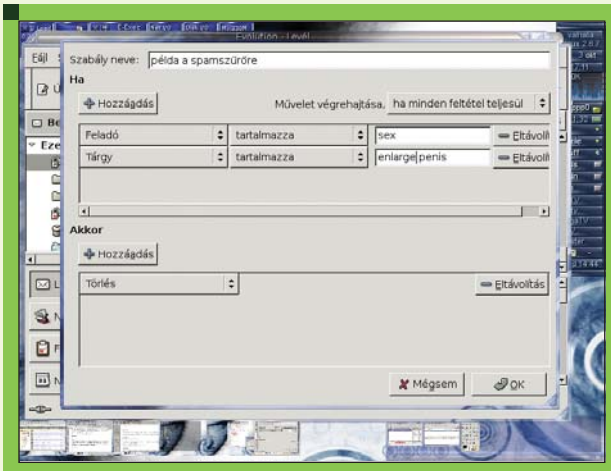
a lehetőségek. Ha az elsőt választjuk, akkor az ***Evolution*** végez egy kis rendszervizsgálatot, majd kiadja az importálható tartalom listáját. Amennyiben kiválasztottuk, majd ismét „előre haladtunk”, a varázsló közli velünk, hogy az „Importálás” gombra kattintva megkezdődhet a művelet. Ha sikeresen befejeződött a folyamat, akkor a tartalom (jelen esetben nálam levelek) bekerül az ***Evolution-be***.
A másik opció választása a kiindulásnál (lásd a ***11. ábra***) már egy érdekesebb dolog. Nevezetesen itt komplett levelezéseket menthetünk át más programokból. A komplett alatt azt értem, hogy mind elküldött, mind fogadott leveleket, piszkozatokkal és törölt levelekkel együtt. Kiemelnék a listából két fájltípust. Az egyik a ***.mbox*** kiterjesztésű állomány, melyet sok közismert levelezőszoftver használ. Többek között a teljes ***Mozilla*** család, az ***Eudora***, illetve az ***Evolution*** is. Hogy egy példával is éljek, a teljes ***Evolution-ös*** levelezés lementése, majd visszaállítása a következő: mentsük el a `~/.evolution` könyvtárat. Ezek után bármit csinálhatunk, mondjuk újratelepíthetjük a rendszert. Helyezzük át az új home könyvtárunkba a rejtett mappát, majd az importáló varázsló segítségével böngésszük ki a ***.mbox*** fájlt. Ha ezt importáljuk, akkor a levelezésünkön nem fog meglátszódni, hogy közben a teljes rendszer lecserélődött. Persze ez csak egy szélsőséges példa volt, de akár ***backup-hoz*** (biztonsági mentés) is hasznos lehet.
A másik fájltípus pedig az ***Outlook Express .mbx*** állománya, szóval akár a világ legnépszerűbb (?) levelezőprogramjának tartalmát is „Evolutionképessé” tehetjük.

Szűrők

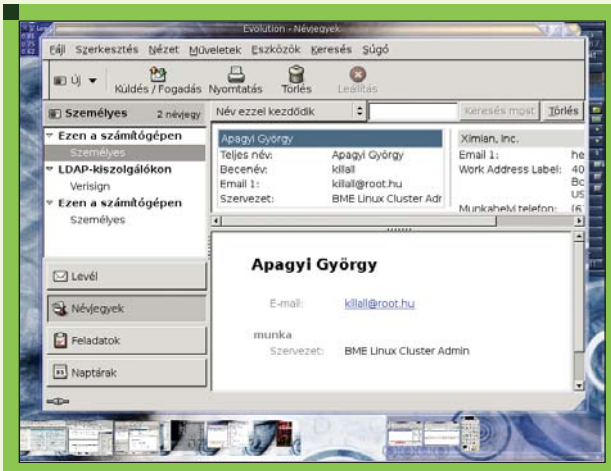
A mai levelezésben nagyon fontos a szűrők alkalmazása. Sajnos egyre több levélszeméttel (***spam***) van dolgunk. Ez lehet vírusos állomány, szexuális hirdetés vagy bármilyen nem kívánt üzenet. Az ***Evolution-nek*** nagyon hatékony és könnyen kezelhető szűrője van. Választhatunk üzenet megtartása/áthelyezése más mappába vagy üzenet törlése között. Az ***„Eszközök/Szűrők”*** menüre kattintva előjön egy lista, amely a már beállított szűrési szabályokat tartalmazza. Természetesen alapesetben ez a lista üres, így, ha szeretnénk létrehozni szűrőt, akkor a ***„Hozzáadás”*** gombra kell kattintanunk. Ekkor megjelenik egy panel melyben értelemszerűen az egyszerűtől a bonyolultig bármilyen szűrőszabályt felállíthatunk. A szűrők beállítására van egy másik mód is, melyet direkten, levélből hozhatunk létre.

Virtuális mappák

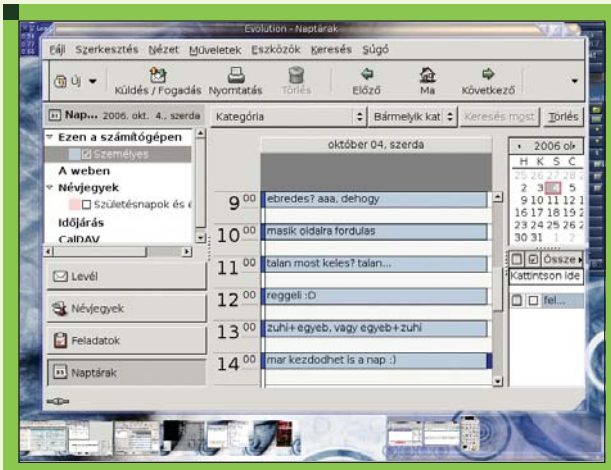
Az ***Evolution-ben*** támogatottak az úgynevezett „virtuális mappák” (továbbiakban ***VMappa***), melyek azt



14. ábra Levélszemét szűrése



15. ábra Az Evolution Névjegyszerkesztőjével létrehozott névjegyem



16. ábra Egy átlagos napom :)

a célt szolgálják, hogy akármilyen módon szortírozhassuk leveleinket. Tehát nem muszáj csak az alapértelmezett (beérkezett, elküldött, kuka, stb.) mappákat használni. A ***VMappák*** jól együttműködnek a szűrőkkel, akár szűrőt is állíthatunk egy ***VMappára***. Például a ***Jancsika*** által küldött leveleknek létrehozunk egy ***VMappát*** az ***„Eszközök/Virtuális Mappák szerkesztése”*** menüből, majd egy szűrőt állítunk a ***Jancsika*** által küldött levelek feladójára, amit a ***VMappába*** irányítunk. Így nem lesz káosz a levelezésünkben, jóval átláthatóbb lesz minden.

Egyéb szolgáltatások

Mint már írtam ezelőtt, vannak az ***Evolution-nek*** egyéb, általam nem használt szolgáltatásai is. Ezen cikk kedvéért viszont belenéztem két extra szolgáltatásba. Az egyik a névjegy létrehozása. Teljes űrlapokat tölthetünk ki, melyek minden szükséges és/vagy szükségtelen információt megkérdeznek. Névjegyek alapján is szortírozhatunk leveleket, névjegyre kattintással küldhetünk levelet. Érdekes és ötletes dolog, de én maradok a régimódi levelezésnél.

A másik számomra új dolog a határidőnapló. Az örökké rohanó mai társadalomnak jól jöhet ez a funkció, akiknek percre pontosan be van táblázva a napjuk, értékelni fogják.

Trükk

„Pár sorral” ezelőtt említettem egy online levélfigyelési trükköt, ami a következő: felesleges futtatni folyton az ***Evolution-t***, használjunk ***GKrellm-et***! Hogy mi is az? Dióhéjban ez egy monitorozó program. Az ábráim jobb felső sarkában látható egy darabja. Nem térnék ki minden részletre, azt már más megtette. Ennek a programnak vagy egy levélfigyelő funkciója, amit párhuzamba lehet hozni az ***Evolution-nel***. Nagyon egyszerű beállítani, a szerver figyelését akár 1 percre is csökkenthetjük (tehát percenként lekérdezi a leveleinket), ha levél jött, egy kis pingvin kezd ugrálni örömében. Ajánlom mindenkinek!

Összefoglaló

Végezetül hadd dicsérjem pár szóban ezt a fantasztikus programot. Sokan – főleg a „hardcore” linuxosok – bírálják a grafikus programokat. Nos, ez egy olyan alkalmazás, amit érdemes megismerni és kiismerni, mivel számtalan lehetőség rejlik benne. És garantálom, hogy aki megismeri, az meg is fogja szeretni. Remélem sikerült elég alaposan bemutatnom. Amennyiben valamit kihagytam volna, a ***Google*** mindenki segítségére lesz.
Jó levelezést mindenkinek!

Apagyi György, (killall)
(killall@root.hu)

25 éves, jelenleg az ELTE programozó matematikus szakán másodéves hallgató. Hobbija a zene (gitározás), az olvasás (Stephen King) és a számítástechnika (Linux, Unix, VMS).

KAPCSOLÓDÓ CÍMEK

Evolution hivatalos weblapja:
➲ http://www.gnome.org/projects/evolution/

Letöltés:
➲ http://www.gnome.org/projects/evolution/download.shtml

Hivatalos dokumentáció:
➲ http://www.gnome.org/projects/evolution/documentation.shtml

Vizslamail, magyar, ingyenes és megbízható levélszolgáltató:
➲ http://vizslamail.hu

Stellarium

Nemrég a Celestia nevű csillagászati programról írtam. Most egy kisebb volumenű, de nem kevésbé értékes planetárium-programot szeretnék bemutatni.

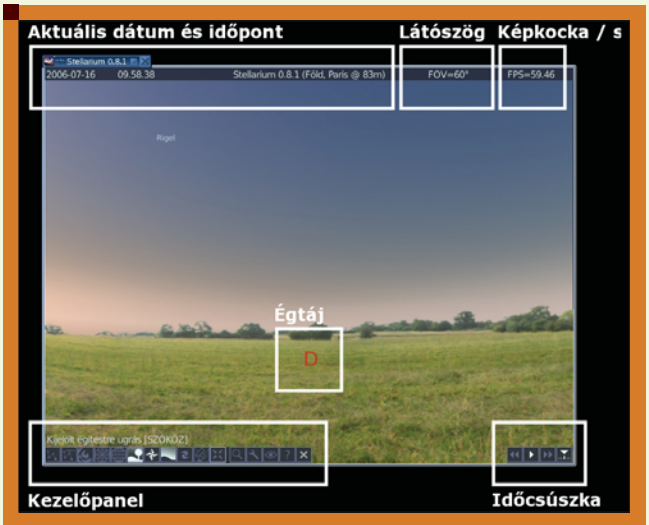
Amikor tudomást szereztem a *Stellariumról*, rövid ideig értetlenül álltam. Hogyan lehetséges az, hogy még nem hallottam e gyöngyszemről, miközben a csillagászat témája kezdő-hobbi szinten már régóta foglalkoztat? Aztán rögvest billentyűzetet ragadtam, és megkérdeztem a *Google* „mindenest”, mit tud erről a projektről! A válaszok alapján egy egészen régóta fejlesztett programról van szó, ami ugyan kisebb ismertségű, mint a *Celestia,* de saját területén kiemelkedő szerepet tölt be. Természetesen azonnal kipróbáltam, mire képes *Linux* alatt *Fabien Chereau* csapatának szabadon elérhető planetáriuma.

Üzembe állítás

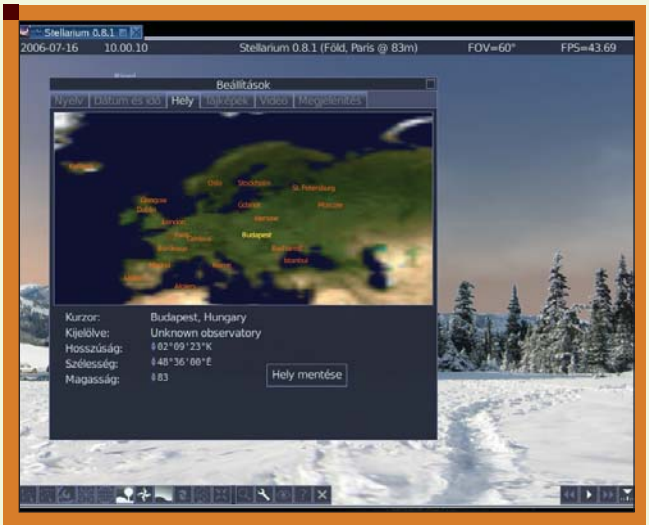
Látogassunk el a *Stellarium* hivatalos oldalára, a ➲ *http://www.stellarium.org* címre! Hathatós bevezető fogadja a „vándort”, nagyjából az alábbi tartalommal: *„E virtuális planetárium úgy képezi le az égboltot a monitorunkra, mint ahogyan azt szabad szemmel, illetve egy kis teljesítményű teleszkóppal látnánk a valóságban”*. Érdemes megnézni az aktuális képernyőfotókat. Önmagukért beszélnek.
Miután felocsúdtunk a látványos ábrák okozta bűvöletből, a letöltés szekcióban keressük a projekt *GPL* licenc szerint terjesztett forráskódját. A bináris felépítése a megszokott módon történhet, tehát *root* jogkörrel kiadott `./configure`, `make`, `make install` parancsokkal lehelhetünk életet a letöltött és kicsomagolt *tarballba*.
A konfiguráló szkript *GLU, SDL, PNG, Zlib* csomagokat keres függésként, így ezekkel mindenképpen rendelkeznünk kell *Linux* rendszerünkben. A futás további feltétele egy működő *GLX* (vagy *DRI)* kapoccsal ellátott *3D* grafikus hardver, valamint egy teljes értékű, korszerű központi egység. A telepített bináris egy felhasználóként terminálra gépelt `stellarium` paranccsal indítható el. A program minden beállítását és paraméterét kulturált módon, személyes mappánkban tárolja, a */home/$/.stellarium* rejtett könyvtárban. A projekt néhány helyen elérhető előre fordított bináris formában is, ám ezzel a megoldással nem voltak túl jó tapasztalataim. Az így megspórolt idő esetemben a stabilitással állt fordított arányban, tehát a telepítés egyszerűbb módját senkinek sem tudom javasolni.

Használatba vétel

Ha elindítjuk a virtuális planetáriumot, egy perspektivikus nézetű tájon találjuk magunkat, előttünk a horizonttal és a tiszta égbolt képével. Nézőpontunk helyzete a bal egérgombot lenyomva vonszolható,



1. ábra A grafikus interfész felépítése



2. ábra A megfigyelés helyszínének rögzítése

■ *3. ábra* A hullócsillagok feldobják az égboltot

■ *4. ábra* Születésem napján itt volt a Szaturnusz

fókusza pedig a görgőgomb segítségével változtatható meg. A vibráló csillagok mellett tekintsük meg a ***GLU*** könyvtáraira bízott grafikus interfészt! A képernyő bal alsó sarkában szolidan bújik meg a kezelőpult, jobb alsó részen az idő gyorsaságát szabályozhatjuk. A felső térben az aktuális időpont, a nézőpont információi és a látószög olvashatóak.
A könnyen értelmezhető lehetőségek között a légkör kapcsolhatósága, valamint a festett ábrák szorulnak leginkább magyarázatra. Az előbbi képesség nappali fényviszonyok mellett hasznos: a légkört kikacsolva a környezeti fényerő megszűnik, sötét teret engedve a csillagok vizsgálatának. Az ábrák ügyén pedig, a vonalakra feszített egyedi konstellációs rajzok ugyan bámulatosan feldobják az (amúgy is csodálatos) égboltot, de természetesen közel sem biztos, hogy bármely más programban hasonló ábrákat fognak találni az érdeklődők, legfőképpen úgy, hogy ezek a rajzok kultúránként minden bizonnyal mások lennének.
Ezeken felül a *„Kijelölt égitestre ugrás”* érdemel néhány szót: egy bolygót kiválasztva, a fókuszt egérgörgővel változtatva tisztességes közelségbe lehet hozni bármely égitestet, melyet a kódba programoztak (és vele együtt a holdjait is, már ha van neki). Apropó, mondtam már, hogy mennyi modellről van szó? A hivatalos verzió szerint nagyjából 120.000 égitestet implementáltak a ***Stellariumba!*** Ezek mozgása, dinamikája egyszerre idő- és valósághű,

```
[main]
version                 = 0.8.1

[video]
fullscreen              = false
screen_w                = 800
screen_h                = 600
bbp_mode                = 32
horizontal_offset       = 0
vertical_offset         = 0
minimum_fps             = 10
maximum_fps             = 10000
distorter               = none

# when you use distorter = fisheye_to_spheric_mirror
```

■ *5. ábra* Aprólékos beállítások a konfigurációs állományban

miközben textúráik is reálisak (természetesen csak ott, ahol a mai csillagászat rendelkezik adatokkal). A planetárium fő erőssége így abban rejlik, hogy a leképezett bolygók állásait és fázisait, a felettünk lévő csillagképek helyzetét minden időben nyomon tudjuk követni, nagyon látványos megjelenítéssel és valós tartalommal fűszerezve. A csillagkereső érzékenysége meglepően jó: mindamellett, hogy a legkisebb csillagokhoz is elvezet, a keresett égitest nevét is megpróbálja kitalálni helyettünk, az eddig leütött karakterekből.
A beállítási lehetőségek között a *„Nyelv”*, *„Helyszín”*, *„Időpont”*, *„Tájkép”*, *„Videó mód”* és a *„Megjelenítés”* hangoló ablakai találhatóak. Ezekről nem érdemes hosszabban szólnom, hiszen kézenfekvő dolgokról van szó: szerencsére minden felirat és üzenet lokalizálva segíti munkánkat. A megjelenítés, a helyszín és az idő kiemelt fontosságú paraméterek: példaként a hazai, éjszakai égbolton hullócsillagokat és konstellációkat kapcsolva maradandó élmény egy fontos időpontra (akár születésnapra) pozicionálni...
Az idő múlását pedig a reálisnál gyorsabbra állítva az égbolt folyamatos változása még lehengerlőbbé teszi ezt a vizuális élményt.
A menü lehetőségei között fellelhető a látómező vetületének meghatározása is, az alábbi sorrendben: perspektivikus, halszem, sztereográf, csavart tükör leképezés. (Ezek közül bármelyik választás a felhasználó szokásán fog alapulni, ennek ellenére a tükör módszernek nem látom a létjogosultságát.) Fontos, hogy némely verzióban a *„Videó mód”* menüben sajnos nincs lehetősége a teljes képernyős / ablakos futás kapcsolásának, így ezt a */home/$/.stellarium/config.ini* állományban lehet beállítani.
A ***Stellarium*** végtelenül profi munka: aki kicsit is vonzódik a csillagászathoz, minden bizonnyal kedvét fogja lelni benne. Nem titkolt célom minden érdeklődőt kipróbálásra buzdítani: a program békében megfér a ***Celestia*** mellett, elképesztően szép, és a hardverigénye sem magas. Ha hozzávesszem a listához azt is, hogy ***Linux*** mellett ***Win32*** és ***OS X*** alatt egyaránt életre hívható, akkor remélem senkiben sem marad kétely a projekt komolyságát illetően.
Tartalmas kikapcsolódást kívánok minden csillagvadásznak!

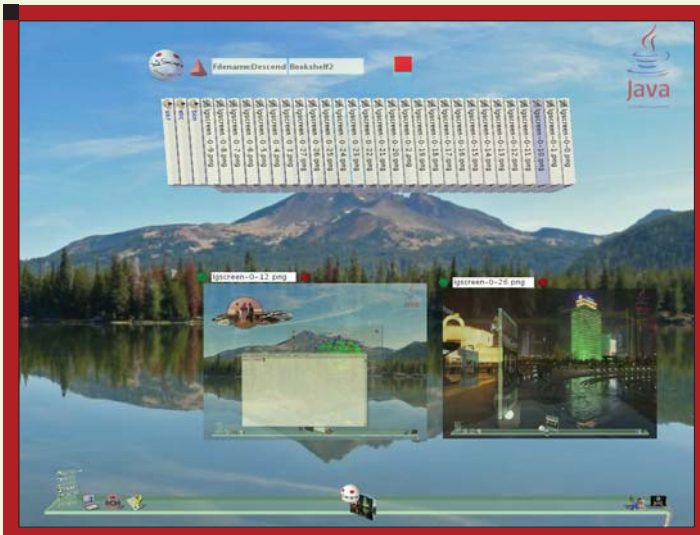
Kovács Zsolt (kovi@linuxforum.hu)

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

Project Looking Glass – Avagy a Sun különvéleménye a Linux desktopról

Egy egyetemi könyv és egy kisiskolásoknak szánt olvasókönyv között számos különbség fedezhető föl, de talán a legszembetűnőbb a küllem. Az olvasókönyv színes, tele van érdekes rajzokkal, nagy betűkkel nyomtatott, szereplői mesefigurák, ellentétben az hétköznapi könyvek „sivárságával". Hogy ez mért így van, arra nagyon egyszerű a válasz: az emberek – és különösen a gyerekek – szívesebben foglalkoznak azzal, ami érdekes, mint azzal ami egyhangú. Lássuk tehát ennek a pedagógiai elvnek a számítástechnikai megvalósítását...

Egyre inkább úgy tűnik hogy a szoftvercégek is igyekeznek ezt a pszichológiai tényt kihasználni, olyan szoftvereket fejleszteni amelyeket – enyhe túlzással élve – a felhasználók nemcsak egy probléma megoldására, hanem már magáért a felhasználás élményéjért is használnak. Mi sem jobb példa erre mint a szoftveróriás ***Microsoft*** megjelenés előtt álló operációs rendszere a ***Windows Vista***. A ***Vista*** legnagyobb újdonsága az ***Avalon*** névre hallgató grafikus felület. Ez elődeitől eltérően három dimenziós teret „teremt" a számítógép képernyőjén, az ablakokat három dimenzióban mozgathatod forgathatod, egészen újszerű módon egymás elé és mögé rendezheted. Ezzel természetesen nem merült ki az ***Avalon*** „trükktára" számos apróbb meglepetést tartogat felhasználói számára, olyanokat mint például az átlátszó ablakok.
A konkurens gyártok sem hagyhatják szó nélkül a ***Microsoft*** lépéseit.
A ***Novell*** saját fejlesztésű ***xgl*** szoftverének segítségével különleges, soha nem látott effekteket képes produkálni a felhasználó asztalán. Az ablakok rongyként mozognak áthelyezéskor, esőcseppek hullanak az asztalra, hogy csak a leglátványosabbakat említsem. Ezek a megoldások már viszonylag készek, a ***Vista*** végleges változata jövőév januárjában fog megjelenni, a ***Novell*** nemrég megjelent ***OpenSuse 10.1*** operációs rendszere már alapból támogatja az ***xgl/Compiz***-t.
Ezúttal egy kevésbé kiforrott (és egyenlőre sokkal kisebb sajtó visszhangot kapott), de annál innovatívabb elképzelést mutatok be, a ***Sun Microsystem*** által támogatott – és kezdetben fejlesztett – ***Looking Glass Projectet***.



1. ábra Az LgScope 3D fájlkezelő működés közben

Looking Glass

A ***Looking Glass*** egy ***Java***-ban írt 3 dimenziós asztali környezet. Különleges hardver – értsd ***3D*** szemüveg, stb. – nélkül képes 3 dimenziós teret teremteni a desktopon. A hagyományos, ***Windows***-szerű ablakkezelőktől eltérően az ablakokat nemcsak egymásra, hanem valóban egymás elé, mögé helyezhetjük, vagy ha éppen úgy tetszik el is forgathatjuk. De a ***Looking Glass*** még ennél is többet kínál. Amellett hogy visszafelé kompatibilis a jelenleg létező ***2D*** alkalmazásokkal, egy keretrendszert nyújt ***3D*** alkalmazások fejlesztéséhez.
Egy jó példa erre a ***LgScope 3D*** fájlkezelő alkalmazás (***1. ábra***).

2. ábra Épp az xterm ablak hátára írok jegyzetet

Az aktuális könyvtár fájljait és alkönyvtárait ikonok helyett egy egy téglatest reprezentálja. A téglatest fizikai mérete jelképezi a fájl méretét, de ha elforgatjuk a programot (mert egy 3 dimenziós térben lévő 3 dimenziós programmal ezt megtehetjük) akkor a téglalap alján megtaláljuk a pontos méretét, és még számos más adatot, amit egy hagyományos fajkezelő *Properties* ablakából tudhatunk meg. Duplán kattintva egy képet reprezentáló téglatestre az megjelenik, mindenféle keret nélkül, csupán a neve van fölé írva, ezzel azt a hatást keltve mintha egy papírkép lebegne a szemünk előtt.
A *Fel* gombnak egy forgáskúp felel meg.
Az eddig felsoroltak mellett számos kisebb de annál látványosabb elemmel rendelkezik a rendszer, például megfordíthatjuk az ablakokat, hogy a hátukra jegyzetet írjunk (*2. ábra*).
Még mielőtt részletesebben belekezdenénk *Looking Glass* megismerésébe, meg kell említenem hogy *Linux* mellet *Solarison* sőt tesztüzemmódban még *Windowson* is fut. A *Looking Glass* *GNU GPL* licenc alatt érhető el, tehát szabad szoftverről van szó.

Vágjunk bele!

A *Looking Glass* képességeihez mért hardver igénnyel rendelkezik, legalább *1,4 GHz*-es processzor és *512 MB* memória szükséges a működéséhez. Talán szokatlan egy desktop környezettől de a rendszerkövetelmények közt szerepel az 1.3-as vagy újabb *OpenGL*-t támogató videokártya is. Ha nem vagyunk benne biztosak, hogy az általunk használt a grafikus kártya vagy a jelenlegi szoftvereink támogatják az 1.3-as *OpenGL*-t akkor ellenőrizzük a *glxinfo* nevű program segítségével. (Néhány terjesztés esetében ezt külön kell telepíteni.)
Én egy *AMD Athlon XP 2000+* processzorral és *512 MB* memóriával szerelt számítógépet használtam a *Looking Glass* kipróbálásához.
A megjelenítésért egy *nVidia GF4 TI 4200*-as grafikus kártya felelt. Tekintettel arra, hogy egy 3 éves konfigurációról van szó, viszonylag jól futott rajta a *Looking Glass*, 2-3 program egyidejű futtatásakor még nem akadozott a megjelenítés.
A szoftveres követelmény nem túl nagy, valószínűleg bármely nagyobb *Linux* terjesztés megfelel a célra.
Én a *Debian Etchet* választottam.
Természetesen a *Looking Glass* nem tartalmazza az *X* szervert ezt előre kell telepíteni. (Én az *Xorg 7.0.22* verzióját használtam.)
Ha a feltételek adottak a *LG* kipróbálására akkor töltsük le a *Linux x86 Mega Bundle* csomagot a ➲ *https://lg3d-core.dev.java.net/binary-builds.html* honlapról.
A cikk írásakor a *0.8.1*-es verzió a legújabb stabil változat. Ha *Debiant*, *Ubuntut* vagy valamelyik hasonló disztribúciót használjuk, akkor érdemesebb a *Debian* csomagokat letölteni és azokat a *dpkg* segítségével telepíteni.
Tehát miután letöltöttük a *lg3d-release-0-8-1-linux-i686-megabundle.bin* fájlt, adjuk ki az

```
sh lg3d-release-0-8-1-linux-
➥i686-megabundle.bin
```

parancsot, amire elindul a telepítési folyamat.
Egyetlen dolgunk az lesz, hogy elolvassuk és elfogadjuk a megjelenő felhasználói engedélyt (licenc). Ha ezt megtettük, akkor már nem lesz más teendőnk, a telepítő ugyanis mindent automatikusan elvégez.
Ha *nVidia* grafikus kártya van abban a számítógépben amire a *LG*-ot telepítettük, akkor érdemes elolvasni a ➲ *https://lg3d.dev.java.net/nvidia-driver-install-tips.html* oldalt. Itt néhány tippet adnak a *Looking Glass* készítői.
Ha el szeretnénk távolítani a *LG*-t, akkor egyszerűen töröljük le a */usr/share/lg3d/*, a */etc/lg3d/* és a */usr/share/lg3d-jdk/* könyvtárat. Ezen kívül abban a könyvtárban, amiben futtattuk a telepítőt egy *lg3d* nevű alkönyvtár keletkezett. Ezt is távolítsuk el.

Próbáljuk ki

Abban a könyvtárban amiben a telepítőt futtattuk, adjuk ki a

```
lg3d/usr/share/lg3d/bin/
➥lg3d-session
```

parancsot. Ez elindítja a *Looking Glass* rendszert. Fontos, hogy a futó *X* szervereket leállítsuk, mielőtt ezt megtesszük.
Miután elindult az *LG*, az *1. ábrához* hasonlót láthatunk a monitoron.
A képernyő alján a tálcát találjuk (*3. ábra*).
A hagyományostól eltérően itt nem a képernyő aljához „ragasztott csíkról" hanem, ha egy térben lebegő üveglapról van szó. Ezen az „üveglapon"

Ha az *LG* kipróbálásához kiszemelt számítógépen nincsen böngésző vagy egyáltalán ablakkezelő, akkor a következő parancsokkal tölthetjük le a telepítéshez szükséges fájlokat:

Linux x86 Mega Bundle:

```
wget -c https://lg3d-core.dev.java.net/files/documents/1834/
➥36801/lg3d-release-0-8-1-linux-i686-megabundle.bin
```

Mindez *Debian* csomagban:

```
wget -c https://lg3d-core.dev.java.net/files/documents/1834/
➥36796/lg3d_jdk1.6.0_i686.deb
get -c https://lg3d-core.dev.java.net/files/documents/1834/
➥36797/lg3d_java3d_1.5.0_i686.deb
wget -c https://lg3d-core.dev.java.net/files/documents/1834/
➥36794/lg3d-core_0.8.1_i686.deb
```

3. ábra A Looking Glass tálcája is 3 dimenziós

4. ábra Ha az egérrel rámutatsz a menüre annak mérete megnő

5. ábra Néhány program futás közben,…

6. ábra …majd ugyanazok elforgatva

található a menü, amiből a programokat indíthatod, néhány indítóikon, középen az éppen futó programok kicsinyített másai, a jobb sarokban pedig a háttérkép választó alkalmazás ikonja és a kilépés gomb.

Próbaképen indítsuk el a bevezetőben már említett ***LgScope 3D*** fájlkezelőt. Ezt úgy tehetjük meg, hogy az egérrel rámutatunk a menüre (***4. ábra***), majd az ***Utilities*** menüpontra kattintunk.

Ezután válasszuk a ***LgScope 3D filemanager*** menüpontot. Ennek hatására elindul az ***LgScope 3D***, és listázza az aktuális könyvtár (az a könyvtár amiből az ***LG***-t indítottuk) tartalmát.

Ahogy korábban már utaltam rá, az ***LgScope 3D*** egy 3 dimenziós program, ezért elforgathatjuk a 3 dimenziós térben mint egy valódi, 3 kiterjedéssel rendelkező testet. Ezt úgy tehetjük meg, hogy a bal felső sarokban lévő ***Java*** logóra mutatunk az egérrel, majd a bal gomb nyomva tartása közben mozgatni kezdjük az egeret. Ennek következtében a futó – nem minimalizált – alkalmazások elkezdenek forogni annak megfelelően, ahogy az egeret mozgatjuk (***5.*** és ***6. ábra***).

A ***Looking Glassban*** találunk egy a háttérképekkel kapcsolatos újdonságot is: a hagyományos asztaltól eltérően itt panorámaképet is beállíthatunk háttérképként. Ez azt az érzést kelti, mintha mondjuk egy tó partján állva nézelődnénk (***7. ábra***).

Mivel a panorámakép sokkal szélesebb, mint a monitor, egyszerre nem látjuk az egészet, csak egy részletét. A képernyő jobb, illetve bal szélére kattintva változtathatunk, hogy épp melyik részt nézzük.

A kattintás hatására a kép egyszerűen „elfordul”. A három dimenziós tér érzetét fokozza az is hogy ha a képernyő széléhez érintjük az egérmutatót, akkor kicsit megrezdül a háttérkép.

Mivel a háttérkép egybefüggő, az alkalmazásokat egyszerűen áttolhatjuk a panorámakép egyik szeletéről a másikra. Ha pedig egyben szeretnénk látni a panorámaképet, akkor kattintsunk a tálcára a jobb egér-

7. ábra Egy a Looking Glassal érkező panorámaképek közül

gombbal. A kép ekkor – a futó programokkal együtt – összezsugorodik akkorára, hogy elférjen a monitoron. Az előbb már említett forgatás így is működik, és nagyon látványos (*8. ábra*). Az előző nézethez úgy tudunk visszatérni, hogy a panorámakép valamelyik szeletére kattintunk. A háttérkép választó alkalmazást a tálcáról tudod elindítani, jobbról a második ikonnal.

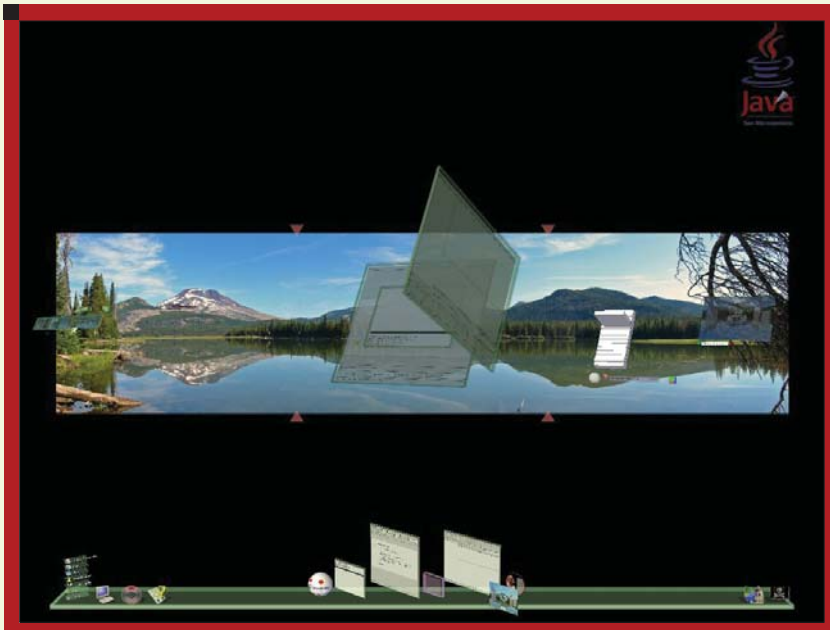
A ***Looking Glass*** visszafelé kompatibilis a jelenlegi 2 dimenziós alkalmazásokkal, azokat ablakban indítja el. Az ablakokhoz is számos látványos elem kapcsolódik, például ahogy már említettem megfordíthatjuk őket, hogy jegyzetet írjunk a hátukra. Ezt úgy tehetjük meg, hogy az ablakkeretre kattintunk az egér jobb gombjával. Én egy hiba miatt nem tudtam ezt kipróbálni, így azt sem sikerült kiderítenem, hogy azok a számítógép újraindítása után is megmaradnak-e.

A hagyományos minimalizálás lehetősége továbbra is adott, de nem ez az egyetlen módja annak hogy az épp nem használt ablakokat „félretegyük". Duplán kattintva az egér középső gombjával az ablakkeretre, az oldalra fordul és a képernyő széléhez simul (*9. ábra*).

Ha minden ablakot „félre akarunk állítani", akkor a képernyő jobb, illetve bal felére kell kattintanunk a jobb egérgombbal annak megfelelően, hogy melyik oldalra szeretnénk állítani őket. Középre visszaállítani úgy tudunk egy ablakot, hogy duplán kattintunk rá, vagy a tálcán lévő ikonjára egyszer.

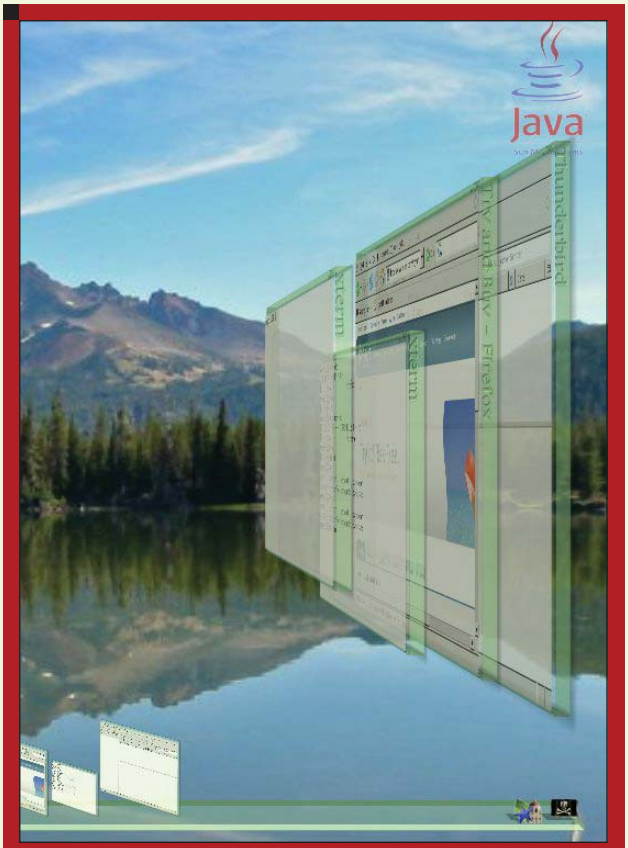
Ellentétben a hagyományos ablakkezelőkkel a 3 dimenziós térben nem csak átfedik egymást az ablakok, hanem valóban egymás mögött, illetve előtt helyezkednek el. (Ezt ellenőrizni tudjuk, ha elforgatjuk őket.) Éppen ezért ha egy ablak (vagy *3D* alkalmazás) nagyon távol van – például mert sok másik ablak van előtte – akkor kicsinek látszódik. Ha aztán rákattintunk, az előtérbe ugrik, visszanyeri az eredeti méretét. Az épp inaktív ablakok átteszőek, így látni lehet, hogy milyen programok vannak egymás mögött.

Érdekes effektus, hogy egy alkalmazás vonszolása (áthelyezése) közben – legyen az akár 2 vagy 3 dimenziós – picit elfordul.



8. ábra A forgatás így még látványosabb...

Kilépni úgy tudunk egy alkalmazásból, hogy a tálcán lévő kicsinyített mására jobb gombbal kattintunk.



9. ábra Az éppen nem használt ablakok oldalra fordítva várakoznak...

Hibák – néha azért fagy...

Alapvetően jó és minőségi szoftvernek tartom a ***Looking Glass***-t, de azért van néhány gyermekbetegsége is.

A legszembetűnőbb az volt, hogy bár a program a */usr* alá települ, innen elindítva lefagy. (Ezért is írtam korábban, hogy a telepítő mellet létrejött *lg3d* könyvtárból indítsuk a rendszert.)

Következő, viszonylag gyakori probléma az volt, hogy miután a kilépés gombra (halálfej a jobb alsó sarokban) kattintottam, szintén teljesen lefagyott a gép. Itt is van azonban gyógyír: használjuk a gomb helyett az ***Alt+Ctrl+Backspace*** billentyűkombinációt.

Harmadik nagyobb hiba a *3D* alkalmazások futtatása közben bukkant fel. Amikor például a *3D* fájlkezelőben egy képet meg akartam nyitni, arra az időre amíg a kép betöltődött az egész ***Looking Glass*** olyan volt,

mintha lefagyott volna. Még az egérmutató sem mozdult. Hasonló probléma más *3D* alkalmazásoknál is jelentkezett.
Van néhány kisebb hiba is. A *Looking Glassal* „érkezett" alkalmazások használata közben nem tudtam gépelni a szövegmezőkbe (épp ezért ezeknek a programoknak a nagy részét nem tudtam kipróbálni), a programok egyszerűen nem fogadták a billentyűzetről érkező bemenetet. Sajnos emiatt jegyzetet sem tudtam írni az ablakok hátára. Az igazsághoz persze az is hozzátartozik, hogy az általam használt *Debian* terjesztés *testing* kiadás volt, a *JVM* pedig amin az *LG* futott beta 2 állapotú.

A Sun különvéleménye a desktopról

Valószínűleg mindenki emlékszik *Tom Hanks Különvélemény* című *Sci-Fi* filmjéből arra a jelenetre, amikor a főszereplő egy nagy kivetítő előtt állva kezével és hangjával irányít egy számítógépet.
A *CeBIT 2006* informatikai kiállítás egyik szenzációja egy ehhez teljesen hasonló rendszer volt. A *Sun* és a *GoMonkey* közös standján minden érdeklődő Tom Hanks bőrébe bújhatott pár perc erejéig.
A technikai részletek közül nem sok minden szivárgott ki, a *Sun* és a *GoMonkey* is igen szűkszavúan fogalmaz honlapján, de annyi biztos, hogy a rendszer törzsét a *Looking Glass* adta. Ehhez párosult a *GoMonkey* által kifejlesztett mozgás és hangfelismerő rendszer, így a szokványos mutatóeszközök helyett kéz- és hangjelekkel lehett vezérelni a rendszert. Utóbbi két kamera segítségével ismerte fel alkalmi felhasználója kézmozdulatait.
A sikeresnek mondható bemutató ellenére, ez még inkább a jövőbe tett kirándulásnak tűnt, mint a holnap valóságának. Az ilyen forradalmi felhasználói felülethez, ugyanis megfelelő alkalmazások is kellenek, például nehéz elképzeli hogy hogyan működne az *OpenOffice.Org* egy ilyen környezetben.
Egy rövid felvétel található a *Sun* oldalán a bemutatóról a ➲ *https://lg3d-core.dev.java.net/files/documents/1834/31064/Sun_GoMonkey_demo_by_Oliver_Jones.avi* címen.

Összefoglalás

Hibái ellenére a *Looking Glass* ígéretes alkalmazás, nekem nagyon megtetszett a próbák alatt és a végén sem töröltem le a számítógépemről. Mindenkinek, aki szereti elsőként kipróbálni az új dolgokat javaslom, tegyen egy próbát a *Looking Glass* rendszerrel.

Szilágyi Attila (szati1@invitel.hu)

Néhány éve használ Linuxot. Alapvetően minden ezzel a témával kapcsolatos felhasználási terület érdekli és szívesen fogadja bárki kérdést, észrevételét.

KAPCSOLÓDÓ CÍMEK

A Sun honlapja:
➲ http://www.sun.com/

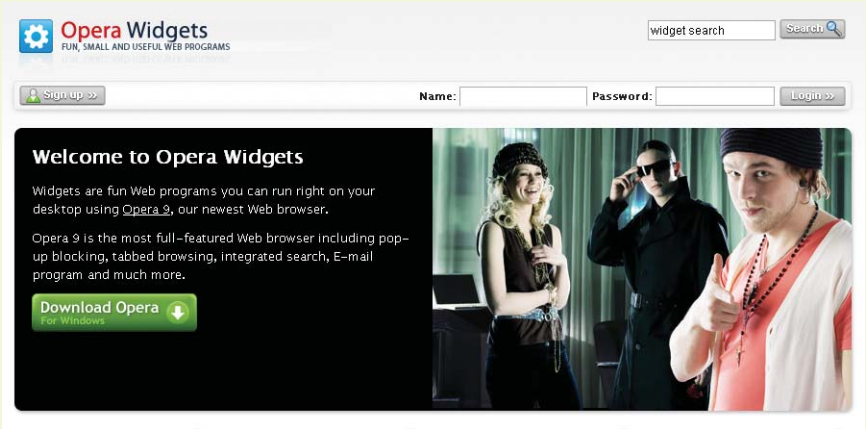
A Looking Glass honlapja:
➲ https://lg3d.dev.java.net/

A GoMonkey honlapja:
➲ http://www.gomonkey.at/

Az Opera új lehetőségei: widgetek

Az Opera egy zárt forrású, de ingyenesen használható böngésző, melyet a norvég Opera Software fejleszt. Az idén kiadott 9.0 változatban jelentek meg (több más újdonság mellett) az úgy nevezett webeszközök (widgetek), melyek kis AJAX (aszinkron kéréseket támogató, Javascripttel segített, XML kommunikációs) programok.

Az *AJAX* technológia lehetővé teszi, hogy olyan *webes* alkalmazásokat írjunk, amelyeknél a kliens és szerver közti kommunikáció aszinkron módon történik, s így olyan élményt nyújt a felhasználónak, mintha egy asztali szoftver használna.
Beszerzésük egyszerű: Az *Operán* belül *webeszközök/webeszközök* hozzáadása menüre kattintás után bejön az ➲ widgets.opera.com/ oldal és itt különböző feltételek szerint válogathatunk közöttük (szerző, típus, ország, népszerűség). A különböző lehetőségek között, jelenleg a játékprogramok a legnépszerűbbek.
Ha esetleg egy másik, nem megbízható helyről szereztünk be *widgetet*, és időközben meggondoltuk magunkat, a telepítés után a biztonság kedvéért még rákérdez a böngésző, hogy meg akarjuk e tartani az éppen akkor letöltött eszközt.
A feltelepített *widgetek* a webeszközök menüpont alatt találhatóak meg, elindításukhoz egyszerűen ki kell választani a megfelelőt a legördülő menüből. A programok egy külön webablakban jelenek meg a *desktopon*, az *Operát*ól nem teljesen függetlenek, mert, ha bezárjuk a böngészőt akkor az magával rántja az elindított webeszközöket is.
Törölni a *widgetek karbantartása...* menü alatt tudunk.
Az *Opera Software* ezen eszközök népszerűsítése érdekében, és közösségépítés céljából különböző nyereményeket ajánl fel *webeszköz-fejlesztőknek*. A jelenlegi



1. Lista A config.xml fájl tartalma

```
<?xml version="1.0" encoding="utf-8"?>
<widget>
  <widgetname>Lathatosag</widgetname><!- Program neve ->
  <description><!- Program leírása  ->
    Az első widgetem!
  </description>
  <width>130</width><!-  A megjelenő ablak maximális
  ➥ széllessége ->
  <height>100</height><!-  és magassága ->
  <author><!- Készítő adatai  ->
    <name>Fekete Imre</name><!- Készítő adatai  ->
    </author>
  <id><!-  A programhoz tartozó egyedi azonosító, mely három
  ➥ részből adódik össze: a hosztnév, a hoszt névnél  megadott
  ➥ tartományban egy egyedi név, és a megjelenés dátuma
  ➥ év-hónap formában (ÉÉÉÉ-HH). ->
     <host>pelda.com     </host>
    <name>pelda</name>
    <revised>2006-08</revised>
  </id>
</widget>
```

2. Lista A megfelelően formázott HTML fájl

```
<!DOCTYPE html>
<html>
  <head>
    <title>Lathatosag</title>
  <link rel="stylesheet" type="text/css" href="opa.css">

    </head>
    <body>
     <div id="torzs">
     Hello Linuxvilág! <!- Ez a szöveg fog megjelenni
     ➥gombnyomás után ->
     </div>
     <p class=gomb id=kapcsolo></p>
   </body>
 </html>


body {
  margin: 0;
  padding: 0;
  background: #ABC;
              }
/* "torzsben lévő szöveg elrejtése:"*/
#torzs{
 opacity:0.0;
}

.gomb{
 opacity: 1.0;
 background: transparent url(kepek/gomb1.png) scroll
 ➥no-repeat 0 0;
 height: 25px;
  width: 75px;
             }
body .gomb:active {
 opacity: 1.0;
 background: transparent url(kepek/gomb2.png) scroll
 ➥no-repeat 0 0;
}
```

versenybe is érdemes benevezni, mert 1000 euró a nyeremény, országonként. Ezen összeg eléréshez nem kell mást tenni, mint írni egy jó programot, és elérni, hogy az *Opera* oldaláról legalább 1000-szer letöltsék azt. Az alkalmazás milyenségétől függően ez a cél lehet könnyű illetve elérhetetlen cél.

Aki még nem készített ilyet, annak nyújtanék egy kis segítséget: Az általam bemutatott programot csak kiinduló pontnak szánom, melyben egy gomb lenyomása után egy addig láthatatlan (`opacity=0`) szöveg kiíródik (vagyis `opacity` tulajdonsága 1 lesz.) Egy *webeszköz* írásához *HTML*, *CSS* és *Javascript* ismeretre van szükség.

A programunk alapja egy *config.xml* nevű konfigurációs *XML* fájl, mely a következőképpen néz ki (*1. Lista*). Bővebb információ, valamint a XML fájl specifikációja a ➲ *oxine.opera.com/widgets/documentation/widget-configuration.html#config-xml* oldalon található.

A következő lépes, hogy felépítsük a programunk vázát, ami egy egyszerű *HTML* fájl, neve: *index.html*. A kinézetét pedig egy stíluslappal határozzuk meg, ez lesz az *opa.css*, melyben szerepel két 25X75px kép is, melyeket a `kepek` könyvtárba helyeztem el (*2. Lista*).

Jelenlegi állapot teszteléséhez egyszerűen rá kell húzni az *Opera* ablakára a *config.xml* fájlt. Ekkor megjelenik kék alapon a gombunk (*gomb1.png*), mely rákattintás idejéig lecserélődik a *gomb2.png*-re. Ha nem adunk meg háttérszínt, akkor csak a gomb jelenik meg, mivel alapesetben a *widget* háttere átlátszó, mely nem válaszol az egér eseményekre, egyszerűen továbbítja azokat az alatta levő alkalmazásnak.

Interaktivitást *Javascript* segítségével adunk a programhoz. Először is tegyük a *HTML* fájl *head* részébe a következőt:

```
<script type="text/javascript"
➥src="opa.js"></script>
```

Majd pedig a *opa.js* nevű fájlba mentsük el következőt, melyben egy eseményt rendelünk a kapcsoló nevű gombhoz:

```
window.addEventListener('load',
➥function(ev){

document.getElementById
➥('kapcsolo').addEventListener
➥('click',function(ev){
  ;document.getElementById
➥( 'torzs' ).style.opacity=1;
  }, false);},false);
```

Ezután, ha lenyomjuk a gombot megjelenik az eddig elrejtett szöveg. Remélem ez kis program jó kiindulópont további *widgetek* írásához.

Az *Opera* oldalára való feltöltés előtt a webeszközünkhöz tartozó összes fájlt be kell csomagolni egy egyszerű *zip* fájlba, de *zip* helyett *wdgt* kiterjesztés kell kapnia.

Fekete Imre
(imre.fekete@gmail.com)

Programtervező-matematikusként végeztem a Debreceni Egyetemen. A Linuxtól kezdetben idegenkedtem, de ma már csak azt tudom mondani róla, hogy remek rendszer.

Első lépések a Condor rendszerrel

Hogyan kezdjünk barátkozni ezzel a több platformon elérhető, elosztott számításokat lehetővé tevő rendszerrel...

A klaszterek használatának gondolata valamikor az 1990-es években született meg, amikor a hardverárak hirtelen esni kezdtek, a *PC*-k pedig egyre nagyobb teljesítménnyel bírtak. A cégek a valójában nagy méretű, ám mini-nek nevezett számítógépek használatáról kezdtek áttérni a már valóban kicsi és ezért „mikro" előtaggal illetett eszközök használatára. Ezzel párhuzamosan többen felismerték azt az érdekes ellentmondást, hogy az új, nagyobb teljesítményű gépek számítási kapacitása az idő nagy részében kihasználatlan marad, hiszen míg régen a teljesítmény egyetlen nagy dobozban volt jelen a cégnél, addig az új korszakban számos doboz között „fragmentálódott". Ma egy komolyabb vállalat asztali gépek százaival, vagy akár ezreivel rendelkezik, amelyek az idő túlnyomó részében kihasználatlanul állnak. Nem ritka ugyanakkor, hogy ugyanezek a vállalatok bizonyos feladatok elvégzéséhez, vagy egyszerűen csak versenyképességük megőrzéséhez komoly számítási teljesítményt igényelnek, vagyis egyszerre van jelen a rendszerben a pazarlás és az „éhezés". Éppen ez az az effektus, amely folyamatosan fenntartja manapság az érdeklődést a szuperszámítógépes megoldások és a klaszterek iránt.

Számos gyártó kulcsrakész klasztermegoldásokat kínál, persze nem föltétlen olcsón. Ugyanakkor szabadon használható, nyílt forrású eszközök segítségével az sem lehetetlen, hogy kihasználjuk az eleve meglevő kapacitásokat gyakorlatilag anélkül, hogy egyetlen fillért kellene költeni beruházásra. A megfelelő szoftverek segítségével megépíthetjük saját klaszterünket a saját gépeinkből. Ebben a cikkben egy a ***University of Winsconsin*** által kidolgozott ilyen megoldásról, nevezetesen a ***Condorról*** lesz szó.

A *Condor* alapötlete rendkívül egyszerű: telepítenünk kell valamennyi gépre, amit a klaszter részeként szeretnénk használni, a többi meg már megy magától. A *Condor* terminológiájában a klasztert *pool*-nak (kb. közös készlet) szokás nevezni, így ebben a cikkben a két kifejezés egyenértékűnek számít. A telepítés után a *pool* bármelyik gépéről indíthatunk számítási feladatokat (*job*). A *Condor* megvizsgálja a program igényeit, összeveti az aktuálisan rendelkezésre álló szabad erőforrásokkal, és oda irányítja a végrehajtást, ahol azt a leghatékonyabban oldható meg. Ha megtalálta a megfelelő gépet, a *jobot* áthelyezi arra, megvárja, amíg lefut, majd begyűjti az eredményeket. A *Condor* egyik legnagyobb előnye tehát az, hogy a klaszter használatához egyáltalán nem kell átírni a meglevő alkalmazásokat.

A gyakorlatban persze a dolog azért valamivel bonyolultabb. Először is a *Condort* a különböző gépekre másként kell telepíteni. Minden *Condor poolnak* van egy központi kezelője (***central manager***), amely – nevének megfelelően – a klaszter belső adminisztrációját végzi. Ez a gép figyeli, hogy a pool melyik tagján vannak szabad óraciklusok, és ez végzi a *jobok* igényeinek illetve az erőforrásoknak az összehangolását is. Egy *Condor pool* ezen kívül tartalmazhat úgynevezett ***„Submit"*** és ***„Full install"*** típusú gépeket. Az előbbi csoportba tartozók beküldhetnek futtatandó feladatokat, de maguk nem futtathatnak egyetlen ilyen programot sem. A másik csapatot értelemszerűen azok a gépek alkotják, amelyek mindkét művelettípusra fel vannak jogosítva.

Igények és telepítés

Ami a hálózatot illeti, a *Condor* használatához nincs szükség semmilyen új hálózati elemre, a már meglevő tökéletesen elegendő. A *Condor* számos különböző operációs rendszerrel képes együttműködni. Lehetőségeink a következők: *Linux*, *Solaris*, *Digital Unix*, *AIX*, *HP-UX*, *Mac OS X*, *MS Windows 2000* és *XP*. Ami a különböző hardver-architektúrákat illeti, ezen a téren is van néhány választási lehetőségünk: *Intel x86*, *PowerPC*, *SPARC* stb. Persze a rendszer működési logikájából az is következik, hogy az egy adott architektúrára fejlesztett *jobok* csak ugyanilyen gépeken lesznek képesek futni, vagyis egy *Intel x86*-ra fordított programtól ne várjuk el, hogy bármi máson fusson. Ebből pedig nem túl nehéz levonni azt a következtetést, hogy a *Condor* klasztereket érdemes egy adott architektúrából összeválogatni. Azért egy kivétel itt is akad: a Java alkalmazások történetesen a különböző architektúrák között is átvihetők, ezekre a fentiek tehát nem vonatkoznak.
Ebben a cikkben kizárólag a *Linuxra* való telepítésről lesz sz, mégpedig a úgy, hogy a platformfüggetlen tarlabdából indulunk el. Természetesen léteznek az egyes operációs rendszerekre illetve architektúrákra specifikus csomagok is, amelyekről bővebb információt a ➲ www.cs.wisc.edu/condor/downloads címen találhatunk.
Töltsük tehát le a platformfüggetlen csomagot erről a helyről, majd tömörítsük ki a következő paranccsal:

```
tar -zvf condor.tar.gz
```

A telepítéshez egyetlen dolgot kell tennünk: lefuttatni az ***sbin*** könyvtárban található `condor_install` nevű szkriptet. Mielőtt azonban így tennénk, hozzunk létre egy `condor` nevű felhasználót is a rendszeren. Biztonsági megfontolások miatt a *Condor* nem engedi, hogy bárhol a *root* nevében futtassunk programokat, így szükségünk lesz egy olyan közönséges felhasználói fiókra, amelynek nevében a dolgok bonyolódnak.
Az első kérdés, amit a szkript nekünk fog szegezni úgy hangzik, hogy hány gépből szeretnénk klasztert létrehozni. Ennek a kérdésnek igazán akkor van jelentősége, ha megosztott fájlrendszert használunk, ilyenkor ugyanis a szkript bekéri az összes gép nevét, majd a *Condor* telepítését azokon is automatikusan elvégzi. Ebben az esetben tehát ezekkel nekünk már nem kell foglalkoznunk. Ha nem használunk elosztott fájlrendszert, akkor a telepítést manuálisan kell mindenütt elvégezni. Ha Java alkalmazásokat is szeretnénk futtatni, akkor szintén telepítenünk kell mindenhova a Sun Java virtuális gépét is.
Az esetlegesen felmerülő problémákkal kapcsolatban maga a telepítőszkript is rengeteg segítséget nyújt, hiszen van súgója, illetve minden feltett kérdéshez tartozik magyarázó szöveg is. Ha pedig ezek nem segítenek, bármikor rendelkezésünkre áll a részletes felhasználói kézikönyv, illetve az alkalmazással foglalkozó levelezési lista.
A továbbiakban feltételezzük, hogy a `$CONDOR` változó tartalmazza annak a könyvtárnak az elérési útvonalát, ahova a *Condort* kibontottuk. Telepítés után a rendszert a következő paranccsal indíthatjuk el:

```
$CONDOR/bin/condor_master
```

Ez a parancs az összes olyan folyamatot elindítja, amelyekre a *Condor* működéséhez szükség van. Ez ezt jelenti, hogy a következő parancsot kiadva a központi kezelőn (*central manager*) összesen öt olyan folyamatot kell látnunk, amelynek neve a `condor_` előtaggal kezdődik:

```
ps -aux | grep condor
```

Az öt folyamat a következő:

- `condor_master`
- `condor_collector`
- `condor_negotiator`
- `condor_startd`
- `condor_schedd`

A *pool* minden más gépén a következő folyamatoknak kell futni a helyes működéshez:

- `condor_master`
- `condor_startd`
- `condor_schedd`

Végül a *submit-only* gépeken csupán két folyamatra van szükség:

- `condor_master`
- `condor_schedd`

Ha mindezek után kiadjuk a `condor_status` parancsot, akkor a központi gépet már mint a *pool* egyik tagját fogjuk látni:

```
$CONDOR/bin/condor_status
Name OpSys Arch State Activity
➥LoadAv Mem ActvtyTime
➥Mycluster
LINUX INTEL Unclaimed Idle
➥0.115 3567 0+00:40:04
Machines Owner Claimed
➥Unclaimed Matched Preempting
INTEL/LINUX 1  0  0  1  0  0
Total 1  0  0  1  0  0
```

Ha pedig a klaszter egyéb gépein is elindítjuk a `condor_master` folyamatot, akkor néhány percen belül azok is mint a *pool* tagjai fognak már megjelenni.
(Ez általában körülbelül öt percet vesz igénybe.)

Folyamatok indítása az új klaszteren

A klaszter teszteléséhez először is hozzunk létre egy egyszer „Hello Condor” alkalmazást:

```
#include
int main()
{ printf("Hello World!\n");}
```

Fordítsuk le a programot a *GCC* segítségével, majd a keletkezett bináris állományt indítsuk el a *Condor* fennhatósága alatt. Ehhez meg kell írnunk egy indítófájlt (***submit*** fájl).
Az indítófájl egy olyan információcsomag, amelyben meg kell adnunk a *Condornak*, hogy a kérdéses programot miként kezelje.
Ez a fájl tartalmazza tehát, hogy a program honnan veszi a bemenetét, hova írja a kimenetét, illetve hogy miként jelezze, ha hiba keletkezett, hol tárolja a hibaüzeneteket.
Esetünkben az indítófájl tartalma a következőképpen fest:

```
Universe = Vanilla
Executable = hello
Output = hello.out
Input = hello.in
Error = hello.err
Log = hello.log
Queue
```

Az első, ***Universe*** nevű bejegyzés azt adja meg, milyen környezetben kell a ***Condornak*** futtatni a kérdéses feladatot. Két említésre méltó ilyen „univerzum” van. A hosszú futásidejű dolgokhoz a `Standard` nevezetű használatos. (Jelen esetben a „hosszú” futásidő heteket vagy hónapokat is jelenthet.) A `Standard` univerzumnak ugyanis számos, a cél szempontjából igen kellemes tulajdonsága van. Képes elmenteni a futó program állapotát, és szükség esetén a „mondat közepén” folytatni annak a futtatását, sőt ha egy gép működésében hiba keletkezik, akkor képes más helyre átköltöztetni a rajta futó feladatokat. Ez természetesen óriási segítség lehet bizonyos helyzetekben, de van egy hátulütője: a `Standard` univerzum használatához magának az alkalmazásnak is ***Condor*** kompatibilisnek kell lennie, vagyis bele kell fordítani bizonyos kiegészítő rutinokat. Erre pedig nyilván csak akkor van lehetőség, ha rendelkezésünkre áll a forráskód.
A másik, `Vanilla` nevű univerzumot a rövidebb lélegzetű feladatokhoz szokás használni, de természetesen a hosszú futásidejűekhez is alkalmazható akkor, ha a gépek működése kellően stabil. Az előny itt az, hogy egyáltalán nem kell módosítani a binárisokat.
Léteznek a ***Condor*** alatt más univerzumok is. Van például `PVM`, `MPI` és `Java` univerzum, amelyek értelemszerűen a kérdéses technológiákra támaszkodó alkalmazások futtatásához használhatók. Ezekkel kapcsolatban a ***Condor*** dokumentációjában találunk bővebb információt.
Esetünkben a végrehajtható állomány neve `hello` (a hagyományos „Hello Condor” program), az általunk használt univerzum pedig a `Vanilla`.
Az `Input`, `Output`, `Error` és `Log` direktívák azt állítják be, hogy a ***Condor*** mely fájlokat kapcsolja a futó program ***stdin***, ***stdout*** és ***stderr*** csatornáihoz.
Végezetül a `Queue` direktívában adhatjuk meg, hogy a program hány példányban futtatható.
Ha elkészültünk a futtatás körülményeit leíró fájllal (***submit file***), akkor magát a végrehajtást a

```
condor_submit hello.sub
```

paranccsal kezdeményezhetjük.
A futó folyamat állapotát a `condor_q` paranccsal ellenőrizhetjük, amely kiírja, hogy a végrehajtási sor hány programot tartalmaz éppen, mi ezeknek az azonosítója (***ID***), valamint hogy éppen futnak, vagy végrehajtásra várakoznak. A program ezen kívül néhány statisztikai adatot is közöl a folyamatokról.
Bár ebben a cikkben egyelőre csak a ***Condor*** telepítésének és üzembe helyezésének részleteit tárgyaltuk, a rendszernek természetesen számos a gyakorlatban használható funkciója van, amelyekről mindenféle oktatóanyagokat találhatunk az interneten. Az elsődleges információforrás természetesen a ***Condor*** felhasználói kézikönyve, amit a ➲ www.cs.wisc.edu/condor/manual webcímen találunk meg. Ezt olvasgatva célszerű különös figyelmet szentelni a Standard és a Java univerzumnak. Az előbbi lehetőséget ad a folyamatok időnkénti ellenőrzésére, míg a másikkal – nevének megfelelően – ***Java*** alkalmazásokat futtathatunk.

Szintén hasznos, ha a *Condor* már a bootfolyamat részeként elindítjuk, különösen, ha gyakran használjuk a gépeket „klaszter üzemmódban”. A dolognak ráadásul az a haszna is megvan, hogy így ha lekapcsolunk egy munkaállomást, miközben fut rajta egy *Condor* folyamat, akkor annak biztosan lesz hova átköltözni (persze attól függően, hogy a `Standrad` vagy a `Vanilla` univerzumban fut). Ez pedig nagy szabadságot kölcsönöz a rendszergazdának az adminisztrációval kapcsolatban.

Túl a klasztereken

A *Condor* nem csak arra jó, hogy segítségével klasztereket hozzunk létre közönséges asztali gépekből. A rendszer egyik kiegészítése azt teszi lehetővé, hogy számítási feladatok ne csak egy adott klaszter gépei között tudjanak „szabadon költözni”, hanem akár két klaszter között is. A *Condor* nevezéktanában ezt a szolgáltatás *flocking*-nak (kb. falkába tömörülés, fürtözés) hívják, a lényege pedig az, hogy ha abban a *pool*-ban, ahol a feladatot elindítottuk, éppen nem áll rendelkezésre a szükséges számítási kapacitás, akkor a job automatikusan megtalálja magának az utat egy másik klaszterbe. Ez pedig végső soron egészen érdekes konfigurációk megalkotását teszi lehetővé.
A legegyszerűbb ilyen *flocking* konfiguráció létrehozásához nincs másra szükség, mint a *condor_config* fájlban megadni néhány speciális változót. Tegyük fel például, hogy van két klaszterünk, A és B, és azt szeretnénk elérni, hogy az A klaszteren elindított feladatok szükség esetén a B klaszterre költözzenek át. Tegyük fel továbbá, hogy az A klaszter vezérlőközpontja az *a.condor.org* címen található, míg a B klaszetré a *b.condor.org*. Ezekkel a konfigurációs fájl megfelelő része a következőképpen fest:

```
FLOCK_TO = b.condor.org
FLOCK_COLLECTOR_HOSTS =
➥$(FLOCK_TO)
FLOCK_NEGOTIATOR_HOSTS =
➥$(FLOCK_TO)
```

A `FLOCK_TO` változóban akár több *pool*-t is megadhatunk vesszővel elválasztva a megfelelő központi vezérlők neveit. A másik két változó általában ugyanoda mutat, mint a `FLOCK_TO` változó tartalma. A B klaszter konfigurációs fájljaiban olyan értékeket kell megadnunk, hogy azok fölhatalmazzák az A klaszter folyamatait a B klaszter csomópontjain való futásra. A következő példa egy ilyen beállítást mutat. Itt a `FLOCK_TO` változóhoz meglehetősen hasonló nevű `FLOCK_FROM` paraméter szolgál azoknak a „baráti” klaszereknek a felsorolására, ahonnan végrehajtási kérések érkezhetnek.

```
FLOCK_FROM=a.condor.org
HOSTALLOW_WRITE_COLLECTOR =
➥$(HOSTALLOW_WRITE),
➥$(FLOCK_FROM)
HOSTALLOW_WRITE_STARTD =
➥$(HOSTALLOW_WRITE),
➥$(FLOCK_FROM)
HOSTALLOW_READ_COLLECTOR =
➥$(HOSTALLOW_READ),
➥$(FLOCK_FROM)
HOSTALLOW_READ_STARTD =
➥$(HOSTALLOW_READ),
➥$(FLOCK_FROM)
```

A fenti beállítások tehát engedélyezik az A klaszter gépeinek, hogy a B csomópontjain fussanak, de ugyanez visszafelé már nem működik. Ha ezt is meg akarjuk oldani, akkor teljesen hasonlóan kell eljárnunk, csak a változókat és a szerepeket kell mindenütt megcserélni. A B klaszteren tehát a `FLOCK_TO`, `FLOCK_COLLECTOR_HOSTS` és a `FLOCK_NEGOTIATOR_HOST` változókban az A pool központi vezérlőjének címét kell megadnunk, míg az A klaszeteren a `FLOCK_FROM` változóval engedélyeznünk kell a B-ről érkező folyamatok végrehajtását.
Ügyeljünk a `HOSTALLOW_WRITE` és a `HOSTLLOW_READ` változók beállítására, ezek ugyanis azt határozzák meg, hogy mely gépek csatlakozhatnak az adott *pool*-hoz, illetve melyek azok a csomópontok, amelyek a klaszter állapotáról információt kérhetnek le, de nem csatlakozhatnak hozzá.
A *Condor* egy kifejezetten hajlékony módszereket kínál a gépek és jogosultságaik megadására. A következő beállítással például lehetőségünk van arra, hogy csak egy adott alhálózathoz tartozó gépek számára biztosítsunk olvasási jogosultságot:

```
HOSTALLOW_READ=127.6.45.*
```

Condor-G

A *Condor* segítségével kialakított klaszterek egymáshoz kapcsolásának másik módja a rendszer grides képességeinek kiaknázása. A *Condor* e tekintetben a *Globus Toolkitre* (➲ www.globus.org) támaszkodik. Ez egy olyan nyílt forrású eszközkészlet, amellyel *Grid*-en futtatható rendszereket és alkalmazásokat készíthetünk. A csomagban megtaláljuk a megfelelő szoftveres infrastruktúrát a folyamatok hitelesítéséhez, a jogosultságok kezeléséhez valamint a távoli eljárások indításához és az ehhez szükséges adatátvitel megvalósításához. A *Condor-G* egy olyan kiegészítés a *Condor* alaprendszerhez, amelynek segítségével alkalmazásaink *Grid*-képesek lesznek, vagyis képessé válnak távoli, csak a *Grid* valamely pontján megtalálható erőforrások használatára is.
A `Condor-G` tulajdonképpen nem más, mint egy átjáró a *Condor pool*-ok és a *Grid* között. Ez az a program, amely egyaránt kezeli a futtatási sorokat, valamint mindazokat az erőforrásokat, amelyeket az elindított folyamatok használni fognak, legyenek azok akár egyetlen klaszter gépein, vagy bárhol másutt a *Griden*. Feladata, hogy a *Globus* mechanizmusait használva biztosítsa az erőforrások és folyamatok közötti kommunikációt, és szükség esetén a fájlok mindkét irányú átvitelét. Aki többet szeretne megtudni a *Condor-G* használatáról, lapozza fel a *Condor* korábban már említett kézikönyvét a megfelelő fejezetnél.
Egy a *Globus* segítségével végrehajtható feladata indítási állománya a következőképpen nézhet ki:

```
executable = mygridjob
globusscheduler =
➥grid.sample.net/jobmanager
input=mygridi.txt
universe = globus
output = mygridjob.out
log = mygridjob.log
queue
```

Amint látható, mindössze két eltérés van a *Griden* illetve a helyi klaszteren (*pool*) végrehajtandó feladatok leíróállománya között. Először is az az univerzum, amelyben a grides alkalmazás fut a *Globus* nevet viseli.

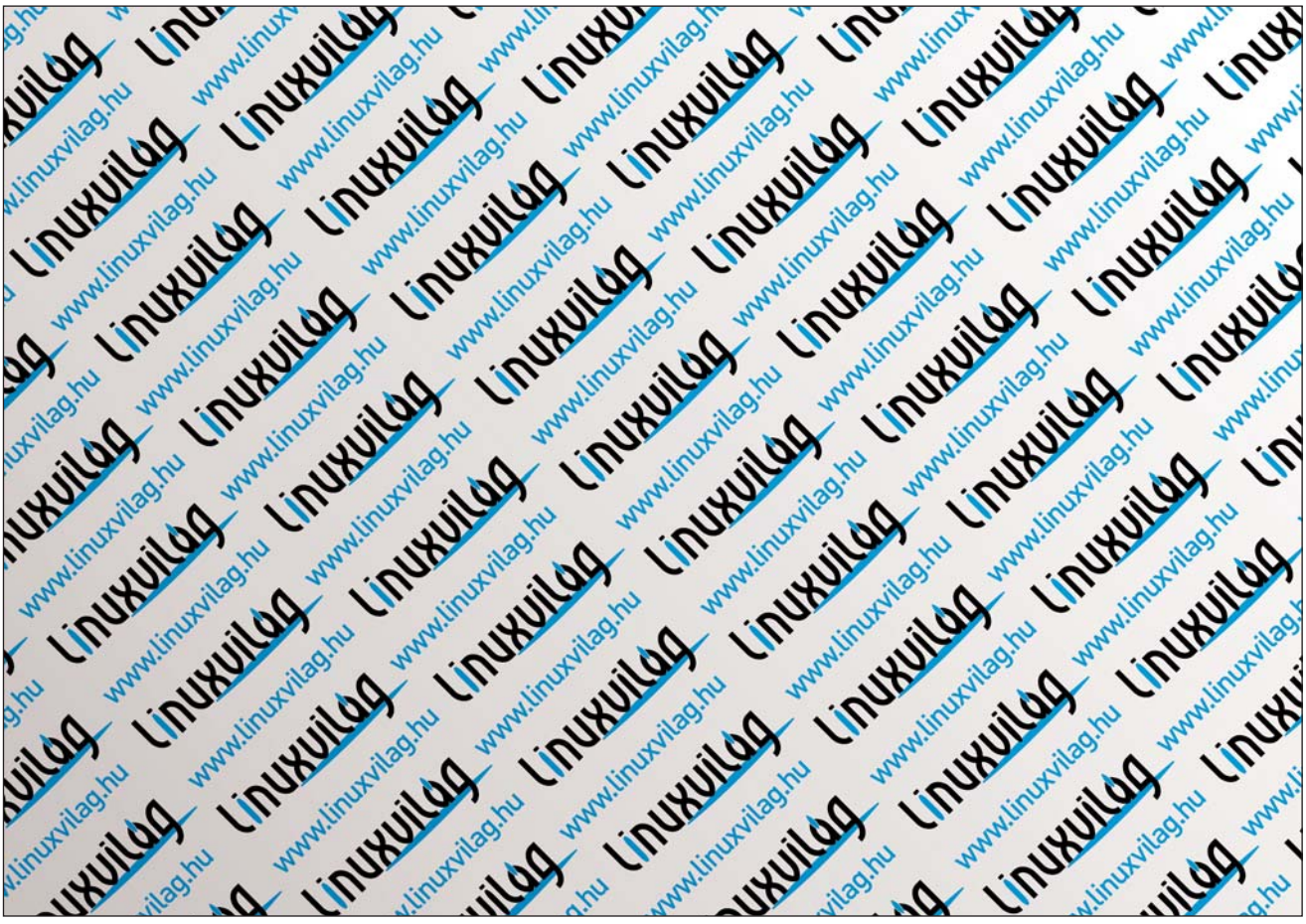
Ez az a beállítás, amelyből a *Condor* tudja, hogy az adott *jobot* a *Globus* segítségével kell elindítania és a *Griden* kell futtatnia. Ennek megfelelően meg kell adnunk a *Globus* ütemező (*Globus Job Manager*) címét is a `globusscheduler` nevű változóban. Ez az ütemező egy a távoli gépen futó folyamat, és az a feladata, hogy folyamatosan kövesse a Griden futó alkalmazások *I/O* műveleteit, általános állapotát, és kezelje azok indítását. A Griden futtatott számítások megfigyelését ugyanúgy a `condor_q` paranccsal végezhetjük, mint a helyi folyamatok esetében.

Összefoglalás

Összességében elmondhatjuk, hogy a *Condor* egyedülálló lehetőséget kínál arra, hogy meglevő számítástechnikai infrastruktúránk segítségével olyan feladatokat oldjunk meg, amelyek az egyes egyes gépek képességeit messze meghaladnák. Telepítése és használata egyaránt könnyű, segítségével szinte pillanatok alatt építhetünk klasztert. Az így kialakított rendszer ráadásul skálázható, hiszen a *Condor* segítségével nem csak újabb csomópontokat csatolhatunk egy adott *pool*-hoz, hanem több ilyen klasztert is összefoghatunk egy fürtbe, sőt a megfelelő kiegészítésekkel a rendszer még a Grid részeként is funkcionálhat. Ennek megfelelően talán nem meglepő, hogy a *Condort* immár számos párhuzamos feldolgozást igénylő projekt megvalósítása során használták. Az egyik legújabb ilyen sikeres esettanulmány a *Micron Technologies*-től származik. A *Micron* a világ egy legnagyobb félvezető elemek gyártásával foglalkozó vállalata. 2006 áprilisában egy a *GridToday*-ben megjelent, a vállalat egyik vezető munkatársával készült interjú szerint nemrég megépítettek egy olyan rendszert, amely 11 *Condor* klaszterből, és összesen 11.000 processzorból áll. A gépek ráadásul a cég 11 telephelyén, négy különböző országban találhatók. A kérdésre, hogy miért éppen a *Condort* választották az illető azt válaszolta, hogy ennek számos oka volt, de elsősorban azért, mert a *Condor* minden, a vállalat számára érdekes platformot támogatott, széles körben használt, s így rendelkezésre álltak a szükséges információk az előzetes tervezéshez, jó a támogatottsága, no és persze mert nyílt forrású. A *Condor* létrehozott párhuzamos számítási kapacitás mára a *Micron* egyik igen komoly és értékes eszközét jelenti, amit a gyártásban, a tervezésben, a szoftverfejlesztésben, a biztonsági rendszer működtetésében, de még a kimutatások készítésében is felhasználnak. Elmondható tehát, hogy a *Condor* nem egyszerűen egy kísérleti eszköz, hanem olyasmi, ami a való életben is kiválóan megállja a helyét.

Irfan Habib

Egyetemi hallgató Pakisztánban a Nemzeti Műszaki és Tudományegyetem szoftvermérnöki karán. Évek óta komolyan érdeklődik a szabad és nyílt forrású technológiák iránt, kutatási munkáját pedig az elosztott és hálózati számításokkal kapcsolatos területen folytatja. Éppen innen ered a Condorral kapcsolatos érdeklődése is, hiszen ez mindkét említett területtel kapcsolatban áll. Irfant az irfan.habib@niit.edu.pk címen érhetjük el.



Heterogén feldolgozás: egy lehetséges stratégia Moore törvényének kiterjesztésére

Ebben a cikkben egy olyan kezdeményezést és a lehetséges fejlődési utakat kívánjuk bemutatni, amelyek lehetőséget teremthetnek a számítástechnikában közismert Moore törvény érvényességének meghosszabbítására.

Az alkalmazások teljesítményének folyamatos növelése olyasmi, amit mindenki akar. A nagy teljesítményű számításokkal foglalkozók körében (***High Performance Computing; HPC***) ráadásul ez nem is egyszerűen csak vágy, hanem egyenesen elvárás.
Az igazság az, hogy talán egy kicsit el is vagyunk kényeztetve ezzel az üggyel kapcsolatban, hiszen az elmúlt négy évtizedben ***Moore*** törvényének köszönhetően valóban folyamatos fejlődésnek lehettünk tanúi.
És bár ez a törvény immár negyvenedik évét tapossa, még mindig érvényesnek tűnik, vagyis a processzorok tranzisztorainak sűrűsége minden 18 hónapban megduplázódik. Persze a figyelmesebb szemlélők már észrevették, hogy gyülekeznek a viharfelhők. A tranzisztorok számának növekedése egy ideje már nem produkál hasonló eredményt a teljesítmény növekedésében. Ennek oka pedig szintén közismert: a több tranzisztor összekapcsolásához több vezetékre van szükség, ez pedig egyben több késleltetést is jelent a rendszerben. Hasonló problémát jelent a memória tartalmának gyors elérése. Ha újabb trükköket vetünk be az egymagos processzorok tervezése során, azzal elkerülhetetlenül növeljük azok összetettségét, és az általuk termelt hőt. Végezetül a skalár processzorok eleve magukban hordozzák a fejlődés gátját, hiszen működési alapelvük a gépi utasítások egymás után történő végrehajtása, ami a bonyolultság egy bizonyos fokán túl rendkívül megnehezíti az utasításszintű párhuzamosítások (***Instruction Level Parallelism; ILP***) felfedezését és kihasználását.
A fent felsorolt problémák pedig immár nem csak a felhasználók azon szűk körét érintik, akik a legnagyobb teljesítményt is képesek azonnal kihasználni, sőt, ez a bizonyos kör talán nem is volt soha szűk, épp csak a helyzet maga akadályozta a növekedését. Manapság egyre világosabb, hogy a számítási teljesítmény növelése gyakorlatilag valamennyi tudományterületre jótékony hatással lenne.
Az ***Elnöki Információtechnológiai Tanácsadó Testület (Presidents Information Technology Advisory Committee)*** éppen ezért máris megkeresett számos, a ***HPC*** területen dolgozó kutatót azzal az ötlettel, hogy 2010-re valós alkalmazások számára is elérhetővé kellene tenni a petaflop-os teljesítményt. A bizottság szerint ez elkerülhetetlen ahhoz, hogy jobb eredményeket érhessenek el az időjárás-előrejelzéssel, a gyártási folyamatok tervezésével, a gyógyszerkutatással és más, a nemzetgazdaság számára stratégiai fontossággal bíró területekkel kapcsolatban. Ugyanezt erősíti meg az a tapasztalat is, hogy az olyan szakmai konferenciákon, mint amilyen a ***Petaflops II*** az iparvállalatok fejlesztői hosszú listákat képesek összeállítani azoknak az alkalmazásoknak a neveiből, amelyeknek meglátásuk szerint kifejezetten jót tenne, ha nagyobb számítási teljesítmény állna rendelkezésre. Ezek között akadnak töréstesztek kivitelezésre alkalmas szoftverek, repülőgépek és űrjárművek tervezésére használt szimulációs eszközök, de vannak gazdasági, járványterjedési, vagy a bioterrorizmus hatásait vizsgáló modellek is.
Mindezekre az igényekre a ***HPC*** közösség olyan fejlesztési stratégiák kidolgozásával válaszol, amelyek segítségével ***Moore*** törvényének érvényessége nem csak meghosszabbítható, hanem át is hidalhatók azok az elvi akadályok, amelyek a ma használatos rendszerek korlátaiból és felépítéséből erednek.
Az alkalmazható stratégiák a következőkben foglalhatók össze:

- Olyan többmagos rendszerek építése, amelyek egy chipen több, többé- kevésbé önálló feldolgozóegységet tartalmaznak, így biztosítva a szükséges többletteljesítményt.
- Speciális processzorok alkalmazása, amelyek kiemelkedően jó teljesítményt nyújtanak az olyan különleges területeken, ahol a közönséges feldolgozóegységek rosszul teljesítenek.
- Olyan heterogén számítógépek tervezése, amelyekben a konvencionális és a specializált processzorok képesek egymással együttműködni.

Elvileg mindhárom említett megközelítés jelentős többletteljesítményt eredményezhet, ha a megfelelő területen alkalmazzák. A ***Cray*** fejlesztői mindhárom fejlődési utat lehetségesnek

tartják, így mindhárommal foglalkoznak. Ugyanakkor ami a hosszú távú lehetőségeket illeti, a kutatók úgy gondolják, hogy a heterogén feldolgozásban óriási tartalékok rejlenek. Ezzel a módszerrel nem csak egyszerűen meghosszabbítható a *Moore* törvény érvényességi ideje, hanem olyan teljesítmény érhető el, amely messze meghaladja a törvény által jósolt mértéket. A heterogén feldolgozás tehát olyasmi, ami ledönti mindazokat a korlátokat, amelyeket a hagyományos architektúrák kezdettől fogva magukban hordoznak. A *Cray* mint a *DARPA Nagy Teljesítményű Számítási Rendszerek Programjának (DARPA High Productivity Computing Systems Program)* egyik résztvevője úgy gondolja, hogy a heterogén feldolgozásnak alapvető jelentőségű szerep jut majd a következő néhány év műszaki történelmében.

A közvetlen megoldás: többmagos rendszerek

Ha egy gyártó a legegyszerűbb és leggyorsabb módszert keresi arra, miként feleljen meg termékeivel a *Moore* törvénynek, valószínűleg a többmagos rendszerek mellett fog dönteni. Kiváló példa erre az *AMD* kétmagos *Opteron* processzora.
A *Cray* természetesen szintén alkalmazza ezt a modellt, hiszen már ma is szállít kétmagos rendszereket, sőt fel van készülve az ilyen irányú továbbfejlődésre is. Ez a stratégia egyrészt azonnali növekedést okoz a felhasználó rendelkezésére álló teljesítményben, másrészt némi lehetőséget teremt arra is, hogy a hőtermelést és a fogyasztást korlátozni lehessen.
Számos alkalmazás számára, különösképpen pedig a rengeteg lebegőpontos számítást igénylők számára a többmagos processzorok azt az elsődleges „menekülési útvonalat” jelentik, amelyen keresztül Moore törvényének érvényessége fenntartható. Ugyanakkor létezik számos olyan terület is, ahol a Moore törvény egyszerű betartása már nem is elég. Ilyenek az olyan sok bitszintű műveletet, rendezést vagy jelfeldolgozást igénylő területek mint az adatbázisok kezelése, kép-, mozgókép- vagy hangfeldolgozás, illetve a titkosítási eljárások használata. Ezeken az alkalmazási területeken a jövő kihívásaival csak úgy boldogulhatunk, ha a jelenleg elérhetőnél nagyságrendekkel nagyobb számítási teljesítmény áll rendelkezésünkre. Pontosan ezzel magyarázható, hogy a *HPC* területen dolgozó szakemberek máris alternatív megoldásokon dolgoznak.

Újszerű feldolgozóelemek

Az elmúlt években a klaszter-alapú megoldások szép lassan kiszorították a *HPC* piacról a nagy méretű, erősen specializált rendszereket. Az ok egyszerű: a klaszterek számos alkalmazási területen olcsóbban képesek stabil és komolynak mondható teljesítményt szolgáltatni. Ugyanakkor egyre több olyan felhasználó jelentkezik, aki munkája során beleütközött a skalár processzorok eredendő, belső korlátaiba, ami arra késztette a *Cray*-t, hogy a fenti tendenciát kicsit megfordítsa. Ez a részleges visszafordulás a következőket takarja:

- *Vektorszámítógépek alkalmazása:* A vektorprocesszorok működésnek alapgondolata az, hogy a nagy adatsorokon elvégzendő azonos típusú számítási műveletek párhuzamosan is elvégezhetők, s így a hagyományos processzorok teljesítményénél jóval nagyobb sebesség érhető el.

- *Többszálú processzorok:* A *HPC* egyik érdekes ellentmondása, hogy a memóriamodulok sebessége sokkal kisebb ütemben nőtt, mint a processzoroké. Ez aztán értelemszerűen szűk keresztmetszet kialakulását eredményezte, hiszen a soros feldolgozást végző processzorok idejük jelentős részét azzal töltik, hogy az adatok megérkezésére várakoznak. A többszálú processzorokat használó rendszerekben (ilyen az *IBM Simultaneous Multi-Threading* processzora, vagy az *Intel Hyper-Threading* technológiája) ezt a problémát úgy oldják meg, hogy a processzor egyszerre több kisebb műveletsort hajt végre, miközben a szálak között megosztja a memóriát és a hozzá vezető sávszélességet. A *Cray* még egy kicsivel továbblépett ezen az úton, hiszen szálak tucatjainak párhuzamos futását, és ezzel a memória-sávszélesség teljes kihasználását teszi lehetővé.

- *Digitális jelfeldolgozó egységek (Digital Signal Processing; DSP)*: Ezek olyan, specializált processzorok, amelyek rendkívül hatékonyan tudnak folytonos jeleket, például audió, videó vagy radar adatfolyamokat feldolgozni. Mivel mindemellett általában kicsi a fogyasztásuk is, kiválóan alkalmazhatók plazmatévékben, mobiltelefonokban és számos más beágyazott rendszerben.

- *Specializált társprocesszorok:* Az olyan lebegőpontos számítások gyors elvégzésére alkalmas matematikai társprocesszorok, mint amilyeneket például a *Clearspeed Technology* gyárt, vagy amilyen a *GRAPE* n-test problémák megoldásának gyorsítására szolgáló eszköze általában egyedi tervezésű mátrixprocesszorokat használnak. Ezekkel óriási mennyiségű lebegőpontos művelet végezhető egyszerre, egyetlen chipen belül, hiszen működésük lényege, hogy rengeteg szorzó és összeadó egységet tartalmaznak. Ennek megfelelően ezek az eszközök jelentős teljesítménytöbbletet tudnak felmutatni az olyan matematikailag intenzív számításokkal kapcsolatban, mint amilyen a mátrixok invertálása vagy az n-test problémák megoldása.

A specializált feldolgozóegységek egyes területeken érzékelhetően nagyobb teljesítményt szolgáltatnak, mint általános célú társaik. A vektor- és többszálú processzorok ráadásul a késleltetésekre sem különösebben érzékenyek, hiszen képesek úgy is folytatni a folyamatban levő számításokat, hogy közben nagy mennyiségű memóriahivatkozás vár kiszolgálásra. Összefoglalva tehát ez előnyöket ezek a fejlesztések úgy juttatják jelentős teljesítménytöbblethez a speciális igényekkel rendelkező felhasználót, hogy közben csökken a cache-ek közötti kommunikáció és a hagyományos gyorstárazási stratégiák megvalósításával kapcsolatos „digitális építészetet” sem kell a tervezőknek túlzásba vinniük.
Ugyanakkor – bár a specializált processzorokat régóta sikerrel alkalmazzák – ezeknek az eszközöknek is megvannak a maguk korlátai. Először is igaz ugyan, hogy a speciális számításokat

nagyon gyorsan tudják végrehajtani, a hagyományos skaláris kódot viszont lassabban, mint a közönséges processzorok. Márpedig a legtöbb a való életben használt szoftver kódjának legalább egy része ilyen „egyszerű” algoritmust tartalmaz. Ezt a problémát általában úgy próbálják áthidalni, hogy a speciális feldolgozást végző egységet nem önállóan használják, hanem egy hagyományos rendszerhez csatlakoztatják *PCI* buszon keresztül, gyakorlatilag egyfajta perifériaként. Ezzel csak az a gond, hogy így a kommunikáció sávszélessége a két világ között meglehetősen kicsi lesz, ami értelemszerűen akadályozza köztük az együttműködést, és így gátat szab az elérhető gyorsulásnak is. (Ami azt illeti, a kiszámított adatoknak a hagyományos rendszerbe való visszaolvasása gyakran több időt vesz igénybe, mint maga a számítás.)
Van ezen kívül még egy nagy gond a célprocesszorokkal, nevezetesen a gyártásukkal kapcsolatos gazdasági megfontolások. Ha egy processzortípusnak nincs egy viszonylag nagy méretű és jól kialakult piaca, amely képes eltartani az eszközzel kapcsolatos fejlesztéseket és magát a gyártást, akkor egy vállalat kétszer is meggondolja, hogy elő merjen-e hozakodni egy új fejlesztéssel. Minden speciális eszköznek kell hogy legyen egy olyan felvevőpiaca, amely gazdaságossá teheti a gyártását. A *DSP*-knél például a fogyasztási elektronika jelenti ezt az életteret.
Mindezek a problémák arra vezették a *Cray*-t és számos más gyártót is, hogy alternatív megoldásokat keressen.

A heterogén modell

A heterogén feldolgozás alapfilozófiája az, hogy a teljes rendszer többféle feldolgozóelemből áll, és munka közben mindegyik azt a feladatot végzi, ami a képességeinek a legjobban megfelel. Ez a modell egyrészt akár százszoros gyorsulást is hozhat a fent felsorolt speciális processzorok alkalmazásával, miközben kiszélesíti a hagyományos mikroprocesszoros architektúrák alkalmazási területeit. Mivel a *HPC* alkalmazások rendszerint olyan kódrészleteket is tartalmaznak, amelyek képesek kihasználni a gyorsításra tervezett céleszközöket, és olyanokat is, amelyek legjobban a közönséges, soros feldolgozást alkalmazó processzorokon futtathatók, ezért bátran kijelenthetjük, hogy nincs „legjobb” processzor. Nincs olyan eszköz, ami valamennyi számítástípusra egyaránt jó lenne. A heterogén feldolgozásnak éppen az a lényege, hogy a felhasználónak és az általa futtatott alkalmazásnak többféle processzor áll a rendelkezésére, és mindegyik részművelethez azt használhatja, amelyek a célnak a legjobban megfelel.
Hagyományosan két nagy akadálya volt a heterogén számítógépes rendszerek széles körben való elterjedésének. Az egyik az ilyen rendszerek programozásának összetettsége, ami abból fakad, hogy a felhasználónak (fejlesztőnek) magának kell gondoskodnia az egyes részfeladatok ésszerű kiosztásáról. A másik probléma a hagyományos és a célprocesszorok eltérő architektúrájából fakad, ami szintén bonyolítja az ilyen rendszerek programozását. Ezek értelemszerűen jelentős gátló tényezők lehetnek, amelyeket mindenképpen figyelembe kell venni, mielőtt az ember belevág egy ilyen vállalkozásba. A jó döntéshez látnunk kell előre, hogy menyit nyerünk az egyik oldalon és mennyit vesztünk a másikon. Ugyanakkor az is igaz, hogy a többmagos rendszerek megjelenésével a HPC területen dolgozó fejlesztők általános hozzáállása is változóban van. Itt gyakorlatilag egy technológiai ugrás következett be, így a programozók és az alkalmazások tervezői egyre inkább hajlandóak elsajátítani új architektúrák használatát, vagy legalább fontolóra venni azt.
És ebbe azok a heterogén rendszerek is belefférnek, amelyek egyre-másra kezdenek megjelenni a piacon.
A *Cray X1E* szuperszámítógép például vektor és skalár processzorokat egyaránt tartalmaz, sőt van hozzá egy olyan speciális fordítóprogram is, amely automatikusan szétosztja a terhelést azok között. De akad hasonló példa más cégek háza táján is. Az új *Cell* processzor-architektúra, amit az *IBM*, a *Sony* és a *Toshiba* közösen fejlesztettek az új *Playstation 3* rendszeren futó játékok támogatására, szintén úgy működik, hogy egy hagyományos processzor szükség estén részfeladatokat oszt ki olyan specializált feldolgozóegységeknek, amelyeknek hozzá hasonlóan közvetlen hozzáférése van a memóriához. A heterogén rendszerek manapság legizgalmasabb területe azonban az úgynevezett térprogramozható kapumátrix processzorok (*Field Programmable Gate Array; FPGA*) alkalmazása.

Az FPGA társprocesszor modell

Az *FPGA*-k olyan hardveresen átkonfigurálható célprocesszorok, amelyek belső logikáját a programozó újra és újra átírhatja az éppen megoldandó problémának megfelelően. *FPGA*-kat tulajdonképpen már több mint egy évtizede használnak programozható logikai eszközként, társprocesszorként való felhasználhatóságuk azonban csak újabban merült fel. Az *Egyesül Államokban* és néhány más országban a közelmúltban több a témával kapcsolatos konferenciát is tartottak, az *Ohio Supercomputing Center* pedig létrehozta az *OpenFPGA* kezdeményezést (➲ *www.openfpga.org*) azzal a céllal, hogy felgyorsítsa az *FPGA* eszközöknek a *HPC* területén való meghonosodását.
Ennek az általános lelkesedésnek természetesen jó oka van: az *FPGA*-k bizonyos típusú problémákkal kapcsolatban több nagyságrenddel növelhetik a feldolgozás sebességét. *FPGA*-k segítségével a tervezők olyan céleszközöket hozhatnak létre minden egyes megoldandó problémához, amelyekben elemi feldolgozóegységek százait vagy ezreit vezényelhetik arra, hogy egymással párhuzamosan működve hajtsanak végre egy műveletet. Ez különösen azokon a területeken jár óriási előnnyel, ahol sok bitszintű műveletet, összeadást, szorzást, összehasonlítást, konvolúciót vagy transzformációt kell végezni. Az *FPGA*-k megfelelően felprogramozva ilyen műveletből egyszerre rengeteget tudnak végrehajtani úgy, hogy a műveletet sem az operációs rendszernek, sem a futó folyamatnak nem kell kívülről felügyelnie. Az már csak a hab a tortán, hogy a kapcsolódó energiafogyasztás is jóval kisebb, mint a hagyományos processzorok esetében.
Az *FPGA*-k széles körben való elterjedésének – amint mondani szokták – történeti okai vannak. Először is ezeket a processzorokat általában hagyományos rendszerek részeként használták úgy, hogy azokhoz *PCI* buszon keresztül, perifériaként kapcsolódtak. Ebből ugyanazok az átviteli sávszélességgel

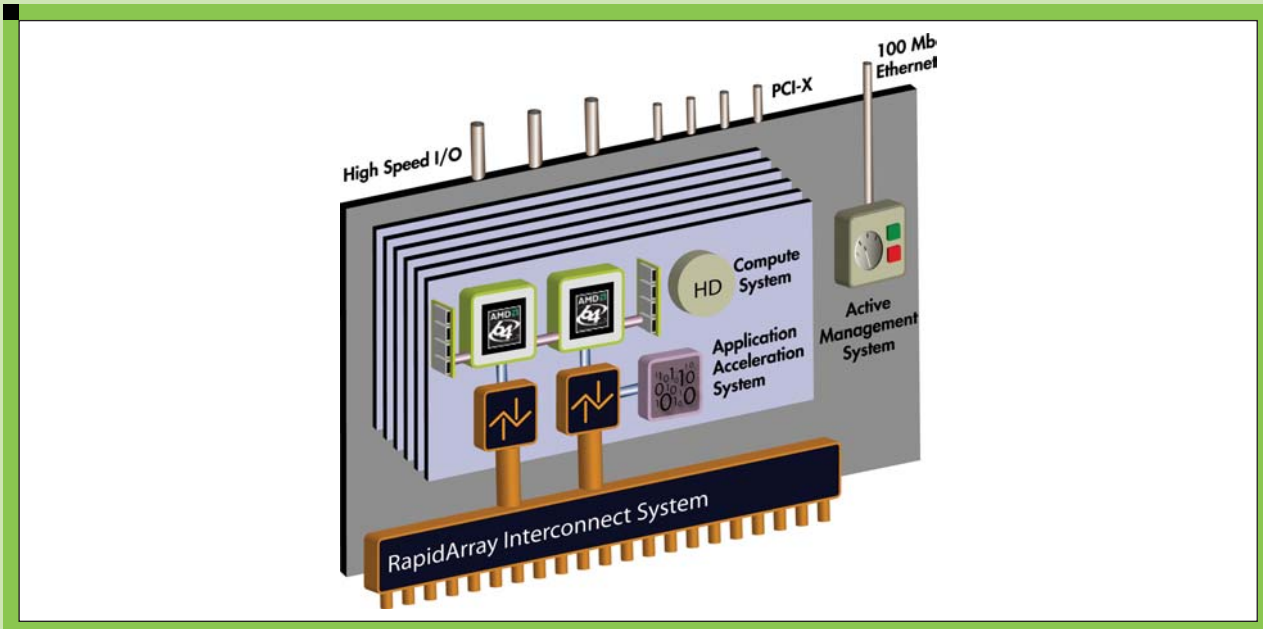
kapcsolatos problémák adódtak, mint amelyeket korábban a specializált processzorokkal kapcsolatban említettem. Az igazán nagy problémát ugyanakkor nem is ez jelentette, hanem hogy a hagyományos alkalmazásokat át kellett volna írni úgy, hogy képesek legyenek együttműködni az *FPGA*-val. Ez pedig kifejezetten nehéznek bizonyult, mivel az *FPGA*-kat *HDL (Hardware Design Language)* nyelven kellet volna programozni. És bár ezt a nyelvet az elektronikával foglalkozó villamosmérnökök naponta használták, a *HPC* rendszerek tervezői, programozó és különösen felhasználói számára tökéletesen ismeretlen volt. Igazából még manapság is épp csak fejlődőben vannak azok az eszközök, amelyekkel a „földi halandók" is képesek kihasználni az *FPGA*-k nyújtotta lehetőségeket. Ami pedig a meglevő alkalmazások átírását illeti, a felhasználók egyelőre csak várnak azokra az eszközökre, amelyekkel ezt viszonylag könnyen megtehetik. A *Cray*-nél éppen ezért számos szakember dolgozik azok, hogy elhárítsa ezeket az akadályokat a fejlődés útjából.
A *Cray XD1* szuperszámítógép például az egyik első olyan kereskedelmi forgalomban kapható *HPC* rendszer, amely *FPGA*-kat használ felhasználó

Smith-Waterman: Gyakorlati példa az FPGA társprocesszorok és a heterogén számítógépek felhasználhatóságára

A genomika legújabb módszereivel egyszerre több millió *DNS* szakaszról nyerhetünk információt, ennek a nyers adattömegnek az értelmes eredményekké való átalakítása azonban nem triviális feladat. A géneket nukleotidok sorrendjeként írhatjuk le. (Ehhez teljesen hasonló a fehérjék leírása, de ott az aminosavak sorrendjét kell megadni.)
A kutatók ezeknek a sorrendeknek a statisztikai elemzésével számos érdekes kérdésre tudnak választ adni. Összehasonlítva például két faj génkészletét érdekes hasonlóságok, átfedések fedezhetők fel, amelyek a genetikai rokonságra utalnak. Ehhez persze kell egy olyan módszer, amivel pontosan meg lehet határozni, hogy két jelsorozat miben és milyen mértékig hasonlít egymásra.
Ez a módszer egy ideje már létezik is, hiszen a megfelelő algoritmus leírását *Smith* és *Waterman* 1981-ben közölte le (*Temple F. Smith and Michael S. Waterman, "Identification of Common Molecular Subsequences", J. Mol. Biol.*, 147:195—197, 1981). Van azonban a dologgal egy apró probléma: az algoritmus olyan bonyolult, hogy hagyományos processzorokkal csak igen lassan hajtható végre. Az *FPGA* társprocesszorokat tartalmazó *Cray XD1* heterogén szuperszámítógép jelentős áttörést hozott ezen a területen, hiszen megfelelően programozva a *Smith-Waterman* algoritmust körülbelül 40-szer gyorsabban képes végigszámolni, mint elődei.

A Smith-Waterman algoritmus

A *Smith-Waterman* algoritmus *DNS* vagy fehérjeszekvenciákat hasonlít össze már létező adatbázisokkal. Mivel mind a minta, mind maga az adatbázis tartalmazhat hibákat hiányzó betűk, vagy véletlenül többletként bekerült jelek formájában, és mivel ezek az apró eltérések adott esetben óriási biológiai különbségnek felelhetnek meg, a feldolgozás során nagyon pontos megfeleltetést kell megvalósítani.
Egy génszekvencia tulajdonképpen nem egyéb, mint négy betű (a négy nukleotid: *G*, *C*, A és *T*) látszólag véletlenszerű sorozata. Egy fehérje ezzel szemben húsz elemből (aminosavak) épül fel, de matematikai szempontból ugyanilyen alaptulajdonságokkal rendelkezik. Mivel a génszekvenciák rendezett jelsorozatok, az összehasonlítás során azt kell megállapítani, hogy az adatbázis tartalmaz-e olyan betűket/jeleket, mint a minta, illetve ha igen, akkor a kettő mennyire hozható fedésbe egymással. Magyarra lefordítva például a STOP és a POTS ugyanazokat a jeleket tartalmazza, de a két jelsorozat nem hozható fedésbe egymással. Ugyanakkor a POTS és a POINTS hiába tartalmaz eltérő betűket is, a genetika játékszabályai szerint mégis fedésbe hozhatók, ha a második sorozatban egy lyukat képzelünk az O és T közé. Ezek a szekvenciák tehát potenciális rokonok. A *Smith-Waterman* algoritmus úgynevezett „dinamikus programozást" használ az optimális illesztések megtalálásához, aminek a végrehajtása rengeteg bitszintű műveletet és egyszerű de párhuzamosan végezhető számításokat tartalmaz. És sajnos éppen ez az a két terület, ahol a hagyományos processzorok rendkívül rosszul teljesítenek.
Ha a *Smith-Waterman* tesztet egy hagyományos processzoron futtatjuk, akkor minden egyes adatnak a mintával való összehasonlítása több ezer egyedi lépés végrehajtását jelenti. A tényleges összehasonlítás elvégzésére fordított elemi lépések száma csupán töredéke azok számának, amelyek ahhoz szükségesek, hogy a következő összehasonlítási pontot kiszámítsuk, illetve hogy az egész rendszer logikáját kezeljük. Ha statisztikailag közelítjük meg a dolgot, akkor azt mondhatjuk, hogy egy skalár processzor 100 elemi lépésből mindössze egyet fordít a tényleges összehasonlítás elvégzésére, vagyis – bizonyos értelemben – 1 százalékos hatásfokkal működik.
Egy *FPGA* processzorokat használó *HPC* rendszer számos ponton ennél sokkal hatékonyabban képes működni. Először is szemben a hagyományos processzorokkal, amelyeket úgy terveznek meg, hogy számos különböző típusú kód futtatható legyen rajtuk, az *FPGA* processzorok utasításkészletét teljesen az adott problémához lehet igazítani. Az *FPGA*-k felépítése emellett eleve olyan, hogy rengeteg lehetőséget adnak a párhuzamosításra. Lehetőségünk van például arra, hogy párhuzamosan kapcsoljunk nagy számú elemi összehasonlító egységet, és összehasonlítások ezreit végeztessük el az *FPGA*-val egyetlen órajel alatt.



1. ábra A Cray XD1 rendszer felépítése

Hasonlóan nagy hatékonyságot érhetünk el a bitszintű műveleteknél is, mivel az *FPGA*-k felépítésükből fakadóan ezeket is sokkal hatékonyabban képesek végrehajtani.

A Cray XD1 által alkalmazott megközelítés

Ahhoz, hogy átláthassuk, miért is olyan hatékony a *Cray XD1* a *Smith-Waterman* algoritmussal kapcsolatban, ismernünk kell a rendszer *FPGA* társprocesszorokkal kapcsolatos architektúráját (lásd az *1. ábrát*), illetve magának az alkalmazásnak a működési logikáját.

A *Smith-Waterman* algoritmus működésének alapja egy ponttáblázat (*scoring matrix*), amelynek felső sorában és oldalsó oszlopában az összehasonlítandó sorozatok vannak, a metszéspontokban levő cellák pedig az egyezések számát tartalmazzák. Utóbbiakat természetesen a programnak kell kitöltenie. Amint a mátrix elkészült, az algoritmus veszi a legnagyobb találatokat tartalmazó mezőket, visszaköveti, hogy miből keletkeztek a találatok, és ebből határozza meg a végső illesztést (*2.* és *3. ábra*).

Ennek a folyamatnak a felgyorsítása végett a *Cray XD1* felosztja az elvégzendő műveleteket az *FPGA* és az *Opteron* processzorok között. A pontozási táblázatot az *FPGA*-n futó kód tölti ki (mivel ez jól párhuzamosítható), a mátrix frissítését (szekvenciális kód) pedig az *Opteronok* végzik az *FPGA*-k által visszaküldött adatok alapján.

Mindez a gyakorlatban úgy valósul meg, hogy a szuperszámítógép *HPC*-re optimalizált *Linux* operációs rendszere kizárólag a *Smith-Waterman* alkalmazás magjának végrehajtásakor hívja meg az *FPGA* processzorokat. Ugyanakkor mivel az *FPGA*-k nagy mennyiségű ilyen speciális számítást tudnak párhuzamosan végrehajtani, a végeredmény körülbelül 24-40-szeres gyorsulás lesz. Az alábbi kódrészlet azt mutatja be, miként használhatja ki egy alkalmazás az *FPGA* képességeit:

Scoring Matrix

	0	A	C	G	T	A	T	G	C
0	0	0	0	0	0	0	0	0	0
A	0	2	0	0	0	2	0	0	0
C	0	0	4	2	1	0	1	0	2
G	0	0	2	6	4	3	2	3	1
A	0	2	1	4	5	6	4	3	2
A	0	2	1	3	3	7	5	4	3
C	0	2	4	2	2	5	6	4	6
C	0	0	2	3	1	4	4	5	6
C	0	0	2	1	2	3	3	3	7
T	0	0	0	1	3	2	5	3	5
T	0	0	0	0	3	2	4	4	4
G	0	0	0	2	1	2	2	6	4
C	0	0	2	0	1	0	1	4	8

2. ábra A pontozási táblázat

Final Alignment

A	C	G	A	A	C	C	C	T	T	G	C
|	|	|		|					|	|	|
A	C	G	T	A	-	-	-	-	T	G	C

3. ábra A végső illesztés meghatározása a Smith-Waterman formula alapján

```
/* Tömbök elforgatása az FPGA segítségével */
static void tilt (int fp_id, u_64
➥*trans_matrix, int row_len)
{
  int i = 0;
  u_64 status = 0;
```

```
  /* Inicializáljuk az FPGA-t, és felkészítjük
  ➥a tömbök egy új sorozatának fogadására. */
  fpga_wrt_appif_val (fp_id, TILT_START,
  ➥TILT_APP_CFG, TYPE_VAL, &e);
  /* Átmásoljuk a mátrixot az FPGA
  ➥memóriaterületére. */
  memcpy((char *) fpga_ptr, (char *)
  ➥trans_matrix,
row_len*sizeof(u_64));
  /* Folyamatosan figyeljük, hogy az FPGA
  ➥elkészült-e a végrehajtással. */
  while (1) {
    fpga_rd_appif_val (fp_id, &status,
    ➥TILT_APP_STAT, &e);
    if (status & TILT_DONE) break;
  }
  /* Amikor az FPGA elkészült, valamennyi
  ➥transzponált adatot kiírja */
  /* a DRAM azon területére, ami az adatcserére
  ➥szolgál. */
  /* Adatok visszamásolása az átviteli
  ➥területről az eredti tömbbe. */
  //  for(i=0;i<row_len;i++) {
  //  trans_matrix[i] = dram_ptr[i];
  //}
  return;
}
```

A Cray XD1 heterogén architektúra előnyei

Azzal a teljesítménytöbblettel, amit a ***Cray XD1*** heterogén architektúrája nyújt, a felhasználók lehetőséget kapnak arra, hogy problémáik megoldására a legjobb algoritmust alkalmazzák ahelyett, hogy egy kevésbé pontos, de gyorsabb módszer mellett kellene dönteniük. Mivel pedig a rendszer kizárólag a ***Smith-Waterman*** módszer magjának végrehajtásához használja az ***FPGA*** társprocesszor szolgáltatásait, ezt a kódrészletet is könnyű frissíteni, ha az alkalmazás későbbi fejlesztése azt megkívánja. Szintén fontos megjegyezni, hogy szemben a különféle dedikált hardveres megoldásokkal a ***Cray XD1*** egy valódi, általános célú ***HPC*** megoldás, ami szemben az előbb említett teljesen specializált rendszerekkel többféle kódot is képes futtatni. Ennek megfelelően más, szintén bioinformatikai célokra ugyanolyan könnyen és hatékonyan használható, mint a ***Smith-Waterman*** algoritmus megvalósítására. Összefoglalva tehát a ***Cray XD1*** hatékony, megfizethető és értékálló beruházást jelent minden olyan az élettudományok területén dolgozó szervezet számára, amelyek kimagasló számítási teljesítményre van szüksége tevékenységének végzéséhez.

által programozható gyorsítóegységekként. Ennek a gépnek a tervezésekor a fejlesztők már számos, fent említett akadályt elhárítottak, amit elsősorban azzal értek el, hogy az ***FPGA***-t kezelő alrendszert szorosan összeépítették a gépen futó, ***HPC*** műveltekre optimalizált ***Linux*** operációs rendszer kódjával. Ráadásul léteznek már olyan új eszközök is, amelyekkel a felhasználó gyakorlatilag egy a C-hez hasonló magas szintű nyelven keresztül programozhatja az ***FPGA*** logikai elemeit. Ez az eszközkészlet tartalmazza például a ***Celoxica DK Design Suite*** nevű csomagot, amely egy a ***Cray XD1*** rendszerébe integrált ***C*** kódot FPGA kóddá alakító fordítóprogram. Hasonló segédeszköz az Impulse C, a Mitrition C, illetve a ***Matlab*** fejlesztői által írt ***Simulink-to-FPGA*** is, amelyek valamennyien támogatják a modell-alapú tervezési megközelítést. Összességében úgy gondoljuk, hogy amint az ***FPGA***-val kiegészített heterogén rendszerek szélesebb körben elterjednek, a ***HPC*** rendszerek felhasználói egyrészt gyorsabban oldhatnak meg olyan problémákat, amelyek megoldhatóvá válását a ***Moore*** törvény alapján is ki lehetett következtetni, másrészt viszont olyan problémák is megoldhatóvá válhatnak, amelyek a szükséges számítási kapacitás miatt eddig reménytelennek tűntek. (Ebbe az utóbbi csoportba tartozik például a keretes részben bemutatott ***Smith-Waterman*** nevű bioinformatikai eljárás, amely ***FPGA*** társprocesszoros rendszerekkel immár kezelhető.)

Előretekintés

Bár a heterogén feldolgozás terén már ma is számos érdekes kezdeményezés figyelhető meg, a dolog még egyáltalán nem tart ott, hogy dominanciát érhetne el a ***HPC*** valamennyi területén. A jelenleg is létező problémák – melyek közül a legfontosabb még mindig a nehézkes fejlesztés, illetve a meglevő alkalmazások átültetése – egyelőre túl nagyok ahhoz, hogy a módszer általánosan elterjedhessen. Ugyanakkor a ***Cray*** és a ***HPC*** piac egyéb szereplői keményen dolgoznak az út járhatóvá tételén.
Akárcsak minden új technológiánál, a heterogén feldolgozásnál is azon múlik majd a további haladás menet, hogy milyen eredményeket lehet általa elérni, és ehhez milyen anyagi és szellemi ráfordítások szükségesek. Összességében azonban úgy gondoljuk, hogy hosszú távon a módszer által nyújtott teljesítménytöbblet mindenképpen túl nagy lesz ahhoz, hogy egyszerűen figyelmen kívül lehessen hagyni.

Linux Journal 2006. 142. szám

Amar Shan

Vezető termékmenedzser a Cray Inc. nevű szuperszámítógépeket gyártó vállalatnál. Shan 2004-ben csatlakozott a céghez, amikor az felvásárolta az OctigaBay Systems Corporation nevű vállalatot. Jelenleg a Cray következő generációs termékeinek fejlesztési irányelveit dolgozza ki, illetve ő felel a Cray XD1 nagy teljesítményű szuperszámítógép fejlesztéséért. Jelenleg ez a világ egyetlen olyan Linux/Opteron rendszere, amit kifejezetten HPC célokra terveztek. Shan a University of Waterloo-n szerzett diplomát a mesterséges intelligencia gyakorlati alkalmazásából, a University of British Columbián pedig villamosmérnöki tudományból kapott BSc fokozatot.

Windows asztal a láthatáron – RDESKTOP

Az Rdesktop egy nyílt forrású terminál kliens Microsoft Windows rendszerekhez...

Minden rendszergazdának ismerős helyzet, mikor a felhasználó ellenmondást nem tűrő hangon hívja fel telefonon, hogy már megint nem tud csatlakozni a *Windows* szerverhez, és jó lenne, ha mihamarabb elhárítanánk a hibát, mert neki most péntek délután hihetetlenül fontos munkája van, és már hívja is a főnöködet, hogy miattad nem végez a munkájával időben. Ekkor fejvesztve rohanhatsz egy szinttel fentebb lévő szerver szobába, és odaérve rémülten figyeled, ahogy az egyik kolléga épp megállított adatbázis kezelő mellett készít biztonsági másolatot... Persze vannak rendszergazdák akik komótosan, lassú, kimért mozdulatokkal, gépeli a shell prompt után: ***rdesktop***...

Az ***rdesktop GNU Public Licence (GPL)*** alatt terjesztett terminál kliens, mely képes a ***Windows NT/2000/XP/2003*** operációs rendszereken futó terminál szerverhez csatlakozni. Natívan támogatja a ***Remote Desktop Protocol-t (RDP)***, mely segítségével képes egy távoli ***Windows*** szerverről egy létező felhasználó munkaasztalát ***X*** grafikus felületen megjeleníteni. Az ***RDP Microsoft*** találmány, a terminál szerverek és a terminál kliensek kommunikációjára találták ki, mely titkosítottan (is) képes az adatokat eljuttatni a terminál szerverről a kliens géphez és vissza. Régebbi verziói támogatták többek közt az ***ISDN, IPX, NetBIOS*** protokollokat is, de a legújabb ***RDP5*** szabvány már csak ***TCP***-n keresztül hajlandó kommunikálni. Az ***RDP5*** (mely a ***Windows 2000 Server*** illetve ***Windows XP*** változatokban jelent meg) az elődeihez képest nem egy, hanem akár 64000 egymástól elkülönített adatátviteli csatornát képes kezelni (egy élő terminál kapcsolat általában egy adatátviteli csatornát használ fel). Lényegében az ***rdesktop*** programot használva kicsit olyan érzésünk van, mintha a távoli számítógép elé ülnénk, hiszen a jól ismert ***Windows*** bejelentkező képernyő vár minket a csatlakozás kezdetén.

1. ábra Ha mindent jól csináltunk, akkor valami hasonló vár minket

A kliens gép egere és billentyűzete segítségével használatba vehetjük ***Windows*** asztalunkat, sőt kliens gépünk soros illetve párhuzamos portját átirányíthatjuk a szerver felé, mintha ezek az eszközök a szerver részei lennének. Lehetőség van a szerver hang kimenetét saját kliensünkre irányítani, így akár a szerveren futó alkalmazások „hangját” is hallhatjuk, sőt saját nyomtatónkra is nyomtathatunk a szerverről.

A kezdet

Terminál kliens természetesen nem csak ***Linux/Unix*** alá érhető el, létezik ***tsclient*** nevű alkalmazás ***Windows*** rendszerekre (is). Mi azonban maradjunk csak a ***Linux*** mellett, és futtassuk az ***rdesktop*** programot... már ha van kedvenc disztribúciónkban, és fel is van telepítve. ***UHU-Linux*** alatt elég egy

```
apt-get install rdesktop
```

parancs kiadása. Más disztribúcióknál, melyek nem tartalmazzák (vagy csak egyszerűen frissebbet akarunk használni a meglévőtől) látogassunk el a ➲ www.rdesktop.org/#download oldalra, és töltsük le az aktuális stabil verzió forráskódját (a cikk írásakor a 1.4.1 verzió elérhető). Csomagoljuk ki a

```
tar -xzvf rdesktop-1.4.1.tar.gz
```

paranccsal, majd a létrejött ***rdesktop-1.4.1*** könyvtárba belépve adjuk ki a

```
./configure && make && make
➥install
```

parancsokat. (Megjegyzem, hogy a `configure` paraméterezhető úgy, hogy többek között az *Ipv6* támogatás is belefordúljon a kódba).
Ha minden rendben lefutott, és nem állítottuk be a `configure -prefix` paramétert, akkor a */usr/local/bin* könyvtárban találjuk a futtatható binárist.

Amire még szükség lesz

Az *rdesktop* kipróbálásához már csak egy terminál szervert futtató *Windows*-ra lesz szükségünk. Saját hálózaton kívüli Terminál szerver eléréséhez közvetlen *Internet* kapcsolatra vagy *NAT*-olt belső hálózatra van szükség. Természetesen otthoni *Windows 2000*, illetve *Windows XP* operációs rendszert futtató számítógépünket is elérhetjük ezzel a módszerrel.
A szervernek szánt gép tűzfalán a 3389 *TCP* porton engedélyezzük a bejövő *TCP* kapcsolatot. *Windows XP*-n alapértelmezett szolgáltatásként fut a *Terminál szerver*. Ezek után ki kell választanunk, hogy mely felhasználók használhassák ezt a szolgáltatást (alapértelmezettként csak a rendszergazda hozzáférése engedélyezett).

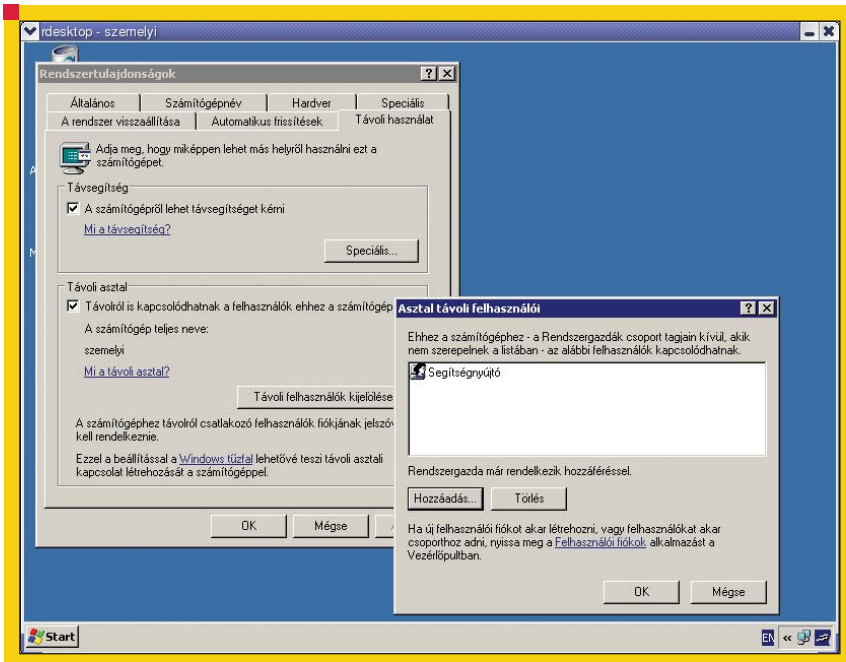
Állítsuk be

Az *rdesktop* rendelkezik néhány hasznos parancssori kapcsolóval, amivel beállíthatjuk kapcsolatunkat.

- `-u <felhasználói név>`: A kapcsolódó felhasználó nevét adhatjuk meg vele. Ha nem adunk meg semmit, akkor a rendszerünk aktuális felhasználójának nevét veszi át.

- `-d <domain név>`: A csatlakozáshoz szükség lehet *windows*-os *domain* név megadására is.

- `-s <parancs>`: a kapcsoló után megadott parancsot végrehajtja bejelentkezés után (ha `-s cmd` kapcsolóval indítjuk az *rdesktop*-ot, akkor sikeres bejelentkezés után egy *command* ablak vár minket). E kapcsoló használatakor csak a megadott programot futtatja a terminál szerver, és a program bezárásakor a *rdesktop* kapcsolat is bezáródik



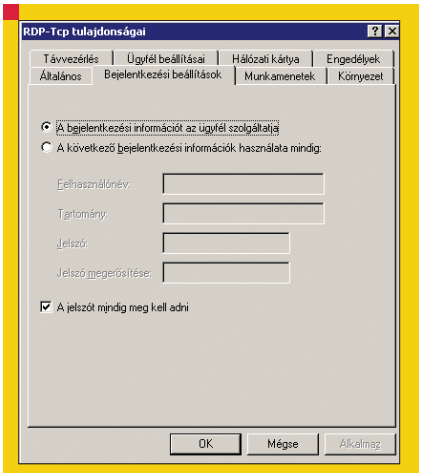
2. ábra Távoli asztalkapcsolat engedélyezése

(Halkan megjegyzem, hogy ez a funkció vékony klienseknél kitűnően használható).

- `-c <könyvtár>`: A felhasználó kezdő könyvtárát állíthatjuk be vele. Önmagában nem sok mindenre jó, de a -s kapcsolóval kombinálva már használható (`-s cmd -c c:\\WINNT` kapcsolók hatására a *command* ablak a WINNT könyvtárban nyílik). A kapcsolók és paraméterei közt nem kötelező szóközt hagyni, bár így olvashatóbb, viszont azt vegyük figyelembe, hogy *Windows*-szerűen kell az elérési utakat megadnunk, így a mappák neveinek elválasztásához „\" jelet használunk, ráadásul ezt is duplán, mivel a *Linux* alatti *shell*-ek vezérlőkaraktert keresnek utána (C szintaxisából adódóan).

- `-p <jelszó>`: A felhasználói névhez tartozó jelszó. Használhatósága függ attól, hogy a *Windows* szerveren futó Terminál szerver „Jelszót mindig meg kell adni" opció engedélyezve van-e (*3. ábra*).

- `-n <kliens név>`: Itt adhatjuk meg, hogy *rdesktop* kliensünk milyen névvel csatlakozzon a terminál szerverhez. Ha nem adjuk meg, akkor a rendszerünk *host* nevét küldi el kliens névként.



3. ábra Jelszó kikényszerítése bejelentkezéskor

- `-k <billentyű kiosztás>`: Billentyűzet kiosztás meghatározása. Az *rdesktop* számára az *en-us* az alapértelmezett. Természetesen csak olyan kiosztásokat használhatunk, melyek telepítve vannak (magyar billentyűzet kiosztást a `-k hu` opcióval érhetjük el).

- `-g <méret %>`: Ez egy igen érdekes opció. Segítségével módosíthatjuk a *Windows* asztalunk méretét. Megadhatjuk az új méretet képpontban (szélesség x hosszúság, 640x480), illetve az eredeti mérethez képest százalékos arányban

(például `-g 80%`). Nem kötelező szabvány méreteket megadnunk (akár a `-g 1023x444` méretet is megadhatjuk). A méretezés kizárólag az asztal méretére vonatkozik, nincs hatással a betű méretre, ikonokra, és minden egyéb grafikus objektumra.

- `-f:` Teljes képernyős mód engedélyezése. Igen hasznos kapcsoló, teljesen lefedi a ***X*** szervert, minden billentyű parancs a Terminál szerveren érvényesül (kivéve persze a jó öreg ***CTRL+ALT+BACKSPACE***).

- `-e:` Titkosított adatátvitel tiltása. Bizonyos ***Windows NT*** verziók esetén lehet szükség rá.

- `-E:` Titkosított adatátvitel tiltása titkosított bejelentkezéssel. A bejelentkezés során megadott jelszó kivételével az összes ***TCP*** csomag titkosítatlanul kerül továbbításra.

- `-m:` Nem küldi el az egér által kiváltott mozgással kapcsolatos eseményeit. Alacsony sávszélesség esetén van jelentősége, mivel csökken az átvitt adatok mennyisége.

- `-C:` Saját színpaletta használata a megjelenítés során, mely szebb megjelenítést tesz lehetővé. Hátulütője, hogy ha az ***rdesktop*** nincs fókuszban, akkor a megjelenített színek nem hasonlítanak az eredetiekhez.

- `-D:` Az ***rdesktop*** körüli ***Window*** dekorációt nem jeleníti meg (***rdesktop***-ot „behasználó” ***GUI*** fejlesztőknek hasznos).

- `-K:` Nem takarja el az ablakkezelő billentyű kombinációit, így a gyorsbillentyűket a kliens oldal értelmezi.

- `-S <gomb méret>:` Engedélyezi a „***single application***” (egy program futtatása) módot. A `-s` kapcsoló után megadott programot futtatja, ablakméretét maximalizálva. Ha „kis méret” gombra kattintunk, akkor az egész ***rdesktop*** ablak minimalizálódik a tálcára.

- `-T <cím>:` Az ablak címét adhatjuk meg.

- `-N:` Engedélyezi a ***numlock*** szinkronizációját az ***X*** szerver és a távoli ***RDP*** folyamat között. Néhány ***X*** szerver esetén előfordulhat, hogy a távoli asztalon futó programok nem képesek a ***numlock*** állapotát lekérdezni, ilyenkor nyújt segítséget ez a kapcsoló.

- `-X <ablak azonosító>:` Beágyazott ***rdesktop*** ablak esetén határozza meg az ***rdesktop*** ablak azonosítóját. Decimális és hexadecimális alakban is megadható.

- `-a <bpp>:` A kapcsolat színmélységét állítja be a megadott értékre. 8, 15, 16, 24 ***bit/pixel*** értékeket használhatunk. A használható színmélység függ a szerver konfigurációjától is. 8 ***bpp***-től nagyobb színmélységet a ***Windows XP*** illetve újabb verziók támogatnak. Nagyobb színmélység szebb képet, és nagyobb hálózati terhelést eredményez.

- `-z:` Tömörítés engedélyezése. Ez a lehetőség csak a 8 bites színmélységnél működik.

- `-x <minőség>:` Kapcsolat minősége. A távoli asztal megjelenítéséhez használt grafikus témákat, effekteket szabályozza. Lehetséges értékei ***m[odem] , l[an] illetve b[roadband]***. Alapértelmezetten a témahasználat engedélyezett, minden egyéb tiltott. ***Modem*** (telefonos kapcsolat) minőség esetén minden „extra” grafikus effektus tiltott, még a háttérkép megjelenítése is. ***L[an]*** (helyi hálózat) értéket beállítva engedélyezett a háttérkép, míg ***b[roadband]*** (szélessáv) esetén a menü animációk, illetve teljes „***window dragging***” is engedélyezett.

- ***-P:*** Folyamatos ***bitmap*** gyorstárazást engedélyezi a terminál kliens lemezén. Esetek többségében javítja a alacsony sávszélességű kapcsolatok minőségét, illetve jelentősen csökkenti a hálózati forgalmat, cserébe megnő az ***rdesktop*** kapcsolódásához szükséges idő (gyorstár feltöltése miatt), illetve a kliens háttértárán színmélységtől függően 10-30 Mb helyet foglal el.

- `-0:` Kapcsolódás a ***Console***-nak nevezett munkaasztalhoz. Ez a „főképernyő” ez jelenik meg a terminál szerver monitorán is. Csak ***Windows 2003*** illetve újabb rendszerek esetén működik.

- `-4:` ***RDP*** 4-es verzióját használja kapcsolat létesítésére.

- `-5:` ***RDP*** 5-ös verzióját használja kapcsolat létesítésére (alapértelmezett).

Itt a papír hol a lemez?

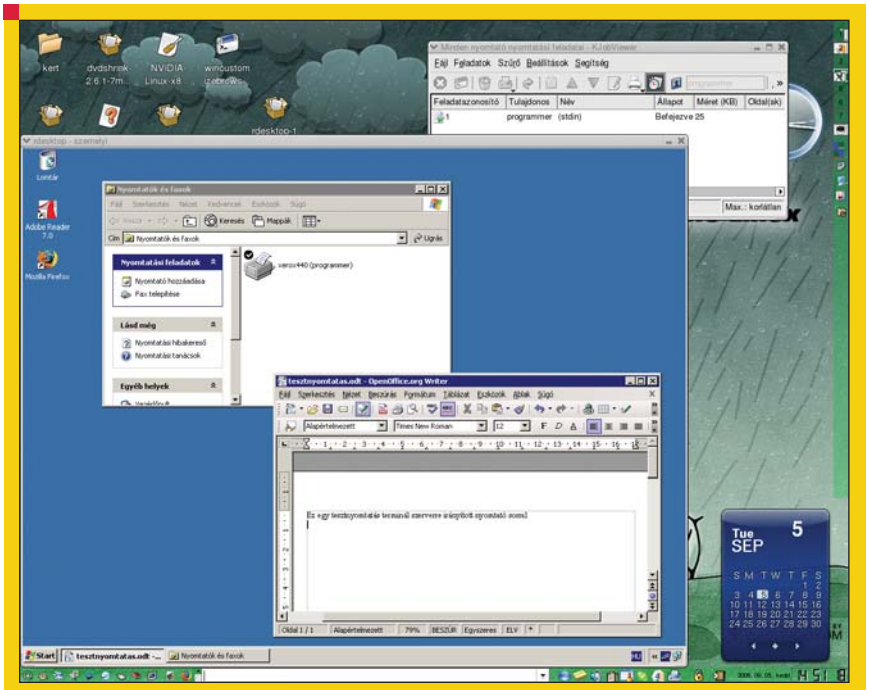
- `-r <eszköz>:` Engedélyezi a lentebb specifikált eszközök átirányítását a szerverre (működése függ a szerver beállításaitól is).

Az alábbi átirányításokat használhatjuk:

- `-r comport:<comport>=<eszköz>:` Kliensünk soros portját irányíthatjuk át a szervernek. Az új port nem jelenik meg az eszközkezelőben, de az alkalmazások tudják azt használni (például a ***Hyperterminal***). Megbízhatóan csak ***Windows XP*** illetve újabb rendszereknél működik, sajnos ***Windows 2000*** alatt nem minden program látja az új soros eszközt.

- `-r lptport:<lptport>=<eszköz>:` Párhuzamos port átirányítása. Működése megegyezik a soros port átirányításnál említettekkel.

- `-r printer:<nyomtató neve>[=<meghajtó program>]:` A kliens nyomtatósorát irányítja át a terminál szerverre. A ***<nyomtató neve>*** paraméternek meg kell egyeznie a helyi nyomtatósor nevével. A kiszolgálón megjelenik egy új nyomtató a megadott névvel. Legtöbb esetben nincs is szükség a meghajtó program



4. ábra Így néz ki egy átirányított nyomtatás

megadására, mivel az alapértelmezett *PostScript* meghajtót használja.

- `-r disk:<megosztás név>=<elérési út>`: Igen hasznos kapcsoló, segítségével kliensünk bármely mappáját csatolhatjuk a terminál kiszolgálónkhoz, hasonlóképp egy hálózati meghajtóhoz. A megosztás neve maximum 8 karakter hosszú lehet, ez a név jelenik meg *Windows* alatt. Az elérési út *Unix* szintaktikával leírt elérési út (például */var/tmp*).
  A terminálkiszolgálón a *rdesktop*-ot futtató felhasználó jogai érvényesülnek. Sajnos ez a funkció is csak *Windows XP* vagy újabb rendszereknél használható (*Windows 2000 Server* sem támogatja).

- `-r sound:[local|off|remote]`: Szerver hangkimenetét irányítja át a kliens gépre. A `remote` paraméter csak a `-0` kapcsolóval együtt működik (ebből következik, hogy csak *Windows 2003 Server* vagy újabb rendszereknél működik), és csak néhány rendszer hangot engedélyez.
  Az `off` jelentését gondolom nem kell taglalni.

Lássuk, hogyan használjuk!

Nézzünk néhány példát arra, hogyan is használhatjuk okosan ezzel a néhány kapcsolóval *rdesktop*-ot.
A szimpla kapcsolathoz körülbelül ennyire van szükségünk:

```
rdesktop szervernév
```

Ebben az esetben az aktuális felhasználónevünkkel próbál kapcsolódni. Állítsunk be egy felhasználói nevet, hozzátartozó jelszóval:

```
rdesktop szervernév -u
➥felhasználói_név -p jelszó
```

Elég sokszor előfordul, hogy hosszú listákat, táblázatokat akarunk használni, és jól jönne, ha minél több sor látszana *rdesktop* ablakunkban. Ekkor érdemes átméretezni munkaasztalunkat, figyelembe véve a kijelzőnk fizikai felbontását, és szép színes 16 bit/pixel képet szeretnénk látni, ahogy a *HyperTerminal* alkalmazást elindítjuk:

```
rdesktop szervernév -u
➥felhasználói_név -p jelszó
➥-g800x1000 -a16 -s
➥"c:\\Program Files\\Windows
➥NT\\hypertrm.exe"
```

Ha nem akar programunk elindulni, és a *Windows*-unk sem szólt, hogy az nem található, akkor ellenőrizzük, hogy a felhasználónk nincs-e helyileg bejelentkezve. Azt vegyük figyelembe, hogy a `-s <parancs>` futtatásakor a *command shell* környezeti változói érvényesülnek.
Már csak egy soros eszközt kell csatlakoztatnunk, és már irányíthatjuk át szerverünk felé:

```
rdesktop szervernév -u
➥felhasználói_név -p jelszó
➥-g800x1000 -a16 -s
➥"c:\\Program Files\\Windows
➥NT\\hypertrm.exe" -r
➥comport:com4=/dev/ttyS0
```

A *ttyS0* eszköz *COM4* néven jelenik meg az alkalmazások számára (Soros portunk beállítását saját magunknak kell elvégeznünk még az *rdesktop* indítása előtt).
Végül szeretnénk nyomtatni a szerverről, ráadásul a linuxos gépünkre csatolt nyomtatóra.

```
rdesktop szervernév -u
➥felhasználói_név -p jelszó
➥-g800x1000 -a16 -r
➥printer:<nyomtatónk neve>
```

Összességében egy nagyon jó minőségű, könnyen használható eszköz, az *rdesktop*. Akinek idegen a parancssoros használata, annak ajánlom figyelmébe a *tsclient* nevű grafikus előtét programot, mely funkcionalitásában majdnem teljesen megegyezik a windowsos „Távoli asztal elérés” alkalmazással. Hasonló megoldás által érhető el a sokak által használt is ismert *Neptun* egyetemi hallgatói rendszer is (például a ➲ *neptunc.unideb.hu*).

Gráma Tibor
(tibor.grama@hoya.lmh.hu)

1997 óta „Linuxozik”, UHU hívő. Szabadidejében gyermekeivel és vizsla kutyáival játszik, ha éppen nem kertészkedik vagy horgászik.

KAPCSOLÓDÓ CÍMEK

- ➲ http://www.rdesktop.org
- ➲ http://support.microsoft.com/kb/186607

AIDE – Ki járt a gépemben?

Ha valaki átrendezné a szobánkat, biztosan észrevennénk. De ha a számítógépünket, vajon azt is? Az AIDE egy olyan alkalmazás, amellyel nyomon követhetjük, mely fájlok, könyvtárak változtak meg a gépünkön.

Sokkal könnyebb egy szabálytalanságot észrevenni egy profi boksz meccsen, mint azt, ha a támadó egy hátsó ajtót telepített a gépünkre. Az ***AIDE*** hasonló alkalmazás, mint a ***Linuxvilág 2006. augusztusi*** számában ismertetett ***Tripwire***, de azzal ellentétben teljesen ingyenes, a ***GNU GPL*** licence alatt bocsátották ki.

Az ***AIDE*** egy fájl integritás ellenőrző alkalmazás, amely egy adatbázisban tárolja a megadott fájlok, könyvtárak különféle paramétereit, pl. tulajdonos, csoport, méret, időbélyegek és más i-node információk. Ezek mellett még kriptográfiai összegeket (például ***MD5, SHA1, RMD160***) is tárol az egyes bejegyzésekhez. Az adatbázis elkészítése után, amikor ellenőrizzük a gépünket, akkor a megadott fájlok paramétereit összehasonlítja az adatbázisban tároltakkal, és ha eltérést talál, akkor a fájl vagy könyvtár nyilvánvalóan megváltozott, így azt kijelzi nekünk.

Ha egy rosszindulatú felhasználó (***hacker***/***cracker***) betör a gépünkre, akkor minden bizonnyal telepít egy ***root kit***et (jogosulatlan rendszergazda hozzáférést biztosító eszközök), lecseréli a ***ps, ls, netstat, who***, stb. programokat, amelyek elárulnák a jelenlétét. Sajnos nem lehetünk akkor sem nyugodtak, ha ezen programok mérete és időbélyegük ismerősek, ezeket ugyanis könnyű manipulálni. Ezekkel szemben sokkal nehezebb hamisítani pl. az ***MD5 hash*** értéket. Ma már találtak ütközést (***collision***) nem csak ebben, de az ***SHA1***-ben is. Szerencsére azt azonban rendkívül nehéz megoldani, hogy például az `ls` programnak mind az ***MD5***, mind az ***SHA1***, de még az ***RMD160*** összege is változatlan maradjon, és még úgy is működjön, hogy az előbb említett fájl méret és időbélyeg paraméterek is megegyezzenek az eredetiekkel. Ezért az ***AIDE*** használatával igen nagy bizonyossággal meg tudjuk mondani egy fájlról, hogy az megváltozott-e vagy sem.

Ha paranoiásabbak vagyunk az átlagnál, akkor érdemes, magát az ***AIDE*** programot is ellenőrizni. Ebben az esetben másoljuk azt át egy másik (és megbízható) gépre, és ott nézzük meg az ***MD5, SHA1***, stb. ellenőrző összegeit.

Nagyon fontos, hogy az adatbázis megbízható legyen. Miután elkészítettük, tehetjük például írásvédetté tett hajlékonylemezre (***gzip***-pel tömörítve éppen ráfér az én ***aide.db*** fájlom), kiírhatjuk ***CD***-re, vagy közvetlenül a futás előtt letölthetjük egy biztonságos helyről a gépünkre.

Az ***AIDE*** fordításához szükséges az ***mhash*** csomag, telepítsük, ha nincs meg a gépünkön. Töltsük le az ***AIDE***-t a ➲ http://sourceforge.net/projects/aide/ címről. Csomagoljuk ki, fordítsuk le, és telepítsük a szokásos módon:

```
tar zxvf aide-0.11.tar.gz
cd aide-0.11
./configure --with-zlib
make
su -c 'make install'
```

Használat

Készítsünk el egy példa konfigurációs fájlt `1.conf` néven, hogy kipróbálhassuk az ***AIDE*** képességeit!

Ha elkészültünk, akkor adjuk ki az

```
aide --config-check -c 1.conf
```

parancsot, amely leellenőrzi a konfigurációs fájlt, és kiírja, ha hibát talált. Következő lépésben hozzuk létre az ***AIDE*** adatbázisát, amihez adjuk ki az

```
aide -i -c 1.conf
```

parancsot. Ha minden rendben ment, akkor az alábbi üzenetet láthatjuk:

```
AIDE, version 0.11
```

1. Lista Egy példa konfigurációs fájl (1.conf)

```
@@define TOPDIR /

database=file:aide.db
database_out=file:aide.db.new

verbose=5

config_version=0.0.1
report_url=stdout

All=R+a+sha1+rmd160
Norm=s+n+b+md5+sha1+rmd160
MyRule1=R+sha1

# ezt a könyvtárat
# ellenőrizzük
/home/sj/temp/aide-0.11
➥MyRule1
```

2. Lista Az AIDE detektált egy ismeretlen fájlt

```
AIDE found differences
➥between database and
➥filesystem!!
Config version used: 0.0.1
Start timestamp: 2006-08-29
➥15:22:07

Summary:
  Total number of files:  162
  Added files:            1
  Removed files:          0
  Changed files:          1

---------------------------
Added files:
---------------------------

added:/home/sj/temp/
➥aide-0.11/teszt

---------------------------
Changed files:
---------------------------

changed:/home/sj/temp/
➥aide-0.11

---------------------------
Detailed information about
➥changes:
---------------------------

Directory: /home/sj/temp/
➥aide-0.11
Mtime: 2006-08-29 15:03:02,
➥2006-08-29 15:22:05
Ctime: 2006-08-29 15:03:02,
➥2006-08-29 15:22:05
```

```
### AIDE database at
/root/aide.db.new initialized.
```

Tegyük fel, hogy másnap reggel szeretnénk látni, hogy vajon nem módosított valaki a gépünkön valamit. Ehhez futtassuk le az ellenőrzést:

```
aide -c 1.conf
```

Egy hibátlan eredmény így néz ki:

```
AIDE, version 0.11
```

3. Lista mindent megtudunk a változásokról

```
Detailed information about
➥changes:
---------------------------

File: /home/sj/temp/
➥aide-0.11/config.h
Size: 7270, 7296
Mtime: 2006-04-20 09:22:54,
➥2006-08-29 15:28:21
Ctime: 2006-04-20 09:22:54,
➥2006-08-29 15:28:21
MD5: EouVGM3qvwXFKmSI8wXmhw==,
f39/zzXsR2UycUd2eoxzbg==
SHA1: yU6M63ipmXZc+56ZrZy+
➥SH7fG7I=, mHIFiEzYVfAi
➥69l27rRh6IuzPXU=
```

```
### All files match AIDE
database. Looks okay!
```

Nézzük meg, hogy mi történik, ha egy idegen állomány kerül a megfigyelt könyvtárba! (Ehhez létrehoztam a */home/sj/temp/aide-0.11/teszt* fájlt). Futtassuk újra az előző parancsot, amely most az alábbi eredményt adja: ***2. lista***.
Az eredmény egy összesítéssel kezdődik (hány fájlt nézett meg, hány változott meg, stb.), majd kiírja az idegen fájl nevét, ill. hogy hol történt a változás (ebben az esetben egy új állomány létrejötte). Ha törlünk egy fájlt, azt is hasonló módon adja tudtunkra az *AIDE*.
Nézzük meg, mi történik, ha módosítunk egy fájlt! A programot ismét futtatva, a kimenet releváns része így néz ki: ***3. Lista***.
Láthatjuk, hogy megváltozott a *config.h* mérete, a fájl *i-node*-ban található időbélyegek és az ellenőrző összegek. Bal oldalon az eredeti paramétereket, míg jobb oldalon a futtatáskor számított értékeket találjuk.
Az *aide.db* egy szöveges állomány, ahol az ellenőrzött fájlok, könyvtárak paraméterei találhatóak, soronként egy bejegyzéssel. Minden bejegyzéshez annyi oszlop (paraméter) tartozik, ahányat a konfigurációs fájlban megadtunk, azaz az állományok neve,

4. Lista Az aide.db fájl szerkezete

```
@@db_spec name lname attr
➥perm uid gid size mtime
➥ctime inode lcount md5 sha1

/home/sj/temp/aide-0.11/
➥INSTALL 0 15293 100644 5001
➥100 7975 MTA5OTc1NDc1Mg==
➥MTE0NTUxNzUzMg== 1147475 1
➥6w12fpYI6temRutxxzbGYA==
➥K7kxSGFRgktBZLl8fCfbVogmpoY=
```

különféle *i-node* adatok (például tulajdonos, csoport, időbélyegek) és kriptográfiai ellenőrző összegek, úgynevezett *hash* értékek (***4. Lista***).
Egy igazi konfigurációs fájl persze bonyolultabb lehet, mint az ***1. Listában*** szereplő minta. Alább egy összetettebb konfigurációs fájl látható (***5. Lista***).
A konfigurációs fájlban szereplő jelek (például R, L, >) alapértelmezett szabályok, amelyek részletes listája és jelentésük megtalálható az *aide*-vel érkező minta *aide.conf* állományban.

Tippek

Az *AIDE*-t a legjobb, ha már rögtön a rendszer telepítése után élesítjük, amikor már minden szoftver telepítve van, de felhasználók még nincsenek a gépen. Célszerű, ha az *aide.db* állományt elkészítése után átmásoljuk egy távoli gépre, és ellenőrzés előtt onnan töltjük le (például *wget* programmal), esetleg összevethetjük a saját és a távoli gépen lévő adatbázisokat.
Az is fontos, hogy a rendszeres jelentések eredményét ne tudja a támadó manipulálni, ezért e-mailben is elküldhetjük saját címünkre. A következő verziókban ez valószínűleg beépített funkció lesz. Addig pedig próbáljuk ki, hogy az *aide.conf* fájlban két `report_url` paramétert adunk meg:

```
report_url=stderr
report_url=stdout
```

Majd így futtassuk az `aide` parancsot, akár *cron* feladatként:

```
2 2 * * * /usr/local/bin/aide
➥-c /usr/local/etc/aide.conf
```

5. Lista Egy gyakorlati példa

```
@@define TOPDIR /

# az adatbázis helye
database=file:@@{TOPDIR}/root/a
➥ide.db

# a létrehozott adatbázis helye
#database_out=sql:host:port:dat
➥abase:login_name:passwd:table
database_out=file:/root/aide.db
.new

# ne tömörítse az aide.db
# fájlt
gzip_dbout=no

# mennyi üzenetet írjon ki
# futás közben
verbose=5

# a konfigurációs fájl verziója
config_version=0.0.1

# a jelentést ide írja
# lehet még pl.
# syslog:LOG_AUTH, stderr
report_url=stdout

# néhány saját ellenőrző
# szabály
All=R+a+sha1+rmd160
Norm=s+n+b+md5+sha1+rmd160
MyRule1=R+sha1
/etc  R
/bin  R
/boot R
/lib  R
/sbin R

/service R

=/tmp L
=/tmp/.ICE-unix L
=/tmp/.X11-unix L
/tmp/.X0-lock L

/usr R

=/var R

/var/lib R
/var/spool L

=/var/log$ R
/var/log/packages R
/var/log/setup R
/var/log/scripts R

# a naplófájlok
/var/log/*log >
/var/log/btmp >
/var/log/cron >
/var/log/debug >
/var/log/dmesg >
/var/log/messages >
/var/log/secure >
/var/log/spooler >
/var/log/wtmp >
```

```
➥2>/root/jelentes_`date
➥+%Y%m%d` | /bin/mail -s
➥"`date +%Y%m%d` aide
➥jelentes" email@cimunk.hu
```

A *security-l* listán olvastam egy olyan fájlintegritás ellenőrző megoldásról (amely tetszőleges implementációval megoldható, például *tripwire, aide, samhain*), ahol egy megbízható gép *ssh*-val bejelentkezik a vizsgálandó gépre, átmásolja a program adatbázisát és magát a program binárisát, lefuttatja az ellenőrzést, annak eredményétől függően küld pl. értesítést, majd letörli az ideiglenes állományait.
Az *AIDE* használata egyéb programokkal is kombinálható.

A *nagios*szal együtt könnyen megoldható az, hogy egyrészt az ellenőrzés automatikus legyen, ill. hogy behatolás detektálása esetén riasztást küldjön.

Hiányosságok

Más integritás ellenőrző alkalmazásokkal ellentétben az *AIDE* nem tud az adatbázisban csak bizonyos állományokra ellenőrizni. Az összes fájl és könyvtár vizsgálata pedig hosszadalmas lehet. Azonban ez a probléma egyszerűen megoldható, ha több konfigurációs fájlt és adatbázist tartunk, és mindig az éppen szükségest használjuk.
Egy másik apróság, hogy – például a *Tripwire* programmal ellentétben – az *AIDE* nem tudja súlyosság szerint rangsorolni a változásokat. Ezt némileg enyhít(het)i az, hogy bizonyára az *AIDE* csak egy biztonsági eszköz a sok közül, amivel gépünket egyébként is védjük.
További gond lehet, hogy az *AIDE* adatbázisa nem védett módosítás ellen, sőt nem is képes detektálni, ha az *aide.db* fájl megváltozott. Itt azonban jól jön az adatbázis egyszerűsége: kézzel beleírhatjuk az *aide.db*-re vonatkozó adatokat.
A *samhain* például démon módban is képes futni, és mind az elindításakor, mind a leállásakor egy jól beazonosítható jelet hagy, így nem lehet magát az integritás vizsgáló programot lecserélni. Ez a funkció sem található meg az *AIDE*-nél.
Az említett *samhain* érdekessége még, hogy fordításkor egy 64-bites egyedi azonosító képződik minden binárisában. Ha a támadó le is tudja cserélni a programot, a naplófájlokban, email üzenetekben látszani fog, hogy az nem az eredeti program.
Ha sok gépet felügyelünk, akkor bizonyára jól jönne egy központi monitorozó és menedzselési lehetőség, de sajnos ezt az *AIDE*, jelenleg nem támogatja, magunknak kell megfelelő körítéssel ezt a funkciót megvalósítanunk.

Összefoglalás

Az *AIDE* hatékonyan képes felderíteni, hogy mely fájlok változtak meg a gépünkön. Értékes segítséget ad nem csak a behatolás detektálásához, de az utólagos védekezéshez is, segítségével nem csak azt tudjuk meg, hogy vajon betörtek-e a gépünkre, de a telepített trójai és egyéb programokat is azonosíthatjuk. Más eszközökkel kombinálva pedig a hiányosságain is enyhíthetünk.

Sütő János
(jsuto@freemail.hu)

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

Zene a konyhában, és azon túl

Nem kérdés: saját zenei ízlésünk vetekszik a legjobbakéval. Akkor viszont miért is ne tudathatnánk mindenkivel, éppen mit hallgatunk?

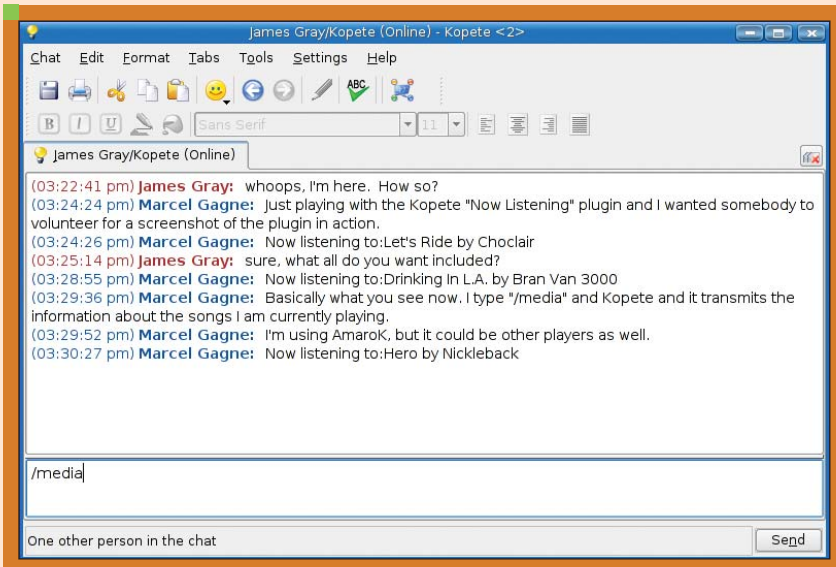
Nos ***Francois***, ez kétségtelenül egy igen érdekes zenei kollekció. Amúgy nem is tudtam, hogy ekkora ***Indie*** rajongó vagy. Ha jól látom jó pár érdekes számot szedtél össze az ***Amarokban***. Az igazat megvallva több olyan is van köztük, amiről még nem is hallottam. Szóval elmesélhetnéd nekem, hogy miféle zenéket szoktál hallgatni, hátha nekem is megtetszik néhány.

Hogyan? Nem, ***Francois***, nem akarom, hogy elküldd nekem a listát e-mailben. Vannak erre sokkal érdekesebb módszerek is. Tudod az a helyzet, hogy a zenei ízlésünk másokkal való megismertetése mostanság egyre népszerűbb. Számos ember blogjában találhatsz „most játszom” (***now playing***), vagy „nemrég hallgattam” (***recently played***) listákat, a zenékkel kapcsolatos társalgás pedig a szociális hálózatok egy jelentős összetartó ereje a weben. Aztán meg pontosan tudom, hogy te is használsz számos olyan linuxos alkalmazást, amelyek lehetővé teszik, hogy a listáidat megoszd egy, két, öt vagy akár ezer emberrel is, ha éppen ahhoz van kedved. Természetesen meg fogom mutatni, hogyan teheted ezt meg, de előbb el kell készülnünk. Amint látod, a vendégeink már meg is érkeztek. Isten hozott mindenkit ***Marcelnél***, ahol a legjobb linuxos és nyílt forrású programok találkoznak a legkiválóbb borokkal, no és persze a társaság is előkelő. Kérem mindenki keresse meg az asztalát, és helyezze magát kényelembe, amíg kiváló pincérünk elszalad a borért. ***Francois***, irány a pince! Azt hiszem a 2004-es ***Maison Chapoutier***-ből származó ***Village Latour Cotes du Rossillon*** kiválóan illik majd a mai menühöz. Hiszen miért is ne párosíthatnánk össze a kiváló zenei ízlést egy zamatos vörösborral? Találsz néhány rekesszel a déli szárnyban a harmadik sorban. Gyerünk, ***Francois***! A vendégeink már bizonyára megszomjaztak.

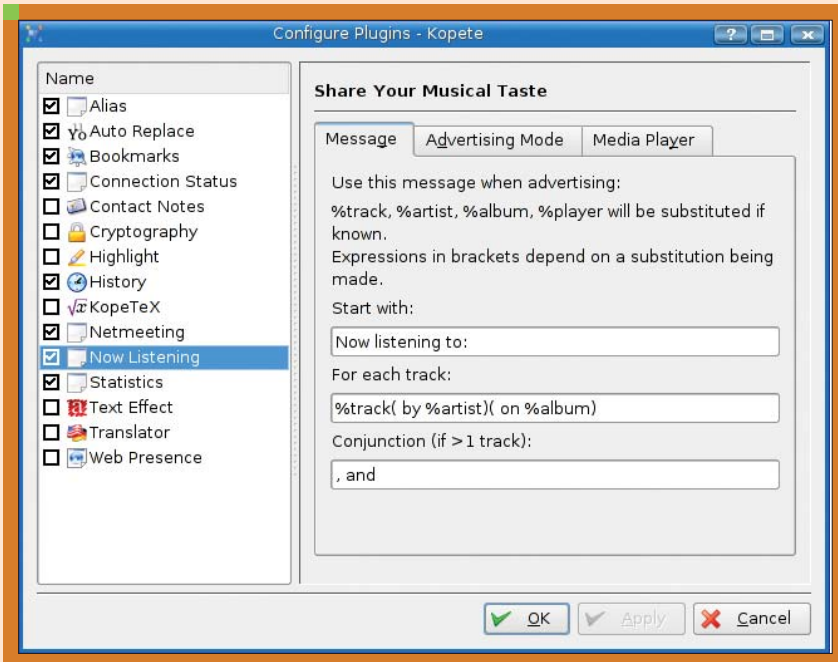
Francois-val éppen arról beszélgettünk, hogyan oszthatjuk meg barátainkkal (és bárki mással, akit ez érdekel), hogy milyen zenét szoktunk hallgatni. Linux alatt természetesen ez is könnyen megoldható, hiszen megvannak hozzá a megfelelő programok. Ha szeretünk valamit, és ezt meg szeretnénk osztani másokkal, annak az egyik egészen kézenfekvő módja az, ha beszélünk róla. A ***Kopete*** egy több protokollt is támogató univerzális üzenetküldő program, amit a ***KDE*** rendszer részeként szerezhetünk be. Kevesen tudják, hogy ennek az alkalmazásnak van egy ilyen „kibeszélő” funkciója is. Tegyük fel ugyanis, hogy be vagyunk jelentkezve egy ***IRC*** csatornára, vagy épp társalgunk egy ismerősünkkel a ***Jabber***, a ***Yahoo*** vagy bármely más szolgáltató segítségével. Nos, ilyenkor nem kell egyebet tennünk, csak kiadni a `/media` parancsot a csevegőablak alján található mezőben, és máris mindenki megtudja, éppen mit hallgatunk. A rendszer ugyanis elküld az összes velünk kapcsolatban álló felhasználónak egy üzenetet, ami az éppen hallgatott zeneszám előadóját, címét, és az album címét tartalmazza (***1. ábra***).

Ah, visszajöttél ***Francois***! Kérlek tölts a vendégeinknek! Mindenkit biztosíthatok, ez egy igazán finom bor.

A játszott zenék listájának a ***Kopete***-vel való elküldése egyaránt működik az ***Amarok***, a ***Kaffeine***, a ***Noatun*** és minden más a ***KDE*** alatt futó alkalmazással. Mi több, még az ***XMMS***-sel is képes együttműködni a program. Persze ahhoz, hogy a ***Kopete*** hajlandó legyen zenei listákat küldeni, előbb be kell kapcsolnunk a korábban említett bővítményét. Kattintsunk tehát a ***Beállítások*** (***Settings***) menüpontra a ***Kopete*** menüsorában, majd



1. ábra A Kopete Now Listening bővítményével a zenék lejátszására használt programunk automatikusan képes közölni barátainkkal, hogy éppen mit hallgatunk



2. ábra A Kopete Now Listening nevű bővítményének aktiválása és beállítása

válasszuk a ***Bővítmények Beállítása (Configure Plugins)*** pontot. Egy ablak tűnik elő, amelyben a Kopete jelenleg telepített bővítményeinek listáját látjuk (***2. ábra***).

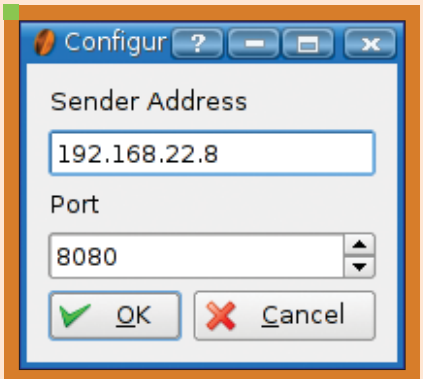
Jelöljük tehát be a ***Now Listening*** nevű bővítmény mellett található jelölőmezőt. Most a jobb szélen három fület láthatunk. Ezekkel megváltoztathatjuk az azonnali üzenetküldő alkalmazásunk által használt megjelenítést, ha éppen elégedetlenek lennénk az alapértelmezett beállításokkal. A számunkra legérdekesebb fül tehát az ***Advertising Mode (közzététel módja)*** feliratú.

Az alapértelmezett beállítások szerint be kell gépelnünk a ***/media*** parancsot ahhoz, hogy az információ nyilvánossá váljon. Ugyanakkor azt is beállíthatjuk, hogy a dolog teljesen automatikusan működjön. Egy másik beállítással a ***Now Listening*** információt akár az on-line állapotunkat jelző információhoz is hozzácsatolhatjuk. Ha beállítottunk mindent, amit kell, kattintsunk az ***OK*** gombra.

Mindez természetesen csak szöveges információ lesz, vagyis amit mi hallunk, azt más – legalábbis ettől – nem fogja. Ahhoz, hogy maga a hang is eljusson másokhoz, sugároznunk kell. Gyakori, hogy az emberek tekintélyes zenegyűjteménnyel rendelkeznek, de azt több különböző gépen tartják, amelyek rendszerint a ház két átellenes végében találhatók. Mármost nem lenne nagyszerű, ha az egyik gépen lejátszott zenét közvetíteni tudnánk a másikra, és így mindig hallanánk, akárhol is vagyunk a lakásban? Saját hálózatunk helyi műsorszórójává válni nem is olyan nehéz, mint elsőre gondolnánk. Igazából az is elképzelhető, hogy azok a programok, amelyekkel ezt megtehetjük, már ott is lapulnak csendben a gépünkön, csak nem tudunk róluk. Az egyik lehetőség a ***Jürgen Kofler*** és ***Christphe Thommeret*** által fejlesztett Kaffeine, ami nem egyéb, mint egy népszerű, ***KDE*** alatt futó médialejátszó. A ***Kaffeine***-t a legtöbben videó-lejátszóként ismerik, az igazság azonban az, hogy a program képes ***DVD***-ket, ***VCD***-ket és közönséges hang ***CD***-ket is „felszolgálni” amellett, hogy valóban számtalan videóformátummal is elboldogul. Egyes ***Linux*** terjesztések, amelyek alapértelmezésként telepítik ezt a lejátszót a ***Konqueror*** böngészőbe is integrálják azt, vagyis a weblapokon megjelenő on-line videókat is ez a program játssza le. A ***Kaffeine***-nek mindezeken felül létezik egy műsorszóró üzemmódja is. Kattintsunk a menüsorában a ***Fájl (File)*** menüre, válasszuk a ***Hálózati Műsorszórás (Network Broadcasting)*** menüpontot, és jelöljük be a ***Műsorszórási Adatfolyam Küldése (Send Broadcast Stream)*** opciót (***3. ábra***).

Egy kis ablak fog megjelenni, amelyben a használni kívánt port számát kell megadnunk. Az alapértelmezés itt

3. ábra A Kaffeine lejátszót könnyedén rábírhatjuk a műsorszórásra is

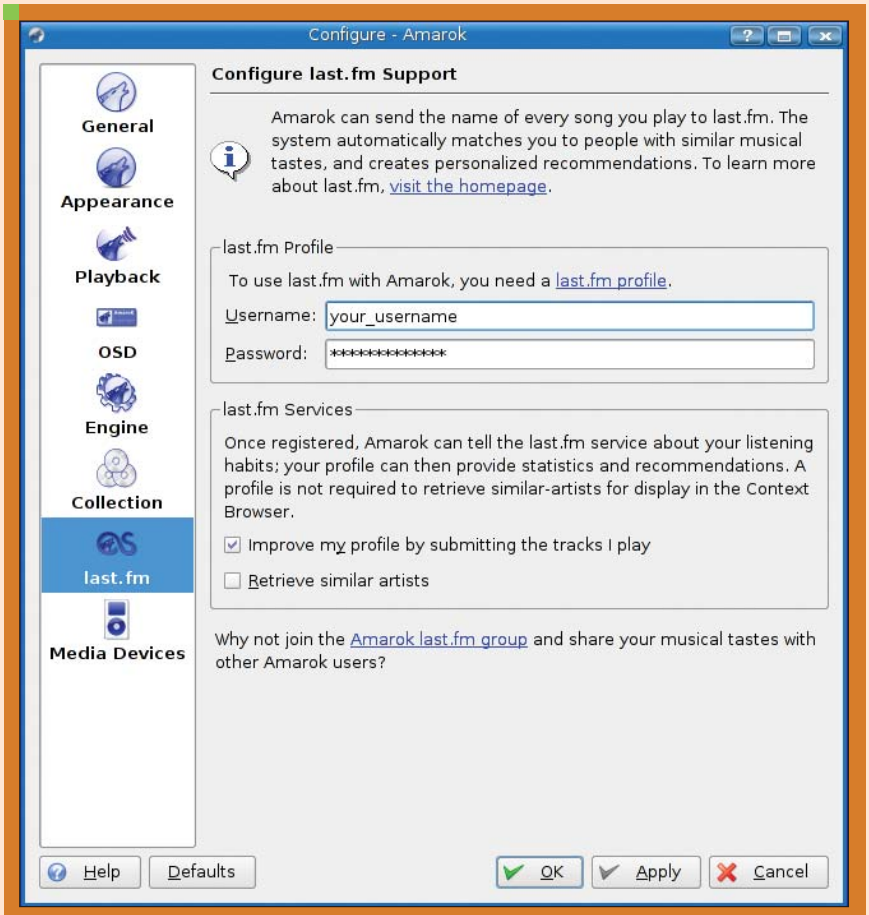


4. ábra A vevő oldalán meg kell adnunk a Kaffeine-nek, hogy honnan jön az adás

Megjegyzés
Marcel Amarokról írt cikke a következő címen olvasható:
➲ www.linuxjournal.com/article/8558

a 8080-as kapu, de természetesen azt adunk meg, amit akarunk (és ami még szabad). Kattintsunk az OK gombra, és a rendszer máris készen áll a műsorszolgáltatásra. Ami azt illeti, ettől a pillanattól kezdve bármit játszunk le a programmal – legyen az zene vagy videó – az „adásba kerül". Annak, aki a mi hálózatunkhoz csatlakozik, és venni akarja az adást, a *Kaffeine* egy futó példányán kívül a konfiguráció némi átszabására lesz szüksége. „Vevő oldalon" tehát kattintsunk a *Fájl* menüre, válasszuk a *Hálózati Műsorszórás (Network Broadcasting)* menüpontot, de ezúttal a *Sugárzott Adás Vétele (Receive Broadcasted Stream)* opció a számunkra megfelelő. Ismét egy beállítóablak jelenik meg (*4. ábra*). Írjuk be a küldő *IP* címét, annak a portank a számát, amelyen az adó kommunikál (ez az, amit az imént a küldő oldalon beállítottunk), aztán kattintsunk az *OK* gombra. És ezzel készen is volnánk. Bármit is sugároz a helyi *DJ*, hallani vagy adott esetben látni fogjuk.

A mostanság tapasztalható *Web 2.0* őrület kapcsán megjelent néhány „organikus" zenei szolgáltatás is. Egyesek közülük őszintén szólva egész érdekesek. A két kedvence a *Pandora* és a *Last.fm*. A *Pandora* egy olyan „mesterséges intelligencia", aminek ha

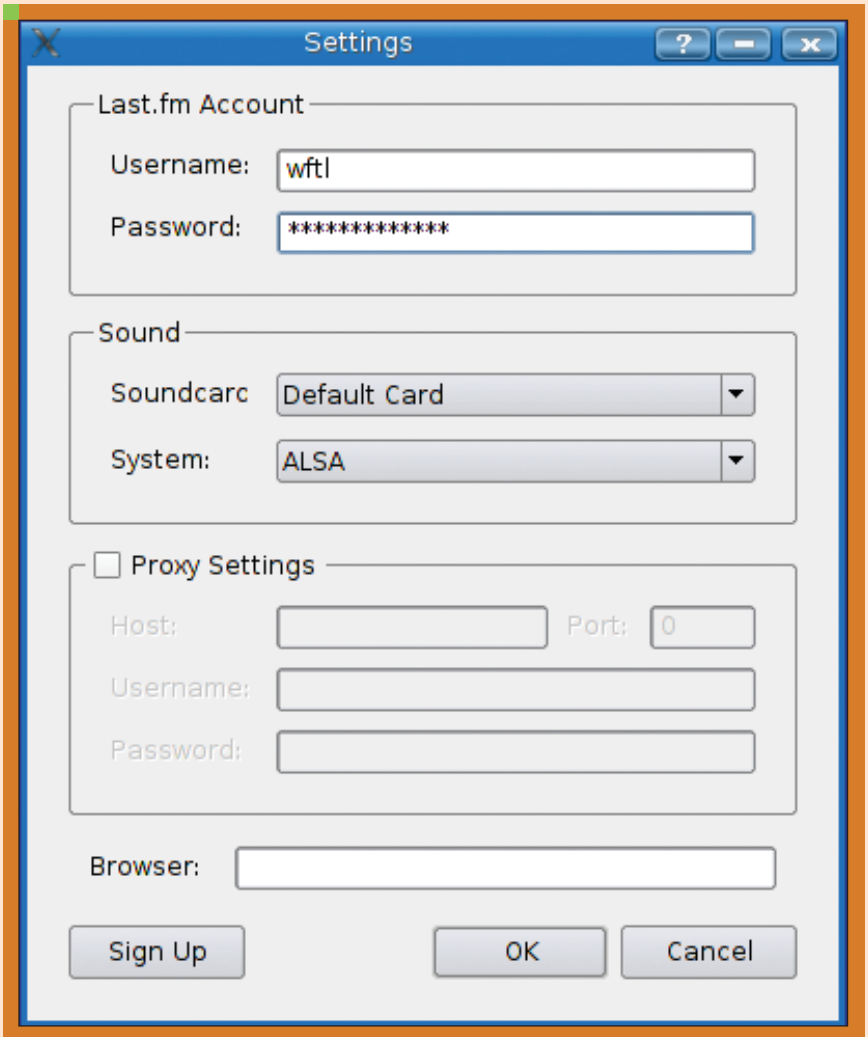


5. ábra Az Amarok legújabb változatával már közhírré tehetjük zenei ízlésünk részleteit is, hiszen a program támogatja a Last.fm szolgáltatásit

megmondjuk, hogy milyen zeneszámokat szeretünk, akkor javasol nekünk más számokat, amelyeket – szerinte – szintén szeretni fogunk, és még részleteket is le tud játszani belőlük. És ami azt illeti, egész jó a találati arány. Persze a *Pandora* amolyan magányos élvezetet tud csak nyújtani, szemben a *Last.fm*-mel, ami viszont részévé válhat a szociális hálónknak. Segítségével ugyanis zenékről társaloghatunk a barátainkkal, blogot írhatunk az általunk preferált számokról, új albumokat ismerhetünk meg, vagy megnézhetjük, mit hallgatnak mások. No és persze mi magunk is közhírré tehetjük, hogy mit szeretünk. Ahhoz, hogy megkezdődhessen az élvezet, először is rendelkeznünk kell egy *Last.fm* fiókkal. Aztán egy *Audioscrubbler* névre hallgató ügyes kis szoftver segítségével máris közvetíthetjük a rendszernek saját zenei ízlésünk részleteit, sőt azt is, hogy ebben a pillanatban éppen mit hallgatunk. Az *Audioscrubbler* amúgy nem más, mint egy adatbázis-kezelő, ami nyilvántartja a zenehallgatási szokásainkat, statisztikát készít belőle, és megjósolja, hogy körülbelül mi az, amit szeretni fogunk, vagy éppen nem. 2005 decemberében is írtam már egy a *KDE* alatt működő és igen kiváló zenei programról, az *Amarokról*. Az az igazság, hogy ennek a programnak egyszerűen fönn kell lennie minden zeneszerető ember gépén, és nekem is az abszolút kedvencem. Az *Amarok* képességeinek csak a felsorolása is túl hosszú ahhoz, hogy itt és most belemerüljek, de azért nem árt egy kis emlékeztető. Először is van neki egy kiváló borítókezelője, ami az Amazonról tölti le a borítóképeket. Van benne

Tipp
Az *XMMS*-hez, *Noatun*-hoz és más *Linux* alatt működő lejátszóprogramokhoz az *Audioscrubbler* bővítményeket a *Last.fm Letöltések* (*Downloads*) oldalán találjuk.

6. ábra A Banshee beállításokat tartalmazó füle



7. ábra A Last.fm Media Player beállítóablaka

olyan böngésző, ami megtanulja, hogy mi az, amit gyakran hallgatunk. Bőrözhető a felülete, támogatja az *iPod*-ot (és persze más lejátszókkal is kiválóan működik), nagyszerű megjelenítő eszközei vannak (a *libvisual*-t használja), és még számos kiváló funkcióval rendelkezik. Ami azt illeti, még a zenék szövegét is letölthetjük vele, tehát akinek kedve van hozzá, az együtt énekelhet a kedvenc előadójával anélkül, hogy a megfelelő szavakon kellene törnie a fejét. Mindezt azért írtam le, mert az *Amarok* legújabb változata már a *Last.fm* rendszert is támogatja, hiszen beépített *Audioscrubbler* támogatással rendelkezik.

Ha a *Last.fm* szolgáltatásait az *Amarokon* keresztül szeretnénk elérni, kattintsunk a *Beállítások (Settings)* menüre, és válasszuk az *Amarok Beállítása (Configure Amarok)* pontot. A beállítóablakban (*5. ábra*) számos kategóriát találunk a bal ablakszárnyon. Ezek közül az egyik a *Last.fm* szolgáltatás, köszönhetően az *Audioscrubbler* programnak. Adjuk meg tehát a *Last.fm* rendszerhez szolgáló nevünket és jelszavunkat, majd kattintsunk az *OK* gombra. Persze a *Last.fm* szolgáltatásait a *GNOME* közösség egyes tagjai is felismerték. Ez megmutatkozik az úgyszintén kiváló *Banshee* zenelejátszó fejlődésében, hiszen immár ez is képes ugyanezekre a funkciókra. Ha tehát meg akarjuk osztani másokkal, hogy mit szoktunk hallgatni, kattintsunk a *Szerkesztés (Edit)* pontra a menüsorban, majd válasszuk a *Bővítmények (Plugins)* pontot. Erre megjelenik a *Banshee* bővítmények beállítására szolgáló ablaka, amelyben már ki is van választva az *Audioscrubbler* program. Kattintsunk tehát a *Beállítások (Configuration)* fülre (*6. ábra*), majd jelöljük be az *„Enable song reporting”* felirat melletti mezőt. Adjuk meg a felhasználói nevünket és jelszavunkat, majd kattintsunk a *Bezár (Close)* gombra.

Akár az *Amarok*, akár a *Banshee* programot használjuk (vagy bármi mást), a lejátszott zeneszámokkal kapcsolatos információt immár megkapja a *Last.fm* rendszere, így minden más felhasználó megtudhatja, mi az, amit szeretünk. Korábban említettem, hogy bizonyos megoldások segítségével a barátaink akár bele is hallgathatnak abba, amit mi is hallgatunk. Nos, erre a *Last.fm* rendszernek is megvan a maga megoldása. Ha valaki *Last.fm DJ*-vé akar előlépni, annak elő kell fizetnie erre a szolgáltatásra. Ahhoz tehát, hogy kipróbálhassam, miről is van szó, én magam is elköltöttem horribilis 3 dollárt, és létrehoztam

8. ábra A Last.fm szabadon hozzáférhető zenelejátszója

egy a *wftl* fiókomhoz csatlakozó rádióállomást. Ezen a fizetős szolgáltatáson keresztül lehetőség van arra, hogy barátaink, egy amúgy a *GPL* licenc alatt terjesztett és a szolgáltató webhelyéről letölthető szoftver segítségével ráhangoljanak a virtuális rádióadónkra, és hallgassák, amit „összehoztunk". (Igen, más operációs rendszerekhez is létezik ennek a lejátszónak a megfelelő változata.)
Ha tehát használni szeretnénk ezt a lejátszóprogramot, akkor egy tetszőleges könyvtárba bontsuk ki a letöltött csomag tartalmát a következő paranccsal:

```
tar -xjvf LastfmLinux-
➥1.1.4.tar.bz2
```

Ebben az esetben semmit nem kell külön lefordítanunk. Ha futtatni akarjuk a programot, egyszerűen lépjünk be abba a könyvtárba, ahova telepítettük, aztán adjuk ki a

```
./player
```

parancsot. Ennyi az egész. Az első futtatásnál egy beállítóablak fog megjelenni, ahol a *Last.fm* fiókunkkal kapcsolatos információkat kell megadnunk.
Ha minden szükséges adatot megadtunk, kezdődhet is a móka. Ha barátain, vagy családtagjaink rá akarják hangolni saját virtuális vevőkészüléküket az általunk „üzemeltetett" csatornára, nem kell mást tenniük, mint elindítani ezt a programot. A lejátszó főablaka mellett egy kisebb *„Radio Control"* feliratú ablak is megjelenik. Mielőtt elkezdhetnénk lejátszani egy olyan csatornát, ami nem a miénk, létre kell hoznunk egy rádióállomást a saját *Last.fm* oldalunkon található listában. Ha ezzel megvagyunk, kattintsunk a *Personal Radio* gombra, és a rendszer átveszi az általunk futtatott lejátszótól az adatfolyamot, legyen annak a forrása az *Amarok*, A *Banshee* vagy bármi más, ami képes az *Audioscrubbler* szolgáltatását használni.
Aki nem ezt a lejátszót szeretné használni, az a jobb alsó sarokban található beállítógombra kattintva megadhatja, hogy mivel akarja a zenéit közvetíteni. Adót amúgy a címének a megadásával is beállíthatunk (például `lastfm://user/wftl/personal`).
Nos, az idő meglehetősen leszaladt, de azért még egy rövid időre had tereljem vissza a szót az *Amarokra*.
Ha részévé váltunk a *Last.fm* szociális hálójának, az nem jelenti azt, hogy kötelesek vagyunk a *Last.fm* lejátszóját használni. Az *Amarok* képes ráhangolódni a *Last.fm* által *Szomszédos Adóknak (Neigbour Radio)* nevezett állomásokra is. Ahogy egyre több és több általunk lejátszott zeneszámról szerez tudomást a rendszer, egyre jobban megismer bennünket, pontosabban az ízlésünket, így képes hozzánk rendelni a megfelelő „szomszédokat". Ezek olyan emberek által üzemeltetett adók, akiknek a zenei ízlése közel áll a miénkhez.
Kattintsunk az *Amarok* menüsorában az *Engage*, majd a *Play Last.fm Stream* pontra, aztán válasszuk a *Neighbour Radio* pontot. Még ha egyelőre nincsenek is „zenei szomszédaink", akkor is hallgathatjuk a *Global Tag Radio*-t, ami olyan zenéket játszik le, amelyek az idők során a teljes *Last.fm* közösség ízlésének leginkább megfeleltek. Minden ilyen adó neve stílus alapján van rendezve, vagyis külön találjuk meg a *Rock*, a *Pop*, a *Dance* a *Rap* és minden más stílus legjobbjait. Ez ugyan nem túl személyes, de kiváló módja új zenék felfedezésének.
Nos, kedves vendégeim, az óra a falon azt mutatja, hogy megint közeleg a záróra. Azt hiszem egy egészen kiváló csatornát lehetne összehozni azokból a zenékből, amiket itt az asztaloknál ma lejátszottunk. Persze nem csoda, hogy ilyen kiváló ízléssel rendelkező embereknek a zenei ízlése is figyelemre méltó. *Francois* kérlek tölts még utoljára a vendégeink poharaiba, had mondjuk köszöntőt erre a szép napra. Emeljük tehát poharainkat, és igyunk egymás egészségére!

Linux Journal 2005., 150. szám

Marcel Gagne, aki számos díjnyertes könyvet írt már, Mississaugaban, Ontario államban él. Ő a szerzője a Moving to Ubuntu Linux című kötetnek, amely történetesen az ötödik, az Addison-Wesley kiadónál megjelent könyve. Rendszeresen szerepel a televízióban is, mint a Linxu operációs rendszerrel kapcsolatos szakértő. Marcel mellesleg repülőt is tud vezetni, valaha TOP-40-es DJ volt, alkot a tudományos fantasztikus irodalom és fantasy területén is, és egy T-Rex origamit hajtogat.
A mggagne@salmar.com címen érheti el, aki írni szeretne neki, a webhelyén (www.marcelgagne.com) pedig szintén számos hasznos dolgot találhatunk, többek között borokkal kapcsolatos linkeket is.

KAPCSOLÓDÓ CÍMEK

A cikkhez kapcsolód anyagok:
➲ www.linuxjournal.com/article/9172

Régi idő játéka – SCUMMVM

A lap Olvasói között remélhetőleg szép számban vannak azok, akik jóleső nosztalgiával gondolnak vissza olyan legendás kalandjátékokra, mint a Maniac Mansion, a Monkey Island, a Sam & Max vagy a Broken Sword. Ezekkel a játékokkal ma a legfőbb gond az, hogy a túl gyors számítógépeken és a nyílt forrású operációs rendszereken nehéz vagy akár lehetetlen működésre bírni őket. Az ilyen játékok szerelmeseinek készül a SCUMMVM, egy olyan futtató környezet, mellyel az előbb felsoroltak mellett még több tucat másikkal játszhatunk akár Linux alatt is.

A *SCUMMVM* név a *Script Creation Utility for Maniac Mansion Virtual Machine* rövidítése. Eleinte a *LucasArts* kalandjátékok futtatására hozták létre, de később több más gyártó cég játékaira is sikerült „ráidomítani". Közülük talán legismertebb a *Revolution*, amely olyan sikeres programokat tudhat magáénak, mint a *Beneath a Steel Sky* vagy a maga idejében egyedülálló *Broken Sword*. De ne feledkezzünk meg az *Adventure Softról (Simon the Sorcerer)* és a *Coktail Visionról (Gobliiins)* sem! Aki kíváncsi a *SCUMMVM* segítségével életre kelthető játékok teljes listájára, látogasson el a ➲ *http://www.scummvm.org/compatibility.php* címre, ahol az is megtalálható, hogy az adott játék mennyire kompatibilis a *SCUMMVM*-mel.

Azoknál a programoknál, ahol ez még a játszhatóság rovására megy, folyamatosan dolgoznak ennek javításán, így érdemes időnként visszalátogatni ide és ellenőrizni, hogy történt-e változás. Fontos elmondani, hogy a *SCUMMVM*-nek nem részei a felsorolt játékok, ahhoz hogy használni tudjuk őket rendelkeznünk kell az eredeti program (jogtiszta) fájljaival. Ez alól két, mostanra ingyenesen letölthetővé tett kivétel van: a *Beneath a Steel Sky* és a *Flight of the Amazon Queen*. Mindkettőnek több változata letölthető

1. ábra A két kép közti különbségen jól látható a Supereagle szűrő által nyújtott minőségjavulás

a ➲ *http://www.scummvm.org/downloads.php#extras* címről. Ugyanitt találhatunk átvezető animációkat a *Broken Sword* első két részéhez; ezekre szükség lesz ha játék közben szeretnénk látni ezeket a képsorokat. Ebben az esetben a *SCUMMVM* fordítása előtt töltsük le és telepítsük a *libmpeg2* és *libVorbis* csomagokat is!

Maga a *SCUMMVM* a lehető legteljesebb mértékben kimeríti a „kereszt-platformos" fogalmat: *Linux* és *Windows* mellett *Mac OS X*, *BeOS*, *Amiga* változata is van, de letölthetjük akár *Playstation 2*-re vagy *Symbanos* okostelefonra is! Keressük ki a nekünk megfelelő változatot a ➲ *http://www.scummvm.org/downloads.php* címről, és telepítsük fel. A sikeres futtatáshoz előzőleg feltelepített *SDL*-re lesz szükség, illetve ha a játék *MP3*-ba tömörített zenéjét is szeretnénk hallgatni, akkor a *madlib*-et is szerezzük be és fordítsuk le a *SCUMMVM* előtt.

A forráskód fordítása a szokásos

```
./configure && make
```

parancssorral történik, ezután pedig a program a

```
./scummvm
```

paranccsal indítható.

A legegyszerűbb dolgunk akkor lesz, ha a játék(ok) fájljait a merevlemezre játékonként külön könyvtárba másoljuk, de használhatjuk a gyári *CD*-ket is. Nincs szükség az eredeti játék minden adatállományára, a *SCUMMVM* által igényelt fájlok listája

2. ábra A Broken Sword hírhedt ír kecskéje

a ➲ *http://www.scummvm.org/documentation.php?view=datafiles* oldalon található. A több *CD*-n elterülő játékoknál (jelen pillanatban a *Broken Sword* első és második része) bizonyos fájlok ugyanolyan néven szerepelnek mindkét lemezen. Ebben esetben az előbb említett oldalon leírt eljárást követve nevezzük át őket: az első *CD*-n található *speech.clu*-ból *speech1.clu* lesz, a másik lemezről származót pedig nevezzük át *speech2.clu*-ra.
A *Broken Sword 2* esetében ezt meg kell tennünk a *music.clu* fájlokkal is.
A megjelenő felületen az *Add Game...* gomb segítségével adhatunk játékot a *SCUMMVM* listájához. Válasszuk ki a játék könyvtárát, kattintsunk a *Choose* (kiválaszt) gombra, és ha mindent jól csináltunk, a program fel is ismeri a játékot. Ha az általános beállítások megfelelőek, nyugtázzuk az *OK* gomb megnyomásával. Ezután a listában kiválasztott játékot a *Start* feliratra kattintva indíthatjuk. A játék futása közben az *F5* billentyűvel juthatunk a menübe, ahol állást menthetünk/tölthetünk be vagy kiléphetünk a *SCUMMVM*-ből.
A hagyományos *320x200*-as felbontást használva egy mai monitoron (főleg teljes képernyős módban) rettenetesen kockás képet kapunk, de a *SCUMMVM* ezen is segít.
A kezelőfelület *Options* gombját kiválasztva a *Graphics* fülön választhatunk számos minőségjavító szűrő közül a *Graphics mode* redőnyt lenyitva. Itt beállíthatjuk a teljes képernyős módot illetve a képarány esetleges torzulását is kiküszöbölhetjük. Az itt elvégzett módosítások az összes játékra érvényesek lesznek, kivéve ha az

3. ábra A világuralomra törő csáp

adott programot a listából kiválasztva az *Edit Game...* gombra kattintunk, ahol lehetőségünk nyílik minden általános beállítást felülbírálni. Ehhez a megfelelő fület aktívvá téve tegyünk egy *X*-et az *Override global* felirat előtti négyzetbe, majd alatta végezzük el az általunk kívánt beállításokat.
Természetesen nem csak a kezelőfelület segítségével, hanem parancssorból is használhatjuk a *SCUMMVM*-et. Ehhez tudnunk kell a használni kívánt játék nevének *SCUMMVM*-es rövidítését, ami megtalálható a játékok listáját tartalmazó oldalon és a *readme* fájlban egyaránt, de lekérhetjük a

```
scummvm -z
```

paranccsal is. A többféle változatban megjelent játékoknak több kódja is van, a *Simon the Sorcerer* első részéhez például nyolc kód tartozik, ilyenkor figyelmesen válasszuk ki a mi változatunkhoz illőt! Ha például az ingyenesen letölthető *Beneath a Steel Sky*-t szeretnénk indítani, melynek kódja `sky`, a következő parancsra lesz szükségünk:

```
scummvm sky
```

Jó néhány kapcsolóval is bővíthetjük a parancsot, ezek közül a legfontosabbak:

`-f` teljes képernyős mód
`-g` grafikus szűrő kiválasztása
`-n` feliratozás bekapcsolása
`--aspect-ratio` képarány korrigálása

Ha tehát teljes képernyőn, feliratozva és a *hq2x* szűrővel szépítve szeretnénk látni a *Beneath a Steel Sky*-t, a következő parancsot gépeljük be:

```
scummvm -f -n -g hq2x sky
```

Ha a játék többszöri végigjátszása után bizonyos részeken szeretnénk gyorsan „végigszaladni”, erre is van mód: a *CTRL-F* billentyűkombinációval gyorsabbá tehetjük a játékot. Célszerű azonban óvatosan bánni ezzel a lehetőséggel, mivel esetenként a szinkron elcsúszásához vagy más hibákhoz vezethet.
Más okból is előfordulhat – bár szerencsére igen ritkán van rá példa – hogy a program lefagy. Ilyen esetekben sem veszik kárba akár több órányi játék, ugyanis a *SCUMMVM* ötpercenként lementi az aktuális játékállást a nulladik mentéshelyre, ezt bizonyos játékoknál az elmentett állások betöltésére szolgáló menüben a *Restore Autosave* gombra kattintva vagy parancssorból indítva a `-x` kapcsoló használatával tölthetjük vissza.
Végül egy érdekes kapcsolót ajánlok a kedves Olvasó figyelmébe: `--alt-intro`. Ennek segítségével a *Beneath a Steel Sky* és a *Flight of the Amazon Queen* című játékok *CD*-s változatának alternatív bevezető képsorait nézhetjük meg.

Jó játékot mindenkinek!

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

Theocracy

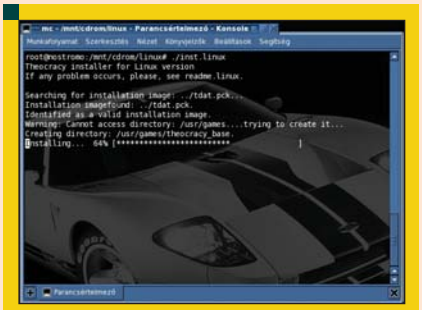
Meglehetősen ritka az a jelenség, amikor egy hazai fejlesztésű játékot a készítők linuxos képességgel ruháznak fel. Most egy ilyen „fehér hollóról” olvashattok, melynek stílusa ötvözi a valós idejű stratégiai, valamint a menedzselős kategória jellemeit.

Formabontás

A *Theocracy* közel hat éve jelent meg, a *Philos Lab* csapatának munkájaként. A nagy reményű kód több szempont alapján is kilógott a sorból – számunkra leginkább azért különleges, mert a kereskedelmi jelleggel terjesztett megvalósítások között elsőként rendelkezett hivatalos támogatású linuxos binárissal. Természetesen a komoly potenciálokat soroló dicsőséglistát szükségtelen lenne említenem, azonban két további fő erény mellett nem mehetek el szótlanul: ebben a játékban egyszerre akár százas nagyságrendben is mozgathatta egységeit a nagyérdemű, miközben a kód a címben említett stílusok jegyeit teljesen egyedien ötvözte. A formabontó projekt így önfeledt játékkal ajándékozta meg a közönségét, a mostani játékrovattal pedig nem titkolt célom kipróbálásra csábítani a kihívásokat kedvelő *linuxos* „hadvezéreket” is.

Keserű körítés

A *Theocracy* egyedi jellemének, rajongói táborának ellenére hányatott sorsú programként vonult be a számítástechnika történelmébe. Már a születése környékén problémák hátráltatták *Vámosi Zsoltékat*, hiszen a kiadást felvállaló *Interactive Magic* röviddel a publikálás előtt felfüggesztette addigi tevékenységi körét. Így hiába ragasztott a játékra *„Age of Empires 2 Killer”* jelzőt a legtöbb külföldi szaksajtó, az elkészült projekt kiadó nélkül maradt. Végül az *UBI Soft* karolta fel a reményteljes próbálkozást, komoly csúszással az eredeti elképzelésekhez



1. ábra A játék telepítése

2. ábra A Theocracy Linuxon, KDE felületen fut

3. ábra Egy küldetés története

képest. Ezzel azonban nem értek véget a megpróbáltatások: *Zsolt* később szoftverjogi problémák miatt sajnos szabadságvesztésre ítéltetett. A programhoz ebből adódóan nincs naprakész frissítés, hiszen maga a *Philos Lab* csapata is megszűnt létezni. Összességében a *Theocracy* mára egyfajta posztumusz-projekt szerepet tölt be, igen nehéz beszerezhetőséggel. A hozzáférést illetően szinte csak két csatorna létezik. Egyikük a ➲ http://www.tuxgames.com *online* rendelési lehetősége, a maga *42 dolláros* ajánlatával. A másik út sajnos az illegalitáshoz közelít: a hazai fejlesztésű játékprogram a *p2p* fájlcserélő hálózatok egyik gyakori vendége.

A töltelék

A játék izometrikus nézetű világa az *Azték* mondák felé kalauzolja az érdeklődőket. A *XV. századi Közép-Amerikában* a tenger felől érkező idegen hódítók és a velünk szomszédságban élő csoportok egyaránt fenyegetést jelentenek frissen szerzett provinciánkra, így a lehető leghamarabb birodalommá kell egyesítenünk a hozzánk kapcsolható népeket – bármi áron. Mindezt az ismerős valós idejű stratégia vonalán kell megtennünk, azonban menedzselési feladatok és (halovány) szerepjáték hatások is felfedezhetőek a játékban, a megszokott „kijelölöm a csapatom, majd beküldöm harcolni” stílus mellett.
A birodalom fenntartása nem egyszerű feladat, mint ahogyan az sem, amiként a próféciák, krónikák instrukciói szerint egy-egy legendás személy

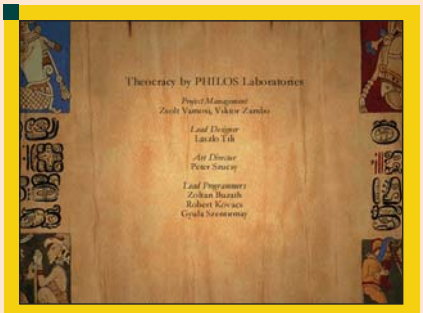
4. ábra Munkában a rabszolgák

5. ábra Már induláskor is hatalmas a sereg!

küldetéseit személyesen kell megélnünk. Az *Azték* világban egyaránt nagy szerepet kap a harc, a diplomácia és a felderítés is – ezen felül a varázslatok, találmányok és a véres áldozatok is a javunkat szolgálhatják. Birodalmunk alappillérei a rabszolgák, akiken felül állnak a varázslók, harcosok (köztük akár jaguáridomárok is). A vezető (csakúgy, mint bármely kiemelt személy) hatékony védekezésre általában képtelen: a fő egységek ezért minden áron megvédendőek, haláluk a küldetés azonnali végét jelenti.
Az összetett játékmenet, és a teljesen szabad lehetőségek adják a *Theocracy* világának sajátosságait. Az irányítást és a kezelést szerencsére könnyen el lehet sajátítani, mivel minden egyes felirat magyarul szól hozzánk, a *„Kezelési útmutató”* menüt pedig még videóbetétek is segítik. Ezeket a videókat nem javallott átlépni: néhányszori megtekintésük megfelelő rutinnal ruházza fel a játékost.

Linuxon...

Installálás ügyén a játék első lemezét befűzve, annak *Linux* mappájába állva el kell indítanunk az ott található *inst.linux ELF binárist*. Amennyiben a *cdrom* eszközünk csatolása nem az */etc/fstab* állomány alapján történik (hanem a *supermount* mechanizmus, vagy a *KDE* csatoló szolgáltatása által), úgy a játék nem telepíthető. Adjuk tehát hozzá az */etc/fstab* bejegyzéseihez az optikai tároló csatolhatóságát, az eszköz paramétereihez pedig az *exec* jelölést. A futtathatóság e jelölése nélkül szintén lehetetlen telepíteni! Miután a lemezt ezek alapján



6. ábra Tagadhatatlanul hazai gyökerek

befűztük, a telepítő fájlt *root* jogkörrel adjuk át a *shell*-nek (`su, sh inst.linux`), mely néhány egyszerű kérdés kíséretében lefut.
A hangeszköz, a csatolási pontok, és az elérési utak meghatározása kiemeltnek számítanak, így az ide vonatkozó válaszok minden esetben a valós környezetet kell leírják.
Miután a játék `theocracy` nevű indító szkriptje sem mai, így az indítás sok esetben problémákba ütközik.
A környezeti beállításokat tartalmazó fájl átírása helyett tegyük a következőket: az alapértelmezett */usr/games/theocracy* útról másoljuk az összes **.so* állományt az */usr/lib* útra, az */etc/X11/xorg.conf* állományban pedig az asztal színmélységét állítsuk *16 bitesre* (`DefaultDepth 16`).
A *Theocracy* ezek után a fő elérési úton található *theocracy_real* állomány segítségével, felhasználóként indítható. A mentések és beállítások kulturált módon, a játékos személyes mappájában (*/home/$/.theocracy*) tárolódnak.
Fontos megemlítenem, hogy a második *CD* lemezre szükség lesz a játék idejére! Annak ellenére, hogy a kettes számú *diszk* csupán videó betéteket és zenéket tárol, nélküle nem indul a program... A linuxos kód étvágya egyébként meglehetősen szerény: egy *500MHz* órajelű *x86* processzoron, *128MByte* memória társaságában már vígan fut, miközben semmilyen *3D* grafikai egységet sem kíván. A hálózati játékmód ezen felül természetesen működő hálózati kártyát feltételez!

Külsőségek

Mivel a *Theocracy 2000*-ben lett publikálva, így a képi világa természetesen mára már túlhaladott. Ez igaz a játékmotor által generált világra és a videóbetétekre egyaránt. A mozik, oktató videók részlethiányai kissé zavaróak, a játék grafikai túlhaladottsága azonban ebben a kategóriában nem jelent automatikusan hátrányt. Mivel a kiadása óta eltelt hat év sem tudta feledtetni az egyik legeredetibb hazai alkotást, így legyen szó bárkiről, csak javasolni tudom e játék beszerzését.
S ha már a második lemeznél említettem a kísérő zenét: a *CD* audió sávja (minősége alapján) egy indián jellegű, önálló zenei lemezként is megállná a helyét! A játék egyedülálló összképét talán azzal tudom legjobban érzékeltetni, hogy amióta a *Theocracy* gyári példánya a polcomat díszíti, azóta erre az ereklyére mint féltve őrzött kincsre tekintek.

Kovács Zsolt (kovi@linuxforum.hu)

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