Beköszöntő

Júniusi számunkban elsősorban a szoftverfejlesztőknek kedveztünk, hiszen számos cikk szól valamilyen módon a programozásról, hibakeresésről, kezdő és haladó szinten egyaránt.
Megismerkedhetünk néhány olyan módszerrel, amelyek segítségével ***Perl*** programokban kereshetünk kódolási vagy logikai hibákat. Szó lesz aztán egy módszerről, amely a ***LaTeX*** közismerten kiváló „matematikai képességeit" kihasználva varázsol képleteket weblapokra a ***PHP*** nyelv közbeiktatásával.
Fülöp Balázs ebben a számban egy új sorozatot nyit, amelyben a ***Java*** nyelvet kívánja bemutatni teljesen kezdő szintről indulva.
Komáromi Zoltán ezúttal legózni tanítja a ***PHP***-fejlesztőket, mégpedig a ***PEAR*** rendszer segítségével. A ***PEAR*** egy olyan „kódgyűjtemény", amely a szabad szoftverek filozófiájának megfelelően lehetővé teszi, hogy a gyakorlott programozók közkinccsé tegyék saját, jól átgondolt, egy-egy speciális feladatot a leghatékonyabban ellátó kódrészleteiket. Az „utókornak" ezek után tulajdonképpen már csak az a dolga, hogy megtanulja ennek a modulrendszernek a használatát, és saját igényei szerint összeillessze az építőkockákat.
Koblinger Egmont, az ***UHU-Linux*** csapat egyik fejlesztőmérnökének cikke a szoftverfejlesztés egy még ennél is magasabb szintjét mutatja be: elmondja, milyen elvek alapján lehet összeállítani egy jól működő ***Linux*** terjesztést. A dolog apropója, hogy márciusban, az előző változatot követő egy éves fejlesztési időszak befejeztével az ***UHU-Linux Kft.*** kiadta az ***UHU-Linux Office 1.2***-es változatát, mely a ***Rajt!*** kódnevet viseli. Egmont természetesen aktív részese volt ennek a munkának, így azon kevesek egyike, akik egy a felhasználók által néhány kattintással feltelepíthető rendszert „belülről" is láttak.
Természetesen folytatódnak korábbi sorozataink is. ***Fábián Zoltán*** tovább folytatja az ***eFltk*** környezet, ***Garzó András*** pedig a számítógéphálózatok alapjainak bemutatását, ***Auth Gábor FreeBSD***-ről szóló sorozata pedig immár a nyolcadik részéhez érkezik.
Levezetésképpen Marcel Gagné bemutat nekünk néhány olyan játékot, amelyeket mintha már láttunk volna valahol, valamikor...

Hasznos időtöltést, jó szórakozást kíván a Linuxvilág stábja!

Az első Mandriva

Megjelent a *Mandrake-Conectiva* név után immár *Mandriva* név alatt futó terjesztés legújabb változata, a *Mandriva*

Linux Limited Edition 2005. A mostani kiadás csak átmeneti változat az utolsó, 10.1-es *Mandrake* és a *Mandriva 2006 Official* között, ettől függetlenül naprakészségére és tudására nem lehet panasz; a terjesztés a 2.6.11.6-os rendszermagra épül, 3.3.2-es *KDE*-t és 2.8.3-as *GNOME*-ot tartalmaz, valamint a 64 bites processzorokat is teljes mértékben támogatja. A *Mandriva Limited Edition 2005* a cég webáruházában 59 eurós áron vásárolható meg, ezért az összegért kettő darab *DVD*-lemezt kapunk. Ha a letölthető változattal is megelégszünk, 5,9 eurót takaríthatunk meg, de ha így is sokalljuk az árat, akkor az *FTP*-n keresztül letölthető, ingyenes, ám lecsupaszított kiadással kell beérnünk. A terjesztés i586, *PPC* gépekre és *Xboxra* érhető el.

➲ www.mandriva.com

Áttérés és átterelés

A *Resolvo Systems* bejelentette, hogy – egy *SourceForge* tervezet létrehozásával párhuzamosan megnyitja

MoveOver nevű, *Windowsról Linuxra* való áttérést segítő alkalmazásának kódját. A cég nem titkolt célja, hogy a nyitással megnyerje magának a közösség tagjait, újabb ötletekkel bővítse a programot, illetve felgyorsítsa annak fejlesztését. A *MoveOver* segítségével elvileg szakember segítsége nélkül elvégezhető egy számítógép átállítása *Windowsról Linuxra*, természetesen a beállítások, a dokumentumok és az elektronikus levelek megőrzésével. Ezzel egy időben a cég saját weblapján is létrehozott egy *Resolvo World* nevű részt, ahol a nyílt forrású közösség tagjainak áttéréssel kapcsolatos tapasztalatait, tanácsait szeretnék összegyűjteni, illetve cikkekkel szeretnék segíteni a területen még járatlanok munkáját.

➲ http://sourceforge.net/projects/openmoveover/

➲ www.resolvo.com

Kettős védelem

A *Paynacea Authentication Systems* bejelentette, hogy kétutas hitelesítési szolgáltatását ingyenesen teszi elérhetővé

az otthoni felhasználók és a mikrovállalkozások számára. A *Paynacea TeleAuth* rendszerének lényege, hogy a felhasználó hitelesítéséhez a megszokott bejelentkezésen túl egy másik csatornát is igénybe vesz, ez pedig a hagyományos, nyilvános telefonhálózat. Amikor a felhasználó be kíván jelentkezni, a szokásos felhasználónév-jelszó páros megadása után előre rögzített telefonszámán kap egy hívást a *Paynacea* központjából, ezt követően a bejelentkezés befejezéséhez meg kell adnia PIN-kódját. A szolgáltatás használatához külön hardvereszközre nincs szükség.

Természetesen az ingyenes szolgáltatás csak egy lebutított változata a *Paynacea* – egyébként havi 100 dollár körüli összegért elérhető – kereskedelmi szolgáltatásának, amelyet széles körben lehet alkalmazni a biztonság növelésére, a webkiszolgálóktól kezdve egészen a távmunkát segítő virtuális magánhálózatokig. A szolgáltatás fenntartásához szükséges infrastruktúrát teljes egészében a cég üzemelteti, a támogatott operációs rendszerek a *Unix*, a *Linux*, a *Mac OS X* és a *Windows*. A *TeleAuth* egyelőre csak néhány országból érhető el, de a megfelelő hívásdíjak kiszámítása után a cég bővítést ígér.

➲ www.paynacea.com

VMware Workstation 5

Hosszú ideje nem láthattunk már új *VMware Workstation* változatot, most azonban a cég elkészült virtualizációs

alkalmazásának legújabb, 5.0-s kiadásával. Az új *VMware* számos értékes új és továbbfejlesztett szolgáltatással rendelkezik, ezek közül néhány csemege:

– Csoportok alakítása virtuális gépekből. A csoporttagok azonos virtuális hálózati szegmensre helyezhetők, melynek sávszélessége és csomagvesztési aránya is megadható.
– Több pillanatkép készítése akár futó virtuális gépekről is, illetve pillanatképek visszaállítása.

Virtuálisgép-sablonok létrehozása. Filmfelvétel, amely például elektronikus tananyagok, bemutatók kialakításához használható.

Bővült a támogatott operációs rendszerek köre is, a *VMware* gazda operációs rendszerként a *Red Hat* és a *SuSE Linuxokat* teljes értékűen képes elfogadni, a 64 bites *Windows XP* és *Windows Server 2003* támogatása viszont egyelőre kísérleti jellegű. Vendég operációs rendszerként immár gond nélkül futtathatók az újabb *Linux* terjesztések is, ide értve a *Sun Java Desktop Systemét* is; ugyanakkor a *Solaris* továbbra is kimaradt a támogatott vendégek közül. Ugyancsak a linuxos rendszerek használhatóságát javítja a *GTK2* alapú felületek továbbfejlesztett támogatása.

Akik 2004. december 15. után vásárolták meg a *VMware Workstation* előző, 4.5-ös változatát, azok ingyenesen térhetnek át az 5-ös kiadásra, a többiek számára a frissítés 99 dollárba kerül.

➲ www.vmware.com

Lelakatolni mégse lehet

A *Cisco Systems* egy a vezeték nélküli hálózati ügyfelek helyzetének követésére alkalmas készülékkel bővítette kínálatát. A *Cisco* által nemrég felvásárolt *Airespace* cég fejlesztéseire épülő *Wireless Location Appliance 2700* a vállalati, szervezeti *WLAN*-on belül akár 1500 ügyfélkészülék helyzetét is képes nyomon követni, ehhez csak egy rádiófrekvenciás azonosítót igényel, illetve azt, hogy az egyes készülékek valóban élő hálózati kapcsolattal rendelkezzenek. A lehetséges ügyfelek között említették a kórházakat, amelyekben rendkívül nagy értékű műszerek, berendezések mozognak folyamatosan, és ezek követése nemcsak vagyonvédelmi okokból fontos, de olyan szempontból is, hogy vészhelyzet esetén könnyen és rövid idő alatt elő lehet őket keríteni. A készülék 14995 dolláros áron lesz megvásárolható, ami tulajdonképpen már egyetlen nagyobb értékű eszköz eltűnésének megakadályozásával is megtérülhet.

➲ www.cisco.com/en/US/products/ps6386/index.html

Mindent megfog

Új változat jelent meg a *Central Command Vexira Antivirus* termékéből. A sokféle operációs rendszer

(*Linux*, *Windows*, *NetWare*, *AIX*, *BSD*, *Solaris*) alá elérhető, levélkiszolgálókhoz készült csomag teljes értékű védelmet nyújt, ugyanis a vírusok mellett a levélszemét és a kémprogramok kiszűrésére is képes. A *Vexira* bármely meglévő kiszolgálóprogrammal együtt tud működni, de önálló *SMTP*-kiszolgálóként is használható, ezzel is fokozva a háttérben üzemelő rendszer védelmét. Szolgáltatásai széles körűek: rendelkezik *LDAP* támogatással, többféle támadás ellen is felkészítették, képes a levelek archiválására, működéséről pedig valós idejű statisztikákat szolgáltat üzemeltetőjének. Ára egyetlen tartományhoz, legfeljebb 6000 postaládához 60 ezer forint alól indul, míg a korlátlan változat körülbelül 750 ezer forintért vásárolható meg.
➲ www.centralcommand.com

Érkezőben az új Intel alaplapok

Japánban egyes üzletekben már felbukkantak az *Intel* következő, *945* és *955X* jelölést kapott lapkakészleteire épülő alaplapok. Szükség is lesz rájuk, hiszen az *Intel* legújabb, kétmagos processzorai kizárólag ezekben az alaplapokban lesznek használhatók. A *945P*, a *945G* és a *955X* lapkakészlet a 800 és az 1066 MHz-es előoldali buszfrekvenciával üzemelő processzorokat képes kezelni, amelyek mellé 667 MHz-es *DDR2* memóriát illeszthetünk. A *G* jelzésű lapkakészlet hagyományosan grafikus vezérlőt is kap, ez esetben ez a *GMA950* mag lesz. Megújul a be/kiviteli vezérlő is, kisebb-nagyobb fejlesztések mellett képes lesz a *SATA-II* merevlemezek kezelésére. A korábbi lapkakészletekhez hasonlóan az *Intel* most is az alacsonyabb sorszámú változatokat szánja a „köznépnek”, míg a 955-ös változat a profi felhasználóknak készül.

Mazsoláznak

Lassan a pulzárokhoz válik hasonlóvá a *Novell*, amely töretlenül szippantja magába a nyílt forráskódú világ legkülönfélébb elemeit, programokat, tervezeteket és embereket egyaránt. A kiválóan felépített stratégia legújabb áldozata a *Samba* egyik szerzője, *Jeremy Allison*, aki a *Novellnél* folytatja munkáját, továbbra is a cég számára rendkívül fontos fájlmegosztási szolgáltatások fejlesztésén munkálkodva. Amiért a *Novell* nem a fekete lyukakhoz hasonlatos, az az, hogy nemcsak elnyel, de ad is. Most például saját termékét, a *SilverStream* alkalmazáskiszolgálót áldozta fel a *Jboss Inc. Jboss Enterprise Middleware System (JEMS)* kedvéért, továbbá a nyílt forrású csoportmunka-kiszolgáló, a *Hula* kedvéért körülbelül kétszázezer sornyi kódot nyitott meg *NetMail* kiszolgálójának kódjából. A sor hamarosan folytatódik, a következő nyitás az *eDirectory LDAP* kiszolgáló kódját fogja érinteni. További érdekesség, hogy az *Allison* és a *Novell* közötti szerződés értelmében *Allison* a cégnél folytatott munkája eredményét szabadon hozzáférhetővé teheti mások számára.
➲ www.novell.com

Nyolcadik Opera

Az *Opera Software* bejelentette az *Opera* böngésző legújabb, 8.0-s változatát. A böngésző *Linux* és *Windows* alá végleges változatban jelent meg, a *Mac* alá készített kiadás egyelőre tesztelés alatt áll. Természetesen az újabb változat szebb, gyorsabb, jobb, mint az előző, a javítások a sebességet, a biztonságot és a használat könnyűségét egyaránt érintik. Az új *Opera* immár hanggal is vezérelhető, igaz, ez a lehetőség egyelőre csak *Windows* alatt érhető el; egy másik kényelmi szolgáltatás, az egérmozdulatokkal történő vezérlés viszont minden változatban megtalálható. Az *Opera 8* ingyenesen letölthető változatáért használója egy reklámcsík formájában fizet, míg a hirdetésmentes változatért 39 dolláros regisztrációs díjat kell fizetni.
➲ www.opera.com

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

Mi újság a rendszermag fejlesztése körül?

Wichert Akkerman, a *Debian Project* korábbi vezetője különös jelenségekre lett figyelmes a *2.6.10-ac10* rendszermag használata közben. A *df* parancs azt jelezte, hogy a lemezhasználat a következő: -7378697629483812773 6. Mivel hibára gyanakodott, küldött egy levelet a *linux-kernel* levelezési listára, ám miközben a többiek különféle ötletekkel álltak elő a hiba okát illetően, *Wichert* az *e2fsck*-val kijavította a hibát, így egyik felvetést sem tudta érdemben megvizsgálni. Időnként találkozni ilyen furcsaságokkal, magyarázat azonban ritkán akad rájuk. Lehet, hogy a rendszermag egy későbbi változatában újra láthatunk majd ilyesmit, és akkor az okot is sikerül majd kideríteni, de az is lehet, hogy apró, múló hardverhiba történt.

Mitch Williams rájött, hogy a *SysFS* által kezelt fájlokhoz nem lehet adatokat hozzáfűzni. Minden ilyen kísérlet a régi adatok újakkal való felülírását eredményezi. Az is ugyanilyen eredménnyel jár, ha a fájl megnyitása után és az írás megkezdése előtt a végére léptetünk. *Greg Kroah-Hartman* megerősítette, hogy bizony nem teljesen ilyen viselkedést vártak a rendszertől, különösen, ha azt nézzük, hogy a *SysFS* mindenféle hibajelzés nélkül írja felül az adatokat. *Mitch* mindkét hibára készített egy foltot, majd vitázgatott egy kicsit *Greggel* a változatoknak a folt miatti szaporodása miatt, de lényeg az, hogy a *SysFS* működése hamarosan módosul, és minden fájlművelet az elvárható módon fog folyni.

A *security@kernel.org* cím alatt új levelezési lista jött létre. A lista célja a biztonsági hiányosságok szélesebb körű közzétételük előtti ismertetése, aminek köszönhetően a *Linux*-fejlesztők időt kapnak a javítások elkészítésére és elérhetővé tételére, még mielőtt az erre hajlamos egyének kidolgozhatnák támadásaikat. A lista fontos jellemzője, hogy a feliratkozás meghívásos alapon történik, az archívum pedig – a *linux-kernel* listával ellentétben – nem válik azonnal elérhetővé. *Linus Torvalds*, aki inkább a teljesen nyílt fejlesztéseket kedveli, maga is csatlakozott a listához, bízva abban, hogy a többek közt *Marcelo Tosatti* által is támogatott, nyilvánosságtól elzárt hibakezelés idővel mégiscsak jó ötletnek bizonyul. Végül is, ezt is ki kell próbálni. Persze a kérdéskör szükségszerűen vitákhoz vezet, főként a nyílt forrású fejlesztési modell elhivatott támogatói körében.

Jake Moilanen egy genetikus algoritmust készített a rendszermag beviteli/kiviteli ütemezőjének és folyamatütemezőjének javítására. Ezeket az ütemezőket – különösen a folyamatütemezőt – hagyományosan nehéz jól behangolni, ugyanis a felhasználók viselkedése rendkívül sokféle lehet. Honnan tudhatnák a fejlesztők, hogy egy adott algoritmus mindenféle felhasználói viselkedés mellett kifogástalanul fog-e működni? Nyilván ezt nem lehet előre megígérni. *Jake* munkája azonban, amennyiben valóban eredményekkel jár, egészen újfajta utat mutathat a rendszermag beállításainak finomhangolásában. Ugyanakkor a genetikus algoritmusok működésének eredménye megjósolhatatlan, márpedig a megjósolhatatlanság a rendszermagban nemkívánatos. Valószínű, hogy a fejlesztők az ilyen jellegű foltokat csak akkor fogják átvenni, ha kiterjedt és jól mérhető teljesítményjavuláshoz vezetnek; de lehetséges, hogy még akkor is csak a genetikus finomhangolás eredményeit fogjuk viszontlátni, magát az algoritmust nem. Meglátjuk.

A szoftveres felfüggesztés története folytatódik, *Pavel Machek* nemrég *SMP* gépekhez is elérhetővé tette az *swsusp* szolgáltatást. Eddig ennek támogatása megoldatlan volt, a 2.6.11-es rendszermaggal kezdve azonban már *SMP* rendszereken is rendelkezésre áll a szoftveres felfüggesztés lehetősége. Az *swsusp* kód szép apránként fejlődget, és a korábbi időkben az egymással versengő kódok miatt látott viták és panaszok remélhetőleg csak rossz emlékek maradnak. Persze a szoftveres felfüggesztés továbbra is veszélyes mutatvány marad, bizonyos hardverelemek ugyanis egyszerűen nem hajlandók együttműködni. Ilyenkor a viták gyakorlatilag elkerülhetetlenek, és mivel nagyon nehéz az egyes kérdések létező legjobb megoldását megtalálni, sokszor lelombozó, ha le kell mondani egyes dolgokról. Az az egy biztos, hogy az swsusp ígéretes fejlesztésnek tűnik.

A közelmúltban az új és a meglévő rendszermagok körül komoly karbantartói tevékenységet láthattunk. *Andrew Vasquez* lett a *Qlogic QLA2XXX FC-SCSI* illesztőprogram hivatalos gondozója. *Tony Luck* átvette *David Mosberger-Tangtól* az *IA-64* vonatkozású karbantartást. *Matthias Kunze* gondjaiba vette az utóbbi időben eléggé elhanyagolt *Enhanced Linux Progress Patchet*, és átültette a 2.6.10-es rendszermag alá. *Andries Brouwer* után *Adrian Bunk* lett a *util-linux* tervezet gazdája, miután még 2004 szeptemberében *Andries* új segítőket keresett.

Szintén a karbantartásokkal kapcsolatos, hogy a jövőben a *MAINTAINERS* fájlban meg fogják jelölni azokat a levelezési listákat, amelyek csak a feliratkozottaktól fogadják a leveleket. A linuxos fejlesztői listák hagyományosan nyitottak, ezzel is segítik a felhasználókat a hibajelentések leadásában, ám a rendszermaggal kapcsolatos tervezetek résztvevői nem mindig fogadják el ezt a szemléletet. Azok számára, akik inkább a zártságot pártolják, többek közt *Domen Puncer* közreműködésével készült egy a csak a feliratkozottak számára elérhető listákat azonosító folt. *Domen* korábban éppen ez ügy kapcsán megpróbálta kivenni a *MAINTAINERS* fájlból például a *linux-arm-kernel* listát, ám *Alan Cox* és mások ellenezték ezt, így inkább a megjelöléses megoldás maradt.

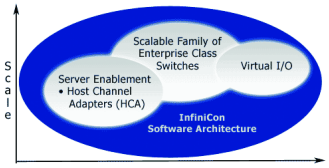
Zack Brown

Linux Journal 2005. május, 133. szám

Új termékek

InfiniCon Software Release 3.0

Az *InfiniCon Systems* megjelentette *InfiniBand* alapú hardver- és szoftverplatformjának 3.0-s



változatát. Az alaprendszer a gazdaprogramokat, a kapcsolókba ágyazott programokat és az *InfiniCon FastFabric* eszközét foglalja magába, és a rendszer megnyitásával külső eszközök és alkalmazások használatát is lehetővé teszi. A 3.0-s szoftverkiadás *InfiniBand* csatolót tartalmazó alaplapokra épülő kiszolgálókon alkalmazható, segítségével állomás csatornacsatoló nélkül is el lehet érni az *InfiniBand* hálózatot. A 3.0-s változat támogatja a 2.6-os *Linuxot*, akár 1000 csomópontig is képes méreteződni, *Oracle* tanúsítványa mellett más kereskedelmi *MPI* csomagok tanúsítványaival is rendelkezik, megbízhatósága és teljesítménye javult, valamint bővítették a *FastFabric* eszközök felügyeleti képességeit is.
www.infinicon.com

NAG Fortran Library Mark 21

A *NAG Fortran Library Mark 21*-e több mint 300 új függvényt tartalmaz, amivel függvényeinek száma 1500-ra növekedett. Az új függvények között egy teljesen új, 2D hálók előállítására használható készletet találunk, amelyet nagy számú segédfüggvény egészít ki. Az új bővítmények polinomok nullahelyeinek megkeresésére, parciális differenciálegyenletek leírására, sajátérték-keresésre (*LAPACK*) és ritkás lineáris algebra problémák kezelésére használhatók. A véletlenszám-előállító (G05) függvény kibővült, új számelőállítót tartalmaz, támogatja az egyváltozós *GARCH*, az aszimmetrikus *GARCH* és *EGARCH* folyamatokat, a kvázivéletlen számok előállítását, illetve egyéb eloszlásokat. A *NAG Fortran Library* többféle megvalósításban érhető el, ezek a géptípusokat a *PC*-ktől kezdve egészen a szuperszámítógépekig lefedik. Használata nem korlátozódik egyetlen környezetre, algoritmusai számos más nyelvből, köztük *C++*-ból is meghívhatók.
www.nag.com

Platform for Network Equipment, Linux Edition

A *Wind River Systems* bejelentette a *Platform for Network Equipment (NE), Linux Edition* elérhetőségét. A *Platform NE* támogatja a *Carrier Grade Linux 2.0* előírásokat és a 2.6-os *Linux* rendszermag eszközszoftverek fejlesztését segítő megoldásait. Segítségével *ATCA* alapú, a kereskedelemben készen megvásárolható megoldások alkalmazhatók a távközlési szintű hálózati berendezések felügyeletére és vezérlésére. Emellett a *Platform NE* számos külső gyártó eszközeihez, programjaihoz biztosít hozzáférést, míg a teljes fejlesztési ciklus támogatását az *Eclipse* alapú *Wind River Workbench IDE* segíti.
windriver.com

CM4000 konzolkiszolgáló

Új konzolkiszolgáló családot dobott piacra az *Opengear, Inc.* A *CM4000* soros konzolkiszolgáló

8, 16 és 48 kapus változatban kapható, *Windows*, *Sun* és *Linux* alapú kiszolgálók soros konzoljához képes hozzáférést biztosítani. A *Opengear CM4000* sorozatú készülékei hálózati készülékek – útválasztók, átjárók, telefonközpontok és áramátkapcsolók – figyelésére és kezelésére is alkalmasak. A távoli telephelyek kiszolgálóinak elérése sávon belül, a vállalati *TCP/IP*-hálózaton vagy modemen keresztül lehetséges, az adatokat mindkét esetben legfeljebb 128 bites *AES* titkosítás védi. Az *Opengear CM4000* konzolkiszolgáló szűrési és hozzáférés-naplózási képességekkel is rendelkezik, ami fontos segítséget jelent a konzolnaplók archiválásához. A *CM4000* készülékek az *okvm* nyílt forrású konzolra és *KVM* kezelőprogramra, illetve nyílt forrású *KVM* hardverre épülnek. Webböngészős és parancssoros felügyeleti lehetőségeket egyaránt kínálnak.
www.opengear.com

IBM eServer Application Server Advantage for Linux

Az *IBM eServer Application Server Advantage for Linux*, más néven *Chiphopper* egyesíti mindazokat a támogatási és teszteszközöket, amelyek alkalmazásával a független szoftvergyártók többféle géptípuson is futtatható Linuxos termékeket fejleszthetnek. *Chiphopper* ingyenes, segítségével meglévő *x86* (*Intel* vagy *AMD*) processzoron futó linuxos alkalmazásokat lehet átültetni tetszőleges *IBM* rendszerre, illetve alkalmas ezek tesztelésére és támogatására is. A *Chiphopper* a kifejezetten az operációs rendszerre írt alkalmazásokat és a middleware alapúakat egyaránt támogatja. Előbbiek esetében a *Chiphopper* a hordozhatóságot a *Linux Standard Base (LSB)* előírásokra alapozza. Emellett a *Chiphopper* azokat az *LSB* alkalmazásokat is támogatja, amelyek nyílt bővítményeket használnak, mint például az *OpenLDAP*, az *OpenSSL*, a *Kerberos*, a *PHP*, a *Perl* és a *Python*. A middleware alapú alkalmazások esetében a *Chiphopper* az *IBM WebSphere*, *DB2* és *Rational* csomagját ismeri; *Java*-, *J2EE*- és webszolgáltatás-támogatást biztosít, valamint támogatja a szolgáltatásorientált, nyílt szabványokra alapuló megoldásokat.
www-1.ibm.com/linux

Linux kis műholdakon

A műhold tervezésére és elkészítésére két évnél is kevesebb idő volt, így a csapat már meglévő érzékelőket, ipari-szabvány alkatrészeket, héjprogramokat és kedvenc operációs rendszerünket használta a projekt összehozásához.

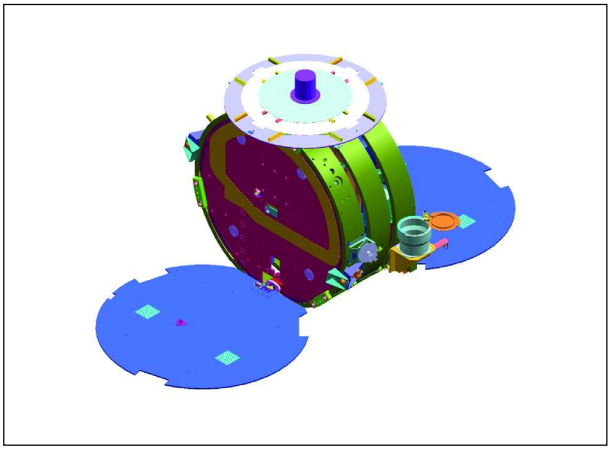
A *Védelmi Minisztérium (DoD) Force Transformation Osztálya (OFT)* egy 100 kilogrammos osztályba sorolt mikro-műhold üzembe helyezésének lehetőségével kereste fel a *Tengerészeti Kutatólaboratóriumot (NRL)*, amelyen technológiai és műveleti kutatásoknak biztosítanának helyet. Az *OFT* legnagyobb kihívást jelentő feltétele a laboratórium felé az volt, hogy mindezt egy évnél kevesebb idő alatt kell véghezvinni. A *TacSat* elképzelésünk sikeréhez új kapcsolatokat és módszereket kellett kiépítenünk valamint figyelembe kellett vennünk a meglévő alkatrészeket, programokat és adottságokat.

A *Copperfield-2* érzékelőrendszer, amit a szerző csapata fejlesztett a haditengerészetnek, vált a *TacSat-1* rakomány infrastruktúrájának sarokkövévé. A *Copperfield-2* érzékelőrendszert (*2. ábra*) eredetileg ember nélküli légijárművekhez (*UAV*) terveztük – kiváló alap egy űrbéli küldetéshez, hiszen sok tervezési követelmény azonos.

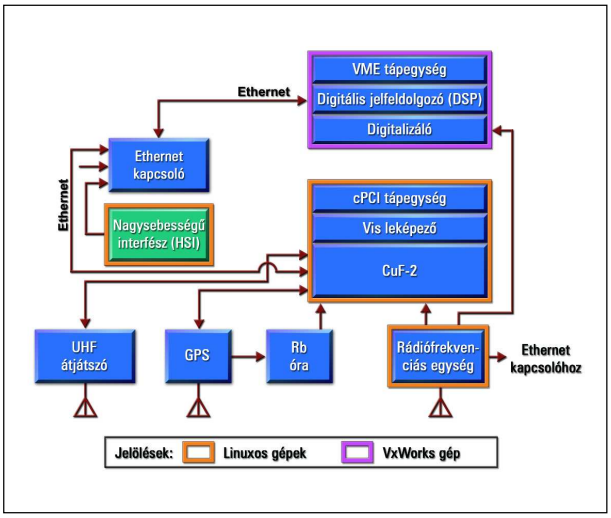
A műhold buszra tekinthetünk űrjárműként. Ez biztosítja a rakomány számára a működéshez szükséges fizikai és elektromos környezetet. A műhold rakománya a buszon szállított érzékelő vagy kísérleti eszköz. A *TacSat-1*-ben használt buszt eredetileg az *ORBCOMM* használta kis méretű kommunikációs műholdakhoz. Ha a *Copperfield-2* repülőn vagy *UAV* gépen üzemelne, akkor az szolgálna buszként, biztosítva a rakomány működéséhez szükséges környezetet.

Moduláris rakomány alkatrészterv

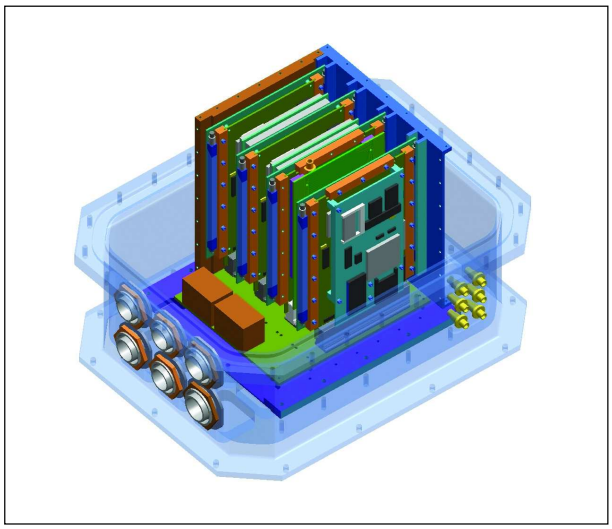
A *Copperfield* rakomány első verzióját örökölt alkatrészekből készítettük el. Ezeket átalakítottunk, hogy az eredeti alkatrészek *Ethernet*-alapú *TCP/IP* csatolófelületen kapcsolódhassanak össze. A második generációs kísérleti képességek tervezése előtti beszerzés során figyelembe vettük a különféle busz szabványokat, a készen kapható üzleti megoldások képességeit és egyéb tényezőket. Úgy döntöttünk, egy *3U CompactPCI* rendszert vásárolunk amely fizikai kialakítás tekintetében a lehető legnagyobb rugalmasságot teszi lehetővé (*3. ábra*). Ugyanakkor saját *PCI* alaplapot használtunk, így a *CompactPCI* felhasználó által megadható *P2* kapcsoló tűit saját céljainkra használhattuk fel. Eredményképpen egy olyan alaplapot kaptunk, amelynek foglalataiba el tudtuk helyezni saját készítésű alkatrészeinket, és vannak készen kapott *Ethernet* kártya befogadására alkalmas foglalatai illetve a *PXI* szabványhoz illeszkedő foglalatai is.



1. ábra TacSat-1 üreszköz, telepített napelemekkel, Nadir (föld-oldali) oldalával felfele



2. ábra TacSat-1 Copperfield-2 Rakomány blokkábrája



3. ábra TacSat-1 Copperfield-2 CompactPCI Cardset és foglalata

A kész rendszer érdekes keverék lett: ***Ethernet*** csatlakozással ellátott ***CompactPCI*** amely ***P2*** hátlapon nyugszik.

Moduláris szabvány alapú rakomány szerkezet

Kevés más program rendelkezik olyan mozgástérrel illetve vállalhat akkora kockázatot mint amilyet a ***TacSat-1*** kísérlet jelent. A ***TacSat-1*** kísérlet két területen is újítónak számít, egyrészt kormányzati készen kapott (***GOTS***) és ***COTS*** alkatrészeket alkalmaz, másrészt a szokatlan megközelítéssel készíti el a rakomány programját amely maximális rugalmasságot és szabvány alapú műveleteket nyújt. Ez a kockázatvállaló filozófia tette lehetővé a moduláris alkatrész felépítést. A ***TacSat-1*** számára hasonló módon bővítettük ki a moduláris program és kommunikációs rendszert, a szabványos nyílt forrású programok szerepének ilyen kibővítése újrahasznosítható programhátteret biztosít számunkra amit ***TacSat-1*** rakományának rugalmas vezérlésére és irányítására használhatunk fel.

A ***Copperfield-2*** rakományszerkezetét úgy készítettük, hogy a lehető legnagyobb rugalmasságot nyújtsa. Mi sem bizonyítja jobban a szerkezet rugalmasságát mint, hogy egy ***UAV*** rakományát át tudtuk alakítani űreszközre való alkalmazássá. Mivel a rakomány program alkotói nem űrrepülés-kritikusak, az űrjármű biztonsága és épsége nem függ azok megbízhatóságától, a programok nagy része pedig légi és űrrendszereken egyaránt használható.

Linux rendszermag mint alap

A ***Copperfield-2*** fejlesztésének kezdetétől szerettünk volna a ***Linux*** forráskód lendületére, képességeire és elérhetőségére alapozni. A ***PowerPC PowerQuicc II*** processzorkártya jóvoltából megoldottnak tekinthettük a robusztus beágyazott rendszer alkatrészigényét. A forráskód elérhetősége volt számunkra az egyik legfontosabb jellemző, hiszen lehetővé tette, hogy kezeljük a lehetséges problémákat, például a lap kiépítési hibáit. Bár a lap tervezése nagyon hasonlított a ***Motorola design-MPC8620ADS-PCI*** mintára, amely bizonyos félreérthetőségek miatt ma már nem elérhető, alkatrész korlátok és más problémák miatt kénytelenek voltunk változásokat eszközölni a rendszermagban.

A ***TacSat-1*** fejlesztésének indulásakor több harcedzett veterán kétkedésének adott hangot a rakomány vezérlőprogramjának otthont adó ***Linux*** miatt. Az ***NRL***-nél fejlesztett űrrendszerek esetében általában üzleti valós idejű operációs rendszereket alkalmaznak. A szerkezeti tervezés során nem találkoztunk komoly valós idejű követelményekkel, ez még inkább megerősítette eredeti elképzelésünket, miszerint ***Linuxot*** használunk a ***Copperfield-2*** és így a ***TacSat-1*** alapjául.

Néhány módosításon kívül, amelyekre a ***Linux*** működésre bírásához volt szükség a rendszerünkön, mindössze három meghajtót készítettünk: az egyik az érzékelő adatformátumát kezelte; a második a ***Xilinx SystemAce*** csatolófelülete volt (ezt a ***CompactFlash*** csatolófelületet használhattuk az ***FPGA***-k betöltésére és az ***OS*** tárolójaként; végül a harmadik a ***PowerPC 823 HSI*** csatolófelület doboza amely az ***FPGA***-val tartja a kapcsolatot. Mivel a ***PowerPC*** processzorunk memóriaterébe nagy méretű ***Xilinx Virtex-II*** részt lapoztunk be, az ***FPGA*** tervek megváltoztatása helyett néhány újítást vezettünk be az eszközmeghajtó fejlesztésben. ***Don Kremer*** az ***Aeronixtől*** kifejlesztett egy eszközkészletet amely képes ***Verilog*** forrásfájlokat olvasni és számtalan makrót, ***C*** kódot sőt még ***HTML*** dokumentációt is készíteni, így lényegében ***Verilog*** alkatrész-specifikációk alapján el tudtuk készíteni a meghajtók nagy részét.

A COTS processzorok hálózati szerkezete

A ***Copperfield-2*** rakomány magja két fő funkciót látott el a küldetésben. Elsősorban egy érzékelőrendszerről van szó amely fogadja a begyűjtött adatokat, feldolgozza azokat és kapcsolatot teremt a rendszer kommunikációs eszközével, hogy az eredményeket más érzékelőknek és földi állomásoknak küldje tovább. Másodsorban, általános célú számítógépes rendszerként működik amely a tároló és adatkezelés hátterét biztosítja. Valójában a ***Copperfield-2*** rakománya közt több általános célú processzort is találunk, amelyek ***Ethernet*** hálózaton keresztül tartják egymással a kapcsolatot. A csillag felépítésű ***Ethernet*** szerkezet központját a ***COTS Ethernet*** kapcsolóját (switch) találjuk.

Átjáró az örökölt busz eszközhöz

Az ***UAV*** rakomány ***Ethernet TCP/IP*** szabvány alapú szerkezetét úgy szerettük volna kihasználni, hogy közben együttműködők maradunk a műhold buszának örökölt ***OX.25*** csatolófelületével (ez biztosítja a tudományos adatok és a egészségi állapot telemetria adatok jellevételét) ezért egy másik beágyazott számítógépes modult is készítettünk, amelynek egyetlen feladata a híd biztosítása volt. Ezt a modult nagy sebességű csatolófelületnek neveztük (***HSI***) amely az űrjármű kommunikációs vezérlőegységéhez kapcsolt 2MB-os szinkron soros buszt tette elérhetővé. A ***HSI*** alkatrész ***FPGA*** alkatrész és ***BSE ipEngine*** általános célú ***PowerPC 823*** beágyazott processzor keverékéből állt.

A ***HSI***-ben az ***FPGA*** biztosította az adatkapcsolathoz szükséges időzítési követelményeket, leválasztva a processzort a szinkron adatkapcsolatról. A ***PowerPC Linux 2.4***-alapú rendszermagot futtatott, ahol a ***HSI FPGA*** csatolófelületet szabványos ***Linux*** eszközmeghajtóként készítettük el. Nem használtunk semmilyen különleges valós-idejű kiterjesztést,

1. táblázat *A TacSat-1 Copperfield-2 Ethernet-kapcsolatú beágyazott rendszer*

Alkatrész	Gyártó	OS	Processzor
Nagy sebességű csatolófelület (HSI)	Bright Star Engineering (módosított lap)	Módosított Linux 2.4 rendszer	PowerPC MPC823
IDM UHF Modem	Innovative Concepts	Üzleti	PowerPC 860
Copperfield-2 MR. DIG Kártya	Aeronix/NRL	Módosított Linux 2.4 rendszer (DENX ELDK-alapú)	PowerPC PowerQuicc II 8260
RF Front End vezérlő	Bright Star Engineering (módosított lap)	Módosított Linux 2.4 rendszer	StrongARM SA1110

a szabványos protokollt használó ***TCP/IP*** hálózati verem illetve az eszközmeghajtó megoldásunk közötti kapcsolatot pedig ***Linux***-alapú program biztosítja. A ***HSI*** rendszer lehetővé teszi, hogy egyszerre több folyamat és ***Ethernet***-kapcsolatú számítógép érje el az űreszköznek küldött adatfolyamot. A ***Copperfield-2*** processzorán futó ***PowerPC*** kommunikációs vezérlő könnyedén el tudta volna látni a TacSat-1 HSI feladatait. Azonban a rendkívül korlátozott alkatrész elérhetőség miatt illetve a párhuzamos fejlesztési lehetőségek bővítése érdekében ezt a csatolófelületet is külön fejlesztettük.

Gyors rakományprogram-fejlesztés meglévő eszközökkel

Általában minden műhold program leginkább „egyedi" része a rakományvezérlő program. Minthogy a ***Copperfield-2*** rakományának legtöbb processzorral rendelkező része ***Linuxot*** futtatott, érdekes programozási lehetőség merült fel. A rakomány programjának nagy része ***bash*** (***Bourne Again Shell***) héjprogramként készült. A rakomány gyors fejlesztése során az volt az alapelvünk, hogy a programfejlesztést két részre, saját és újrahasznosított programmodulokra osztjuk fel. Ezt az elgondolást annak érdekében vezettük be, hogy a saját programozási munkánkat néhány függvény és speciális célú program készítésére csökkentsük. Esetenként úgy találtuk, hogy a meglévő eszközök nem igazán teljesítik az igényeinket. Ezeket módosítottuk és saját verzióval helyettesítettük.

Ezek a különleges célú programok és meghajtók apró parancssoros felületen keresztül vezérelhették a rakomány elemeit, így korlátozott képességeiket teljes körűen ki lehetett próbálni és tesztelni.

A programokat a ***UNIX*** parancssoros képességeket szem előtt tartva készítettük el, ideértve a szabványos bemeneten (***STDIN***) beérkező adatfolyamokat illetve a szabványos kimeneten kiküldött (***STDOUT***) kimenetet. Ezeket az elveket szem előtt tartó csatolófelülettel ellátott programeszközök fejlesztése a kezdeti ***UNIX***-os időktől kezdve rengeteg operációs rendszerben szabványnak számít. Ezt a stratégiát szerettük volna mi is folytatni és erre akarunk építkezni, hiszen rendkívül rugalmas lehetőség, amellyel komoly képességeket hozhatunk létre, egyszerű de mégis hatékony eszközökkel.

GNU és nyílt forrású eszközök

A programszerkezet tervezése során az első lépés annak megvizsgálása volt, hogy milyen kész eszközök állnak a fejlesztők rendelkezésére. Esetünkben a ***Linux*** terjesztés és az igen jó ajánlásokkal rendelkező ***GNU*** és nyílt forrású eszközök biztosították a szükséges funkciókat. Időről időre, ahogy a rakomány programját fejlesztettük, újra lenyűgözött bennünket az a rugalmasság és rendkívüli mennyiségű lehetőség amit a különféle parancsok nyújtottak.

Az egyik példa a ***GNU gzip*** tömörítőprogramja. Az földi kapcsolat kialakításakor a rakomány adatokat vezet át több program csövön valós időben. A ***flash*** fájlrendszeren található kiindulási állomány több eszközön megy keresztül, többek közt tömörítési állomásokon és a műhold buszán. Mint kiderült, kicsit hangolnunk kellett a ***gzip*** tömörítési/teljesítmény arányán, hogy az 1MB-os jellevételünket teljesen kitölthessük adatcsomagokkal. A jellevételbe szúrt gzip viszonylag új megoldás volt és segítségével teljes

1. lista tar, gzip és netcat eszközöket felvonultató jellevételi csővezeték

```
# fájlletöltési csővezeték beállítása
tar -cf - ${downloadFileList} | gzip -c -l | \
    file_downloader -tqid ${target_qid} -rlp \
                          ${return_link_path} \
                    -dri ${dump_request_id} \
                    -fmt ${dataFormat} | \
    netcat localhost ${!returnLinkService}
```

2. lista érzékelő adatfolyam feldolgozó csővezeték

```
# adatfeldolgozó csővezeték indítása
# (with cpf ignoring SIGINT,SIGTERM)
   eval "cat $dig_data_stream | \
      tee $raw_file | \
      cpf -i -v$cpf_verbosity $cpfparams \
      > $output_file &"
   # dig csatorna engedélyezése
   set_hardware 'echo $dig_channel \
     channelEnable ena | mapper 2>&1'
```

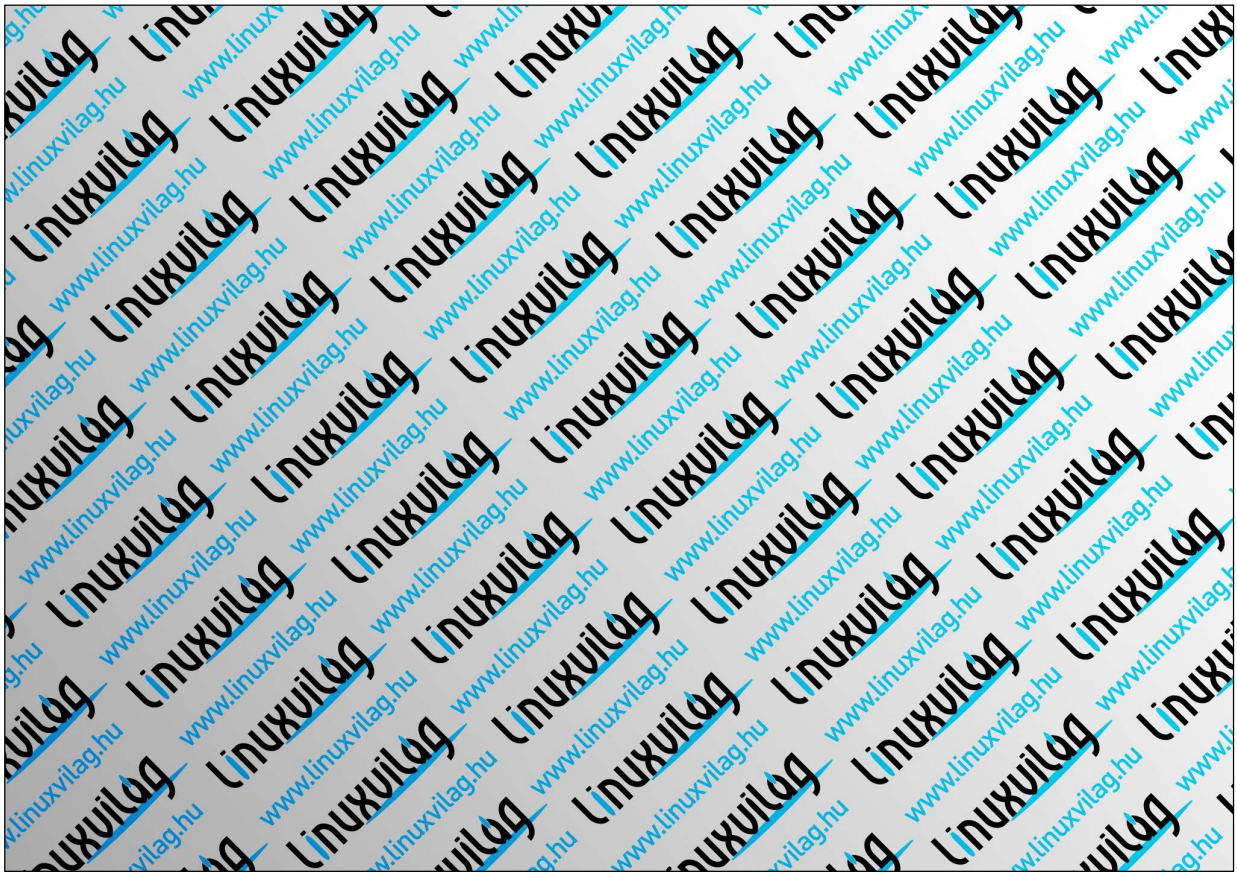
szélességében ki tudtuk használni a lefele irányuló kapcsolatunkat. A parancssor és STDIN/STDOUT csatolófelületeken alapuló felépítés lehetővé tette, hogy az ilyen típusú képességeket, számítási rendszerünk teljesítményének határain belül, átlátszóan szúrjuk be az adatfolyamba.

Rakományvezérlő alrendszer bash alapokon

Parancsfájlkezelő nyelv kiválasztása nem könnyű feladat. A nyílt forrású világban igazán sok alkalmas megoldást találunk. A *Perl* talán jó választás lett volna, de nem nagyon tetszett nekünk a telepítési mérete és memóriaigénye. A *Python* szintén jó választás tűnt, de azzal meg a fejlesztőcsapatnak nem volt tapasztalta. A leghatékonyabb programozható héj nyelvnek a *bash* tűnt, igaz egyben ez volt a legmemória-igényesebb is. A legkisebb beágyazott rendszerünk nem is bírta volna el a *bash* teljes méretét, azonban a *Busybox* könnyűsúlyú héjértelmezője, az *ASH*, majdnem ugyanolyan hatékonynak bizonyult az apró célon előforduló feladatok vezérlésében és megfigyelésében.
A rakomány vezérlő programterv teljes szerkezeti megvitatásához sajnos nincs elegendő helyünk, röviden összefoglalva a program magját a rakomány különféle funkcióit támogató *bash* programok alkotják. A rendszer a *POSIX*-stílusú fájlrendszer biztonsági megoldásait használja ki. Rendszerindításkor az első folyamat root jogon indul. Ahogy a rakománykezelő program üzembe lép *BOOT* felhasználóként kezd el működni. A rendszer *BOOT* jogon képes ellátni néhány kritikus rendszerfeladatot, például biztosítani a bináris telemetria folyamokat, fájlátvitelt és közvetlen parancsokat. Amikor az érzékelő feladatokra kerül a sor, a rendszer átlép *TRANSITION* állapotba végül minden további adatgyűjtés *OPS* felhasználóként történik, akinek más jogosultságai vannak. Az adatgyűjtés végén

3. lista Adat kimenetei csővezeték minta az adatkiküldés előtti utolsó lépében elvégzett védett adatformátumra alakítással

```
# Start the pipeline
format_event -severity $severity_level \
             -status $status_code \
     -failcmd $fail_cmd \
                -text "${event_text}"  \
                -debug $debug_level  2>>
                ➥$logFile \
    | ox25  -tbox  ${tbox}  -tque    ${tque}   \
            -sbox  ${sbox}  -sque    ${sque}   \
             -cflgs ${cflgs} -seq     ${seq}   \
                -func  ${func}  -subfunc
                ➥${subfunc} \
                -debug ${debug_level} \
                2>> $logFile  \
        | netcat $ncVerbose localhost
        ➥${!returnLinkService} \
                2>> $logFile
```



az *OPS*-t leállásra utasítjuk. A *BOOT* könyvtárakból több redundáns változatot is terveztünk a rendszerbe így fájlrendszer meghibásodás vagy más komoly probléma esetén van tartalék változatunk.
Minden rakományvezérlő rendszerfunkciót ***bash*** parancsfájlok indítanak. Ezek készítik a beállítások, idő, dátum és egyéb információkat követésére használt összetett fájlneveket. Segítségükkel kicsomagoljuk a műholdra felküldött parancsokat és állományokat. A parancsok maguk is egyszerűsített képességekkel rendelkező ***bash*** héjprogramok. Ezek más ***bash*** programokat hívnak meg amelyek a tényleges adatgyűjtést végzik vagy környezeti változókat állítanak be amelyek más parancsfájlok futását befolyásolják.
A ***bash*** parancsnyelv, a ***GNU*** és nyílt forrású eszközök és a saját készítésű parancssoros alkalmazások ilyen kombinációja egyedülállónak számít a műholdprogramokban.
A ***TacSat-1*** esetében a legnagyobb saját készítésű kód feladata az érzékelők adatainak kezelése volt, ugyanis a ***TCP/IP*** világ adatait át kellett alakítani a védett ***OX.25*** formátumra.

Osztott fejlesztés és együttműködés

A ***TCP/IP***-alapú rendszerek és az egységes ***Linux*** operációs rendszer egyedülálló lehetőséget biztosított a megoszló fejlesztési környezethez. A ***TacSat-1*** fejlesztésének kezdetén, saját ***PowerPC 8260*** fejlesztői eszközünk elérhetősége erősen korlátozott volt. A rakomány programok legtöbbjének tervezési ciklusa ***Intel x86***-alapú számítógépeken indult meg, amit később átvittünk általános ***PowerPC*** beágyazott processzorok alá, míg végül átkerültek a végső rendszerre.
A programtervező csapat fizikailag szétszórtan dolgozott és egy ***virtuális belső hálózat (VPN)*** kötötte össze őket. A távoli energiavezérlő eszközök tették lehetővé a külső munkahelyen dolgozó fejlesztőknek, hogy az alkatrészek áramellátást ki- és bekapcsolják. A létfontosságú ***kommunikációs és kapcsolattartó vezérlési dokumentációk (ICD-k)*** terjesztését és küldését a webalapú együttműködési eszköz tette lehetővé. Néhány fejlesztő ***azonnali üzenetküldő (instant messaging)***technológiát is alkalmazott hogy kapcsolatot tarthasson a többiekkel. Az együttműködési munkakörnyezetbe nem rég került bele a megtanult leckéket hálózati adatbázisban nyilvántartó ***E-Log***. Szeretnénk a ***Bugzilla*** képességeit is beépíteni a rendszerbe a viszonylag kidolgozatlan ***Message Forum***-alapú ***probléma jelentő (PR)*** követőeszközünk kiváltására.
A rakomány adathálózat ***TCP/IP*** jellege lehetővé tette a fejlesztőknek, hogy a kommunikációt a szabványos ***PC***-n történő fejlesztéstől kezdve a végső kommunikációig a tervezés minden egyes lépése során kipróbálják, mielőtt beilleszthettük volna a busszal kapcsolatot tartó speciális alkatrészt. az ***Ethernet*** teszt kapu még azután is hálózati elérést biztosított a műholdhoz miután a rakományt teljes egészében a buszba építettük. Ez a rendszer kollektív hibakeresésekor felbecsülhetetlen segítségnek bizonyult. A teszt kapuk hozzáférést biztosítottak a legtöbb rakomány elem soros konzolához, illetve néhány esetben a ***JTAG*** vagy más alkatrész hibakereső kapukhoz is.
A rakomány programtervező csapatban találhattunk tapasztalt műhold és földi állomástervező szakembereket, csakúgy mint a ***TCP/IP*** adatforgalom és ***Web/CGI*** alkalmazás fejlesztésben jártas tervezőket illetve beágyazott-rendszer szakértőket. Bár ez az összetétel merőben eltért a szokásos műhold programtervező csapattól, viszont közel tökéletes egyensúlyt biztosított a tudás és a újító módszerek között így a lehető legtöbbet lehetett kihozni az eredetileg légi járművekhez tervezett programokból. A kiterjedt távoli együttműködés, a csatolófelület tesztelés és a hálózati képességek lehetővé tették a zökkenőmentes busz-rakomány összeépítést.
A rakományvezérlő program magja, ideértve a legtöbb parancsot és vezérlőhéjprogramot, kevesebb mint négy hónap alatt készült el, elejétől a végéig. Amikor újabb érzékelőket kaptunk, a további érzékelők életre keltése érdekében újabb parancsfájlok kerültek a rakományvezérlő program magjába. A műholdra igény szerint újabb képességeket és foltokat lehet feltölteni.

Összefoglalás

Kevés műholdas programnak van olyan támogatási mozgástere és kockázatvállaló lehetősége mint amilyet a ***TacSat-1*** kezdeményezés biztosított. Ebből a szemszögből a ***TacSat-1*** program lehetővé tette a ***GOTS*** és ***COTS*** alkatrész eszközök újító kihasználását valamint a rakomány program maximális rugalmasságot és szabvány alapú működtetést biztosító újszerű megközelítését. A ***Copperfield-2*** moduláris felépítése gyors alkatrész beépíthetőséget tett lehetővé, bizonyítva a moduláris rakomány használhatóságát, melynek alkalmazási köre az ***UAV*** rendszerektől az űralkalmazásokig terjed és teljes mértékben ***Linuxra*** illetve ***GNU*** programokra alapoz. Írásunk születésekor a ***TacSat-1*** indítását 2005 februárjára ütemezték.

Köszönetnyilvánítás

A szerző szeretne köszönetet mondani a ***TacSat-1*** fejlesztésekhez nyújtott jelentős hozzájárulásukért ***Stuart Nicholson*** tanácsadónak és ***Eric Karlinnek***, ***Mike Steiningernek*** valamint ***Brian Davisnek*** az ***SGSS***-től a rakományvezérlő program magjáért. A ***Titan Corptól Brian Miceknek***, ***Chris Gembaroskinak***, ***Don Kremernek***, ***Tim Richmeyernek*** és a ***Copperfield-2*** csapatának az ***Aeronixnál***, valamint a ***PTR Csoporttól Jeff Angielskinek*** a ***Linux*** átültetésért, eszközmeghajtókért és érzékelőtámogató programokért. Köszönet illeti még ***Wolfgang Denxet*** és a ***Linux PowerPC*** közösséget amiért ilyen hibatűrővé és stabillá tették a ***PowerPC Linuxot***.

Linux Journal 2005. április, 132. szám

Christopher Huffine villamosmérnök az Amerikai Haditengerészet kutatólaboratóriumában, és a Haditengerészet Űrtechnológiai Központjában (Naval Center for Space Technology) dolgozik. A főiskola óta használ Linuxot különféle gépeken, asztali munkaállomásoktól kezdve a beágyazott számítógépekig.

KAPCSOLÓDÓ CÍMEK

- www.nrl.navy.mil/content.php?P=04REVIEW207
- www.nrl.navy.mil/content.php?P=04REVIEW212

Hatékony memóriakezelésű, kettősen láncolt lista

A jól ismert adattípus újabb megvalósításával értékes bájtokat takaríthatunk meg.

A gyártók fontos szempontja a kisméretű eszközök előállítási költségének csökkentése, amelynek kapcsán sokszor a memória kisebbre vétele is felmerül. Az egyik lehetőség a mindennapok során széles körben használt absztrakt adattípusok a megszokottól eltérő megvalósításának használata. Ilyen absztrakt adattípus a kettősen láncolt lista is. Írásomban a kettősen láncolt lista egy hagyományos és egy újfajta megvalósítását fogom ismertetni, illetve kitérek a beszúrás, a bejárás és a törlés műveletének végrehajtására is. Szóba kerül az egyes megvalósítások idő- és memóriaigénye is, így össze lehet hasonlítani előnyeiket és hátrányaikat. Az újabb megvalósítás a mutatók távolságára alapul, ezért ez alkalommal mutatótávolság alapú megvalósításnak fogom hívni. Esetében mindegyik csomópontnak csak egy mutatót kell tárolnia, ez a lista előre és hátra irányú bejárhatóságát egyaránt biztosítja. A hagyományos megvalósításnál egy előre irányú mutató mutat a lista következő, valamint egy visszirányú az előző elemére. Az emiatt jelentkező többletterhelés a hagyományos megvalósításnál 66 százalék, míg a mutatótávolság alapúnál 50 százalék. Ha többdimenziós kettősen láncolt listát használunk, például dinamikus rácsozatot akarunk létrehozni, akkor a megtakarítás ennél jóval nagyobb is lehet.
A kettősen láncolt listák hagyományos megvalósítását részletesebben nem fogom elemezni, leírása ugyanis gyakorlatilag bármely adatszerkezetekkel vagy algoritmusokkal foglalkozó könyvben megtalálható. A hagyományos és a mutatótávolság alapú megvalósítás használata nagyon hasonló, így memória- és időigényük is könnyen összehasonlítható.

A csomópont

A mutatótávolság alapú megvalósítás egy csomópontja a következőképpen épül fel:

```
typedef int T;
typedef struct listNode{
        T elm;
        struct listNode * ptrdiff;
};
```

A `ptrdiff` mutató a következő és az előző csomópontra irányuló mutatók közötti távolságot adja meg. A mutatótávolságot kizáró *VAGY (EXOR)* művelettel kapjuk meg. Minden ilyen listának van egy kezdő és egy záró csomópontja.
A kezdő csomópont a lista fejére mutat, a záró pedig a farkára. Definíció szerint a kezdő csomópont előtt egy *NULL* csomópont található, ahogy a záró csomópont után is. Egy egyetlen csomópontból álló listánál az előző és a következő csomópont egyaránt *NULL* csomópont, ezért a `ptrdiff` mezőbe is *NULL* mutató kerül. Egy két csomópontból álló listában a kezdő csomópont előtti csomópont *NULL*, az utána lévő pedig a záró csomópont. A kezdő csomópont esetében a `ptrdiff` mezőbe a záró csomópont és a *NULL* mutatója közötti *EXOR* művelet eredményét kell írnunk, amivel a záró csomópontot kapjuk. A záró csomópont `ptrdiff` mezőjébe hasonló eljárással a kezdő csomópont kerül.

Bejárás

Az egyes csomópontok beillesztésének alapja a bejárás. Az előre- és a hátrafelé mozgáshoz egyetlen egyszerű eljárásra van szükség. Ha átadott értékként a kezdő csomópontot adjuk meg, akkor – mivel az előző csomópont *NULL* – a bejárás értelemszerűen balról jobbra irányul. Ha átadott értékként a záró csomópontot adjuk meg, akkor a bejárás jobbról balra halad. A jelenlegi megvalósítás a lista közepéről indított bejárást nem támogatja, ugyanakkor ennek megvalósítása sem okozhat különösebb nehézséget. A `NextNode` (következő csomópont) a következőképpen épül fel:

```
typedef listNode * plistNode;
plistNode NextNode( plistNode pNode,
                    plistNode pPrevNode){
    return ((plistNode)
      ((int) pNode->ptrdiff ^ ( int)pPrevNode) );
}
```

Minden elem mutatótávolságát úgy számítjuk, hogy *EXOR* műveletet végzünk a következő és az előző csomópont között. Ha tehát *EXOR*-t végzünk az előző csomóponttal, megkapjuk a következőre irányuló mutatót.

Beillesztés

Tegyük fel, van egy új csomópontunk, valamint ismerjük egy meglévő csomópont értéktartalmát, és az új csomópontot a megadott értéket bejárási irány szerint elsőként tartalmazó csomópont után szeretnénk beilleszteni. (*1. kódrészlet*) Meglévő listába új csomópontot három csomópont mutatóinak módosításával illeszthetünk be, ezek a jelenlegi, a jelenlegi után következő és az új csomópont. Ha átadott értékként az utolsó csomópont értékét adjuk meg, akkor a lista végére fogjuk fűz-

1. kódrészlet Új csomópont beillesztésére szolgáló függvény

```
void insertAfter(plistNode pNew, T theElm)
{
   plistNode pPrev, pCurrent, pNext;
   pPrev = NULL;
   pCurrent = pStart;
   while (pCurrent) {
      pNext = NextNode(pCurrent, pPrev);
      if (pCurrent->elm == theElm) {
         /* véget ért a bejárás */
         if (pNext) {
            /* módosítjuk a meglévő következő
            ➥csomópontot */
            pNext->ptrdiff =
                (plistNode) ((int) pNext->ptrdiff
                           ^ (int) pCurrent
                           ^ (int) pNew);
            /* módosítjuk a jelenlegi csomópontot */
            pCurrent->ptrdiff =
              (plistNode) ((int) pNew ^ (int) pNext
                         ^ (int) pCurrent->ptrdiff);
            /* módosítjuk az új csomópontot */
            pNew->ptrdiff =
                (plistNode) ((int) pCurrent
                           ^ (int) pNext);
         break;
      }
      pPrev = pCurrent;
      pCurrent = pNext;
   }
}
```

ni az új csomópontot. Ha ilyen lépések sorozatával építünk fel egy listát, az időigényt is könnyen meg tudjuk vizsgálni. Ha az `InsertAfter()` (beillesztés mögé) eljárás nem találja a megadott értéket, akkor nem illeszt be új elemet.
Először a `NextNode()` eljárás segítségével ellépkedünk a megadott értéket tartalmazó csomópontig. Ha megtaláltuk, akkor mögé illesztjük az új csomópontot. Mivel a következő csomópont mutatókülönbözetet tárol, a megtalált csomóponttal ***EXOR*** kapcsolatba hozva feloldjuk a benne tárolt értéket. Következő lépésként ***EXOR*** műveletet végzünk az új csomópont felhasználásával, hiszen a következő csomópont előző csomópontja az új csomópont lesz. A jelenlegi csomópontot hasonló eljárást követve módosítjuk. Először egy ***EXOR*** művelettel feloldjuk a következő csomópontra vonatkozó mutatókülönbséget. Újabb ***EXOR*** műveletet végzünk az új csomóponttal, ezzel megkapjuk az új mutatókülönbséget. Végül, mivel az új csomópont a jelenlegi és a következő közé kerül, ezeket ***EXOR*** kapcsolatba hozva megkapjuk az új csomópont mutatókülönbségét.

Törlés

A törlés jelenlegi megvalósítása a teljes listát törli. Írásom célja a dinamikus memóriahasználat szemléltetése és a megvalósított primitívek futtatási idejének vizsgálata. Nyilván nem volna nehéz primitív műveletek mindenre kiterjedő gyűjteményével előállni. Mivel a bejáráshoz két csomópontra irányuló mutatót is ismernünk kell, a jelenlegi csomópontot nem törölhetjük azonnal a következő csomópont megtalálása után. Ehelyett, ha megtaláltuk a következő csomópontot, mindig az előzőt töröljük. Ha a jelenlegi csomópont helyének felszabadításakor a jelenlegi csomópont az utolsó, akkor végeztünk is. Egy csomópont akkor számít utolsónak, ha a vele végrehajtott `NextNode()` függvény ***NULL*** csomópontot ad vissza.

Memória- és időigény

Az itt tárgyalt megvalósítás kipróbálására a második kódrészletet a ***Linux Journal*** FTP-helyéről (***ftp.ssc.com/pub/lj/listings/issue129/6828.tgz [1]***) lehet letölteni. Saját ***Pentium II*** (349 MHz, 32 MB RAM és 512 KB másodszintű gyorsítótár) gépemen futtatva a mutatótávolság alapú megvalósítás 15 másodperc alatt hozott létre húszezer csomópontot. Ennyi időre van szükség húszezer csomópont beillesztéséhez.
A teljes lista bejárása vagy törlése egy másodpercig sem tart, ezért ezeket a műveleteket nem ilyen időfelbontásban érdemes vizsgálni. Rendszerszintű megvalósításnál az időzítéseket inkább milliszekundumos felbontással szokták figyelni. Ha ugyanazt a mutatótávolság alapú megvalósítást tízezer csomóponttal futtatjuk, akkor a beillesztés mindössze három másodpercig tart. A teljes lista bejárása vagy törlése kevesebb mint egy másodpercet igényel. Húszezer csomópont tárolásához 160000 bájtra van szükség, tízezer csomópont pedig 80000 bájtnyi területen fér el. 30000 csomópont tárolásakor a beillesztéshez 37 másodpercre van szükség. A teljes lista bejárása vagy törlése ez esetben is egy másodperces időtartam alatt történik meg. Az időigényekből nagyjából látható, hogy a dinamikus memória (kupac) használata egyre erőteljesebb. A dinamikus memóriában egyre nehezebb szabad szakaszt találni, ennek ideje nemlineárisan, sőt, inkább hatványosan növekszik.
A hagyományos megvalósításnál tízezer csomópont beillesztése ugyancsak három másodpercig tart. A bejárás ideje előre és hátra haladva egyaránt egy másodperc alatt van. Tízezer csomópont tárolásához összesen 120000 bájtra van szükség. Húszezer csomópont kezelésekor a beillesztés 13 másodpercet igényel, a bejárás és a törlés ideje pedig továbbra is egy másodpercen belül marad. Húszezer csomópont kezeléséhez összesen 240000 bájtot kell biztosítani. Harmincezer csomópontnál 33 másodpercig tart a beillesztés, és továbbra is kevesebb mint egy másodpercig a bejárás és a törlés. Harmincezer csomópont összesen 360000 bájt területet foglal le.

Összefoglalás

A kettősen láncolt lista hatékony memóriakezelésű változata az időigény tekintetében minimális többlet mellett valósítható meg. Okos tervezéssel mindkét megvalósításhoz össze lehet állítani a primitív műveletek átfogó gyűjteményét, ám a műveletek időigénye jelentős mértékben eltérhet.

Linux Journal 2005. január, 129. szám

Prokash Sinha 18 éve rendszerprogramozással foglalkozik. Dolgozott már fájlrendszereken, hálózat- és memóriakezelésen, UNIX, OS/2, NT, Windows CE és DOS operációs rendszer alatt egyaránt. Fő érdeklődési területe a rendszermag és a beágyazott rendszerek. A prokash@garlic.com címen érhető el.

Előadóművészek a weben

Az UpStage segítségével a legközelebbi színház mindössze pár egérkattintásnyira van.

Írók, zenészek, festők, filmkészítők és más művészek régóta használják a webes eszközöket. Mindössze egyetlen hagyományos művészeti forma nem képviseltette magát eddig a kibertérben – a színház. Ha azonban hajlandóak vagyunk az új médiumhoz alkalmazkodni, máris új művészet születik, a kiberszínház.
A ***kiberszínház (cyberformance)*** kifejezés a ***Helen Varley Jamieson Új zélandi*** művésztől származik, aki a következőképpen határozta meg: ***„előadás, amely valós időben az Internet segítségével hozza össze a távoli előadóművészeket egy élő színházi esemény kialakításához”***.
Néhány évig az ***Avatar Body Collision*** kiberszínház csapattal dolgozott, ingyenes Internet csevegőkkel alakítva ki előadásokat az kibertérben. Végül, hogy ő maga, segítői és hallgatósága Web alapú színpadhoz juthassanak, ő kezdeményezte a ***Douglas Bagnall*** által készített ***UpStage*** nyílt forrású projektet (lásd a hálózati forrásokat). A 2004. januárjában megjelent első kiadás az ***Újzélandi Kutatási, Tudomány és technológia és Kreatív Újzéland Minisztériuma*** támogatta. Jelenleg a fejlesztések folytatásához keresnek forrásokat.
Természetesen a programot nem csak hálózati előadóművészek használhatják. Az ***UpStage*** érdekes lehetőséget biztosít a hálózati tanításra, valamint a termék, illetve egyéb bemutatók tartására. Akár együttműködési rendszerként is használhatjuk virtuális munkacsoportok esetében. Az ***UpStage*** nagy erőssége a felhasználóbarát és kiválóan használható felülete: Az előadóknak és a hallgatóságnak nincs szüksége többre egy szokásos böngészőnél és Internet kapcsolatnál ahhoz, hogy részt vehessen az előadáson. Az új fiúk nagyon hamar megtanulják az alapokat és azon kapják magukat, hogy máris szövegeket kopácsolnak és avatart váltogatnak.

Színházunkat gondosan meg kell terveznünk

A kiszolgálóprogram ***Python*** nyelven készült és saját webkiszolgálóval rendelkezik, így a művészek bárhol könnyedén elő tudják készíteni a színpadot, ahol csak a noteszgépüket a hálózatra csatlakoztathatják. A webkiszolgálón kívül (amelyhez a ***Python Twisted*** keretrendszerére van szükségünk) a program igen széles körben használ más nyílt forrású eszközöket, amelyek általában megtalálhatóak egy ***Linux*** rendszeren. Ilyen például a ***Festival*** szöveg-beszéd átalakító, a ***netpbm*** eszközök és a ***gif2png***. A cikk melletti széljegyzetben megtudhatjuk milyen problémák vannak a ***GIF*** képekkel.
Két csomag viszont gyakran hiányzik a ***Linux*** terjesztésekből: az ***swfttools*** és a ***lame MP3*** kódoló. A ***The Coroner's Toolkit*** időzítő programja, amelyet a beszédszintetizáláshoz használ, szintén gyakorta kimarad. Ez azonban nem olyan nagy probléma, ha valaki nem fél hozzányúlni a forráskódhoz.
A színpad ***Flash*** ügyfél így kerül a képbe az ***swfttools***. Ezzel alakítjuk át a színpad dekorációhoz és az avatarokhoz használt ***PNG*** és ***JPEG*** képeket ***Flash*** formátumúvá. Következésképpen az előadóművészeknek és a hallgatóknak egyaránt szükségük lesz a ***Macromedia Flash*** bővítményre a böngészőjükben. A ***KHTML***- és ***Mozilla***-alapú böngészők kiválóan használhatóak, az opera viszont jelenleg még nem megfelelő.
Sajnos, írásunk születésekor az ***UpStage*** aktuális verziója nem támogatja a ***PATH*** beállításokat. Ezért aztán érdemes ellenőrizni, hogy a fent említett programok megtalálhatóake az ***sh***-ba fordításkor beégetett könyvtárak egyikében:

```
$ strings /bin/sh | grep -E “/(bin|sbin)”
[...]
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
/sbin:/bin
```

Amennyiben nem, érdemes létrehoznunk néhány csatolást. Különben a hibakeresés nem lesz egyszerű, ugyanis az ***UpStage*** nem kifejezetten nagymestere a minden eshetőséget lefedő értelmes hibaüzeneteknek. A dolgok csak tovább bonyolódnak ha a hangeszközt is használni szeretnénk.
Az ***UpStage*** ugyan a ***/usr/local/bin*** könyvtárban keresi a grafikus eszközöket, nem feltétlen találja meg a ***lame***-t is itt. Ezért azok akik nem szeretik a forrást piszkálni, úgy tűnik kénytelenek lesznek létrehozni egy közvetett hivatkozást ***/usr/bin/lame*** néven.

A színház felállítása

Ideje beindítani a kiszolgálót. Tömörítsük ki a forrásállományt ***Upstage-2004-09-28.tar.gz***, majd lépjünk be a frissen létrejött ***Upstage*** könyvtárba. Az itt található ***go.sh*** program megpróbálja megölni a ***Upstage/twistd.pid*** állományban megadott régi twisted-kiszoglálót majd elindít egy újat. Ezért aztán ne essünk pánikba, ha elsőre néhány hibaüzenettel találkozunk amikor előjogok nélkül futtatatjuk a

```
./go.sh
```

programot. Csak annyit jelentenek, hogy az ***Upstage*** létrehozta a ***pid***-állományt.
Biztonsági megfontolásokból az ***UpStage***-et nem ajánlatos root jogon futtatni. Pontosan emiatt használ a kiszolgáló 1024 feletti, előjogok nélküli kaput. Az ***UpStage*** kiszolgáló által használt kapu szabadon beállítható. Amennyiben valakinek nem tetszik az alapértelmezett 8081-es kapu, változtassa meg a következő sort az ***Upstage/upstage/config.py*** állományban:

```
WEB_PORT = 8081
```

majd futtassa le újra a `./go.sh`-t.
Tekintve hogy az ***UpStage*** 2004 szeptemberi változatából hiányzik az a könyvtár, ahol a kiszolgáló az ideiglenes ***MP3*** állományokat tárolja, sok gondtól kíméljük meg magunkat ha előre létrehozzuk azt:

```
mkdir html/speech
```

Most már a helyi böngészőnket átirányíthatjuk a ***http://localhost:8081/*** címre, és máris a színházunk bejáratában állunk (*1. ábra*). Ezt az ***Upstage/html/index.html*** állomány ***HTML*** szövegének és ***Upstage/html/style/main.css*** stíluslapjának átszabásával igazíthatjuk saját igényeinkhez. Érdemes egy relatív hivatkozást felvenni a színpadhoz "`<a href="/stages"></a>`" (a hallgatóságunk hálás lesz) illetve kialakítani egy bejelentkező felületet a színészeknek. A színházban van egy hátsó ajtó is a személyzetnek.
A ***http://localhost:8081/admin URL*** illetve az ***http://localhost:8081/login.html*** azonnal a bejelentkező képernyőhöz visz bennünket amelyet az ***Upstage/html/login.html*** állományban változtathatunk meg.

Társulat felfogadása

Az ***UpStage*** alapértelmezett színházigazgatójának neve „z", és z-nek nincs jelszava. Ezt valószínűleg meg szeretnénk változtatni, úgyhogy jelentkezzünk be és lépjünk a színház igazgatóságára. Az ***„Add a new player link"*** segítségével ugorjunk a ***http://localhost:8081/admin/new/player*** lapra és vegyük fel az új igazgató nevét és jelszavát. Ha őt szeretnénk megtenni nagyfőnöknek, aki felfogadhat és elbocsáthat ne feledjük el az ***„Add or Remove Players"*** melletti jelölőnégyzetet megjelölni (*2. ábra*).
Az új szereplő a ***Upstage/config/players.xml*** beállító állománya kerül, valahogy ilyenformán:

```
<player
password="551a9c1c68844936b0d182080fe7dcc0"
 name="lj" rights="act,admin,su">
</player>
```

A jelszó attribútum nem a tényleges jelszót (amely jelen esetben az `upstage` volt) hanem annak md5 összegét tartalmazza tartalmazza. Ha kedvenc szövegszerkesztőnkkel szeretnénk felvenni egy felhasználót, a jelszót a következőképpen is létrehozhatjuk:

```
$ echo -n "upstage" | md5sum
551a9c1c68844936b0d182080fe7dcc0  -
```

1. ábra Az alapértelmezett belépőterem egyértelműen utal a program származására

A név attribútum a szereplő nevét tartalmazza, és legfeljebb három jogot adhatunk meg. A nagyfőnöknek a ***su*** jogra van szüksége. Akiknek a színpadon látható vagy használható dolgokat szerkeszthetik, azoknak ***admin*** jogot kell adnunk, végül valamennyi szereplőnek ***act*** jogot kell osztanunk.
Sajnos a webes felület nem éppen hibátlan, amikor a felhasználók törlésére vagy szerkesztésére kerül a sor. Először is nem a helyes jogokat mutatja nekünk, ráadásul nem is engedi megváltoztatni őket (még rendszergazdai jogokkal sem) valamint törölni sem tudjuk őket. Ha az megfelelő felhasználó előtti jelölőnégyzetekre kattintunk a ***http://localhost:8081/admin/edit/player/*** lapon, majd rendszergazdakén megnyomjuk a ***„Remove Players"*** (játékosok eltávolítása) gombot, az ***UpStage*** ugyan eltávolítja a vonatkozó játékost a folyamat végezetéig, de nem törli a ***players.xml*** állományból. A kiszolgáló újraindítása után valamennyi játékos ismét vígan élővé válik. ***Douglas Bagnall*** ígérete szerint hamarosan javítja a hibát.

Gif gondok

Még ha megfelelően is tettük fel a ***gif2png*** eszközt, a 2004 szeptemberi ***UpStage*** verzió nem képes ***GIF*** képeket felhasználni az avatarokhoz, kellékhez vagy hátterekhez.
Az új verzió megérkezésééig, magunk is kijavíthatjuk ezt a hibát ha megjegyzésbe tesszük a ***Upstage/img2swf.py*** 38. sorát majd töröljük a

```
"giftopnm" flag "-background "#fff""
```

bejegyzést a 63. sorban. A soroknak tehát a következőképpen kell kinézniük:

```
[...]
35 def do_gif(tfn, swf):
```

2. ábra LJ éppen nagyfőnökké válik

```
[...]
38 #     os.path.remove(png)
[...]
57 def thumbnailer(filetype, tfn, thumb, log):
[...]
63          'image/gif'       :      'giftopnm %s |
pnmscale -height=10 | pnmtojpeg > %s'
```

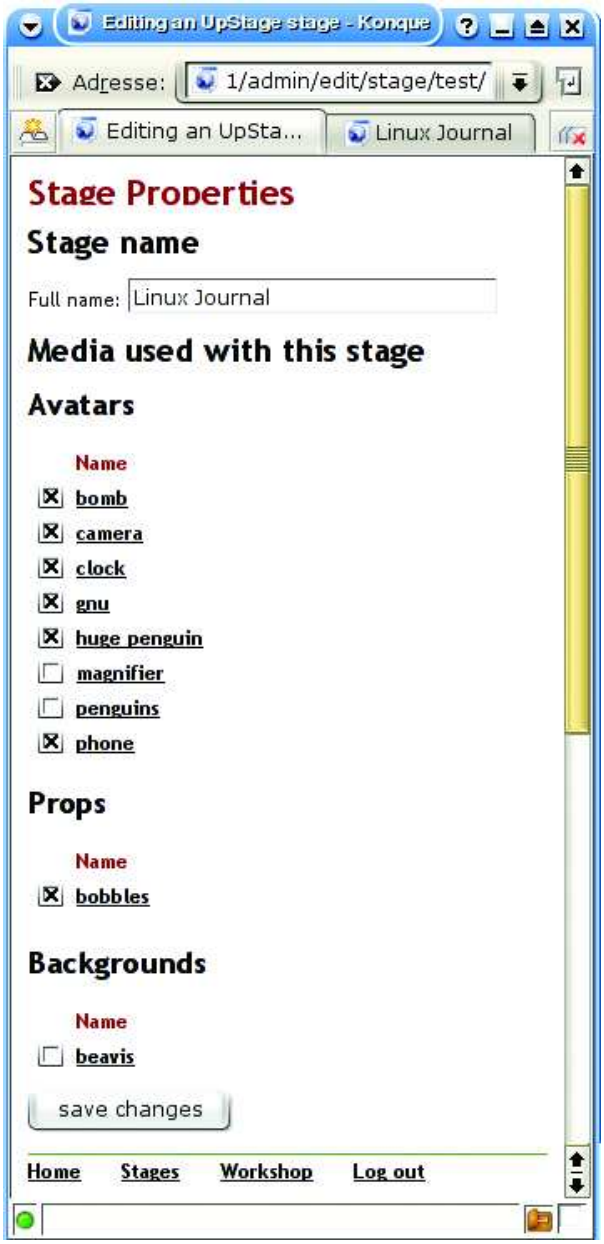
Szerepek és kellékek kialakítása

A felhasználókkal és jogosultságokkal kapcsolatos problémák színházunk kelléktárában már nem jelennek meg. Helyszíneket avatarokat (az avatar előadásunkban egy karakter adott öltözetének felel meg), háttérfüggönyöket, színpadterveket és kellékeket hozhatunk létre és szerkeszthetünk a ***http://localhost:8081/admin/ URL***-en elérhető munkapadon. Ez utóbbiakat hordozhatják az avatarjaink, így azok mindig megjelennek a az avatar bal felső részén, mint a *6. ábrán* látható bombához csatolt kék buborékok.

Amikor új avatarokat, kellékeket és háttréfüggönyöket hozunk létre, több választásunk is lehet: használhatunk két dimenziós képeket, ***Flash*** animációkat, és videó folyamokat. A mozgóképekkel azért legyünk óvatosak, nagy sávszélességet igényelnek és igazi teljesítményrombolók.

A videófolyamaok helyileg elérhetőeknek kell lenniük, és az ***Upstage/html/media/*** könyvtárban kell tárolnunk őket. ***Linux*** alatt az ***UpStage*** felhasználói kézikönyv a ***webcamd*** programot javasolja a video folyamok ***FTP*** rendszerű feltöltéséhez. Sajnos a ***webcamd*** eredeti fejlesztői oldala úgy tűnik már bezárt (lásd a forrásokat), azonban binárisként és forrásként még mindig elérhető a Debian kiszolgálókról.

A valódi színházaktól eltérően, az avatar, díszlet, vagy kellék egyszerre több színpadhoz is rendelhető. Ezt a ***Manage***



3. ábra Bár a raktárbejegyzésekre rákattinthatunk, a hivatkozások nem az adott elem szerkesztőpaneléhez visznek bennünket, hanem a Flash állományra mutatnak

an existing stage (meglévő szín kezelése) részben tehetjük meg (***http://localhost:8081/admin/edit/stage/<stagename>/***, *3. ábra*).

A színpadok beállításállományai ***XML*** formátumban tárolódnak az ***Upstage/config/stages.xml*** és az ***Upstage/config/stages/<stage-id>/config.xml*** állományokban. Az elsőben a színpadok felsorolását találjuk; az utóbbiak egy-egy színhez rendelt kelléktárat tartják nyílván.

Talán mondanunk sem kell, hogy a három típusú raktárkészlet eltérő szöveges beállítóállománnyal rendelkezik, ezek: az ***Upstage/config/props.xml***, ***avatars.xml*** és a ***backdrops.xml***. Valamennyi az *1. listában* bemutatott szerkezetet követi.

Bár a gyökérelem neve nem igazán számít, az ***UpStage*** az ***avatars***, ***props*** és ***swamp*** jelzéseket használja a fájlok létre-

1. lista Az avatars.xml szerkezete

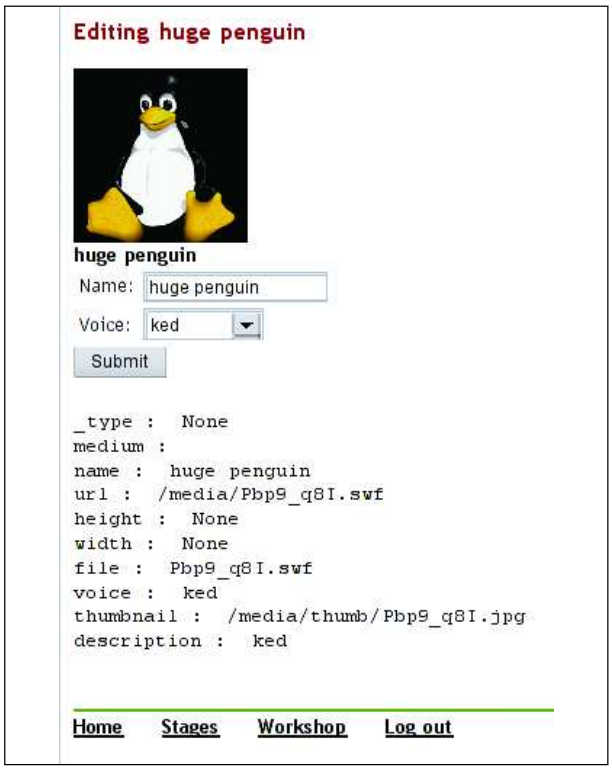
```
configuration file
<avatars>
<avatar url="/media/Pbp9_q8I.swf" voice="ked"
name="huge penguin" file="Pbp9_q8I.swf"
thumbnail="/media/thumb/Pbp9_q8I.jpg">
</avatar>
<avatar url="/media/clock.swf" name="clock"
file="clock.swf"
thumbnail="/media/thumb/clock.jpg">
</avatar>
</avatars>
```

2. lista A config.py hangdefiníciói

```
VOICES = {
          'kal': ("| timeout 15 text2wave -eval
'(voice_kal_diphone)' -otype raw",
                  _fest),
        [...]
        }
```

1. táblázat ***A Stage Inventory és Avatar elemek tulajdonságai***

Tulajdonság	Érték
`url`	A megfelelő Flash állomány elérési útja, az Upstage/ html alatti médiakatalógus címtől számítva. Az UpStage véletlen fájl neveket használ. Ha magunk szerkesztjük a bejegyzéseket, nem rossz gondolat értelmes neveket adni.
`name`	Az elem neve. Ez jelenik meg a színpadon úgyhogy körültekintően válasszuk meg. Az előadás közben, a csevegőablak alatti szövegmezőbe gépelt `/nick <name>` paranccsal tudjuk megváltoztatni.
`file`	A megfelelő Flash állomány neve elérési út nélkül.
`thumbnail`	JPEG formátumú előnézeti kép az Upstage/html könyvtárához képest. Az UpStage ezeket az Upstage/html/media/thumb könyvtárban tárolja. ezek az előnézeti képek jelennek meg a színen, segítve a színészeket a megfelelő eszköz kiválasztásában.
`voice`	Ez az elem csak az avatarokra vonatkozik, és nem is kötelező. Itt adhatjuk meg a szöveg-beszéd átalakítás paramétereit. A hangfájlokat az Upstage/upstage/config.py állományban tároljuk.



4. ábra Avatarunk szerkesztőpanelén nem láthatjuk, hogy ez a pingvin olyan hatalmas, hogy csaknem a teljes képernyőt elfoglalja

hozásakor. Igazándiból csak az alárendelt elemek (***avatar***, ***prop*** és ***backdrop***) számítanak. Minden alárendelt elem négy kötelező és egy elhagyható paraméterrel rendelkezik, ezeket az *1. táblázatban* foglaltuk össze.
Válasszuk a munkapadon a ***http://localhost:8081/admin/edit/avatar/*** hivatkozást, és a megfelelő elemre kattintva szerkesszünk át egy létező avatart. A megfelelő űrlap (*4. ábra*) két lehetőséget kínál fel, ahol az elem hangját és nevét változtathatjuk meg.
Sajnos az űrlap nem sok segítséget nyújt amikor a kép színpadi méretét szeretnénk megbecsülni. Az ***UpStage*** ügyfél a színeket úgy jeleníti meg, hogy beleférjenek a böngésző ablakába, míg a kellékek és avatarok körülbelül eredeti méretük háromszorosaként jelennek meg. A felhasználói kézikönyvben (lásd a forrásokat) megtaláljuk a képek készítéséhez ajánlott méreteket és formátumokat.

Zajkeltés

Amikor a hangbeállításokhoz érkezünk, többet már nem kell az ***XML***-el foglalkoznunk, innentől a ***Pythoné*** a terep. Az ***Upstage/upstage/config.py*** állományban találunk egy ***VOICES*** nevű szakaszt, azaz lényegében egy könyvtárobjektumot, ahol a szöveg-beszéd előállítás parancsait találjuk meg (*2. lista*). Elmondhatjuk, hogy az ***UpStage*** beszédgenerálása nem feltétlenül a ***Festival***-tól függ. Ez különösen a nem-angol anyanyelvűek számára fontos hiszen a ***Festival*** terjesztés maga kizárólag angol nyelven működik.
Amennyiben új hangokat szeretnénk felvenni, egyszerűen kezdjünk egy új sort kapcsos zárójelek között, melyet a ***VOICES*** kulcsszó követ. Gépeljük be az új hang nevét idézőjelek között, és írjuk a következőket:

```
: ("| ", _fest),
```

Ügyeljünk rá, hogy a sort pontosan annyi szóköz karakterrel kezdjük, hogy a nyitó idézőjelünk a többi hangbeállítás alá kerüljön. A ***Python*** igen érzékeny a bekezdésekre, és a hibás behúzás eredményeképpen az ***UpStage*** könnyen leállhat.
A csővezeték (|) karakter után, tetszőleges parancsot (csővezetéket) bevihetünk, feltéve, hogy az szöveget olvas a szabványos bemenetről és 16 kHz-es nyers ***PCM*** kimenetet készít a `stdout`-ra. Megoldásunkat a következő paranccsal próbálhatjuk ki:

```
echo "Say something in the relevant language" |
<command> | timeout 15 lame -S -x -m s -r -s 16
–resample 22.05 –preset phone - /tmp/test.mp3
```

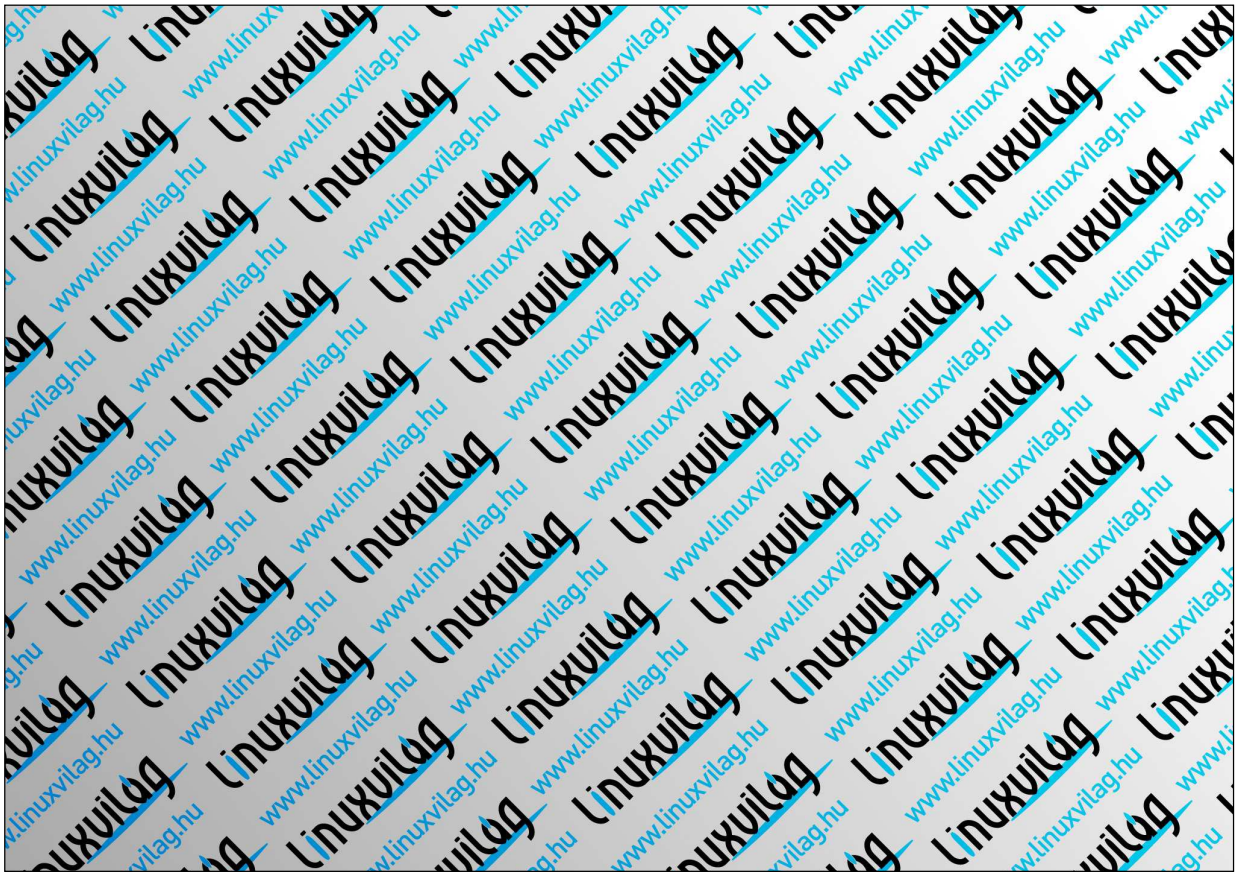
Ha az ***MP3*** lejátszónkkal lejátszott ***/tmp/test.mp3*** állomány azt adja vissza amit kell neki, parancsunkat felvehetjük a ***config.py*** állományba. Minthogy az ***UpStage*** kényes az elérési utakra, inkább teljes elérési utat használjunk.
Az eredeti ***config.py*** valószínűleg jóval több szöveg-beszéd átalakító parancsot tartalmaz, mint amennyit a telepítésünk támogat. Tekintve, hogy ezek mind megjelennek a hang legördülőmenüben az avatarok készítésekor és szerkesztésekor, érdemes megjegyzésbe tennünk őket a # jellel. Legyünk óvatosak, hiszen az eredeti hangdefiníciók megjegyzésbetételekor két vagy három sorhoz is hozzá kell nyúlnunk. Ha valamelyiket kihagyjuk, és a változtatásaink érvényesítése kedvéért újra szeretnénk indítani a kiszolgálót a `./go.sh` paranccsal, ilyen hibaüzenetekkel találkozhatunk:

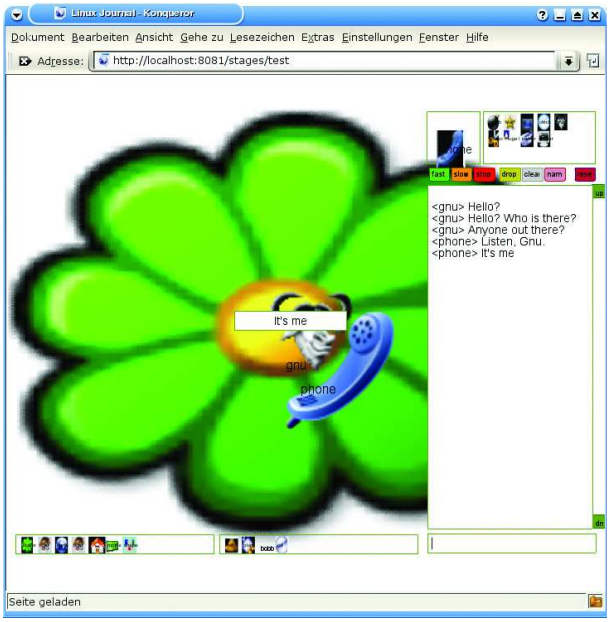
```
Failed to load application: invalid syntax
(config.py, line 92)
```

Ha ezek után minden avatarunk elveszti a hangját, valószínűleg az alapértelmezett hangdefiníciót is megjegyzésbe tettük. Nem jó ötlet! Az alapértelmezett bejegyzés mögötti parancs megváltoztatása még rendeben van, de fosszuk meg az ***UpStage***-et teljesen tőle.

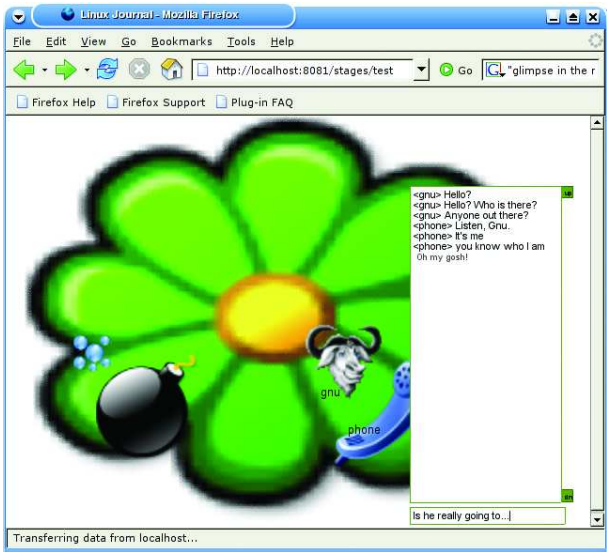
Eljött a főpróba ideje

Ha a színpad készen áll, eljön a főpróba ideje. Ez annyit jelent, hogy az összes játékosnak be kell jelentkeznie a megfelelő színre műhely ***Stages*** hivatkozásának segítségével (***http://localhost:8081/stages/***). Belépés után nagy üres terülten találjuk magunkat, azaz a színen, amit jobbról a csevegőablak keretez, ahol a kimondott szöveg olvasható. A képgaléria a csevegőablak alatt találjuk. A képgaléria bal oldalán található díszlet ikonokkal változtathatjuk meg a szín arculatát. A jobboldalon találjuk a kellékeket (*5. ábra*).
A csevegőablak felett a felhasználók egy gombsort találnak amelyekkel elsősorban az avatart lehet vezérelni. Maguk a karakterek a jobboldali gombok feletti szekrényben találhatók. A színészek itt találják meg a színben szereplő vala-





5. ábra Amikor díszletet választunk, ügyelnünk kell rá, hogy a jobb külső részét a csevegőablak takarja majd el



6. ábra A hallgatóság a csevegőablakba gépelve tapsolhat vagy huhoghat

mennyi avatar előnézeti képét. Ha valamelyikre rákattintunk, a szekrény baloldalán tükrözve jelenik meg. Így aztán egyetlen pillantása a tükörbe és máris tudjuk, milyen szerepet játszunk.

A karakterünk azonban nem jelenik meg azonnal a színen. Ha ilyenkor valami szöveget gépelünk az csevegőablakba, avatarunk narrátorként szerepel. Amikor először kattintunk a színpad ablakra, az avatarunk megjelenik, szövege pedig léggömbként olvasható (*5. ábra*). A rózsaszín név gombbal állíthatjuk be, hogy az avatar neve látszódjon-e a szövegként.

Ha máshová kattintunk a színpadon, avatarunk lassan odavándorol. Ha azonnal oda szeretnénk ugrani, előbb a zöld gyorsítógombot nyomjuk meg; Az eredeti sebességünkhöz a narancs színű lassítógombbal térhetünk vissza. A karakter megállításához a vörös állj gombot kell lenyomnunk.

Ha avatarunknak valamilyen kelléket szeretnénk adni, kattintsunk a megfelelő előnézeti ikonra, a képgaléria jobb oldalán, a színpad alatt. Ezek után az követi majd avatarunk minden mozdulatát.

Amikor egy másik előképre kattintunk a szekrényben, régi karakterünk a színpadon marad, de egy más színésztársunk átveheti azt. Amikor az éppen használt avatarunknak el kellene hagynia a színt, használjuk a sárga (***drop***) gombot. Jelenleg ez az egyetlen módja a kellékek eltüntetésének is. bár a kellékeket meg tudjuk változtatni ha egy másik kellék ikonra kattintunk (igaz nem minden mellékhatás nélkül játszódik le) az ***UpStage*** jelenlegi verziójában egyenlőre nincsen ***„kellék eltüntetése”*** gomb.

A szürke törlés gomb kiüríti a teljes színpadot, kivéve a színésztársaink által mozgatott avatarokat. A művelet azonban mellékhatással jár: színésztársaink csak akkor tudják ismét mozgatni a karakterüket, ha ismét kiválasztják azt a szekrényből.

Néha úgy tűnhet, hogy néhány dolog nem tűnt el a színről. A legtöbb esetben a böngésző frissítés gombja segít, de ilyenkor ismét ki kell választanunk az avatarunkat.

Ha valamilyen okból teljesen előröl kellene kezdenünk, azt a vörös ***reset*** gombbal tehetjük meg. Ezt jobb ha nem tesszük előadás közben, amikor mások is vannak a színen, ez ugyanis kegyetlenül kidob mindenkit a színpadról, és csak böngésző frissítéssel térhetünk vissza. néhány játékosnak esetleg újra be kell jelentkeznie. A ***reset*** gomb áthelyezése valami kevésbé nyomkodásra ingerlő helyre előkelő helyet foglal el a fontos javítások listáján.

Ha valaki nincsen bejelentkezve, kizárólag a színpadot és a csevegőablakot láthatja (*6. ábra*). Ez azonban nem jelenti azt, hogy a hallgatóságnak nincsen semmi szava. A hallgatóság által gépelt szöveget mindenki olvashatja a csevegőablakban, így az ***UpStage*** kiváló választás lehet online tanításhoz vagy bemutatókhoz. A hallgatóság üzeneteire válaszolhatunk vagy figyelmen kívül hagyhatjuk őket. Az egyetlen különbség, hogy a hallgatóság üzenetei szürke színű betűkkel jelennek meg, amely előtt nincsen avatar név és nem persze szóban sem hangzik el. Így az ***UpStage*** tapsviharai csendesek.

Akár az ***UpStage*** feltelepítése nélkül is kipróbálhatjuk. Az ***Avatar Body Collision*** minden hónapban nyílt előadást tart azok számára, akik szívesen kipróbálnák vagy megismerkednének az ***UpStage*** alapú interaktív színjáték lehetőségeivel. Ne mulasszuk el a következő alkalmat (lásd a forrásokat). További segítséget a felhasználói kézikönyv és a levelezőlista segítségével kaphatunk.

Linux Journal 2005. április, 132. szám

A cikk forrásai: ➲ ***www.linuxjournal.com/article/8056***

Patricia Jung (trish@answergirl.de) az Open Source Press (www.opensourcepress.de) szerkesztője és rendszergazdája. Ebben a szerepkörben igazán örül, hogy lehetősége van kizárólag Linux és UNIX témával foglalkozni.

Fd.o: Üljünk át egy szebb asztalhoz

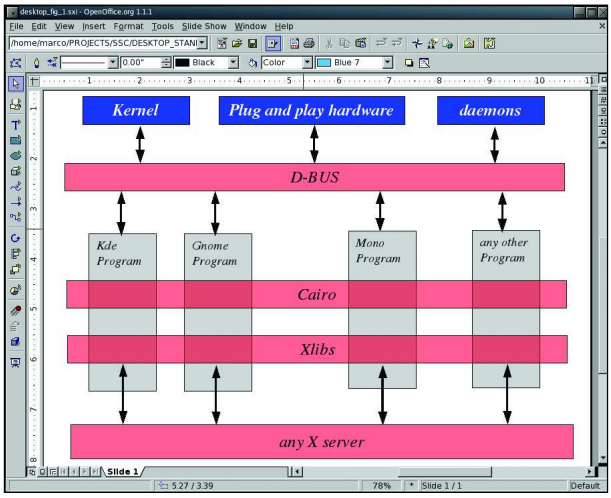
Senki ne gondolja, hogy az asztali környezetek között háború folyik. Az alkalmazások és az asztali környezetek a háttérfalak mögött egymással együttműködnek, és a való életben is helytállt szabványokat alkalmazva segédkeznek abban, hogy mindenki alkalmazásai gond nélkül működjenek, sőt, együttműködjenek.

A végfelhasználókat csak a kívánt feladatokat ellátó alkalmazások érdeklik. Azért választják a ***Linuxot***, hogy ezeket az alkalmazásokat szabadon, egyenként választhassák ki. Számukra az integrált asztali környezet tetszőleges programelegy kiválasztásának szabadságát jelenti, és a garanciát arra, hogy ennek összetevői működni fognak egymással. Egy monolitikus asztali környezet a programozókat is korlátozná. Saját kódunk más alkalmazásokkal való együttműködésének biztosítása alapvető követelmény, ha nem az első számú jellemző, mely a program hasznosságát megszabja. Ha célunk eléréséhez csak egy vagy két fejlesztői eszközláncot vehetünk igénybe, annak nem sok értelme van. Korábban az ***XFree86 GNU/Linux*** asztali környezet fejlesztése túlságosan lassan haladt, és teljesítménye sem volt kielégítő. Sok eszközt, a ***fontconfig***-tól egészen a ***zlibig*** a külső függőségek elkerülése miatt meg kellett kettőzni. Ha egy illesztőprogram megváltozott, az egész csomagot újra ki kellett adni. Mindezt tetézte, hogy az ***XFree86*** felhasználói szerződése a múlt évben úgy változott meg, hogy a ***GPL*** programok számára megtiltotta az új kódok becsatolását. A szerződés miatt számos terjesztés összeállítói egyszerűen az új változat elhagyása mellett döntöttek.

A ***Freedesktop.org (FD.o)*** 2000 márciusában alakult, elsődleges célja a fejlesztők segítése volt a fenti műszaki kérdések megoldásában. Alapítói olyan alaprendszert akarnak létrehozni, amelyre bármilyen asztali környezet felépíthető. A megoldást független specifikációk kidolgozásában látják, amelyeket szükség szerint működő kódok egészítenek ki. A formális szabványosítást más testületekre kívánják hagyni. Az előírásokat követve valódi együttműködést lehet megvalósítani az alkalmazások között, méghozzá fejlesztésük során a lehető leghamarabb – ideális esetben még annak megkezdése előtt. Minden program ***LGPL*** vagy ***X***-jellegű szerződés hatálya alá kerül. A ***FD.o*** számos rokonszenves tervezetet támogat, ám írásomban csak a fő eszközöket szeretném bemutatni, ezek alkotják az úgynevezett ***FD.o*** alaprendszert.

Xlibek

Az ***X Window System*** egy átlátszó hálózati protokoll grafikus megjelenítéshez. A grafikus felületű alkalmazások az ***X***-et használják arra, hogy az ***X***-kiszolgálónak nevezett,



1. ábra Integráció a Freedesktop.org elképzelései szerint: kiszolgálók, bármely alkalmazás által használható könyvtárak és kommunikációs protokollok; mindez független attól, hogy az alkalmazások milyen asztali környezetben készültek.

a képernyőt ténylegesen kezelő programnak rajzolási utasításokat adjanak. Egészen a múlt évig a kiszolgálók és a könyvtárak jellemzően egyetlen monolitikus csomagba kerültek. ***FD.o*** részekre osztotta a csomagot, ezek a részek már önállóan is fejleszthetők és csomagba illeszthetők. Mindennek a fő előnye az, hogy a linuxos fejlesztők tetszésük szerint keverhetik össze és szabhatják testre az egyes részek megvalósításait.

További ***X***-es fejlesztés a fán belüli függőségek eltávolítása, az ***autotools*** használata fordítórendszerként, valamint az ***iconv*** könyvtár alkalmazása az ***Unicode*** és egyéb kódolások közötti átalakításokra. Az ***X*** protokollt burkoló könyvtárakat nevezzük ***Xlib***-eknek. A ***FD.o*** ezek első változatát 2004 januárjában adta ki. Követik az ***X*** szabvány előírásait, így bármely ***X***-kiszolgálóval képesek együttműködni.

A kiterjedt optimalizálások ellenére az ***Xlib***-ek mérete a kisebb teljesítményű gépeken továbbra is gondokat okozhat. További probléma, hogy bizonyos ***Xlib***-kérések blokkolódnak, amíg választ nem kapnak, függetlenül attól, hogy sok

esetben ez elkerülhető volna. Ez viszont ütközik a 2.6-os rendszermagok lappangási idők csökkentésére irányuló fejlesztéseivel. Az ***Xlib***-ek gyorsítótárazás, rétegezés és hasonló megoldások révén sokat tesznek a protokoll elrejtése érdekében is. Ezek a törekvések bizonyos esetekben előnyt, más esetekben csak többletterhelést jelentenek. Végül, de nem utolsóként meg kell említeni, hogy az X-bővítmények készítésének támogatása is korlátozott.

Az ***FD.o*** javaslata mindezen problémák megoldására az ***X C Binding***, röviden ***XCB***. Ez egy második könyvtár, és alkalmas arra, hogy új eszközkészletek és az ***Xlib API*** egyes részeinek egyszerűbb emulációja alapjául szolgáljon. Az ***XCB***-t úgy tervezték, hogy zökkenőmentes együttműködésre legyen képes a ***POSIX*** szálakra vagy egyetlen szálra épülő programokkal. A kód fenntartja a bináris kompatibilitást az ***Xlib*** bővítményekkel és alkalmazásokkal, vagyis használatához nincs szükség a bővítmények újrafordítására. Ezzel a megoldással a fokozatos ***Xlib - XCB*** áttérés könnyebben, a szolgáltatások egy részéről való lemondás nélkül is megvalósítható. A következő lépés ezen az úton az ***Xlibs Compatibility Layer (XCL)***, amelynek segítségével a meglévő, ***Xlibre*** alapuló alkalmazások is kihasználhatják az ***XCB*** előnyeit.

X-kiszolgálók

A ***FD.o*** az ***XFree86*** helyett két alternatívát támogat. Az első az ***XFree86 4.4-RC2*** kód egy mellékágaként indult, még a felhasználási szerződés megváltoztatása előtt. Ez a kiszolgáló az ***X.org***, és az ***XFree86***-tal azonos módon használható. A másik az ***Xserver***, hosszú távon ez tűnik a legígéretesebb választásnak. A ***Kdrive*** egy változata, amely több évvel ezelőtt szintén mint az ***XFree86*** egyszerűbb, erősen módosított kiadása indult. A ***Kdrive*** kisméretű, részben azért, mert kevés kódkettősséget mutat a rendszermaggal. A méretcsökkentést szolgálta néhány elavult szolgáltatás és illesztőprogram-modul eltávolítása is. Jóval kisebb méretének köszönhetően a ***Kdrive*** megfelelőbb alapot nyújt egy teljesen új kiszolgáló felépítéséhez.

Az ***Xserver*** ma elérhető változatát elsősorban tesztcélokra, az új bővítmények és szolgáltatások – mint például az átlátszóság vagy az ***OpenGL*** gyorsítás – kipróbálására használják. A memóriahasználatot úgy csökkentik, hogy sok számítást a program futási időben végez el ahelyett, hogy mindig a memóriában tartaná az eredményeket.

Az ***Xserver*** célja a sebesség növelése, és az egyéb olyan jelenségek (például villódzás) megszüntetése, amelyek miatt egyszerűen rossz ránézni a képernyőre. Egy új ***X***-bővítmény, a ***Composite*** lehetővé teszi a teljes képernyő kettős pufferelését. Természetesen egyetlen kiszolgáló sem lehet okosabb, mint a legbutább terminálja, ám az egyszerűbb felépítésnek köszönhetően könnyebb megtalálni és kijavítani a lassú kódrészeket, bárhol is legyenek. Az új kiszolgáló az eszközkészletek szintjén semmilyen hatással nem bír, hacsak a programozó nem akarja kifejezetten kiaknázni az új bővítmények által kínált lehetőségeket.

Cairo

A vektorgrafikánál a képek több-kevesebb bonyolult vonal, valamint az így kapott területek színekkel való kitöltése révén állnak elő. Az ilyen fájlok kisméretűek, és veszteség nélkül is tetszőleges felbontásra méretezhetők. Ez a megoldás tehát minden olyan felhasználó érdeklődésére számot tarthat, aki biztos akar lenni abban, hogy amit ki fog nyomtatni, az pontosan megegyezik azzal, amit lát. Sajnos az ***X*** bár képes kezelni a szövegek, négyszögek és egyebek képernyőn megjelenő bittérképeit, a nyomtatás és a vektorgrafika egyszerűen kimarad a látóteréből. Ez az egyik oka annak, hogy még jelenleg sem tökéletesen azonos az, amit a képernyőn látunk, amit a papíron kapunk, illetve amit a fájlokba elmentünk.

Az ***FD.o*** megoldása minderre a ***Cairo***, egy új ***2D*** vektorgrafikai könyvtár, mely többféle készüléken is biztosítja az egységes kimeneteket. Érthetőbben fogalmazva: minden adathordozón ugyanazt a kimenetet kapjuk. A külvilág felé a ***Cairo*** felhasználói szintű ***API***-kat biztosít, hasonlóan a ***PDF 1.4*** képkezelő modellhez.

Különféle háttérrendszerek segítségével a ***Cairo*** különböző kimeneti eszközöket képes támogatni. Az első eszközcsoportot a képernyők alkotják, ezek az ***Xlib*** vagy az ***XCB*** révén érhetők el. Ugyancsak fontos eszközt jelentenek a memóriában tárolt képpufferek, amelyeket fájlba lehet menteni vagy más alkalmazásoknak lehet átadni.

A ***PostScript*** és a ***PNG*** kimenet szintén megoldott, a ***PDF*** pedig tervezés alatt áll. Az ***OpenGL***-gyorsított kimenetet a ***Glitz*** nevű háttérrendszer fogja megvalósítani, illetve a ***Glitz*** önálló, ***OpenGL*** feletti rétegként is használható lesz.

A ***Cairo***-hoz nyelvi kötések ***C++***, ***Java***, ***Python***, ***Ruby*** és ***GTK+*** nyelvekhez léteznek.

Az ***OpenOffice.org*** fejlesztői az ***OOo 2.0***-ás változata után szintén tervezik a ***Cairo*** használatát, akár mint külön letölthető grafikai beépülő modulét. A ***Cairo*** jelenleg is fejlesztés alatt áll, ám mivel még nem rendelkezik tökéletesen üzembiztos ***AIP***-val, a hivatalos ***FD.o*** kiadásoknak egyelőre nem része.

D-BUS

A ***D-BUS*** egy bináris protokoll ***folyamatok közötti adatcsere (Interprocess Communication, IPC)*** céljára; itt a folyamatok azonos asztali munkamenet alatt futó alkalmazásokat jelentenek, illetve magát az operációs rendszert. A második esetbe tartoznak a felhasználóval folytatott párbeszédek is, amikor új hardverrel vagy szoftverrel bővül a számítógép.

A ***D-BUS*** belső világát ***Robert Love*** részletesen ismertette ***„Get on the D-BUS”*** című írásában, a ***Linux Journal 2005. februári számában***. A felhasználót tekintve a ***D-BUS***-nak legalább olyan szolgáltatásszintet kell biztosítania, mint amilyet a ***GNOME*** és a ***KDE*** is nyújt. Ez egy ***message bus (üzenetbusz)*** nevű rendszerdémon és egy felhasználónkénti, munkamenethez kapcsolódó démon futtatásával válik lehetővé. A ***D-BUS***-on keresztül bármely két program kapcsolatba léphet egymással, amivel a lehető legjobb teljesítményt lehet elérni. Hasonlóan a teljesítmény javítását szolgálja az is, hogy mivel a ***D-BUS***-t igénybe vevő programok szinte mindig azonos gépen futnak, ezért egyszerű ***XML*** helyett bináris formátumot használunk.

Az üzenetbusz démon lényegében útválasztóként üzemel. Mivel az alkalmazások között nem bájtfolyamokat, hanem üzeneteket továbbít, szolgáltatásai a munkaasztalról is elérhetők. Normál esetben a démon több, egymástól független példányban fut. Az egyik példány rendszerszintű kommunikációra szolgál, esetében szigorú biztonsági előírások szabják meg a fogadható üzenetek körét. A többi példány

az egyes felhasználói munkamenetekhez jön létre, és a bennük futó alkalmazásokat szolgálja ki. A ***D-BUS*** rendszerpéldánya biztonsági kockázatot is jelenthet, ugyanis a rootként futó szolgáltatásoknak képeseknek kell lenniük adatok és események cseréjére a felhasználói alkalmazásokkal. Éppen ezért eleve korlátozott jogosultságokkal való használatra tervezték, és ***chroot jailben*** fut. A ***D-BUS***-szal kapcsolatos biztonsági előírások az ***Fd.o*** webhelyén (lásd az internetes forrásokat) találhatók meg.

A legtöbb programozónak nem kell közvetlenül szólnia a ***D-BUS*** protokollhoz, gyakorlatilag bármely keretrendszerhez és nyelvhez léteznek megfelelő burkolókönyvtárak. Ez azt is jelenti, hogy mindenki megtarthatja megszokott környezetét, csak az ***IPC*** miatt nem kell váltani. A végfelhasználók szintén jól járnak, hiszen a ***KDE***, a ***GNOME*** és a ***Mono*** alapú programok az eszközkészlettől függetlenül képesek lesznek kapcsolatba lépni egymással. Ahogy a ***Cairo***, úgy ***D-BUS*** is hiányzik az ***FD.o*** első változataiból, ugyanis ***API***-ja még nem minősül üzembiztosnak. Mindettől függetlenül a fejlesztők a ***D-BUS***-t a jövőbeli kiadások egyik sarokkövének tekintik. A ***D-BUS*** az elképzelések szerint a ***KDE 4***-ben található ***DCOP*** helyét is át fogja venni.

Biztosan ez a jó megoldás?

Csak az idő múlásával adhatunk egyértelmű választ arra, hogy az ***Fd.o*** első megvalósításai elég jók-e, a lefektetett irányelvek pedig megállják-e a helyüket. Utóbbi alatt itt azt kell érteni, hogy olyasvalamit állítottak-e össze a fejlesztők, amely nulláról indulva újra megvalósítható – ha valakinek ehhez támadna kedve. Én biztos vagyok abban, hogy maga a megközelítés helyes, és több lehetőséget tartogat, mint a jelenleg létező „teljes értékű munkakörnyezet” megoldások bármelyike.

A leggyakrabban két aggállyal találkozom: 1) a jelenlegi asztali környezetek elveszítik önazonosságukat, egyetlen feladatuk a felhasználói felület biztosítása lesz 2) az ***FD.o*** nem szabvány, csak egy újabb megvalósítás. Az első felvetésre személy szerint visszakérdeznék: biztos, hogy ez baj? A legtöbb végfelhasználó sosem fogja észrevenni, de ha mégis, hát nem fog foglalkozni vele. Egyszerűen tudomásul veszik a korábban említett fejlesztéseket, és többet nem foglalkoznak a kérdéssel. Ha biztosítjuk az alkalmazások együttműködésének lehetőségét, tekintet nélkül azok fejlesztésének módjára, akkor a ***Linux*** a vállalati körökben is elfogadhatóbbá válik, és a zárt megoldások mellett szóló érvek közül egy újabb veszti érvényét.

Ha a protokollok és a formátumok többé nem meghatározott megvalósításokhoz vagy eszközkészletekhez kötődnek, akkor több asztali környezet között is meg lehet osztani őket. Ezzel a kód üzembiztossága és egyszerűsége terén csak nyerhetünk. A teljesen új programok fennakadások nélkül, azonnal kommunikálni tudnának a meglévőkkel. Bízom abban, hogy ez a szemlélet egyre nagyobb teret nyer, és egyre több alkalmazásfüggetlen szabvány kerül az ***FD.o*** szárnyai alá, ide értve a fájlformátumokat, a hangsémákat, a szín- és feladatbeállításokat egyaránt. Képzeljünk el egy levelezési beállító fájlt, amely tetszőleges levelező ügyfélprogrammal használható, az ***Evolution***-től egészen a ***mutt***-ig; vagy egy olyan könyvjelzőfájlt, amit a ***Mozilla*** és a ***Lynx*** egyaránt képes értelmezni.

Ami a második kifogást illeti – az ***FD.o*** nem szabvány, csak egy újabb megvalósítás –, nos, a szabad szoftverek és az ***RFC***-k világa pontosan így működik. Ha a szabályokat a kóddal együtt írjuk, akkor az elképzeléseket a lehető leghamarabb ki tudjuk próbálni a gyakorlatban is. Csak megemlíteném, hogy hasonló dolog történik az ***OO.o***-val és az ***OASIS*** irodai szabvánnyal. A fájlformátum a ***StarOffice***-on és az ***OO.o***-n belül született meg és finomodott ki, de ma már saját életet él. A bizottságban immár megtaláljuk a ***KOffice*** képviselőit is, és bármely jövőbeli irodai csomag használhatja natív formátumaként, akár nulláról indulva, kizárólag az előírásokat követve.

Persze ennek az útnak is vannak buktatói, de úgy látom, a fejlesztők tisztában vannak ezekkel, és képesek elkerülni őket. Megvan például a kockázata annak, hogy kizárólag ***Linux*** alatt működő szabványokat készítünk, a többi ***UNIX***-változatról elfeledkezünk. Hasonlóan ügyelni kell az erőforrás-használatra is, hiszen hiába a legvarázslatosabb elképzelés, ha kétszer annyi memória kell a zökkenőmentes működéséhez. Szerencsére – legalábbis egyelőre úgy látom – ilyesmitől nem kell tartani. Az viszont biztos, hogy itt az ideje csatlakozni ezekhez az erőfeszítésekhez. Kellemes buherálást!

Köszönetnyilvánítás

Köszönettel tartozom ***Waldo Bastian-nek***, ***Keith Packardnak***, ***Daniel Stone***-nak és ***Sander Vesiknek***, amiért magyarázataikkal segítették munkámat.

Linux Journal 2005. május, 133. szám

Marco Fioretti hardver-rendszermérnök, a szabad szoftverekkel mint EDA alaprendszerekkel és a hatékony asztali környezet létrehozására irányuló RULE Project vezetőjeként áll kapcsolatban. Marco Olaszországban, Rómában él a családjával.

KAPCSOLÓDÓ CÍMEK

XFree86: ➲ www.xfree86.org

X Protocol: ➲ www.x.org/X11_protocol.html

Libiconv: ➲ www.gnu.org/software/libiconv

Xlibs: ➲ www.freedesktop.org/Software/xlibs

Xserver: ➲ freedesktop.org/Software/xserver

Cairo: ➲ www.cairographics.org

Glitz: ➲ www.freedesktop.org/Software/glitz

D-BUS: ➲ www.freedesktop.org/Software/dbus

➲ www.linux.org.uk/~telsa/Trips/Talks/g3-swamp.html

A Perl Hibakereső

Perl kódunkba print utasításokat tűzdelve is kereshetünk hibákat, de egy teljes értékű hibakereső ennél sokkal több információt szolgáltat.

A hibakeresés minden nyelvben bosszantó, ám szükséges rossz, akár saját kódunkat javítjuk, akár másvalakiét, aki éppen a rendszerünkön dolgozik. Bármi, ami megkönnyíti a hibakeresést, komoly előnyt jelenthet. A *Perl* parancssoros hibakeresővel rendelkezik, amely jelentősen megkönnyítheti a hibakeresés folyamatát. Cikkünkben áttekintjük a hibakereső használatának alapjait és bemutatunk néhány hasznos trükköt.

Hibák elkerülése figyelmeztetések és Strict kulcsszó használatával

A *Perl* maga elképesztő mennyiségű hibát képes küszöbölni, pusztán azzal, hogy a programunk elején bekapcsoljuk a figyelmeztetéseket és a szigorú szabályértelmezést (`strict`). Amennyiben programunk tartalmazza a `use warnings;` sort egy tucatnyi általános hibát máris elfoghatunk. Megtalálhatjuk például a csak egyszer használt változóneveket, melyek rendszerint elírások, a beállításuk előtt felhasznált skaláris változókat, vagy a újradefiniált alprogramokat.
A hibakereső üzeneteket kicsit beszédesebbé tehetjük, ha a `use diagnostics;` sort is a kódba szúrjuk, amely minden egyes figyelmeztetéshez leírást is mellékel. A másik megoldás, ha a figyelmeztetések jelentésére rákeresünk a `man perldiag` paranccsal.
Amennyiben 5.6-os változatnál régebbi *Perlt* használunk , a figyelmeztetések használata helyett a -w kapcsolót kell alkalmaznunk a script első sorában, valahogy így: `#!/usr/bin/perl -w`.

Végül sok általános hibát elkaphatunk a `use strict;` kifejezéssel, amely tulajdonképpen néhány nem teljesen biztonságos programozói egyszerűsítést tilt le. A `strict` kulcsszó által bevezetett korlátozások a következők:

- A változókat használat előtt vagy definiálnunk kell a `my` illetve `our` kulcsszó valamelyikével, vagy importálni kell őket a `use vars` kulcsszóval, vagy a csomagnevet is tartalmazó teljes nevet kell használnunk.
- A csupasz szavak csak alprogramnevek lehetnek, karaktersorozatok nem (például: `$string = blabla;`).
- A hivatkozások nem lehetnek szimbolikusak; lásd a széljegyzetet.

Mint láthatjuk, a figyelmeztetések és a `strict` kulcsszó a *Perl* néhány olyan képességét korlátozzák, amelyeket egyébként jó célra is lehet használni, de gyakran sikerül velük visszaélni. Ezek a parancsok megkönnyítik a hibakeresést, hiszen a Perl maga kapja el a hibákat.

Szimbolikus hivatkozások

A szimbolikus hivatkozások abban különböznek a szokásos "kemény" hivatkozásoktól, hogy itt a változó egy másik változóra hivatkozik. A szimbolikus hivatkozások olyankor jönnek létre amikor a programozó egy karaktersorozatot használ azonosítónak. A következő két sor például alapesetben érvényes *Perl* kód:

```
$name = "username";
$$name = "da";       # $username beállítása
```

Az ilyen kóddal könnyen előfordulhat, hogy az értelmező azt hatja végre amit kiadtunk és nem azt amire gondoltunk. Könnyen használhatunk szimbolikus hivatkozást ott, ahol kemény hivatkozást akartunk megadni, illetve elkeverhetjük a hivatkozott változónevet, hiszen soha nem szerepel a kódban. Sokkal biztonságosabb és a ugyanezt a hatás érhető el, ha az ilyen változókat inkább hashben tároljuk majd a `strict` kulcsszó használatával bekapcsoljuk a szigorú változófigyelést.

Mi a print utasítások hátránya?

Sokan kérdezhetik, miért baj az, ha `print` utasításokat szórunk szét a kódunkban? Semmi probléma nincs ezzel a hibakereső technikával, csak éppen az interaktív hibakeresővel sokkal hatékonyabban tudunk dolgozni. Futtatáskor a program és a környezet valamennyi összefüggését vizsgálni tudjuk, nem csak azokat melyekre előre gondoltunk, és tisztábban láthatjuk pontosan mit is csinál a kód. Reményeim szerint a cikk végére mindenki egyet ért majd velem abban, hogy az a kevéske idő amit a hibakereső megtanulásába fektettünk bőségesen megtérül.

A hibakereső indítása

A hibakeresőt a parancssorból indíthatjuk el, a *Perlnek* átadva a `-d` kapcsolót :

```
perl -d filename.pl
```

Amennyiben `CGI.pm`-el készített ***CGI*** programot vizsgálunk, az átadandó paraméterek mellé egyszerűen tegyük be a `-d` kapcsolót is a parancssorba:

```
perl -d filename.pl param=value param2=value
```

A parancssoros megoldáson kívül a ***Perl*** hibakeresőt bizonyos ***IDE***-k (fejlesztői környezetek) részeként is használhatjuk, ilyen a ***GNU Emacs*** vagy a ***Activestate Komodo*** programja, valamint léteznek ***GUI*** hibakereső felületek is, például a ***ddd*** vagy a ***ptkdb***. A rövidség kedvéért e cikkben csak a parancssorral fogok foglalkozni, de az alapelvek a ***GUI*** hibakeresőben is hasonlóak lesznek.
Ha parancssoros hibakeresőt használunk, sokat segíthet a `Term::ReadLine` modul telepítése , ez ugyanis lehetővé teszi a lépkedést a parancs-történetben.
Következzen a cikkünkben használt példaprogram. Másoljuk a következőket egy ***sample.pl*** nevű fájlba:

```
#!/usr/bin/perl
use warnings;
use strict;
my $name = “Pengu”;
foreach (1..20) {
  &shout($name);
}
sub shout {
  my $name = shift;
  print “*** $name ***\n”;
}
```

Legfontosabb hibakereső parancsok

Az hibakeresés megkezdéséhez a következő hét alapvető parancsot kell ismerünk:

- ***s***: egy lépés végrehajtása a következő sorig, belelépve az alprogramokba.
- ***n***: egy lépés végrehajtása a következő sorig, átugorva az alprogramokat.
- ***r***: folyamatos végrehajtás, amíg az adott alprogramból vissza nem térünk.
- ***c <sor száma>***: folyamatos végrehajtás a megadott sorig.
- ***l <sor száma, tartomány vagy alprogram neve>***: forráskód listázás.
- ***x <kifejezés>***: a ***<kifejezés>*** kiértékelése és olvasható kiíratása.
- ***q***: kilépés a hibakeresőből.

Tesztprogramunkat lefuttatva a hibakeresőben próbáljuk ki a fentieket:

```
perl -d sample.pl
```

Most a hibakereső indulási információit kell látnunk:

```
Default die handler restored.
Loading DB routines from perl5db.pl version 1.07
Editor support available.
Enter h or h h for help or
man perldebug for more help:
main::(sample.pl:6):     my $name = “Pengu”;
  DB<1>
```

Ez a program indulása előtti állapot. Az utolsó előtti sorban a hibakeresés állapotáról olvashatunk hasznos információkat: láthatjuk, hogy a main csomagban vagyunk, a ***sample.pl*** fájl 6. sorában, végül megmutatja azt a sort amit éppen futtatni fogunk.
Az utolsó sorban láthatjuk a parancssort és a parancs sorszámát (amely a begépelt parancsokkal folyamatosan növekszik) valamint „kacsacsőröket”, melyek száma a beágyazott parancsokat jelzi. Ezzel egyelőre nem kell foglalkoznunk.
Gépeljük az `s` parancsot a parancssorba majd üssük le az ENTER billentyűt, ezzel végrehajtva a program egyetlen sorát:

```
  DB<1> s
main::(sample.pl:8):     foreach (1..20) {
  DB<1>
```

A parancs ismétléséhez üssünk ENTER-t és ismételgessük tetszés szerint, hogy lássuk a program valóban lépésenként fut. Valahányszor egy `print` utasítást hagyunk el, az kimenet a hibakereső adatokkal együtt megjelenik a képernyőn.
Most próbáljuk ki az alparancsokat átugró utasítást (n) és nyomjunk néhány ENTER-t. Ahogy végiglépkedünk a cikluson, láthatjuk, hogy az alprogram eredményét azonnal visszakapjuk, és nem lépkedünk végig az alprogram utasításain.
Most próbáljuk ki az alprogramból való visszatérést (`r`). De ne indítsuk el most rögtön, akkor ugyanis a program végéig fog futni, hiszen a főprogramból fogunk „visszatérni”. Először csináljunk néhány ismétlést az `s` paranccsal míg az alprogramba nem lépünk. Ezután az `r` parancsot kiadva a következőket kell látnunk:

```
  DB<1> s
main::(sample.pl:8):     foreach (1..20) {
  DB<1>
main::(sample.pl:9):       &shout($name);
  DB<1>
main::shout(sample.pl:13):         my $name =
➥shift;
  DB<1> r
*** Pengu ***
void context return from main::shout
main::(sample.pl:8):     foreach (1..20) {
  DB<1>
```

Figyeljük meg a `void context return from main::shout` sort. Ha a ciklusban lekérdeznénk a visszatérési értéket, itt megnézhetnénk. ***Perlben*** a függvények és az alprogramok a hívási környezettől (skalár, tömb vagy `void`) függően különböző értékeket adhatnak vissza. A ***Perl*** hibakereső egyik hasznos képessége az `r` parancs, amely megmutatja a hívó környezettípusát. Könnyen megtalálhatjuk vele bizonyos hibákat, például amikor konstans értéket várunk, de véletlenül az alprogramunk tömböt ad vissza.
Következik az `l` parancs. Próbáljuk ki:

```
  DB<1> l
8==>     foreach (1..20) {
9:         &shout($name);
10       }
11
12       sub shout {
13:        my $name = shift;
```

```
14:         print “*** $name ***\n”;
15        }
  DB<1>
```

Az `l` önmagában egy oldalnyi forráskódot listáz, a következő végrehajtandó sortól kezdve, szöveges nyíllal jelölve meg az éppen végrehajtandó sort. Kilistázhatunk egy tartományt is ha sorszámokat adunk meg: `l 200-230`. Továbbá, az alprogramok nevét megadva azok listáját is megkaphatjuk: `l shout.`

A `c` parancs egy adott sorig folytatja a végrehajtást, így előreugorhatunk a minket érdeklő tetszőleges pontra:

```
  DB<1> c 14
main::shout(sample.pl:14):  print “*** $name ***\n”;
  DB<1>
```

A parancssorba gépelve bármilyen *Perl* kifejezést végre tudunk hajtani, beleértve a futó kódot megváltoztat utasításokat is. Akár kézzel is beállíthatunk változókat a programban. Az `x` parancs kiértékeli, majd olvasható formában kiírja a kifejezés értékét. Valamennyi kimenet elé egy sorszámot fűz, minden visszafejthető elemet ***visszafejt (dereference)*** valamint a visszafejtés minden szintjét egyel beljebb kezdi. Példaképpen alább beállítottunk egy `@sample` nevű tömböt, majd kiírattuk:

```
  DB<1> @sample = (1..5)
  DB<2> x @sample
0  1
1  2
2  3
3  4
4  5
  DB<3>
```

Figyeljük meg, hogy a hasheket kulcsokkal és értékekkel együtt jeleníti meg, mégpedig soronként egyet. A hasheket a `\` jellel kezdve tudjuk helyesen megjeleníteni, ez a jel ugyanis a hasht hash-hivatkozássá alakítja így az helyesen fejtődik vissza. Ez a következőképpen néz ki:

```
  DB<4> %sample = (1 .. 8)
  DB<5> x \%sample
0  HASH(0x83d53bc)
   1 => 2
   3 => 4
   5 => 6
   7 => 8
  DB<6>
```

Ha elégedettek vagyunk az eredménnyel, lépjünk ki hibakereső gyakorlatunkból a `q` paranccsal.

További négy hibakereső parancs

Sok ember kizárólag a fenti parancsokat használja a *Perl* hibakeresőben. Amikor azonban már kényelmesen tudjuk használni őket, a következő négy paranccsal még hatékonyabbá tehetjük a hibakeresést, különösen ha objektum orientált kódot készítünk:

- */<minta>*: listázás a szabályos kifejezés következő egyezésénél.
- *?<minta>*: listázás a szabályos kifejezés előző egyezésénél.
- *S*: a programban hozzáférhető valamennyi alprogram és metódus listázása.
- ***m <objektum vagy csomag>***: az adott objektum vagy csomag összes metódusának listázása.

A `/` és `?` jelekkel előre és hátrafelé kereshetünk és jeleníthetünk meg kódrészleteket, tetszőleges karaktersorozat vagy szabályos kifejezés egyezését vizsgálva. A keresendő szöveg elé nem kell szóközt tenni:

```
  DB<6> /name
6:       my $name = “Pengu”;
```

Az `S` és `m` parancsok akkor lehetnek hasznosak, ha az elérhető alprogramokat szeretnénk megismerni: az `S` a programban elérhető összes alprogramot kilistázza. Ezek éppen fordított sorrendben jelennek meg mint ahogy a `use` vagy `require` kulcsszóval beillesztettük őket, és a hibakeresőből beillesztett alprogramokat is tartalmazzák így például a `Term::ReadLine`-t is. Az `m` parancs az objektumban vagy egy egész csomagban elérhető valamennyi metódust listázza.
Nézzünk egy példát:

```
  DB<7> use CGI
  DB<8> $q = new CGI
  DB<9> m $q
AUTOLOAD
DESTROY
XHTML_DTD
_compile
_make_tag_func
_reset_globals
_setup_symbols
add_parameter
all_parameters
[...]
```

Műveletek, Töréspontok és Figyelőpontok

A műveletek, töréspontok és figyelőpontok, még ennél is nagyobb vezérlést adnak a kezünkbe a hibakereső és a futó program felett. Ezeket elképzelhető, hogy szívesebben használjuk egy grafikus *Perl* hibakereső felület, például a ***ddd***, ***ptkdb*** vagy ***Activestate Komodo*** alól. A *Perl* hibakeresőjét sokan bírálják amiatt, hogy a parancsok kiadásához meg kell jegyeznünk a megfelelő parancs gyorskombinációkat, ráadásul a parancsokon belül további megjegyezendő gyorskombinációkkal kerülünk szembe.
Ráadásul *Perl 5.8* alatt a jobb belső összhang érdekében néhány billentyűkombinációt megváltoztattak. Gyakran előfordul, hogy az ember kénytelen 5.8-as és korábbi verziókat is használni, így aztán sokszor egyszerűbb ***GUI*** alatt dolgozni. A parancsokat a parancssor alapján mutatom be, az alapelvek természetesen mindenhol ugyanazok.
A ***művelet (action)*** célja, hogy kódrészletet szúrjunk be a programunkba a forráskód megváltoztatása nélkül. Hasznos lehet, ha a kód élesben fut, és ki akarunk próbálni valamit. Akkor is jól jön, ha éppen a hibakeresés közepén vagyunk, és nem szeretnénk egy változtatás miatt újraindítani az egész hibakeresőt.

Műveletet a következő formában adhatunk meg:
```
a <line-number> <code>.
```

Nézzünk egy példát:
```
DB<10> a 9 $index = $_;
```

A fenti sorral egy új parancsot adtunk a `foreach` ciklushoz, amely tárolja a folyamatosan növekvő indexszámlálót. Ha kilistázzuk a programot, a műveleteket tartalmazó sorok mellett egy a betűt láthatunk. A művelet mindig előbb hajtódik végre mint a sor amelyhez kapcsolódik. A beállított műveleteket az `L` paranccsal tudjuk listázni, törölni pedig úgy lehet őket, hogy az a és a sorszám után nem írunk semmilyen parancsot. Az eddigiek a Perl 5.6 és korábbi verzióira vonatkoztak; ***Perl 5.8*** alatt a műveletet az A parancs és a művelet sor száma törli.
A töréspontok és figyelőpontok folyamatos végrehajtás során adják vissza a vezérlés a hibakezelőnek, így például a korábban bemutatott `r` vagy `c` kiadása után. Akkor lehetnek hasznosak, ha éppen egy adott számú ciklusismétlődés után megjelenő problémát szeretnénk vizsgálni, és nem szeretnénk mindig kézzel végiglépkedni a cikluson.
Töréspontot adott sorra vagy alprogramra állíthatunk be és feltételhez is köthetjük. A töréspont megadása a b paranccsal történik a következő formában:
```
b shout
```

Ha kilistázzuk a programot, a `shout` alprogram első sorának sorszáma mellett láthatjuk a `b` betűt. A végrehajtás folytatásához üssük be a `c` parancsot, és az az alprogram belsejében fog megállni.
Amennyiben követtük az előző példát és a 9. sorban létrehoztunk a műveletet, be tudunk állítani egy olyan töréspontot, amely éppen a ciklus adott körbefordulásánál áll meg:
```
b shout ($index eq 8)
```

Könnyen elképzelhető milyen hatékonyak lehetnek ezek a műveletek és töréspontok, ha egy hosszabb, összetett feltételes kifejezéseket és külső adatforrásokat tartalmazó programban keresünk hibákat.
A töréspontokat a `L` parancs listázza, törölni őket ***Perl 5.6*** és korábbi verzióiban a `d` paranccsal lehet. ***Perl 5.8*** alatt a `B` utasítás törli a töréspontokat.
A figyelőpontokra talán még jobb elnevezés a kifejezésfigyelés. Ezek akkor állítják le a programot, ha egy adott kifejezés megváltozik. ***Perl 5.6*** alatt a `w` paranccsal a következő módon állíthatjuk be őket:
```
w $name
```

A figyelőpontokat az `L` listázza és valamennyit törölhetjük ha a `w` utasítást paraméter nélkül adjuk ki. ***Perl 5.8*** alatt a w paranccsal vehetünk fel és a W paranccsal törölhetünk figyelőpontokat.

A Perl hibakereső testreszabása

Az első dolog amit tudnunk kell, hogy a ***Perl*** hibakereső nem más mint egy ***Perl*** könyvtár, amely kihasználja a ***Perl*** értelmezőbe épített programhurkokat. Ha akarjuk akár a teljes hibakeresőt is lecserélhetjük, ha lemásoljuk valahová az eredeti állományt, majd kódunk `BEGIN` ciklusában hivatkozunk rá:
```
cp /usr/lib/perl5/5.6.1/perl5db.pl ~/myperl5db.pl
```

A kódunk elejére a következőt írjuk:
```
BEGIN { require "~/myperl5db.pl" }
```

A fenti megoldást például akkor érdemes használni, ha a hibakereső 5.6-os változatának formátumát és működését jobban kedveljük mint az 5.8-asét.
Ezen kívül a `-d` parancskapcsolóval is megadhatunk másik hibakeresőt. A 5.6 maga is tartalmazz a ***Dprof profiler***-t amely a hibakereső hurkokat használja. Ezt a következőképpen használhatjuk:
```
perl -d:DProf mycode.pl
```

Akár saját programunkban is használhatjuk a hibakereső hurkokat. Kódunkban közvetlen töréspontot helyezhetünk el a `$DB::single = 1;` változó beállításával ami például akkor jöhet jól, ha a BEGIN blokkban szeretnénk hibát keresni. Normál esetben ugyanis ez a blokk az előtt fut le, hogy a hibakeresőtől megkapnánk a parancssort. Ezen kívül a hurkokat arra is felhasználhatjuk, hogy egy adott kódot futtassunk bármely alprogram lefutásakor. Ezekről, és a többi hibakereső hurokról további érdekességeket tudhatunk meg a `perldebug` kézikönyvoldalból.
A hibakereső belső változókészlettel rendelkezik, amelyet szintén megtalálunk a `perldebug` kézikönyvoldalon. A változók átírására a saját könyvtárunkban, vagy az aktuális könyvtárban elhelyezett `.perldb` beállításállományt használhatjuk. A beállításfájlban található ***Perl*** kód a hibakereső indulásakor fut le. A következő utasításokkal például saját parancsokat készíthetünk:
```
$DB::alias{'quit'} = 's/^quit(\s*)/q/';
```

Ez a sor lehetővé teszi, hogy a `quit` begépelésével is kiléphessünk. A `perldebug` kézikönyvoldalain néhány további hasznos `alias` tippet találunk.
Számos hibakereső opciót állíthatunk be közvetlenül a hibakeresőben az `O` paranccsal. Én ezek közül csak egyet használtam, amely a lapozóprogramot változtatja meg:
```
O pager=|less
```

Ezzel a megoldással minden, egy lapnál hosszabb kiíratást kedvenc lapozóprogramunkon keresztül tekinthetünk meg, ha egy pipe karaktert írunk a kiadott parancsok elé.

Linux Journal 2005. március, 131. szám

Daniel Allen (da@coder.com)
Egy 1200-baudos modem, ingyenes helyi tárcsázási lehetőség és egy MIT vendég azonosító jóvoltából ismerte meg a UNIX rendszert, amikor ezek a dolgok még léteztek. 1995 óta elhivatott Linux felhasználó.
A Prescient Code Solutions, szoftvertanácsadó cég elnöke.

A KDE „Kiosk” módja

Ha a felhasználó véletlenül elrontja egy program beállításait, az nemcsak neki, de a belső támogatásnak is komoly bosszúság. Nézzük, hogyan lehet a fontosabb beállításokat megóvni a módosításoktól.

A *KDE* asztal egyik legfontosabb jellemzője, hogy teljes egészében testreszabott „felhasználói élményt” tud biztosítani. A legtöbb *KDE* program az asztali rendszer által biztosított szolgáltatásokra és beépülő modulokra támaszkodik, így egységes felhasználói felület jön létre, és a beállításokat is könnyű elérni. A felület egyik népszerű kiterjesztése a ***KDE „Kiosk” (kioszk, nyilvános terminál)*** módja, melynek révén a rendszergazda a végfelhasználók asztalának minden jellemzőjét meg tudja adni, és szükség esetén azt is megakadályozhatja, hogy a felhasználók módosítsák az előzetes beállításokat.
A *KDE* alkalmazások a ***Microsoft Windows INI*** fájljaihoz hasonló beállító keretrendszert használnak. Ennek egyik előnye, hogy a rendszergazda vagy a felhasználó közvetlenül, kézzel is könnyen át tud szerkeszteni egy-egy beállító fájlt. Az *INI* fájlok normál szövegfájlok. Több, névvel ellátott részre oszlanak, ezek mindegyike egy vagy több kulcs/érték párt tartalmaz. Az értékeket közvetlenül az alkalmazások adják meg és használják:

```
...
[CsoportNév]
kulcs=érték
kulcs2=érték2
...
```

A beállító fájlok sokféle helyre kerülhetnek, attól függően, hogy melyik terjesztést használjuk. Amikor egy alkalmazás megpróbálja megkeresni saját beállításait, akkor a keresést egy előre megadott sorrend szerint végzi. A beállító fájlok keresése során végiglátogatott könyvtárak listáját a `kde-config –path config` paranccsal tekinthetjük meg.
A könyvtárak fordított sorrendben jelennek meg ahhoz képest, ahogy tényleges bejárásuk történik. A keresési lista összeállítása az alábbi szabályok alapján történik:

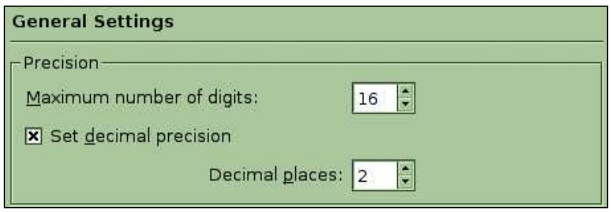
1. */etc/kderc*: Ebben a fájlban lehet megadni a bejárandó könyvtárakat.
2. `KDEDIRS`: Normál környezeti változó, a *KDE* alkalmazások számára a *KDE* könyvtárak és alkalmazások telepítési könyvtárát adja meg. A legtöbb esetben már bejelentkezés előtt megkapja értékét. A *KDE* telepítési könyvtára önműködően a lista végére kerül, ha még nem szerepel benne.
3. `KDEDIR`: Régebbi környezeti változó, használata a `KDEDIRS` megjelenése óta ellenjavallt. Ha a `KDEDIRS` kapott már értéket, akkor a `KDEDIR` nem jut szerephez.
4. A kérdéses futtatható fájlt tartalmazó könyvtár.
5. `KDEHOME` vagy `KDEROOTHOME`: Általában `~/.kde`. Előbbi minden felhasználónál létezik, utóbbi csak a root esetében.

1. ábra A Konqueror fő eszköztára feliratos módban

2. ábra A Konqueror fő eszköztára ikonos módban

A beállító fájlok a könyvtárfa */**share**/**config*** végződésű ágaiban találhatók, tehát például a `KDEHOME` környezeti változó értékéhez a */**share**/**config*** kerül hozzáfűzésre, amikor a rendszer összeállítja a beállító fájlt tároló könyvtár nevét.
Amikor egy alkalmazás saját beállításait kéri, a *KDE* az alkalmazáshoz tartozó fájlokat kutatva a fenti könyvtárak mindegyikét végigkeresi, majd egyetlen beállító objektumba egyesíti őket a program számára. A beállítások egyesítése kulcsonként haladva történik, ütközés esetén a kulcs az utolsóként beolvasott értéket kapja. Mivel a `KDEHOME` fájljai mindig utolsóként kerülnek beolvasásra, a helyi felhasználó által a fájlban végrehajtott módosítások felülbírálják az egyéb beállító fájlok által tárolt beállításokat. Ez az oka annak, hogy a `kde-config` parancs kimenetében a könyvtárak fordított sorrendben jelennek meg. Ez ugyanis a bennük elhelyezett beállító fájlok fontossági sorrendje.
A beállító fájlok többszintű kezelésének köszönhetően a rendszergazdák számos alapértelmezett beállítási értéket megadhatnak egy magasabb szinten, ezek egészen addig érvényben maradnak, míg a felhasználók meg nem változtatják őket. Ha például a rendszergazda be szeretné állítani az összes felhasználó alapértelmezett háttérképét, akkor valamelyik felsőbb szintű könyvtár ***kdesktoprc*** fájlját átírva

3. ábra A kcalc General Settings (Általános beállítások) párbeszédpanelje a pontossági beállítások zárolása után



4. ábra A kcalc General Settings (Általános beállítások) párbeszédpanelje a pontossági beállítások feloldása után

könnyen elérheti célját; beállítása addig marad érvényben, míg az egyes felhasználók felül nem bírálják:

```
[Desktop0]
...
Wallpaper=/usr/kde/3.3/share/wallpapers/
➥hatterkep.jpg
...
```

A ***KDE Kiosk*** módjának egyik lehetősége az, hogy képes a korábban beolvasott beállító fájlokban szereplő értékek zárolására, így a már megadott értékeket később nem lehet felülírni. Ez nemcsak arra jó, hogy a rendszergazdák előre megadjanak bizonyos beállításokat, de annak megakadályozására is alkalmas, hogy a felhasználók egyedi módosításokat eszközöljenek. A beállítási értékek ily módon történő zárolása rendkívül egyszerű.
Tegyük fel, a rendszergazda úgy szeretné rögzíteni a ***Konqueror*** beállításait, hogy a navigációs eszköztár mindig szövegesen jelenjen meg.
A `$KDEHOME/share/config/konquerorrc` fájlt átfutva rátalálhatunk a következő részre:

```
...
[KonqMainWindow Toolbar mainToolBar]
IconText=TextOnly
...
```

A fenti beállítás értelmében a ***Konquerornak*** a fő eszköztáron feliratokat kell megjelenítenie, és nem ikonokat. ***Konqueror*** alatt rendkívül könnyen meg lehet ezt változtatni, csak rá kell kattintani az egér jobb gombjával az eszköztárra, majd a ***Text Position (Feliratok helye)*** parancs kiválasztása után átválthatunk a kívánt megjelenítésre. Az *1.* és a *2. ábrán* a szöveges és az ikonos stílusú eszköztár látható.
Ha zárolni szeretné a beállítást, a rendszergazdának létre kell hoznia a ***konquerorrc*** fájlt az egyik magasabb szintű, beállításokat tároló könyvtárban, illetve módosítania kell azt. A megadott érték módosításának megakadályozásához az alábbiak szerint kell módosítani a fájlt:

```
[KonqMainWindow Toolbar mainToolBar]
IconText[$i]=TextOnly
```

A fentiekben a `[$i]` jelzi, hogy a beállítási érték módosítása tilos, vagyis a ***Konquerornak*** ezt az értéket kell alkalmaznia, ***nem*** veheti figyelembe az alsóbb szintű könyvtárakban található, normál esetben ezt a beállítást felülíró értékeket.
A könyvtárfa alsóbb szintjén található, a `[KonqMainWindow Toolbar mainToolBar]` szakaszt szintén tartalmazó beállító fájlok nem tudják felülbírálni az `IconText` értékét.
A fájl elmentése és a ***Konqueror*** újraindítása után bármilyen változtatást is eszközölünk a navigációs eszköztáron, az a ***Konqueror*** újraindításával elvész. Ennek oka az, hogy az érték egy felsőbb szintű könyvtárban zárolva van, ezért alsóbb szinten nem írható felül.
Ha nagyobb méretekben gondolkodunk, akár egész beállításcsoportokat is megvédhetünk a módosításoktól. Ha egy csoportot megváltoztathatatlanként jelölünk meg, akkor összes tagja is ilyen lesz. Példa a ***KCalc*** beállító fájljából, a ***kcalcrc***-ből:

```
...
[Precision][$i]
fixed=true
precision=12
...
```

Ha a ***kcalc***-ot a ***Precision (Pontosság)*** csoport módosításának letiltásával indítjuk, akkor a megfelelő értékek módosítása lehetetlen lesz. A *3.* és a *4. ábrán* látható a zárolt és a feloldott ***kcalc*** pontossági beállítások közötti különbség.
Arra is van lehetőség, hogy egy-egy alkalmazás beállító fájlját teljes egészében zároljuk, ehhez a [$i] jelölést a fájl elején kell elhelyeznünk. Ekkor a jelzés a fájl összes csoportjára és kulcs/érték párjára érvényes lesz. Ha a beállító fájlt ezzel a módszerrel védjük, akkor az adott alkalmazás beállításait illetően semmilyen módosítás végrehajtására nem lesz lehetőség.
Egy másik lehetőséget kínál az a tény, hogy ha egy ***KDE*** alkalmazás nem kap írási jogot egy beállító fájlra, akkor azt olyanként kezeli, amelynek módosítása tiltott. A fájlelérési engedélyeket közvetlenül is meg lehet adni a ***KDEHOME*** könyvtárban található fájlokra, így meg lehet akadályozni a felhasználókat a beállítások módosításában.
Ha például a ***kickerrc*** fájlt csak olvashatóként mentjük el, akkor elvesszük a felhasználóktól a ***kicker*** panel módosításának lehetőségét. Számos más ***KDE*** alkalmazás is így működik, igaz, ilyenkor ahhoz, hogy beállításaikat újra beolvassák, újra kell indítani őket.

Műveletek korlátozása

A rendszergazdák nemcsak a beállításokat védhetik a felhasználóktól, de bizonyos műveletek végrehajtásában is megakadályozhatják őket. Műveletnek hívunk minden olyan tevékenységet, amelyet a felhasználó végezhet, ilyen például a ***Fájl›Új*** parancs elérése. Mivel a legtöbb ***KDE*** alkalmazásban sok közös művelet szerepel, az előre

5. ábra A kicker panel módosíthatatlanként megjelölt beállító fájllal. Láthatóan kevesebb a benne futó alkalmazások testreszabását lehetővé tévő kezelősávot tartalmaz.

6. ábra A kicker panel alapállapotban

7. ábra A Konqueror a Súgó menüponttal

8. ábra A Konqueror a Súgó menüpont nélkül

megadott, normál műveletek zárolása könnyedén kivitelezhető. Az alkalmazásokon belüli műveletekre vonatkozó megszorítások beállítása a ***kdeglobals*** fájlban történik, mely szintén a korábban ismertetett könyvtárfában helyezkedik el.
Az alábbi kódrészlettel letilthatjuk a ***KDE*** alkalmazások fő menüsorában látható ***Súgó*** pont megjelenítését:

```
...
[KDE Action Restrictions][$i]
action/help=false
...
```

Egy másik lehetőség a Konqueror könyvjelzőinek letiltása. Ez a következő módon történik:

```
...
[KDE Action Restrictions][$i]
action/bookmarks=false
...
```

A műveleti megszorítások nem csupán menüpontokra vonatkozhatnak. Az alábbi kódrészlettel például a root hozzáférést igénylő szolgáltatásokat tilthatjuk le:

```
...
[KDE Action Restrictions][$i]
user/root=false
...
```

Rengeteg további lehetőségünk is van, ezek listája a ***kiosk*** mód leírásában található meg. Számos művelet minden ***KDE*** alkalmazásban egységesen szerepel. Az alkalmazások egy része ugyanakkor saját műveletekkel rendelkezik, ezek elérhetőségét szintén lehet korlátozni. Néhány érdekesebb ezek közül:

- ***print/system***: Megtiltja a nyomtatórendszer kiválasztását.
- ***shell_access***: Megtiltja a héj indítását.

9. ábra A Kiosk Admin Tool segítségével a rendszergazda kiosk beállításegyütteseket hozhat létre és telepíthet

- ***logout***: Megtiltja a felhasználók kijelentkezését.
- ***run_command***: Letiltja az ALT-F2-vel megnyitható parancs futtatása ablakot.
- ***lineedit_text_completion:*** Letiltja a sorszerkesztőknek azt a szolgáltatását, mely szerint rögzítik a korábbi beviteleket, majd részleges bevitelek kiegészítéseként felkínálják őket.

Egyéb erőforrások elérésének korlátozása

A beállító fájlokon kívül a ***KDE*** alkalmazások további erőforrásokat is igénybe vesznek a ***KDEDIRS*** könyvtárakból. A fenti példákhoz hasonlóan ezeket az erőforrásokat is ki lehet bővíteni a ***KDEHOME*** alá elhelyezett további erőforrásfájlokkal. A ***KDE*** az ilyen típusú fájlok elérhetőségének korlátozására is biztosít lehetőséget. A vonatkozó beállítások tárolása a ***kdeglobals*** beállító fájlban történik. Az alábbi ***kdeglobals*** kódrészlet például megakadályozza a felhasználót abban, hogy a felsőbb szintű erőforráskönyvtárak valamelyikében megadottaktól eltérő ikonkészleteket telepítsen és használjon:

```
...
[KDE Resource Restrictions][$i]
icon=false
...
```

A ***KDE*** által megadott erőforrások típusait az *1. táblázatban* foglaltuk össze.

A Vezérlőközpont használatának korlátozása

A ***Vezérlőközpont*** elemeit el tudjuk ugyan távolítani, ám a felhasználók ettől még hozzáférnek az illető beállításokhoz. A ***Vezérlőközpontra*** vonatkozó korlátozásokkal azonban biztosíthatjuk, hogy a felhasználók az átfogó érvényű beállítások jelentős részéhez ne férjenek hozzá.
A ***kdeglobals*** fájlban az alábbi részt elhelyezve megtilthatjuk a felhasználóknak a megadott vezérlőmodulok elérését.

1. táblázat *Erőforrástípusok*

Típus	Hely
apps	share/applnk
config	share/config
data	share/apps
exe	bin
html	share/doc/HTML
icon	share/icon
lib	lib
locale	share/locale
mime	share/mimelnk
pixmap	share/pixmaps
services	share/services
servicetypes	share/servicetypes
sound	share/sounds
templates	share/templates
wallpaper	share/wallpapers
xdgdata-apps	share/applications

A modulok teljes listáját a `kcmshell –list` paranccsal jeleníthetjük meg:

```
...
[KDE Control Module Restrictions][$i]
kde-crypto.desktop=false
kde-clock.desktop=false
...
```

URL-ekre vonatkozó korlátozások

A *KDE* lehetőséget kínál arra is, hogy szűrjük a *Konqueror* címsorába, illetve a *KDE* belső *URL* könyvtárainak közreműködésével más programokba beírt *URL*-eket. Ha az *URL* elérést megadott webhelyre szeretnénk korlátozni, az alábbi szakaszt kell beillesztenünk a *kdeglobals* fájlba:

```
...
[KDE URL Restrictions][$i]
rule_count=n
rule_1=open,,,,http,pelda.com,,false
rule_2=open,,,,file,,/mnt/share,false
rule_3=list,,,,file,,/mnt/cdrom,true
...
rule_n=...
...
```

A szabályok a következő formátumot követik:

```
rule_N=<művelet>,<protokoll>,<állomás>,
➥<elérési út>,<URL protokoll>,<URL állomás>,
➥<URL elérési út>,<enabled>
```

A határozott módon meg nem adott értékek mindennel egyezést mutatnak.

A fentiek közül az első szabály a `pelda.com` webhely elérésében akadályozza meg a felhasználókat. A második szabály értelmében a felhasználók a `/mnt/share` könyvtár fájljait nem nyithatják meg, és nem is menthetnek ide semmit. A harmadik szabály szerint a felhasználók a `/mnt/cdrom` könyvtárnak még a fájllistáját sem tekinthetik meg.
Az alábbi listával megadott tartomány *http*-n keresztül való elérését akadályozzuk meg, ezzel a *https* használatára kényszerítve a felhasználókat:

```
..
[KDE URL Restrictions][$i]
rule_count=2
rule_1=open,,,,http!,*pelda.com,,false
rule_2=open,,,,https,*pelda.com,,true
..
```

Az *URL*-ekre vonatkozó megszorításoknál az alapeljárás az, hogy az egyezés a hasonló nevű protokollokkal is létrejön, ezért egy a *http*-re megadott szabály a *https*-re is érvényes. A fenti példában a `http!` kulcsszóval biztosítottuk, hogy a szabály csak a *http* protokollra vonatkozzon, a *https*-re ne.

A KDE Kiosk Tool

Az utóbbi időben a *kiosk* környezet önműködővé tételével kapcsolatos munkálatok eredményeként elkészült a *Kiosk Admin Tool (felügyeleti eszköz, ➲ www.linuxjournal.com/article/7927)*. A programmal önműködővé lehet tenni a *KDE* által támogatott *kiosk* szolgáltatások egy részének felügyeletét. A rendszergazda a *Kiosk Admin Tool* segítségével a korábban említett elemek jelentős részét úgy szabhatja testre, hogy a beállító fájlokat nem kell kézzel szerkesztenie. A *Kiosk Admin Tool* alkalmazásával a rendszergazda többféle *kiosk* profilt is létre tud hozni, ezeket el tudja helyezni egy központi gépre, majd a beállításokat hálózaton keresztül, például *SSH* protokollon tudja terjeszteni. Bár az eszköz egyelőre nem ismeri az összes testreszabható értéket, későbbi változatai minden bizonnyal nagyobb szabadságot adnak majd a beállítások rögzítése terén.

Összegzés

A *KDE Kiosk* keretrendszere által biztosított fejlett beállító szolgáltatások révén az asztali „élmény” teljes mértékben testreszabható. Legyen szó akár több munkaállomás felügyeletéről, akár az új felhasználók alapértelmezett beállításainak megadásáról, a rendszergazdák több mint elegendő mozgásteret kapnak a kívánt eredmény eléréséhez. Írásomban a létező beállítási értékek csak egy elenyészően kis részére tudtam kitérni. Kísérletezgetéssel, tapasztalatok gyűjtésével bárki mélyebben is megismerheti a testreszabott asztali környezetek létrehozására használható beállításokat.

Linux Journal 2005. február, 130. szám

Caleb Tennis tervezőmérnök egy kis kutató-fejlesztő cégnél az indianai Columbus városában. Számos nyílt forrású tervezetbe kapcsolódott be, köztük a KDE, a Comedi és a Gentoo Linux fejlesztésébe. Ha az időjárás engedi, görkorcsolyázással vagy vízideszkázással piheni ki magát.

Reflektorfényben az OpenOffice.org

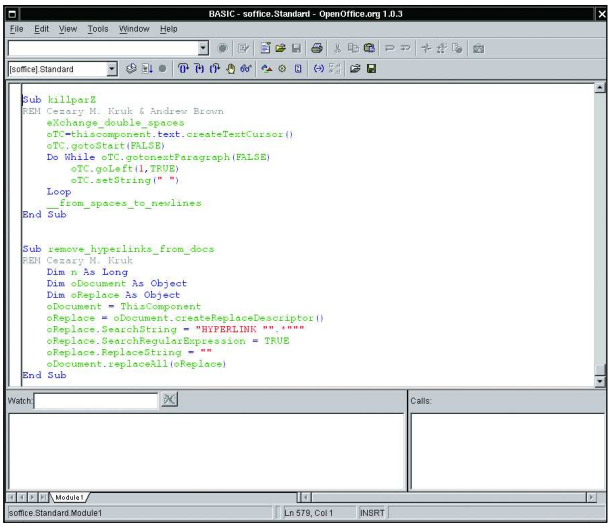
Vajon jól szolgál az OpenOffice.org Writer, mint szerkesztői eszköz egy többféle géptípust is használó kiadóvállalatnál?

Az *OpenOffice.org* egy nagyszerű alkalmazáscsomag, számos hasznos összetevőből áll, melyek számtalan lehetőséget kínálnak használójuknak. Testreszabható, és jó néhány nyílt dokumentumformátumot támogat. Ha az alapbeállításokat saját, egyéni igényeinkhez szeretnénk igazítani, az *OpenOffice.org* makrók és egyéb parancsfájlok készítésének lehetőségével van segítségünkre. Szerkesztőként dolgozom egy szabad szoftverekkel foglalkozó lengyel magazinnál. A szerkesztési folyamat kezdetén a szerző átadja a szöveget a szerkesztőnek, aki átszerkeszti azt. Az átszerkesztés a különféle tartalmi és formai hibák javítását jelenti, illetve a szöveg felkészítését, szabványos formába hozatalát a további lépésekben történő feldolgozáshoz. Ezután a korrektor feladata a szöveg további javítása, majd újra a szerkesztő lép a színre, aki elvégzi a végső módosításokat. Végül a szedő előkészíti a szöveget a nyomtatásra, majd a teljes anyag utolsó ellenőrzését újfent a szerkesztő végzi el. A szöveg minden egyes lépésnél más-más formát ölt. A mi kiadóvállalatunk a nyílt dokumentumformátumokat támogatja, ezért szerzőink ***HTML*** formátumban vagy egyszerű szövegként szállítják anyagaikat, a képeket pedig ***PNG*** vagy ***EPS*** formátumban mellékelik. Az anyag szerkesztése után a szerkesztő küld egy másolatot a szerzőnek, ez ***HTML*** formátummal történik. A korrektorok ***Microsoft Windows*** rendszereken dolgoznak, ***Microsoft Worddel***, ezért ők .doc formátumban igénylik az írásokat. A szedők Macintosh gépeket és ***QuarkXPresst*** használnak. Nekik kétféle dokumentumra van szükségük: a ***Microsoft Word*** fájlra a nyomtatáshoz és a cikkhez előírt formátum ellenőrzéséhez, valamint egy ***Macintosh*** szövegfájlra, amivel ***Quark*** alatt is tudnak dolgozni.

Negyedévente megjelenő kiadványunk 2000 őszén indult, akkor még ***StarOffice***-ot használtam. Azóta áttértem az ***OpenOffice.orgra***. A szerzők szövegfájljaival ***StarOffice*** és ***OpenOffice.org*** alatt nagyon hasonlóan tudok dolgozni. Korábban ***StarWriter***, jelenleg ***OpenOffice.org Writer*** alatt beimportálom a szöveges vagy ***HTML*** formátumban lévő dokumentumot, majd amikor végeztem vele, ***HTML***, ***Microsoft Word***, illetve ***SDW*** vagy ***SXW*** formátumba mentem.

Szöveg- és HTML fájlok importálása

Ha egy forrásfájl megfelelő formában jut el hozzánk, akkor a megnyitásával nem lehet gond. Ha egy fájl megsérült,



1. ábra A KillparZ makró végzi el a beimportált szövegfájlok előzetes feldolgozását

akkor helyre kell állítani. Figyelembe véve a dokumentumok nyílt formátumát, ez nem jelenthet különösebb kihívást. Miután beimportáltunk egy fájlt, a megfelelő formátumra kell alakítani. A lengyel, német, francia és egyéb nem angol nyelvű írások szerkesztőinek a kódlapot is át kell állítaniuk. A lengyel dokumentumoknál például a szabványos kódlap az ***ISO-8859-2***, viszont az összes OpenOffice.org dokumentum alap kódlapja az ***UTF-8***. Ha a beimportált dokumentumokat kényelmesen akarjuk átalakítani, akkor egy makróra lesz szükségünk. Az általam ***OpenOffice.org*** alá készített makrók több kódlap-átalakítót is tartalmaznak, például képesek az ***ISO-8859-2*** – ***UTF-8*** és fordított irányú átalakításra. A szövegszerkesztők némelyikében készült szövegfájlok bekezdései sokszor több sorba kerülnek. Összevonásukra a ***KillparZ*** makrót használhatjuk, ez az ***Andrew Brown*** által készített killpars továbbfejlesztett változata (*1. ábra*). A ***KillparZ*** az ***ooo-macro*** csomag egyik eleme.

Feltéve, hogy a szerző a megfelelő kódlapot adta meg, a ***HTML*** fájlok importálásakor sem lehet gond a karakterkészlettel. Az viszont kellemetlen meglepetésként érhet, hogy a ***HTML*** dokumentumok esetében a makrókhoz rendelt gyorsbillentyűk nem működnek. Ahhoz, hogy szóra

2. ábra HTML fájl OpenOffice.org alól kimentve. A fájlban stílusok, osztályok és egyéb nemkívánatos formázások szerepelnek.

3. ábra Ugyanaz a HTML fájl a soffice2html szűrővel átalakítva – jóval könnyebben olvasható és kezelhető formában

bírjuk őket, létre kell hoznunk egy üres *OpenOffice.org Writer* dokumentumot, meg kell nyitnunk a *HTML* fájlt, másolnunk kell a tartalmát, be kell zárnunk a *HTML* fájlt, majd be kell illesztenünk a tartalmat a *Writer* dokumentumba.

Kódlapok és DOC fájlok

Mivel a mi kiadványunk lengyelül jelenik meg, használnom kell a lengyel nyelvre jellemző írásjeleket, tehát kifinomultabb módszerre van szükségem a fájlok exportálásához.

A *StarWriterrel* és az *OpenOffice.org Writerrel* végzett próbáim során arra jutottam, hogy ha el akarjuk kerülni a nem angol nyelvű dokumentumokkal kapcsolatos gondokat, akkor *Type1* betűtípusok helyett *TrueType* típusokat kell használnunk. Sőt, a legjobb eredményre úgy juthatunk, ha a *Microsoft Word* formátumba kimentett dokumentumokat a *Microsoft Windows* alatt is használt betűtípusokkal exportáljuk. A *Microsoft FontPackben* szereplő *Microsoft* betűtípusok, így a *Times New Roman*, az *Arial* és a *Courier New* a legtöbb esetben bőségesen elegendőnek bizonyulnak.

A *StarOffice* és az *OpenOffice.org* készítőinek némi visszafejtést is kellett végezniük, csak így tudták feltérképezni a *Microsoft Word* formátumának felépítését. Emiatt a *Writer Word* formátumú mentésért felelős része nem működik tökéletesen. Ha tehát szabványos típusú dokumentumokat akarunk másoknak átadni, illetve ilyeneket szeretnénk tőlük fogadni, akkor készítsünk egy minden szükséges formázást – fejlécek, dőlt és félkövér betűk – tartalmazó mintát. Ezután adjuk oda a mintát munkatársainknak, és kérjük meg őket, ellenőrizzék, mindent jól látnak-e.

4. ábra A CHIP Special szerkesztői gárdája, balról jobbra: Robert Bielecki (szerkesztő), Romek Gnitecki (főszerkesztő), Cezary M. Kruk (CHIP Special Linux) és Tomek Borukalo (szerkesztő)

Az általunk kezelt írások egyszerű dokumentumok. Szerkesztőségünk a fent említett három betűkészletet használja, természetesen dőlt és félkövér betűket is alkalmazunk, továbbá kétféle fejlécet és egyszerű táblázatokat. A dokumentumokba nem szoktunk képeket illeszteni, egyszerűen csak felsoroljuk bennük a ***PNG*** vagy ***EPS*** formátumú képfájlok neveit. Ezeket a dokumentumokat ***SDW*** vagy ***SXW*** formátumból is gond nélkül lehet Microsoft Word formátumba kimenteni.

HTML formátum

A ***HTML*** formátumú dokumentumok előállítása már kicsit nehezebb feladat. A ***StarWriter*** és az ***OpenOffice.org Writer*** kétségkívül kifinomult ***HTML*** fájlokat állítanak elő, amint az a 2. *ábrán* is látható. Semmi nem akadályozza meg ugyanakkor, hogy ezeket a fájlokat egy egyszerű Perl parancsfájllal továbbalakítsuk. Nálam ez a parancsfájl a soffice2html nevet kapta. A parancsfájl elején a sorvégeket cseréljük le szóközökre, valahogy így:

```
s/\n/ /;
```

Ezután a kód bizonyos elemeit cseréljük le másokra. Például:

```
s/<(\/?)B>/<$1STRONG>/g;
s/<(\/?)I>/<$1EM>/g;
```

A fenti parancsokkal az összes `<B> ... </B>` és `<I> ... </I>` címkepárt lecseréljük `<STRONG> ... </STRONG>` és `<EM> ... </EM>` párokra, így a félkövér és a dőlt szakaszok jelölése szabványos lesz. Ezután vegyük ki a felesleges címkéket, például:

```
s/<EM><EM>/<EM>/g;
s/<\/EM><\/EM>/<\/EM>/g;
```

A sorvégek egy részét érdemes visszaállítani. Használjuk például a következő parancsokat:

```
s/(.+?)</$1\n</g;
s/>(.+?)/>\n$1/g;
```

Ezzel az egyes ***HTML***-címkék elé és mögé sorvégjeleket helyezünk el. Ha szakemberhez méltó parancsfájlt akarunk írni, talán még egy záró parancsot adjunk hozzá:

```
print OUT "<!- ", "soffice2html: ",
          scalar localtime, " ->\n";
```

A záró paranccsal egy az alábbihoz hasonló megjegyzés kerül a feldolgozáson átesett ***HTML*** fájlba:

```
<!- soffice2html: Wed Jul 23 17:34:35 2003 ->
```

Tehát, ha a dokumentum.sxw-t kimentjük, mint dokumentum.html-t, akkor ez utóbbit a `soffice2html dokumentum.html` paranccsal tudjuk feldolgozni. (*3. ábra*)
A ***HTML*** fájlok ilyen jellegű átalakításával érhetjük el a legjobb eredményt, vagyis szabványosabb, könnyebben olvasható kódhoz és 15-40 százalékkal kisebb fájlokhoz jutunk.
A `soffice2html` parancsfájl szintén megtalálható az ***ooo-macro*** csomag jelenlegi változatában.
Ha egy dokumentumból egyszerű ***Macintosh*** szövegfájlt akarunk előállítani, akkor a megfelelő karakterkészletet használó ***„Text Encoded”*** fájltípussal kell mentenünk.
A lengyel dokumentumoknál a megfelelő készlet az ***Eastern Europe (Kelet-Európa)***.
Ez a kimentési mód a hétköznapi feladatokhoz tökéletesen megfelel, de nyomdai célokra nem. Cikkeinkben sokszor szerepelnek a különféle műveletekhez tartozó billentyűkombinációk szimbólumai és egyéb különleges karakterek. Ha a normál eljárást követjük a Macintosh szövegfájlok előállításakor, akkor ezeket a karaktereket elveszítjük. Ahhoz, hogy megmaradjanak, egy makróra van szükségünk, amely az ***UTF-8*** kódlap szerinti karaktereket Macintosh kódlap szerintiekre alakítja. Az e célra használható makró, a `recode_utf_8_to_apple_macintosh` szintén az ***ooo-macro*** csomag része.
A makróval úgy készíthetünk szövegfájlt, hogy lefuttatjuk, majd a dokumentumot ***„Text Encoded”*** típussal mentjük, a ***System*** karakterkészlet és ***CR*** bekezdésjelek használatával. A fájlba így minden a szedő munkáját segítő és megkönnyítő információ bekerül.

Reflektorfényben

Az ***OpenOffice.org Writer***, mint szerkesztői eszköz segítségével elvégezhetjük a dokumentumok feldolgozását, megoszthatjuk őket a szerzők, a korrektorok és a szedők között – s mindez az összes résztvevő számára kényelmes, átlátszó módon történik. Mindössze a Writerre, néhány TrueType betűkészletre és makróra, valamint a letisztultabb HTML fájlokat előállító Perl parancsfájlra van szükségünk.

Linux Journal 2005. február, 130. szám

Cezary M. Kruk Lengyelországban, Wroclaw városában él. A negyedévente megjelenő lengyel CHIP Special Linux egyik szerkesztője.

LaTeX egyenletek és ábrák PHP-ben

Oldalainkba LaTeX tartalmat ágyazva be a ködös múltba száműzhetjük a matematikai egyenletek webes megjelenítésének nehézségeit.

Nyugodtan kijelenthetjük, hogy a weblogok és a wiki oldalak világát éljük. Bár újságok, általános szövegek vagy akár fényképek közzétételére ezek a rendszerek tökéletesen megfelelnek, ám ha egyszerű szövegek és képek kezelésénél többre van szükségünk, korlátaik hamar nyilvánvalókká válnak. A műszaki jellegű weblogok esetében különösen fontos a grafikonok, matematikai kifejezések, diagramok stb. megjelenítésének támogatása. Pusztán *HTML* alapon mindezt meglehetősen nehéz, ha nem egyenesen lehetetlen megoldani.
A külső alkalmazások, mint a *dia*, az *xfig* vagy a *Microsoft Egyenletszerkesztő* használata szintén bonyolult, hiszen először össze kell állítani az ábrát vagy egyenletet az adott alkalmazásban, majd kép formájában fel kell tölteni a weboldalra. Ha egy közös weblog egy másik szerkesztője módosítani szeretne egy ábrát, akkor neki is rendelkeznie kell az adott alkalmazással, illetve a kép készítésekor létrejött eredeti fájllal. Az ilyen megoldások természetes velejárója a bonyolultság, valamint az oldalakon megjelenő egyenletek és egyéb ábrák hullámzó minősége.
Írásomban bemutatom, hogyan lehet a *LaTeX*-et, ezt a kifejezetten műszaki dokumentumok készítésére fejlesztett szedőeszközt és -nyelvet *PHP*-ből hasonló igények kielégítésére használni. A *LaTeX*-et *PHP*-ből hívom meg, amikor a *HTML* tudása az összetett igények kezelésére elégtelennek bizonyul, az eredményeket pedig *PNG* képek formájában jelenítem meg; a *PNG* formátumot minden korszerű böngészőprogram támogatja. Mivel ilyenkor minden szükséges program a kiszolgálón található, minden szerkesztő és felhasználó ugyanazokat az eszközöket és csomagokat használhatja anyagai közzétételére.

Miért nem MathML?

A *W3C* szerint a *MathML* egy alacsony szintű *XML* szabálygyűjtemény matematikai anyagok leírására.
Bár a *MathML* emberi szem számára is olvasható, a legegyszerűbb esetektől eltekintve az *XML* kód előállítására egyenletszerkesztőt vagy egyéb segédprogramot kell használni. Mindemellett a jelenlegi böngészők a *MathML* nyelvnek csak egy részét támogatják, sok esetben azt is csak külső beépülő modul segítségével. Bár maga a nyelv valószínűleg ígéretes jövő elé néz, jelenlegi támogatottsága és használhatósága gyakorlatilag nulla.

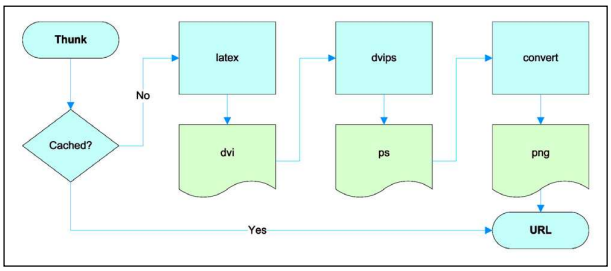
A helyzetet tovább árnyalja, hogy *Leslie Lamport LaTeX* szedőrendszere a műszaki és tudományos dokumentumok készítése terén gyakorlatilag szabvánnyá vált.
A *Donald Knuth* az 1970-es évek elejéről származó *TeX* dokumentumszervező rendszerére épülő *LaTeX* valamikor 1994-ben jelent meg. Mára kiforrott, közismert, elkötelezett felhasználói bázissal rendelkező, műszaki dokumentumok készítésére használt megoldássá vált. Természetesen szó sincs arról, hogy a *LaTeX* megismeréséhez elég volna egy róla szóló leírást a párnánk alá helyezni. Éppen ellenkezőleg. Ám akkor is tény, hogy a *MathML* jelenleg még nem képes megfelelő alternatívát kínálni a meglévő rendszerrel szemben.

Követelmények

A *UNIX* „írjunk egymással együttműködni képes programokat” filozófiáját követve ismert linuxos eszközöket fogok egymáshoz csatolni. A programok egymással együttműködve fogják előállítani a *LaTeX* forrás *PNG* formátumú megfelelőjét. A *LaTeX* egy újabb, a *dvips* és az *ImageMagick* eszközkészlettel kiegészített változatára lesz szükségünk. A végeredményt az *ImageMagick* készlet *convert* segédprogramjával fogjuk *PNG* képpé alakítani. Szerencsére a legtöbb héj hozzáférést is biztosító tárhelyszolgáltató eleve rendelkezik ezekkel az eszközökkel.

Áttekintés

A leképező rendszer egy szöveges karakterláncot vesz át, majd további feldolgozásra kiolvassa belőle a `[tex]` és `[/tex]` párok közé illesztett részeket. Ezeket a kinyert szegmenseket *thunkoknak* nevezzük. Ha egy *thunk* korábban már feldolgozásra került, vagyis a kódjának már van képi megfelelője, akkor helyére egy erre a képre vezető *URL* kerül. Ha új thunkról van szó, akkor átadásra kerül a *LaTeX*-nek, amelynek kimenete egy *DVI* fájl formájában jelenik meg. A *DVI* fájlt ezután az *ImageMagick* segítségével *PNG* fájllá alakítjuk, majd behelyezzük a gyorsítótár könyvtárba. Az újonnan létrehozott kép *URL*-jét az eredeti szövegben szereplő *thunk* helyére tesszük. Amikor az összes thunk feldolgozása befejeződött, a kapott szöveget átadjuk a hívónak. Az egyes thunkok átalakításának folyamatát az *1. ábra* szemlélteti.



1. ábra A thunkok leképezésének folyamatábrája

Használat

Azt hiszem, a legjobb az, ha felülről lefelé haladva először a leképező folyamat meghívásának módjával ismerkedünk meg, a megvalósítás részleteit hagyjuk későbbre. A motor szerepét egy egyszerű ***HTML*** felület tölti be, mely lehetőséget biztosít a ***LaTeX*** leképező rendszer tesztelésére. Segítségével láthatjuk, hogyan kell meghívni a ***render (leképező)*** osztályt. Kezdésnek az *1. kódrészletben* szereplő egyszerű sablont állítottam össze.

A fenti ***PHP*** alapú oldal egy űrlapot biztosít a ***LaTeX*** kód beírására, majd a `transform` eljárással a thunkokat a kész ***PNG*** képek ***URL***-jeire cseréli. Minden más művelet a háttérben, a `render` osztályban történik.

Alapszintű beállítások

A `render` osztály váza a *2. kódrészletben* található.

A ***PHP***-vel közölnünk kell, hogy eszközeink hol találhatók, valamint meg kell adnunk egy könyvtárat, ahova a ***PHP*** kiírhatja ideiglenes fájljait és gyorsítótárát. Az `URL_PATH` értéket szintén meg kell adni, erre a ***HTML***-ben szereplő image címkék előállításakor lesz szükség.

A látszólagos egyszerűség senkit ne tévesszen meg.

A kimenő ***PNG*** fájl módosítása céljából a ***LaTeX***-nek és az ***ImageMagicknek*** rengeteg beállítást lehet megadni, és ezekkel érdemes is megismerkedni. Itt csak a mindezt segítő keretrendszert mutatom be.

A wrap eljárás

A ***wrap (burkoló)*** eljárás fogja a ***LaTeX*** thunkot, majd bevezető és záró résszel látja el, így képez belőle érvényes ***LaTeX*** forrásfájlt. Az eljárás hasonló ahhoz, mint amikor külső eljárásokat illesztünk be egy C fájlba, vagy Java program készítésekor csomagok beemelésével bővítjük a nyelv tudását. (*3. kódrészlet*)

Mint látható, a ***LaTeX*** wrapper használata során általánosan szükséges csomagokat illesztem be. Ilyen az ***Amerikai Matematikai Társaság (AMS)*** csomagja, mely további matematikai elemeket tartalmaz, továbbá a vektoros grafikák leképezésére használható ***PSTricks*** csomag.

A `pagestyle` beállítása `empty` ***(üres)***, így a képekbe nem kerülnek oldalszámok. A thunk a `document` címkék közé kerül.

Lehetséges, hogy rendszerünkön a fenti csomagok egy része nem áll rendelkezésre. Ha alap ***LaTeX*** rendszerünk tudását növelni szeretnénk, akkor a ***Comprehensive TeX Archive Network (CTAN)*** weboldalról tölthetünk le további csomagokat. (Lásd az internetes forrásokat.) Szerezhetünk például oszlopdiagramok, ***UML*** jelölések és

1. kódrészlet render_example.php

```
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML
➥1.1//EN”

“http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd”>
<html>
<head>
<title>LaTeX egyenletek és ábrák PHP
➥alatt</title>
</head>
<body>
<!– a LaTeX kód beírására szolgáló űrlap –>
<form action=”render_example.php” method=”post”>
<textarea rows=”20”
          cols=”60”
          name=”render_text”></textarea><br />
<input name=”submit”
       type=”submit”
       value=”Leképezés” />
</form>
<?php
if (isset($_POST[‘submit’])) {
   echo ‘<h1>Result</h1>’;
   require(‘render.class.php’);
   $text = $_POST[‘render_text’];
   if (get_magic_quotes_gpc())
      $text = stripslashes($text);
   $render = new render();
   echo $render->transform($text);
}
?>
</body>
</html>
```

2. kódrészlet render.php

```
class render {
 var $LATEX_PATH = “/usr/local/bin/latex”;
 var $DVIPS_PATH = “/usr/local/bin/dvips”;
 var $CONVERT_PATH = “/usr/local/bin/convert”;
 var $TMP_DIR =  “/usr/home/barik/public_html/
                 ➥gehennom/lj/tmp”;

 var $CACHE_DIR = “/usr/home/barik/public_html/
                  ➥gehennom/lj/cache”;

 var $URL_PATH = “http://www.barik.net/lj/
                 ➥cache”;

 function wrap($text) { ... }
 function transform($text) { ... }
 function render_latex($text) { ... }
}
```

3. kódrészlet wrap.php

```
function wrap($thunk) {
s   return <<<EOS
    \documentclass[10pt]{article}
    % itt adhatjuk hozzá a további csomagokat
    \usepackage{amsmath}
    \usepackage{amsfonts}
    \usepackage{amssymb}
    \usepackage{pst-plot}
    \usepackage{color}
    \pagestyle{empty}
    \begin{document}
    $thunk
    \end{document}
EOS;
}
```

Karnaugh-térképek készítésére alkalmas csomagot. Bármire is van szükségünk, a keresést mindig kezdjük a gyűjteménnyel.

A render_latex eljárás

A `render_latex` eljárás (*4. kódrészlet*) az összes thunkot kigyűjti, majd egyenként dolgozza fel őket.
A thunk átadott érték szerepe egyértelmű: ez az éppen vizsgált ***LaTeX*** kódrész. A hash átadott érték a thunk ***md5*** kivonata.
Megváltoztatom az ideiglenes könyvtárat, majd a thunkot egy ideiglenes ***LaTeX*** fájlba írom. A ***LaTeX*** ezután létrehoz egy ***DVI*** fájlt. A ***LaTeX***-et a megfelelő parancssori kapcsolóval utasítom a nem interaktív működésre. A létrejött ***DVI*** fájlt a ***dvips*** segítségével ***PostScript*** formátumra hozom, eközben a -E kapcsolóval egy szövegdobozba illesztem. Ezután a ***PostScript*** fájlt átadom a `convert`-nek, mely ***PNG*** képet készít belőle. A `convert` segédprogramnak rendkívül sok kapcsolója van; az, hogy melyik beállításhalmaz a leginkább megfelelő, az adott webhelytől függ.
Végül, nem árt tudni, hogy az `exec` parancs egy hibajelző állapotkóddal tér vissza. A rövidség kedvéért a hibaellenőrzést itt elhagytam, és feltételeztem, hogy minden művelet sikeresen befejeződik. A ***LaTeX***-nek van néhány veszélyes kapcsolója is, amelyek használata egy többfelhasználós webkiszolgálón gondokat okozhat. Ha tehát bizonyos kulcsszavakat találunk a thunkban, inkább adjunk hibajelzést.

Amikor valami félrecsúszik

Ha a leképezési folyamat során valamilyen hiba történik, akkor próbáljunk kézzel átalakítani egy ***LaTeX*** fájlt. A hibakeresést az alábbi parancsokkal végezhetjük el:

```
latex –interaction=nonstopmode sajat.tex
dvips -E sajat.dvi -o sajat.ps
convert -density 120 sajat.ps sajat.png
```

Így kideríthetjük, hogy pontosan melyik lépésnél akad el a ***LaTeX*** leképezés.

4. kódrészlet render_latex.php

```
function render_latex($thunk, $hash) {
  $thunk = $this->wrap($thunk);
  $current_dir = getcwd();
  chdir($this->TMP_DIR);
  // ideiglenes LaTeX fájl létrehozása
  $fp = fopen($this->TMP_DIR . "/$hash.tex",
        ➥"w+");
  fputs($fp, $thunk);
  fclose($fp);
  // a LaTeX futtatásával ideiglenes DVI fájl
  ➥létrehozása
  $command = $this->LATEX_PATH .
             " –interaction=nonstopmode " .
             $hash . ".tex";
  exec($command);
  // a dvips futtatásával ideiglenes PS fájl
  ➥létrehozása
  $command = $this->DVIPS_PATH .
             " -E $hash" .
             ".dvi -o " . "$hash.ps";
  exec($command);
  // a PS fájl átadása a ImageMagicknek, amely
  // létrehozza a PNG fájlt
  $command = $this->CONVERT_PATH .
             " -density 120 $hash.ps
             ➥$hash.png";
  exec($command);
  // a fájl átmásolása a gyorsítótár könyvtárba
  copy("$hash.png", $this->CACHE_DIR .
       "/$hash.png");
  chdir($current_dir);
}
```

5. kódrészlet cleanup.php

```
function cleanup($hash) {
  $current_dir = getcwd();
  chdir($this->TMP_DIR);
  unlink($this->TMP_DIR . "/$hash.tex");
  unlink($this->TMP_DIR . "/$hash.aux");
  unlink($this->TMP_DIR . "/$hash.log");
  unlink($this->TMP_DIR . "/$hash.dvi");
  unlink($this->TMP_DIR . "/$hash.ps");
  unlink($this->TMP_DIR . "/$hash.png");
  chdir($current_dir);
}
```

A cleanup eljárás

A ***LaTeX*** leképezési folyamat során elég sok ideiglenes fájl jön létre. A `cleanup` ***(takarítás)*** eljárással törölhetjük ezeket – nem mintha valami bonyolult dologról volna szó. (*5. kódrészlet*)

6. kódrészlet transform.php

```
function transform($text) {
  preg_match_all("/\[tex\](.*?)\[\/tex\]/si",
  ➥$text, $matches);
  for ($i = 0; $i < count($matches[0]); $i++) {
    $position = strpos($text, $matches[0][$i]);
    $thunk = $matches[1][$i];
    $hash = md5($thunk);
    $full_name = $this->CACHE_DIR . "/" .
                 $hash . ".png";
    $url = $this->URL_PATH . "/" .
           $hash . ".png";
    if (!is_file($full_name)) {
      $this->render_latex($thunk, $hash);
      $this->cleanup($hash);
    }
    $text = substr_replace($text,
      "<img src=\"$url\" alt=\"Formula:
      ➥$i\" />",
      $position, strlen($matches[0][$i]));
  }
  return $text;
}
```

A transform eljárás

A *6. kódrészletben* látható `transform` eljárás a leképező osztályt irányítja, illetve nyilvános hozzáférési pontot biztosít a programozónak.

A *PHP* `preg_match_all` függvénye kigyűjti az összes thunkot, illetve ezek helyzetét. Ezután a hurokban sor kerül a thunkok egyenkénti feldolgozására. A következő lépés egy egyedi md5 kivonat készítése az adott thunk szövege alapján. Ebből tudjuk eldönteni, hogy egy adott thunk korábban már bekerült-e a gyorsítótárba. Ha igen, akkor meghívjuk a ***LaTeX*** leképező eljárást, és azonnal töröljük a létrejött ideiglenes fájlokat. A thunkot mindkét esetben egy ***URL***-lel helyettesítjük. Amikor az összes thunkot feldolgoztuk, a `text` változóval térünk vissza.

Egyenletkészítési példák

Nézzünk néhány példát a ***LaTeX*** segítségével előállítható egyenlettípusokra. Az egyenletek túlnyomó részét a ***Helmut Kopka*** és ***Patrick W. Daly*** által írt A ***Guide To LaTeX*** című könyvből vettem, ezt sokan az egyik legfontosabb ***LaTeX***-es alapműnek tartják.

$$\frac{a^2 - b^2}{a + b} = a - b$$

2. ábra Példa – törtek

```
[tex]
\begin{displaymath}
\frac{a^2 - b^2}{a + b} = a - b
\end{displaymath}
[/tex]
```

$$\mathrm{corr}(X,Y) = \frac{\displaystyle\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\left[\displaystyle\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2\right]^{1/2}}$$

3. ábra Példa – két változó, X és Y kapcsolata

```
[tex]
\begin{displaymath}
\mathop{\mathrm{corr}}(X,Y)=
\frac{\displaystyle
\sum_{i=1}^n(x_i-\overline x)
(y_i-\overline y)}
{\displaystyle\biggl[
\sum_{i=1}^n(x_i-\overline x)^2
\sum_{i=1}^n(y_i-\overline y)^2
\biggr]^{1/2}}
\end{displaymath}
[/tex]
```

$$I(z) = \sin(\frac{\pi}{2}z^2)\sum_{n=0}^{\infty}\frac{(-1)^n\pi^{2n}}{1\cdot 3\cdots(4n+1)}z^{4n+1} - \cos(\frac{\pi}{2}z^2)\sum_{n=0}^{\infty}\frac{(-1)^n\pi^{2n+1}}{1\cdot 3\cdots(4n+3)}z^{4n+3}$$

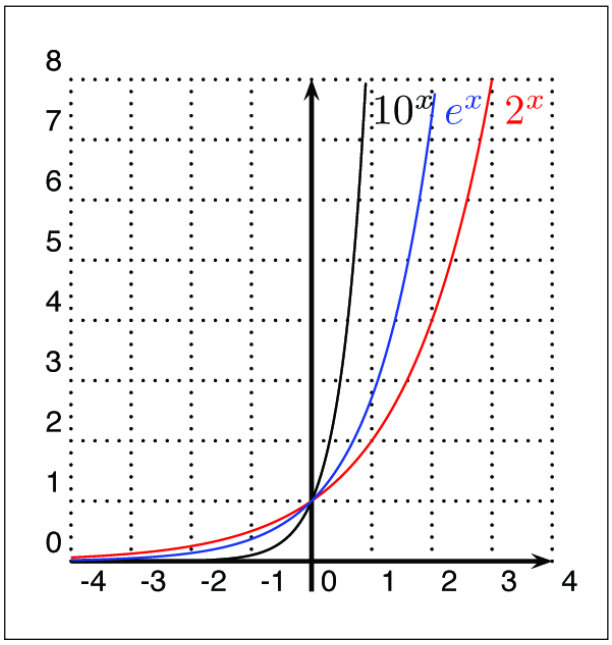
4. ábra Példa – összetettebb egyenlet

```
[tex]
\begin{displaymath}
I(z) = \sin( \frac{\pi}{2} z^2 ) \sum_{n=0}^\infty
    \frac{ (-1)^n \pi^{2n} }{1 \cdot 3
    \cdots (4n + 1) } z^{4n + 1}
    -\cos( \frac{\pi}{2} z^2 ) \sum_{n=0}^\infty
    \frac{ (-1)^n \pi^{2n + 1} }{1 \cdot 3
    \cdots (4n + 3) } z^{4n + 3}
\end{displaymath}
[/tex]
```

Grafikonrajzolási példák

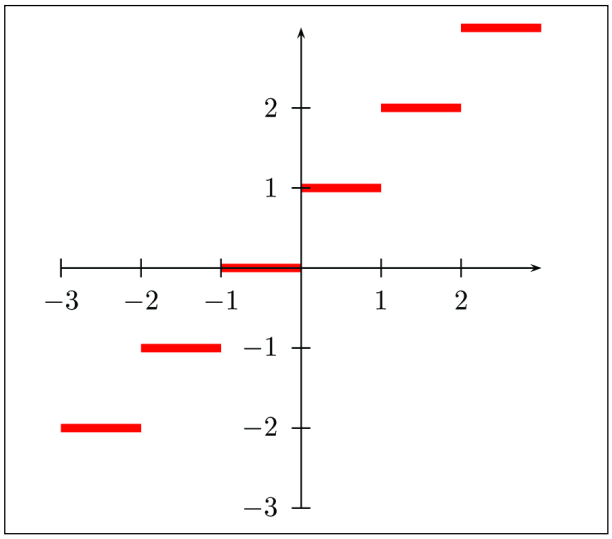
Bár a ***LaTeX*** elsősorban matematikai szedésre szolgál, kiegészítő csomagok segítségével, mint például a ***PSTricks***, más területeken is jól alkalmazható. A grafikonokat ***Herbert Vosstól*** kaptam. Weblapján (lásd a forrásokat) további példákat is lehet találni arra, hogy a ***PSTricks*** segítségével hogyan lehet tesztelni a ***LaTeX*** leképező rendszert. Összetettebb példáinak helyes megjelenítése, miért is tagadnánk, bizony komoly munkával jár.

```
[tex]
\psset{unit=0.5cm}
\begin{pspicture}(-4,-0.5)(4,8)
\psgrid[subgriddiv=0,griddots=5,
   gridlabels=7pt](-4,-0.5)(4,8)
\psline[linewidth=1pt]{->}(-4,0)(+4,0)
\psline[linewidth=1pt]{->}(0,-0.5)(0,8)
\psplot[plotstyle=curve,
   linewidth=0.5pt]{-4}{0.9}{10 x exp}
\rput[l](1,7.5){$10^x$}
\psplot[plotstyle=curve,linecolor=red,
```



5. ábra Példa – a 10^x, az e^x és a 2^x függvény grafikonja

```
   linewidth=0.5pt]{-4}{3}{2 x exp}
\rput[l](2.2,7.5){\color{blue}$e^x$}
\psplot[plotstyle=curve,linecolor=blue,
   linewidth=0.5pt]{-4}{2.05}{2.7183 x exp}
\rput[l](3.2,7.5){\color{red}$2^x$}
\rput(4,8.5){\color{white}change\normalcolor}
\rput(-4,-1){\color{white}bounding
box\normalcolor}
\end{pspicture}
[/tex]
```



6. ábra Példa – a Ceil függvény

```
[tex]
\SpecialCoor
\begin{pspicture}(-3,-3)(3,3)
   \multido{\i=-2+1}{6}{%
      \psline[linewidth=3pt,linecolor=red]
      (\i,\i)(! \i\space 1 sub \i)}%
      \psaxes[linewidth=0.2mm]{->}(0,0)(-3,-3)(3,3)
\end{pspicture}
[/tex]
```

A rendelkezésre álló megvalósítások

Jelenleg a weben számos *LaTeX* leképezőt lehet találni, ezek egy része jobban, más része kevésbé jól működik. *Steve Mayer* például jelenleg *Benjamin Zeiss* eredeti *PHP*-s *LaTeX* leképezőjét tartja karban. *Mayer* több beépülő modult is készített népszerű weblog rendszerekhez, többek közt a *WordPresshez*. Ha valaki moduláris megoldást keres a weboldalához, akkor ezt bátran tudom javasolni.
Érdemes még megemlíteni *John Walker textogif*-jét. Ez egy *Perl* program, mely a *LaTeX2HTML* eszközökkel, *CGI* alapon képezi le a képeket *GIF* vagy *PNG* formátumba. Szintén kiváló megoldás *John Forkosh* C-ben íródott, *CGI*-ként futó *mimeTeX*-e. Előnye, hogy használatához nincs szükség a *LaTeX*-re vagy az *ImageMagickre*, ám ennek a leképezési minőségben jelentkezik az ára.

Összefoglalás

A *LaTeX*-et *wiki* vagy *weblog* oldallal összeilleszteni első látásra rémisztő feladatnak tűnik. Ha viszont sikerül ráéreznünk a megoldásra, többé nem fogjuk érteni, korábban hogyan bírtuk ki nélküle. A fenti modellt követve azt is láthatjuk, hogy a *LaTeX* mellett más nyelveket hogyan tudunk beágyazni a *PHP*-be. Érdemes még megfontolni a *Gnuplot* használatát függvényábrázolások előállítására, míg az *Octave* segítségével komplex kifejezéseket tudunk kiértékelni, a *POV-Rayjel* pedig 3D ábrákat tudunk készíteni.
Jelenleg a weblogíró közösség által lefedett témakörök, viszonylagos sokszínűségük ellenére, eléggé egyoldalúnak mondhatók. A programozástól különböző területekkel foglalkozó műszaki szakemberek egyelőre távol maradnak a weblogírás világától, egyszerűen azért, mert a gondolataik átadásához szükséges eszközök hiányoznak. Bízom benne, hogy a webes *LaTeX* leképező rendszer segítségével sikerül majd áthidalni ezt a szakadékot.

Linux Journal 2005. március, 131. szám

Titus Barik kisvállalkozásokkal foglalkozó informatikai tanácsadó. Aktív weblogíró és műszaki könyvmoly. Weblogja a barik.net címen található.

KAPCSOLÓDÓ CÍMEK

- www.mayer.dial.pipex.com/tex.htm
- www.fourmilab.ch/webtools/textogif/textogif.html
- www.forkosh.com/mimetex.html
- www.pstricks.de
- www.imagemagick.org
- tech.irt.org/articles/js081

Útkeresés GpsDrive-al

Sok eszköz létezik, amellyel utunkat megtervezhetjük a térképen, jóval kevesebb amely megmutatja a barátaink helyzetét, több térképforrást is használhat, és akkor még nem is mondtunk el mindent.

Az egyiptomiak felfedezték a geometriát, a földmérés matematikai alapját. A Nílus éves áradásai eltörölték a jelzéseket így ezek a akkurátus bürokraták kénytelenek voltak újramérni az utak, mezők és más terepjellemzők határait.
Amikor a puskapor nyugati kezekbe került, hamar feltalálták a tüzérséget. Ehhez pedig szükség volt a tengeri és tüzérségi egységek valamint az ellenség helyzetének ismeretére. Így aztán a hadsereg régóta érdeklődik a dolgok helymeghatározása iránt és továbbfejlesztették az egyiptomiak által elkezdett módszereket.
Az 1970-es években, az ***Amerikai Védelmi Minisztérium (Department of Defense, DoD)*** elkezdte a ***Global Positioning System (GPS)*** rendszer fejlesztését. Ebben 24 műholdat helyeztek alacsony föld körüli pályára. A ***GPS*** azonnali pozíciót adott néhány tíz méteren belül. A Szovjetek megindították saját, hasonló rendszerüket, a ***Glonass***-t amelyet ***Oroszország*** ma is fenntart. Valamint az ***EU*** is megkezdte saját fejlettebb, ***Galileo*** nevű rendszerének kialakítását, amelyet 2008-ban indítanának be.
A hadsereg boldog; most már sokkal nagyobb pontossággal tudják eltalálni a célt. De akárcsak a másik ***DoD*** projekt, az internet esetében, a polgári alkalmazás haszna messze túlszárnyalja a katonai előnyöket. A ***GPS*** segítségével megtalálhatjuk az eltévedt hegymászókat, segíthetünk a végveszélybe került hajókon, és sokkal pontosabban és olcsóbban kereshetünk olajkutakat, mint a korábbi módszerekkel.
Az ***EU*** valójában elsősorban üzleti vállalkozásnak tekinti a Galileo rendszert.
Mindhárom rendszer a műholdakon elhelyezett atomórákon alapszik. A vevő az időjelek alapján tudja megmondani távolságát az egyes műholdaktól. A gömbi geometria kimondja, hogy három műhold, már megad egy pontot a síkon. Három dimenzióban már minimum négy műholdra van szükségünk. A modern ***GPS*** vevők akár 12 műholdat is követni tudnak, azaz éppen annyit amennyit egyszerre láthatnak.
Manapság a ***GPS*** által használt frekvencia és jelerősség egyben a legnagyobb korlátja a ***GPS*** vevőknek, ugyanis kizárólag szabadban vagy ahhoz nagyon közel használhatók, illetve külső antennával kell rendelkezniük, amely követheti a műholdakat.

Mi is a GpsDrive?

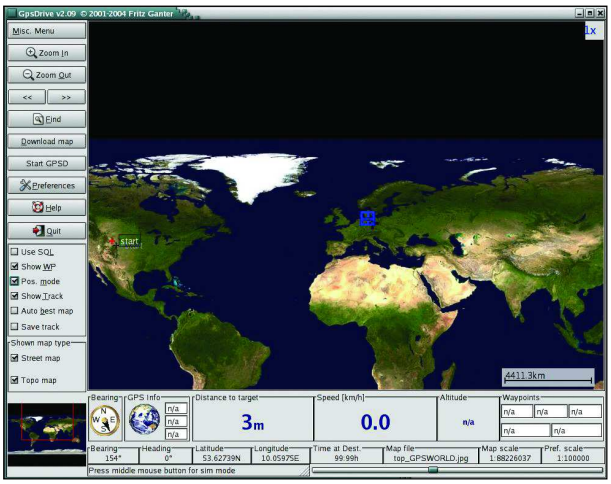
A ***GpsDrive*** egy ***GNU General Public License (GPL)*** védelem alá eső program, amely valós időben képes megmutatni valakinek a pozícióját. A legtöbb ***Linuxot*** futtató laptopon és ***PDA***-n képes működni, így például a ***Yopy*** vagy ***Zaurus*** gépeken is. Jelenleg 12 nyelvet támogat.
Mielőtt nekifognánk, egy fontos dolgot meg kell említenünk: soha ne tekintsük a ***GPS***-t másnak, mint az egyéb navigációs eszközök kiegészítésének. A ***GPS*** megjelenése nem jelenti, hogy ki kellene hajítanunk a ***Bowditch*** navigációs eszközünket.

Lássunk neki

A ***GpsDrive*** a ***Gnome Toolkit plus*** keretrendszert használja (***GTK+***), annak is a 2.2 vagy magasabb verziószámú változatát, amely a legtöbb ***Linux*** terjesztésben megtalálható.
Az élsimított fontkészletek szépen néznek ki ugyan, de nem szükségesek.
Az útvonalpontokat tárolhatjuk ***MySQL***-ben, a ***GpsDrive*** automatikusan használni fogja ha az elérhető.
A ***Kismet*** olyan drótnélküli-hálózat figyelő, amely képes a ***Wi-Fi*** elérési pontok felderítésére. Ahogy a ***Kismet*** rátalál ezekre, a ***GpsDrive*** automatikusan átalakítja a kapcsolatadatokat útvonalpontokká, és eltárolja őket az ***MySQL*** adatbázisban. Ezáltal a ***GpsDrive*** kiváló wardriving (drótnélküli-hálózat kereső) eszközzé válik.
A ***festival Linux*** alá készült emberi hangot előállító program. A ***GpsDrive*** ezt használja fel a megjegyzések kimondásához ahogy a útvonalpontok mellett elhaladunk. Ez kiváló biztonsági lehetőség a ***GpsDrive*** felhasználók számára.
A ***Flite*** a ***Festival*** lecsupaszított változata.

Telepítés

A ***GpsDrive*** könnyedén telepíthető a szokásos csomagtelepítési módszerrel.
A ***GpsDrive***-ot saját honlapjáról vagy az ott megadott tükrökről tölthetjük le (lásd a hálózati forrásokat). Töltsük le a legfrissebb stabil verzió tarlabdáit, ***md5*** összegeit illetve ***RPM*** csomagjait. A legfrissebb munkaközi szintű állományokhoz is hozzáférhetünk ***CVS***-en keresztül.
A tarlabda változat rugalmasabb, hiszen eltávolíthatjuk a nem kívánatos összetevőket.



1. ábra A fő ablak kinézete a GpsDrive első használatakor

A tarlabda telepítése során először is másoljuk azt egy tetszőleges helyre, majd tegyük a következőket:

```
tar -xvzf gpsdrive*tar.gz
cd gpsdrive
./configure
make
```

Amennyiben kizárólag a *NMEA* protokollt használjuk és nincs szükségünk a *GARMIN* protokollra, a következőképpen állítsuk be a *GpsDrive*-ot:

```
./configure --disable-garmin
```

Ha optimalizált fordítókapcsolókat szeretnénk az `--enable-auto-optimization` paramétert adjuk meg. Aztán root-ként telepítsük a programot a *gpsd* démont és a nyelvi állományokat a következő paranccsal:

```
make install
```

Az *RPM* változat a szokásos utat követi:

```
rpm -ivh gpsdrive*.rpm
```

Ha túl vagyunk a telepítésen, már el tudjuk olvasni a kézikönyv oldalakat, ahol a legfrissebb információkhoz juthatunk hozzá. Az első dolgunk hogy megvizsgáljuk, vajon működik-e a *GpsDrive* a *GPS* vevőnkkel. A rendszer kipróbálásához indítsuk be a *gpsd*-t, azaz a *GPS* adatokat kezelő démont. A */dev/gps* eszközön fog figyelni, hacsak másképpen nem utasítottuk a parancssor `-p` kapcsolójával:

```
gpsd -p /dev/ttyS1
```

Mivel a *GpsDrive*-ot és a *gpsd*-t nem root felhasználóként kell futtatnunk, bizonyosodjunk meg arról, hogy felhasználónak van írási és olvasási joga az eszközön.
Ha a *gpsd* már fut adjuk ki a következő parancsot:

```
telnet localhost 2947
```

Amikor megkapjuk a kapcsolat üzenetet, üssük le az *R* gombot, és a *gpsd* nyers *NMEA* mondatokat kezd el küldözgetni nekünk, valahogy ilyenformán:

```
[ccurley@charlesc ccurley]$ telnet teckla 2947
Trying 192.168.1.32...
Connected to teckla.
Escape character is '^]'.
r
GPSD,R=1
$PRWIRID,12,01.05,07/29/96,0003,*46
$GPRMC,235947,V,4333.1694,N,10812.0068,W,0.000,0.0,
➥120895,13.3,E*42
$PRWIZCH,00,0,00,0,00,0,00,0,00,0,00,0,00,0,00,0,
➥00,0,00,0,00,0,00,0*4D
ASTRAL
ASTRAL
$GPRMC,235949,V,4333.1694,N,10812.0068,W,0.000,0.0,
➥120895,13.3,E*4C
....
GPSD,R=0
^]
telnet> quit
Connection closed.
```

Mindez akkor is működik, ha a vevő semmilyen jelet sem kap, ugyanis a vevő ilyenkor is küld adatokat, jelezve, hogy nem fog semmit sem.
Ha már tudjuk melyik eszközön találjuk a *GPS* vevőnket, készítsünk egy közvetett hivatkozást (rootként) */dev/gps* néven hogy a *gspd* vagy a *gpsdrive* alapértelmezés szerint használni tudja:

```
ln -s /dev/ttyS0 /dev/gps
```

Az eszköznevet a *GpsDrive* felületén is beállíthatjuk, de a *gpsd* nem fogja ezt a beállítást használni.
Amennyiben az útvonalpontok tárolására *MySQL*-t szeretnénk használni, (ez a *Kismet* működéséhez elengedhetetlen), olvassuk el a *README.SQL* állományt. A *create.sql* állományt a *MySQL* parancssoros ügyfélfelületébe kell beolvasnunk, tehát megfelelő jogosultságokkal kell rendelkeznünk *MySQL* alatt. Bármilyen elfogadható *MySQL* ügyfelet használhatunk az útvonalpontok szerkesztéséhez, így akár az *OpenOffice.org*-ot is.

A GpsDrive beüzemelése

A *GpsDrive* és a tetszőleges kiegészítő programok telepítése és a GPS vevőnk működőképességének kipróbálását követően, próbáljuk ki a *GpsDrive*-ot is. Először egy bemutatóképernyőt látunk, majd a főablakot. Ezt követően először és utoljára egy emlékeztető ablakot láthatunk. A szerző *Fritz Ganter*, a weblapot futtató kiszolgálót saját zsebéből fizeti és nagyra értékelné a hozzájárulásokat.
Miután bezártuk az emlékeztetőt, egy képet találunk a *GpsDrive* ablak térkép részében. Ez tölti ki a térkép helyét amíg be nem szerezzük a magunkét. Az első dolgunk a szimulációs mód kikapcsolása a *Preferences* menüben. Ha már itt vagyunk, és angol vagy tengeri mérföldben szeretnénk látni az adatokat válasszuk azt az opciót.

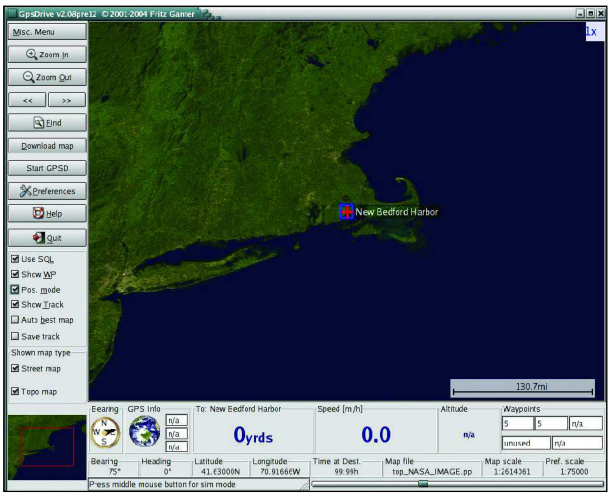
Az első térképünk letöltéséhez állapítsuk meg az új térképünk szélességét és hosszúsági fokát. Majd tegyük a programot pozicionáló módba (a menü bal-alsó részén). Készítsünk egy útvonalpontot az ***X*** gombbal, majd írjuk be a térkép középpontjának szélesség és hosszúság adatait. Dél és észak nyugat jelzésére használjuk a mínusz jelet (*1. ábra*).
A ***find (keresés)*** eszköz segítségével (bal felső menü) léphetünk az útvonalpontunkra. Kattintsunk a térkép letöltése bejegyzésre a főablak baloldalán. Figyeljük meg, hogy az általunk megadott szélesség és hosszúságértékek váltak alapértelmezetté. Válasszuk ki a méretarányt és a forrást, majd szedjük le a térképet. Bingo! Az új térkép azonnal megjelenik. Amennyiben ezt a helyet sokat használjuk, érdemes letöltenünk több, különféle méretarányú térképet.

GpsDrive üzemmódok

GpsDrive három üzemmódban működhet: pozicionálás, normál és szimulációs.
A pozicionálás módban mozoghatunk a térképünkön.
A pozicionáló módba a főablak bal alsó részén található ***Pos. mode*** bejelölésével léphetünk át. Amikor pozicionáló módban vagyunk, ahogy egy egérkattintással odébbugrunk a térképen, a ***GpsDrive*** megmutatja nekünk a jelenlegi helyzetünktől (ezt kék négyzet jelöli) mért távolságot és a irányt a célig (amit váltakozó kék és piros kereszt jelképez).
Például, ha van egy kis méretarányú térképünk egy nagyobb területről, körbejárhatunk és letölthetjük a kiválasztott nagy méretarányú térképeket az érdekesnek tűnő részekhez. Pozicionáló módban útvonalpontokat is megadhatunk.
Normál módban a ***GpsDrive*** a ***GPS*** vevővőtől kapja a pontot és a vevő által megadott pozíciót követi. Ha a pozíció változása szükségessé teszi, a ***GpsDrive*** odébb tologatja a rendelkezésre élló térképeket. A ***GpsDrive*** normál módban indul.
Szimulációs módban, a ***GpsDrive*** útvonalat hoz létre az indulási ponttól kezdve egy vagy több végpontig. Szimulációs módba kapcsoláshoz hozzuk be a ***Preferences*** menüt, lépjünk az első beállításfülre és jelöljük be a szimulációt. Igen szórakoztató mód, hiszen megnézhetjük amint egy képzeletbeli jármű nagy sebességgel keresztülszáguld az országon.

Térképek letöltése

Nyilván több térképet is szeretnénk különféle méretarányban. Javaslom szerezzünk be egy nagyon kis méretarányú térképet, amely lefedi az összes szokásos utazási útvonalunkat. Ha ez már a helyén van, nem fogunk leesni a térképünkről, ha véletlenül kikattintunk a területről pozicionáló módban. A ***NASA*** térképei (amennyiben van elég helyünk) vagy az alapértelmezett térkép kiválóan megteszi.
A ***GUI***-ban, egyszerűen válasszuk ki a térképünkhöz szükséges paramétereket, a kiszolgálót és töltsük le. Ez az egyszerű út. Azonban elképzelhető, hogy az eredmény nem igazán illeszkedik jól. Az ***Államok*** geológiai felmérési térképeit a ***topozone.com***-ról, az utcatérképeket az ***expedia.com***-ról szerezhetjük be.
Amennyiben ismerjük a középpont hosszúság és szélességi koordinátáit, valamint a méretarányt nincs más dolgunk mint begépelni azokat a letöltési űrlapba. Dolgozhatunk pozicionáló módban is, ahol a meglévő térképekre kattintgatva megtalálhatjuk a kívánt térkép középpontját és letölthetjük.
Ezen kívül ott vannak még a ***NASA*** topográfiai adatai.
A ***README.nasamaps*** állományban olvashatjuk a részleteket a *2. ábrán* pedig bemutatunk egy példát.
Ha rendszerezettebb térképgyűjtést szeretnénk, vessünk egy pillantást a ***gpsfetchmap.pl*** programra.



2. ábra A New England Déli része a GpsDrive-ban, a NASA topográfiai adatainak felhasználásával

Jogi kérdések

Néhány tréképi forrás jogvédett adatot tartalmaz. Ügyeljünk rá, hogy az adatokat csak a honlapon leírt engedélynek megfelelően használjuk.

Saját térképek bevitele

Saját térképeinket is felvihetjük. Ehhez ismernünk kell a térkép középpontjának hosszúság és szélesség értékeit, valamint a térkép méretarányát. A ***GpsDrive*** ablakának bal felső sarkában található ***Misc.*** menü alatt találunk egy druidát amely segít nekünk a térképek bevitelében.

A GpsDrive használata

Miután szereztünk néhány térképet, akár el is kezdhetünk játszani az új játékunkkal.
A ***GpsDrive*** jól el van látva ***Eszköztippekkel (Tool tips)***, ezért itt csak a legérdekesebb elemeket mutatjuk be.
A főablakban közvetlenül a térkép alatt találjuk a navigációs adatokat. A következő útvonalpont távolságán és a jelenlegi sebességünkön nincs mit magyarázni. Ezektől jobbra információkat találunk az útvonal pontokról, a barátaink kiszolgálóján látható mozgó célpontokról, és a ***GPS*** vevő szerinti aktuális időről.
Az útvonalpont távolság kijelzőjének bal oldalán találjuk a ***GPS*** információkat. ***GPS*** nélkül csak egy forgó földgömböt láthatunk itt. Amennyiben csatlakoztattuk a ***GPS***-t, a földgömb helyén a látható műholdak jelerősségének mértékét láthatjuk. A háttér piros színű ahol nincs fixpont; illetve zöld ha szolgáltat pontot.
A ***GPS*** adatoktól balra találjuk az iránytűt. Az iránytű teteje a jelenlegi haladási illetve a hajózási irányunkat mutatja. A fekete mutató a következő útvonalpont irányát jelzi.

A főablak bal oldalán található ***Preferences*** menüben rengeteg beállítás között választhatunk. A mérési egységünk kiválasztását már megismerhettük. Amennyiben régebbi számítógéppel dolgozunk, korlátozhatjuk a ***GpsDrive*** által felhasznált ***CPU*** időt, ha kikapcsoljuk az árnyékokat, melyek kirajzolása többletteljesítményt igényel.
A második beállítás fülben néhány ***GPS*** beállítást találunk. Például beállíthatjuk, hogy a ***GpsDrive*** közvetlenül a ***gpsd*** kikerülésével érje el a vevőnket.
Az ***SQL*** fülön kiválaszthatjuk, mely különféle útvonaltípusokat szeretnénk bevenni illetve kihagyni a megjelenítésből. Ezáltal az útvonalpontokat kategóriákba csoportosíthatjuk és eldönthetjük, melyeket szeretnénk megjeleníteni.
Én például ezt használom a kedvenc töltőállomásláncom útvonalpontjainak megjelenítéséhez. Ki és bekapcsolhatom őket a képernyőn, attól függően, hogy éppen szükségem van-e benzinre vagy nincs.
A térképek beszerzése után számos vezérlő áll rendelkezésünkre a kezelésükhöz. Azokról a területekről ahol sokat utazunk valószínűleg több különféle méretarányú térképpel is rendelkezünk. Ezek között többféleképpen is választhatunk. Az első módszer szerint a bal menü alsó részén az ***„Auto best map”*** pontot választjuk. Ezzel arra utasítjuk a ***GpsDrive***-ot, hogy mindig az adott területhez rendelkezésre álló legjobb (legnagyobb méretarányú) térképet válassza ki számunkra. Ez alatt, közvetlenül a terület térképe felett, válthatunk az utca vagy topográfiai térképek között, illetve kiválaszthatjuk őket egyszerre is. Ha mindkettőt kérjük, a ***GpsDrive*** a két típus között váltogat, ahogy a rendelkezésünkre álló térképekhez a legjobb lefedettséget biztosítja.
Amennyiben az ***„Auto best map”*** funkciót kikapcsoljuk többféleképpen is kiválaszthatjuk a méretarányt. A főablak bal felső részén, találunk néhány nyilat. A balra mutató nyíl segítségével választhatunk nagyobb méretarányt, a jobbra mutató nyíllal pedig a kisebb méretarányt. Az jobb alsó részen látható csúszka segítségével ugyanezt a hatást érhetjük el. A kívánt méretarány beállítása után a ***GpsDrive*** a lehető legközelebb próbál maradni ehhez a mérethez.
Egy adott térképen belül tetszés szerint nagyíthatunk és kicsinyíthetünk a főablak bal felső részében található két nagyítóüveg ikon használatával. Az aktuális nagyítási arányt a főtérkép jobb felső sarkában találjuk. A ***GpsDrive*** térképváltások során megtartja a nagyítási arányt, ami elég lehangoló.
Először is nézzük meg bekapcsoltuk-e az útvonalpontokat, illetve használjuk-e az ***SQL***-t vagy nem.
Az útvonalpontokat többféleképpen is beállíthatjuk. Szerkeszthetjük őket kézzel egy szövegállományban vagy tárolhatjuk ***MySQL*** adatbázisban, a ***gpsbabel*** program segítségével, átalakíthatjuk más állományformátumokról valamint akár a ***Wayhoo.com*** oldalairól is letölthetjük őket.
Pozicionáló módban a jelenlegi helyzetünk útvonalpontját az ***X*** gombbal tudjuk rögzíteni, illetve az Y gomb segítségével az aktuális egérkurzor pozíciónál rögzíthetünk útvonalpontot. A paramétereket rögzítés előtt minden esetben módosíthatjuk.

Hálózatkeresés GpsDrive-al

A ***hálóvadászat (wardriving)*** nevű „sport” lényege, hogy vezetés közben ***Wi-Fi*** elérési pontokat keresünk. (További információkat találunk a ***Linuxvilág*** egy korábbi, a drótnélküli hálózatok felkutatásáról szóló cikkében.)

No és a barátok?

A ***GpsDrive*** barátkiszolgálóval együtt érkezik. Ennek segítségével az ismerősök nyomon követhetik egymás pozícióját a rendszereinken. Beállíthatjuk a sajátunkat, vagy használhatjuk bárki másét, a nyílt Interneten. Több jármű mozgásának valós idejű tervezéséről van szó. Ezáltal válik a ***GpsDrive*** igazán hasznos segítőtárssá egy autóversenyen vagy egy mentőakció során.
Amennyiben a felhasználó jelvesztés miatt ideiglenesen leszakad a hálózatról ***Wi-Fi***, a felhasználó utolsó ismert pozícióját látjuk. Amikor visszatér a hálózatra, a pozíciója másodpercek alatt frissül.

Ami hiányzik a GpsDriveból

Az egyetlen dolog ami hiányzik a ***GpsDrive***-ból, az utca szintű keresés. Ehhez ugyanis nyílt forrású utcaszintű adatokra lenne szüksége. Az üzleti adatok 10,000 eurós nagyságrendben mozognak, ami hamar letöri a kedvünket. Aki esetleg tudna egy ilyen adatforrást, kérjük tudassa a szerzővel.

Nyelvi támogatás

A ***GpsDrive*** fordításokra van szüksége, különös tekintettel a ***Festivalra***. Jelentkezők?

Összefoglalás

A ***GpsDrive*** kiváló eszköz, ha egy vagy több ***GPS*** vevő helyzetét szeretnénk megjeleníteni, valós időben. Több feladatra is használható, olyan mókás céloktól kezdve mint a vasárnapi felfedezőút követése, egészen a komoly mentőmunkálatokig.

Linux Journal 2005. április, 132. szám

Charles Curley (www.charlescurley.com) Linuxot oktat a Wyoming-i főiskolán. Ezen kívül programokat cikkeket és könyveket is ír, olyan nyílt forrású eszközök segítségével mint az Emacs.

KAPCSOLÓDÓ CÍMEK

Expedia: ➲ www.expedia.com

Festival: ➲ fife.speech.cs.cmu.edu/festival

gpsbabel: ➲ gpsbabel.sourceforge.net

GpsDrive: ➲ www.gpsdrive.cc
➲ www.gpsdrive.cc/download.shtml

Kismet: ➲ www.kismetwireless.net

OpenOffice.org: ➲ OpenOffice.org

Six of One: ➲ www.netreach.net/~sixofone

Topozone.com: ➲ topozone.com

Wayhoo.com: ➲ wayhoo.com/index

Rajt! – UHU-Linux Office 1.2

2005 márciusában, az előző verziót követő egy éves fejlesztési időszak befejeztével az UHU-Linux Kft. kiadta az UHU-Linux Office 1.2-es változatát, mely a „Rajt!” kódnevet kapta. Ezt mutatja most be Koblinger Egmont, a cég egyik fejlesztője.

Célunk egy olyan disztribúció fejlesztése és tökéletesítése, amely a kezdő felhasználók számára is könnyedén és hatékonyan használható általános, mindennapos otthoni és irodai feladatokra, valamint kiemelkedően támogatja a magyar nyelvet és a magyar piac egyes speciális igényeit.
Ugyanakkor fontosnak tartjuk, hogy szakmai szemmel nézve is jó minőségű, „rendesen összerakott”, élvonalbeli megoldásokat tartalmazó rendszert nyújtsunk át felhasználóinknak, melyet a haladók könnyű szerrel használhatnak fejlesztésre, szerverüzemeltetésére vagy egyéb kevésbé szokványos feladatokra.
A gyakori félreértések miatt azonban leszögezném, hogy rendszerünket nem azoknak szánjuk, akik egy operációs rendszer összerakásának technikai részleteit szeretnék elsajátítani, avagy saját egyéni ízlésük és hitük (nem ritkán tévhitük) szerint testre szabni a rendszer egyes mélyebben fekvő részeit. Az ***UHU-Linux*** nem egy barkácsolási hajlamok kiélésére szánt „rakd össze magadnak” típusú rendszer, hanem mi igyekszünk a legjobb tudásunk szerint összerakni és így egy előre elkészített, tesztelt egészként nyújtani a felhasználók számára.
Az ***UHU-Linux 1.1*** és ***1.2*** között a fejlesztési erőforrásaink nagy részét a rendszer mélyebben fekvő komponenseinek átszervezésére fordítottuk, és a korábbi kiadásokhoz képest relatíve kevesebb (de még mindig nem kevés) újdonságot hoztak a kezdőbb felhasználó által is látott grafikus felületek és alkalmazások. Írásommal ezért nem a ***Linuxszal*** most ismerkedő, hanem inkább a haladóbb, technikai részletek iránt fogékonyabb olvasókat célzom meg, felvázolva az ***UHU-Linux 1.2*** által nyújtott technológiai újdonságokat.

Hardverigény, telepítés

Csomagjainkat az ***Intel Pentium*** (586-os) utasításkészletre fordítottuk, így a rendszer futtatásához ezeket ismerő processzorra van szükség. Természeten minél gyorsabb, annál jobb, a grafikus környezetekhez legalább 500 MHz ajánlott. Az ***UHU-Linux*** alapértelmezésben támogatja és kihasználja a többprocesszoros ***(SMP)*** rendszereket és a ***HyperThreading (HT)*** technológiát.
A rendszer telepítéséhez legalább 96 MB, futtatásához legalább 64 MB RAM szükséges, ugyanakkor a népszerűbb grafikus környezetekhez 256 MB igencsak ajánlott. Amennyiben csak 64 MB memória van a gépben, és a telepítést kölcsönkért memóriamodulokkal sem tudjuk megoldani, még mindig van egy kerülő lehetőség. Indítsuk a telepítőt hibakereső módban, a kapott parancssorban először particionáljuk a merevlemezt a `cfdisk` paranccsal, majd inicializáljuk és aktiváljuk a cserepartíciót az `mkswap` és `swapon` parancsok segítségével. Ezt követően már 64 MB memóriával is eldöcög a grafikus telepítő. (A telepítéskor a nagyobb igényt az indokolja, hogy a telepítő kódja teljes egészében a memóriában ül.)
Az első CD-ről történő teljes telepítéshez legalább 3 GB lemezterület javasolt.
A telepítés menete lényegében megegyezik az 1.1-es ***UHU***-éval. Egy nagyobb kaliberű változtatás történt, ez pedig a CD-n használt ***rendszerbetöltő (boot loader)*** cseréje. Az 1.1 idején még a ***syslinux CD***-re szánt változatát, az ***isolinux***-ot használtuk. Idő közben azonban a ***GRUB*** rendszerbetöltőben (melyet a telepített ***UHU-Linux*** indítására már korábban is használtunk) megjelent a CD-ről bootolás támogatása. Mivel a ***GRUB*** egy sokkal rugalmasabb rendszerbetöltő, a CD lemezen is átálltunk ennek a használatára. A ***GRUB*** egyik óriási előnye, hogy nemcsak az előre elkészített bejegyzések indíthatók, hanem (szöveges felületéből, melyre az ***Esc*** gombbal léphetünk ki) tetszőleges fájlt betölthetünk indítandó ***kernel*** és ***initrd*** gyanánt, illetve tetszőleges partíció boot szektorára is ugorhatunk. Ily módon egy kis gépeléssel akár a korábban telepített ***Linux*** rendszert is könnyedén el tudjuk indítani, ha a merevlemez elején lévő rendszerbetöltő megsérül. A ***GRUB*** részletes használatáról, beleértve az imént említett lehetőséget is, annak grafikus felületén a hypertext rendszerű súgóban olvashatunk.
A régi ***isolinux*** rendszerbetöltő a második CD elejére került, vésztartaléknak arra az esetre, ha valahol a ***GRUB*** indítása problémába ütközne. Ebben az esetben a második CD-ről indítsuk a rendszert, és amint bejelentkezik a bootképernyő, cseréljük ki a CD-t az elsőre, ezt követően válasszuk a telepítés opciót.

A kernel

Disztribúciónk előző verziója még a ***Linux*** kernel 2.4-es verzióján alapult, a mostani kiadás viszont már a 2.6-os sorozatra épült. A 2.6-os számtalan újítást hozott, egy teljes disztribúció felépítése tekintetében többet, mint a korábbi fő kernel verziók. Így ennek a lépésnek köszönhetően lehetővé vált több technikai váltás is, melyek a rendszert egyszerűbbé, áttekinthetőbbé és sokkal modernebbé tették.

Többszálú futtatás

A ***POSIX*** szabványok régóta rögzítik azt a programozói interfészt, mely segítségével több szálon futó alkalmazást lehet írni. Ugyanakkor a Linux kernel 2.4-es verziója még nem támogatta a több szálon futó folyamatokat. Éppen ezért a POSIX szálkezelést kivitelező függvénytár, az úgynevezett ***LinuxThreads*** minden egyes szálat külön kernelfolyamatra képzett le. Ennek eredményeképp a szálak közti váltás tulajdonképpen teljes folyamatváltást igényelt a kernel részéről. A 2.6-os kernelben jelent meg a több szálból álló folyamatok tisztességes támogatása, mely például a szálak közötti jóval kisebb erőforrás-igényű váltás miatt komoly teljesítménynövekedéshez vezethet többszálú programok esetén.
A kernelben lévő támogatással ugyanakkor semmit sem érnénk, ha azt a felhasználói programok nem használnák ki, vagyis ha a régi ***LinuxThreads*** függvénykönyvtárat meghagynánk. A kernelszintű szélkezelés kihasználása a függvénytár részéről olyan gyökeres átalakítást igényelt, hogy a fejlesztők a ***LinuxThreads*** jobbá tétele helyett újragondolt, újratervezett, elölről megírt alternatív szoftver mellett döntöttek, ez pedig az ***NPTL (Native POSIX Thread Library)*** nevet kapta.
A disztribúcióban tehát lecseréltük a régi ***LinuxThreads***-et az új ***NPTL***-re. A `ps ax` parancs kimenetében a többszálú programok (***Java***-alkalmazások (például ***jdictionary***), ***nscd***) ***LinuxThreads*** használatával több példányban jelentek meg, az NPTL rendszerrel azonban már egy folyamatként látszanak. A ***/proc*** fájlrendszerben a folyamat azonosítója alatt a ***task*** könyvtárban látszik, hogy az adott folyamat valójában több szálból áll.
Az új rendszer nemcsak jobb teljesítményt nyújt a többszálú programok futtatása előtt, hanem a szabványban leírtaknak (például szignál küldése) is pontosabban felel meg. Ugyanakkor, mint szinte minden ilyen átállás, apró hátulütői is vannak. Az új ***UHU*** binárisainak futtatásához 2.6-os kernel szükséges, 2.4-es kernel nem képes futtatni a programokat, így például nem lehetséges futás közben a régi ***UHU-Linux*** rendszert 1.2-re frissíteni (a frissítés CD-ről bootolva végezhető el), illetve futó ***UHU-Linux 1.1*** alatt nem lehetséges ***UHU-Linux 1.2***-höz csomagot gyártani sem. Továbbá az új rendszer a ***glibc 2.0***-s verziójával linkelt programokat már nem tudja futtatni, csak azokat, melyeket a ***glibc***-nek legalább az (egyébként immár 6 évvel ezelőtt megjelent) 2.1-es változatához fordítottak.

Sys fájlrendszer

Megjelent egy új virtuális fájlrendszer, a ***sysfs***, melynek tartalma a ***/sys*** csatolási pont alatt érhető el. Itt a kernel a rendszer hardver felépítésével kapcsolatos információkat teszi egységes formátumban elérhetővé a kíváncsi alkalmazások (leginkább rendszerprogramok) számára.

Udev eszközkezelés

A ***Unix*** rendszerek a legtöbb hardver eszközhöz és egy-két speciális szolgáltatáshoz a ***/dev*** könyvtár alatti virtuális fájlokon keresztül biztosítanak hozzáférést. A ***/dev*** könyvtár tartalmának összeállítása nem egyszerű feladat.
A legősibb megközelítés a statikus ***/dev*** könyvtár. Az összes lehetséges hardver eszközhöz tartozik egy bejegyzés, így nemritkán akár kétezer speciális fájl is lehet a ***/dev*** alatt.
A ***/dev*** könyvtár tartalma ezáltal nehezen menedzselhető, átláthatatlan, nehézkes a fájlok tulajdonosának, csoportjának és hozzáférési jogainak beállítása és karbantartása.
A bejegyzések nagy része pedig értelmetlen, fölösleges az adott gépen.
Amennyiben olyan eszközt próbálunk meg megnyitni, amelyhez nem tartozik meghajtó a kernelben, az esetben egy külső segédprogram segítségével a kernel megpróbálja betölteni a szükséges meghajtó modult, és ha sikerrel járt, akkor így az eszközfájl megnyitása is sikeres lesz. Ily módon megoldható az automatikus meghajtó-betöltés. Néhány évvel ezelőtt még tipikus gyakorlat volt, hogy ilyen módon például a hangkártya meghajtója akkor töltődött be a háttérben, amikor először megpróbáltunk zenét lejátszani az operációs rendszer indítása után.
A fenti rendszer nehézkes karbantarthatósága miatt pár éve a ***devfs*** volt a divat, ezt használták az ***UHU-Linux*** korábbi kiadásai is. Itt a kernel magára vállalta a ***/dev*** alatti eszközfájlok menedzselését. A ***/dev*** csatolási pont alá egy virtuális, ***devfs*** típusú fájlrendszert csatoltunk, mely alatt csak azok az eszközök voltak láthatók, melyek meghajtója be volt töltve. Ugyanakkor lehetőség volt nem létező fájl megnyitására is, a kernel ilyenkor (név alapján beazonosítva az eszközt) megpróbálta betölteni a modult. Utóbbi szolgáltatásra nem volt túl nagy igény, mivel szinte természetessé vált az elmúlt pár évben, hogy a hardverkezelő modulokat nem igény esetén, hanem már a rendszer indulásakor betöltik a disztribúciók.
A ***devfs*** megközelítésnek is voltak problémái, túl sok minden volt a kernelbe beégetve, olyan dolgok is, melyek felhasználói térben megoldhatók (és az ilyeneket nem szeretik a kernelben látni a fejlesztők), és ennek megfelelően a rendszer nem volt kellőképpen rugalmas. Végezetül, az ***udev*** alternatíva megjelenése kapcsán a fejlesztők a ***devfs*** rendszert elavulttá nyilvánították.
Az ***UHU-Linux 1.2***-ben mi is átálltunk a legújabb megközelítésre, az ***udev***-re. Itt a ***/dev*** könyvtár tartalmát egy felhasználói program (az ***udev démon***, ***udevd***) menedzseli, a kerneltől kapott úgynevezett ***hotplug üzenetek*** segítségével (a kernel, valahányszor hardver eszközzel kapcsolatos esemény történik, elindítja az ***/sbin/hotplug*** szkriptet, amely az ***udev*** rendszert is értesíti), valamint az újonnan megjelent ***/sys*** könyvtár tartalmára is erősen támaszkodik az ***udev***.
A ***/dev*** könyvtár lehetne egy közönséges könyvtár is (merevlemezen tárolt fájlrendszer részeként), de a javasolt megoldást követve ez egy virtuális, memóriában elhelyezkedő ***ramfs*** típusú fájlrendszer az ***UHU***-ban.
Az eredeti kernel maga tehát, azon túl, hogy a ***/sys*** fájlrendszer tartalmával és az ***/sbin/hotplug*** szkript indításával tájékoztatja az érdeklődő rendszerprogramokat a hardverek helyzetéről, semmit nem tesz a ***/dev*** alatti bejegyzések létrehozásával, jogosultságainak beállításával kapcsolatban, ezt

teljes egészében külső programra bízza. Az ***UHU*** kernelében ezen icipicit módosítottunk, a rendszer indítási folyamatát lényegesen leegyszerűsítendő a kernel végzi a */dev* könyvtár alá a ***ramfs*** típusú üres ***RAM***-ban található fájlrendszer csatolását, és létrehozza alatta a néhány legfontosabb bejegyzést, melyekre már az első folyamatként induló ***init*** program is számít.

A */dev* könyvtár alatt tehát memóriabeli fájlrendszer található, így ha itt bármit kézzel változtatunk, az a rendszer újraindítása során elvész. Viszont szükségünk lehet arra, hogy a saját ízlésünk szerint állítsuk be az egyes eszközfájlok tulajdonosát, csoportját, vagy hozzáférési jogosultságát. Erre az ***udev*** konfigurációs fájljait, elsősorban az */etc/udev/permissions.d* könyvtárat használhatjuk. Átírhatjuk az itt lévő ***50-udev.permissions*** fájlt is, de ajánlott ezt a fájlt inkább változatlanul hagyni, és más néven létrehozni egy új fájlt és abba írni saját igényeinket. Fontos a ***.permissions*** kiterjesztés megtartása, és értelemszerűen a fájlnév elején lévő sorszám szerinti sorrendben bírálják egymást felül a fájlok, így saját bejegyzéseink számára érdemes lehet például a ***90-sajatbeallitasaim.permissions*** fájlnevet választani.

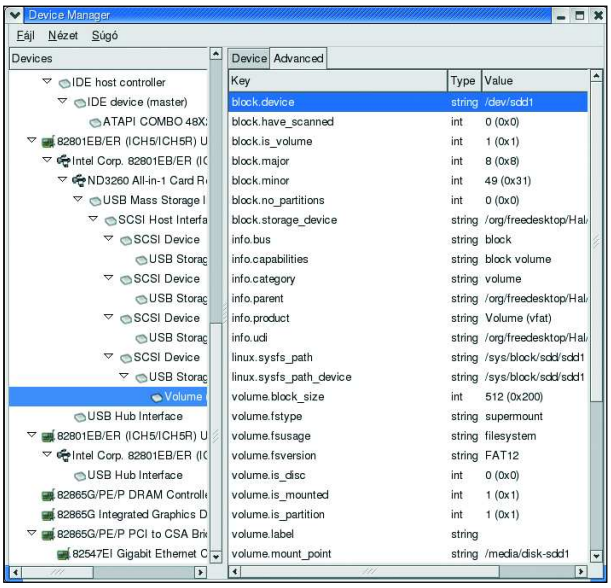
Az ***udev*** rendszernek előnye tehát, hogy a kernel részéről nem igényel támogatást, mindössze azt a hardverdetektálási infrastruktúrát, amely egyébként is rendelkezésre áll a kernelben, nemcsak az ***udev*** kedvéért. A többi felhasználói térben van kivitelezve, dinamikusabban fejleszthető, könnyebben és rugalmasabban testre szabható (akár az eszközfájlok neve is megváltoztatható).

Ugyanakkor van két apró hátránya is az ***udev*** rendszernek. Az egyik, hogy mivel nem speciális típusú a fájlrendszer, a ***devfs*** megoldással ellentétben itt nincs lehetőség a nem létező fájlra irányuló megnyitási kísérletet elkapni és gyorsan intézkedni a megfelelő modul betöltéséről. (Bár, mint láttuk, ezt már a ***devfs*** esetén sem igazán használtuk.) A másik apró probléma speciális alkalmazások készítőit bosszanthatja, amennyiben nem hardvert kezelő modulról van szó (hanem például hálózati protokoll támogatásáról). A két régi rendszerben elég volt megnyitni egy eszközfájlt: ha az azt kezelő modul még nem volt betöltve, akkor betöltődött a háttérben, majd ezt követően sikeresen folytatódott a program futása. Az új rendszerben a programnak explicit módon kérnie kell a modul betöltését, majd ezt követően bizonytalan ideig várnia, amíg a vele aszinkron módon futó ***udev*** rendszer létrehozza az igényelt fájlt. Ezekkel a problémákkal azonban az legtöbb felhasználó aligha fog találkozni.

A hagyományos statikus ***dev***-hez képest tehát az evolúció során két lépésben szinte teljesen „kifordult" a rendszer. Régen az eszközfájl megnyitása idézte elő a modul betöltését. Most viszont az elérhető hardverekhez az azonosítójuk (többnyire ***PCI*** azonosító) alapján előre betöltjük a modulokat, és a modul betöltése hozza létre (az ***udev*** rendszeren keresztül) a */dev* alatt a megfelelő eszközfájlt.

Hal

Az új */sys* fájlrendszert használja ki a ***HAL (Hardware Abstraction Layer, hardver absztrakciós réteg)*** is, amely szintén az */sbin/hotplug* szkript segítségével értesül az eseményekről, melyeket a központi ***hald*** nevű démon folyamat



dolgoz fel és továbbít (a ***D-BUS*** üzenetküldő rendszer segítségével) az arra kíváncsi felhasználói alkalmazások felé.

A ***HAL*** célja a rendszerben használt hardver elemekről, azok javasolt és tényleges felhasználási módjáról egy minél részletesebb valósidejű adatbázist karbantartani. Tehát éppúgy tárolja a hardverek fizikai jellemzőit, mint például olyan információkat, hogy adott partíción milyen fájlrendszer található, milyen módon ajánlott azt csatolni (az ajánlásokat további szkriptek dolgozhatják fel), hova csatoltuk valójában a fájlrendszert, és így tovább.

Egy lehetséges ilyen alkalmazás a ***HAL eszközkezelő (hal-device-manager)*** program, amelynek kifejezetten az a célja, hogy a ***HAL*** által nyilvántartott adatokat grafikusan megtekinthessük. Elsősorban tehát hibakereséshez vagy fejlesztéshez használható a program, de mindenképp érdemes vetni rá egy pillantást felhasználóként is. Az igazán izgalmas adatok a jobb oldali ***Advanced*** fülre kattintva jelennek meg. A háttérben munkálkodó üzenetküldő rendszer létezését mi sem demonstrálja jobban, mint hogy változás (például ***USB*** tároló csatlakoztatása) alkalmával a ***hal-device-manager*** által mutatott lista is azonnal megváltozik.

A ***GNOME*** grafikus környezet is támaszkodik a ***HAL*** démontól érkező eseményekre. Ilyen módon válik lehetővé, hogy új eszköz csatlakoztatásakor automatikusan megjelenjen számára egy ikon az asztalon, megnyíljon egy ***Nautilus*** ablak a fájlrendszer tartalmával, vagy éppenséggel automatikusan elinduljon a film lejátszása. Természetesen az automatikus programindítás beállítható, akár le is tiltható a ***Gnome Vezérlőpultban***, a ***Cserélhető adathordozók*** pont alatt.

Az ***UHU-Linux*** következő verziójában nemcsak a ***GNOME***, hanem a ***KDE*** grafikus felület kedvelői is élvezhetik majd a ***HAL*** által nyújtott lehetőségeket.

Végül, de nem utolsó sorban az ***uhu-automount*** rendszert is, mely az eszközök tartalmát automatikusan csatolja a */media* (korábban */mnt*) könyvtár alá, átültettük a ***HAL*** használatára, így az automatikus csatolást végző rendszer az alatta lévő infrastruktúrának köszönhetően egyetlen rövidke héjprogrammá (*/etc/hal/device.d/automount.hal*) redukálódott.

/media

Szóba került már, hogy az automatikus eszközöket a korábbi */mnt* helyett immár a */media* könyvtár alá csatoljuk. Ezt a váltást a ***Filesystem Hierarchy Standard*** ajánlását és több vezető disztribúciót követve léptük meg. Az */mnt* könyvtárat az ***UHU-Linux 1.2*** semmire nem használja, ideiglenes csatolások számára ideális csatolási pont.
A */media* könyvtár tartalma szintén egy memóriában létező (*ramfs* típusú) fájlrendszer. Ez azt jelenti, hogy ha itt bármit módosítunk kézzel (például új könyvtárat hozunk létre csatolási pont gyanánt), az a rendszer újraindítása után már nem lesz meg. Éppen ezért célszerű a */media* könyvtárat meghagyni teljes egészében az ***UHU automount*** rendszere számára, és ha másvalamit is szeretnénk csatolni (például hálózati kötetet), akkor számára például az */mnt* könyvtárat, vagy annak egy általunk létrehozott alkönyvtárát választani.
A */media* könyvtárral kapcsolatban megjegyzendő még, hogy tartalmához csak a *media* csoport tagjai férhetnek hozzá. A *media* csoport a korábbi *cdrom*, *cdwriter*, *floppy* csoportokat váltotta fel, mivel manapság annyiféle csatlakoztatható eszköz létezik, hogy értelmetlen és lehetetlen volna mindegyik számára külön csoportot létrehozni, így inkább összevontuk mindet eggyé.
A */etc/group* fájlban, vagy az ***UHU Vezérlőpultban*** hiába nézzük, azt fogjuk látni, hogy senki sem tagja a csoportnak. Ez azért van így, mert ebbe a csoportba nem fix felhasználók tartoznak bele, hanem dinamikusan (a *PAM* rendszer segítségével) azok kerülnek bele, akik helyileg jelentkeznek be a rendszerbe (használjuk az `id` parancsot a tagság ellenőrzésére). Így még ha hozzáférési jogot is adunk valakinek távoli belépésre a gépünkre, biztosak lehetünk benne, hogy a floppylemezen, CD-n, pendrive-on tárolt adatainkhoz nem fér hozzá, hacsaknem azt külön elérhetővé tettük számára. Ha ezt szeretnénk, betehetjük állandóra a *media* csoportba az illetőt, vagy a */media* könyvtár hozzáférési módját is átállíthatjuk megfelelőre (ám ez utóbbi változtatás a rendszer újraindítása során elvész, tehát gondoskodnunk kell róla, hogy bootolás során is végrehajtódjon a megfelelő `chmod` parancs).

CD-írás

A CD-írás tekintetében is hozott újdonságokat a 2.6-os kernel, és a disztribúció egyéb változtatásai. Ezek az újdonságok grafikus CD-író programok (például ***K3b***) használóinak aligha tűnnek fel, viszont a parancssor kedvelőinek annál inkább.
Lehetőség vált az *ATA* eszközökre történő közvetlen CD-írásra, így a korábbi *SCSI* emulációra immár nincsen szükség. A `cdrecord` parancsnak eddig szüksége volt egy olyasmi paraméterre, mint például `dev=1,0,0`. A lehetséges értékeket (SCSI eszközazonosítót) a root-ként futtatott `cdrecord -scanbus` parancs írja ki.
Az új *UHU*-ban *ATA* CD-író esetén a `dev=` opciónak értékül a `cdrecord -scanbus` kimenetéből kilesett értéket egy `ATA:` előtaggal (prefix) kell ellátni, például `dev=ATA:1,0,0`. Szerencsére van egyszerűbb lehetőség is, adhatjuk közvetlenül a */dev* fájlrendszer alatti eszköznevet (például `dev=/dev/hdc`), és mivel a */dev/cdwriter* szimbolikus linket beállítjuk a CD-íróra (több író esetén az egyikre) és a `cdrecord`-nak is megtanítottuk, hogy ez az eszköz az alapértelmezett, így az esetek túlnyomó részében egyáltalán nincsen szükség a `dev=` opció megadására.

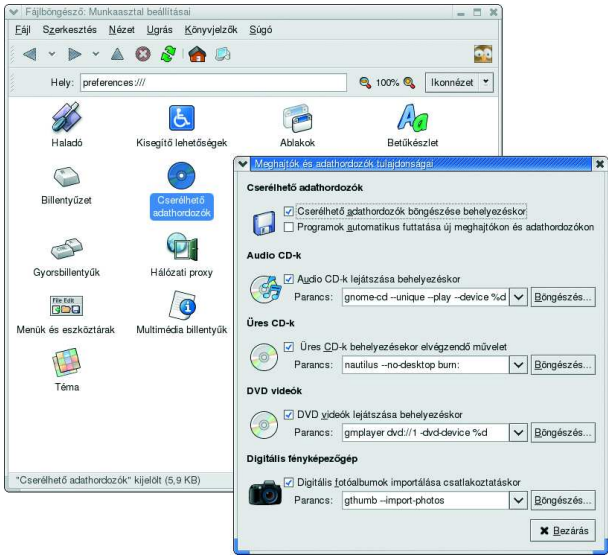
Kernelmodulok

Még egy rövidke fejezet erejéig maradjunk a 2.6-os kernelnél. A modulok betöltése terén is sok technikai részletet átszerveztek a kernelfejlesztők. A modulok betöltését a korábbi *modutils* helyett az újabb, *module-init-tools* nevű csomag programjai (`modprobe`, `insmod` stb.) végzik, melyek működése lényegében megegyezik elődjeik működésével. Különbség, hogy a `modprobe` konfigurációs fájlja, melyben a modulok alapértelmezett paramétereit adhatjuk meg, a korábbi */etc/modules.conf* helyett immár */etc/modprobe.conf* névre hallgat.
A modulok automatikus betöltését is kissé átdolgoztuk. A korábbi */etc/modules/AUTOLOAD* fájlt átkereszteltük */etc/modules.load*-ra, az ebben felsorolt modulokat a rendszer mindenképp betölti az indulás folyamán. Megjelent továbbá az */etc/modules.skip* fájl is, melyben soronként egy modul nevét adhatjuk meg, ezeket a modulokat semmiképp nem fogja betölteni a rendszer, akkor sem, ha a hardverdetektálás alapján szükségét érezné.
A 2.6-os kernel moduljainak nevében az aláhúzás és a kötőjel azonos szerepű karakterek, így nem szabad meglepődnünk, ha például az *snd-pcm* modul betöltése után *snd_pcm* jelenik meg a betöltött modulok listájában.

PATH környezeti változó

A *Linux* disztribúciók egyik kritikus gyenge pontja a környezeti változók beállítása a felhasználó számára. Különféle programok helyes működése számára fontos lehet egy-egy környezeti változó helyes beállítása, nem ritka, hogy egy alkalmazás több tucat ilyen változót is felismerjen és azok alapján másképp viselkedjen. Az egyik legfontosabb mind közül a *PATH* nevű, mely a programok keresési útvonalát tartalmazza, ennek segítségével válik lehetővé, hogy egy alkalmazás a teljes útvonal ismerete nélkül is indítható legyen, például elegendő legyen a `mozilla` programot indítanunk a `/usr/bin/mozilla` helyett, a rendszer megkeresse és megtalálja azt.
A környezeti változók folyamatonként külön léteznek, alap értelmezésben öröklődnek (a gyermekfolyamat megkapja az őt indító szülő környezeti változóit), de a szülő másmilyen változókkal is indíthatja a gyermek folyamatot.
A disztribúciók nagy része a környezeti változók beállítását a bejelentkező felhasználó nevében elsőként futó programra bízza (a `PATH` beállítása az ***UHU-Linux 1.1***-ben is még így történt). Ez a program szöveges bejelentkezés esetén az úgynevezett parancsértelmező (shell, héj), grafikus belépés esetén általában egy héjprogram (a parancsértelmező nyelvén megírt utasítássorozat), de lehet közvetlenül az ablakkezelő is. Idáig még nem reménytelen megoldani a problémát, viszont nehezít, hogy a felhasználó többféle parancsértelmező közül is választhat, mindegyiknek másképpen kell megadni a beállítandó környezeti változókat, így mindenképpen

a rendszerben több helyen is be kell állítani megfelelően ezeket a változókat, ami semmiképp sem szerencsés.
Az igazi probléma azonban ott kezdődik, hogy be lehet lépni úgy is a rendszerre, hogy ezen programok egyike se hajtódjon végre, vagy ha maga a héj végre is hajtódik, ne olvasson ki semmiféle inicializáló fájlt. Erre a legtipikusabb példa az, amikor `ssh`-val távolról úgy lépünk be, hogy rögtön egy végrehajtandó parancsot is megadunk, tehát nem parancsértelmezőt kérünk. Ilyenkor az általunk megvizsgált disztribúciókban mind-mind hibás volt a PATH értéke.
A PATH-t azért hangsúlyozom, mivel azt általában nehezebb a többi változónál beállítani. Míg a legtöbb környezeti változó számára a rendszer összeállítója (disztribútor, rendszergazda) fix értéket óhajt beállítani (vagyis az érték független a rendszer tulajdonságaitól, a felhasználótól), addig a PATH esetén ez nem így van, egy jó PATH elemei függenek a feltelepített szoftverektől (újabb csomag behozhat újabb keresési könyvtárat), valamint függenek a felhasználótól is, hiszen igencsak illik a felhasználó egyik könyvtárát (általában a ***~/bin*** könyvtárat) is felvenni, sőt, a root-nak az adminisztrátori teendők ellátására szolgáló parancsok könyvtárait (***/sbin***, ***/usr/sbin***, ***/usr/local/sbin***) is meg kell kapnia.
Ahhoz tehát, hogy a PATH-t beállítsuk, le kell futtatni valami e célra elkészített kódot, amelyik összerakja, hogy minek is kell az értéknek lennie.
A konstans értékű környezeti változók esetén több disztribúció is választotta már azt a megoldást, hogy a parancsértelmezők inicializáló fájljai és a grafikus bejelentkezéskor lefutó szkript helyett egy lépéssel korábbra, a ***PAM*** nevezetű hitelesítési rendszerbe helyezi azok beállítását. Erre a hivatalos ***PAM*** szoftvercsomagban is található modul, mely ***pam_env.so*** névre hallgat. Ekkor tehát maga a belépést engedélyező program állítja be a környezeti változókat (amennyiben a felhasználó azonosítása sikeres volt), és a felhasználó legelső folyamatát, vagyis a parancsértelmezőt vagy a grafikus környezetet már eleve a kívánt változókkal indítja. Ebben a megközelítésben megszűnik tehát a kódtöbbszörözés, és megszűnik a nem interaktív távolról parancsfuttatás problematikája is, hiszen ilyenkor is helyesen lesznek a környezeti változók beállítva. A PATH kivételével már az ***UHU-Linux 1.1*** is ezt a megoldást követte.
Több probléma adódott azonban az ***UHU***-ban abból, hogy a PATH változónk nem volt mindig helyesen beállítva.
A beállítást a parancsértelmező végezte, így grafikus környezetbe belépve a terminálokban indított parancsértelmezőket leszámítva a programok PATH értéke hibás volt (ezt még csak-csak meg lehetett volna oldani), illetve nem interaktív parancsfuttatás esetén sem a végleges PATH volt beállítva. Utóbbi problémát már csak a PATH beállításának koncepcionális átdolgozásával volt esély orvosolni, nyilvánvaló volt, hogy a ezt is a ***PAM*** hitelesítési rendszerbe kell átemelnünk, a többi változó mellé, és megoldani a PATH értékének kitalálását, lehetőség szerint plusz csomagok által könnyen bővíthető módon.
A megoldás megtervezése során végül is két részre szedtük a feladatot. Egyik komponensként elkészült a ***pam_envfeed.so*** nevezetű modul. Ez a modul annyit



csinál, hogy elindít egy külső programot (alap értelmezésben az ***/sbin/pam_envfeed***-et), figyeli ennek a kimenetét, és a kimenetben talált minden változó=érték párt beállít környezeti változóként. Ez tehát egy teljesen általános, bármely környezeti változó beállítására roppant dinamikusan használható modul.
Másik lépésként pedig megírtuk azt a ***pam_envfeed*** szkriptet, mely az ***/etc/envfeed.d*** könyvtár alatti, ***.sh*** kiterjesztésű szkripteket hajtja végig sorra, és minden ezek által ENVFEED_ kezdetű névvel beállított környezeti változóra kiírja azok nevét az ENVFEED_ előtag nélkül, valamint az értékét. Ily módon több szkript összjátékának eredményeképpen kitaláljuk a leendő PATH értékét az ENVFEED_PATH változóban, miközben nyugodtan használhatunk a szkriptekben sok egyéb változót (köztük a PATH-t is teljesen másmilyen értékkel), és végső soron ezektől függetlenül mondjuk meg, hogy mi legyen a leendő PATH.
Az átállásnak van egy érdekes mellékhatása is. Amennyiben a `su` paranccsal váltunk át rendszergazdává, az esetben a ***pam_env.so*** és ***pam_envfeed.so*** modulok nem hajtódnak végre (lásd az ***/etc/pam.d/su*** fájlt). A régebbi ***UHU***-kban a környezeti változók ilyenkor egy kivétellel változatlanok maradtak, az egy kivétel természetesen a PATH volt, melyet a rendszergazdaként indított héj beállított magának. Persze itt sem volt tökéletes a helyzet: ha a su-nak megadtunk indítandó parancsot, akkor itt sem állítódott be a PATH. Az új ***UHU***-ban azonban már egyáltalán nem állítódik be a PATH (hiszen a shell inicializáló fájljából kikerült az a kód, amit a ***PAM*** rendszerbe ültettünk át), vagyis olyan parancsértelmezőt kapunk, amely ténylegesen szigorúan véve kizárólag rendszergazdai jogosultságot nyújt nekünk, de nem rendszergazdai környezetet. Hiába gépeljük be az `ifconfig`, a `chroot` vagy bármely hasonló parancsot, azokat nem találja meg a parancsértelmezőnk, mivel nincsen benne a PATH-ban.
A fenti probléma kiküszöbölésére használható a `su -` parancs, mely rendeltetésének megfelelően már nemcsak rendszergazdai jogosultságot, hanem rendszergazdai környezetet is biztosít, beleértve többek között a PATH ennek szellemében történő beállítását is, hiszen végrehajtja

a *pam_env.so* és *pam_envfeed.so* modulokat is (lásd az */etc/pam.d/su-* és az általa hivatkozott *system-auth-rootok* és *system-auth* fájlokat).

Inotify, Gamin

A hagyományos *Unix* rendszerhívások között nem szerepel olyan, amellyel egy fájl megváltozását figyelhetnénk. A régi nagy *Unix* rendszerekben erre a szolgáltatásra úgy látszik hogy nem igazán volt szükség. A *Linux* asztali rendszereken történő elterjedése során jelent meg az igény erre elsősorban a grafikus fájlkezelők részéről, hiszen ezek szeretnék az ablakuk tartalmát megfelelően módosítani, mihelyst valamely általuk megjelenített fájl vagy könyvtár bármilyen tulajdonsága megváltozik (például új fájlt hozunk létre). Nyilvánvaló, hogy a *fájlstruktúra folyamatos lekérdezése (polling)* nem igazán működőképes megoldás, hiszen hatalmas az erőforrásigénye. Egy olyan rendszerre van szükség, amelyben a kerneltől kapunk értesítést, amikor minket érintő változás történik. Jelenleg mind a *Gnome*, mind a *KDE* grafikus felület támaszkodik erre a szolgáltatásra.
Az *UHU-Linux 1.2* mind a kernelben található támogatás, mind az erre épülő függvénytár terén újítással szolgál.
A kernelben a korábbi *dnotify* rendszer mellett megjelent az *inotify*. (Ez egyelőre nem része a hivatalos kernelnek, az *UHU* kernelbe külön foltként került bele.)
A *dnotify* rendszerben (a „d" betű a directory-ra, vagyis könyvtárra utal) könyvtárat van lehetőségünk figyelni, fájlt ezáltal csak közvetve (az őt tartalmazó könyvtárra értesülünk eseményről, majd innen kezdve az összes fájlt végig kell nézni annak eldöntéséhez, hogy melyik változott). Minden egyes figyelt könyvtár egy újabb nyitva tartott fájlleírót igényel, így nem kizárt, hogy egy folyamat beleütközzön az egyszerre megnyitható fájlok korlátjába. A megnyitás megfogja az adott könyvtárat, így azt például, ha az egy csatolási pont, nem lehetséges lecsatolni. Ezen felül a *dnotify* a felhasználói folyamattal szignálokon keresztül kommunikál, ami erőforrásigényes és nehézkesen kezelhető.
Az új, *inotify* nevű rendszer mindezeket a hibákat hivatott kiküszöbölni. Az „i" betű az i-node-ra utal, hiszen tetszőleges *i-node (fájlrendszerbejegyzés)* figyelhető, fájl és könyvtár egyaránt. A kommunikáció egy speciális eszközön (*/dev/inotify*) keresztül történik, a processz itt közli a megfigyelendő objektumok listáját, és itt értesül a változásokról is. Ezáltal az eseményre várakozás lényegesen könnyebben és letisztultabb módon programozható le, valamint lehetőség nyílik például arra is, hogy értesüljünk, ha a megfigyelt objektumot tartalmazó fájlrendszer lecsatolódik.
Az alkalmazások általában nem közvetlenül a *dnotify* vagy *inotify* interfészt használják, hanem egy köztes réteget, egy függvénytárat. Korábban a *FAM* nevű csomag szolgált erre, mely a háttérben a *dnotify*-ra támaszkodott. A *FAM*-mal is több probléma volt, többek között nem képes kihasználni az *inotify* nyújtotta előnyöket, valamint globális démonfolyamat futását igényli, ami megközelítés mind stabilitás, mind pedig biztonság terén igencsak megkérdőjelezhető döntés.

A *FAM* rendszert szintén lecseréltük modernebb utódjára, a *Gamin*-ra, amely az alkalmazás felől nézve visszafelé kompatibilis a *FAM*-mal, így a csere nem igényelt változtatást a *FAM*-ot használó alkalmazások (*Gnome*, *KDE* stb.) részéről. A *Gamin* alapértelmezésben (amennyiben rendelkezésre áll) az *inotify* rendszert használja, de futási időben képes visszaállni *dnotify*-ra, ha az *inotify* valami miatt nem elérhető. Továbbá ha egy megfigyelt fájl túl gyakran változik, akkor arra a fájlra automatikusan átáll lekérdezéses módba. A *Gamin* teljes egészében a felhasználó jogosultságával fut, nem tartalmaz központi, rendszergazdaként futó démont. Ugyanakkor technikai okok miatt indít egy külön folyamatot *gam_server* néven, amely az adott felhasználó összes folyamata számára szolgáltatja a megfelelő információkat. Így a rendszerben mindig felhasználónként mindössze egy *gam_server* folyamatot látunk, amelyet a legelső, ilyen szolgáltatást igénybe vevő program a háttérben elindít, a többi program már ehhez csatlakozik, és a *gam_server* akkor lép ki, amikor a felhasználónak immár egyetlen folyamata sem figyel meg egy fájlt sem. A külön folyamatok ily módon történő összefogása szintén erőforrás megtakarításához vezet.

A jövő

Nehéz kérdés most előre megjósolni, hogy milyen lesz az *UHU-Linux* következő verziója. Természetesen bőven vannak terveink, több kiadásra is elegendően.
Szinte biztos, hogy a következő kiadás 2.0 sorszámot fogja viselni, és főként inkább felhasználói szemmel nézve, a grafikus felületen fog újdonságokat hozni. Könnyen elképzelhető egy vadonatúj grafikai stílus megjelenése is.
Ami már egészen biztos, az az, hogy át fogunk állni *UTF-8* karakterkódolásra (ez a Unicode egyik ábrázolási formája), lényegében teljesen magunk mögött hagyva az idejétmúlt *Latin-2 (ISO-8859-2)* kódolást. Az átállást számtalan olyan ékezetkezelési probléma indokolja, melyek a régi rendszerben nem oldhatók meg. A mostani *UHU* a legtöbb helyen még a régi *Latin-2* karakterkészletet használja, de pár helyen már a modernebb *UTF-8* kódolást. Különösen a fájlnevek terén nagy a kavarodás, mivel bizonyos alkalmazások az egyik, míg mások a másik ábrázolást részesítik előnyben, így sok esetben az egyik programmal létrehozott ékezetes fájlneveket a másik program nem látja helyesen, és viszont. Az ilyen és ehhez hasonló problémákra fog végleges és megnyugtató megoldással szolgálni az átállás, onnan kezdve biztonsággal lehet majd gyakorlatilag bárhol használni az ékezetes betűket.
Egyéb elképzeléseinkről egyelőre nem írnék, ez maradjon meglepetés. A fejlesztés menete iránt érdeklődőket szívesen látjuk a *dev@uhulinux.hu* levelezési listán (feliratkozni a *lists.uhulinux.hu* címen lehet), ahol további fejlesztési kérdések megvitatásával is találkozhatnak, valamint szívesen fogadjuk az ötleteket. Ugyanakkor kérjük, hogy ezt a listát ne használjátok olyan felhasználói kérdések feltevésére, melyek nem az *UHU-Linux* fejlesztésével kapcsolatosak, az ilyen kérdések megvitatására üzemeltetjük a *kezdo@uhulinux.hu* és *halado@uhulinux.hu* listát, ahol szintén mindenkit szívesen látunk.

Koblinger Egmont
Informatikus, az UHU-Linux fejlesztője

WLAN-ok védelme WPA és FreeRADIUS alkalmazásával (2. rész)

A vezeték nélküli hálózatok védelmi eszközeinek új generációja nem csupán a WEP hiányosságait tünteti el, de lehetővé teszi a vezeték nélküli felhasználók RADIUS kiszolgálóval végzett hitelesítését is.

A múlt hónapban ismertettem a vezeték nélküli *LAN*-ok új biztonsági protokollját, a *WPA*-t (***Wi-Fi Protected Access***, ***Wi-Fi védett elérés***). Megmutattam, hogy erőteljes és rugalmas hitelesítéssel, továbbá dinamikus titkosítással és kulcsegyeztetéssel hogyan foltozza be a *WEP* protokoll biztonsági réseit. Szó volt arról is, hogy a *WPA* összetevő-protokolljai, köztük a ***802.1x***, valamint az *EAP* és a *RADIUS* különféle változatai milyen kapcsolatban állnak egymással. Ebben a hónapban megmutatom, hogy *FreeRADIUS* és *OpenSSL* használatával hogyan helyezhetjük üzembe *WPA*-ra vagy egyéb ***802.1x*** alapú megoldásra épülő környezetünk hitelesítő kiszolgálóját.

Gyors áttekintés

A *WPA*, mint remélem, sokan emlékeznek, modulárisabb, mint a *WEP*. Míg *WEP* használatakor a hitelesítés és a titkosítás egy az összes ügyfél által közösen használt titkos kulcs segítségével történik, *WPA* használatakor a hitelesítés alapesetben a ***802.1x*** protokoll használatával folyik. A *WEP*-re sokban hasonlító előre megosztott kulcs (***pre-shared key, PSK***) mód alkalmazása egy másik lehetőség. A *WPA* esetében az egyes ügyfelek egyedi titkosító kulcsait a hozzáférési pont állítja elő, és rendszeresen frissíti.

A ***802.1x*** egy rugalmas, az *EAP*-ra (***Extensible Authentication Protocol***, ***bővíthető hitelesítő protokoll***) épülő hitelesítő protokoll. A *WPA*-képes termékek az *EAP* számos különböző változatát támogatják, köztük az *EAP-TLS*-t és a *PEAP*-t. Aki a ***802.1x*** elhagyását és a *WPA* egyszerűbb, *PSK* módban történő használatát választja, amely ugyan biztosítja a titkosító kulcsok dinamikus előállítását, ám a hitelesítő adatokat hozzáférhetően, nyílt szövegként továbbítja, annak csupán annyit kell tennie, hogy azonos előre megosztott kulcsot állít be a hozzáférési ponton és a vezeték nélküli ügyfeleken.

Ha viszont a ***802.1x*** jóval erőteljesebb hitelesítési eljárásainak alkalmazásával a *WPA* teljes tudását ki akarjuk használni, akkor szükségünk van egy *RADIUS* kiszolgálóra. Erre a célra léteznek kereskedelmi megoldások is, mint például a *Funk Software Steel Belted RADIUS* programja, de annak sem kell elkeseredni, aki a nyílt forrású programokat pártolja, a *FreeRADIUS* ugyanis az *EAP* minden fontosabb változatát támogatja, és ugyanolyan üzembiztos és biztonságos. Nézzük, hogyan bírhatjuk munkára.

A használati környezet

Természetesen nincs elég helyem arra, hogy a *FreeRADIUS* és a ***802.1x*** együttes használatának minden lehetséges módját ismertessem, sőt, még a vezeték nélküli alkalmazásokat sem tudom mindenre kiterjedően tárgyalni. Nézzünk tehát egy példakörnyezetet, amelyet az alábbi eljárásokkal fogunk létrehozni.

A *WPA* használatba vétele kapcsán a legfontosabb döntés a használni kívánt *EAP*-változat kiválasztása. E tekintetben nemcsak a *RADIUS* kiszolgáló alkalmazás, de az ügyfelek képességei is jelenthetnek korlátozást. A vezeték nélküli hozzáférési pont – érdekes módon – *EAP*-független, feltéve persze, hogy támogatja a ***802.1x***-et és/vagy a *WPA*-t. Egyszerűen közvetíti az *EAP*-forgalmat az ügyfelek és a kiszolgálók között, az *EAP* egyik altípusának kifejezett támogatására sincs szüksége. Az, hogy az ügyfélrendszer pontosan mit képes támogatni, a rajta futó operációs rendszertől és a vezeték nélküli hozzáférést biztosító hardvereszköztől egyaránt függ. Egy ***Microsoft Windows XP*** rendszer ***Intel Pro/2100 (Centrino)*** lapkakészlettel például támogatja az *EAP-TLS*-t és a *PEAP*-t, de az *EAP-TTLS*-t nem. Ha ***Linuxot*** és ***wpa_supplicantot*** (lásd az internetes forrásokat) használunk, akkor jóval szélesebb körű választási lehetőséget kapunk.

Példámban az *EAP-TLS* használatát fogom feltételezni.

Az *EAP-TLS* alkalmazásához az ügyfeleknek tanúsítványokkal kell rendelkezniük, amihez viszont szükségünk lesz egy ***hitelesítő szervezetre (certificate authority, CA)***. A nehézségek ellenére mégis érdemes az *EAP-TLS* mellett dönteni. Először is, széles körben támogatott. Másodszor, a ***TLS (X.509 tanúsítvány)*** alapú hitelesítés erőteljes biztonságot nyújt. Harmadszor, valójában nem kell túlságosan sok munka ahhoz, hogy *OpenSSL* segítségével összeállítsuk saját *CA*-nkat. Példakörnyezetünkben tehát egy *EAP-TLS*-t alkalmazó, *Windows XP*-t futtató ügyfél fog csatlakozni egy *WPA*-képes hozzáférési ponthoz. A hozzáférési pont egy *Linuxon* üzemelő *FreeRADIUS 1.0.1* kiszolgálónak fogja továbbadni a hitelesítési feladatokat.

1. kódrészlet Az openssl.cnf módosítása optimális tanúsítványkészítéshez

```
# Először módosítjuk a CA a CA_default részben szereplő gyökér elérési útját
# a létrehozni kí ánt CA-nak megfelelően
[ CA_default ]
dir             = ./micksCA               # Mindent itt tárolunk
# Az alábbi sorok kicsit lejjebb  annak az openssl.cnf-ben:
countryName_default             = US
stateOrPro inceName_default     = Minnesota
0.organizationName_default      = Industrial Wiremonkeys of the World
```

A FreeRADIUS beszerzése és telepítése

SuSE 9.2, ***Fedora Core 3*** és ***Red Hat Enterprise Linux*** alá külön, saját ***FreeRADIUS RPM*** csomag létezik, ***freeradius*** névvel. A ***Debian Sarge (Debian-testing)*** ugyanilyen nevű ***DEB*** csomaggal rendelkezik. ***Red Hat***, ***Fedora*** és ***Debian-testing*** alatt további csomagok is rendelkezésünkre állnak, ha ***MySQL*** adatbázist szeretnénk használni a hitelesítésre. Emellett a ***Debian-testing*** néhány további szolgáltatást is kínál, ezek újabb csomagokba kerültek. Mind a négy terjesztésre igaz azonban, hogy magához a ***802.1x*** alapú hitelesítéshez csupán az alap ***freeradius*** csomagra van szükség. Ha kedvenc ***Linux***-terjesztésünkhöz nincs ***FreeRADIUS*** csomag, vagy annak változata nem elég új az igényeinknek, akkor töltsük le a legújabb ***FreeRADIUS*** forráskódot a fejlesztők webhelyéről (lásd a forrásokat).

A ***FreeRADIUS*** lefordítása egyszerű, a megszokott `./configure - make - make install` eljárással történik. Aki még nem nagyon fordítgatott programokat, az a forráscsomag ***INSTALL*** fájljában részletesebb útmutatást is talál. A `configure` és a `make` parancsot lehetőleg normál felhasználóként adjuk ki, root jogokra csak a `make install` futtatásához van szükség.

Megjegyzem, hogy alapesetben a parancsfájl a ***/usr/local*** könyvtár alkönyvtáraiba telepíti a ***FreeRADIUS***-t. Mivel a ***Makefile*** eltávolításra nem alkalmas, javaslom a telepítési könyvtár változatlanul hagyását, így ugyanis – ha valamiért szükségtelenné válna – később könnyebb eltávolítani a ***FreeRADIUS***-t.

Hitelesítő szervezet létrehozása

Mielőtt megadnánk a ***FreeRADIUS*** beállításait, létre kell hoznunk néhány tanúsítványt. A tanúsítványok létrehozásához viszont szükség van a hitelesítő szervezetre. ***Linux Server Security*** című könyvem 5. fejezete tartalmaz egy ***„How to Become a Small-Time CA" (Kisméretű CA üzembe helyezése)*** című részt, ami részletesen tárgyalja a témát; itt most csak néhány szóban foglalnám össze az eljárás menetét.

Először is, mi az a ***CA***, és hol kell elhelyezkednie? A ***CA*** egy a nyilvános kulcs infrastruktúra gyökereként üzemelő rendszer. Központi hatóság, amely digitális aláírások alkalmazásával jótáll a szervezeten belül kibocsátott tanúsítványok hitelességéért. Rendszeres időközönként ***tanúsítvány visszavonási listákat (certificate revocation list, CRL)*** bocsát ki, ezek azokat a tanúsítványokat sorolják fel, amelyekért a CA többé nem szavatol. Ilyenek például a vállalattól kilépett munkatársak vagy az üzemen kívül helyezett kiszolgálók tanúsítványai.

Egyik feladat ellátásához sincs szükség arra, hogy a ***CA*** ténylegesen kiszolgálóként üzemeljen, sőt, jobb is, ha nem az. Egy ***CA*** akkor megbízható, ha gondosan védjük az illetéktelen hozzáférésektől, visszaélésektől. Saját ***CA***-imat éppen ezért egyre inkább olyan rendszerekre telepítem, amelyekkel csak időszakosan lépek fel a hálózatra, például ***VMware*** virtuális gépekre.

Lehetséges, hogy már rendelkezünk ***CA***-val, segítségével webkiszolgálók, ***stunnel*** vagy egyéb, ***TLS***-t használó alkalmazásokhoz is létrehozhattunk már tanúsítványokat. Ha ez a helyzet, akkor a ***CA*** a ***WPA***-hoz is megfelel. Ha nincs ilyenünk, akkor lássuk, hogyan helyezhetjük üzembe a ***CA***-t. Először ellenőrizzük, hogy a kiszemelt rendszerre telepítve van-e az ***OpenSSL***. Az ***OpenSSL*** minden ismertebb ***Linux***-terjesztésnek része, ahogy a ***FreeBSD***, az ***OpenBSD*** stb. is tartalmazza. Az ***OpenSSL*** meglétét a legegyszerűbben a `which openssl` paranccsal ellenőrizhetjük, mely megadja, hogy hová van telepítve gépünkön az ***OpenSSL*** – ha telepítve van. Lépjünk át abba a könyvtárba, ahol rendszerünk az ***OpenSSL*** beállító és tanúsítványfájljait tárolja. ***SuSE*** alatt ez a ***/etc/ssl***, de a pontos könyvtár terjesztésenként változhat. Ha megkeressük az `openssl.cnf` fájlt, akkor valószínűleg jó helyre fogunk kerülni.

Most nyissuk meg valamilyen szövegszerkesztőben az ***openssl.cnf*** fájlt. Néhány alapbeállítást át kell írnunk, így később gyorsabb lesz a tanúsítványok előállítása. Az *1. kódrészlet* az ***openssl.cnf*** módosítandó sorait tartalmazza. Ezután a ***CA*** tanúsítványkészítő parancsfájlját kell átírnunk, a ***CA*** alapértelmezett ***demoCA*** gyökérkönyvtára helyett az ***openssl.cnf*** fájlban a `dir` változónál választott könyvtárat kell megadnunk itt is. Én a ***CA.sh*** parancsfájlt használom, ez ***SuSE*** rendszereken a ***/usr/share/ssl/misc*** könyvtárban található; más rendszereken lehetséges, hogy máshova kerül. A módosítandó sor a következő:

`CATOP=./micksCA`.

Miután átírtuk a fájlt, lépjünk vissza az ***SSL*** beállítások könyvtárába, ami például a ***/etc/ssl*** lehet. Innen indítsuk el a ***CA.sh*** parancsfájlt a `-newca` kapcsolóval. Például: `/usr/share/ssl/misc/CA.sh -newca`. Ekkor módot kapunk új gyökértanúsítvány létrehozására és a hozzá tartozó titkos kulcs jelszavának megadására. Lehetőleg nehezen kitalálható jelszót válasszunk, és jegyezzük fel valamilyen biztonságos helyre; ha elveszítjük, képtelenek leszünk használni a ***CA***-t. Miután a parancsfájl végzett, az ***SSL*** beállításkönyvtárban lennie kell egy új könyvtárnak, példánknál maradva `micksCA` névvel. Ennek a könyvtárnak a gyökérszintjén

2. kódrészlet Az xpextensions fájl tartalma

```
[ xpclient_ext]
extendedKeyUsage = 1.3.6.1.5.5.7.3.2
[ xpserver_ext ]
extendedKeyUsage = 1.3.6.1.5.5.7.3.1
```

találjuk új *CA*-nk nyilvános tanúsítványát, a fájl neve alapesetben ***cacert.pem***. Mint később még lesz róla szó, ezt a fájlt át kell másolnunk a ***FreeRADIUS*** kiszolgálóra és minden egyes vezeték nélküli ügyfélre. Ha ***Windows XP*** ügyfelekkel dolgozunk, akkor a tanúsítványok létrehozása előtt egy további lépést is végre kell hajtanunk. A ***Windows XP*** bizonyos jellemzőket elvár az ügyfél- és a kiszolgálótanúsítványoktól egyaránt, ezért a 2. *kódrészletben* látható sorokkal létre kell hoznunk egy ***xpextensions*** nevű fájlt.
Az ***xpextensions*** fájlra a később szereplő ***OpenSSL*** parancsok egy része hivatkozni fog. Az ***openssl.cnf*** fájllal azonos könyvtárban kell lennie.

Az EAP-TLS működése

EAP-TLS használatakor a vezeték nélküli ügyfél és a ***RADIUS*** kiszolgáló kölcsönösen hitelesítik egymást. Mindkettő bemutatja saját tanúsítványát, majd kriptográfiai eszközökkel ellenőrzik, hogy a tanúsítványok rendelkeznek-e a vállalat hitelesítő szervezetének aláírásával. Tulajdonképpen ez a hitelesítés kezelésének egy egyszerű és elegáns módja. Miután telepítettük a *CA* nyilvános tanúsítványát a ***FreeRADIUS*** kiszolgálóra, az ügyfelek egyéb adatait, mint a felhasználónév és a jelszó, már nem kell kézzel megadnunk.
Persze szó sincs arról, hogy az ***EAP-TLS*** használata kevesebb munkával járna, mint a felhasználónév-jelszó alapú megoldásé, ugyanis az ***OpenSSL*** segítségével az összes felhasználóhoz létre kell hoznunk egy tanúsítványt, majd ezeket át kell másolnunk az ügyfélgépekre. Arra is ügyelnünk kell, hogy a megfelelő helyre telepítve mindenki rendelkezzen a gyökér CA tanúsítvány egy másolatával.

Tanúsítványok létrehozása

EAP-TLS használatakor a *CA* tanúsítványa mellett legalább kettő további tanúsítványra is szükségünk van, mégpedig egy kiszolgálótanúsítványra a ***FreeRADIUS*** kiszolgálóhoz, továbbá egy-egy ügyféltanúsítványra a hálózat minden vezeték nélküli ügyfeléhez. A tanúsítványok létrehozása három lépésből áll:

1. Aláírási kérvény létrehozása, ez lényegében egy aláíratlan tanúsítvány.
2. Az aláírási kérvény aláírása a *CA* kulcsával.
3. Az aláírt tanúsítvány átmásolása arra az állomásra, amelyik használni fogja.

A kiszolgálótanúsítvány aláírási kérvényét az ***OpenSSL*** `req` parancsával készíthetjük el:

```
$ openssl req -new -nodes -keyout
➥kiszolgalo_kulcs.pem -out kiszolgalo_kerveny.pem
➥-days 730 -config ./openssl.cnf
```

A parancs két fájlt hoz létre, a tényleges kérvényt, vagyis az aláíratlan tanúsítványt tartalmazó `kiszolgalo_kerveny.pem`-et, valamint a jelszó nélküli titkos kulcsot tartalmazó `kiszolgalo_kulcs.pem`-et. Előbb persze meg kell adnunk szervezetünk ***országkódját (Country Code)***, ***államát (State)*** stb., ezeknél sokszor az ***openssl.conf*** fájlban megadott alapértékeket is használhatjuk. A ***közös névvel (Common Name)*** már más a helyzet. Amikor a gép bekéri, írjuk be kiszolgálónk teljesen minősített tartománynevét, mint például ***kiszolgalo.wiremonkeys.org***.
Ezután a *CA* kulcsot használva, az ***OpenSSL*** `ca` parancsával aláírhatjuk a kérvényt:

```
$ openssl ca -config ./openssl.cnf \
-policy policy_anything -out
➥kiszolgalo_tanusitvany.pem \
-extensions xpserver_ext -extfile ./xpextensions \
-infiles ./kiszolgalo_kerveny.pem
```

A parancs beolvassa a `kiszolgalo_kerveny.pem` fájlt, bekéri a *CA* kulcsához tartozó jelszót, majd a `kiszolgalo_tanusitvany.pem` fájlba elmenti a kérvény aláírt változatát és a hozzá tartozó titkos kulcsot. Felhívnám a figyelmet a `-extensions` és `-extfile` kapcsolóra; ezek miatt hoztuk létre korábban az `xpextensions` fájlt.
Nyissuk meg valamilyen szövegszerkesztőben az aláírt tanúsítványt, és minden a `-----BEGIN CERTIFICATE-----` sor előtt lévő dolgot töröljünk belőle. Másoljuk össze egyetlen fájlba a tanúsítványt és a kulcsunkat:

```
$ cat kiszolgalo_kulcs.pem
➥kiszolgalo.tanusitvany.pem > \
kiszolgalo_kulcstanusitvany.pem
```

Van tehát kulccsal kiegészített kiszolgálótanúsítványunk, ezt kell átmásolnunk a ***FreeRADIUS*** kiszolgálóra. Titkos kulcsa nincs jelszóval védve, ezért miután minden a helyére került, minden felesleges másolatát töröljük.
Most az ügyféltanúsítvány aláírási kérvényét kell összeállítanunk. Az erre a célra szolgáló ***OpenSSL***-parancs hasonló a kiszolgálótanúsítvány létrehozására használthoz:

```
$ openssl req -new -keyout ugyfel_kulcs.pem \
-out ugyfel_kerelem.pem -days 730 -config
➥./openssl.cnf
```

Mint látható, az aláírási kérvényt és a kulcsot rendre az `ugyfel_kerveny.pem` és az `ugyfel_kulcs` fájlba írjuk. A kiszolgáló aláírási kérvényével ellentétben a `-nodes` kapcsoló itt elmaradt, ezért a parancs futtatásakor meg kell adnunk a tanúsítvány titkos kulcsának titkosításához használt jelszót.
A következő lépésben aláírjuk az ügyféltanúsítvány aláírási kérvényét:

```
$ openssl ca -config ./openssl.cnf \
-policy policy_anything -out
➥ugyfel_tanusitvany.pem \
-extensions xpclient_ext -extfile ./xpextensions \
-infiles ./ugyfel_kerveny.pem
```

Ismétlem, a parancs hasonló a kiszolgáló esetében használthoz, kivéve azt, hogy a `-extensions` parancs az

`xpextensions` fájl egy másik szakaszára hivatkozik. Ha az ügyfeleken ***Linux*** fut, akkor a `kiszolgalo_tanusitvany.pem` fájlnál látott módon törölni kell belőle a felesleges részeket. A tanúsítványt és a kulcsot külön fájlban is hagyhatjuk, de össze is fűzhetjük. Ezután másoljuk az ügyféltanúsítvány fájlját vagy fájljait a linuxos ügyfélgépre.
Ha egy tanúsítványt ***Windows XP***-t futtató ügyfélen szeretnénk használni, még egy lépést el kell végeznünk: ***PKCS12*** formátumúra kell hoznunk a tanúsítványfájlt. A szükséges parancs a következő:

```
openssl pkcs12 -export -in ugyfel_tanusitvany.pem \
-inkey ugyfel_kulcs.pem -out ugyfel_tanusitvany.p12
➥-clcerts
```

Meg kell adnunk az `ugyfel_kulcs.pem` jelszavát, majd az új fájl jelszavát. Ha gondoljuk, akár ugyanazt a jelszót is használhatjuk. Csábító lehetőség, hogy a jelszó begépelése helyett csupán lenyomjuk az ***Entert***, különösen, ha azt nézzük, hogy a ***Windows XP WPA***-kérvényezője csak akkor működik, ha a tanúsítványait jelszavak nélkül tároljuk. Nagyon, nagyon rossz ötlet ez, a titkos kulcsokat nem szabad védtelenül másolgatni a hálózaton keresztül. Nyomatékosan javaslom mindenkinek, hogy a jelszavakat csak akkor távolítsa el, ha a fájlokat már biztonságosan átmásolta a ***Windows XP***-t futtató ügyfelekre.
Gondolom, szinte kínálja magát az alkalom, hogy ennek kapcsán szidjuk egy kicsit a Microsoftot, de el kell árulnom, hogy a linuxos `Xsupplicant` és `wpa_supplicant` szintén csak úgy működik, hogy üres jelszót adunk meg, vagy elmentjük a jelszót nyílt szövegként egy beállító fájlba. Természetesen mindez ellentétben áll az igazán biztonságos tanúsítványkezelés elveivel. Remélem, hogy hamarosan elkészülnek azokat a ***WPA***-kérvényezők, amelyek indításkor képesek bekérni a felhasználótól a jelszavát.
A létrejött fájl, példánkban az `ugyfel_tanusitvany.p12` az aláírt tanúsítványt és a hozzá tartozó titkos kulcsot egyaránt tartalmazza. Ezt kell átmásolni a ***Windows XP*** alapú ügyfélgépre.

Összefoglalás

sTelepítettük a ***FreeRADIUS***-t, létrehoztunk egy hitelesítő szervezetet, előállítottuk a kiszolgáló és az ügyfelek tanúsítványait, majd átmásoltuk őket a megfelelő gépekre. Természetesen messze nem végeztünk még. Meg kell még adnunk a ***FreeRADIUS***, a hozzáférési pont és a vezeték nélküli ügyfelek beállításait. Erre a következő alkalommal kerítünk sort. Addig is mindenkinek biztonságot kívánok!

Linux Journal 2005. május, 133. szám

A cikk forrásai: ➲ ***www.linuxjournal.com/article/8134***

Mick Bauer (mick@visi.com)
Biztonsági szakember, a Linux Journal biztonsági témákkal foglalkozó szerkesztője, biztonsági tanácsadó a Minnesota állambeli Minneapolisban található Upstream Solutions LLC Inc.-nél.

Sunbird és iCalendar

A Mozilla Sunbird naptára egyesíti a központosított, csoportmunkát segítő webalkalmazások és a többféle operációs rendszer alatt is futtatható asztali eszközök előnyeit.

Amikor elkezdtem kiszolgáló oldali programok írásával foglalkozni, még a gondolatán is nevettem annak, hogy alkalmazásokat készítek. Tulajdonképpen csak kisebb kódokat írtam, és nem sejtettem, hogy egy valódi, valakinek az asztali számítógépén futó alkalmazás mi mindenre lenne alkalmas.

Persze a kezdetek óta sok minden megváltozott a számítógépes iparágban. Napjainkban a web alapú alkalmazások nem csupán mindennapi életünk megszokott elemei, de egyre fontosabb szerepet követelnek maguknak. Nemrég olyan alkalmazást kerestem, amivel el tudtam volna készíteni az amerikai bevételeimről szóló adóbevallásomat. Azt vártam, hogy jó néhány cégnél találok web alapú adószámító programot. Az ***ASP (application service provider, alkalmazásszolgáltató)*** kifejezés körül néhány éve szinte forrongott a szakma, sokan úgy gondolták, hogy weben keresztül bármilyen program használható.

Bár volt néhány nagyszerű sikertörténet, még több kudarcot láthattunk, amelyek mögött műszaki és üzleti okok egyaránt húzódtak.

Azt, hogy a vállalatok számára miért vonzó a web alapú alkalmazások használata, könnyű megválaszolni: az alkalmazásokat többé nem kell különféle gépeken és operációs rendszereken tesztelni, elég csupán néhány böngészővel kipróbálni őket. Nincs többé szükség a szoftverek különféle változatainak támogatására, ugyanis egyszerre mindig csak egy változat érhető el. A hibajavításokat és a frissítéseket gyakorlatilag folyamatosan be lehet építeni a rendszerbe. A szoftverek internetkapcsolat birtokában bárhonnan elérhetők, nem csupán arról a gépről, amelyikre telepítették őket. A lista egyre csak gyarapodik és gyarapodik. Sok szempontból ennek a megoldásnak sokkal több értelme van, mint több ezer CD-lemez legyártásával és a szoftverek több száz különféle gépen való tesztelésével vesződni, miközben fenn kell tartani egy nagyméretű, az összes létező változat támogatására képes központot is.

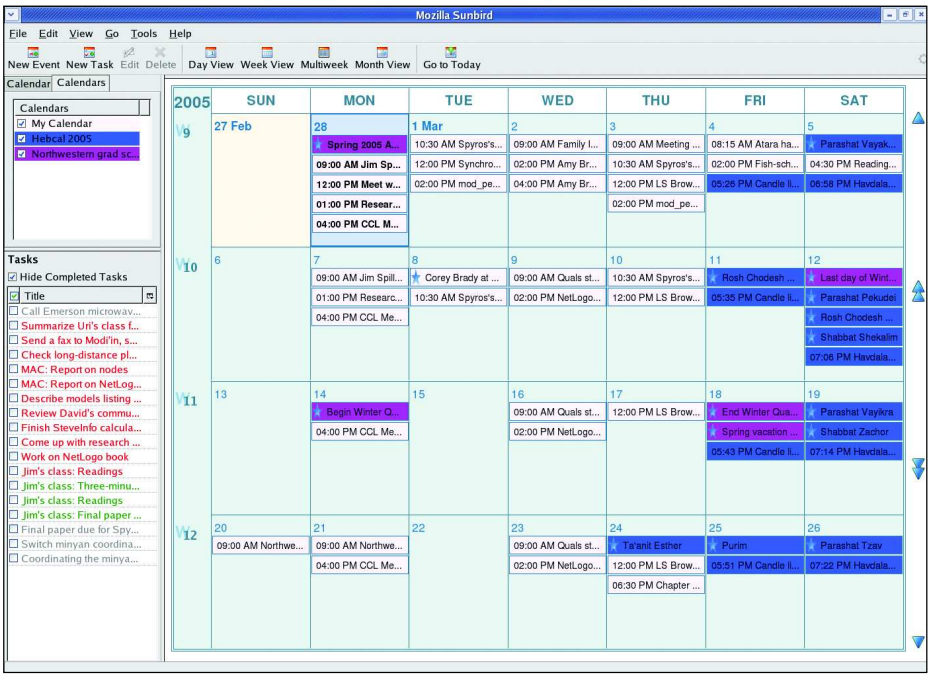
Elfeledkeztünk azonban arról, hogy a webes alkalmazások asztali párjukhoz képest korlátozottabb képességűek. Mivel minden komolyabb adatfeldolgozás a kiszolgálón történik, ide értve az adatbázisokban és a fájlokban végzett írási és olvasási műveleteket is, a felhasználói felülettől azonnali visszajelzést gyakorlatilag nem várhatunk. Hiába a leggyorsabb kiszolgáló és mindenféle varázslat, az ilyen programok mindig némi várakozásra kényszerítik használójukat. A Google új térképrendszere (lásd az internetes forrásokat) például kiválóan szemlélteti, hogy bár lehet, azért meglehetősen nehéz olyan webes alkalmazást készíteni, amely asztali megfelelőjével közel azonos viselkedést mutat.

Mivel a legtöbbünk nem rendelkezik a ***Google*** erőforrásaival, másféle megoldást kell találnunk, mégpedig hibrid alkalmazásokat kell használnunk – olyan asztali alkalmazásokat, amelyek nagymértékben támaszkodnak a webes technológiákra. Korábban a webes technológia fogalma elég jól körülhatárolható volt: ***HTML*** formázású dokumentumok halmaza, amelyeket ***HTTP***-n keresztül, ***URL***-ek segítségével érünk el. Sokáig kizárólag a webböngészők használták komolyabban ezeket a szabványokat.

Mára azonban egyre több asztali alkalmazás használ ***HTML***-t, ***HTTP***-t és ***URL***-eket, függetlenül attól, hogy a legkevésbé sem mondhatók webböngészőknek. ***URL***-ek segítségével keresik meg a távoli erőforrásokat, egyszerűségének és univerzálisságának köszönhetően ***HTML***-t alkalmaznak a hiperhivatkozásokkal összekötött dokumentumok létrehozására, a ***HTTP***-t pedig megbízható, egyszerű, általános célú és gyorstárazható működése miatt veszik igénybe. Sajnos túl sok példát nem tudok felhozni olyan szövegszerkesztőre vagy táblázatkezelőre, amely mindezeket a megoldásokat alkalmazná – persze ettől még lehetnek ilyenek –, csupán azt az egy programot, amely egyre nagyobb hangsúlyt kap az életemben, és ez a ***Mozilla Sunbird***. A ***Sunbird*** *(1. ábra)* a ***Firefox*** vagy a ***Thunderbird*** mellett kiegészítő jelleggel telepíthető naptár önálló változata.

A két említett programmal való együttműködése messze van a tökéletestől, és én például sokszor szeretném használni vagy újraindítani az egyiket a másik nélkül. Ezért múlt nyáron telepítettem a ***Sunbird***-öt, és azóta is örömmel fogadom minden újabb kiadását.

Most nyilván sokan azt gondolják, hogy semmi érdekes nincs egy webes technológiákat alkalmazó naptárban, csakhogy a ***Sunbird*** és az ***iCalendar*** szabvány segítségével nyilvános naptárakat is létre tudunk hozni. Ez alkalommal egy több hónapos, az ***iCalendar*** szabványra épülő naptárak létrehozását, terjesztését és megosztását tárgyaló sorozatot



1. ábra A Sunbird fő ablaka többhetes módban. Három naptáram közül kettő (Hebcal 2005 és Northwestern Egyetem) színkódolt, a beviteli panelen és a főablakban egyaránt. A tennivalóim listája is színes, így követni tudom, hogy melyik feladatomnak közeledik a határideje, melyikkel késtem el, mire kell kiemelt figyelmet fordítanom, illetve mi az, ami már folyamatban van.

indítunk el. Ennek során nemcsak azt tekintjük át, hogy hogyan dolgozhatunk az ***iCalendarrel***, de azt is megvizsgáljuk, hogy a hibrid alkalmazások milyen sokoldalú szolgáltatáskészletet és „felhasználói élményt” biztosítanak.

Az iCalendar és a Sunbird

Az ***iCalendar*** egy internetes szabvány naptáradatok számítógépek közötti megosztására. Alapötlete egyszerű: ha egy iroda minden munkatársa a saját gépén tartja nyilván teendőit, akkor az egyének ugyan hatékonyan tudnak dolgozni, ám a csoport semmivel sem jár jobban, mintha mindenki egy zsebben hordható határidőnaplót használna. Például az értekezletek ütemezése továbbra is nehézkes lesz. A közös eseményeket mindenki naptárába külön be kell írni, majd, ha például hétfőről szerdára tesszük át a megbeszélést, akkor mindenkinek egyénileg kell módosítania naptárja tartalmát.

Az ***iCalendar*** ezt a problémát maguknak a naptárfájloknak a szabványosításával oldja meg, amelyeket így szabadon lehet mozgatni a programok között. Az eredeti változat, már amennyire tudom, azt az elgondolást követve készült, hogy mindenki saját ***iCalendar*** alapú programot használ majd a gépén, a bejegyzéseket pedig hálózaton és interneten keresztül osztják meg egymással. A valóság ugyan kicsit nehezen követte az elméleteket, ám mára már több az iCalendar készletnek bizonyos részeit megvalósító program is létezik. Meg kell jegyeznem, hogy az egész ***iCalendar Project*** szerencsétlen és hibás névadások áldozata lett. Az adattároló és egyben az adatok cseréjére is használt fájlok a ***vCalendar*** nevet kapták, hasonlóan az elektronikus névjegykártyák ***vCard*** elnevezéséhez. Sokan – emberek és programok egyaránt –, ide értve a ***Sunbird***-öt is, a fájlformátumot ***iCalendar*** névvel említik, annak ellenére, hogy a fájlok ***vCalendarként*** azonosítják magukat. Innen már csak egy lépés volt az ***iCalendar*** kifejezés rövidítése ***iCalra***, ami különösen rossz ötlet volt, ugyanis az ***Apple Mac OS X*** operációs rendszere tartalmaz egy ***iCal*** nevű programot, ami a ***vCalendar*** fájlformátumot használja. Mivel a ***vCalendar*** formátum használata a jelek szerint kihalóban van, én a továbbiakban az ***iCalendar*** kifejezést használom mind a fájlformátum, mint a szabvány esetében. Töltsük le és telepítsük a különálló ***Sunbird***-változat megfelelő kiadását; az ***URL*** megtalálható a források között. Aki bátrabb, az a múlt éjjeli kiadások valamelyikét is telepítheti. Én, mivel a ***Sunbird***-öt éles naptáradatok kezelésére használom, inkább a hivatalos kiadásoknál maradok. Aki inkább ***Firefox*** vagy ***Thunderbird*** böngészőjével szeretné egyesíteni a naptárját, az lépjen át a fő letöltési oldalra, majd válassza ki a telepíteni kívánt kiterjesztést és változatot. A ***Firefox***- és ***Thunderbird***-bővítmények telepítése után a gazdaprogramot újra kell indítani.

A ***Sunbird*** alatt kétféle típusú elemet hozhatunk létre, eseményt és tennivalót. Az események normál esetben magában a naptárban tűnnek fel, a találkozóktól egészen a nyaralásokig minden ide tartozik. A tennivalók alapbeállítás szerint a képernyő bal oldalán sorakoznak, és elvégzendő feladatainkra emlékeztetnek, szükség esetén kezdő és záró dátummal kiegészítve. A ***Sunbird*** attól függően változtatja a tennivalók színét, hogy milyen hamar kell elvégezni őket. A késésben lévő feladatok színe piros, az aktuálisaké kék, a jövőbelieké pedig zöld. A szürke feladatok a távoli jövőbe esnek, míg az áthúzottakat – már ha egyáltalán megjelenítjük őket – már elvégeztük.

Az események és a tennivalók egyaránt ismétlődhetnek, vagyis ha a következő tíz hét minden szerdáján délután 4 órakor megbeszélésünk lesz, akkor ez egyetlen bejegyzéssel is meg tudjuk adni, nem kell tíz különálló eseményt bejegyeznünk. Az ismétlődő eseményekhez kivételeket is megadhatunk, továbbá ismétlésüket egyes napokra, hónapokra vagy évekre is korlátozhatjuk, ahol ezeket a bizonyos hónapokat, éveket mi adjuk meg.

iCalendar fájlok

Az, hogy a ***Sunbird*** hogyan kezeli az eseményeket és a feladatokat, tükrözi azok ***vCalendar*** fájlokban való tárolását. Bár elvárhatnánk, hogy egy korszerű internetes szabvány

XML-alapú legyen, az ***iCalendar*** fájlformátuma kettőspont-tal elválasztott név-érték párokra épül. Minden esemény és feladat saját kezdő és záró sorral rendelkezik, továbbá az egész fájltartalom is hasonló kezdő és záró sorok között található. Normál esetben az ***iCalendar*** fájlokban minden név-érték pár külön sorban található. Ha egy sort valamilyen nem látható karakterrel behúzunk, akkor az azt jelenti, hogy az adott sor az előző folytatása. Például:

```
név:érték
név2:
 érték2
név3
 :érték3
```

A fenti példában három név-érték párt láthatunk, mindegyikben másképp használjuk a nem látható karaktert. A ***Sunbird*** alapesetben a harmadik megoldást használja, vagyis minden nevet a saját sorába helyez, az egyes nevekhez tartozó értékek pedig a rákövetkező sorba kerülnek. A ***Sunbird***, mint a többi Mozilla termék is, minden adatfájlját egy profilkönyvtárba helyezi, amely a program első indításakor véletlenszerű névvel jön létre. Maguk az ***iCalendar*** fájlok a profilkönyvtár ***Calendar*** alkönyvtárába kerülnek.
Az ***iCalendar*** szépsége az, hogy nem muszáj minden naptáradatunkat egyetlen fájlban tartanunk, sőt, még egyetlen gépen sem. Az ***iCalendar***-megfelelő alkalmazások a beolvasásra megadott helyeken talál naptáradatok összesítését jelenítik meg. Vagyis a gépünkön több különböző naptárfájlt is el tudunk helyezni, tükrözve mindennapi életvitelünket; hogy csak a legalapvetőbb példánál maradjunk, szétválaszthatjuk személyes és munkával kapcsolatos dolgainkat. Egyéb forrásokból is lekérdezhetünk naptárfájlokat, például ***HTTP*** felett, vagyis a csoportnaptárakat központi kiszolgálókra is el tudjuk helyezni, miközben a megjelenítés marad a saját számítógépünkön.
A ***Sunbird*** első elindításakor, illetve az első naptár létrehozásakor létrehoz egy ***CalendarDataFile.ics*** fájlt. Ha egynél több naptárunk van, akkor rendszerünkön több ilyen fájlunk is lesz. A fájlok a ***CalendarDataFileN.ics*** nevet kapják, ahol az ***N*** a létrejött naptár számát adja meg.
Magának a fájlnak a szerkezete rendkívül egyszerű. Példaként nézzünk egy egyetlen eseményt – a Linuxvilág e havi leadási határidejét – tartalmazó ***iCalendar*** fájlt:

```
BEGIN:VCALENDAR
VERSION
 :2.0
PRODID
 :-//Mozilla.org/NONSGML Mozilla Calendar V1.0//EN
BEGIN:VEVENT
UID
 :05e55cc2-1dd2-11b2-8818-f578cbb4b77d
SUMMARY
 :Linuxvilág határidő
STATUS
 :TENTATIVE
CLASS
 :PRIVATE
X-MOZILLA-ALARM-DEFAULT-LENGTH
 :0
DTSTART
 :20050211T140000
DTEND
 :20050211T150000
DTSTAMP
 :20050209T132231Z
END:VEVENT
END:VCALENDAR
```

Mint látható, a fájl a `VCALENDAR` deklarációval kezdődik és végződik. Minden eseményt a `BEGIN:VEVENT` és az `END:VEVENT` elem határol. Minden eseményhez tartozik egy egyedi azonosító; egy összegzés, amely normál esetben a naptárban jelenik meg; egy állapot; egy osztály, amely azt adja meg, hogy másokkal meg szeretnénk-e osztani a naptárbejegyzést; továbbá egy kezdő és egy záró időpont. Az időbélyeg azt adja meg, hogy az eseményt mikor módosítottuk utoljára.
Az ***iCalendar*** fájlokban lévő időbélyegek szokatlan formátumot követnek, a dátumot ***ÉÉÉÉHHNN*** formában adják meg, ezt egy `T` karakter követi, majd a 24 órás formátumú időpont; a sort az elhagyható időzóna és egy `Z` karakter zárja. Mivel én jelenleg ***Chicagoban*** lakom, az időbélyeg nem azt az időpontot adja meg, amikor létrehoztam a bejegyzést, ehelyett a bejegyzés hat órával előrébb jár nálam, a `GMT` után egy órával (`1Z`).
Mi történik, ha minden hónapban van egy határidőm, és ezt ebben a naptári eseménybe is bele szeretném foglalni? A ***Sunbird*** alatt ezt úgy lehet megoldani, hogy az eseményszerkesztőben duplán rákattintva az eseményre megnyitjuk a ***Recurrence (Ismétlődés)*** lapot. Itt jelezhetjük, hogy az esemény minden hónapban megismétlődik, ekkor a felület megváltozik, és meghatározhatjuk, hogy – például – pontosan minden hónap 11-én (vagyis azonos dátumon) vagy minden hónap második péntekén, relatívan változva ismétlődik az esemény. Ha az első lehetőséget választjuk, akkor a ***iCalendar*** fájl tartalma így alakul:

```
BEGIN:VCALENDAR
VERSION
 :2.0
PRODID
 :-//Mozilla.org/NONSGML Mozilla Calendar V1.0//EN
BEGIN:VEVENT
UID
 :05e55cc2-1dd2-11b2-8818-f578cbb4b77d
SUMMARY
 :Linuxvilág határidő
STATUS
 :TENTATIVE
CLASS
 :PRIVATE
X-MOZILLA-ALARM-DEFAULT-LENGTH
 :0
X-MOZILLA-RECUR-DEFAULT-UNITS
 :months
RRULE
 :FREQ=MONTHLY;INTERVAL=1
```

```
DTSTART
 :20050211T140000
DTEND
 :20050211T150000
DTSTAMP
 :20050211T132231Z
LAST-MODIFIED
 :20050211T153505Z
END:VEVENT
END:VCALENDAR
```

Vegyük észre, hogy a bejegyzés egy RRULE tulajdonsággal bővült, melynek értéke FREQ=MONTHLY (*gyakoriság=havi*) és INTERVAL=1 (*intervallum*). Gondolnánk, hogy ha a határidő kéthetente volna, akkor elég lenne átírni a fájlt FREQ=WEEKLY (*gyakoriság=heti*) és INTERVAL=2 formába. Ez így is van, azzal a kikötéssel, hogy a BYDAY=FR mezőt is hozzá kell adnunk, ami jelzi, hogy az esemény péntekre esik (*FR, Friday*).
Ha azt választjuk, hogy az esemény minden hónap második péntekjére essen, az *iCalendar* fájl tartalma így alakul:

```
BEGIN:VCALENDAR
VERSION
 :2.0
PRODID
 :-//Mozilla.org/NONSGML Mozilla Calendar V1.0//EN
BEGIN:VEVENT
UID
 :05e55cc2-1dd2-11b2-8818-f578cbb4b77d
SUMMARY
 :Linuxvilág határidő
STATUS
 :TENTATIVE
CLASS
 :PRIVATE
X-MOZILLA-ALARM-DEFAULT-LENGTH
 :0
X-MOZILLA-RECUR-DEFAULT-UNITS
 :months
RRULE
 :FREQ=MONTHLY;INTERVAL=1;BYDAY=2FR
DTSTART
 :20050211T140000
DTEND
 :20050211T150000
DTSTAMP
 :20050211T132231Z
LAST-MODIFIED
 :20050211T153824Z
END:VEVENT
END:VCALENDAR
```

Mint látható, a RRULE tulajdonság most FREQ=MONTHLY (gyakoriság=havonta), az esemény ugyanis havonta következik be, míg az INTERVAL 1 lett. Ugyancsak felhívnám a figyelmet a BYDAY=2FR hozzáadására, ami arra utal, hogy az esemény minden hónap második péntekjén lép fel.

Végül használjuk ki a *Sunbirdnek* azt a képességét, hogy távoli naptárak kezelésére is képes – helyezzük át a fájlt a könyvtárból egy másik gépre. Én a *CalendarDataFile7.ics* fájlt – azért ez volt a neve, mert ezt a naptárat hetedikként hoztam létre – a */tmp* könyvtárba helyeztem. Ezután átmásoltam a weboldalamra, ahol a *http://reuven.lerner.co.il/CalendarDataFile7.ics URL* alatt tettem elérhetővé. A fájl elérhetőségét úgy ellenőriztem, hogy *wget*-tel letöltöttem. Mivel működött a dolog, bátran nekiláthattam, hogy az *URL*-t a *Sunbirdnek* is megadjam. Átváltottam a *Sunbirdre*, és töröltem az *ATF* naptárat.
A *Sunbird* meglévő, helyi naptár fájlnevét nem engedi eltávolítani. Ezután a Fájl menüből távoli naptár előfizetését választottam, majd megadtam a *.ics* fájl *URL*-jét. A naptár letöltése után a linuxvilágos határidő saját naptáramban minden hónapban láthatóvá vált, pontosan úgy, mintha az adatok a helyi gépen lettek volna. Sőt, ha a kísérlet végrehajtása után vetünk egy pillantást a *Calendar* könyvtárra, akkor láthatjuk, hogy a fájl *valóban* a helyi gépen található. Az alkalmazás letöltötte és telepítette a könyvtárba; kérésre természetesen frissíthető. Ehhez elég rákattintani az egér jobb gombjával a naptárra, majd a távoli naptár újratöltését választani.

Összefoglalás

A hibrid, web-asztali alkalmazások megismerését az *iCalendar/vCalendar* fájlformátum vizsgálatával kezdtük, a *Mozilla* önálló *Sunbird* alkalmazását hívva segítségül. Sikerült különféle típusú eseményeket létrehoznunk, majd az így létrejött *iCalendar* fájlt a helyi gépről áthelyeztük egy távoli kiszolgálóra.
Naptárunk egyelőre statikus, vagyis, ha valaki módosítani akarja, akkor azt kézzel vagy a módosított naptárfájl feltöltésével teheti meg. A következő alkalommal megnézzük, hogy webes, adatbázis alapú alkalmazással hogyan hozhatunk létre dinamikusan *iCalendar* fájlokat. Ezután megvizsgáljuk, hogy az egyik számítógépen létrehozott naptárakat milyen módszerekkel lehet átmásolni kiszolgálóra, illetve megosztani másokkal.

Linux Journal 2005. május, 133. szám

Reuven M. Lerner (➲ http://www.lerner.co.il/atf) Nyílt forrású programokra, valamint web- és adatbázis-alkalmazásokra szakosodott tanácsadó. Könyve, a Core Perl, 2002 januárjában jelent meg a Prentice Hall gondozásában. Reuven feleségével és lányaival nemrég költözött Chicagóba.

KAPCSOLÓDÓ CÍMEK

- ➲ www.mozilla.org/projects/calendar/download.html
- ➲ maps.google.com
- ➲ developer.apple.com/internet/appleapplications/icalendarfiles.html
- ➲ developer.apple.com/internet/appleapplications/icalendarfiles.html

FreeBSD – a szomszéd vár (8. rész)
Vállalati tűzfal készítése

A vállalati tűzfalak sokrétű feladatokat kell ellássanak: a gépek védelmén túl általában a belső hálózat címfordítását és a forgalom rendszabályozását is el kell végezniük. Ezen feladatokat nyugodt szívvel rábízhatjuk egy FreeBSD alaprendszerre, hiszen tartalmaz minden szükséges eszközt.

NAT – hálózati címfordítás

A hálózati címfordítással megbízott gép a belső hálózat gépeitől kapott csomagokat úgy módosítja, hogy azok feladójaként a saját címét tünteti fel: így a válaszként érkező csomagok is hozzá kerülnek vissza, amelyeket ismét módosítva – immár a belső hálózaton – az eredeti kérést indító géphez továbbít. A belső hálózaton lévő gépek ezáltal meg vannak védve minden kívülről induló támadás ellen, s mégis képesek teljes körű szolgáltatások elérésére (bár van néhány megkötés is). Linux rendszereknél az ***iptables***/***ipchains*** programok végezték a ***NAT*** beállításait, ***FreeBSD*** operációs rendszer esetén viszont erre egy külön program készült: az ***ipnat***. Felesleges keresgélnünk a ports adatbázisban, az ***ipnat*** az alaprendszer része, sőt a rendszermag is tartalmazza, ha az alaprendszerhez tartozó ***GENERIC*** kernelt használjuk. Az ***ipnat*** bekapcsolásához mindössze a

```
ipnat_enable="yes"
```

sort kell elhelyezni a ***/etc/rc.conf*** állományban, s a következő induláskor már használhatjuk is a címfordítást. Alapesetben a ***/etc/ipnat.rules*** állományból olvassa be a program a rá vonatkozó szabályokat, amely helyett más állományt is megadhatunk a ***/etc/rc.conf*** állományba írt

```
ipnat_rules="/root/ipnat.rules"
```

sorral. Ha újraindítás nélkül szeretnénk megúszni az első próbálkozásokat, akkor az

```
# ipnat -C -f /etc/ipnat.rules
```

parancsot kell kiadnunk, amely törli az előzőleg beolvasott szabályokat, majd a megadott állományból újakat olvas be. Ha az aktív ***NAT*** kapcsolatokat is törölni szeretnék a belső táblázatból, akkor a `-F` opciót is megadhatjuk:

```
# ipnat -C -F -f /etc/ipnat.rules
```

Érdemes „konzolközelben” végezni ezen tevékenységeket, mivel egy hibásan megállapított szabállyal könnyedén elvághatjuk magunkat a hálózaton történő további adminisztrációtól...

Alapvetően négy parancs áll rendelkezésünkre ahhoz, hogy a ***NAT*** szolgáltatásunkat megfelelően beállíthassuk, ebből kettőt használunk rendszeresen. ***iptables*** programhoz szokott kézzel először nehézkes lesz a megfelelő szintaxis megadása, pár perc után azonban nevetségesen egyszerűvé áll össze a kép.
Leggyakoribb tevékenység a belső hálózat címfordítása, amelyet rábízunk egy ***NAT*** szolgáltatásra. Ezt a `map` paranccsal tudjuk megtenni, mint a következő példában:

```
map xl0 192.168.1.0/24 -> 195.199.153.218/32
```

Ekkor az `xl0` eszközön át (amely ebből adódóan a külső hálózat felé kapcsolódik), a `192.168.1.0/255.255.255.0` belső privát hálózatot a `195.199.153.218` cím felé irányítjuk címfordítással. Ezzel minden szükséges adatot megadtunk az `ipnat` programnak, s a kliens gépek megfelelő beállításai után már eléri a belső hálózat az internet adta lehetőségeket. Néhány probléma felmerül a megoldással kapcsolatban, amelyek ismert hiányosságai a ***NAT*** szolgáltatásnak. Ha sok gépet engedünk ki egyetlen IP cím alatt, akkor szükségszerűen működésképtelenek azok a programok, amelyek ***PtP*** módon szeretnének működni (játékok, videó/audiókonferenciák, stb). Ennek oka az, hogy a külső hálózatról a tűzfal IP címe felé induló független (nem választ tartalmazó) csomagok nem tudnak bejutni a belső hálózatba, lévén „ismeretlenek” a tűzfal számára, illetve az általuk használandó hálózati kapu sincs nyitva. Ezen csorba részleges javítását a csomagok átirányítása nyújtja, amelyre az `rdr` parancs szolgál. A vállalati intranet külső elérését (például távmunka okán) a tűzfal lehetővé teheti a megadott címekről (lásd előző rész). Persze ehhez egy átirányítást kell készíteni, amely ezt lehetővé is teszi:

```
rdr xl0 195.199.153.218/32 port 80 -> 192.168.1.254
port 80 tcp
```

Ennek megfelelően az `xl0` eszközön érkező csomagot, amely a `195.199.153.218` IP címre érkezik (az ipfw is átengedte), s a TCP 80-as (http) kaput határozta meg célként, a `192.168.1.254` címmel rendelkező gép 80-as kapujára to-

továbbítja. Az átirányítás másik használati lehetősége a terhelés elosztása. Ekkor a tűzfalunknak az a dolga, hogy felváltva dobálja a kéréseket más-más belső hálózaton lévő gép számára:

```
rdr xl0 195.199.153.218/32 port 80 ->
192.168.1.254,192.168.1.253,192.168.1.252 port 80 tcp
```

Ennek megfelelően a három megadott kiszolgáló számítógép felváltva kapja a tűzfal IP címére érkező kéréseket.

DummyNet – hálózati esélyegyenlőség

Otthoni körülmények között ritkán támad arra igényünk, hogy az „egygépes" kis hálózatunkban a sávszélességet korlátozzuk. Vállalati közegben a felhasználók száma indokol bizonyos korlátozásokat, hiszen többnyire ugyan azzal az ***ADSL*** vonallal van internet kapcsolata az egész cégnek, mint amit – jó esetben – otthon is használunk. Ha egy felhasználó elkezd letölteni egy nagyobb állományt – például egy videót – azonnal használhatatlan lesz a többi résztvevő számára hálózat. Ez jó esetben nem okoz problémát, de ha internetes technológiát igénylő programokkal dolgozni kellene, akkor súlyos bevételkiesést is okozhat a hálózat felesleges terhelése.
Ha a sávszélességet növelni nem lehet, illetve az igényeket sem tudjuk szóbeli figyelmeztetéssel csökkenteni, akkor fizikailag kell beavatkozni, hogy a hálózat terhelését mérsékelni tudjuk. ***FreeBSD*** esetén az alaprendszer tartalmazza a ***DummyNet*** nevezetű eszközt, amely eredetileg hálózati tesztek elvégzésére szolgált, de napjainkra önálló életre kelt. A ***DummyNet*** képes a csomagok egy részét elveszejteni, késleltetni, illetve a hálózat sávszélességét csökkenteni; így kiváló lehetőséget adott a hálózatot használó programok tűrőképességének vizsgálatára. Ugyanakkor az `ipfw` programmal együttműködve lehetőségünk nyílik egyes felhasználók tűrőképességének vizsgálatára is, hiszen az általuk használt gépről indított csomagok egy részét el tudjuk veszejteni, késleltetni vagy a sávszélességüket csökkenteni...
A program használatához a rendszermagot újra kell fordítani a

```
options DUMMYNET
```

megadásával, amely sort például a */usr/src/sys/i386/conf/MYKERNEL* állományba kell írnunk. Ehhez sajnos újra kell indítani a gépet, de ebben a szomorú tényben bőven kárpótol majd minket – gonosz rendszergazdákat – a felhasználóink szomorúsága, amint működni kezd a sávszélesség személyre szabott csökkentése...
Mint említettem, a ***DummyNet*** az `ipfw` programba épül be, mégpedig a `pipe` kulcsszón keresztül, s így a tűzfalunkat beállító programocska a következő sorokat tartalmazhatja:

```
#!/bin/sh

/sbin/ipfw -q flush

# A helyi forgalom engedélyezése
/sbin/ipfw add 1 allow ip from 10.1.1.0/24 to any
/sbin/ipfw add 2 allow ip from me to 10.1.1.0/24
# A kiszolgáló forgalmának engedélyezése
/sbin/ipfw add 60100 allow ip from me to any
/sbin/ipfw add 60200 allow ip from any to me

# A főnök gépének sávszélessége
/sbin/ipfw add 257 pipe 257 ip from any to
➥10.1.1.1/32
/sbin/ipfw pipe 257 config bw 20kByte/s
# Az „nemszeretem" kolléga sávszélessége
/sbin/ipfw add 258 pipe 258 ip from any to
➥10.1.1.2/32
/sbin/ipfw pipe 258 config bw 100Byte/s
# A kedves titkárnéni sávszélessége
/sbin/ipfw add 259 pipe 259 ip from any to
➥10.1.1.3/32
/sbin/ipfw pipe 259 config bw 200kByte/s
```

Látványosan olvasható a ***DummyNet*** működése: megadjuk a megfelelő ***ipfw*** szabályt, a hozzá tartozó cső számát, majd felkonfiguráljuk az adott számú csövet. A `bw` kulcsszó jelenti a sávszélességet, a mértékegység magáért beszél. Ha késleltetést szeretnénk beállítani, azt a `delay` parancs segítségével tehetjük meg:

```
# /sbin/ipfw pipe 258 config delay 30000
```

Ennek megfelelően a közutálatnak örvendő kolléga fél percet kénytelen várni minden csomagjára. Folyamatos letöltésnél nem jelent sokat, de az interaktivitást igénylő programokkal sokat fog szenvedni. Ha nagyon gonosz módon állunk a kollégához, akkor még egy 90%-os csomagvesztéssel is megterhelhetjük a hálózati kapcsolatát:

```
# /sbin/ipfw pipe 258 config plr 0.9
```

Sajnos IP címhez tudunk csak rendelni ilyen ***DummyNet*** szabályokat, ha felhasználókat szeretnénk korlátozni s nem gépeket, akkor ennél sokkal nehezebb dolgunk lesz.

Melyik programot válasszam?

A ***FreeBSD*** alaprendszer és ***ports*** adatbázis ennél a pár programnál sokkal több eszközt tartalmaz, amelyek mind-mind ugyan arra a feladatra más-más módszerrel adnak megoldást, ezek felsorolására sem vállalkozok, a pontos működésüket sem tudom a hely hiányában bemutatni. Csak ajánlani tudom minden kedves ***FreeBSD*** felhasználónak, hogy próbálja ki a lehetőségeket és válassza a hozzá legközelebb álló programcsomagot. Nem érdemes az összes lehetőséget pontosan ismerni, elég egyet mélyebben, ahogy magam is csak az ipfw és az `ipnat` programokat használom gyakrabban (holott használhatnám az `ipf` és a `natd` programokat is :).

Auth Gábor (auth.gabor@enaplo.hu)

KAPCSOLÓDÓ CÍMEK

A FreeBSD projekt honlapja: ➲ http://www.freebsd.org
A magyar FreeBSD honlap: ➲ http://www.freebsd.hu
A magyar BSD honlap: ➲ http://www.bsd.hu
A kézikönyv magyar fordítása
➲ http://www.enaplo.hu/FreeBSD/handbook/

Az eFltk bemutatása (3. rész)

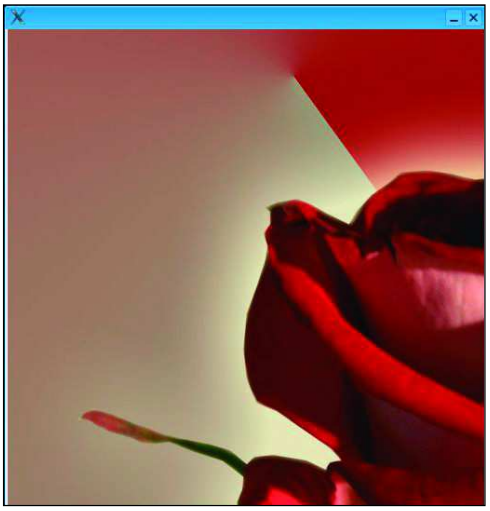
Saját vezérlőelemek készítése.

A sorozat harmadik részében bemutatom, hogyan készíthetünk saját vezérlőelemeket az ***eFltk*** megfelelő osztályainak felhasználásával. Sok esetben szükségünk lehet olyan grafikus elemekre, melyek nem találhatók meg a rutinkönyvtárban, gondoljunk csak egy folyamatosan mintavételezett adatokat megjelenítő grafikonra vagy akár egy kép nagyítását és eltolását lehetővé tévő elemre. Ezen oldalakon a későbbiekben az utóbbi példával fogom megvilágítani az ismeretek gyakorlati alkalmazását.

Mielőtt nekifognánk a megjelenítő osztály elkészítéséhez, hasznos lesz megismerkednünk a rendelkezésre álló rajzoló függvényekkel. Kezdjük az egyszerűbbekkel, mindenekelőtt ne felejtsük el beszúrni az *efltk/fl_draw.h* feljécállományt a forráskódba.

Lássuk tehát a színek beállítására szolgáló `fl_color()` függvényt, melyhez szükségünk lesz az *efltk/Fl_Color.h* állomány beszúrására is. A függvény első változatában megadhatunk három byte méretű összetevőt, melyből a függvény előállítja a szükséges `Fl_Color` típusú értéket. Használhatjuk a tizenhatos számrendszerben megadott színmeghatározás átalakítására is, amennyiben paraméterként csak egy `#RRGGBB` formájú (*HTML* dokumentumokban szokásos forma) karakterláncot adunk át a függvénynek. Amennyiben előre megadott színt szeretnénk használni, tekintsük át az *Fl_Color.h* állományt, melyben megtalálhatók az alapszínek (például `FL_RED`, `FL_GREEN`, `FL_BLUE`, `FL_GRAY`, `FL_YELLOW`, `FL_MAGENTA`, `FL_BLACK`, `FL_WHITE`) és használhatunk további színeket módosító függvényeket is.

Ilyen módosító függvény lehet például az `fl_lighter()` és az `fl_darker()`, melyek a megadott szín világosabb és sötétebb változatát adják vissza. A másik hasznos függvény ebben az állományban, az `fl_color_average()`, mely két szín súlyozott átlagát számítja ki. Az első két paraméterben a színeket adjuk meg, míg a harmadik paraméter a súlyozási tényező. Ez az érték 0.0-tól 1.0-ig vehet fel értékeket, minél közelebb áll az 1-es értékhez, a visszaadott szín annál inkább az első paraméterben megadott színhez fog hasonlítani.



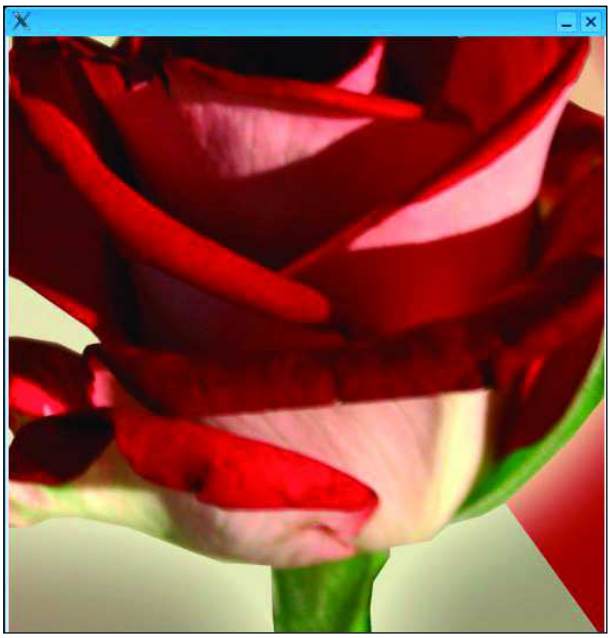
1. ábra A kiinduló kép

Miután a színekkel megismerkedtünk következzenek a rajzoló eljárások. Ezeket az eljárásokat általában a vezérlőelemek rajzolására használjuk, ahogyan azt majd a későbbiekben látni fogjuk. Az `fl_color()` eljárás alkalmas az aktuális rajzoló szín beállítására is, ilyenkor nem függvényként használjuk, hanem csak meghívjuk a megfelelő paraméterrel. Az `fl_line_style()` függvény alkalmas a rajzoláshoz használt vonal stílusának beállítására. Első paramétere a vonal stílusa, második a vonalvastagság, a harmadik pedig a szaggatott vonalak formáját meghatározó tömböt címző mutató. Ez utóbbi paraméternek általában `NULL` értéket adhatunk, hiszen rengeteg előre beállított vonalstílus közül válogathatunk. Ilyenek például az `FL_SOLID`, `FL_DASH`, `FL_DOT`, `FL_DASHDOT` és az `FL_DASHDOTDOT`. A vonalstílusokat kombinálhatjuk a vonalak végződését meghatározó `FL_FLAT`, `FL_ROUND` és `FL_SQUARE` értékekkel, amennyiben ezt nem adjuk meg, az ***eFltk*** rendszer az adott rendszerben leggyorsabban megvalósítható megoldást használja.

Egyszerű téglalapokat rajzolhatunk az `fl_rect()` függvénnyel, míg kitöltött téglalapokat az `fl_rectf()` fügvény használatával. Mindkét esetben a bal felső és a jobb alsó koordinátákat kell paraméterként átadni.

Az `fl_line()` függvény alkalmas vonalak rajzolására, a szokásos megoldás szerint a kezdő- és végpontok meghatározásával. Az `fl_point()` függvény segítségével pontokat rajzolhatunk a megadott koordinátán.

Körív rajzolására használható az `fl_arc()` függvény. Első négy paramétere meghatározza a körívhez tartozó téglalapot, melyben a kör vagy ellipszis elhelyezkedik, míg az ötödik és a hatodik paraméter a kezdő- és végszöget adja meg. Szintén használható körív rajzolására az `fl_pie()` függvény is, mégpedig a hetedik paraméterben megadott típus segítségével. Az első hat paraméter megegyezik az előző függvénynél használtakkal, azonban a hetedik egy konstans



2. ábra A képmegjelenítő a kép eltolása után

érték lehet. Ez az érték lehet `FL_ARC`, `FL_PIE` és `FL_CHORD`. Amennyiben nem adunk meg semmit, az alapértelmezés az `FL_PIE` és kitöltött körcikket kapunk eredményül.
Az `fl_circle()` függvénnyel kört rajzolhatunk. Első két paramétere a kör középpontja, míg a harmadik a kör sugara. Az `fl_ellipse()` négy paraméterével egy téglalapba foglalható ellipszist tudunk rajzolni.
Tömörítetlen képeket, melyek a memóriában találhatók, az `fl_draw_image()` függvény segítségével jeleníthetünk meg. A függvény első paramétere a képpontokat címző mutató, a következő négy határozza meg a megjelenítendő kép bal felső sarkának pozícióját és a kép méreteit. Az ötödik paraméter a képpontonkénti byte-ok száma, míg a hatodik egy konstans érték, mely a rajzoláskor minden sorban egy eltolást ad a forrásadatokhoz. Így valósítható meg például a függőleges irányú átméretezés vagy tükrözés. Amennyiben ez utóbbi értéket nem adjuk meg, az ***eFltk*** rutinok alapértelmezésben a kép szélességét és a képpontok mérete (byte-ban) alapján számítja ki a sorok eltolási értékét.
A rajzoló eljárások használata közben beállíthatunk vágási területet is. A terület beállítása után a függvények csak ezen a területen belül produkálnak látható eredményt. A vágási terület beállítására szolgáló eljárás az `fl_push_clip()`, melynek négy paramétere határozza meg a vágáshoz használandó téglalapot. A rajzoló műveletek befejezése után ne felejtsük el az `fl_pop_clip()` függvénnyel visszavonni a beállítást.
Az alábbiakban látható egy összefoglaló a rajzoláshoz használható függvényekről és paraméterezésükről.

```
fl_color(Fl_Color c)
fl_color(r, g, b)
Fl_Color fl_color()

fl_push_clip(x, y, w, h)
fl_pop_clip()

fl_line_style(style, width, *dashes)

fl_arc(x, y, w, h, a1, a2)
fl_circle(x, y, r)
fl_ellipse(x, y, w, h)
fl_pie(x, y, w, h, a1, a2, what)

fl_point(x, y)
fl_line(x1, y1, x2, y2)
fl_rect(x, y, w, h)
fl_rectf(x, y, w, h)
fl_rectf(x, y, w, h, Fl_Color c)
fl_rectf(x, y, w, h, r, g, b)

void fl_draw_image(*im, x, y, w, h, D, ld)
void fl_draw_image_mono(*im, x, y, w, h, D, ld)
```

Lehetőségeink megismerése után ideje hozzáfogni a saját vezérlőelemünk elkészítéséhez. Az objektum-orientált programozás szemléletének köszönhetően az ***eFltk***-ban már megvalósított elemeket felhasználhatjuk saját vezérlőelemek készítéséhez, amit most meg is teszünk. Az első példában az `Fl_Double_Window` lesz a kiindulási alap, és a célunk legyen egy olyan elem készítése, mely egérkattintáskor egy kört rajzol az ablakba. Amikor az egérgombot felengedjük, a kör már ne legyen látható.
Elsőként tehát határozzuk meg az osztályt. Az ősosztály az `Fl_Double_Window`, és célszerű bevezetni egy logikai változót a lenyomott állapot tárolására. Erre szolgál a `_pushed` osztálytag.
Tulajdonképpen az ***eFltk***-ban az új vezérlőelemek létrehozásakor két dologra kell figyelnünk. Gondoskodnunk kell az elemek kirajzolásáról és az események kezeléséről. Az alábbi listán látható a leszármazott osztály definíciója.

```
class myWindow:
 public Fl_Double_Window
{
private:
 bool _pushed;
public:
 myWindow (int w, int h);
 ~myWindow ();

protected:
 void draw ();
 int handle (int event);
};
```

A kirajzolásról gondoskodik a `draw()` osztályfüggvény, tehát itt kezeljük majd a lenyomott állapot és a felengedett állapot közötti megjelenésbeli eltérést. Ne felejtsünk el gondoskodni a változók kezdőértékének beállításáról, például az osztály konstruktorában. Szintén a konstruktorban kell gondoskodnunk az alaposztály életre keltéséről, tehát ne feledkezzünk meg az `Fl_Double_Window` konstruktorának meghívásáról. A rajzolást megvalósító függvény nagyon egyszerű, az alábbi néhány sor segítségével jól érthetővé válik.

```
myWindow::draw()
{
 // Ha az egészet újra kell rajzolni
 if (damage() & FL_DAMAGE_ALL) {
  fl_color(FL_WHITE);
  // alap téglalap kirajzolása
  fl_rectf(0, 0, w(), h());
  fl_color(FL_BLACK);
  // Csak ha lenyomva, akkor kell kör
  if (_pushed)
   fl_circle(w()/2, h()/2, h()/2-5);
 }
}
```

A másik fontos függvény, amit meg kell valósítanunk az eseménykezelő handle() függvény. Paraméterként kapja az esemény típusát. Az alábbi táblázatban látható az eFltk által kezelt események listája.

- FL_PUSH, FL_RELEASE egérgomb lenyomása, felengedése
- FL_DRAG egér mozgatása + lenyomott gomb
- FL_MOVE egér mozgatása
- FL_MOUSEWHEEL egérgörgő
- FL_ENTER, FL_LEAVE egérmutató az elem területén, elhagyja
- FL_FOCUS, FL_UNFOCUS billentyűzetfókusz, annak megszűnése
- FL_KEYUP, FL_KEYDOWN billentyű lenyomás-, felengedés
- FL_SHOW, FL_HIDE elem megjelenik, eltűnik

Ebben az egyszerű példában csak az FL_PUSH és az FL_RELEASE eseményre kell figyelmet fordítani, tehát az eseménykezelő függvény igen egyszerű lehet. Az ***eFltk*** mindaddig próbálja továbbadni az eseményeket az elemek között, amíg valamelyik le nem kezeli azokat. A handle() függvénynek kell tehát értesíteni a rendszert arról, hogy kezelte-e az eseményt vagy további feldolgozásra van szükség. Amennyiben az eseménykezelő 0 értékkel tér vissza, akkor szükség van további feldolgozásra, vagyis az esemény nem érdekes az aktuális elem szempontjából, míg ha nullától különböző a visszatérési érték, azzal jelezhetjük, hogy sikeresen feldolgoztuk az adott eseményt. Lássuk tehát a példabeli eseménykezelő függvényt.

```
int myWindow::handle(int event)
{
  switch (event) {
    case FL_PUSH:
      _pushed = true;
      redraw();
      return 1;
    case FL_RELEASE:
      _pushed = false;
      redraw();
      return 1;
    default:
      return Fl_Double_Window::handle(event);
  }
}
```

A fenti eseménykezelő eljárásban csak az egérgombra vonatkozó eseményeket kezeljük, mégpedig egyszerűen átállítjuk a lenyomást jelző _pushed változó értékét és újrarajzoljuk az ablakot. Amennyiben számunkra érdektelen eseményről van szó, meghívjuk az ős osztály eseménykezelő eljárását és visszatérünk az általa visszaadott értékkel.
Miután ezt az egyszerű példát megértettük, következzen egy összetett feladat megoldása. A feladat az, hogy olyan képmegjelenítő elemet készítsünk, melyben az egérgomb lenyomásával és az egér húzásával a képet mozgatni tudjuk. Néhány kiegészítő változót kell bevezetnünk, ahogyan az az osztálydefinícióban is látható. Szükségünk van egy Fl_Image objektumra, melyben tároljuk a megjelenítendő képet és egy kiegészítő képre, amire minden egérmozgatás után átmásoljuk a kép látható részét. Itt a sebességgel lehetnek apróbb gondok, de most az egyszerű megoldásra törekszünk. Szükségünk van továbbá egy eljárásra (copy_image_region()), ami a tényleges másolást elvégzi. Az eseménykezelő eljárásban az FL_DRAG eseményre kell figyelni, azonban ez az esemény csak akkor jut el az elemhez, ha az reagál az FL_PUSH és az FL_RELEASE eseményekre is.
Az eseménykezelő eljárásban kiszámítjuk az egér elmozdulását, átmásoljuk a kisebb képre az eredeti kép megfelelő területét és újrarajzoljuk a képet. Itt figyelnünk kell arra is, hogy a képnek csak a ténylegesen látható területével foglalkozzunk, vagyis ne csökkenhessen 0 alá az eltolást meghatározó érték és ne is növekedhessen olyan érték fölé, ami a kép túlcímzését eredményezné. Az eseménykezelő eljárás az alábbi listán látható.

```
int myWindow::handle (int event)
{
 int _dx, _dy;
 bool _changed = false;

 switch (event) {
 case FL_PUSH:
 case FL_RELEASE:
 return 1;
    case FL_DRAG:
      // koordináták kezdeti értéke
      if (_ox == 0 || _oy == 0) {
         _ox = Fl::event_x ();
         _oy = Fl::event_y ();
         return 1;
       }
      _dx = (_ox - Fl::event_x ());
      _dy = (_oy - Fl::event_y ());
      // ellenőrzés vízszintes irányban
      if (_sx+_dx>0 && _sx+_dx+this->w () <
               _image->width ()) {
         _sx += _dx; _changed = true;
       }
      // ellenőrzés függőleges irányban
      if (_sy+_dy>0 && _sy+_dy+this->h () <
               _image->height ()) {
         _sy += _dy; _changed = true;
       }
```

```
        // Csak akkor rajzolunk, ha változás volt
        if (_changed) {
           copy_image_region (_sx, _sy, w (),
           ➥h ());
           redraw ();
         }
         // koordináták frissítése
        _ox = Fl::event_x ();
        _oy = Fl::event_y ();
        return 1;
      default:
        return Fl_Double_Window::handle
        ➥(event);
      }
}
```

A rajzolást végző eljárásban egészen egyszerűen csak kirajzoljuk a már átmásolt képrészletet a grafikus elem megjelenítendő részére.

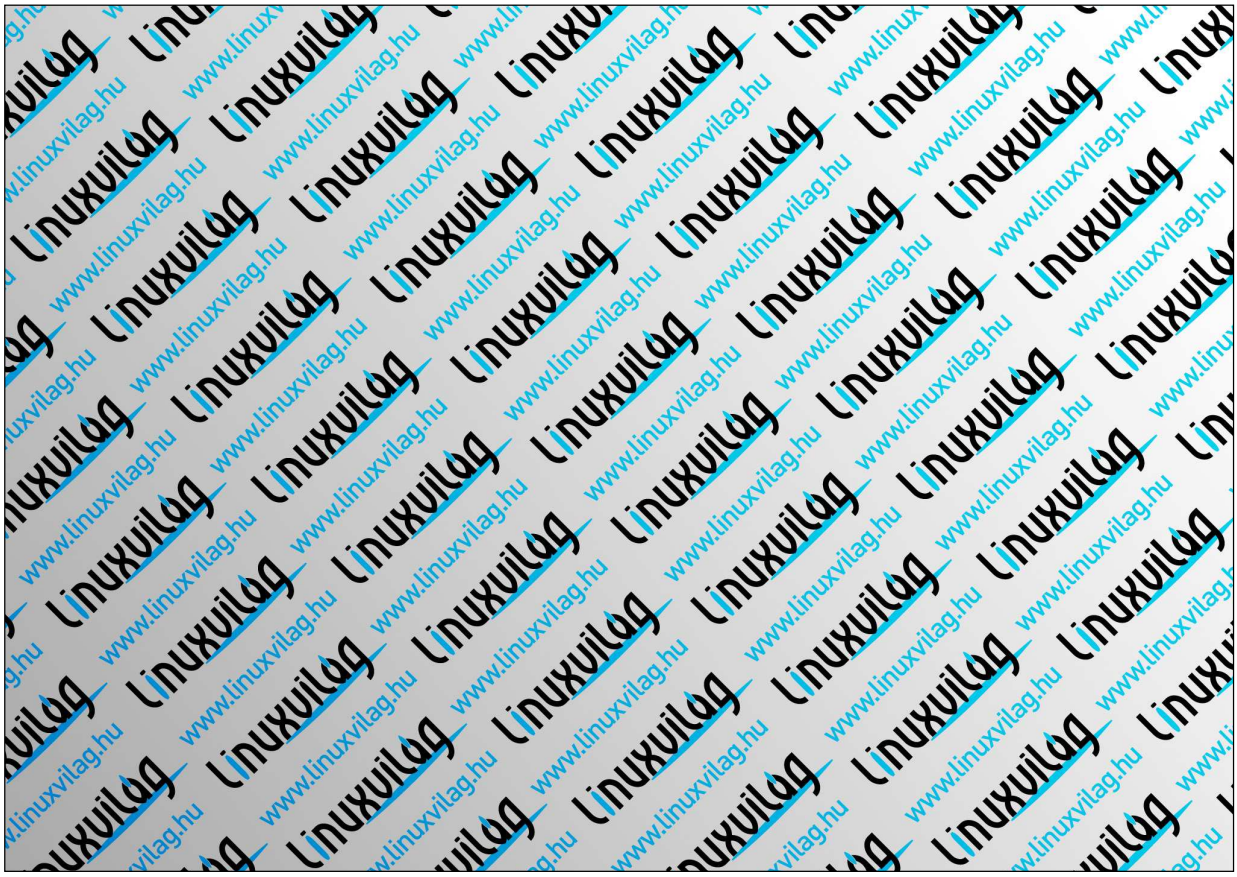
```
void myWindow::draw ()
{
 if (_cimage && _image)
 _cimage->draw (0, 0, w (), h ());
}
```

Amiről még szót kell ejtenem, a kép betöltését végző `set_image()` függvény. Az *eFltk*-ban egyszerűen betölthetünk *JPG*, *PNG* és *GIF* formátumú képeket, mégpedig az `Fl_Image::read(filename)` statikus metódus segítségével. A függvény egy *Fl_Image* típusú objektumot ad vissza, amit akár beállíthatunk valamilyen vezérlőelem képeként (az `Fl_Widget::image()`) függvény használatával) vagy további feldolgozás után tetszőleges módon feldolgozhatunk.

Összefoglalásképpen tekintsük át az új grafikus elem készítéséhez szükséges ismereteket. Először is kiválasztjuk a megfelelő ős osztályt. Kiegészítjük a szükséges osztálytagokkal a saját osztályunkat, majd megírjuk a kirajzolást végző `draw()` függvényt. Miután eldöntöttük, hogy milyen eseményekre kell reagálnia az elemnek, következhet a `handle()` eljárás elkészítése. Ebben az eljárásban a 0 visszatérési értékkel jelezzük az *eFtlk* rendszer számára, hogy az elem nem dolgozza fel a kapott eseményt, így azt továbbítani kell. A 0-tól eltérő érték jelzi, hogy az eseményt feldolgoztuk. Itt végül célszerű meghívni az ős osztály eseménykezelő eljárását és ha nem kezeltük az eseményt, akkor ennek visszatérési értékét felhasználni.

Végezetül láthatunk néhány képet, mely a cikk alapját képező programot próbálja szemléltetni működés közben. A program forráskódja szabadon hozzáférhető a *http://dzooli.uw.hu/efltk_draw.tgz* hivatkozáson keresztül. Mindenkinek kellemes időtöltést kívánva búcsúzom, elektronikus levél útján továbbra is szívesen válaszolok a felmerülő kérdésekre.

Fábián Zoltán



Kávéfőzés lépésről lépésre (1. rész)

Lubickolás Java partjainál, avagy programozzunk divatosan.

Ez a cikk egy olyan sorozatnak az első darabja, amely azoknak szól, akik egy keveset programoztak már C-ben, és szeretnék megtanulni, mit is jelent ez a manapság agyondicsért objektumközpontúság. A szükséges C előismeret igazán alapszintű, ezért aki még soha semmilyen programnyelvvel nem találkozott, kis töprengés árán annak is megérthetőek az alábbiakban közölt példák. Csupán egy kis türelemre, és egy adag kávéra lesz szükség, a többiről én gondoskodom.

Az objektumközpontúság egy programozási módszertan. Iránymutató gondolatok és elvek összessége, aminek elsődleges célja az, hogy a programozó munkáját megkönnyítse. Tudatosságra tanít, aminek a betartásával mind magunk, mind mások számára átlátható kódot, könnyen továbbfejleszthető programot kapunk. Talán azonnal érthető, hogy egy nagyobb szoftver készítésekor csapatjátékosként szempont a könnyen emészthető forráskód. Amit a legtöbben csak a saját kárukon tanulnak meg, az a másik fele. Saját magunkra is gondolnunk kell akkor, amikor programot írunk, mert eltelik egy hónap, egy év, akár több is, és egy sebtiben összecsapott kódról fogalmunk sem lesz, hogy mit takar.

Először is el kell, hogy szomorítsam az elméletek embereit: ez a sorozat nem a módszertanról, hanem annak gyakorlati alkalmazásáról fog szólni. Az elv legfontosabb fogalmait a Java szemüvegén keresztül fogom bemutatni. Azért erre a programozási nyelvre esett a választás, mert érzésem szerint a manapság széles körben használt objektumközpontú nyelvek közül ez a leginkább letisztult megvalósítás, és igen sok támogatásra lelhetünk az interneten bevezetők, kézikönyvek és fórumok formájában.

Mielőtt még belevágnánk a munkába, vessünk egy pillantást az alcímre, és az első pár mondatra. Noha nem merném állítani, hogy a szójátékok koronázatlan királya volnék, ez a rövid bevezető akkor is árulkodó. A következőkben használt programozási nyelv Java szigetéről kapta a nevét, amely igen híres kávéültetvényeiről. Így többen komoly valóságalapot tulajdonítanak annak a legendának, miszerint a készítők a nyelv kifejlesztése közben elfogyasztott kávé mennyisége okán hódoltak a névadással a szigetnek. Vágjunk is bele! A ***Sun***-féle ***Java*** megvalósítást fogjuk használni, mivel ez tekinthető a hivatalos és kiforrott változatnak.

Telepítés

Minden, ***Java***-val kapcsolatos igényünk kielégítést nyerhet a ***Sun Java*** oldalán (➲ ***http://java.sun.com/***), legyen szó könyvekről, ingyenes leírásokról, példaprogramokról, vagy magáról a fejlesztői csomagról. Ezért érdemes egy kicsit elidőzni az oldalon, esetleg feliratkozni a fórumra, ha komolyak a szándékaink a programozással kapcsolatban. Amire a cikk példáinak kipróbálásához szükség lesz, az egy ***JDK 5.0***.

A rövidítés a ***J2SE Development Kit (fejlesztői csomag)*** kifejezést fedi, amit még mindig lehet tovább boncolgatni. A ***J2SE*** jelentése ***Java 2 Standard Edition***. Hogy mi köze a 2-nek az 5.0-hoz, azt néhány további változatszám ismertetésével tudom csak érzékeltetni. Az 1.3-as változat óta a ***Sun Java 2*** néven is emlegeti szeretett gyermekét. Ezt követte az ***1.4***-es, amire még mindig illett a ***Java 2*** jelző, majd kisvártatva megérkezett az ***1.5***-ös változat. Ez már egyszerre ***Java 2***, ***1.5***, sőt mi több, 5.0. Aki követte, kérem e-mailben jelentkezzen.

Miután megbarátkoztunk a ***Sun*** változatszámkezelési logikájával, töltsük le a ***Java***-t az alábbi címről:
➲ ***http://java.sun.com/j2se/1.5.0/download.jsp***. Itt többféle összeállítással találkozhatunk. A ***NetBeans IDE + JDK 5.0*** egy olyan csomag, ami a fordítón kívül tartalmaz egy ***integrált fejlesztői környezetet*** (***Integrated Development Environment***) is. Ez gyönyörű, nagyon kényelmes, viszont kisebb programokhoz felesleges és egy kicsit lassú is. A ***JDK 5.0*** cím alatt tölthető le az, amire nekünk szükségünk lesz. Ugyanerről az oldalról elérhető még a ***JRE (J2SE Runtime Environment)***, ami kizárólag a futtatókörnyezetet jelenti. Erre azért érdemes odafigyelni, mert a ***Java*** kapcsán sokat hangoztatott felületfüggetlenség, azaz hogy szinte bármilyen operációs rendszer alatt módosítás nélkül fut ugyanaz az alkalmazás, azzal a kitétellel teljesül, hogy ez telepítve van. A cikk végére a kedves olvasó már elkészítette az első Java programját. Ha szeretné ezt másoknak is megmutatni, hívja fel a figyelmüket a futtatókörnyezet telepítésének szükségességére.

Meg kell még említenem, hogy a teljes ***API (Application Programming Interface)*** leírás szintén letölthető, ami erősen ajánlott, ha programozás közben nem szeretnénk újból feltalálni a kereket. Erre még később visszatérünk. Ami viszont minden ***Linux***-hívő szívét megmelengeti, az ezután jön: a teljes forráskód úgyszintén szabadon letölthető. Ennek letöltése a nagyon sok szabadidővel rendelkező Olvasóknak ajánlott.

Térjünk vissza arra, amiért meglátogattuk az oldalt. Töltsük le tehát a legújabb fejlesztői csomagot. A cikk írásának pillanatában ez a ***JDK 5.0 Update 3*** nevet viseli. A licensz szerződés elfogadása után már csak egy kattintásra vagyunk a hőn áhított csomagtól. ***Linux*** egy önkicsomagoló ***.bin*** állomány érhető el, ***RPM*** és sima változatban. A sima itt azt jelenti, hogy minden csomagkezelő nélkül csak beleömleszti egy könyvtárba a fájlokat. Mindkét változat 45 MB körüli méretet képvisel.

A nekünk jobban tetsző változat letöltése után egy grafikus telepítő lépésről lépésre végigvezet a szükséges lépéseken. Természetes igényként merülhet fel többünkben, hogy ne kelljen rendszergazdai jogosultsággal használni a grafikus felületet. Erre szolgálna a `-console` kapcsoló, amire azonban én a következő hibaüzenetet kaptam:

```
The wizard cannot continue because of the
➥following error: Invalid command line option:
➥console is not supported (1001) (403)
```

Könnyen elképzelhető, hogy ezt a hibát azóta kijavították. Ha nem így lenne, kénytelen kelletlen a grafikus telepítővel kell megküzdenünk a fejlesztői csomagért. Mindkét esetben szükséges még egy utolsó lépés a kényelmes használat érdekében. Miután több, mint valószínű, hogy nem egy olyan könyvtárba telepítettük a `java`, illetve a `javac` programokat, amely benne lenne a `PATH` környezeti változóban, érdemes ezekre egy szimbolikus hivatkozást létrehozni egy olyan könyvtárban, amit tartalmaz a `PATH`, például:

```
$ ln -s /opt/jdk1.5.0_03/bin/java
➥/usr/local/bin/java
$ ln -s /opt/jdk1.5.0_03/bin/javac
➥/usr/local/bin/javac
```

Hasonló eljárással érdemes még a `javadoc`-ra is létrehozni egy hivatkozást, mivel a későbbiekben még jól jöhet. Indulásnak azonban bőven elég ennyi.
Elég a telepítésből, próbáljuk már ki!

Első Java programunk

Először is indítsunk egy kényelmes szövegszerkesztőt. Grafikus felület alatt nyugodt szívvel tudom ajánlani a ***SciTE***-t. Ez egy nagyon gyors, ***GTK***-s eszközkészletet használó alkalmazás, ami mindent tud, amire csak szükségünk lehet, és a legtöbb terjesztés tartalmazza. Van benne színkiemelés, sorszámozás, és képes több forráskódot is kezelni egy, a ***Mozilla***-nál látott „füles” megoldással. Gépeljük tehát be az alábbi kódot a szövegszerkesztőbe:

```
public class Hello {
    public static void main(String[] args) {
        System.out.println(“Hello világ!”);
    }
}
```

Bizony, ez a jól ismert „Helló világ!” program. Mentsük el a forrásunkat ***Hello.java*** néven, majd konzolban adjuk ki a következő parancsokat:

```
$ javac Hello.java
$ java Hello
```

A `javac` a ***Java Compiler***, azaz a ***Java*** fordító. A `Hello.java` forrásból hoz létre egy ***Hello.class*** nevű, úgynevezett bájtkódot. Ez a forrás és a gépi kód között valahol félúton helyezkedik el, és ez az, amit bármelyik ***Java*** értelmező képes futtatni, legyen szó ***Windowsról***, ***Linuxról***, vagy akár egy ***Mac OS X***-ről. A második sorban meghívott `java` az előbb említett értelmező, vagy futtatókörnyezet, ahogy tetszik.
Figyelem: bár a Hello.class fájlt futtatjuk, nincs szükség a kiterjesztés megadására, a fent látott sor nem elírás!
Ez a példa a létező legrövidebb Java program. Ahelyett azonban, hogy nekiállnánk átrágni magunkat az egyes kulcsszavak jelentésén, nézzük meg, mit is jelent az osztály és az objektum fogalma az objektumközpontú szemléletben.

Tehén mondja „múúú”

A most következőkben megszólaltatunk néhány állatot. Már nagyon régóta tudjuk, hogy a tehén azt mondja „múúú”, a ló azt mondja „nyihaha”, a kutya pedig azt mondja „vauvau”. Képzeljük el, hogy programot kell írnunk, amiben ezek az állatok beszélnek. Egy objektumokat nem ismerő programozási nyelvben, például C-ben ez valahogy így nézne ki:

```
#define TEHEN 1
#define LO 2
#define KUTYA 3

void beszelj(int allat) {
    if (allat == TEHEN) {
        printf(“múúú”);
    } else if (allat == LO) {
        printf(“nyihaha”);
    } else if (allat == KUTYA) {
        printf(“vauvau”);
    }
}
```

Másképp ezt nem is igazán lehetne megoldani C-ben.
A `beszelj(int)` függvénynek az állat azonosítóját átadva a képernyőn megjelenik az állat hangja. Akkor mi a probléma ezzel? Röviden szólva az, hogy itt nem az állatok beszélnek, hanem mi utánozzuk őket.
Gondoljuk meg, mi lenne, ha az állatok beszéde nem csak egy egysoros kiíratás lenne, hanem valami bonyolultabb. A feltételes elágazások hasa megnőne, esetleg még rosszabb, létre kellene hoznunk három, szinte ugyanolyan segítő függvényt. Vagy tegyük fel, hogy nem csak a beszédüket kell kiíratnunk a képernyőre, hanem azt is, hogy mit esznek. Ez egy külön függvényt igényel, amiben pontosan ugyanezt az elágazást kell megvalósítanunk, csak más kiíratással. Felmerül a kérdés: miért kódoljuk le kétszer ugyanazt? Mellesleg mit keres a tehén kódja a lóé mellett?
Az egyik nagy ötlete az objektumközpontú filozófiának az egységbezárás. Ez az adat, és a rajta végezhető művelet egységét jelenti. Jelen esetben az adatot az állatok hangjai, a műveletet pedig a kiíratás jelenti. Próbáljuk meg általánosan megfogalmazni a kérdést. A feladatban állatok szerepelnek. Minden állatnak van egy hangja, amit felfoghatunk úgy, mint az adott állat egy tulajdonsága. Minden állat megkérhető arra, hogy szóljon meg.
Ez az általánosítás noha egyszerű szöveges leírása a problémának, rögtön megadja a megoldás módszerét. Létrehozzuk az állatok közös leírását, amiben megadjuk a tulajdonságuk típusát, és a rajtuk végezhető műveletet. Ezt a leírást hívják osztálynak. Majd minden állathoz létrehozunk egyegy példányt, melyek már egyediek lesznek abban az értelemben, hogy különböznek tulajdonságaikban. Ezeket objektumoknak hívjuk. Hozzunk létre egy ***Allat.java*** állományt az alábbi tartalommal:

```
/**
* Ez az osztaly egy allatot ir le.
* Tulajdonsaga a hangja.
```

```
* Meg lehet szolaltatni.
*/
public class Allat {

    /**
    * Az allat hangja.
    */
    private String hang;

    /**
    * Letrehoz egy uj allatot
    * a megadott hanggal.
    */
    public Allat(String h) {
       hang = h;
    }

    /**
    * Kiirja a kepernyore az
    * allat hangjat.
    */
    public void szolaljMeg() {
       System.out.println(hang);
    }

    /**
    * A program belepesi pontja.
    * Letrehoz harom allatot, es
    * megszolaltatja oket.
    */
    public static void main(String[] args) {
       Allat tehen = new Allat("múúú");
       Allat lo = new Allat("nyihaha");
       Allat kutya = new Allat("vauvau");
       tehen.szolaljMeg();
       lo.szolaljMeg();
       kutya.szolaljMeg();
    }
}
```

Ez első ránézésre sokkal hosszabbnak tűnik a C-beli megoldáshoz képest. Valójában nem az, mert a forráskód fele megjegyzés, ami komoly fejtöréstől szabadít meg minket, ha a jövőben szeretnénk újra felhasználni a kódunkat. Továbbá egy nagyon egyszerű programnál még kevéssé érezhetőek az objektumközpontúság áldásos tulajdonságai, ez az alkalmazás méretének növekedtével azonban egyre élesebben előjön.
Miután lefordítottuk, és futtattuk az alkalmazást, próbáljuk meg lépésről lépésre megérteni, mit is csinál. Az első és legszembetűnőbb dolog a sok *-os sor. Java-ban kétféle megjegyzésjel van. A /*, illetve */ jelek között szereplő szöveget a fordító figyelmen kívül hagyja, és ez lehet több soros is. A // jel ezzel szemben csak a sor végéig „hat”, egy sortörés mindenképpen lezárja. Azért formáztuk ilyen különlegesen a megjegyzéseket, mert később így nagyon egyszerű lesz az osztályhoz leírást készíteni. Az ehhez használatos segédprogram a *javadoc*, amivel egy-két részen belül találkozunk.
Az osztály megadása a `class` kulcsszóval történik. Ezt egy név követi, ami meg kell, hogy egyezzen a forrásfájl nevével, a .java kiterjesztést leszámítva. A `class` előtt álló módosító azt határozza meg, hogy az osztály mindenki számára látható (`public`). Az osztály leírása az ezután következő blokkban van, aminek elejét a {, végét a } jelzi. Nem lehet ***Java*** programot írni legalább egy osztály létrehozása nélkül, ezért készítettünk már a ***„Hello világ!”*** példában is egy `Hello` osztályt.
Az osztály tulajdonságait szokás tagváltozóknak is hívni. Ilyen ebben a példában a `String` típusú `hang`. A String egy szövegfüzért jelképez. Láthatóságát személyesre állítottuk (`private`), ami annyit tesz, hogy csak az osztály műveletein keresztül hozzáférhető. Egy külső kód tehát pusztán azáltal, hogy elérhet egy `Allat` típusú objektumot, még nem módosíthatja közvetlenül a `hang` változóját. Megszokott gyakorlat az osztály összes tagváltozóját személyesre állítani, és csak azokhoz biztosítani beállító, lekérdező műveletet, amelyeknél ez megengedhető.
A feladat szöveges megfogalmazásából eredő megoldási módszernél említettem, hogy osztályok példányaival, azaz objektumokkal dolgozunk. Felfoghatjuk úgy is a dolgot, mint a házépítést. Az osztályleírás adja a szerkezeti vázat, az objektumok pedig az emberektől nyüzsgő, kész építményt. A példányosítás ez esetben a ház átadását jelenti.
Az ehhez kapcsolódó művelet az úgynevezett konstruktor, ami a példányosításkor fut le, és legtöbbször értelmes értékekkel tölti fel a tagváltozókat.
A konstruktor mindig az osztály nevével megegyező művelet, vagy tagfüggvény, és szükségképpen nyilvános, különben a példányosítás meghiúsulna. Jelen példában egy paraméterrel is rendelkezik, ami `String` típusú, és ezt adja értékül `hang` nevű tagváltozójának. A tagfüggvény nagyon hasonlít például egy C-beli függvényhez. Jelen esetben a fontos különbség abban áll, hogy a konstruktor, mint kitüntetett tagfüggvény, `Allat` típusú objektum létrehozásakor hívódik meg.
Létrehozunk egy másik tagfüggvényt, a `szolaljMeg()`-et, amely kiíratja a képernyőre az állat hangját. Ez a ***„Hello világ!”*** példában is látott módon történik. Utolsó tagfüggvényünk a `main(String[])`, ami a program belépési pontját jelenti. Paraméterül a futtatókörnyezettől a parancssori argumentumok tömbjét kapja, amit itt nem használunk fel.
A függvény törzsében létrehozunk három változót, és ezekhez rögtön példányosítunk is egy-egy `Allat`-ot. Majd minden objektumnak meghívjuk a `szolaljMeg()` tagfüggvényét (metódusát).
A figyelmes olvasóban bizonyára felmerült a kérdés, hogy ha egy osztály csak a szerkezeti vázát adja egy háznak, akkor azon közvetlenül nem is lehet műveletet végezni, kizárólag objektumon. Akkor hogyan jelentheti egy tagfüggvény a program belépési pontját, hiszen induláskor `Allat` típusú objektumból egy darab sincs? A válasz a metódus `static` módosítójában rejlik. Az ilyen kulcsszóval ellátott elemek ugyanis közösek az osztályra és anélkül elérhetők, hogy akár példány is létezne az osztályból.
Ennek tudható be az is, hogy képesek voltunk a képernyőre írni! A `System` ugyanis nem más, mint egy osztály. Ennek az out egy olyan tagváltozója, ami statikus. Ráadásul ez egy olyan tagváltozó, ami egy objektum, és ezért van metódusa. A `System.out.println()` elsőre végiggondolva zavarbaejtő, de ez ne rémisszen meg. Ezzel kezdődik az objektumközpontúság, ami a kezdeti nehézségek után hasznos befektetésnek bizonyulhat.

Fülöp Balázs

Bemutatkozik a PEAR
A LEGO-zás művészete PHP nyelven (1. rész)

Hurrá, megérett a körte! Tudom, tudom, nyár eleje van, de mégis. A PEAR (PHP Extension and Application Repository, angolul a betűszó „körtét” jelent) ma már annyira népszerű, hogy nemrégiben belekerült a hivatalos PHP kiadásba is, azaz minden PHP fejlesztő számára alapértelmezetten rendelkezésre áll. Egy olyan közösségi projektről van szó, amelynek célja a nyílt forrású PHP bővítmények, gyakran használt alkalmazás-komponensek összefogása, kezelése és a fejlesztők rendelkezésre bocsátása.

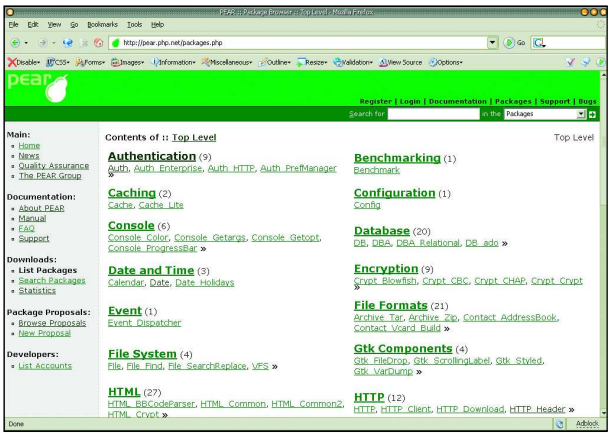
Amikor a rendszert használjuk, akkor egy csomagkezelő segítségével adott problémák megoldását célszó bővítményeket telepítünk a helyi gépre. Az összetevők eredetileg a ***PEAR*** kiszolgálóján egy központi tárban helyezkednek el. Ezeket a bővítményeket aztán a megfelelő ***PHP*** kódba beemelve (`include`) használhatjuk fel.
A ***PEAR*** elsősorban tehát egy olyan szervezett bővítményhalmaz, amelyhez a világ bármely fejlesztője hozzáférhet, szabadon felhasználhatja a saját programjában. Nem csak egy alkalmazás-tárról van azonban szó, hanem egy teljes körű, jól felépített, fejlesztő és felhasználóbarát, kényelmes keretrendszerről, amely nyílt forrású összetevőkön alapul. A ***PEAR*** felhasználók és a ***PEAR*** fejlesztők számára egyaránt nyújt felületet, szolgáltatásokat.
A most induló cikksorozatunkban először áttekintjük, hogyan épül fel a ***PEAR***, hogyan kell telepíteni, használni, milyen lehetőségek állnak rendelkezésünkre a keretrendszer használata közben, valamint szétnézünk a ***PEAR*** honlapján – amely szerves részét képezi a rendszernek. A későbbi epizódok során aztán sorra vesszük, hogy az egyes alkalmazásterületeken milyen megoldásokat kínál számunkra a ***PEAR***, és hogyan tudjuk ezeket használni.

A PEAR létjogosultsága

Mi, fejlesztők számtalan esetben kerülünk szembe jól ismert, gyakori problémákkal, amelyre a legtöbb esetben teljesen ugyanaz a megoldás. Ilyenek például az adatbáziskezelés problémája, levélküldés, matematikai függvények, és még sorolhatnám. Ravaszabb fejlesztőknek ilyenkor már eszébe jut, hogy feltehetően más is összefutott már a problémával, hátha megírta, és elérhetővé tette. Megkezdődik a komponensvadászat ***Google*** barátunk hatásos közreműködésével. A módszer hátránya a „pillanatnyiságában” rejlik: jó lenne a legmegfelelőbb megoldást megtalálni, népszerű összetevőt felhasználni, amelynek van támogatása is, az esetleges hibák kijavítása zajlik, s talán fejlődik is a program. Erről azonban nincs semmilyen biztosíték. Ezen kívül jó lenne egy egységes dokumentáció, és az sem ártana, ha nem kellene minden hasonló esetben fél napokat tölteni a komponensek keresésével. Szerencsénkre ezt a problémát már mások is felismerték, és létrehoztak egy keretrendszert a különböző megoldások szervezett összefogására, amelyet ***PEAR***-nek neveztek el.
Jelenleg 40 kategóriában 405 csomag áll a fejlesztők rendelkezésére, 215 fejlesztő dolgozik a csomagok naprakészen tartásán és továbbfejlesztésén, és a csomagokat együttesen eddig közel 12 milliószor töltötték le.

A PEAR alkalmazás-tárházának felépítése

A ***PEAR*** alapvetően a kiterjesztések és komponensek köré épül, amelyek hierarchiába szervezve, csomagok formájában találhatók meg a tárban. A csomagok hierarchiája valójában egy kategóriafa, ahol az egyes ágakban az azonos területre vonatkozó megoldások kapnak helyet. Egy-egy ilyen csomag magából a meghatározott felépítésű és stílusú programkódból és a hozzá tartozó ***XML***-alapú leíró információból áll. Minden egyes projekt önálló csomagot alkot, és minden projekthez tartoznak fejlesztők, dokumentáció, változatszám, kiadás, bejegyzés a ***PEAR*** weboldalán, stb.
A csomagok az előre meghatározott kódolási stílust, és az egyéb elhelyezési és névkonvenciókat követik egységesen. A komponensek csomagolását a fejlesztők végzik, akik egy új ***PEAR*** projekt regisztrálásával kezdhetik meg munkájuk közzétételét. Az egyes projektek más ***PEAR*** összetevők eredményeit is felhasználhatják, azaz hivatkozhatnak egymásra. Ezáltal egy függőségi viszony alakulhat ki az egyes csomagok között. Mint látható, nem csak felépítésében, de filozófiájában is rendkívül hasonlít a különböző ***Linux*** terjesztések csomagkezelő rendszereinek szerkezetére, még saját csomagkezelő szoftvere is van.



1. ábra A kategorizált csomaglist

A PEAR csomagkezelő

Nem csak a csomagok tárolására és a karbantartására nyújt megoldást a környezet, de lehetővé teszi a csomagok felhasználó- és gépbarát elérését is. A felhasználóbarát böngészés a ***PEAR*** weboldalán keresztül történik, a gépbarát megoldás pedig egy ***XML-RPC (XML alapú távoli eljáráshívás)*** megoldásra épülő, a helyi gépen futó csomagkezelő szoftver formájában testesül meg. Ez utóbbinak az a lényege, hogy az egyes szolgáltatások (a weboldalhoz hasonlóan) a ***PEAR*** kiszolgálóján futnak, s csak az eredmények jönnek át, természetesen a számítógép számára értelmezhető formában, kis túlzással olyan, mintha a felhasználói utasítások hatására a gép böngészné helyettünk a már sokat emlegetett weboldalt. A csomagkezelő egy az egyben a ***DEBIAN***-os ***apt-get*** csomagkezelő alkalmazás „kezelőfelületét” másolja – bár parancssoros alkalmazásról lévén szó, nem igazán beszélhetünk felületről... Ez a bizonyos alkalmazás teszi lehetővé a csomagok böngészését, helyi gépre történő telepítését, függőségek kezelését, majd a későbbiek folyamán azok karbantartását.

A PEAR környezet telepítése

A ***PEAR*** környezet a helyi gépen gyakorlatilag ezt az egy alkalmazást jelenti. A szoftver maga is egy ***PHP*** szkript, amelyet a ***PHP*** parancssoros értelmezője futtat (a héjprogramokkal azonos módon). Feladata a kapcsolattartás, a megadott csomag megkeresése és letöltése a ***PEAR*** tárból, majd annak kitömörítése és az előre meghatározott helyen történő elhelyezése. Ezt a bizonyos könyvtárat, ahová a helyben telepített alkalmazások kerülnek, el kell helyezni a ***PHP*** `include` elérési útvonalában, ami után a fejlesztőnek már csak annyi a dolga, hogy egy `include()` művelettel felhasználja a bővítményt.
A működéshez hasonlóan a telepítés is rendkívül egyszerű. A ***PHP 4.3.0*** változatától kezdve ez a bizonyos „manager” (amit mi csomagkezelőnek nevezünk) alapértelmezetten része a ***PHP***-nak, ha tehát van ***PHP***-nk, akkor van csomagkezelő is, amely a `pear` paranccsal indítható. Egy parancssoros programról van tehát szó, amely a debianos ***apt-get PHP***-s megfelelője. Ebben az esetben nincs semmi dolgunk.
Ha netán mi olyan ***PHP***-t használunk, amelyben ez nem szerepel (például a terjesztésben külön van választva), akkor telepítenünk kell a ***php4-pear*** vagy ***php5-pear*** nevű csomagot (a legtöbb rendszerben így hívják). Van kézi lehetőség is, de erre csak igen ritkán van szükségünk ***Linux*** alatt. Aki erről szeretne tájékozódni, az a ***http://pear.php.net/manual/hu/installation.getting.php*** címen olvashat róla.
Győződjünk meg róla, hogy a ***/etc/php[4-5]/php.ini*** fájlban az `include_path` változóban szerepel-e a ***PEAR*** helyi tár (általában a ***/usr/share/php*** könyvtár) elérési útja, ha nem, akkor adjuk hozzá, de közben vigyázni kell, hogy a mindenkori helyi könyvtár (.) is szerepeljen. Ez általában akkor fontos, ha még nem volt beállítva a változónak semmilyen érték, ekkor ugyanis az aktuális könyvtár az érvényes, de ha adunk meg értéket, akkor az alapértelmezett érték már nem érvényes.

Ismerkedés a csomagkezelővel

Aki nem ismeri az `apt-get`-et, annak álljon itt némi ízelítő, hogy hogyan kell használni. Az alapvető szerkezet:

```
pear művelet <csomagnév>
```

1. táblázat

Telepítés: `pear install <csomagnév>`	Megkeresi az adott nevű csomagot, letölti, kitömöríti a megfelelő helyre (függőségek esetén az összes többi szükséges csomaggal ugyanezt teszi)
Eltávolítás: `pear uninstall <csomagnév>`	Eltávolítja a megadott csomagot
Csomagok listája: `pear list-all`	Kilistázza az összes elérhető csomagot.
Telepített csomag összetevői: `pear list <csomagnév>`	Kilistázza egy már telepített csomag összes alkotórészét a teljes elérési útvonal feltüntetésével.
Információ egy csomagról: `pear info <csomagnév>`	Kilistázza az adott csomaghoz tartozó.
Keresés a csomagok között: `pear search <keresőszó>`	Visszaadja az összes olyan csomag nevét, amelyekre illeszkedik a keresőfeltétel (a nevében szerepel a keresőszó). Ha több szót is megadunk, azok között vagy kapcsolat áll fenn.
Csomag letöltése: `pear download <csomagnév>`	Letölti a csomagot (tar.gz formátumban) az aktuális könyvtárba, de nem telepíti azt.

Az első paraméter mondja meg, hogy mit szeretnénk, a második, hogy melyik csomaggal akarjuk ezt művelni (ha a művelet csomaghoz köthető). Ezek természetesen különböző kapcsolókkal tovább bővíthetők. A kapcsolók egy része a csomagkezelőnek szól, másik részével a műveletet pontosíthatjuk, de ezekre az egyszerűbb esetekben (ez a mostani helyzet is ilyen) nincs szükség. Nézzünk néhány egyszerűbb műveletet, amire szükség van (*1. táblázat*). Ezen kívül számtalan egyéb szolgáltatás áll még a rendelkezésünkre, amelynek nagy részét nem is fogjuk használni, ha nem teszünk közzé csomagokat, vagy nem kezelünk már meglévőeket.

Az olyan műveletek esetén, ahol az összes csomagra vonatkozó adatokról van szó (keresés, összes csomag megjelenítése, stb.) a teljes leíróadatbázis letöltődik a gépünkre. Az első keresés tehát a hálózat sebességétől függően elég lassú, de az utána következők már gyorsak, mert a csomagkezelő gyorstárazza az adatokat. Ez egyébként minden egyes csomagra igaz. Ha kérünk információt egy csomagról, az arra vonatkozó leíróinformáció tárolódik a helyi gépen. Ha közben megváltozott, akkor természetesen frissül on-line. Ezt az átmeneti tárat egyébként a pear clear-cache paranccsal lehet törölni, ez esetben egyébként a már letöltött csomagok (amelyek szintén megtalálhatók egy átmeneti tárban) is törlődnek.

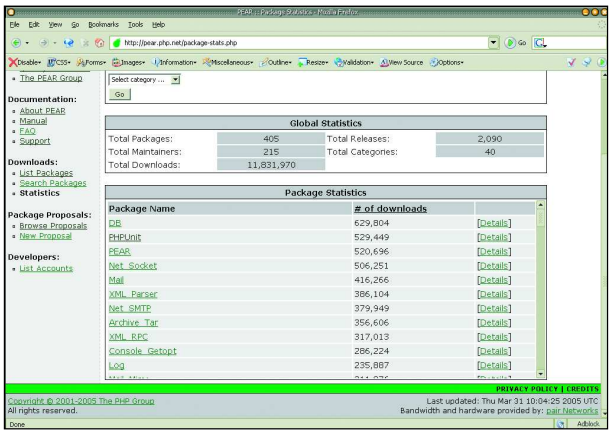
Többször belefutottam már abba a problémába, hogy az ilyen, teljes tárat érintő műveletek során hiba történt, hibaüzenetként pedig azt kaptam, hogy nincs elég memória. Ennek az az oka, hogy alapértelmezetten a *PHP* 8 megabájt memória lefoglalását engedélyezi, egy ilyen *XML* leíró struktúra viszont ennél többet kíván. A hiba úgy orvosolható, hogy a *PHP* parancssori felületének beállításaiban (*PHP5* esetében */etc/php5/cli/php.ini*) a `memory_limit` nevű változó segítségével megváltoztatjuk a memóriakorlátot. Hasonló problémával kerülünk szembe, ha túl sok a hálózati késleltetés, ezáltal a kód futása meghaladja a 30 másodpercet, ennyi ugyanis az alapértelmezett időkorlát *PHP* szkriptek futtatásakor. Bár ennek is és az előző memóriakorlát megemelésének is vannak biztonsági kockázatai, néhány esetben nem ússzuk meg, hogy ne változtassunk az értékeken.

A PEAR használata

Használat gyanánt (a környezet beállítása és a telepítés után) csak annyi a dolgunk, hogy a *PHP* szkriptben behúzzuk az adott fájlt. Lássunk egy példát erre is, Telepítsük, majd használjuk a *PEAR Date* csomagját, amelynek segítségével egy dátummal végezhetünk mindenféle varázslatos műveletet. Egy dátumot a `Date` osztály egy példánya reprezentál, és annak egyes tagfüggvényeivel vezérelhető meglehetősen egyszerűen.

Az első lépés: Adjuk ki rendszergazdaként a `pear install date` parancsot. Helyes működés esetén az alábbi kimenet keletkezik:

```
root@teve:/etc/php5/cli# pear install date
downloading Date-1.4.3.tgz ...
Starting to download Date-1.4.3.tgz (42,048 bytes)
............done: 42,048 bytes
install ok: Date 1.4.3
```



2. ábra A PEAR statisztika oldala

A második lépés: Írjuk meg az alkalmazást, amelynek kulcsmomentuma, hogy az első sorban szerepeltessük az `include('Date.php')` műveletet. A kód egy egyszerű példázatot szemléltetve így néz ki:

```
<?php

  require_once("Date.php");

  $date = new Date("1990-12-30");

  echo "A dátum évszáma: ";
  echo $date->getYear()."<br>";

  echo "Az évnek eme sorszámú napjára esik: ";
  echo $date->getJulianDate().".<br>";

  echo "Az évnek eme sorszámú hete: ";
  echo $date->getWeekOfYear().".<br>";

?>
```

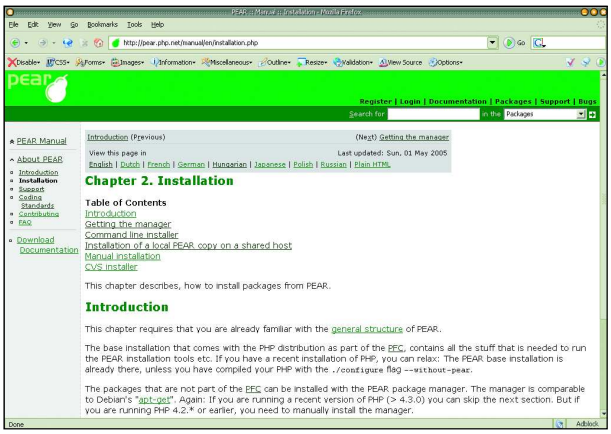
A PEAR csomagok leírása

Fejből nehéz volna kitalálni, hogy a fenti `Date` osztálynak milyen tagfüggvényei vannak, melyik mit csinál, és hogyan kell azokat használni. Ennek érdekében minden *PEAR* projektnek része egy *API* dokumentáció, és számos esetben tartozik hozzá végfelhasználói leírás is, amelyeket a *PEAR* honlapján, az adott projekt oldaláról lehet elérni. A fent tárgyalt esethez a dokumentáció a ***http://pear.php.net/package/Date/docs/1.4.3/*** címen érhető el.

A PEAR honlapja

Ha már szó volt róla: a honlap ugyanolyan szerves részét képezi a *PEAR* környezetnek, mint maguk a csomagok. Minden egyes csomaghoz felhasználóbarát, részletes információt találunk a ***http://pear.php.net*** címen. Az oldalon egyébként minden művelet elvégezhető, amelyet a parancssoros csomagkezelővel már megtanultunk, az csupán arra jó, hogy automatizált, és gyorsabban telepít, mint mi kézzel. A tájékozódás, csomagkeresés, részletes információk olvasása, stb. általában a webes felületen történik.

Minden projektnek van saját aloldala a *PEAR* honlapján.



3. ábra A PEAR felhasználói (fejlesztői) kézikönyv

Egységes kinézet és egységes tartalom jellemzi ezeket. Az egyes projektek legegyszerűbben a ***http://pear.php.net/packages.php*** címen érhetők el, ahol kategorizáltan böngészhetünk az egyes csomagok között. A csomag nevére kattintva juthatunk erre a bizonyos projekt oldalra, ahonnan akár le is tölthetjük a csomagot, és kézzel, úgynevezett félautomata módon is telepíthetjük. Ennek módja:

```
pear install <csomagnev>.tar.gz
```

A módszer előnye, hogy olyan helyen is telepíthetjük, ahol mondjuk nincs közvetlen internet-hozzáférés (ilyen lehet egy telep egyik csomópontja, és még sorolhatnám. Hátránya, hogy így nem kezeli a függőségeket a parancssori telepítő, azaz csak akkor megy fel gond nélkül a csomag, ha minden szükséges másik csomag is rendelkezésre áll.
Egyik hátránya az oldalnak, hogy a keresés során csak a csomag nevében kereshetünk (hasonlóan a parancssoros csomagkezelőhöz), a projektekhez tartozó hosszabb és rövidebb leírásban nem. Szerencsére egyelőre nincs olyan sok csomag, hogy ne lehetne átlátni azokat. Ez részben a kategorizált nézetnek köszönhető. A csomagok ugyanis a felhasználás illetve a technológiák szerint csoportosulnak, mi pedig általában tudjuk, hogy az adott probléma mely területhez kapcsolódik, és azonnal ott keressük.
Nem szabad megfeledkeznünk a letöltési statisztikákról. A ***PEAR*** ugyanis ezekkel méri az egyes csomagok népszerűségét. A ***http://pear.php.net/package-stats.php*** címen elérhető oldalon a csomagok aszerint vannak rendezve, hogy melyiket hányszor töltötték le. Ez persze nem pontos, de közelítőleg jó. Minél népszerűbb egy projekt, annál sokoldalúbb, és nagyobb a valószínűsége annak, hogy folyamatosan fejlődik és mindig karban van tartva. Adott esetben előfordulhat, hogy választanunk kell ***PEAR***-es és nem ***PEAR***-es megoldás között. A választást nagyban megkönnyíti, ha ismerjük a két csomag népszerűségét.
Mindezeken túl a ***PEAR***-ról minden információt megtalálhatunk az oldalon. Egy ***PEAR***-t gyakran használó programozó számára olyan a webapplikáció, mint a ***php*** oldala: élete jelentős részét tölti el annak használatával.

PEAR egyéb

A ***PEAR*** az bővítmény táron, a fejlesztők és felhasználók támogatásán túl egyéb feladatokat is ellát. A csomaghierarchia felsőbb szintjei önálló részhalmazokat képeznek. Ilyen részhalmaz például a ***PFC (PHP Foundation Classes)***, amelybe a mezei ***PEAR*** csomagoknál sokkal szigorúbb előírásoknak megfelelő csomagok kerülhetnek be. Csak stabil változatok lehetnek a ***PFC*** tagjai, azok közül is olyanok, amelyek zavartalanul képesek együttműködni másokkal azáltal, hogy szabványos alkalmazásfejlesztői felülettel (***API***) rendelkeznek. Mindemellett még teljesen általánosak is, azaz semmilyen környezethez sem kötődnek az indokoltnál jobban.
Egy másik ilyen részhalmaz (amely mára már külön projektté nőtte ki magát) a ***PECL (PHP Extension Community Library)***. Ez olyan ***C*** nyelvű bővítményeket, kiterjesztéseket tartalmaz, amelyek a ***PHP4*** részei. A ***PECL*** létrehozásának egyik motivációja az volt, hogy legyen hová átmozgatni a ***PHP*** részeként kezelt különböző bővítményeket. A legfontosabb különbség a ***PEAR*** és a ***PECL*** között az, hogy a ***PECL*** csomagok többnyire ***C*** nyelvűek, míg a ***PEAR*** csomagok ***PHP***-ben íródtak. A ***PECL*** csomagok tehát alacsony szintű, valódi bővítmények, amelyeket a ***PHP*** használ, és a fejlesztők a ***PHP***-n keresztül érhetik el. Tipikusan ilyen csomagok a különböző adatbázisokat kezelni tudó bővítmények. (***PHP4***-ben hasonló bővítmény végzi a ***MySQL***, ***PostgreSQL*** adatbázis kezelését is).
Ez a részhalmaz mára teljesen külön projektté nőtte ki magát, de továbbra is a ***PEAR***-től kölcsönzi szerkezetét.
A gyakorlatban ez annyit tesz, hogy a már ismertetett csomagkezelőn keresztül érhető el, a felhasználók számára áttetsző, hogy ***PEAR***, vagy ***PECL*** csomagot használnak.
A ***PECL*** csomagok telepítéséhez általában szükség van a PHP fejlesztői csomagjára, valamint komplett ***C*** fejlesztői környezetre (például ***C*** fordító, `make`), mivel a telepítendő alkalmazások a telepítés során helyben fordulnak le.
A projekt a ***http://pecl.php.net*** címen érhető el.
Mindezek mellett számos levelezőlista áll a fejlesztő közösség rendelkezésére, ahol tanácsokat kérhetnek a ***PEAR***-rel, csomagok használatával és minden egyébbel kapcsolatban.

Zárszó

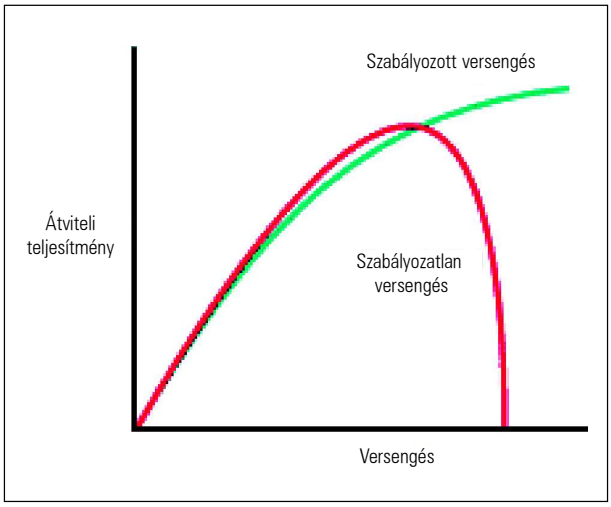
Hatalmas előnyhöz juthat azzal a fejlesztő, ha a felmerülő problémákat okosan, időkímélő módon, kevés fáradsággal, mások kárán tanulva tudja megoldani. Még nagyobb az előnye abban az esetben, ha egy ilyen keretrendszer áll rendelkezésére ahhoz, hogy meg is találja a neki megfelelő komponenst. Továbbmenve az összetevő felhasználása is szervezettséget sugall: egy átgondolt, egységes rendszerben dolgozhat a tervező, fejlesztő. Ez a rendszer lehetővé teszi a folyamatos karbantartást, az egyszerű használatot, és az is rengeteget számít, hogy a csomagok mögött valóban áll egy szervezet, némi garanciáját nyújtva annak, hogy ez hosszú távon így is marad.
A sorozat következő részében a ***PEAR*** alapú adatbázis-kezelő megoldásokkal ismerkedünk meg, élén a legnépszerűbb csomaggal, a ***DB***-vel.

Komáromi Zoltán
(komi@kiskapu.hu)
25 éves, a BME hallgatója, mellette PHP-programozóként dolgozik. Kedvenc területe a multimédia.

Számítógép-hálózatok (18. rész)
Torlódásvédelem

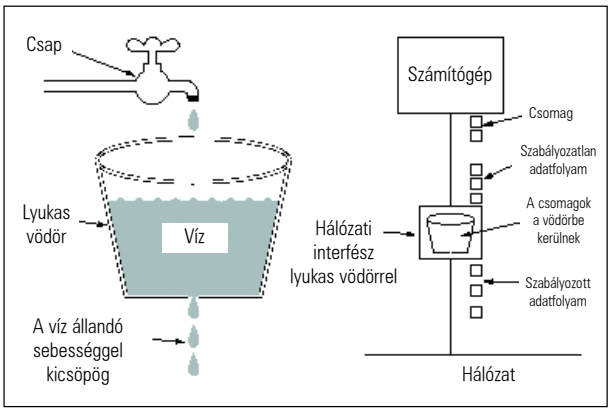
A torlódások elleni védekezés nélkül előbb-utóbb egyetlen csomagot sem tudunk átvinni a hálózaton. Éppen ezért a torlódásvédelem a hálózati réteg egyik legfontosabb feladata, és egyben a sorozat jelen részének témája.

Az előző részből megtudhattuk, hogy a hálózatunkra leselkedő legfélelmetesebb veszély a torlódás kialakulása. Torlódást sok minden előidézhet, és ráadásul saját magát gerjeszti, azaz ha egyszer valahol kialakul, akkor rövid úton átterjed az alhálózat más pontjaira is. Az *1. ábrán* láthatjuk, milyen következményekkel járhat az, ha a torlódások ellen nem lépünk fel kellő eréllyel. Itt a kézbesített csomagokat ábrázoltuk az elküldött csomagok függvényében. Amikor még az alhálózatnak átadott csomagok száma az átviteli kapacitáson belül marad, addig minden rendben zajlik, a csomagok csak átviteli hiba következtében veszhetnek el. (Tehát az elküldött és a kézbesített csomagok egyenes arányban vannak egymással).
Ha viszont egyszerre túl sok csomaggal árasztjuk el szerencsétlen alhálózatunkat, akkor az útválasztók puffer-e előbb-utóbb megtelik, így bizonyos csomagok el fognak veszni. Ez azonban tovább súlyosbítja a helyzetet, ugyanis a gépek az elveszett csomagot ismét megpróbálják elküldeni, ezzel is növelve a már amúgy is leterhelt alhálózat forgalmát. A helyzet egészen odáig fajulhat, hogy egyetlen csomagot sem leszünk képesek átküldeni az alhálózaton. A torlódásvédelem tehát a hálózati réteg egyik legfontosabb feladata.
A torlódásvédelemnek tehát biztosítani kell azt, hogy az alhálózat képes legyen megbirkózni a legkülönfélébb forgalmi helyzetekkel, és gond nélkül továbbíthassa a rajta áthaladó csomagokat. A legelső dolog, amit a torlódásvédelemmel kapcsolatban tudnunk kell az, hogy ez egy globális, az egész hálózatot átszövő kérdés. Amikor megpróbálunk védekezni a torlódások ellen, nem elég, ha például csak az útválasztók csomagküldési mechanizmusán próbálunk valamit állítgatni. Minden olyan tényezőt figyelembe kell vennünk, amely hatással lehet az alhálózaton jelenlévő forgalomra, mint például a gépek viselkedése. Az eredményes védekezés érdekében tehát a hálózat összes alkotóelemének együtt kell működnie. Így egy torlódás kialakulásakor a gépeknek is meg kell tenniük a megfelelő lépéseket, például egy ideig lelassítani, vagy esetleg teljesen felfüggeszteni a csomagok küldését.



1. ábra

Fontos, hogy ne keverjük össze a forgalomszabályozás és a torlódásvédelem fogalmát. Igaz, hogy az előbbi esetben is szükség van arra, hogy a gépek időnként képességeikhez mérten lassabban adjanak, a két fogalom mégis merőben más. A forgalomszabályozás csak az adóval és a vevővel, és a kettőjük között kiépült kétpontos kapcsolatban kialakuló torlódásokkal foglalkozik.
Forgalomszabályozásra például akkor lehet szükség, amikor egy processzorokkal jól megpakolt szuperszámítógép küld adatokat a kis személyi számítógépünknek mondjuk egy 1 Gb/s-os sávszélességű vonalon. A szuperszámítógépnek nem okoz gondot ekkorra sebességgel adni, a kisebb kapacitású gépünk képességeit ez a sebesség azonban meghaladja. A gyorsabb gépnek tehát vissza kell vennie a sebességből, pedig magán a hálózaton nem lépett fel torlódás.
Ezzel ellentétben a torlódásvédelem a hálózaton megjelenő torlódásokkal foglalkozik. Például amikor egyik LAN-ról a másik LAN-ra egyszerre sok állományt szeretnénk átküldeni. A LAN-ban lévő gépek ezzel a feladattal simán



2. ábra

megbirkóznának, az alhálózat azonban beledöglene, hiszen az átküldendő csomagok száma meghaladja az áteresztőképességét. A baj elkerüléséhez a gépeket rá kell bírni arra, hogy ne egyszerre árasszák el az alhálózatot, az útválasztóknak pedig a forgalmat szét kell osztaniuk több útvonal között.

A torlódásvédelmi algoritmusok durván két nagy csoportra oszthatók: a nyílt hurkúakra ***(open loop)*** és a zárt hurkúakra ***(closed loop)***. Az előbbi osztályba tartozó módszerek úgy próbálják a torlódásokat elkerülni, hogy eleve nem hagyják azokat kialakulni. Ehhez különböző irányelveket határoznak meg, amelyek alapján az útválasztók és a gépek döntéseiket meghozzák. Fontos, hogy ezek kőbevésett irányelvek, amik örökérvényűek, azaz nem változnak a hálózat működése közben. Ez azt is jelenti, hogy a gépek és az útválasztók döntéseikben nem veszik figyelembe azt, hogy éppen mi is történik a hálózaton (azaz mi a hálózat aktuális forgalmi állapota).

Ezzel gyökeresen ellentétes felfogást képviselnek a zárt hurkú algoritmusok, amelyek csak akkor avatkoznak be, amikor már valahol kialakult a torlódás. Ilyenkor gyors információgyűjtésbe kezdenek, amelyből megállapítják, milyen lépéseket kell tenni a torlódás megszüntetésének érdekében. Ezután felszólítják a megfelelő útválasztókat, illetve gépeket, hogy tegyék meg a megfelelő intézkedéseket.

A következőkben mindkét típusú algoritmusra mutatunk példákat, és megvizsgáljuk, miként valósítják meg a gyakorlatban a fent leírtakat.

Nyílt hurkú torlódásvédelmi algoritmusok

A nyílt hurkú eljárások tehát arra törekednek, hogy esély se legyen a torlódás kialakulására, ehhez pedig meg kell szüntetni a torlódás legfőbb okát, a lökésszerű forgalmat. A gépek általában még véletlenül sem adnak egyenletesen, hanem mindig a legváratlanabb pillanatban halmozzák el az alhálózatot csomagokkal. Ha a gépeket rábírhatnánk, hogy egyenletesebben küldjék csomagjaikat, akkor kisebb lenne az esély a torlódás kialakulására.

Ezzel el is érkeztünk a ***forgalomformálás (traffic shaping)*** fogalmához, amely az alhálózaton folyó adatátvitel ***átlagos*** sebességének szabályozását jelenti. Most megnézünk két olyan algoritmust, amelynek segítségével ezt a sebességet kordában tarthatjuk.

Ezek közül az első a ***lyukas vödör (leaky bucket)*** algoritmus, amelynek nagyon találó neve van, hiszen tényleg úgy működik, mint egy valódi vödör, amelynek az alján lyuk tátong *(2. ábra)*. Amikor egy ilyen vödörbe vizet eresztünk, az alján a víz állandó sebességgel fog szivárogni, mégpedig úgy, hogy ez a sebesség nem függ attól, milyen gyorsan eresztjük a vizet a vödörbe.

Most képzeljük el ugyanezt, csak víz helyet csomagokat lapátolunk a lyukas vödrünkbe, amit a gépek egy sorként valósítanak meg. A sorba folyamatosan érkezhetnek a csomagok, egészen addig, amíg meg nem telik.

Ha egy csomag nem fér be a sorba, akkor az eldobásra kerül. Minden óraütéskor egy csomag kikerülhet a sorból, és elküldhető az alhálózatnak. Ezzel a módszerrel tehát megkímélhetjük az alhálózatot a löketszerű forgalomtól.

Mivel a csomagok egyenletesen továbbítódnak, az útválasztóknak több idejük van egy-egy csomag feldolgozására, nehezebben telik meg a pufferük, így csökken a torlódás kialakulásának esélye.

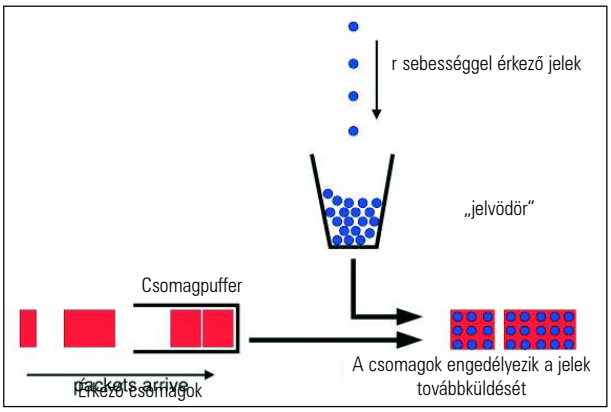
Ez a módszer azonban nem minden esetben tökéletes. Hibája abban rejlik, hogy sziklaszilárdan ragaszkodik az általa kialakított átlagos csomagküldési sebességhez (más szóval kimeneti mintához). Bizonyos esetekben azonban szükség lehet egy kis rugalmasságra. Néhány alkalmazás ugyanis úgy működik, hogy sokáig számolgat ***(vagy épp semmit sem csinál)***, majd hirtelen egy nagyobb löketet indít útnak, majd jó ideig megint nem küld semmit.

Ha a gép sokáig csendben volt, akkor nem nagy baj, ha egy kicsit nagyobb kimeneti sebességgel kezd adni, majd utána fokozatosan visszavesz belőle. Ha ilyenkor is ragaszkodunk a lyukas vödör merev kimeneti mintájához, akkor a csomagok lassabban jutnak célba. Probléma merül fel akkor is, ha az alkalmazás több csomagot akar küldeni, mint amennyi a vödörbe belefér, hiszen ilyenkor az utolsó csomagok elvesznek.

Ezekre a problémákra kínál megoldást a ***vezérjeles vödör (token bucket)*** algoritmus *(3. ábra)*. Itt a vödörben nem csomagokat, hanem úgynevezett vezérjeleket tárolunk. A vezérjelek óraütésenként, folyamatosan keletkeznek. Amikor a vödör megtelik vezérjelekkel, akkor az újonnan születő vezérjelek megsemmisülnek. A gépet egy csomag csak akkor hagyhatja el, ha kivesz egy vezérjelet a vödörből. Ha a vödör kiürült, akkor addig kell várnia, amíg új vezérjel nem keletkezik.

Ez az algoritmus két jelentős dologban is eltér a lyukas vödörtől. Először is itt a gépek némileg nagyobb önállósággal bírnak, hiszen spórolhatnak a verzérjelekkel. Azok a gépek, akik egy ideig nem adnak, félretehetik vezérjeleiket egy nagyobb löket számára. Persze ez a löket nem lehet nagyobb, mint a vödör mérete. Másik fontos különbség, hogy a vödör megtelése után itt csak vezérjeleket dobunk el, csomagokat azonban soha.

Érdemes megjegyeznünk, hogy e két algoritmust nem csak a gépek kimeneti mintáinak szabályozására használják, hanem például két útválasztó közötti forgalomkisimításra is. Ilyenkor azonban kell némi változtatás. Gépek esetében a vezérjeles vödör használatakor nem jelent túl nagy problémát, ha nem engedünk ki újabb csomagot addig, amíg új vezérjel nem keletkezik. Útválasztók eseté-



3. ábra

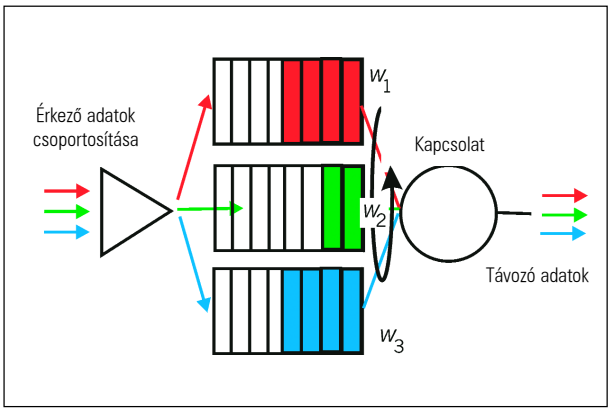
ben a csap nem mindig zárható el. Tegyük fel például, hogy a vezérjelek elfogynak, de továbbra is érkeznek befelé jövő csomagok. Ha ilyenkor korlátozzuk a kimenetet, akkor fennáll a veszély, hogy a puffer megtelik, és egyes csomagok elvesznek.

Zárt hurkú torlódásvédelem

A torlódásvédelmi algoritmusok másik nagy csoportja nem a torlódásokat dinamikusan próbálja szabályozni, azaz folyamatosan felügyeli a hálózat aktuális forgalmi állapotát, és ahol és amikor szükségét látja, beavatkozik.
Felmerülhet a kérdés, hogy a hálózaton kialakuló torlódások nagyságát milyen mennyiséggel tudjuk jellemezni. Erre sok választási lehetőség adódik. Figyelhetjük például az útválasztók átlagos sorhosszait (egy útválasztó pufferjében átlagosan hány darab csomag vár továbbításra), az újraküldött csomagok számát, vagy éppen az útválasztók puffertúlcsordulásából adódó csomagvesztések arányát. Bármelyik paramétert figyelje is ezek közül egy algoritmus, annyiban biztosak lehetünk, hogy minél magasabb értéket mér, annál súlyosabb a helyzet a hálózaton.
Amikor egy útválasztó torlódást észlel, akkor azt azonnal jelezni kell társainak, mivel csak együttes fellépéssel fékezhetik meg a kialakult forgalmi dugót. Erre is sok módszer létezik. Ezek közül a legkézenfekvőbb az, ha speciális csomagokat küldünk szét, amelyből a többi útválasztó értesül a kialakult helyzetről. Ezzel azonban az a baj, hogy az amúgy is már terhelt hálózatot tovább lassítanánk. Így sokkal praktikusabb megoldásnak ígérkezik az, ha minden csomag fejlécében fenntartunk egy bitet, és annak segítségével figyelmezteti a többieket. Ha a terhelés meghalad egy bizonyos küszöböt, akkor minden kimenő csomagnál az útválasztó 1-esre állítja ezt a bitet, ezzel jelezve a többieknek, hogy vegyenek vissza az adási sebességből.
Másik megoldás lehet a forgalom megosztása. Ilyenkor az útválasztók folyamatosan mérik a vonalak terheltségét, és úgy irányítják a csomagokat, hogy azok lehetőleg teljesen elkerüljék az épp torlódás alatt álló részt.
Most nézzünk meg egy-két konkrét algoritmust! Virtuális áramkör alapú alhálózatok esetében (ilyen például a telefonhálózat) a legegyszerűbb megoldás a ***belépéses ellenőrzés (admission control)***. A módszer azon az egyszerű feltételezésen alapul, hogyha valahol torlódás alakult ki, akkor a helyzeten valószínűleg nem fog segíteni az, ha engedünk további virtuális áramkörök ***(kapcsolatok)*** kiépítését. A vonalas telefonhálózatoknál is hasonló a helyzet: a tárcsahangot egészen addig nem kapjuk meg, amíg a hálózat fel nem szabadul a terhelés alól.

Lefojtó csomagok

Ezen az eljáráson sok torlódásvédelmi algoritmus alapul, és alkalmazható mind datagram, mind virtuális áramkör alapú alhálózatok esetében is.
Az alapötlet itt is az, hogy az útválasztók folyamatosan figyelik a kimeneti vonalaikat, és mindegyikhez, terhelésüktől függően hozzárendelnek egy 0 és 1 közé eső valós számot. (Egyes algoritmusok az útválasztó más erőforrásait kísérik figyelemmel, például a puffer méretét). Ha ez a szám meghalad valamiféle küszöbértéket, akkor az útválasztó minden befelé menő csomag forrásához visszaküld egy úgynevezett ***lefojtó csomagot (choke packet)***. Amikor a gép egy ilyen csomagot kap, akkor azonnal visszavesz az adási sebességéből (például a lyukas vödör algoritmus segítségével). A lefojtó csomagok ugyan „meg vannak jelölve", s így az útválasztók nem generálnak továbbiakat, de mivel egy torlódás általában több útválasztót is érint, a gép egyszerre több lefojtó csomagot is fog kapni. Ilyenkor a sebesség visszavétele után egy ideig nem fog reagálni az lefojtó csomagokra.
Ezután a gép várakozik és figyel. Ha továbbra is érkeznek lefojtó csomagok, akkor ez azt jelenti, hogy a torlódás még mindig fennáll, így az adási sebességet tovább csökkenti. Ha azonban bizonyos idő elteltével nem érkezik újabb elfojtó csomag, akkor a torlódás valószínűleg megszűnt, így nem kell már korlátozni a forgalmat.
Nem egyszerű kérdés, hogy a gép mennyi idő múlva adja fel a sebességkorlátozást. Ha túl hamar teszi, akkor nagy rá az esély, hogy újabb torlódást idéz elő, ha pedig túl lassan, akkor meg nem működik gazdaságosan. A megoldás minden bizonnyal a két véglet között lesz, de hogy pontosan hol, arra csak az adott algoritmus pontos elemzése adhat választ. Hasonló megfontolásokat kíván annak megállapítása is, hogy az algoritmus a lefojtó csomagok hatására milyen ütemben csökkentse sebességét, illetve a torlódás megszűnésével miként növelje azt. A jól bevált módszer az, ha torlódás esetén a gépek megfelezik adási sebességüket, viszont utána csak kisebb lépésekben növelik azt.
Ennek az algoritmusnak van viszont egy olyan hátránya, hogy feltétel nélkül hisz a gépek becsületességében. A hálózatok világa azonban igazságtalan. Ha egy gép kötelességtudóan lecsökkenti adási sebességét, de a többiek ezt nem teszik meg, akkor a becsületes gép jár a legrosszabbul, mivel neki drasztikusan le fog csökkenni a sávszélessége.
Egy igazságosabb világ ígéretét rejti magában a ***pártatlan sorbaállítás (fair queueing)*** algoritmus. Itt az útválasztók minden kimeneti vonalhoz pontosan annyi várakozási sort rendelnek, ahány bemenetük van. A várakozási sorokon egyenként, körkörösen végigmegyünk, és kiveszünk egy csomagot, amit ráteszünk a kimenő vonalra. Ennek köszönhetően az egymással versengő gépek egyenlő mennyiségű csomagot tudnak célba juttatni.



4. ábra

A dzsitter szabályozás

Minden felhasználó azt szeretné, ha a neki szánt csomagok a lehető leggyosabban elérnének hozzá, azaz a letöltési sebessége minél nagyobb legyen. A felhasználók tehát rendkívül rosszul tűrik, ha a csomagjaik késnek, például azért, mert egyes útválasztók feltartóztatják azokat.

Egyes alkalmazások azonban nem a csomagkésleltetésre, hanem inkább az úgynevezett dzsittere érzékenyek, azaz a csomagok (a forrástól a célig történő) átviteli idők durva változásaira. Például a hang, illetve mozgókép továbbításkor simán belefér, ha minden csomag csúszik pár századmásodpercet, viszont akkor előbb se jöjjön senki, mindenki ugyanannyit késsen.

Ehhez először meg kell becsülni a csomagok várható átviteli idejét (beleszámolva az átlagos torlódást is), ugrásokra lebontva. Így minden útválasztó tudni fogja, hogy optimális esetben mekkora időközönként kell egyegy csomagot továbbítania. Amikor egy ilyen csomag érkezik az útválasztóhoz, megnézi, hogy nincs-e késésben, illetve nem jött-e előbb a kelleténél. Az utóbbi esetben nincs már dolga, mint hogy kivárja a megfelelő időt, majd útjára bocsátja. Ha siet, akkor nem tehet mást, minthogy elsőbbséget ad neki az összes többi csomaggal szemben, és reménykedik, hogy így még be tudja hozni a késését, és a dzsitter nagysága nem nő meg számottevő mértékben.

Terhelés eltávolítása (load schedding)

Amikor már nagyon veszélyes a helyzet, és a csomagok az útválasztók fejére nőttek, akkor jön a végső megoldás: az útválasztók se szó, se beszéd, elkezdik kidobálni a csomagokat. Na de milyen elv alapján döntsük el, mely csomagok kerüljenek végérvényesen a süllyesztőbe? Nem hangzik túl jó ötletnek, ha véletlen szerűen, például minden második csomagnak kegyelmeznénk meg, a többit kérdés nélkül kidobnánk.

Sokkal jobb lenne, ha ezt a csomagot küldő alkalmazástól tennénk függővé, ugyanis mindegyiknek más és más csomag a legfontosabb. Mit is értünk ezalatt? Egy fájlküldő programnak, például egy FTP kliensnek inkább a korábban jött csomagok a fontosak, mert ha például egy vagy több csomag kimarad, akkor elképzelhető, hogy a forrásnak a sikeresen megkapott csomagokat is újra meg kell ismételnie. A ***RealPlayer*** azonban, ha választhatna, inkább a legújabb csomagokat kapná meg.

A legjobb megoldás tehát az, ha maga az alkalmazások mondhassák meg, mely csomagok a legfontosabbak számukra. Ehhez lehetőséget kell nekik biztosítani arra, hogy a csomagok fejlécébe jelöljék, az adott csomag mennyire fontos. Magyarul prioritásokat rendelhetünk a csomagokhoz, és ha torlódás van, akkor az útválasztók először a legalacsonyabb prioritási osztályba sorolt csomagoktól válnak meg.

Ahhoz, hogy ez a módszer jól működjön, figyelembe kell vennünk egy nagyon is emberi tényezőt, mégpedig azt, hogy az emberek maguktól valószínűleg nem fognak alacsonyabb prioritású csomagokat küldeni. Ha nincs semmifajta kényszerítő erő, nyílván mindenki a legmagasabb prioritásra fogja állítani a csomagjait. A megoldás erre lehetne például az, hogy a szolgáltatók drágább tarifát számítanak fel a magasabb prioritású csomagokért.

Bizonyos hálózatokban azonban nem kecsegtet akkora haszonnal az, ha egyes csomagokat magasabb prioritással láthatunk el. Az ATM esetében például a csomagok fix méretű cellák, és ezek egy nagyobb egység részei.

Egy üzenet több fix méretű cellaként halad tovább, és ha az útválasztónak egy cellát el kell dobnia, akkor valószínűleg a forrásnak az egész üzenetet újra meg kell ismételnie. (Mindezek ellenére az ATM cellák fejlécében van egy bit, amellyel megkülönböztethetjük a fontosabb csomagokat a többiektől).

A végére még egy megjegyzéssel élnénk. A szimulációs tesztek azt mutatják, hogy az útválasztó, és vele együtt az alhálózat akkor jár a legjobban, ha a torlódás észlelése után minél előbb nekiáll a csomagok pusztításának, mivel így elejét vehetik egy komolyabb forgalmi dugó kialakulásának. Tehát megint egy újabb dilemma. Inkább minél több csomagot próbáljunk célba juttatni, és lefojtó csomagok, illetve egyéb technikákkal próbáljuk csillapítani a kialakuló torlódásokat, vagy a torlódásvédelem minden felett álljon, és a baj legkisebb jelére már pusztítsuk a csomagokat.

Nos, ennyit a torlódásvédelemről. A sorozat következő részei sokkal életszerűbbek lesznek, mostantól ugyanis elvetjük azt a feltételezésünket, hogy minden hálózat egyforma. Eddig vígan küldözgettünk csomagokat hálózatok között, ám fel sem merültek bennünk olyan alapkérdések, mint például miként tudnak együttműködni eltérő protokollokat használó hálózatok.

A hálózatok persze nem csak protokolljaikban különböznek egymástól, hanem még vagy száz másik dologban. Ezek között bizony átjárást kell biztosítani, amelyekről az ***átjáró (gateway)*** nevű eszközök fognak gondoskodni. Hogy pontosan miképp, az az elkövetkezendő pár rész témája lesz.

Garzó András (garzoand@interware.hu)
Körülbelül három éve foglalkozik Linux- és más Unix-rendszerekkel. Legjobban az operációs rendszerek lelkivilága érdekli, de nyitott egyéniség. Kedvenc étele a palacsinta, és van egy Richard nevű macskája. Minden észrevételt, megjegyzést, levelet szívesen fogad.

Számolás a uniq segítségével

A parancshéj szakértői jól elboldogulnak a szabványos segédprogramok egyszerű kombinációival is. Ismerjük meg két egyszerű parancs együttes használatának egyik leggyakoribb példáját.

A *UNIX*-szerű operációs rendszerek egyik igazán nagy előnye a parancsok együttes használatának a lehetősége. A parancsok kombinálásával rengeteg feladatot oldhatunk meg, a lehetőségeinknek csak az ötletességünk és képzelőerőnk szabhat határt.
Bár a lehetséges kombinációk száma igen nagy, a tapasztalat azt mutatja, hogy bizonyos párosítások sokkal gyakrabban fordulnak elő, mint a többi. Az egyik általam gyakran alkalmazott párosítás a `sort` és a `uniq` parancsok összetétele, amellyel egy fájlban az egyes karaktersorozatok előfordulásait számolhatom meg. Ez az új ***Linux*** felhasználók számára egy jó fogás és egy olyan tudás, aminek az elsajátítását soha nem fogjuk megbánni.

Egy egyszerű példa

Vizsgáljunk meg először egy egyszerű példát az alapelvek tisztázása céljából. Adott egy `gyumolcs` nevű fájl az alábbi tartalommal:

```
alma
narancs
alma
```

A következő módszerrel eldönthetjük, hogy az egyes szavak hányszor fordulnak elő:

```
% sort gyumolcs | uniq -c
   1 narancs
   2 alma
```

Mi is történik valójában? Először is, a `sort gyumolcs` parancs rendezi a fájlt. Az eredmény rendes körülmények közt az alapértelmezett kimenetre (ebben az esetben a képernyőnkre) kerülne, de most látunk egy | (csővezeték-) karaktert is a parancs után. Ez a csővezeték a `sort gyumolcs` parancs kimenetét a következő parancs, a `uniq -c` bemenetére irányítja, amely minden sort kiírat, a sor elején jelezve az előfordulásuk számát.

Egy gyakorlatiasabb példa

Az egyszerű példa alapján még nem annyira nyilvánvaló, hogy miért is olyan hatékony ez a módszer, bár mindjárt használhatóbbnak tűnik, ha a kérdéses fájl például egy ***Apache*** webkiszolgáló hozzáférési naplója több százezer sorral. A hozzáférési napló tele van értékes információval. A `sort` és a `uniq` parancsok használatával meglepően sok egyszerű adatelemzést hajthatunk végre pillanatok alatt a parancssorból. Képzeljünk el egy munkatársat, akinek szörnyen szüksége van arra a tíz IP-címre, ahonnan a leggyakrabban érkezett kérés a ***foo.php*** nevű parancsfájl futtatására január folyamán. Néhány pillanattal később már kezünkben is van a kért információ. Honnan tudtuk ilyen gyorsan megkapni a választ? Nézzük meg lépésenként a megoldást. A példa kedvéért tegyük fel, hogy a kiszolgálónk a következő formátumban naplózza az eseményeket:

```
192.168.1.100 - - [31/Jan/2004:23:25:54 -0800]
➥"GET /index.php HTTP/1.1" 200 7741
```

A napló több hónap adatait tartalmazza, nem csak a 2004 januári bejegyzéseket, tehát az első teendőnk, hogy a `grep` segítségével csökkentsük az adatmennyiséget:

```
% grep Jan/2004 access.log
```

Ezután a kimenetben megkeressük a ***foo.php*** karakterláncot:

```
% grep Jan/2004 access.log | grep
➥foo.php
```

Amennyiben csak az IP-címek előfordulásait akarjuk megszámlálni, jobb ha a kimenetünket erre az egy mezőre korlátozzuk, vagyis:

```
% grep Jan/2004 access.log | grep foo.php | awk '{
➥print $1 }'
```

Az `awk` parancs ismertetésére itt most nincs lehetőség, de elég annyit tudnunk, hogy az `awk '{ print $1 }'` az első szóközt megelőző karakterláncot írja ki, ami esetünkben éppen az IP-cím.
És végül alkalmazhatjuk a sort és uniq parancsokat. Íme a végső parancslánc:

```
% grep Jan/2004 access.log | grep foo.php | \
awk '{ print $1 }' |  sort -n | uniq -c | \
sort -rn | head
```

A fordított perjel (\) jelzi, hogy a parancs a következő sorban folytatódik. Egyetlen hosszú sorban is begépelhetjük a parancsot perjelek nélkül, vagy a csővezetéket felbontva több sorban is beírhatjuk a képernyőn. Láthatjuk, hogy az első példánktól eltérően az első rendezés (`sort -n`) szám szerinti rendezés, ami helyénvaló, hiszen ebben az esetben számokkal dolgozunk.
A másik különbség a `| sort -rn | head` rész beszúrása. A `sort -rn` parancs a `uniq -c` kimenetét rendezi fordított számsorrendbe. A `head` parancs csak a kimenet első tíz sorát írja ki. Az első tíz sor éppen megfelel erre a feladatra, mivel csak a tíz legnagyobb számban előforduló tételt keressük:

```
43 12.175.0.35
16 216.88.158.142
12 66.77.73.85
 9 66.127.251.42
 7 66.196.72.78
 7 66.196.72.28
 7 66.196.72.10
 7 66.147.154.3
 7 192.168.1.1
 6 66.196.72.64
```

A parancsláncot alkotó parancsok bármelyik összetevőjének megváltoztatásával módosíthatjuk a csővezeték működését. Például ha a legnagyobb tíz helyett történetesen éppen a legkisebb tíz értéket keressük, csak a `head` parancsot kell `tail`-re cserélnünk.

Összegzés

Az adatok `sort` és `uniq` parancsokkal történő csővezetékes feldolgozása nagyon jól használható módszer, remélem a bemutatott példák kedvet csináltak a csővezeték-technika további tanulmányozásához. A használt parancsokról bővebb információt a megfelelő kézikönyvlapokról (`man`) szerezhetünk.

Linux Journal 2005. január, 129. szám

Brian Tanaka
1994 óta UNIX rendszergazda,
olyan cégeknek dolgozott, mint a The Well,
az SGI, az Intuit vagy a RealNetworks.
A btanaka@well.com címen érhető el.

Szintről szintre

Marcel ez alkalommal egy klasszikus műfaj játékaiból szemezget. Gyerünk! Ugrás!

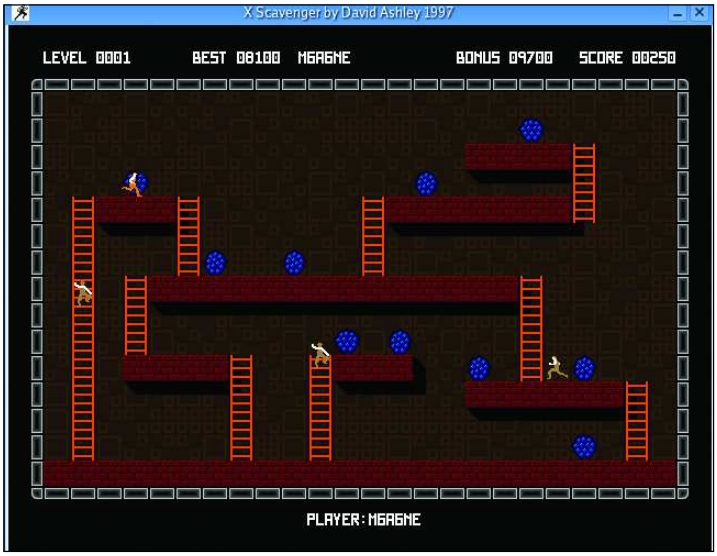
Óvatosan, ***François***, túl gyorsan mész! Le fogsz esni… késő. Jaj, ***mon ami***, újabb életed veszett oda a mélykék tengerben. Sőt, mind a négyet elveszítetted, úgyhogy most én jövök. Tudom, hogy játszunk, de egyben a mai menüt is kóstolgatjuk, ***mon ami***, márpedig a tesztelés a ***Chez Marcel*** mindenre kiterjedő minőségellenőrzésének része. Most látom, hogy hamarosan megérkeznek a vendégek, és még nem választottunk bort.

Hogy már meg is jöttek? Ne haragudjatok, ***mes amis***, egy picit szórakozottak vagyunk. Üdvözöl benneteket a ***Chez Marcel***, ahol mindig finom borok kísérik a nagyszerű linuxos fogásokat. Gyerünk, ***François***, indulj a pincébe! A mai menü könnyedségére tekintettel valamilyen könnyű, önmagát itató borocskára lesz szükségünk. Hozd fel azt a 2001-es ***Modello Italian*** vöröset.

Készséges felszolgálóm és jómagam a gyakran platformjátékoknak emlegetett videójátékok közül ismerkedtünk meg néhánnyal. Az ilyen játékok alapötlete rendkívül egyszerű, legtöbbjük két dimenzióban gördülő háttérre és kisebb-nagyobb küzdelmekre épül. A szintek közötti továbblépéshez tárgyakat kell megkeresni és összegyűjteni, miközben futva, ugorva kell elkerülni különféle akadályokat és ellenségeket. A különféle helyek között a platformok között átmászva vagy átugorva mozoghatunk. A játéktípus hosszú éveken át szinte egyeduralkodó volt, de mind a mai napig népszerű, és folyamatosan készülnek újabb és újabb képviselői. Mindez a linuxos világban is igaz.

Évekkel ezelőtt, mikor még valamivel fiatalabb voltam, egy ***Lode Runner*** nevű játék volt nálam a sláger. ***Dave Ashley*** nagyszerű ***Scavengerje*** hasonló hozzá, és remek szórakozást kínál. (*1. ábra*) Alapötlete egyszerű. Létrákon mászva, kötelekeen lengve, a platformok között szaladva kincseket kell gyűjtenünk, miközben elkerüljük a rosszfiúkat. Attól függően, hogy éppen mi van a lábunk alatt, képesek vagyunk akár lyukakat is ásni, amivel megmenekülhetünk vagy csapdába csalhatjuk ellenségeinket.

Szerezzük be a ***Scavenger*** egy példányát a ***Scavenger*** webhelyről (lásd az internetes forrásokat), ahol megtaláljuk a legújabb forrást. Úgy láttam, hogy sok terjesztés saját oldalán előre fordított bináris fájlokat is lehet találni belőle, így talán jobb, ha saját terjesztésünk csomagjai között kezdjük a keresést. Persze forrásból sem nehéz a ***Scavengert*** lefordítani. Az eljárás a szokásos kibontás-fordítás eljárás egy picit módosított változata:



1. ábra A Scavenger méltó utódja a Lode Runnernek

```
tar -xzvf xscavenger-1.4.4.tgz
cd xscavenger-1.4.4/src
xmkmf
make
su -c “make install”
```

A szükséges parancs a `scavenger`, ezt kiadva már kezdődik is a játék. Először bemutató módban indul, így áttekinthetjük a mozgásokat, műveleteket, valamint ráérezhetünk a program működésére. Az F1 gombbal saját legmagasabb szintünkre léphetünk. Az első alkalomnál ez az 1. szintet jelenti. Az irányítás a numerikus billentyűzet gombjaival történik, a kurzorgombokkal fel, le, jobbra és balra mozoghatunk. A balra és jobbra fúrás további két műveletet jelent. Ha az alapértelmezett billentyűkiosztás nem felel meg az igényeinknek, mert például hordozható gépet használunk, de már megkezdtük a játékot, akkor nyomjuk le az Esc, majd a szóköz billentyűt. Ezzel a beállító menübe jutunk. Innen az F10 gomb lenyomásával módosíthatjuk a billentyűkiosztást.

2. ábra Feladatod, katona, a hadifogságba esett bajtársaid megmentése! Van kérdés?

3. ábra Vajon ki tudja menteni Tux imádott Pennyjét az elvetemült Nolok karmai közül?

Nálam a mozgások maradtak a kurzorgombokon, de a balra és a jobbra fúrást az A és a D gombbal végzem. Miközben a billentyűkiosztás módosításával foglalatoskodunk, néhány további figyelemre méltó beállítást is észrevehetünk. Például a bemutatót többféle szinttel is indíthatjuk. A bemutató mód futása közben az F7 és az F8 gombbal léphetünk szintet lefelé vagy felfelé. A legkedvesebb szolgáltatás számomra azonban a szintszerkesztő. Nyomjuk le az F3 gombot, és máris nekiláthatunk saját ***Scavenger*** pályáink elkészítésének – kiváló módja szabadidőnk elfecsérlésének. Egy szép, barátságos játéknál jobban semmi sem kelt nagyobb étvágyat. Hatalmas kísértést éreztem, hogy az utolsó mondathoz egy smiley-t fűzzek, főként azért, mert a következő játék, amit be szeretnék mutatni, gonosz smiley-król szól. A játék címe ***Blob Wars: Metal Blob Solid***, készítője a ***Parallel Realities***, és az egyik legkülönösebb az általam valaha is megismert játékok közül.

A Blob Warsban a feladatunk hadifoglyok megmentése – picit más formában. Minden karakter egy-egy smiley, más néven ***blob (paca)***, nekünk pedig a ***Bob*** nevű blobkatonát kell irányítanunk, aki különféle barátságtalan, bár sokszor nagyon szép helyeken keresztültörve keresi börtönbe jutott bajtársait. Smileykatonánkkal futva különféle fegyvereket kell használnunk – a lézert ne feledjük el felvenni –, és platformról platformra jutva kell egyre közelebb küzdenünk magunkat a győzelem felé. (*2. ábra*)

Előfordított RPM-eket és deb fájlokat egyaránt elérhetünk a webhelyéről, ahogy a forrást is, ha valamiért magunk akarjuk végezni a fordítást. Néhány SDL fejlesztői könyvtárra is szükségünk lesz, de ettől eltekintve az egész művelet csupán négy lépésből áll:

```
tar -xzvf blobwars-1.02-1.tar.gz
cd blobwars-1.02
make
su -c "make install"
```

A program indításához a `blobwars` parancsot kell kiadni. Miután a játék elindult, egy menü jelenik meg, amelyből új játszmát indíthatunk, illetve folytathatunk egy már megkezdettet. Az ***Options (Beállítások)*** menü segítségével ablakosból teljes képernyős módba válthatunk. A zene és a hanghatások hangerejét külön állíthatjuk, módosíthatjuk a fényerőt, továbbá megadhatjuk, hogy akarunk-e vért látni a játék folyamán. Bizony, amikor az ellenséges blobokra tüzelünk, azok felsikítanak, majd véres tűzcsóvává válnak. Hát igen, egy kicsit túllőttek a célon, de azért jó a játék. De tényleg…

Ezután megjelenik egy térkép, amelyen számos helyszín szerepel, ezeken tartják fogva a katonákat. Minden hely egy-egy küldetés. A mentési akció során számos hasznos tárgyat – például repülőcsomagot, lézerfegyvert, gránátokat vagy kulcsokat – tudunk magunkhoz venni. A tárgyakat sokszor a megsemmisített ellenségektől kell elvennünk. Ha rátalálunk egy fogolyra, elég elsétálnunk fölötte, máris elteleportál a küldetésből. Régóta nagy ***Star Trek*** rajongó vagyok, úgyhogy nekem nagyon tetszettek a szállítók és a hanghatások.

Vajon miféle menüt lehetne összeállítani linuxos platformjátékokból ***Tux***, kedvenc kabalánk nélkül? A ***SuperTux*** egy klasszikus, ugorj és fuss stílusú platformjáték – mint sokan sejtik már, a ***Super Mario Bros*** nyomdokain halad. Fejlesztését ***Tobias (Tobgle) Glaesser*** vezeti, de eredetileg ***Bill Kendrick*** készítette; a ***SuperTux*** jelenlegi alfa kiadása garantáltan több óra kikapcsolódást nyújt bárkinek. Senkit ne tévesszen meg a program alfa állapota, tényleg érdemes kipróbálni.

A történet a következő: ***Tux*** és ***Penny*** (***Tux*** kedvese) egy szép napon éppen kirándulnak, amikor ***Tuxot*** leütik, ***Pennyt*** pedig elrabolja a rettenetes ***Nolok***. ***Tuxnak*** a legkülönfélébb veszélyekkel kell szembenéznie, hogy megmentse a gonosz ***Nolok*** nem kevésbé gonosz erődítményében bebörtönzött szerelmesét. Feladatunk az, hogy segítsük ***Tuxot*** a kutatásban. (*3. ábra*)

A ***SuperTux*** webhelyről (lásd a forrásokat) sokféle terjesztéshez és géptípushoz tölthetünk le bináris csomagokat, így némi szerencsével elkerülhetjük a forrásból fordítást. Ha mégis inkább emellett döntünk, akkor a ***SuperTux*** lefordítása a hagyományos öt lépésből áll:

```
tar -xjvf supertux-0.1.2.tar.bz2
cd supertux-0.1.2
./configure
make
su -c “make install”
```

A játék a kurzorgombokkal is gond nélkül irányítható, de botkormányt és játékpadot is használhatunk. A program elindításához a `supertux` parancsot kell kiadni. Indításkor többféle lehetőség közül is választhatunk, mint az azonnali játszmaindítás, bónuszszint betöltése vagy saját szint szerkesztése. Különféle beállításokat is módosíthatunk, ide értve az ***OpenGL*** támogatását, a hanggal és a zenével kapcsolatos beállításokat stb.

A játék pörgős és élvezetes. Az ellenségeket elég átugornunk, több gondunk nem lesz rájuk. A tárgyak különböző szinteken vannak, így az akadályokat a platformok között ugorva, mászva küzdhetjük le. Út közben aranyérméket kell gyűjtenünk. A jégtömböket fejjel szétzúzva energiavirágokat vagy jéglabdákat találhatunk, ezek ***Tuxot SuperTux***-szá változtatják, jóval komolyabb erővel és energiával. Ha befejezünk egy szintet, a következő, bonyolultabb szintre jutunk. Cikkem születésekor én a negyedik szinten jártam. Jaj, ne! Megint eltalált ez a csúszó jégtömb! Azt hiszem, újra kellene töltened a poharakat, ***François***. Az utolsó játékra tekintve az jutott eszembe, hogy jól esne egy ***Baked Alaska***. Látom, a záróra is közeleg, de ne féljetek, ***mes amis***. Az ajtók nyitva maradnak, és a bor sem fogy el. Nem sietünk sehova. A ***Baked Alaska*** még nem fogyott el, és ***Pennyt*** is ki kell még menekíteni, mielőtt ***François*** és én bezárhatnánk éjszakára az éttermet. Azt mondom tehát, igyunk egymás egészségére, ***mes amis***! A ***votre santé! Bon appétit!***

Linux Journal 2005. május, 133. szám

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

KAPCSOLÓDÓ CÍMEK

Blob Wars: ➲ www.parallelrealities.co.uk/blobWars.php

Scavenger: ➲ www.xdr.com/dash/scavenger.html

SuperTux: ➲ super-tux.sourceforge.net

➲ www.marcelgagne.com/wine.html