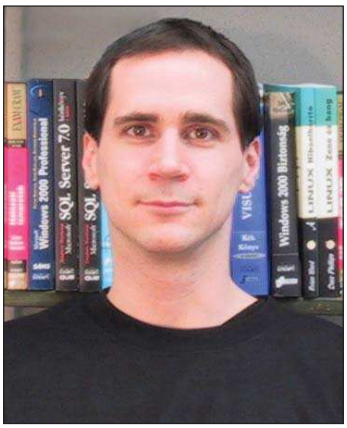
Beköszöntő



Szy György
a Linuxvilág főszerkesztője,
a Kiskapu Kiadó vezetője.
Mindenki levelét örömmel
várja a következő levélcímen:
Szy.Gyorgy@linuxvilag.hu

Szép új világ?

Szokásainknak köszönhetően szeptemberben mindig új élet kezdődik. Már augusztusban elkezdődik, mozgolódik és miután mindenki kibámészkodta magát a dunai csillagok háborújában, megszületik. Talán logikusabb volna az újévet augusztus huszadika környékére tenni. Bár... Végülis megszoktuk már ezt a Gregorián-számolgatósdit. Szóval, új évad. Ezalatt a nyár alatt, míg az emberek többségének a hőségtől még fagylaltot nyalni sem volt kedve, nagyon sok dolog megváltozott. Szervezetek átalakultak, új terveket szövögetnek, új emberek és csoportok léptek be a linuxos univerzumba, sőt, egyre-másra hallani a híreket, hogy ez vagy az, így vagy úgy szeretné meglovagolni a hullámokat. Ma már a politika és a jogi élet is visszhangos a szabad szoftverektől, a nyílt forrástól, és – nos, igen – a visszaélésektől.

Jó ez nekünk, vagy rossz?

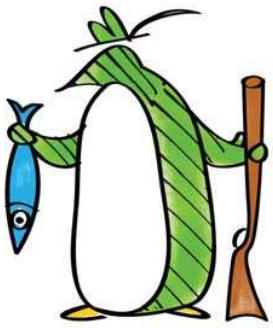
Bánkódjunk-e amiatt, hogy a nyugodt kis világunkat felforgatja a politika, átitatják a felhasználási szerződések miatt kitört botrányok? Ma már közhely, hogy ez várható volt. Amit sajnálok, hogy a közösség nem tudott elég gyorsan felkészülni ezekre a gondokra. És hogy rajtunk csattan az ostor, amikor a kihasználható és lefizethető politikusok bamba lustaságukban és személyes érdekeiket is jócskán szem előtt tartva hagyják magukat „meggyőzni" arról, hogy a pénzfaló óriásokat etetni kell, hogy a szabad szoftvert nem szabad támogatni. Hiszen, ha nem milliárdokba kerül, akkor még rendes kenőpénzekre sem számíthat az ember! Ugyanakkor ne legyünk igazságtalanok. Itt most nem arra gondolok, amit félig viccesen szoktunk mondani, hogy „nekik is élniük kell valamiből". Abban bízok, hogy a szabad szoftver – nyílt forrás – akárhogy is hívjuk a társadalom elég erős már ahhoz, hogy megvédje magát. A berozsdált, pénzéhes, hatalommal viszont művészien taktikázó óriások nyilván ki akarják lukasztani ezt a lufit. Bízom benne, hogy kénytelenek lesznek rájönni: nem lufit, hanem egy sziklatömböt szurkálnak.

Változások saját házunk táján

Igyekszünk mi is változni. Örömmel hívom fel olvasóink figyelmét például dr. Dudás Ágnes sorozatnyitó cikkére (Szoftverjog – barát vagy ellenség? 80. oldal), amelyben nem kisebb feladatra vállalkozik, minthogy bemutatja nekünk a szoftverjogot – egyszerű halandók által is érthető nyelvezetben. Remélem, sikerül tisztáznia sok kérdést és félreértést ezen a korántsem letisztult területen.
Szerkesztőségünkben személyi változások is történtek, remélem, ezután fő bűneink eltűnnek a szem elől és mindenkinek tetszik majd a végeredmény. Remélem, hogy már ebben a lapszámban sikerült olyan kínálatot összehozzunk, mely mindenkit érdekel. Számomra az egyik legkedvesebb téma e hónapban örömtelien nagy hangsúlyt kap: ez pedig a Windows és a Linux közötti átjárhatóság – programozás szempontjából. Rendkívül örülök például a Monóról szóló cikknek (Mono: gépfüggetlen hálózati alkalmazások, 16. oldal), remélem, hogy e gyönyörű félszerzet továbbra is megtartja gyors fejlődési képességét.

Mindenkinek kellemes olvasást kívánok. És ne feledjék! Érdemes körbenézni a nagyvilágban is, szép számmal jönnek a konferenciák, rendezvények, amelyeken kedvenc pingvinünk ügyes-bajos dolgait vitathatjuk meg!

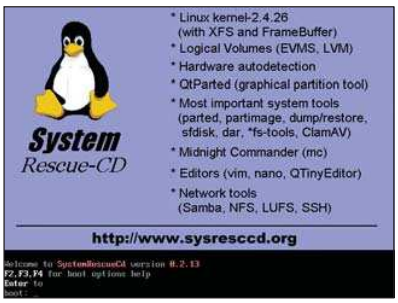
Programvadászat



Előző számunk CD-mellékletén a Slackware 10.0-s operációs rendszer kapott helyet. A visszajelzésekből következtetve, arra jutottunk, hogy a második korong anyagai közül is közreadunk jó pár előre elkészített csomagot. Ezzel is elősegítve eme kiváló rendszer terjedését.

SystemRescueCD

Ez a szintén Gentoo alapokkal rendelkező CD-ről indítható linuxváltozat, egy kitűnő segédeszköz azoknak a kezében, akik tudják, hogy a linuxszal hogyan lehet linuxot vagy más rendszereket menteni. Nagyon sok segédeszközt tartalmaz a rendszer, így semmilyen feladat végrehajtása nem okozhat gondot, használata nagyon egyszerű, csak elindítjuk a CD-ről és máris nekikezdhetünk a munkának.



A főbb rendszerelemek:

- GNU Parted – Lemezfelosztó eszköz
- QtParted – Partition Magichoz hasonló lemezfelosztó felület
- Fájlrendszer-eszközök: e2fsprogs, reiserfsprogs, xfsprogs, jfsutils, ntfsprogs, dosfstool
- Sfdisk – felosztás tábla mentő/visszaállító program

Nagyon fontos lehetőség a vak felhasználók támogatása, a Linux Speakup 1.5-ös képernyőolvasó remekül teszi a dolgát.

Ezt a CD-t leginkább rendszermentési feladatokra használhatjuk, arra viszont tökéletes. A fentebbi lista korántsem teljes, de az ott felsorolt programok jó esélyt adnak arra, hogy az ember a féltve őrzött, egy esetleges rendszerösszeomlás, vagy gondatlanság miatt elveszni látszó adatait megmenthesse. A rendszerről egy részletesebb leírás található Fábián Zoltán tollából a weboldalunkon.
(➲ http://www.linuxvilag.hu/system_rescue)

BitDefender

A BitDefender család, szinte mindenféle vírussal kapcsolatos gondra nyújt megoldást, a mellékleten a BitDefender for Linux Mailserver, BitDefender for Samba Servers és a webes (webmin) karbantaró felület található meg.

BitDefender for Linux Mailserver

A leveleket közvetlenül a levelező kiszolgálón ellenőrzi és a fertőzött üzeneteket hatástalanítja. Függetlenül a fogadott vagy küldött üzenet formátumától, a mellékletek számától és beágyazott mélységétől, BitDefender védelmet nyújt a fertőzött küldeményekkel szemben. Részletes jelentést küld a rendszergazdának ezzel is segítve a rendszer karbantartását. Támogatott levelező kiszolgálók: Sendmail, Qmail, Postfix. Szolgáltatások:

- vírusellenőrzés, az üzenetet és mellékleteit is ellenőrzi anélkül hogy lényeges terhelést róna a kiszolgálóra
- gyors működés, több címzettnek küldött levelek csak egyszer kerülnek ellenőrzésre, nem pedig minden egyes üzenet továbbításkor
- beépített SPAM szűrő
- telepítés internetről, a telepítés konzolon kiadott paranccsal indítható:

```
wget -q -O -
http://linux.bitdefender.com/
webinstall.sh | sh
```

- egyszerű telepítés és üzemeltetés
- az anti-vírus védelem egyszerűen beállítható bármilyen Linux-változaton
- figyelmeztető üzenet küldése
- vírusos küldemény észlelésekor jelentést küld a rendszergazda számára
- részletes jelentés és statisztika
- automatikusan készülő jelentés az ellenőrzött, fertőzött, hatástalanított, törölt és kiszűrt üzenetekről
- automatikus frissítés
- az okos anti-vírus adatbázis frissítés folyamatosan biztosítja az email védelem megbízhatóságát
- moduláris felépítés
- távoli felügyelet támogatása

BitDefender for Samba Servers

A BitDefender anti-virus felületfüggetlen motorja valós idejű védelmet nyújt a Windows, Dos, Linux és Unix vírusok és trójai programok ellen. Az alkalmazott belső gyorsótár-technológiának köszönhetően a valós idejű állományok ellenőrzési sebessége nagyon jó. Támogatott Samba kiszolgálók: v2.2.x és 3.0. Rendszerkövetelmény:

- Minimum P II 300 Mhz, 64 MB
- Linux rendszer: kernel 2.4.x glibc 2.2.3
- Samba kiadás: v2.2.x vagy 3.0

Rendszermag

A legfrisebb rendszermagforrások is helyet kaptak a korongon.

Csontos Gyula
(Csontos.Gyula@linuxvilag.hu)
A Linuxvilág szakmai és CD-szerkesztője. Szabadidejében szívesen mászik hegyet és kerékpározik.

Támogatás? Van.

Az EnfoBridge a jövőben magas szinvonalú támogatást biztosít az OpenOffice.org 1.1-es változatához is (sajnos csak angol nyelven). Ezzel a támogatás hiánya miatti, elsősorban a vállalati ügyfelek részéről esetleg felmerülő kifogások alaptalanná váltak. A támogatás 30 napig akár ingyenesen is elérhető, a 90 napos segítséget pedig mindössze 14 dollár fejében vehetjük igénybe. Mindez a Flexiety Software Company szolgáltatása, a cég már egyéb formákban, például frissítésekkel is segíti a szabad irodai csomag használóit.

➲ www.flexiety.com
➲ www.downloadopenoffice.org

A780 – mi maradt ki belőle?

A Motorola A780 jelöléssel immár harmadik Linuxra és Javára épülő mobiltelefonját jelentette be. Az egyszerűnek éppen nem mondható szétnyitható

készülék a zsebtitkárokéhoz hasonló, egynegyed VGA felbontású, színes érintőképernyőt kap majd, 240 kbit/s sebességű, GPRS/EDGE alapú adatátvitelre lesz alkalmas, rendelkezik majd Bluetooth-csatolóval, képes lesz a PDF és a Microsoft Office formátumú dokumentumok megnyitására, 1,3 megapixeles kamerája lesz, le tudja majd játszani az MP3 formátumú zenéket, cserélhető-bővíthető memóriával fog rendelkezni… További érdekessége, hogy négysávos telefon lesz, vagyis az amerikai 800/850 és 1900, illetve az európai 900/1800 MHz-es hálózatokat is támogatja.

Vizsgázni öröm

A PHP programozással foglalkozó ***php|architect*** magazin bejelentette

Zend tanúsítványközpontjának elindítását. A központ minden olyan szolgáltatást – online képzést, útmutatókat, gyakorló teszteket és vizsgalehetőséget – biztosít, amely a PHP programozói tudásukról valamilyen papírra vágyók számára szükséges. Az indulás után a központ akciókkal próbálja megnyerni a leendő ügyfelek rokonszenvét, illetve a mindent egy csomagban megvásárlók számára egyéb kedvezményeket is kínál.

➲ www.phparch.com

Bukik a SCO

A SCO gyakorlatilag elveszítette azt a pert, amelyet a DaimlerChrysler ellen indított az autóipari cég Linux-használata okán. A SCO célja a GPL szerződések érvénytelenségének és a Linux-használat jogszerűségének megkérdőjelezése volt, ám a bíróság gyakorlatilag semmiben nem adott igazat a SCO-nak. A SCO egy másik autós céget is beperelt, ám azt az ügyet elnapolták – igaz, megfigyelők szerint a SCO nem sokat veszít a késlekedéssel, legfeljebb annyit, hogy nem mondják ki a számára kedvezőtlen ítéletet. A SCO az IBM-mel is perben áll, de érvei meglehetősen ingatagok, így az ügy megnyerésére alig van esélye. A pereskedés legfeljebb arra volt jó, hogy néhány érdeklődőt megingasson vagy hátráltasson a Linux használatba vételében – ez viszont senkinek nem hoz hasznot. *Bruce Perens* szakértő ettől függetlenül óva int mindenkit attól, hogy a GPL-t jogszerűnek tartsa, erről ugyanis csak egy szerzői jogi per során lehetne ítéletet hozni; a mostaniak viszont nem ilyenek.

Itanium + Xeon = ?

2007-re várhatóan egyesül az Intel jelenlegi Itanium és Xeon processzortermékvonala. Az AMD Opteron processzorainak megjelenése miatt az Intel arra kényszerül, hogy megfelelő teljesítményű, 64 bites versenytárssal lépjen piacra, a megoldás a jelek szerint az eddig elit kategóriásként szereplő Itanium alsóbb piaci szegmensekre való bevezetése.

Az egyesítéshez először is olyan alaplapokat, foglalatot és sínrendszert kell tervezni, amelyek mindkét processzortípust képes támogatni, ezek a munkák már megkezdődtek. A váltás nemcsak a rendszerépítők számára jelent majd könnyebbséget, akik egyszerűbben építhetnek majd különféle teljesítményű kiszolgálókat, de a felhasználóknak is, akik könnyedén bővíthetik majd meglévő rendszerük erőforrásait.

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

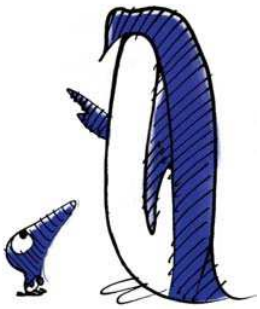
Mi újság a rendszermag fejlesztése körül?

Roland Dreier és az OpenIB.org csapat elkészítette az InfiniBand verem durva vázlatát, melybe belepakolták a Mellanox HCA alkatrészek alacsony szintű meghajtóit, néhány felső szintű protokollt (IP-over-InfiniBand, SCSI RDMA protokoll, sockets direct protokoll (SDP), uDAPL és az MPI), ezen kívül még néhány felhasználói eszközt.
A kód maga nyílt forrású, de a Microsoft fenntartja szellemi tulajdoni jogát az SDP-re és automatikusan nem engedélyezi annak használatát a nyílt forráskódú projektekben.
Ezért aztán *Roland* és társai kettéosztották az InfiniBand csoportot egy szabad és egy terhelt csomagra, a döntés egyelőre úgy tűnik mindenkit kielégít.
Az Intel SourceForge projektet indított a PRO/Wireless 2100 miniPCI hálózati adapterének 2.4 és 2.6-os sorozatú rendszermag meghajtójához.
Bár a firmware csak bináris formában érhető el, a projekt egyéb szempontból úgy tűnik nyílt forrású módszerekkel dolgozik. A fejlesztőket nyílt levelezőlistán tartják a kapcsolatot, a frissítéseket gyakran közzéteszik, így mindenki ki tudja őket próbálni és elmondhatja az észlelt hibákat és észrevételeit.
Egyelőre a kódot korai béta változat kategóriába sorolták, így különféle hibák és hiányzó képességek várhatók. Ugyanakkor az Intel fejlesztői nagyon oda kívánnak figyelni a különféle Linux terjesztésekkel kapcsolatos problémákra így remélhetőleg a legtöbb terjesztés alapértelmezett telepítéskor csakis ismert, dokumentált módon fog megjelenni.
Niraj Kumar áthozta Linux alá az UFS1 és UFS2-t. Az UFS1 már régóta használatos BSD fájlrendszer, az UFS2 pedig ennek friss kiterjesztése, melyben olyan bővítéseket találunk, mint a 64-bites blokkmutatók és továbbfejlesztett fájltárolás. *Niraj* Linuxos változata jelenleg csak olvasható, hiszen a munka még csak most kezdődött.
Az UFS2 maga is egészen új és még a BSD operációs rendszeren sem támogatja például a GRUB rendszerindítót. Csak a FreeBSD rendszereken alapértelmezett; a NetBSD továbbra is a hagyományos FFS fájlrendszert készíti alapértelmezés szerint. Az UFS2, amelyet eredetileg az UFS1 rendszerből vezetett le *Kirk McKusick* és *Poul-Henning Kamp* aktív fejlesztés alatt áll. Linux támogatás várhatóan gyorsan követi majd a BSD megvalósítást.
Michael Geng GPL engedélyű eszközmeghajtót készített az I2C-alapú SAA5246A Videotext/Teletext dekóderhez, amely a SAA5249 lapkameghajtóéval megegyező felületet használ.
Bizonyos részeken *Martin Buck* munkájára alapozó *Michael* kitisztította a meglévő kódot és befejezte a munkát, így végül a rendszermag hivatalos részeként *Andrew Morton* elfogadta a 2.6-os fába. Mint Michael rámutatott, az újabb TV kártyák már nem tartalmazzák ezeket a Teletex dekódereket, inkább a CPU-ra bízzák a funkciókat.

De ha mégis megtalálhatóak, ezek a lapkák úgy tűnik jobb munkát végeznek, így ha lehet érdemes támogatni őket.
Az Emulex úgy döntött, nyílt forrásúvá teszi a LightPulse Fibre Channel Adapter családjának meghajtóját és ennek érdekében SourceForge projektet hozott létre. Remélik a kód kitisztul és befejeződik végül pedig elfogadják a 2.6-os rendszermag fába. Általában amikor egy cég úgy dönt, hogy felszabadítja valamelyik meghajtójának forrását, egy sereg baráti üdvözletet kap a rendszermagfejlesztőktől, továbbá megjegyzéseket, kritikát a foltokat a kódot elsőként átnéző fejlesztőktől. Ez esetben *Jeff Garzik* végezte a legtöbb vizsgálódást, visszajelzések tonnáival árasztva el az Emulex fejlesztőit. Jelenleg van egy két elég csúnya rész a kódban, amire az Emulexes fiúk figyelmeztettek is bejelentésükben, de az Emulex úgy tűnik hűen követi a tisztítási kívánságokat amelyek szükségesek ahhoz, hogy Andrew és a rendszermagfejlesztők elfogadják a meghajtót.
Kristian Soerensen mostanában az Umbrellán dolgozik. Ez az új, kézigépekhez tervezett biztonsági eszköz segít a vírusok és egyéb feltörési próbálkozások elleni harcban. Az Umbrella egyik fő előnye a félreérthetetlen beállítási rendszer. Minden összetettséget száműztek, így a felhasználó nem fog szándéka ellenére nemkívánatos réseket létrehozni.

Zack Brown

Linux Journal 2004. július, 123. szám



A Linux Journal honlapján számtalan gond megoldásához találhattok további segítséget. A Sunsite tüköroldalait, a gyakori kérdéseket és az egyéb útmutatásokat a ➲www.linuxjournal.com honlapon olvashatjátok el. A rovatban közzétett válaszokat Linux-szakértők kis csapata készítette el. További kérdéseiteket szívesen fogadják (angol nyelven) a ➲www.linuxjournal.com/lj-issues/techsup.html címen, ahol csak egy kérdőívet kell kitöltenetek, de a bts@ssc.com címre levelet is írhattok. A levél tárgyában szerepeljen a „BTS" kulcsszó.

A hónap szakmai tanácsai

AOL Linux alatt?

Kettős rendszerindításra képes gépem van (GRUB-ot használok), az egyik lemezrészre Red Hat Linux 9.0, a másikra Windows XP Professional rendszert telepítettem. Az internetszolgáltatóm által adott AOL 9.0 szélessávú alkalmazást is használom. Létezik illesztőprogram az AOL Broadband Blaster modemhez, amellyel a Red Hat 9.0 rendszerem is képes lenne az AOL szélessávú kapcsolatán keresztül az internet elérésére? Nem szeretnék másik szolgáltatóhoz átmenni, új DSL modemet sem szeretnék vásárolni, de képtelen vagyok rávenni a Red Hatot a Broadxent modem felismerésére. És a Linux az AOL szélessávú programcsomagját sem ismeri fel.
Natosha, NZimardo@aol.com

Az AOL programcsomagja Linux alatt nem használható, ezért a hagyományos módszerrel nem tudod elérni az AOL szolgáltatásait. Ha viszont általános célú internetelérést szeretnél, akkor ajánlom figyelmedbe, hogy számos Broadxent DSL modem USB és Ethernet csatolófelülettel egyaránt rendelkezik. Ha módodban áll áttérni az Ethernet felület használatára, illetve be tudsz szerelni egy ethernetkártyát a gépedbe, akkor mindkét operációs rendszer alól el fogod tudni érni az internetet. Ha DSL modemed nem rendelkezik ethernetcsatolóval, akkor vagy olyan USB-s eszközre kell lecserélned, amelyet a Linux is támogat (például az Alcatel és az ECI termékei), vagy olyanra, amely Ethernet csatlakozással is fel van szerelve. Ha az AOL nem hajlandó segíteni neked mindebben, akkor javaslom, hogy nézz másik szolgáltató után. A linuxos asztali gépek napjainkban egyre népszerűbbek, különösen a Lindows térnyerésének és az olyan terjesztők erőfeszítéseinek köszönhetően, akik a kereskedőkkel együttműködve könnyen megfizethető gépekhez mellékelik rendszerüket.
Chad Robinson, crobinson@rfgonline.com

Linuxos erőmű összeállítása

A weben és nyomtatásban egyaránt figyelemmel követtem Glenn Stone „Az év linuxos erőműve" című cikksorozatát, és úgy döntöttem, megépítem a saját példányomat. Anyagi okokból úgy gondoltam, részegységenként vásárlom össze a gépet: minden hónapban megveszem egy-egy darabját, majd amikor minden megvan, összeállítom a masinát. Nem kevés kutakodás után két tanulságot vontam le. 1. Moduláris rendszerben kell gondolkodni. 2. A 64 bit a jövő. Úgy határoztam tehát, hogy egy egyprocesszoros AMD Opteron gépet állítok össze, operációs rendszere a Mandrake Linux 9.2/64 változata lesz. Az ASUS SK8N alaplapját szemeltem ki, emellé 2 GB RAM, két darab Maxtor 120 GB SATA HDD, 3,5-ös hajlékonylemezes meghajtó, 5,25-ös hajlékonylemezes meghajtó (a munkám miatt szükséges) kerülne, lesz hozzá CD-RW-meghajtó (48x/12x/48x) és, ha minden igaz, egy DVD-lejátszó is. Vajon jól működik ez az alaplap Linux alatt? Ha nem, akkor tudsz olyan ajánlani, amivel nem lesz gond?
S.W. Bobcat, swbobcat@hotmail.com

Ha gondjaid vannak ezzel az alaplappal, akkor szinte biztos, hogy az illesztőprogramok okozzák a bajt. Az alaplapokkal kapcsolatos hibák túlnyomó része valamilyen módon megkerülhető, például a noacpi rendszerindítási beállítással. Az említett alaplapot 2003-ban számos SPEC.org teljesítménypróbánál használták, így erősen kétlem, hogy az esetleges hibáira ne volna megoldás.
Szeretném viszont javasolni, hogy a 64 bites rendszer építését jól gondold meg. Ha nem végzel komoly képleképezési vagy matematikai munkákat, a 64 bites gép nem fog számottevő előnyöket nyújtani. Sőt, azt fogod észlelni, hogy bizonyos alkalmazások lassabban futnak. A teljesítménypróbák eredményei jelenleg azt mutatják, hogy ha egy programot nem kifejezetten 64 bites gépre írtak, akkor nem képes a gép tudásának kihasználására.
64 bites gépet elsősorban különféle számítási feladatok végrehajtására érdemes alkalmazni, illetve olyan területeken, ahol nagyobb egész számokat kell indexeléshez használni, például adatbázisok kezelésére. A Descent 3 egy cseppet sem lesz gyosabb.
Nem kérdéses, hogy számos területen jól ki lehet használni ezeket a képességeket, feltéve persze, hogy minden programot újrafordítasz. Nem vitatható viszont, hogy az Opteron egy nagyszerű processzor, amely a HyperTransport megoldásnak köszönhetően kitűnően fog teljesíteni a te gépedben is.
Chad Robinson, crobinson@rfgonline.com

Tartok tőle, a modularitás tekintetében nem értek egyet veled. A személyi számítógépek mára bonyolultakká váltak, így előfordulhat, hogy amikor összeáll a gép, a részei nem működnek megfelelően együtt, vagy pedig titokzatos hibákat tapasztalsz. Persze nem törvényszerű, hogy így történik, ám a PC-k sebességnövekedése több ezres nagyságrendű, így az „összekutyulom, és csak lesz belőle valami" megközelítéssel elég nagy esélyed van arra, hogy rejtélyes időzítési hibákra bukkanj két alkatrész között. Amint magad is írod, a 64 bit a jövő. Ez így van, de egyáltalán nem biztos, hogy amire te fogod használni a gépet, ott tényleg kell az a 64 bit. Lehet, hogy esetedben a „várjunk és majd meglátjuk" szemlélet jobb eredményre vezetne, s közben talán még egy jó 32 bites gépre is szert tehetnél.

Arról nem is beszélve, hogy csak az utolsó alkatrész megvásárlása után lesz valóban működő rendszered, feltéve persze, hogy szerencséd van, és valóban működik az a vaskupac. Ha az alkatrészeket egyenként veszed meg, akkor nemcsak többet fogsz fizetni értük, de az elsőként kifizetett részegység ára a felére fog esni, mire az utolsó darabot is hazaviszed. Ha nem vagy megszállott gépbütyköl, és nem akarsz amiatt bosszankodni, hogy a különféle alkatrészeket cserélgetned kell, mert nem működnek együtt, akkor javaslom, hogy előre összeállított gépet vegyél.
Marc Merlin, marc_bts@google.com

Szerintem inkább tedd félre az alkatrészek árát, és vedd meg őket egyszerre. Így az első alkatrész garanciája nem fog lejárni, mire az utolsót is megveszed. Azt se feledd, hogy, amint Marc is írja, a PC-alkatrészek ára folyamatosan csökken. Ha saját linuxos gépet szeretnél építeni, akkor látogass el a linuxos rendszerépítők weboldalaira, és nézd meg, hogy ők milyen eszközökkel dolgoznak. Valószínűleg azok gond nélkül működnek egymással.
Don Marti, dmarti@ssc.com

Új szolgáltatás hozzáadása

Írtam egy SMS rendszert Javában, amelyet háttérszolgáltatásként szeretnék futtatni Linux alatt. A rendszer egy ****.jar*** állományban van. Hogyan valósíthatom ezt meg a Red Hat-ben szolgáltatásként?
Kasun Perera, kasun@teamwork.lk

Létre kell hozni egy parancsállományt, amit a ***/etc/rc.d/init.d/*** könyvtárba teszel. Nagyon pontosan meghatározott formátumúnak kell lennie, ahogy ezen az oldalon világosan kiderül: ➲ http://www.sensi.org/~alec/unix/redhat/sysvinit.html. Azt javaslom, nézz meg más parancsfájlokat abban a könyvtárban, hogy megértsd a fájl általános formátumát, különösen az első, hozzávetőleg 15 sort.
Felipe Barousse Boué, fbarousse@piensa.com

Új szolgáltatás indításakor szeretem lemásolni az SSH beállítófájlját, mert általában ez a legegyszerűbb. Tedd bele az összes parancsot, ami ahhoz kell, hogy parancssorból el tudd indítani a programot. Esetleg be kell állítani néhány környezeti változót egy Java program futtatásához. Futtasd a parancssorból a beállító fájlt, hogy meggyőződj, rendesen elindítja és leállítja az új szolgáltatást, majd a terjesztés eszközeinek segítségével állítsd be a futási szinteket, hogy indításkor futni kezdjen. Red Hat-ben használd a `chkconfig`-ot.
Don Marti, dmarti@ssc.com

A Webmin gyorsítása

Webmint és Zonemindert használok a 2.2GHz, 1GB RAM rendszeremen, és az eredmény elmarad a várttól. Van valami mód arra, hogy felgyorsítsam a hurok eszközt?
Howard Watts, howardwatts@sbcglobal.net

Nemrég frissítettem néhány Webmin rendszert az e sorok írásakor legfrissebb, 1.140-es Webminre. Lényeges javulást tapasztaltam a működésben. Az összes modult is frissítettem, mindezt közvetlenül a Webmin alól.
Felipe Barousse Boué, fbarousse@piensa.com

A hurokeszközön (`lo`) az egyetlen késést a TCP verem tiltása okozza, de ennek gyorsan kellene működnie. Ha teljesítmény-problémáid vannak, érdemes megnézni a top nevű programot (lásd a top súgóoldalát), hogy kiderüljön, vannak-e ellenőrizetlen folyamatok.
Christopher Wingert, cwingert@qualcomm.com

Winmodem: bütyköljem vagy cseréljem?

Nem tudom működésre bírni a Creative Labs Blaster v92 PCI belső modememet. A Linux Conexant lapkakészletet ismert fel, és megpróbálta telepíteni a meghajtót, de hibaüzenetet kaptam. Próbáljam meg inkább a SuSE Pro 9.0-val?
Manny, manuel61@joimail.com

Valószínűleg a legkönnyebb, és legjobban ajánlható megoldás egy nagyon olcsó modem beszerzése, ami nem Winmodem. Bizonyára könnyebb lesz a telepítés, kevesebb bonyodalommal, és olyan modemed lesz, amely sok-sok Linux nemzedéken át kitart. Emellett jelezd a gyártóknak, hogy mindannyian szabványos modemeket akarunk, nem szabadalmazottakat.
Felipe Barousse Boué, fbarousse@piensa.com

Szinte sosem kell az egész operációs rendszert frissíteni csak azért, hogy egy eszközt támogassunk. Anélkül, hogy ismerném a pontos lapkakészletet, amit használsz, csak azt javasolhatom, hogy először állapítsd meg, hogy a te modemed „tiszta vas”, vagy úgynevezett Winmodem. Azt gyanítom, hogy az utóbbi, különben nem lennének nehézségeid. A Linux terjesztés frissítése helyett derítsd ki, melyik meghajtót próbálta betölteni, és keress egy frissebb meghajtót a weben. A Winmodemek támogatottsága a Linux alatt egyre növekszik (a ➲ http://www.linmodems.org például jó kiindulási hely). Ha szerencséd van, megtalálod a keresett meghajtó újabb, működő változatát, és megspórolsz egy csomó fáradságot.
Timothy Hamlin, thamlin@zeus.nmt.edu

Valamilyen nyakatekert tanulmányt keresel, ami segít megérteni a Winmodemeket, vagy egyszerűen internet-kapcsolatot szeretnél? Legyen cél a szemed előtt, mielőtt választanál a fenti két válasz közül, és ne feledd, ha frissítesz, lehet, hogy újra telepítened kell a modemet.
Don Marti, dmarti@ssc.com

Első lépések

Szeretném megtanulni a Red Hat 9-et. Telepíthetem a Windows 2003 Serverrel (Beta) egy gépre? A következő számítógépem van: 700MHz-es Dell PIII, 6GB-os merevlemezzel és 128MB RAM-mal. Tudom, hogy fel kell majd osztanom a merevlemezt. Láttam egy programot, amit az *Amazon.com*-on árultak, kb. 70 USD-ért. Ez még új nekem és próbálok tanulni, szóval bármilyen segítséget megköszönök.
Bill, whitesock95829@yahoo.com

Igen, lehet ugyanarra a gépre Linuxot és Windowst telepíteni, ezt kettős indítású rendszernek hívják. Néhány részletre azonban figyelni kell. A Red Hat 9 fejlesztését abbahagyták, szóval a Linuxszal való játékra és tanulásra a Red Hat által támogatott Fedora Core 1-et választanám inkább.
A ➲ http://fedora.redhat.com címről letöltheted.
Felipe Barousse Boué, fbarousse@piensa.com

A Linuxot úgy próbálhatod ki legkönnyebben, hogy letöltöd a Knoppixot a ➲http://knoppix.org-ról. Így kísérletezhetsz a Linuxszal és nem kell módosítanod a merevlemezt. A legtöbb terjesztés ingyenesen letölthető, a Red Hat 9 is. Érdemes inkább egy újabb terjesztést keresni, mint például a Fedora Core 2.
Christopher Wingert, cwingert@qualcomm.com

Én magam nem próbáltam indításkezelőt telepíteni a Windows 2003 Serverhez, és nem is áll szándékomban, de sok emberről hallottam, akiknek sikerült. Az eljárás hasonlónak tűnik, mint a Windows korábbi változatainál. Íme néhány fontos dolog, amit tudni kell:

1) A Knoppix tartalmazza a *QtParted*-et, egy ingyenes felosztásszerkesztőt. A felülete nem olyan kifinomult, de a lemez felosztására ugyanolyan jó, mint a PowerQuest Partition Magic-je, és más fizetős programok. Talán még lelkes is leszel, hogy ingyenes programmal végzed el a feladatot. A Knoppix mellesleg nagyon jó biztonsági lemezt készít.
2) Legalább egy lemezrész kell a Linuxnak, és még 128MB lapozórésznek (swap partition). A vélemények eltérnek a felosztással kapcsolatban, van aki szereti a */home*, */var* és egyéb könyvtárakat külön lemezrészeken elhelyezni. Mivel te csak most kezded, jobb, ha csak egyet használsz.
3) Szükséged lehet egy FAT32 lemezrészre is, hogy megoszthasd a fájlokat az operációs rendszerek között.
4) Ha a Windows még nincs telepítve, akkor először telepítsd azt, utána oszd fel a lemezt. A régi Windowsok nem tűrtek meg másik operációs rendszert telepítés közben. Megpróbálhatod megoldani a problémákat, de egyszerűbb, ha megelőzöd.
5) Amikor telepítesz, ne felejtsd el telepíteni a GRUB indításkezelőt. A telepítés magától észleli a Windows jelenlétét. Az indításkezelő betöltődik a gép indításakor, és felajánl egy menüt, amiből kiválaszthatjuk, melyik operációs rendszert akarjuk indítani.

Bruce Byfield, bbyfield@axionet.com

Nem kell átméretezni a meglévő lemezrészeket felosztásszerkesztővel, ha van még helyed a merevlemezen. Minden Linux terjesztés tartalmaz valamilyen egyszerű lemezfelosztó eszközt. Ha felosztásszerkesztőt használsz, ne feledd, hogy ha valami elromlik, fontos adatok veszhetnek el. Mindenképp csinálj biztonsági mentést az aktuális rendszerről, és ellenőrizd, hogy a mentés sikeres volt, mielőtt átméreteznéd bármelyik lemezrészt. Vagy, ahogy Rick Moen javasolja, ha megvan a biztonsági másolat, vissza is állíthatod új lemezrészekre, ezzel teljesen megspórolod az átméretezést.
Viszont a 6GB-os lemezed túl kicsi, hogy kényelmesen futtass két operációs rendszert. Beszerezhetsz egy új, nagyobb meghajtót a Linux számára.
Még több tanács található az új felhasználók számára, (többek között arról, hogy miért rossz ötlet a kettős indítás), a Linux Journal webhelyén, a „Welcome to Linux, 2004” című cikkben.
(➲ http://www.linuxjournal.com/article/7516).
Don Marti, dmarti@ssc.comL

A Linux Journal weboldalairól számos súgó és egyéb erőforrás érhető el. A Best of Technical Support rovatban közzétett válaszokat egy Linux-szakértőkből álló csoport adja. Ne feledd el megadni, hogy milyen terjesztést használsz, a rendszermag melyik változatát futtatod, mi a gondod, valamint írj meg minden olyan adatot, amelyről úgy gondolod, köze lehet a hibához.

Linux Journal 2004. április, 120. szám

Linux Journal 2004. július, 123. szám

2004-es Szerkesztői Díj (Editors' Choice Awards)

Izgatottan vártunk néhány linuxos eszközt és programújdonságot de nem igazán tudtunk elszakadni a régi favoritoktól sem.

Egyre nehezebb és nehezebb az újonnan megjelenő Linuxos termékekkel, szolgáltatásokkal és projektekkel lépést tartani. Szerencsére a szerkesztőink körét az utóbbi néhány évben kibővítettük és Szerkesztői díj egészen előkelővé vált. Minden további hűhó nélkül lássuk a 2004-es év Szerkesztői Díjait.

Kiszolgáló: HP ProLiant BL20p G2

Az Ibrahim Haddad által javasolt HP ProLiant BL20p G2 gép két Intel Xeon processzorral rendelkezik, alaplapra szerelt RAID, két gyorscserélhető SCSI meghajtóval, három Gigabit Ethernet csatlakozóval vértezték fel, valamint egy további Ethernet kapcsolattal is bír a karbantartás végett. Ezen felül Fibre Channel kapcsolattal is felszerelhető. Mindez már egy 1U fiókkiszolgálóban sem lenne rossz, csakhogy ez egy pengegép, így akár nyolc darabot is elhelyezhetünk egyetlen 6U szekrényben maximum hat redundáns áramforrással, tetszőleges hálózatkapcsolóval vagy más kapcsolattartó eszközzel egyetemben.
Ha eddig csak a csinos laptop merevlemezek miatt nem rajongtunk a pengékért, vessünk még egy pillantást a nehézsúlyú pengekiszolgálók új nemezedékére.

Személyi számítógép és munkaállomás: IBM ThinkPad T41

Minthogy köztudottan minden szerkesztő más és igen mélyen gyökerező véleménnyel van saját munkakörnyezetéről, mindannyian meglepődtünk, mikor ***Doc Searls***, ***Ibrahim Haddad*** és ***Robert Love*** egybehangzóan állította: az IBM ThinkPad T41 ma a legkívánatosabb Linux laptop. És nem csak egyszerűen a ThinkPad vagy ThinkPad T sorozatban egyezett a véleményük – mindannyian ugyanazt a modellt kedvelik és használják.
Doc, aki így dícséri a T41-est „Külsőre ipari erőt sugároz és dolgozni vele mintha versenyautót vezetnél" a nagy teljesítményét kedveli benne. „Minden működik Linux alatt", jegyezte meg ***Robert***. Hova tűntek a régi szép napok, amikor a kernelbütykölőkre vártunk, hogy megvegyék a még nem támogatott laptopokat és elvégezzék számunkra a munkát? A T41 1400×1050-es kijelzővel rendelkezik és az IBM híres három éves garanciája, valamint gyors és szakértő szervízszolgáltatása jár hozzá.
Bármely gép, amelynek sebessége a zsírozott egerekhez mérhető legalább egy elismerő említést megérdemel, ***Greg Kroah-Hartman*** pedig éppen ilyen hasonlattal illette a Apple Power Mac G5 kétprocesszoros változatát, ami mindössze egyetlen Linux telepítésnyire áll a nagyszerű rendszerré válástól. „Gyors, csendes és jó ránézni. Teljes 64-bites teljesítmény nagyon olcsón, ki ne szeretne ilyesmit?" írta.

Biztonsági eszköz: Clam AntiVirus (AV)

Reuven Lerner szerint, „a ClamAV miatt az üzleti vírus-ellenőrző programok tényleg futhatnak a pénzük után. A ClamAV és a SpamAssassin együttese nagy mértékben lecsökkentette a kiszolgálómon áthaladó idegesítő (és potenciálisan veszélyes) levelek mennyiségét."
Az idei év nem-Linux felületeken kitört e-mail féreg invázióját a ClamAV igen jól kezelte és a levelezőlistákon a Linux rendszergazdák jelentése szerint az adatbázis frissítési idők elérték vagy túl is szárnyalták az üzleti változatokét. Ó igen, ma már üzleti támogatás is elérhető.

Webböngésző vagy ügyfél: Mozilla Firefox

„Kezdem azt hinni, hogy a Mozilla az új Emacs – az a gépfüggetlen program, amely egységes és bővíthető" írja ***Reuven***. Júliusi számunkban rövid példakódot valamint bemutatót találunk a Mozilla alapú alkalmazások fejlesztéséről, ha pedig felbukkanó ablaktól mentes, szabványtámogató böngészésre vágyunk, csak vessünk egy pillantást a legközelebbi Linux asztalra.

Grafikus program: The GIMP

A GIMP Projekt végre kiadta várvavárt 2.0-ás változatát, visszaszerezve szerkesztőink kedvenc grafikus eszközének megtisztelő címét. ***Marcel Gagné*** szerint, „Az EXIF kezelés, CMYK támogatás és az egyszerűbb, jobb felület megjelenésével a The GIMP program továbbra is vetélytárs nélkül maradt a Linux asztalomon."

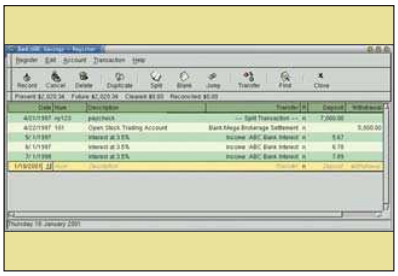
Kommunikációs eszköz: mutt

Bár a gyorsüzenet-küldők és grafikus levelezők általában a teljes bemutatóidőt kitöltik a Linuxos rendezvénye-

ken, a szöveges alapú mutt levelező – melyben lényegében mindent beállíthatunk – továbbra is kultusz-remekmű marad. *Greg* szerint, „e nélkül sehogy sem boldogulnék a napi 500 üzenetre rúgó levéladagommal."
Don Marti Ximian Evolution-t használ naptárként és kapcsolatai nyilvántartására, de levelezésében ragaszkodik a mutthoz. A muttot és a Mozillát együtt használva kényelmesen nézhetjük a csatolmányokat de egy kis egészségesen felspécizett beállításfájl felhasználásával a webkeresést is kipróbálhatjuk a „`my .muttrc`" segítségével.

Felhasználói program: GnuCash

„Pár hónappal ezelőtt kezdtem el használni a GnuCash-t és teljesen lenyűgöztek a képességei", írja *Reuven*.



GnuCash

„lenyűgöző mennyiségű képességgel rendelkezik és Guile segítségével programozni is lehet. Ha még soha nem próbáltuk meg magunk kezelni a pénzügyeinket esetleg bizonytalanok vagyunk a kettős könyvvitelt illetően, a beépített útmutató segít nekünk az indulásban." Pénzügyi eszköz kettős könyvelés nélkül, olyan mint egy grafikus program rétegek nélkül.

Programkönyvtár vagy modul: Pango

Ez ugyan új kategória, de épp itt az ideje hogy elismerjük a könyvtár-karbantartókat is. A kódkönyvtár időt takarít meg és segít elkerülni jó pár hibát, hiszen lehetővé teszi, hogy az emberek kihelyezzék alkalmazásaik egy részét. Mindig örömmel látjuk amikor a fejlesztők jó könyvtárakat használnak fel és nem írnak mindent újra a kezdetektől. *Reuven* kijelentette, „Szeretnék köszönetet mondani valamennyi keményen dolgozó embernek, akik a Pango és egyéb nemzetközi támogatást nyújtó könyvtárak és programok fejlesztésében részt vállaltak és ezzel a nem-nyugati típusú írásokat is elérhetővé tették Linux alatt. Nektek köszönhetően, emberek milliárdjai, akik nem beszélnek, olvasnak vagy írnak angolul, használni tudják a nyílt forrású programokat. Az, hogy Mozilla hagyományos változatával Héber leveleket írhatok és OpenOffice.org normál változatával is megtehetem ugyanezt, folyamatosan lenyűgöz."

Fejlesztőeszköz: BitKeeper

Greg szerint, „Segítségével rendszermagfoltok millióival birkózhatok meg sikerrel. Ez az egyetlen lehetőség, hogy sikeresen karbantartsak hét különböző rendszermagfát, és még aludni is maradjon időm."
Linus Torvalds meglepő kijelentést tett a BitKeeper cég sajtóközleményében – „Több mint kétszer gyorsabbá tette a munkámat" mondta. Mintha ez idáig olyan lassú lett volna. Ilyen ajánlólevéllel a BitKeeper minden új forráskódkezelő programot kereső cég jelöltjei között helyet kell kapjon.

Adatbázis: PostgreSQL

„Továbbra is szeretem a PostgreSQL és előnyben részesítem a MySQL-lel szemben képességei, üzembiztonsága, méretezhetősége, Unicode támogatása és a szabványokat követő megvalósítása miatt ", írja *Reuven*. „Azt mondják a MySQL csapat lenyűgöző betörést hajtott végre és nagyon várom, hogy az elkövetkező években felzárkózzanak a PostgreSQL szintjéhez. Egyelőre azonban mindenkinek a PostgreSQL-t ajánlom akinek relációs adatbázis-kezelőre van szüksége." ért egyet *Marcel*. „Nálam még mindig a PostgreSQL az első", írja. „felnőtt, nagy tudású adatbázis-kezelő, és az első amihez nyúlok ha adatbázist használó alkalmazást készítek vagy használok."

Játék: Really Simple Syndication (RSS)

Szerkesztőink valamennyien üzletemberek és egyből felhúzzák az orrukat ha játékokra kell szavazni. Éppen ilyen emberekre van szükségünk ha ki szeretnénk használni cégünk asztali gépeit. Igaz ugyan, hogy nem egészen úgy néz ki mint a Quake vagy a Frozen Bubble amikor a főnök arra jár, de a Neten játszó Linuxosok körében ez az új favorit. Hívhatjuk bloggolásnak vagy szociális programnak, a játékosok mindenünnen érkeznek. Olyan mint *Dungeons and Dragons* figurákat festegetni vagy focikártyákat gyűjteni, csak éppen valódi emberekkel.
A ragasztó pedig ami mindezt egyben tartja egy egyszerű XML-alapú szindikátus formátum, amelyet RSS néven ismert meg a világ, s melynek honlapjai (például a Technorati) és programprojektjei (például a Planet) teljesen új megközelítéssel kezdték el összeszervezni a webtartalmat.
Ki a Blogkirály és ki a bozo? Ugorj be a Technoratira és kukkantsd meg a pontokat.
Reuven rámutatott, hogy a LinkedIn, Orkut és Ryze stílusú, mindent az egyben szociális lapok nem különösebben hasznosak, de az állítja, hogy „mindannyian valami olyasmit feszegetnek ami új és érdekes." A dolog akkor kezd izgalmas lenni, amikor a szociális információ átszeli a lap határait és bárki végigmászhat rajta. Indul a játék!

Szakkönyv (döntetlen): Real-World XML és Hacking the Xbox

Paul Barry januári számunkban azt írja: *Andrew „nyuszi" Huang* Hacking the Xbox című műve „fene jó olvasmány". A könyv gyakorlatias útmutató arról, hogyan használjuk a gépet úgy, ahogy nekünk tetszik és nem úgy ahogy bizonyos cégek üzleti modelljében elgondolták.
Reuven szerint a Real-World XML *Steven Holzner*-től „egy újabb nagy, vaskos könyv az XML-ről, amelynek valójában nemigen van szüksége nagy vaskos könyvekre. Azonban tartalmaz néhány példát, mintakódot valamint megvitat pár alkalmazást, többek közt a SOAP-ot."
Aki a szépen megszerkesztett könyveket kedveli, nézze meg *Huang* könyvét; aki inkább a jó kötőanyagra vágyik, szerezze be *Holzner* művét. Tágítsuk ki látókörünket.

Nem szakkönyv: Free Culture (Szabad kultúra)

Vajon a Beatallica néhány tagja a Beatles tiszteletbeli tagja szeretne lenni, míg

más tagjai inkább a Metallicát választanák? Sajnos nem mehetünk el meghallgatni a „Got to Get You Trapped Under Ice" és az „Everybody's Got a Ticket to Ride Except for Me and My Lightning" című számaikat, ugyanis a Beatallica a lemezkiadók ügyvédjeitől való félelmében elbujdosott.
Nem mindig volt ez így. Régebben, amikor még ***Walt Disney*** a Steamboat Willy-t, Buster Keaton Steamboat Bill, Jr. Művének paródiáját rendezte, a copyright törvények mások voltak és a kreativitást támogatták, nem az ügyvédek számláját. ***Lawrence Lessig professzor*** Free Culture című művében számunkra, Linux és Internetes népek számára is hasznos dolgokat mesél a szerzői jogról, elmagyarázza a jelenkor copyright gyakorlatát azoknak akik még nem ismernék az egész szomorú történetet. Lessig a gyakran figyelmen kívül hagyott középutat képviseli a szerzői jog vitájában.

Műszaki weblap: LWN

LWN ismét nyert. Ugyanazt tudjuk mondani erről a lapról amit tavaly: kiváló Linux-történet gyűjtemény egyéb oldalakról, ideértve a Linux Journal lapjait, valamint néhány eredeti műszaki tartalom. Az aktuális sorozat néhány naptár, képnézegető és rajzoló-programot mutat be.

Nem műszaki vagy közösségi lap: Groklaw

Ha eladtad a televíziódat amikor a L.A. Törvény felröppent ez a te lapod. Hagyd, hogy magába szippantson a bukott UNIX forgalmazó, a korábban Caldera néven ismert SCO csoport tárgyalótermi drámája, és a cég véget nem érő pereskedése az AutoZone, Daimler-Chrysler, IBM és Novell cégekkel. Vajon a SCO ki tudja majd kerülni a Red Hat perét? Átadta a Novell a UNIX szerzői jogait a SCO-nak? Vajon ***Grace Victorral*** együtt jön vissza? Greg szerint a Groklaw „lapjain több IBM igazgatót találni mint bármely más lapon."

Mobile Eszköz: Sharp Zaurus SL-6000 PDA

Ibrahim Haddad választása a legfrissebb Zaurus. A korábbi Zaurusoktól eltérően, ebben már megtaláljuk az USB támogatást, így USB eszközeinkkel együtt használhatjuk tárolásra, hálózatkezelésre és adatbevitelre. A képernyő finom 480×640, négyszerese az eredeti Zaurus felbontásának és megegyezik a múlt évben bemutatott, előző, csak japánban forgalmazott Zaurus SL-C700 felbontásával.

Sharp Zaurus SL-6000 PDA

Levelezőlista vagy más támogatói fórum: linux rendszermag lista (linux-kernel list)

Greg sokáig latolgatta támogassa-e a linux rendszermag listát: „nagyhangú, gyakran goromba, de mindig informatív és soha nem unalmas. És ha a felhasználó hajlandó rendes lenni, igen hasznos is" mondja. Úgyhogy légy rendes. Lehetőleg.

Az év projektje: Ardour

Az Ardour nevű digitális hangstúdió volt a Linux-alapú rögzítőstúdió központi eleme ***Aaron Trumm*** májusi cikkében. ***Dave Phillips*** pedig a Linux Journal weblapjára szánt cikkében azt írja, „az Ardour az érdeklődés középpontjába került mindannyiunk szemében, akik profi szintű hangvágással dolgoznak" valamint „az, hogy az Ardour ilyen messze eljutott és ilyen gyorsan fejlődik, ékesen bizonyítja programozóinak tehetségét és hozzáértését." gratulálunk ***Paul Davis***-nek és az Ardour csapat többi tagjának.

Az év terméke: EmperorLinux Toucan

Emlékeznek még az IBM ThinkPad T41-re, a laptopra amit mindenki kedvel? Doc a sajátját az EmperorLinux-on keresztül vásárolta, e barátságos emberekkel teli cégnél, akik a fővonalbeli laptopokat nekünk tetsző terjesztéssel szerelik fel, foltozott és tesztelt rendszermaggal látják el, amely megfelelően támogatja a laptop alkatrészeit. Az Emperor, Linuxszal felvértezett T41-esét „Toucan" néven árusítja és a hat terjesztés közül bármelyiket fel is telepíti rájuk, illetve kérhetünk kettős indítási lehetőséget Microsoft OS rendszerrel.
Ami a legjobb az egészben, az EmperorLinux nagyon gyorsan válaszol Linux ügyekben miközben a gyártó eredeti garanciáját is megtarthatjuk az alkatrészekre.
Most, hogy a T41 megjelent a Linux színen, vajon az IBM operációs rendszer nélküli változatot is elad majd az EmperorLinuxnak, hogy a Linux vásárlóknak ne kelljen kifizetniük az öröklött OS díját? Talán ha pár percre befejezik a Groklaw olvasását és megegyeznek, jövőre kicsit szerencsésebbek leszünk.

Linux Journal 2004. augusztus, 124. szám

Linux Journal Csapat

Mono: gépfüggetlen hálózati alkalmazások

Érdekel a .NET? Próbáld ki ezt az érdekes mintaalkalmazást amely a Mono GUI és XML-RPC képességeit aknázza ki.

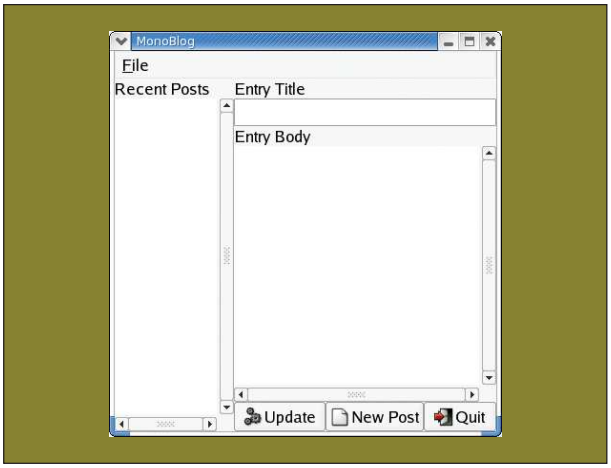
A Mono nem más mint a Microsoft .NET fejlesztői keretrendszerének Ximian által készített, nyílt forrású változata. A .NET több különböző technológiát foglal össze: fordítókat több különféle nyelvhez (többek közt a Microsoft új programnyelvéhez a C#-hoz) amelyek gépfüggetlen bájtkódot készítenek; a ***Common Language Runtime*** (CLR) nevezetű virtuális gépet, amely értelmezi a bájtkódot, valamint hasznos programokkal teli osztálykönyvtárakat a fájlkezelési szolgáltatásoktól kezdve a GUI készítésig és kezelésig. A Mono megvalósítás tartalmazza a Linux, BSD-alapú rendszerekez (ideértve a Mac OS X rendszert) és Windows alatt működőképes CLR-t, valamint fordítókat a C# és Basic nyelvekhez. A Mono fejlesztés alatt áll, és a .NET osztálykönyvtár sok része még nem készült el, különösen igaz ez a Windows.Forms csoport esetében amely a Windows GUI-val dolgozó osztályokat tartalmazza. Ugyanakkor a Mono fejlesztői kiadtak egy csomagot a GTK felhasználói felület eszközkészletéhez, így 100% .NET kompatibilitás nélkül is tudunk gépfüggetlen grafikus alkalmazásokat fejleszteni. Cikkünkben bemutatjuk, hogyan lehet a C#, Mono és a Linux hármasával olyan hasznos programot készíteni mint a MonoBlog, amely bármely rendszeren képes futni ahol a Mono és a GTK megtalálható. Valamennyi tájékozottságot feltételezünk a Glade és a C# terén, persze csak teljesen alapszinten. A hálózati források közt hasznos útmutatókat találunk.

Mono beszerzése

A Mono weblap útmutatójából megtudhatjuk, hogyan telepítsük Linux, Mac OS X és Windows rendszerek alá (lásd a forrásokat). Két további C# könyvtárra is szükségünk lesz, nevezetesen a ***GTK#*** és ***XmlRpcCS*** könyvtárakra. A MonoBlogot futtató rendszeren fenn kell lenniük az alap GTK könyvtáraknak, melyek egyébként a legtöbb Linux rendszeren megtalálhatók. Windows és Mac OS X rendszerek alatt azonban valószínűleg telepítenünk kell, a csomagokat a GTK weblapján találjuk (lásd a források között). A könyvtárak telepítésére vonatkozó útmutatókat a megfelelő honlapokon találjuk.

MonoBlog, a weblogszerkesztő

A MonoBlog egy weblogszerkesztő program, amely új küldeményeket tud elhelyezni és régieket tud szerkeszteni a naplóban, továbbá lehetővé teszi a felhasználónak, hogy beállításait megváltoztassa. A legtöbb weblog-rendszeren megtaláljuk a MetaWeblog API-ként ismert közös funkciókészletet. A MonoBlog ennek segítségével kommunikál a különféle weblog programokkal, és nem készít külön-külön háttérprogramot a ***Movable Type***, ***LiveJournal*** vagy ***Radio Userland*** rendszerekhez. A példa teljes C# forráskódját a ***Linux Journal*** FTP oldaláról tölthetjük le (***lásd a forrásokat***).



1. ábra A főablak

Az *1.* és *2. ábra* a MonoBlog kezelőfelületét mutatja be, amelyet Linux alatt Glade-del készítettünk. Az *1. ábra* főablakába a weblog címének és tartalmának felviteléhez használható szöveges mezőket, valamint a weblog frissítésére, az űrlap törlésére és a kilépésre szolgáló gombokat találjuk. A baloldali fehér rész a GTKTreeView vezérlőelem, amely a régebbi küldeményeket jeleníti meg, és ahol a felhasználó kiválaszthatja, melyiket szeretné frissíteni. A *2. ábrán* látható ablakban a felhasználó a MonoBlog és a saját weblogja közötti párbeszédhez szükséges adatokat adhatja meg.

GUI készítés libglade-del

A GTK egyik hasznos szolgáltatása a ***libglade*** könyvtár, amely lehetővé teszi, hogy a Glade által készített XML állományból készítsük el a program GUI felületét, a grafikus

2. ábra A beállítások ablak

3. ábra MonoBlog Red Hat 9 környezetben

elemek (widgets) kinézetét közvetlenül a kódban adva meg. A GTK# változat is tartalmazza ezt a képességet, így a GUI elkészítése nagyon egyszerű. A MonoBlog indításakor a `using` paranccsal importáljuk a `GTK` és `Glade` névtereket, majd a konstruktorban megadjuk a következőket:

```
Application.Init();

Glade.XML gxml = new Glade.XML("monoblog.glade",
                               ➥null, null);
gxml.Autoconnect(this);

Application.Run();
```

Az Application osztály meghívása minden `GTK` programban kötelező. Az `Application.Init()` a GTK-t állítja alaphelyzetbe, majd az `Application.Run()` átadja vezérlést a GTK főciklusnak, amely figyeli az eseményeket és üzenetet küld a jelzések (***signals***) bekövetkezésekor. A szabványos `Glade.XML` konstruktor három paramétert vár: a Glade-fájl nevét tartalmazó szöveget, egy olyan szöveget, amely megmutatja az objektumnak, hogy a Glade fában melyik csomóponton kezdje el építeni a felületet, végül, a harmadik szövegben a kérdéses Glade fájlhoz tartozó fordítási tartományt adhatjuk meg.
A MonoBlog-nak az ***Xml*** állományban található valamennyi csomópontot el kell tudnia érni, azaz a főablakot és a *„**preferences**”* ablakot is. Fordításra nincs szükség, így a második és harmadik paraméter üres érték marad. Az `Autoconnect()` tagfüggvény (method) a paraméterként megkapott objektumhoz fűzi a ***Glade*** állományban megadott jelzéskezelőket (***signal handlers***) és objektumokat, így az objektum válaszolni tud az eseményekre és kezelheti a grafikus elemeket. Mivel a MonoBlog aprócska program, valamennyi jelkezelést a főablakban végeztem el. Egy nagyobb, összetettebb rendszerben, általában jobb megoldás a jelkezelést másik osztályba sorolni.
A grafikus elemek eléréséhez különleges megadási mód szükséges. Mindegyiket példányváltozóként (***instance variable***) kell megadnunk:

```
[Glade.Widget] GTKWidgetType widgetname;
```

ahol a `GTKWidgetType` helyére az éppen aktuális objektumtípust írjuk, és a `widgetname` pedig a Glade fájlban megadott megfelelő elemnév. Az `Autoconnect()` visszatérése után az elemeket pontosan ugyanúgy használhatjuk, mintha maga program hozta volna létre őket.

Régi bejegyzések lekérése

A program betöltése során első lépésként lekérdezi a weblogot és letölti a friss küldeményeket, ezután pedig megjeleníti őket a TreeView elemben. Mindezt a MonoBlog osztály `getRecentPosts()` tagfüggvénye kezeli; a fő konstruktor hívja meg, feltéve, hogy a beállítások már megvannak, hiszen a tagfüggvénynek tudnia kell milyen webloghoz akar csatlakozni. A MetaWeblog API egyik függvényhívása, a `metaweblog.getRecentPosts`, megadott számú régi küldeményt ad vissza, illetve, annyit amennyit talál, ha többet kértünk le mint amennyi létezik.
A webloggal folytatott párbeszéd nagyon egyszerű:

```
XmlRpcRequest client = new XmlRpcRequest();
client.MethodName = "metaWeblog.getRecentPosts";
client.Params.Add(BlogID);
client.Params.Add(ServerUser);
client.Params.Add(ServerPass);
client.Params.Add(NumberOfPosts);
XmlRpcResponse response = client.Send(ServerURL);
```

Új `XmlRpcRequest` objektum létrehozásához csak a kiválasztott API függvény nevét kell megadnunk, majd feltöltenünk a szükséges paramétereket és elküldeni a weblognak. A weblog visszaküldi a válaszát, jelen esetben a küldemények tömbjét, amit a `XmlRpcResponse` objektum Value mezőjében tárolódik. Ezután a `GTKTreeView` vezérlőelemet kell frissítenünk.
A GTK 2.0 és újabb rendszer esetében a vezérlő modell-nézet-vezérlő (***model-view-controller***) megközelítést alkalmaz. Itt létrehozzuk az új objektummodellt, és átadjuk a vezérlőnek:

```
System.Type[] ListTypes = new System.Type[3];
ListTypes[0] = typeof(string);
ListTypes[1] = typeof(string);
ListTypes[2] = typeof(string);
ListStore store = new ListStore(ListTypes);
treeview1.Model = store;
```

Ez az objektummodell egy háromoszlopos táblázatot hoz létre. A `ListStore` objektumnak `Type` objektumok tömbjét

kell átadnunk; a tömb minden egyes eleme megfelel az adott oszlop típusának. A weblog-küldemények három elemet tartalmaznak – a címet, a tartalmat és egy egyedi azonosítót. Minthogy mindhárom elem szöveg, az összes oszlopnak `String` típust adunk meg. A tagfüggvény további része végiglépdel a tömbön és feltölti a modellt:

```
TreeIter iter = new TreeIter ();
foreach (Hashtable post in results) {
  String title = (String) post ["title"];
  String postid = (String) post ["postid"];
  String description = (String) post
                       ➥["description"];

  store.Append (out iter);
  store.SetValue (iter, 0, new GLib.Value(title));
  store.SetValue (iter, 1, new
                  ➥GLib.Value(postid));
  store.SetValue (iter, 2,
  new GLib.Value(description));
}
```

Mindez önmagában még nem elegendő ahhoz, hogy a címeket is megjelenítsük a fában. Ezért egy kis kódot szúrunk a konstruktorba, a `getRecentPosts()` meghívása után:

```
TreeViewColumn TitleCol = new TreeViewColumn();
CellRenderer TitleRenderer = new
➥CellRendererText();
TitleCol.AddAttribute (TitleRenderer, "text", 0);
treeview1.AppendColumn (TitleCol);
```

Ezzel az új oszlopnézetet hozzáadtuk a fához. Az `AddAtrribute()` függvény a modell első oszlopához (`title`) kapcsolódik a 0 paraméterrel. A felhasználónak csak a bejegyzések címét kell látnia a `TreeView` vezérlőelemben; több oszlopnézetre nincs szükségünk. Az információt azonban a modellben tároljuk, így programunk hatékonyabban működik.

Régi küldemények szerkesztése

Amikor a felhasználó egy bejegyzésre kattint, azt szeretnénk ha a program a régi bejegyzést megjelenítené az ablak jobb oldalán található szöveges mezőben. A MetaWeblog API-ban találunk egy `metaweblog.getPost` nevű függvényt, amely a küldeményeket kéri le weblogból. Mivel azonban mi már korábban letöltöttük őket a `getRecentPosts()` függvénnyel, a program a modellből is lekérheti az adatokat, nem kell ismét a webloggal társalognia. A Glade segítségével a `row_activated` jelzést összekapcsoltuk a `SelectOldPost` függvénnyel, így amikor az elemre duplán kattintunk, a következő kód fut le:

```
public void SelectOldPost(System.Object obj,
                       ➥EventArgs e) {
  TreeSelection currentSelection =
                       ➥treeview1.Selection;
  TreeIter iter;
  TreeModel model = treeview1.Model;
  currentSelection.GetSelected (out model,
                              ➥out iter);
  String selected = (string) model.GetValue
                              ➥(iter,1);
```

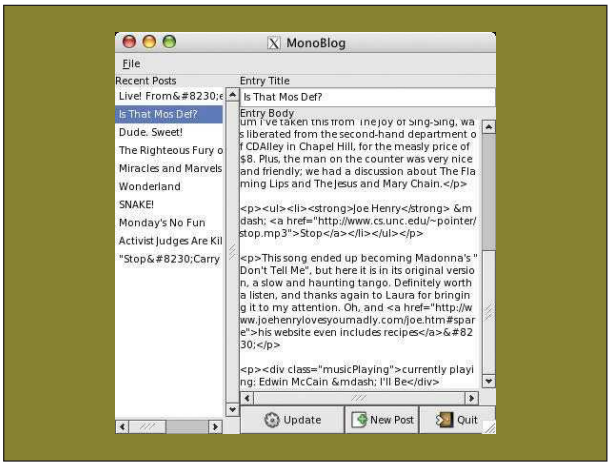
4. ábra MonoBlog Windows XP környezetben

```
  String oldTitle = (string)
                  ➥model.GetValue(iter,0);
  String oldEntry = (string)
                  ➥model.GetValue(iter,2);

  TextBuffer buffer = textview1.Buffer;
  entry1.Text = oldTitle;
  buffer.SetText(oldEntry);

  OldPostID = selected;
  EditingOldPost = true;
}
```

A függvény lekéri a `GTKTreeView` vezérlőben aktuálisan kiválasztott elemet, majd közelítő (*iterator*) segítségével címzi meg a modellt és keresi ki a szükséges értéket. Ezután kitölti a szöveges mezőket a kapott információkkal, majd frissít két példányváltozót amelyekre akkor lesz szükségünk ha a felhasználó a ***Post*** gombra kattint. Ha a program új bejegyzés készítése helyett egy régit szerkeszt, másik MetaWeblog API hívást kell kiadnunk, amihez szükségünk lesz a küldemény egyedi azonosítójára. Ezért kell beállítanunk a `OldPostID` és `EditingOldPost` változókat.

A weblog frissítése

Az ***Update*** gomb `clicked` jelzését az `OnUpdateClicked` tagfüggvényhez rendeltük. Ez a függvény sajnos túl hosszú, így nem tudjuk teljes egészében bemutatni a cikkben, de működése nem annyira bonyolult. Először is lekéri a szöveget a két szöveges mezőből és elkészíti a küldemény hash-tábla megfelelőjét; erre a MetaWeblog API híváshoz lesz szükségünk. Attól függően, hogy az `EditingOldPost` jelet beállítottuk-e, a függvény `metaweblog.newPost` vagy `metaweblog.editPost` hívással XML-RPC kérelmet küld a weblognak. Amikor a weblog sikeres választ ad vissza, jelezve, hogy a frissítés megtörtént, a függvény tisztítja a mezőket és lehetővé teszi a felhasználónak, hogy új bejegyzést készítsen. A főablak további gombjai, a ***New Post*** és a ***Quit*** rövid programocskát tartalmaznak. Akárcsak a ***Post*** gomb esetében itt is a clicked jelzéseket csatoltuk a MonoBloghoz. A New Post a szöveges mezőket törlő és az `EditingOldPost` jelet hamisra állító függ-



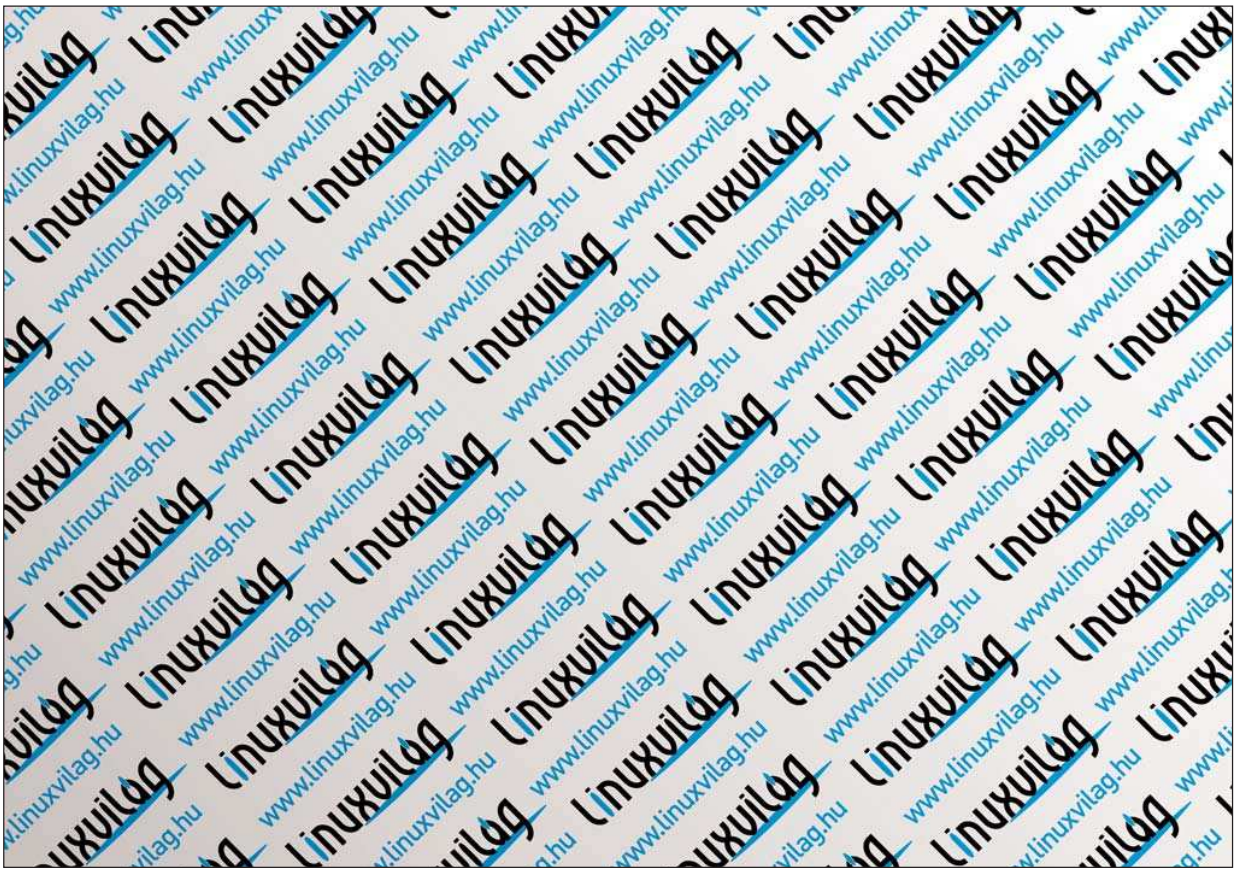
5. ábra MonoBlog Mac OS X környezetben

vényhez kapcsoltuk, így a felhasználó újrakezdheti a munkát. A Quit, ahogy annak lennie kell, az `Application.Exit()` GTK hívás segítségével kilép a MonoBlogból.

Beállítások (preferences)

A .NET osztálykönyvtár osztályokat is tartalmaz, amelyekkel XML állományokból olvashatjuk be beállításainkat. Az *1. listában* a MonoBlog beállításállományára láthatunk példát. Ezeket az értékeket az alább bemutatott `getConfig()` függvény olvassa be:

```
private bool getConfig {
  try {
  AppSettingsReader config = new
              ➥AppSettingsReader();

  ServerURL = (string)
              ➥config.GetValue("ServerURL",
              ➥typeof(string));

  ServerUser = (string)
              ➥config.GetValue("ServerUser",
              ➥typeof(string));

  ServerPass = (string)
              ➥config.GetValue("ServerPass",
              ➥typeof(string));

  BlogID = (string) config.GetValue("BlogID",
              ➥typeof(string));

  NumberOfPosts =  (string)
  config.GetValue("NumberOfPosts",
              ➥typeof(string));

  catch(Exception problem) {
    return false;
    }
  return true;
}
```



Alapértelmezés szerint az `AppSettingsReader` objektum *`programfájl-neve`*`.config` nevű állományt keres, ami jelen esetben `monoblog.exe.config` lesz. Ezután a `GetValue()` függvény segítségével kaphatjuk meg a szükséges beállítás-értékeket. A MonoBlog ezt a függvényt a konstruktorában hívja meg, még mielőtt megpróbálná letölteni a régi küldeményeket a weblogról, így megkapja a szükséges információkat. Ha az állomány nem létezik vagy az adatok betöltése nem sikerül, a függvény hamis értéket ad vissza.
A konstruktor csak akkor hívja meg a `getRecentPosts()` függvényt ha a visszatérési érték igaz, így biztosan nem használunk fel sérült értékeket.
A beállítások frissítése már kicsit nehezebb feladat. Először is a főablak menüsorában található ***Preferences*** pontot a ***Glade Menu Editor***-ral az `OnPrefsActivate` függvényhez kötöttük. Ez a *2. ábrán* látható párbeszédablakot hozza fel majd a mezőket kitölti az aktuális értékekkel, ha van ilyen. Amikor a felhasználó a párbeszéd OK gombjára kattint, a MonoBlog frissíti a változókat, és a friss adatokat visszaírja a beállításállományba. Sajnos a .NET osztálykönyvtár nem rendelkezik beállításfájl frissítő osztállyal. Minthogy a jelen beállítás nem túl bonyolult, készítettem egy `saveConfig()` nevű függvényt, amely megnyitja az alapértelmezett beállításfájlt és a frissített információkat `Write()` parancsok sorozatával visszaírja a lemezre. Ezt valami kifinomultabb megoldással kellene helyettesíteni amely felépíti a helyes XML dokumentumot, de ennél az alkalmazásnál egyszerűbb volt körülményeskedés nélkül kiírni az értékeket.

Hibakezelés

Minthogy a MonoBlog az interneten dolgozik, ahol a programunktól függetlenül mindenféle gondok adódhatnak (hálózati hibák, névkiszolgáló problémák és így tovább), valamilyen alapvető hibakezelési megoldásra is szüksége van. A `getRecentPosts()` és `OnUpdateClicked()` tagfüggvények `try...catch` blokkba kerültek. Végrehajtjuk az internetet használó kódot, és ha valamilyen problémába ütközünk a következő `catch` blokk fut le:

```
catch(Exception problem) {
  MessageDialog md =
  new MessageDialog(MonoBlogWindow,
  DialogFlags.DestroyWithParent,
  MessageType.Error,
  ButtonsType.Close,
  problem.ToString());

  md.Run();
  md.Destroy();
}
```

következményképpen a képernyőn a problémát ismertető hibaüzenete jelenik meg, szöveges üzeneteként átadva a `Mono CLR`-től kapott üzentet. Így a felhasználó folytathatja a munkát és lehetőség van javítani a problémán. Ugyanakkor jelenleg a `Mono CLR PPC` ágán a hibakezelés nemigen működik így ha a program Mac OS X rendszeren fut, a hibakezelés nem fog működni és a program a csendben nem csinál semmit. Folyik a munka a PPC átiraton, szóval, mire ez cikk a nyomdába kerül, ez a probléma lehet, hogy már a múlté.

1. lista XML beállításfájl minta

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<appSettings>
<add key="ServerURL"
value="http://www.test.com/mt-xmlrpc.cgi"/>
<add key="ServerUser" value="example"/>
<add key="ServerPass" value="password"/>
<add key="BlogID" value="1"/>
<add key="NumberOfPosts" value="10"/>
</appSettings>
</configuration>
```

Fordítás és futtatás

A C# programok fordítását az `mcs` fordítóval végezzük. A MonoBlogot a következő paranccsal fordítottuk:

```
mcs -r gtk-sharp.dll -r glade-sharp.dll
➥-r XmlRpcCS.dll -r glib-sharp.dll monoblog.cs
```

A `-r` kapcsoló mutatja milyen háttérre van szüksége a programnak; itt egyszerűen csak azt kell megmutatnunk, milyen könyvtárakat használ a MonoBlog. Eredményképpen egy `monoblog.exe` nevű lefordított bájtkódot kapunk. A program futtatásához a Mono CLR-nek ezt a fájlt kell paraméterként megadnunk:

```
mono monoblog.exe
```

Kifejlesztettünk egy programot Linux alatt amit szinte gond nélkül tudunk Windows vagy Mac OS X környezetben futtatni. Egyszerűen másoljuk a ***monoblog.exe, monoblog.exe.config*** és ***monoblog.glade*** állományokat a másik rendszerre és futtassuk le az imént bemutatott módon a Mono CLR programmal. A *3., 4.* és *5. ábra* a MonoBlogot mutatja működés közben Linux, Windows és Mac OS X gépeken. Semmilyen kódot nem kell megváltoztatnunk; a program úgy működik ahogy van, feltéve, hogy a futásához szükséges könyvtárak elérhetők.

Összefoglalás

Remélhetőleg cikkünkben be tudtuk mutatni, hogyan lehet a Mono és a C# segítségével könnyen és gyorsan gépfüggetlen alkalmazásokat készíteni. Fejleszthetünk az egyik rendszeren és biztosak lehetünk benne, hogy a program bármely rendszeren futni fog, amely ismeri a Mono és GTK könyvtárakat. Maga a MonoBlog program megérett a további kísérletezésre. Néhány lehetséges fejlesztési irány lehet a további formázási lehetőségek, részletesebb hibajelentések kidolgozása, GtkHTML# kötésekkel készíthetnénk HTML előnézet ablakot, valamint a MetaWeblog API egyéb lehetőségeit is kihasználhatnánk, például a weblog küldeményeinek törlését is felvehetnénk lehetőségek sorába.

Linux Journal 2004. július, 123. szám

Ian Pointer munkanélküli végzős számítástechnikus az Egyesült Királyságban. A ian@snappishproductions.com címen érhető el.

Windows-fejlesztés Linux alatt

Fordítsuk és teszteljük a projekt Linux és Microsoft Windows változatát újraindítás nélkül! A MinGW és a Wine ingyenes eszközökkel a Win32 felületet akár több felületen működő API-nak is nevezhetjük.

A legtöbb ***Linux Journal*** olvasóhoz hasonlóan én is elvakult rajongója vagyok minden Linux, GNU és nyílt forráskódú dolognak. Saját gépeimen Linuxot használok, ezeken programozom, játszom és térítek másokat, amikor csak lehet. A létező programozási munkák nagy része viszont olyan alkalmazások írását jelenti, amelyek a Washington állam-béli Redmondból származó operációs rendszeren futnak. A munkám révén kénytelen voltam néhány kisebb alkalmazást írni Microsoft Windows felületre. Mivel a futás sebessége lényeges volt, C-ben akartam megírni a programokat, közvetlenül használva a Win32 alkalmazásprogramozói felületet. Felmerült bennem, hogy ha olyan szabványos nyelvet használok, mint a C, akkor fejleszthetek a szép és kényelmes Linux birodalmamban.
Ez a cikk egy rövid útmutató ahhoz, hogyan lehet windowsos alkalmazást fejleszteni linuxos környezetben. Rövid bevezetőt adok a windowsos programozásból, majd lépésről lépésre végigmegyünk egy példaprogram fordításán és tesztelésén. Szó lesz még a Wine-ról, hogy hogyan lehet a segítségével leegyszerűsíteni a Windows forráskód átemelését a Linuxba.

Win32 programozás

Azoknak, akik a UNIX-féle folyamat-elvonatkoztatás egészséges táplálékán nőttünk fel, a Windows-modell egyenesen eretneknek tűnhet. A Windows-modell egy időosztásos, többfeladatos, többszálú, üzenetátadásos operációs rendszer. Most csak az NT-re és a leszármazottaira, az 2000-re és az XP-re szorítkozom. Az operációs rendszer minden folyamatot egy szálnak tekint. Ez a folyamat-összefüggést valamivel könnyebbé teszi, mint a hagyományos nehézsúlyú folyamat-modell, amely a UNIX-hoz hasonló operációs rendszerekben használatos. Ennek a „minden egy szál" modellnek a következtében viszont minden egy globális memóriacímtérben van. A megfelelő jogosultságokkal és a megfelelő cím birtokában az egyik program piszkálhatja a másik program bitjeit. A másik következmény, hogy a mag által létrehozott adatszerkezetek nem rögzített memóriacímeken vannak. Ez azt jelenti, hogy a felhasználó programnak kell lefoglalni a kapcsolódó memóriát, mielőtt bármilyen globális adatszerkezetet használna, például grafikus környezetet. Arról sem szabad megfeledkezni, hogy felszabadítsuk ezeket a szerkezeteket, miután végeztünk, különben memóriatöredezettséget okozhatnak. Az *1. lista*, amely a Linux Journal FTP oldalán elérhető, egy alap „Hello World" program. Nagyrészt szabványos, csak a `switch` utasításon belüli rész érdekes igazán. Egy alap programhoz képest elég soknak tűnik a kód, de épp ez a baj az alacsony szintű API használatával. Jó összehasonlítás lenne erre a Linuxban, ha az Xt segítségével akarnánk kódot írni az X-hez.
A `main()` függvény helyett a Windows GUI (grafikus felhasználói felület) alatt futó program `WinMain()`-nel kezdődik. Ehhez a függvényhez hozzátartozik, hogy a program végzi el az egész kezdeti értékadást. Ennek a kezdeti értékadásnak része a windowosztály megadása a főablakhoz, és egy visszahívandó (callback) függvény hozzárendelése. Ezután hozzuk létre a főablakot és jelenítsük meg az asztalon. Ekkor a vezérlés az üzenetkezelő ciklushoz kerül, és a visszahívandó függvény feldolgozza a főablakhoz küldött üzeneteket.
A ➲ http://www.winprog.org-on elérhető egy jó és gyors bevezető a Windows programok írásába.
A webhely szerkesztői egy jó *faq*-t és egy egész jó, az összes alapkérdést átfogó segédletet kínálnak. Természetesen, a Windows programozás bibliája a vastag „***Windows Programming***" könyv, ***Charles Petzold***-tól. Ha ebben a kötetben nem találjuk, amire szükség van, még mindig elég tekintélyes méretű ahhoz, hogy kiverjük vele az információt a legközelebbi Windows-guruból.

Keresztfordítás

A GCC-ben az az egyik bámulatos dolog, hogy annyi különböző felületre és operációs rendszerre átültették már. Ennek nagy előnye, hogy olyan binárisokat fordíthatunk egy felületen, amelyek teljesen más felületen futnak. Én rendszeresen fordítok Solaris vagy Windows binárisokat a linuxos hordozható gépemen. Ez hihetetlen előny, mert lehetővé teszi, hogy a fejlesztés egy ismerős, kényelmes környezetben történjen.
A legtisztább módon úgy kezdhetünk hozzá, hogy visszamegyünk a forráshoz (lásd források). Így pontosan olyan beállításokkal és olyan felületen fordíthatjuk a kódot, ahogy szeretnénk. Szerencsére, ezt a munkát már elvégezték he-

lyettünk. A MinGW Project kedves munkatársai fenntartanak egy GCC-változatot windowsos binárisok fordítására. Ez tartalmazza az összes kapcsolódó állományt, például a fejállományokat (headers). Itt megtalálhatók a források, a bináris archívumokkal együtt. A programokat RPM-alapú és deb-alapú terjesztések számára is becsomagolták. Ha Debiant használunk, az `apt-get` segítségével letölthetjük a ***mingw32*** vagy a ***mingw32-runtime*** csomagot. Ha testing vagy unstable (Sarge, SID) változaton dolgozunk, akkor a ***mingw32-binutils***-t is le kell tölteni.
A GCC legtöbb fordítási lehetősége megtalálható itt a ***MinGW***-ben, néhány ráadással. Ha egyszerűen csak lefordítunk egy programot, különleges lehetőségek nélkül, akkor futtatható a konzolról, például ha egy kicsi, egyszerű programot szeretnénk írni, amelyikhez nem kell GUI. Mivel ez Windows és GUI programot akarunk, felkészítjük a kódunkat a fent leírtak szerint, és a fordítási parancshoz hozzáadjuk a `-mwindows` kapcsolót. Ez létrehozza egy szabvány Windows végrehajtható állomány fordításához szükséges makrókat és könyvtári hivatkozásokat. Ha úgy döntünk, hogy bonyolultabb Windows programot írunk, amely más Windows-szolgáltatásokat is használ, akkor pontosan meg kell adni a könyvtárakban a szükséges hivatkozásokat. A Windowsban meghatározhatjuk a program erőforrásait, például menüket, bitképeket, szöveget és más elemeket. Ezeket az erőforrásokat egy külön állományban tároljuk és külön le kell fordítani, mielőtt a végrehajtandó állományhoz kapcsoljuk. Ez a feladat a `mingw-windres` programra marad, amely létrehoz egy objektumállományt, amit később a végrehajtható állományhoz kapcsolhatunk. Az 1. listában látható egyszerű példaprogram fordításához a következő parancsot használjuk:

```
mingw-gcc -o example1.exe example.c -mwindows
```

Cseréljük ki a `mingw-gcc` parancsot bármire, ami az adott csomaghoz jó, mint ***Compiler executable***. Íme, van egy életképes Windows programunk. Ilyen egyszerű!

Hibakeresés a Winenal

A Wine a másik nagy adomány azoknak a fejlesztőknek, akiknek Windows-felületen kell programokat írniuk. A fejlesztők elképesztő mennyiségű munkát fektettek a Wine-ba. Ez a nagyszerű program lehetővé teszi, hogy windowsos programokat futtassunk Linux alatt. Ennek az az eredménye, hogy a frissen fordított programunkat lefuttathatjuk, és megnézhetjük, hogy valóban működik-e, ahogy beharangoztuk. Ehhez adjuk ki a `wine example1.exe` parancsot, és látnunk kell az ablakot megjelenni az asztalon. A Wine indításakor megvan a lehetőségünk, hogy az ablakokat a saját ablakkezelő felügyelje, és ne a saját asztalukon jelenjenek meg. Ez befolyásolja azt, hogy néz ki az ablak futtatáskor. Ha nem voltunk elég ügyesek ahhoz, hogy hibátlanul gépeljük be a programot, akkor hibakeresésre lesz szükség, hogy kiderüljön, mi nem jó. A Wine ebben nagy segítség lehet.
A `-debugmsg [debugchannel]` lehetőség a Wine egy vagy több hibakereső csatornájának eredményét adja kimenetként. Példa a lehetséges hibakereső csatornákra:

- `relay`: naplóüzenetet ír, akárhányszor egy Win32 függvény meghívódik.
- `win`: nyomon követi a Windows-üzeneteket.
- `all`: nyomon követi az összes üzenetet.

Az `all` lehetőséget ne használjuk, csak ha valóban szükség van rá, mert a kimenet mennyisége könnyen kifoghat legrészletmániásabb programozón is. A Wine oldalon megtalálható a lehetséges hibakereső csatornák teljes felsorolása.

Eredeti változat fordítása Linuxra

Most van egy csodálatos, működő, hibamentes programunk, amely Windows alatt fut. Tekintve, hogy az egész munkát Linux alatt végeztük, nem lenne nagyszerű, ha a program Linux alatt is futna? A Wine Project kedves munkatársai ismét a segítségünkre sietnek. A projekt egyik része tartalmazza a ***winelib*** könyvtárat, amely a Windows forráskód számára biztosítja a linuxos illesztőfelületet. Hogy használhassuk ezt a szolgáltatást, telepíteni kell a ***wine-devel*** csomagot a terjesztéshez. Ha forrásból telepítettünk, a szükséges fájloknak már hozzáférhetőnek kell lenniük.
A ***wine-devel*** csomag egy `winemaker` nevű Perl parancsállományt tartalmaz, amelynek az a feladata, hogy végigmenjen a forrásfájlokon és könyvtárakon, és végrehajtsa a Linux ***winelib*** könyvtárában történő helyes fordításhoz szükséges változtatásokat. Olyan dolgokat ellenőriz, mint az állománynevek nagy- vagy kisbetűs írásmódja és a sorvég karakterek. Ráadásként a fájlok elérési útjában a fordított perjeleket kicseréli hagyományos perjelekre, és sok más dolgot elvégez. Alaphelyzetben biztonsági másolatot készít azokról az állományokról, amelyeket meg kell változtatnia. Átalakítja a projektet winelibbé, miden változtatást magától elvégezve. A fordításhoz egyszerűen futtassuk a következő parancsokat, ami által linuxos végrehajtható állományt hozunk létre.

```
winemaker .
./configure --with-wine=/elérési/út/wine
make
```

A fenti pont szándékosan van ott. Megadjuk az elérési utat, amelyben a `winemaker` megtalálja a vizsgálandó forrásfájlokat. Itt a fájlok az aktuális könyvtárban vannak.
A mi esetünkben a példában nincsenek projektfájlok, és ezt a `winemaker` problémaként érzékeli. A szükséges lépéseket egyszerűen elvégezhetjük kézzel. A program fordítására a `mingw-gcc` végrehajtása helyett `winegcc`-t használjuk, pontosan ugyanazokkal az értékekkel. Ez létrehoz egy `.so` végződésű állományt, és egy héjprogramot, amely a program futását felügyeli. Most már lefordítottuk a Windows forráskódot, és működik Linux alatt.

Következtetés

Remélem, sikerült meggyőznöm néhány windowsos fejlesztőt arról, hogy hatékonyan dolgozhatnak Linux alól. A GCC-vel, ami lehetővé teszi windowsos végrehajtható állományok fordítását és a Wine-nal, amely sok támogatást nyújt a futtatáshoz és a hibakereséshez, legtöbbször nincs oka, hogy egyáltalán elindítsuk a Windowst. Az egyetlen ok az lehet, ha a kedvenc fejlesztőkörnyezetünk nem fut rendesen Wine alatt. Szerencsére egyre több rendszer fut hibátlanul Wine alatt is.

Linux Journal 2004. július, 123. szám

Joey Bernard

A DMA használata

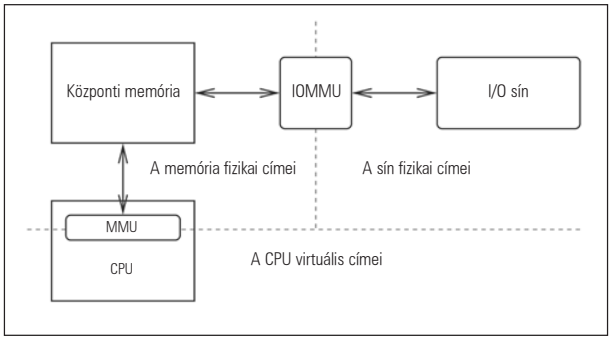
DMA segítségével az eszközök a processzor terhelése nélkül tudják írni és olvasni a memóriát, amivel felgyorsul a be- és kiviteli műveletek végrehajtása. Tekintsük át, hogy a rendszermag hogyan követi figyelemmel a processzor háta mögött történő eseményeket.

A DMA (***direct memory access***) a közvetlen memóriahozzáférés rövidítése, és segítségével az eszközök, rendszerelemek képessé válnak a központi memória tartalmának módosítására. Mindezt úgy, hogy az adatoknak nem kell áthaladniuk a processzoron. A DMA használatának előnye pontosan a processzor munkájának zavartalanságából fakad. A rendszer kérheti az adatok elhelyezését adott memóriaterületen, és a művelet befejezéséig nyugodtan foglalkozhat egyéb teendőivel. A DMA használatával kapcsolatos bajok nagy része ugyanakkor éppen abból ered, hogy a processzort kihagyjuk a játékból.

Ezek a bajok három csoportra oszthatók. Az első oka, hogy a processzor jó eséllyel memóriakezelő egységgel rendelkezik. Az a cím, amelyet a processzor a memória adott területének címzésére használ, nem feltétlenül egyezik meg a terület fizikai címével. Másodszor, a központi memóriába történő írások miatt előfordulhat, hogy a memória és a processzor közötti gyorstárak tartalma elavulttá válik. (Lásd: *„A gyorstárazás rejtelmei”*, ***Linuxvilág, 2004. április***)

Harmadszor: a be- és kiviteli (I/O) sínen is lehet memóriakezelő egység (IOMMU). Ebből következően előfordulhat, hogy az eszköz által az adatok továbbításakor használt síncím nem egyezik meg a fizikai memóriacímmel vagy a processzor által használt képzetes memóriacímmel. Az ilyen megoldások az x86-os világ lakói számára meglehetősen idegennek hatnak. A GART-ok (***graphical aperture remapping table*** – grafikus ablakleképező tábla) AGP sínen való használatával ugyanakkor az x86-hívők IOMMU vonatkozású ellenállása gyengülni látszik.

A Linux rendszermag DMA-kezelő API-jának mindhárom problémaforrást figyelembe kell vennie, és a gondok tényleges fellépését meg kell előznie. Mindezek mellett, mivel az eszközök felől a DMA alapú átvitelek túlnyomó része egy külső sínen zajlik, újabb három kérdés merül fel. Az első az, hogy a I/O eszköz címszélessége eltérhet a fizikai memóriacímek szélességétől. Az ISA-s eszközök például 24 bites, a 64 bites rendszerekben pedig egyes PCI sínre illeszkedő eszközök csak 32 bites címzésre képesek. A második, hogy a I/O sín vezérlője is gyorstárazhatja a kéréseket. Ilyesmire elsősorban a PCI-sínen kerülhet sor, itt az írási kéréseket a PCI-vezérlő visszatarthatja abban a reményben,



1. ábra Címtartományok DMA alatt

hogy több kérést is összegyűjtve gyorsabb átvitelt valósíthat meg. Ezt a műveletet, jelenséget nevezzük PCI-kötegelésnek. A harmadik kérdéses dolog azzal függ össze, hogy az operációs rendszer olyan területre kérhet továbbítást, amely saját képzetes memóriaterében összefüggő ugyan, ám a fizikai memóriában töredezett; ennek oka általában az, hogy az átvitel több lapot is érint. Az ilyen átvitelek lebonyolításához szétszóró/gyűjtögető (***Scatter/Gather – SG***) listákra van szükség.

Írásomban kizárólag az eszközök kezelését segítő DMA API-val foglalkozom. A 2.6-os Linux új általános eszközmodellje rendkívül elegáns lehetőséget biztosít az eszközök jellegzetességeinek leírására és – egy hierarchikus fa révén – sínjellemzőik megállapítására. Az említett csatolófelületek a 2.4 - 2.6 áttérés során fontos ellenőrzéseken és módosításokon estek át. Az itt szereplő általános elvek ugyan a 2.4-es sorozatra is érvényesek, az API és a rendszermag itt említett képességei csak a 2.6-os sorozattal jelentek meg.

SG listák

A DMA alapú átviteleknél figyelembe kell venni, hogy a felhasználó nagy mennyiségű, akár több MB-nyi adat továbbítását kérheti az adott pufferbe. A képzetes memória kezelési módja miatt előfordulhat, hogy ez a képzetes térben összefüggő terület számos, a fizikai memória különféle részeire szétszórt lapból áll. A Linux elvárja, hogy minden lapnyi méretűnél (x86-os rendszereken ez 4 KB) nagyobb

átvitel SG lista segítségével történjen. A listát normál esetben a blokkos I/O réteg (BIO layer) állítja elő. Az illesztőprogram egyik alapvető feladata a blokkos I/O réteg azon működési beállításainak megadása, amelyek alapján a I/O műveletek SG listaelemekre való felosztása történik.
Szinte minden olyan eszközt, amely nagy mennyiségű adat mozgatását végzi, úgy terveznek, hogy az ilyen átviteli kéréseket SG listák formájában tudja fogadni. Ugyan a listák pontos formátuma a legtöbb esetben eltér a rendszermag által előállítottól, az átalakítás csak kivételes esetben jár komolyabb nehézséggel.

I/O memóriakezelő egységek (IOMMU)

Az IOMMU olyan memóriakezelő egység, amely a I/O sín (vagy több szintre rendeződő sínek) és a központi memória között helyezkedik el. Működése teljesen független a processzorba épített memóriakezelő egységétől. Ha egy eszköz és a központi memória között átvitelt szeretnénk indítani, az IOMMU-t fel kell programozni az átvitel során szükséges címfordításokkal, nagyjából úgy, ahogy a processzor memóriakezelő egységével tennénk ezt. Ha ezt megtesszük, akkor a blokkos I/O réteg által létrehozott SG listák úgy adhatók meg az IOMMU-nak, hogy a sínen lógó eszköz számára a memóriaterület újfent összefüggőnek tűnjön.

GART-ok és az IOMMU kihagyása

A GART alapvetően egy egyszerű IOMMU. Egy, a fizikai memóriában található ablakból és egy laplistából áll. Feladata az ablakba eső fizikai címek leképezése a listában szereplő fizikai lapokra. Az ablak általában elég keskeny, például 128 MB méretű, így a fizikai memóriának az ablakon kívülre eső elérései nem képeződnek le.
Ez az apró hiányosság azonnal rá is világít a Linux rendszermag jelenlegi DMA-kezelésének egyik hibájára: a DMA API-k egyike sem képes hibával visszatérni, ha a memórialeképezés sikertelen. A GART-ok csak korlátozott nagyságú leképezési címtérrel rendelkeznek, és ha ez kimerül, akkor nincs több leképezésre lehetőség, legalábbis amíg néhány I/O művelet be nem fejeződik, és kellő nagyságú terület fel nem szabadul.
Egyes esetekben a GART-okhoz hasonlóan az IOMMU-kat is fel lehet programozni úgy, hogy a I/O sín és a meghatározott ablakokba eső memória között ne végezzenek címleképezést.
Ezt kihagyásos módnak nevezzük, és érdemes tudni róla, hogy nem minden IOMMU-típus támogatja. A kihagyásos módot akkor érdemes használni, ha a leképezés miatt romlik a teljesítmény, és az IOMMU-t eltávolítva az adatok útjából nagyobb átviteli sebességet lehet elérni.
A blokkos I/O réteg alapesetben feltételezi, hogy ha található IOMMU a rendszerben, akkor használjuk is azt, és ennek megfelelően számítja ki az eszközök SG listája által igényelt hely méretét. Jelenleg nincs lehetőség arra, hogy a blokkos I/O rétegnek jelezzük, az eszköz ki szeretné hagyni a folyamatokból az IOMMU-t.
Baj akkor van, ha a blokkos I/O réteg IOMMU jelenlétét feltételezi, valamint azt, hogy az SG bejegyzéseket az IOMMU állítja össze. Ilyenkor, ha az illesztőprogram az IOMMU kihagyása mellett dönt, akkor több SG bejegyzés keletkezik, mint amennyit az eszköz megenged.

Már megkezdődtek azok a munkák, amelyek e két hibaforrásnak a 2.6-os rendszermagból való eltüntetését célozzák. Az IOMMU kihagyásos hibára készült javítást már vizsgálják. A megoldás az illesztőprogramok készítői számára láthatatlan lesz, a kihagyásokról ugyanis az alapkód fog határozni. A leképezési hiányosságok elhárításához valószínűleg a leképező API-k hibajelzési képességeinek megvalósítására lesz szükség. Mivel a munka a rendszer minden DMA-illesztőprogramjára kihat, elvégzése várhatóan eléggé el fog húzódni.

Sínszélességek és DMA maszkok

Annak érdekében, hogy a lehető legnagyobb címzési szélességgel végezhesse az adatcseréket, minden általános eszköz rendelkezik egy DMA maszk nevű értékkel, amely az elérhető címvonalakat kiválasztó maszkbiteket tartalmazza, és amelynek beállítása az illesztőprogram feladata. A DMA szélességmaszk kétféle jelentést hordozhat, attól függően, hogy használunk-e IOMMU-t. Ha igen, a DMA maszk egyszerűen a leképezhető síncímek tartományát korlátozza, ám az IOMMU segítségével az eszköz a fizikai memória minden részét el tudja érni. Ha nincs IOMMU, a DMA maszk abszolút korlátot jelent az eszközre nézve. Ekkor az eszköz a fizikai memóriának a maszkon kívülre eső részeire nem tud adatokat továbbítani.
A szétszóró/gyűjtögető listák készítésekor a blokkos réteg a DMA maszk alapján dönti el, hogy az adott lapot át kell-e tükrözni. Tükrözés alatt most azt értjük, hogy a blokkos réteg vesz egy, a DMA maszkon belülre eső lapot, és minden adatot átmásol rá egy a tartományon kívülre eső lapról. Amikor a DMA használata befejeződött, a blokkos réteg visszamásolja az adatokat a tartományon kívüli lapra, majd felszabadítja a tükrözött lapot. Természetesen az oda-vissza másolgatás rontja a hatékonyságot, ezért a gyártók általában arra törekednek, hogy kiszolgáló kategóriájú gépeiknek ne legyenek DMA maszk-vonatkozású korlátaik.

A DMA és az egységesség viszonya

A DMA alapú átvitelek a processzor közreműködése nélkül folynak, tehát a rendszermagnak biztosítania kell egy olyan API-t, amely képes a processzor gyorstárainak tartalmát összhangba hozni a DMA alapú átvitel által érintett memóriaterületek tartalmával. Fontos, hogy a DMA API a rendszermag képzetes címeinek figyelembe vételével biztosítsa a processzor gyorstárainak frissítését. A gyorstáraknak a felhasználói térbeli memóriatartalom változásait követő frissítéseire egy másik API-t kell használni.

Sínkötegelés

A felső kategóriás sínvezérlő lapkák néha gyorstárral is rendelkeznek. Az alapötlet az, hogy a processzor felől a lapkakészlet felé végzett írások végrehajtása gyors, a sínen keresztül végrehajtottaké viszont lassú, vagyis ha a sínvezérlő gyorstárazza az írásokat, a processzornak nem kell megvárnia befejezésüket. A sínkötegeléssel ahogy az ilyen jellegű gyorstárazást nevezzük az a baj, hogy a processzornak nincs utasítása a sín gyorstárának kiürítésére. A gyorstár ürítése – a helyes sorrend biztosítása érdekében – szigorú szabályrendszer szerint történik. Az első szabály az, hogy csak memória alapú írásokat szabad gyorstárazni, a I/O té-

ren keresztül történőket nem. A második szabály szerint a memória alapú olvasások és írások sorrendjét szigorúan meg kell őrizni, még az írások gyorstárazásakor is. Az utolsó szabály lehetővé teszi az illesztőprogram írója számára a gyorstár kiürítését. Ha memória alapú olvasást indítunk az eszköz memóriaterületének bármely szakaszára, az összes gyorstárazott írás elvégzésére garantáltan sor kerül az olvasás megkezdése előtt.
A kötegelés kezelésében semmilyen API segítségére nem számíthatunk, az illesztőprogramok íróinak tehát maguknak kell ügyelniük a kötegelési szabályok betartására, amikor olvassák vagy írják az eszköz memóriaterületét. Egy ötlet: ha végképp nem tudunk semmilyen értelmes és szükséges olvasási műveletet kitalálni, amivel elérhetnénk a függőben lévő írások végrehajtását, egyszerűen olvassunk ki egy szakaszt az eszköz sínbeállításai közül.

A DMA API használata illesztőprogramban

Az API-ról kimerítő leírást találhatunk a rendszermag leírásait tartalmazó könyvtárban (***Documentation/DMA-API.txt***). Az általános DMA API-nak van egy párja is, amely csak PCI eszközökkel működik, ennek leírása a ***Documentation/DMA-mapping.txt*** fájlban található. A továbbiakban szeretném nagy vonalakban áttekinteni azokat a lépéseket, amelyek a DMA helyes működésének eléréséhez szükségesek. Részletesebb tájékoztatást az imént említett leírásokban lehet találni.
Az illesztőprogram elindításakor az első lépés a DMA beállítása:

```
int dma_set_mask(struct device *dev, u64 mask);
```

Itt a `dev` az általános eszköz, a `mask` pedig a beállítani kívánt maszk. A függvény igaz értékkel tér vissza, ha a maszkot a rendszer elfogadta, és hamissal, ha nem. A maszk visszautasítására akkor kerülhet sor, ha a tényleges rendszerszélesség kisebb. Egy 32 bites rendszer például elutasítja a 64 bites maszkot. Ha tehát tudjuk, hogy az eszköz 64 bites címzésre képes, először 64 bites maszkkal kell próbálkoznunk, majd ha ennek beállítása sikertelen, akkor 32 bitessel.
A következő művelet a várakozási sor lefoglalása és kezdeti értékek megadása. Ennek ismertetése meghaladja írásom kereteit, de a ***Documentation/block/*** alatt pontos leírást lehet találni róla. Ha megvan a várakozási sor, két alapvető beállítást kell megadnunk. Elsőként állítsuk be az SG tábla legnagyobb méretét (vagy engedélyezzük tetszőleges nagyságú elfogadását):

```
void blk_queue_max_hw_segments(request_queue_t *q,
                               ➥unsigned short
                               ➥max_segments);
```

Másodikként – szükség szerint – adjuk meg a legnagyobb méretet:

```
void blk_queue_max_sectors(request_queue_t *q,
                           ➥unsigned short max_sectors);
```

Az utolsó lépés a DMA maszk beprogramozása a várakozási sorba:

```
void blk_queue_bounce_limit(request_queue_t *q,
                            ➥u64 max_address);
```

Általában a `max_address` a DMA maszkkal egyenlő, ám ha IOMMU-t használunk, akkor a `max_address`-t `BLK_BOUNCE_ANY` értékre kell állítani, ezzel tilthatjuk meg a blokkos réteg számára a tükrözést.

Az eszköz működése

Egy eszköz működéséhez szükség van egy `request` függvényre (lásd a blokkos I/O leírását), az alábbi paranccsal ez fogja kivenni a várakozási sorból a kéréseket:

```
struct request *elv_next_request(request_queue_t *q);
```

A kérésben igényelt leképezési bejegyzések számát a `req->nr_phys_segments` tartalmazza. Létre kell hoznunk egy ilyen méretű ideiglenes táblát, `sizeof(struct scatterlist)` méretű egységekkel, majd ideiglenes leképezést kell megadnunk:

```
int blk_rq_map_sg(request_queue_t  *q,
                  ➥struct  request  *req,
                  ➥struct scatterlist *sglist);
```

Ezzel megkapjuk a felhasznált SG listabejegyzések számát. Az alábbi paranccsal a blokkos réteg által biztosított ideiglenes táblát kapjuk meg, ennek leképezése végül így történik meg:

```
int dma_map_sg(struct  device  *dev,
        ➥struct scatterlist *sglist, int count,
        ➥enum dma_data_direction dir);
```

Itt a `count` a visszatérési érték, az `sglist` pedig a `blk_rq_map_sg` függvénynek átadott listával egyezik meg. A visszatérési érték a kérésben igényelt SG listabejegyzések tényleges száma.
Az SG listát újra felhasználjuk és a tényleges bejegyzésekkel töltjük fel, amelyeket az eszköz SG bejegyzéseibe kell beprogramozni. A `dir` némi segítséget nyújt a gyorstárak tartalmának egységesen tartásához. Háromféle értéket vehet fel:

1. `DMA_TO_DEVICE`: Az adatátvitel a memória felől az eszköz felé történik.
2. `DMA_FROM_DEVICE`: Az eszköz a központi memóriába továbbít adatokat.
3. `DMA_BIDIRECTIONAL`: Nem tudunk semmit az adatátvitel irányáról.

Az SG lista végigjárásakor és az eszköz SG listájának feltöltésekor két makrót kell használni:

```
dma_addr_tsg_dma_address(struct scatterlist
                         ➥*sglist_entry);
```

és:

```
int sg_dma_len(struct scatterlist *sglist_entry);
```

Ezek rendre az egyes bejegyzésekhez tartozó fizikai síncímet és szegmenshosszt adják vissza. A kérés leképezését azért két lépésben végezzük, mert a blokkos I/O réteget általános kódként tervezték, és így nincs kapcsolatban a géptípustól függő, az IOMMU programozására képes alapréteggel. Az egyetlen dolog tehát, amelyet a blokkos I/O réteg ki tud számítani, az az IOMMU által a kérés miatt létrehozott SG szegmensek száma. A blokkos I/O réteg nem ismeri az IOMMU által ezekhez a szegmensekhez rendelt síncímeket, ezért egy az összes leképezendő lap fizikai címét tartalmazó listát kell átadnia. A géptípustól függő réteggel a `dma_map_sg` függvény tartja a kapcsolatot, továbbá ez végzi az IOMMU programozását és a fizikai síncímek listájának lekérését is. Ez a két tényező az, ami miatt a blokkos I/O réteg listája több elemet tartalmaz, mint a DMA API által visszaadott.

A DMA alapú átvitel befejezése után a DMA tranzakciót le kell zárni:

```
int dma_unmap_sg(struct  device *dev,
                 ➥struct scatterlist *sglist,
                 ➥int hwcount,
                 ➥enum dma_data_direction dir);
```

Itt az összes átadott érték a `dma_map_sg` hívásakor alkalmazottal egyezik, kivéve a `hwcountot`, amely a függvény visszatérési értékét tartalmazza. Végső lépésként a korábban lefoglalt SG listát fel kell szabadítani, a kérés ezzel teljesítettnek tekinthető.

DMA területen található adatok elérése

Az illesztőprogramok általában nem nyúlnak hozzá az általuk továbbított adatokhoz. Előfordulhat azonban, hogy az illesztőprogramnak meg kell vizsgálnia vagy módosítania kell az adatokat, mielőtt továbbadná őket a blokkos rétegnek. Ehhez a processzor gyorstárait összhangba kell hozni az adatokkal:

```
int dma_sync_sg(struct device *dev,
                ➥struct scatterlist *sglist,
                ➥int hwcount,
                ➥enum dma_data_direction dir);
```

Az átadott értékek a `dma_unmap_sg` hívásnál látottakkal egyeznek.

Az adatok elérésével kapcsolatosan a legfontosabb tényező az elérés időzítése. Az elérési szabályok a `dir` értékétől függenek:

- `DMA_TO_DEVICE`: Az adatok módosítása után és az eszköznek való elküldése előtt meg kell hívni az API-t.
- `DMA_FROM_DEVICE`: Az API-t az után kell meghívni, hogy az eszköz átadta az adatokat, de még az előtt, hogy az illesztőprogram megpróbálná olvasni őket.
- `DMA_BIDIRECTIONAL`: Az API-t kétszer kell meghívni, először az adatok módosításának elvégzése és az eszköznek való elküldése között, másodszor az eszköz munkájának befejezése és az adatok illesztőprogram által történő elérése között.

Egységes memóriaterület foglalása

A legtöbb eszköz postafiók jellegű memóriaterületek segítségével tartja a kapcsolatot az illesztőprogrammal. A postafiók-memóriatartományt az illesztőprogramon kívül más általában nem használhatja. A postafiók egységességének fenntartása az előbbi API segítségével eléggé nehézkes volna, ezért a rendszermag lehetőséget biztosít olyan memóriaterület foglalására, amelyre minden pillanatban garantált az eszköz és a processzor közötti adategyezés:

```
void *dma_alloc_coherent(struct device *dev,
                         ➥size_tsize,
                         ➥dma_addr_t *physaddr,
                         ➥int flag);
```

Ezzel egy `size` méretű, egységes tartomány képzetes címét kapjuk vissza, amely egyben egy fizikai síncímmel (`physaddr`) is rendelkezik az eszköz felé.

A `flag` segítségével a foglalás típusát adhatjuk meg, ez `GFP_KERNEL` (a memóriafoglalás alvó állapotba is kerülhet teljesítése előtt) vagy `GFP_ATOMIC` (a foglalás nem kerülhet alvó állapotba, sikertelenség esetén `NULL` értékkel kell visszatérni) lehet. Az API segítségével foglalt terület garantáltan folytonos a képzetes és a fizikai sínmemóriában egyaránt. Megkötés, hogy a méretnek 128 KB alatt kell lennie.

Az illesztőprogram eltávolításának részeként az egységes memóriaterületet a következő módon kell felszabadítani:

```
void dma_free_coherent(struct device *dev,
                       ➥size_tsize,
                       ➥void *virtaddr,
                       ➥dma_addr_t *physaddr);
```

Ebben az esetben `size` az egységes terület mérete, a `virtaddr` és a `physaddr` visszatérési érték pedig rendre képzetes processzor oldali és fizikai síncíme.

Összefoglalás

Írásomban villámgyors, felületes áttekintést adtam a blokkos réteg és az illesztőprogramok közötti, az eszközök programozását szolgáló SG listák előállítását célzó párbeszédről.

A DMA API számos egyéb hasznos összetevőt tartalmaz, köztük olyan API-kat, amelyek töredezéstől mentes fizikai memóriaterületeket kezelnek. Ha sikerült felkeltenem az érdeklődésedet, akkor következő lépésként a rendszermag ***Documents*** könyvtárának és forráskódjának tanulmányozását javaslom.

Linux Journal 2004. május, 121. szám

James Bottomley a SteelEye programtervezője és a nyílt forrású közösség aktív tagja. Ő felel a SCSI alrendszer, a Linux Voyager átültetés és az 53c700 illesztőprogram karbantartásáért, továbbá DMA/eszköz modellezési-elvonatkoztatási munkájával a PA-RISC Linux fejlesztésébe is bekapcsolódott.

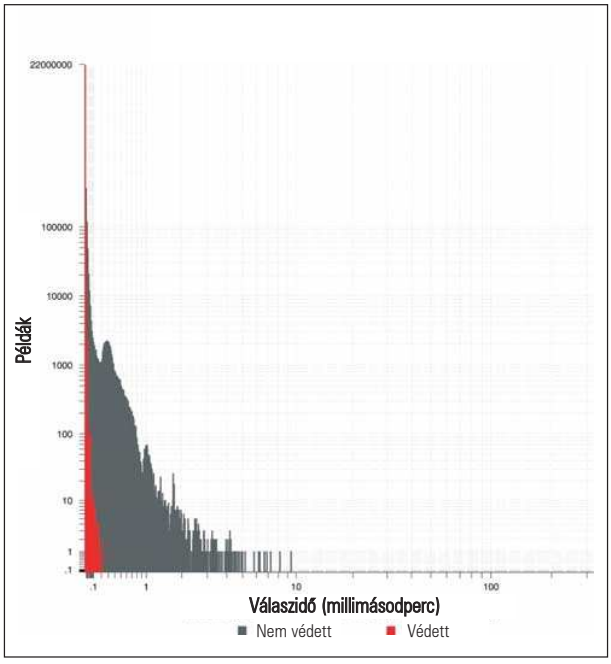
Védett processzorok: valós idejű teljesítmény szabványos Linux alatt

Tartsunk fenn egy processzort a nagy fontosságú feladatok számára, és máris magasabb szinvonalú valós idejű szolgáltatásokat biztosíthatunk.

A többprocesszoros rendszerekben a védett processzor olyan egység, amelyet a nagy fontosságú, valós idejű folyamatok futtatására tartanak fenn. A védettként kiválasztott processzor erőforrásai teljes egészében a nagy fontosságú folyamatok futtatását szolgálják. A védett processzorok futtatási környezete biztosítja a valós idejű alkalmazások üzemeltetéséhez szükséges előre jósolhatóságot.
Másként fogalmazva, ha rendelkezünk egy védett processzorral, akkor garantáltan gyors választ tudunk adni a külső megszakításokra, és előre meghatározható működésű környezetet tudunk biztosítani a valós idejű folyamatok futtatásához.

Linux és védettség

Korábban védett processzort csak szimmetrikus többprocesszoros rendszerekben lehetett kijelölni. A hiperszálas (***hyperthreading***), vagyis logikailag egynél többnek látszó processzorok megjelenésével azonban már egyprocesszoros rendszerben is kijelölhetünk védett processzort.
A védett processzoros megközelítéssel a fejlesztők a valós idejű futtatási környezetekéhez hasonló eredményeket érhetnek el.
Az eredmények összehasonlíthatóak az RTAI vagy RT/Linux környezetben mértekkel, ahol a Linux egy valós idejű futtatási környezetben önálló folyamatként kel életre.
Ha tisztán linuxos környezetet használunk az alkalmazások fejlesztéséhez, akkor jó néhány előnyt élvezhetünk az ilyen jellegű futtatási környezetekhez képest. Például a Linux számos illesztőprogramot támogat, így a teljes értékű alkalmazási megoldások megvalósításának költsége lényegesen alacsonyabb. A magas szintű programozási nyelvek széles választékával az alkalmazások elkészítése is lényegesen hatékonyabb.
Kereskedelmi alkalmazásoknál ez fontos tényező, és bár a valós idejű rendszerek tervezésénél nem feltétlenül a programozás hatékonysága a legfontosabb, a fejlesztési fázisban mégis komoly segítséget jelenthet, valamint a végső termék szolgáltatásainak bővítéséhez is hozzájárulhat.
Mindemellett a Linux összetett protokollkészletekkel – mint például a CORBA –, kiterjedt grafikai képességekkel és fejlett alkalmazásfejlesztői eszközökkel rendelkezik. A normál Linux-terjesztésekben elérhető szolgáltatásokon túl a Linux tudása – köszönhetően a mögötte álló közösség erejének – napról napra nő.
Ha az alkalmazások tervezésekor a Linuxot választjuk alapnak, a felhasználó a jövőben lényegesen több választási lehetőséggel élhet majd.



1. ábra A védett processzorral és az anélkül mért válaszidők összehasonlítása

A valós idejű működés megjósolhatóságot jelent, nem sebességet!

Valós idejűnek azt az alkalmazást nevezzük, amelynek a külvilág eseményeire kell válaszolnia, és a műveleteket adott határidőn belül kell befejeznie.
A határidő letelte után adott helyes válasz nem számít helyes válasznak. Maguk a határidők az alkalmazástól függnek,

néhány mikromásodperc és több másodperc között változhatnak. A szigorúan valós idejű alkalmazásoknál a határidőket be kell tartani. Kizárólag a rendszer által a legrosszabb esetben teljesített mutatók érdekelnek bennünket, ugyanis ezek azok az esetek, amelyekben a határidők elmulasztása előfordulhat.

Mivel a való világ eseményeiről a számítógépes rendszer megszakítások révén értesül, egy valós idejű operációs rendszernek a legrosszabb esetben is adott időn belül garantáltan válaszolnia kell a megszakításokra.

Ha ez sikerül, és át tudjuk adni a vezérlést a megfelelő valós idejű alkalmazásnak, már meg is tettük az első lépést a határidő teljesítése felé.

Ha a valós idejű alkalmazás már fut, a rendszernek előre meghatározható futási időt is garantálnia kell az alkalmazás számára. Ha a valós idejű alkalmazás futtatásának ideje túlságosan széles tartományban ingadozik, a határidőket túllépjük.

Ha a megszakításokra jó ütemben akarunk válaszolni, az operációs rendszernek képesnek kell lennie arra, hogy megszakítás fellépése esetén bármely futó programtól azonnal el tudja venni az erőforrásokat. Mivel a 2.4-es sorozatú Linux nem engedi a feladatoknak, hogy más, a rendszermagon belül futó feladatoktól elvegyék az erőforrásokat, ez a sorozat legrosszabb esetben meglehetősen gyenge válaszidőket biztosít. Létezik persze egy folt, amely lehetővé teszi a rendszermagon belül futó feladatok erőforrásainak elvételét. Sajnos ezt hiába telepítjük, maradnak olyan rejtett tényezők, amelyek a megszakításokra adott válaszok jelentős elhúzódását okozhatják.

Az operációs rendszer feladata az, hogy több, a rendszer erőforrásain osztozó feladat végrehajtását irányítsa. A megosztott erőforrások jellemzőit tartalmazó adatszerkezetek megsérülhetnek, ha egyszerre több program is használja őket.

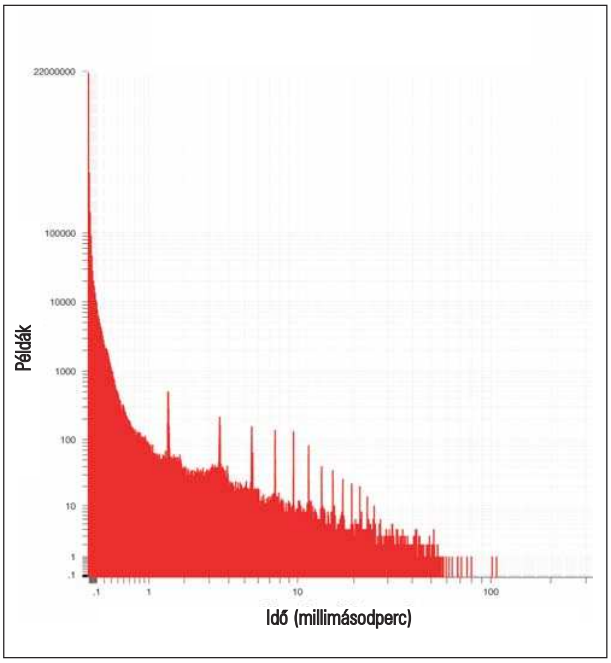
Ebből következően minden operációs rendszernek vannak olyan kritikus kódrészletei, amelyeket a programok csak egymás után érhetnek el.

Ha egy nagy fontosságú feladat – megszakítás fellépése miatt – hirtelen futtathatóvá válik, a processzor nem adható át neki azonnal, ha az éppen futó program pont egy ilyen kritikus szakaszban van.

A hosszúra nyúló kritikus szakaszok tehát jelentős mértékben befolyásolják a rendszer megszakításokra való válaszadási képességeit. Az alacsony késleltetéseket biztosító foltok megfelelő algoritmusok segítségével a hosszú kritikus szakaszok egy részét eltüntetik a Linux rendszermagból, illetve lerövidítik azokat.

Általában elmondható, hogy minél bonyolultabb egy alrendszer, annál hosszabbak a kritikus szakaszok. Mivel a Linux számos összetett alrendszert támogat, köztük fájlrendszereket, hálózati és grafikai alrendszereket, kritikus szakaszai rendkívül hosszúak, legalábbis egy kisméretű, valós idejű operációs rendszerhez hasonlítva.

Az erőforrások elvételét lehetővé tévő folt és az alacsony késleltetéseket garantáló foltok jelentős mértékben javítottak a Linux válaszidőin. A kritikus szakaszok ennek ellenére több tíz msec időtartamot is felölelhetnek, márpedig sok valós idejű alkalmazás számára az ilyen válaszidők elfogadhatatlanok.



2. ábra Válaszidők (kernel.org 2.4.21-pre4)

Mi az a védett processzor?

Mint korábban már említettem, a védett processzort a nagy fontosságú feladatok futtatására és a hozzájuk tartozó megszakítások kezelésére tartjuk fenn.

Védett processzor megadásának előfeltétele, hogy az operációs rendszer képes legyen a folyamatok és a megszakítások affinitásának (futtató processzorának) beállítására. A 2.4-es sorozatszámú Linux képes a megszakítások affinitásának beállítására, és ugyanez a képessége folyamatokra a megfelelő nyílt forrású foltok segítségével biztosítható. (Lásd: *„Processzorhoz kötés”*, Linuxvilág, 2003. szeptember).

A védett processzor előnyei

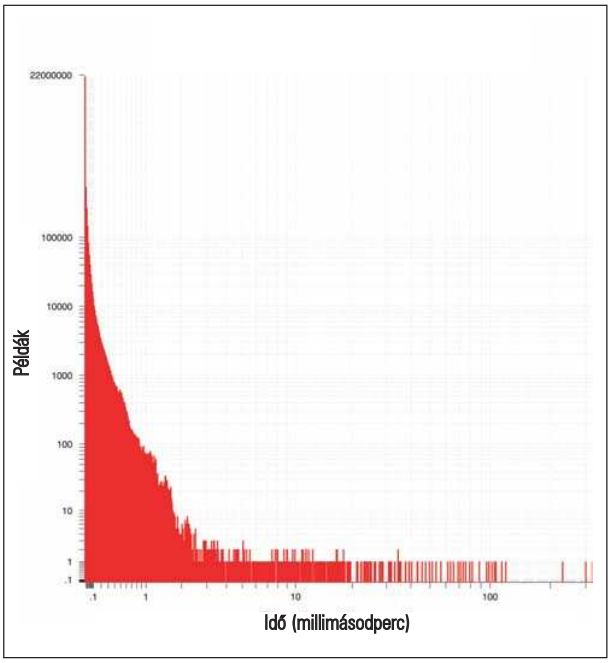
Mivel a védett processzorok nem futtatnak háttérfolyamatokat, a védett processzorokra kerülő nagy fontosságú feladatoknál nincs meg annak a veszélye, hogy azért ne tudnának válaszolni egy megszakításra, mert egy másik, éppen kritikus szakaszban lévő folyamat köti le a processzort.

A megszakítások mindig nagyobb fontosságot kapnak, mint bármely folyamat, és mivel fellépésük előre nem jósolható meg, a nem valós idejű megszakítások jelentős bizonytalansági tényezőt hozhatnak a folyamatok előre megbecsült futtatási idejébe.

Egy védett processzor nem foglalkozhat a megszakítások kezelésével, hacsak az adott megszakítás nem egy nagy fontosságú, a védett processzorhoz rendelt feladathoz tartozik.

Védett processzoros rendszer megvalósítása

Ha folyamatok és megszakítások affinitását egyaránt meg tudjuk adni, akkor olcsón meg tudjuk oldani a kiszemelt processzor védelmét is. Ez a megoldás azonban csak akkor működőképes, ha egyik folyamat sem változtatja meg saját affinitását úgy, hogy a védett processzort is igénybe vegye.



3. ábra Válaszidők (Red Hat 8.0)

Általánosságban tehát ennél erőteljesebb védelemre van szükség – alábbiakban egy ilyen megoldást fogok ismertetni.
Processzor védelmét a */proc* felületen keresztül írhatjuk elő, itt a rendszergazda egy maszkkal választhatja ki a védett processzorokat, valamint egy megfelelő paranccsal a maszk módosítását is megoldhatja.
A felület lehetővé teszi a processzorok dinamikusan történő védetté nyilvánítását.
Ha egy processzort védettként jelölünk meg, akkor egyik folyamat sem módosíthatja saját affinitását úgy, hogy a védett processzort is igénybe vegye, feltéve, hogy a tiltás figyelembe vételével marad olyan processzor, amelyen a folyamat futhat.
A felhasználóknak tehát kifejezetten ki kell választaniuk a védett processzort folyamataik futtatására, ha igénybe akarják venni annak erőforrásait.
Saját affinitásmaszkjához csak privilegizált folyamat adhat hozzá processzort.
Ennél a megvalósításnál módosítani kell a folyamatok affinitását beállító programkódot. A `sys_sched_setaffinity()` függvénnyel processzoraffinitást lehet beállítani. A függvény módosításával eltávolíthatjuk a védett processzort minden felhasználóra specifikus maszkból, amikor az affinitást beállítottuk :

```
p->cpus_allowed_user = new_mask;
if (new_mask & ~shielded_proc)
    new_mask &= ~shielded_procs;
set_cpus_allowed(p, new_mask);
```

Megjegyzem, a védett processzort jelölő bitek nem törlődnek, ha ezáltal a folyamat futtatása mindegyik processzoron tilossá válna. A `cpus_allowed_user` egy új mező a `task` adatszerkezetben, amely az eredeti, a felhasználó által megadott affinitást tárolja.

Amikor a védett processzorok maszkja megváltozik, a fenti kódot a rendszer összes folyamatára meg kell hívni.
Ehhez kell tudni a felhasználó által eredetileg beállított processzoraffinitást. A védett processzor maszkjának megváltoztatására alkalmas kód a következőképpen néz ki:

```
for_each_task(p) {
   new_mask = p->cpus_allowed_user &
cpu_online_map;
   if (new_mask & ~shielded_proc)
      new_mask &= ~shielded_procs;
   if (new_mask != p->cpus_allowed)
      set_cpus_allowed(p, new_mask);
}
```

Teljesítménypróbák

A megszakításokra adott válaszok megadási idejének mérésére az ***Andrew Morton*** weboldalán talált Realfeel teljesítménymérő programot használtuk.
Azért erre a programra esett a választásunk, mert a valós idejű óra (***Real Time Clock***, RTC) illesztőprogramot használja, és ezt több Linux-változatban is alkalmazzák megszakítások létrehozására.
A program az RTC illesztőprogram által létrehozott megszakításra adott válaszok megadási idejét méri. Az RTC-t megszakítások periodikus előállítására használjuk, pontosan 2048 Hz-es frekvenciával.
Egy `read` rendszerhívást is támogat, amely a következő megszakítás létrehozásakor tér vissza. A válaszidő mérésére az IA-32 TSC időzítőt érdemes használni, ennek felbontása a processzor órajelétől függ.
A megszakításokra adott válaszok létrejöttének idejét úgy tudjuk mérni, hogy először kiolvassuk a TSC értékét, majd ismételt `read` hívásokat indítunk a */dev/rtc*-re.
Amikor minden `read` hívás véget ért, a program kiolvassa a TSC aktuális értékét.
A két egymást követő TSC érték különbsége adja meg, hogy az RTC megszakításra várva a folyamat mennyi ideig volt blokkolva. A várt válaszidő 1/2048 másodperc.
Minden a várt válaszidőn túli idő lappangási időnek számít a megszakítás kezelését tekintve.
Ha mérni akarjuk, hogy legrosszabb esetben mennyi idő alatt válaszol a rendszer a megszakításokra, akkor aktív háttérterhelést kell rónunk rá.
A háttérterhelésnek akkorának kell lennie, hogy a rendszer csak késve tudjon reagálni a megszakításokra, és a futtatás előre jósolhatóságát rontó versengést kell okoznia az erőforrásokért. A háttérterhelés előidézésére a Red Hat ***stress-kernel*** csomagja tökéletesen megfelel.
A csomagból a következő programokat választottuk ki: TTCP, FIFOS_MMAP, P3_FPU, FS és CRASHME.
A TTCP nagyméretű adatcsomagokat küld és fogad a hurokeszközön keresztül.
A FIFOS_MMAP több próba kombinációja, felváltva cserél adatokat két folyamat között FIFO elven, illetve végez műveleteket egy `mmap`-pal kezelt fájlon. A P3_FPU próba lebegőpontos mátrixokon végez különféle műveleteket.
Az FS próba egy maréknyi fájlon változatos műveleteket végez, például nagyméretű, középen üres fájlokat hoz lét-

re, majd csonkolja és bővíti ezeket. Végül a CRASHME próba véletlenszerű adatokat tartalmazó puffereket hoz létre, majd ezekre az adatokra ugrik, és megpróbálja futtatni őket.
Bár Ethernet-forgalom nem keletkezik a rendszeren, a hálózati kapcsolat fennmarad, és a gépnek a próbák futtatása közben is kezelnie kell a normál szórásos forgalmat.
A ***stress-kernel*** csomagban szereplő NFS_COMPILE próba egy újabb változatát használtuk, az eredeti erőforrás-felszabadítási hibái miatt ugyanis a próbát nem lehetett hosszabb ideig futtatni.
Az NFS_COMPILE parancsfájl újra és újra lefordítja a rendszermagot, miközben egy a helyi hurokeszközön keresztül elérhetővé tett NFS fájlrendszert használ.
A próbák futtatására egy 1 GB memóriával és SCSI merevlemezzel felszerelt kétprocesszoros Pentium 4 Xeon gépet használtunk.

Eredmények

A védett processzor megszakításokra adott válaszainak idejét a ***Concurrent Computer Corporation*** által készített RedHawk Linux 1.3-as változata alatt mértük. A RedHawk 2.4.21-es rendszermagra épülő Linux rendszer. Érdemes megjegyezni, hogy a RedHawk Linux rendszermag különleges szolgáltatásai közül csak az egyik a védett processzorok kijelölésének lehetősége.
Az alább közölt eredmények eléréséhez – kisebb-nagyobb mértékben – az egyéb fejlesztések is hozzájárultak. A rendszermag például több nyílt forrású foltot is tartalmaz, köztük ***Robert Love*** erőforrások elvételét lehetővé tévő foltját, ***Andrew Morton*** alacsony késleltetést biztosító kiegészítését valamint a 2.5-ös Linux-ág O(1) ütemezőjét.
A teljesítménypróbák eredményét további módosítások is befolyásolhatták, így például azok az algoritmus-változtatások, amelyek a Linux rendszermag kritikus szakaszainak lerövidítését szolgálják; és az, amelyik az alsóbb megszakítások állítható fontosságú és ütemezési házirenddel szabályozható rendszermag démonon belüli kezelését teszi lehetővé.

Red Hawk és Red Hat

Az *1. ábrán* látható, hogy RedHawk Linux alatt védett processzor használatával illetve anélkül milyen eredményeket sikerült elérni. A két eredmény közötti eltérés szembeszökő. A rendszer a legtöbb esetben mindkét módban kevesebb mint 100 mikromásodperc alatt képes volt válaszolni az RTC megszakításokra.
Általában tehát elmondható, hogy a Linux kellő gyorsasággal válaszol a megszakításokra. Csakhogy, mint már említettem, a valós idejű rendszereknél a legfontosabb mutató a legrosszabb esetben adott válaszidő. Ennek oka az, hogy éppen ilyenkor fordulhat elő, hogy egy valós idejű alkalmazás túllépi a megadott határidőt.
Védett processzoros módban a legrosszabb esetben is 220 mikromásodperc alatt megérkezett a válasz az RTC megszakításra. Amikor nem használtunk védett processzort, akkor a felső határ 10 msec volt, ami egy nagyságrenddel nagyobb, mint a védett processzoros mód legrosszabb eredménye.
Ugyan a válaszidő az esetek kevesebb mint egy százalékánál volt nagyobb mint 200 mikromásodperc, több ezer esetben az 500 mikromásodperc időtartamot is meghaladta. Egy valós idejű rendszerben ezek az esetek nagy valószínűséggel egy-egy elmulasztott határidőt jelentenek.
Ugyanezt a megszakításpróbát egy módosítások nélküli 2.4.21-es kernel.org rendszermagon (*2. ábra*), valamint egy 8.0-s Red Hat terjesztésen (*3. ábra*) is lefuttattuk. Ez a Red Hat rendszermag nem tartalmazza az erőforrások elvételét lehetővé tévő foltot, de az alacsony késleltetéseket biztosítót igen, vagyis esetében túl hosszúra nyúló kritikus szakaszokról nem beszélhetünk.
Mivel védett processzort ezek közül a rendszermagok közül egyik alatt sem lehet kijelölni, esetükben eredményeket csak a nem védett módnál tüntettünk fel.
Ezek a rendszermagok a RedHawk rendszermaggal mértekhez hasonló, általában 100 mikromásodperc alatti válaszidőket mutatnak.
A legrosszabb eset azonban messze kedvezőtlenebb, mint a RedHawk Linux akár védett processzor nélküli teljesítménye, a ***kernel.org*** Red Hawk összeállítása 107 msec, a Red Hat pedig 323 msec alatt válaszol, ha minden kötél szakad.
Az eredmények cseppet sem meglepők, hiszen ezeket a rendszermagokat inkább az erőforrások egyenlő folyamatok közötti elosztására tervezték, és feladatuk inkább általánosan jó rendszerjellemzők, és nem valós idejű válaszidők biztosítása.

Összefoglalás

Egyértelmű, hogy védett processzor kijelölésével jelentős javulást tapasztalhatunk a linuxos rendszerek legrosszabb esetben adott válaszidőinek tekintetében.
A védett processzorok alkalmazása hatékony megoldás, mivel így a rendszer legfontosabb feladatai számára erőforrások tarthatók fenn.
Mindezt a Linux normál alkalmazásprogramozási felületének módosítása nélkül érhetjük el.
Írásomban csak az RTC megszakításra adott válaszokról esett szó. Az RTC kiválasztását az indokolta, hogy a legtöbb Linux-változatban alapvető szolgáltatásnak számít. Más megszakításforrások és jobban optimalizált illesztőprogramok használatával ennél jobb garantált válaszidőket is el lehet érni.
Aki szeretne részletesebben megismerkedni a védett processzoros megoldásokkal, illetve szeretné látni, hogy jobb válaszidőket garantáló illesztőprogrammal milyen eredményeket lehet elérni, az tanulmányozza át a ➲ http://www.ccur.com/isddocs/wp-shielded-cpu.pdf dokumentumot.

Linux Journal 2004. május, 121. szám

Stephen Brosky a Concurrent Computer Corporation Integrált megoldások részlegének vezető kutatója. Tagja volt a valós idejű alkalmazás- és programszálfelületekre vonatkozó Posix 1003.1b és 1003.1c szabványokat elkészítő IEEE bizottságnak.

Feladat-automatizálás az Aap segítségével

Az Aap rugalmas eszköz, amely képes mindenre, amire a make, sőt még egyébre is. Most megtudhatjuk, hogyan hozhatunk létre mobil megoldásokat a weboldalunk kezeléséhez és a programjaink lefordításához.

Sokan Makefile-t, vagy héjprogramokat használnak a műveletek önműködővé tételéhez, ha egy fájl változása valamilyen művelet végrehajtását teszi szükségessé. Elvégezzük a fájlon a kívánt módosításokat, majd segítségül hívjuk a `make`-et remélve, hogy a változtatásokkal kapcsolatos minden szükséges teendőt elvégeztük. Gyakran előfordul, hogy a Makefile-lal kapcsolatban cselhez kell folyamodnunk, és sokszor jutunk odáig, hogy a `touch` segítségével kell valamilyen hiányzó függőséget pótolnunk. Ez oda vezethet, hogy amikor mások kezébe kerül a gondosan beállított Makefile-unk, nehezen tudják kibogozni, hogyan is működik.

Az ilyen feladatok sokkal könnyebben és megbízhatóbban végezhetők el az ***Aap*** segítségével, mint a `make`-kel, sőt előbbi rendelkezik beépített Internet-támogatással is.

A szükséges letöltések és feltöltések végrehajtódnak anélkül, hogy erre parancsokat kellene megadnunk, vagy időbélyegző-fájlokat használnánk. A megbízhatóság forrása az önműködő függőség-felismerés és az aláírások használata az időbélyegzők helyett. Az Aap segítségével könnyebben adhatjuk meg az elvégzendő feladatot és így kevesebb hibát vétünk. Szükség esetén használhatjuk a héj parancsait is.

Ebben a cikkben két példán keresztül világítjuk meg az Aap működését: az első egy weboldal kezelése, a második pedig egy program lefordítása.

Az Aap telepítése

Az `Aap` használatához szükségünk van a Python 1.5-ös vagy újabb változatára. Ha abban a valószínűtlen helyzetben vagyunk, hogy nincs Python a rendszerünkön, töltsük le a ➲ http:\\www.Python.org oldalról, telepítsük a Linux rendszerünk telepítőlemezéről, vagy frissítsük a rendszerünket. Az `Aap` telepítése négy egyszerű lépésben elvégezhető. Kezdjük a legfrissebb ***Aap.zip*** csomag letöltésével (lásd a hálózatos ***Resources*** – források részt), ezután bontsuk ki a csomagot egy ideiglenes könyvtárba (`unzip aap-1.53.zip`).

Ha root-ként jelentkeztünk be, futtassuk az `./aap install` programot. Ha közönséges felhasználók vagyunk, telepítsük a programot a saját könyvtárunkba az `./aap install PREFIX=$HOME` paranccsal. Az `Aap` egy GNU GPL licenc alatt terjesztett nyílt forráskódú program.

Egy weboldal karbantartása

A friss `Aap` programmal felvértezve vágjunk bele az ismerkedésbe, és egy példafeladaton keresztül vizsgáljuk meg a program használatát. Korábban létrehoztunk egy egyszerű weboldalt HTML fájlokkal és képekkel. A fájlok jelenleg a helyi gépünkön vannak, fel kellene tölteni azokat a webkiszolgálóra. Az *1. listán* láthatjuk azt az Aap-receptet, amely elvégzi ezt a feladatot. Receptnek az `Aap` parancsfájljait nevezzük.

Mentsük ezt a parancsfájlt ***main.aap*** néven. Ez az a fájl, aminek a létrehozása a Makefile feladata: az alapértelmezett esetben futtatandó fájl. Az `aap` paraméterek nélküli futtatása az aktuális könyvtárban lévő ***main.aap*** fájl végrehajtását eredményezi.

Egy Aap-receptben lévő megjegyzések kettős kereszttel (`#`) kezdődnek és a sor végéig tartanak, éppen úgy, mint egy `Makefile` vagy héjprogram esetén. A recept első végrehajtandó sora egy értékadás: a `Files` változóhoz hozzárendelődik a feltöltendő fájlok listája. Nincsenek perjelek vagy az értékadás végét jelző egyéb írásjelek, az `Aap` a parancs folytatását a használt behúzás mértékéből ismeri fel. Ez első pillantásra furcsa lehet, de hamar hozzá lehet szokni. Ezzel az eljárással kiküszöbölhetjük az írásjelek használatából adódó hibalehetőséget és kényszerítjük magunkat a könnyen olvasható külalak használatára. A sorok behúzására használhatjuk a szóközt és a TAB billentyűt egyaránt.

Az `:attr` kulcsszóval kezdődő sor egy `Aap`-parancs. Minden Aap-parancs kettősponttal kezdődik, így könnyen felismerhető. Ez a parancs a `publish` tulajdonságot rendeli a megadott paramétereihez, ez határozza meg a fájlok feltöltésének a helyét, amikor azok közzétételre kerülnek. Az itt használt eljárás az ***scp:***//, a secure copy (biztonságos másolás). Az ***rsync:***// és az ***ftp:***// a másik két támogatott eljárás. Az `:attr` parancs utolsó paramétere a `$Files`, a Files változó értéke. A tulajdonság a `$Files` minden egyes eleméhez hozzárendelődik.

Amikor az `Aap`-t paraméter nélkül futtatjuk, az `all` célobjektum által meghatározott elemeket frissíti. Az utolsó sor azt határozza meg, hogy az alapértelmezett all célobjektum a publish tulajdonságtól függjön. Ez egy speciális célobjektum, amely arra utasítja az `Aap`-t, hogy azokat a tételeket töltse fel, amelyek publish tulajdonsággal rendelkeznek.

Most már szerkeszthetjük a HTML-fájljainkat, képeket adhatunk hozzájuk és nézegethetjük az eredményt a saját gépünkön. Amikor elégedettek vagyunk az eredménnyel, futtathatjuk az `aap`-t. Az `Aap` észleli, hogy mely fájlok változtak meg és feltölti azokat. Ehhez aláírásokat (ellenőrző összegeket) használ, így ha egy fájl korábbi változatát állítjuk vissza, akkor is megfelelően működik. Ha a `make`-et használnánk, a visszaállított fájlhoz is hozzá kellene nyúlnunk, hogy beállítsuk az időbélyegzőjét.
Ha ki szeretnénk próbálni ezt a példát, de nincs olyan kiszolgálónk, ahova feltölthetnénk az anyagot, használhatjuk a `file:/tmp/html/%file%` kifejezést a publish tulajdonság beállításánál. Ebben az esetben az `aap` szükség esetén létrehozza a */tmp/html* könyvtárt. Egy figyelmeztetés: az `Aap` nem törli a kiszolgálóról a már nem használt fájlokat. Ezt a `make` sem teszi meg nekünk. Saját magunknak kell a törlést kézzel elvégeznünk. Remélhetőleg az Aap rövid időn belül kiegészül az önműködő törlés lehetőségével is.

A képfájlok listázása

A képek kiválasztásánál használtunk egy dzsókerkaraktert: ***images/*.png***. Ez ugyan kényelmes, de magában rejti azt a veszélyt, hogy olyan képek is felkerülnek, amelyeket nem akarunk feltölteni. Minden egyes fájl megnevezésével elkerülhető ez a csapda, de így a felsorolásból könnyen kifelejthetünk valamit. Mivel ez egy elég általános probléma, az Aap egy függvényt biztosít a képek fájlneveinek a HTML-fájlokból történő kiszűrésére. A *2. lista* mutatja ennek a módját. A `get_html_images` Python-függvényt hívjuk segítségül, az aposztrofok között egy Python-kifejezés szerepel. Az Aap kiértékeli a kifejezést, majd az eredményt, a képfájlok neveit, berakja a helyükre.
A `get_html_images()` függvénynek vannak azonban korlátai is: csak tiszta HTML-fájlokkal képes dolgozni, amelyekben a képek relatív elérési útvonallal rendelkeznek.

HTML-fájlok előállítása

A legtöbb HTML-fájl fejrészből, címből, törzsrészből és lábrészből áll. Nyilvánvaló, hogy nem akarjuk minden egyes alkalommal újra begépelni a közös részeket. Egyszerű megoldás a több fájlból történő összefűzés. A *3. lista* az ezt megvalósító receptet mutatja. Öt részt használunk: `header` (fejrész), `title` (cím), `middle` (középrész), `contents` (tartalom), és `footer` (lábrész). A cím és a tartalom minden oldalon különböznek, de a másik három rész megegyezik.
A *2.* és *3. lista* közti lényeges eltérés a *3. listában* hozzáadott `:rule` parancs. Ez azt meghatározza, hogy a célobjektum (a HTML-fájl) öt forrásfájltól függ, és a parancs ezeket sorolja fel, aminek alapján a forrásfájlokból előáll a célfájl. A % jel a fájl neve helyett használatos, hasonlóan a * dzsókerkarakterhez. A `rule` parancs minden %-jele ugyanazt a nevet helyettesíti, így az ***index.html*** előállításakor az index szó helyett használatos. A forrásfájlok közt foglal helyet tehát az ***index_title.part*** és az ***index.part*** fájl. A `:rule` sora alatt azoknak a parancsoknak a – behúzott formában lévő – felsorolása következik, amelyek akkor futnak le, amikor a rule parancs célpontjának frissítése szükségessé válik. Vagyis a recept két szintre bomlik: a felső szinten lévő parancsok akkor futnak le, amikor a recept olvasásra kerül, a `rule`-parancsblokk pedig később, amikor szükségessé válik.

1. lista Recept fájlok webkiszolgálóra való feltöltésére

```
# A feltöltendő fájlok listája.
Files = index.html
        info.html
        download.html
        images/*.png

# A publish tulajdonság jelzi a feltöltés
# helyét.
:attr {publish =
➥scp://my.server.net/html/%file%}
        $Files

# Amikor target (cél) meghatározása nélkül
# futtatjuk: nyilvánossá teszi a fájlokat.
all : publish
```

2. lista A képek fájlneveinek kinyerése a HTML-fájlból

```
# A feltöltendő fájlok listája.
Files = index.html
        info.html
        download.html
Files += `get_html_images(Files)`

# A publish tulajdonság jelzi a feltöltés
# helyét.
:attr {publish =
➥scp://my.server.net/html/%file%}
        $Files

# Amikor target (cél) meghatározása nélkül
# futtatjuk: nyilvánossá teszi a fájlokat.
all : publish
```

A `:cat` parancs összefűzi a fájlokat, hasonlóan a UNIX `cat` parancsához. Valójában ennél sokkal többre képes, például egy megadott címről történő fájlbeolvasásra. A `rule` parancsban a `$source` a forrásfájlok teljes listáját jelenti.
A HTML-fájlokat létre kell hoznunk, még mielőtt kinyernénk a bennük szereplő képfájlok listáját. Ehhez az `:update` parancsot kell segítségül hívnunk a `get_html_images()` függvény hívását megelőzően. Ennek hatására a megadott szabály alapján megtörténik a HTML-fájlok frissítése. Ez a parancs a recept felső szintjén foglal helyet, vagyis mindig végrehajtódik, amikor az `Aap` olvassa a receptet.
Most, hogy ennyi fájlunk van, honnan fogja az `Aap` tudni, hogy milyen tennivalókat kell végrehajtania? Az `Aap` erre a függőségeket használja, akárcsak a `make`. Azzal a célobjektummal kezdi, amit a parancssorban megadunk, ha nem adunk meg semmit, akkor az `all` (minden) a feltételezett cél. Az `Aap` ezután megkeresi azokat a függőségeket és szabályokat, amelyekben ez a cél megjelenik a kettőspont előtt. A kettőspont lényegében a függést jelenti, a kettőspont

3. lista HTML fájl létrehozása öt részből

```
Files = index.html
        info.html
        download.html

:rule %.html : header.part
               %_title.part
               middle.part
               %.part
               footer.part
    :cat $source >! $target

:update $Files
Files += `get_html_images(Files)`

:attr {publish =
➥scp://my.server.net/html/%file%}
        $Files

all : publish
```

után helyezkednek el azok a forrásfájlok, amelyektől a cél függ. Ezután ezeket a forrásfájlokat vizsgálja meg az Aap és megkeresi az összes olyan szabályt, amelyben ezek célként szerepelnek. A folyamat rekurzívan folytatódik mindaddig, amíg el nem fogynak a szabályok. Az eredmény egy függőségekből álló faszerkezet. Az `Aap` ezután a fa végétől (a legmélyebb résztől) kezdve lefuttatja azokat a parancsokat, amelyek ennek a függőségi rendszernek a felépítéséhez szükségesek. Kicsit bonyolultnak tűnik, igaz? Mivel azonban az `Aap` elvégzi ezeket a teendőket, nekünk nincs más dolgunk, mint hogy minden egyes célnál megadjuk, hogy milyen forrásoktól áll függésben. Az `Aap` ezután már el tudja dönteni, mi a teendő.

Időbélyegző létrehozása

Egy hasznos kiegészítésként adjunk időbélyegzőt is a HTML-fájlhoz, így a weboldalon láthatóvá válik, hogy mikor volt az oldal legutoljára frissítve. Írjuk be valahova a fájl lábrészébe (***footer.part***) a `@TIMESTAMP@` karakterláncot.
A *4. lista* mutatja azt a szabályt, amelynek alapján ez a karakterlánc az aktuális dátummal lesz helyettesítve. A recept többi része ugyanaz, mint a *3. listában*. Az `:eval` parancs kiértékeli a Python-kifejezést, amelyben a `string.replace` egy szabályos Python-függvény egy karakterlánc másikkal való kicserélésére. Ezzel a módszerrel bármilyen Python-kifejezést használhatunk a szöveg szűrésére. A HTML-oldal épp úgy halad át az `:eval` parancs csővezetékén, mintha a héj alól tennénk ugyanezt. A szabály használatának első alkalmával minden HTML-fájl frissítésre kerül. Ennek oka, hogy az Aap megjegyzi a parancsok aláírását, emiatt egy receptben történt változtatás után nem kell foglalkoznunk a fájlok újbóli előállításának végrehajtatásával.

Feltöltés az rsync segítségével

Amikor egy honlapon valamilyen kis módosítást eszközölünk, nem túl gazdaságos teljes fájlt feltölteni.

4. lista Szabály a létrehozott HTML-fájl időbélyegzővel történő ellátására

```
:rule %.html : header.part
               %_title.part
               middle.part
               %.part
               footer.part
    :print Generating $-target
    :cat $source
        | :eval string.replace(stdin,
                  ➥´@TIMESTAMP@´, _no.DATESTR)
        >! $target
```

A feltöltés hatékony módszere az `rsync` segítségével valósítható meg, amely a fájlnak csak azokat a részeit tölti fel, amelyek megváltoztak. Az Aap akkor alkalmazza az `rsync`-et, ha a `publish` tulajdonságban megtalálja az `rsync://` beállítást. Alapértelmezésben az `rsync` SSH-kapcsolaton keresztül kommunikál, ezt a beállítást a `$RSYNC` változó megváltoztatásával módosíthatjuk. Az `rsync` nem szabványos parancs, amennyiben nincs a rendszerünkre telepítve, részünk lehet az Aap egy hasznos szolgáltatásában: a program választási lehetőségként felajánlja az `rsync` telepítését.

```
% aap
Aap: Uploading [´index.html´] to

rsync://my.server.net/html/index.html
Cannot find package "rsync"!
1. Let Aap attempt installing the package
2. Retry (install it yourself first)
q. Quit
Choice:
```

Az `Aap` rendelkezik a programtelepítés képességével, éss ha szükséges, az `Aap` honlapjáról még recept letöltésére is képes, amely leírja az adott program telepítésének módját. Itt kapóra jön az Aap letöltési képessége.

Programfordítás az Aap-vel

Az `Aap` támogatja a C vagy C++ forráskódból történő fordítást is. Íme egy egysoros recept, amely lefordítja a négy C forrásfájlból álló `myprog` nevű programot:

```
:program myprog : main.c common.c various.c args.c
```

A recept egyszerűsége ellenére az Aap gondoskodik az alábbiakról:

- A függőségeket önműködően feltérképezi. Nincs szükség a befoglalandó fejállomány megadására vagy a `make depend` parancs futtatására.
- A recept a legtöbb rendszeren módosítás nélkül működik. Az `Aap` megkeresi a használandó fordító és összefűző programot és meghatározza a szükséges paramétereket.
- Az objektumfájlok minden egyes rendszer esetén külön könyvtárban tárolódnak, így több változatot is fordíthatunk anélkül, hogy az előzőt el kellene távolítanunk.

- Az Aap létrehoz egy naplófájlt, az ***AAPDIR/log*** fájlt, amely a végrehajtott műveleteket részletezi. Ha a fordításunk nem sikerül és a kimenet már nem látszik a képernyőn, nincs szükség arra, hogy a kimenet átirányításával újrafuttassuk a fordítást.
- Néhány alapértelmezett cél önműködően létrejön: az `aap install` telepíti a programot, míg az `aap clean` törli a létrehozott fájlokat.

Ugyanezt a munkát a make-kel is elvégezhetnénk, de a Makefile sokkal hosszabb lenne.

Változatok létrehozása a fordításkor

Most hozzuk létre egy program két változatát, egy kiadásra szánt verziót, egyet pedig hibakeresés céljából. Az Aap támogatja a különféle változatok létrehozását. Mindössze azt kell megadnunk, hogy milyen változatokra lesz szükségünk, és mi különbözteti meg ezeket egymástól. Az 5. lista mutatja a szóban forgó receptet. Az első sor, a `:variant` parancs meghatározza a változat nevét, amelyre a fordításkor hivatkozhatunk. Ezt a változót parancssorban állíthatjuk be; az `aap Build=debug` a hibakereső változatot adja. Ha nem adunk meg paramétert, akkor a nyilvános változat készül el, mivel az szerepel először. A behúzás mértéke azonosítja a `:variant` parancs további részeit. A lehetséges értékek behúzása kicsit kisebb, az egyes értékeknél használt parancsok nagyobb behúzással szerepelnek. Az egyes részek illesztése kötelező, ezzel az olvasás is könnyebbé válik. A `release` változat beállítja az `OPTIMIZE` változót. Ez egy nullától kilencig terjedő szám lehet, ami az optimalizálás szintjét határozza meg. Ez önműködően beállítja a használt fordítóprogram megfelelő paraméterét. A `debug` változat a `DEBUG` változót `yes` értékre állítja. Az alapértelmezett érték `no`. A `Target` változó az eredményként kapott program nevét tartalmazza. A két változat különböző nevet használ, így mindkét program létezhet egy időben. A változatok ilyen módon való használatának nagy előnye, hogy az objektumfájlok minden változathoz önműködően külön könyvtárban tárolódnak. Amikor átváltunk a két változat között, nem szabad elfelejtenünk, hogy az Aap nem hoz létre újra minden fájlt.

Fordítás más nyelvből

Ha nem C vagy C++ nyelvű kódot szeretnénk fordítani, szükség van a megfelelő nyelvmodul importálására. Néhány alapmodult már az Aap is magában foglal. Például a következőképpen fordíthatunk le D nyelvű forráskódot (a D egy új programozási nyelv):

```
:import d
:program myprog : main.d common.d various.d args.d
```

Az `:import d` parancs a D nyelv támogatásának betöltésére szolgál. Ettől eltekintve a folyamat hasonló a C-források fordításához. Saját magunk is írhatunk olyan modult, amely megteremti egy nyelv támogatottságát. Mivel az Aap nyílt forráskódú alkalmazás, nyugodtan átadhatjuk a modult az Aap részeként való terjesztésre. Amíg ez megtörténik, helyezzük el a fájlt az Aap modulkönyvtárába, amely bővítményként működik.

5. lista Kibocsátársa és hibakeresésre szánt változatok fordítása

```
:variant Build
   release
      OPTIMIZE = 4
      Target = myprog
   debug
      DEBUG = yes
      Target = myprogd

:program $Target : main.c common.c various.c
➥args.c
```

6. Lista Az Aap használata a make helyettesítésére.

```
 all : hello

# A hello program manuális fordítása
hello : hello.c
   :sys cc -o $target $source

# Egy C program előállításának nehézkes módja.
hello.c:
   :print Generating $target
   :print >! $target $(#)include $(<)stdio.h$(>)
   :print >> $target main() {
   :print >> $target   printf("Hello World!\n");
   :print >> $target   return 0;
   :print >> $target }
```

Egy KDE-alkalmazás fordítása

Egy KDE-alkalmazás lefordítása egy csomó eszköz használatát vonja maga után, például a Qt Designer alkalmazását a párbeszédablakok létrehozására, a fejállományok létrehozását a felhasználói felület leírásaiból és a folyamatközi kommunikáció kódjának előállítását. Mindezek ellenére egy KDE-program lefordítását végző recept is lehet ilyen egyszerű:

```
:import kde
:program logger : main.cpp
                  logwidget.ui
                  dcop.h {filetype = skel}
                         {var_OBJSUF = _skel.o}
```

A három bemeneti fájl közül a `main.cpp` közvetlenül lefordítható. A Qt Designer ***logwidget.ui*** nevű fájlját először az `uic` használatával kell feldolgozni, melynek során létrejön egy `include`-fájl, ezután pedig a `moc`-ot kell használni. Az Aap felismeri az ***.ui*** kiterjesztést és odafigyel minderre. Az ilyen többlépcsős fordítófolyamat-kezelés, amely során először az `ui`-fájlból `h`-, majd ebből `moc`- és objektumfájl jön létre, az Aap rendkívül hasznos szolgáltatása. Ugyanennek a `Makefile` segítségével történő végrehajtása sokkal több explicit szabályt igényel. A ***dcop.h*** fájl különleges KDE-elemeket tartalmaz, de normál kiterjesztéssel rendelkezik,

7. lista A Python használata a fájlnévlisa előállítására.

```
LASTPATCH = 144

# A foltok fájlneveinek előállítása.
@Patches = ''
@for i in range(1, int(LASTPATCH) + 1):
@    Patches = Patches + ("6.2.%03d " % i)

# Alapértelmezett target (cél): minden foltot
# alkalmazunk.
all: done/$*Patches

# A két könyvtár létezésének biztosítása..
:mkdir {force} patches done

# Szabály a folt alkalmazására.
:rule done/% : patches/% {fetch =
➥ftp://ftp.vim.org/pub/vim/%file%}

    :sys patch < $source
    :touch $target
```

ezért önműködően nem ismerhető fel. Ez az oka, hogy a fájltípus tulajdonságát be kellett állítani. A `:program` parancsnak szintén tudnia kell az objektumfájl nevét, ezt adja meg a `var_OBJSUF` tulajdonság.
A használt KDE-eszközöket nem kell külön megadnunk, ezt az összetettséget a KDE modulfájl rejti el a szemünk elől. Ez sokkal kevésbé bonyolult, mint az `automake` használata.

Az Aap használata a make helyett

Idáig olyan magas szintű `Aap`-parancsokat használtunk, amelyekkel gyorsan megadhatóak a szükséges teendők. A nem szabványos feladatok végrehajtásához pontosan meg kell határoznunk a függőségeket és a végrehajtandó parancsokat. Ez nagyjából úgy működik, mint egy `Makefile`. A héjparancsok mellett használhatunk hordozható `Aap`-parancsokat is. Ha ez sem elég, a Python parancsfájlok is rendelkezésünkre állnak.
A *6. lista* mutatja, hogy miképpen fest egy ilyen alacsony szintű recept. Ebben minden függőséget pontosan meghatároztunk – az `all` a `hello`-tól függ, a `hello` a ***hello.c*** fájlból fordítódik, a ***hello.c*** pedig az alapoktól jön létre lépésenként. Mivel egy receptben a fordítóparancsok Aap-parancsok, a héjparancsok futtatásához szükség van a `:sys` használatára. A példában szereplő `:sys cc` a C fordítóprogramot futtatja. A héjparancsok használatával tehát csökken a receptek hordozhatósága. A ***hello.c*** fájl `:print` parancsok használatával jön létre.
Az első sor a `>! $target` kifejezést használja, amely felülírja az esetleg már létező `hello.c` fájlt. A felkiáltójel nélkül csak egy hibaüzenetet kapnánk, amely a fájl létezéséről tájékoztatna. Ugyanez a sor tartalmazza a `$(#)` kifejezést is, amely megváltoztatja a # karakter megjegyzés kezdetét jelző különleges jelentését. Ugyanígy a `$(<)` és `$(>)` kifejezésekkel kapjuk meg a `<` és `>` karaktereket az átirányítás helyett.
A `hello.c` fájl létrejön, ha még nem létezne, nincs semmilyen forrásfájl-függőség megadva. A fájl egy másik helyzetben is újragenerálódhat: amennyiben valamelyik `:print` parancs megváltozik, mivel ez megváltoztatja a fordítóparancsok aláírását. Amennyiben ezek a parancsok megváltoznak, az `Aap` tudja, hogy a célfájlt újra létre kell hozni.
A fájlt `Aap`-parancsokkal hozzuk létre, nincs szükség héjparancsok használatára. A receptnek ez a része ezért bármilyen rendszeren használható.
Az `Aap`-parancsok száma azonban korlátozott. Ha további szolgáltatásokra van szükségünk de a hordozhatóságot is meg szeretnénk őrizni, a Python parancsfájlok állnak rendelkezésünkre. Az Aap-receptek minden forgalomvezérlési megoldása a Pythonra épül. A *7. listán* láthatunk egy példát egy receptre, amely a `Vim` foltjait telepíti.
Egy ciklust használunk a foltok fájlnévlistájának előállítására a `vim-6.2.001`-től kezdődően és az utolsó foltszámmal bezárólag, amelyet a `LASTPATCH` változó ad meg. Minden egyes foltot le kell tölteni és telepíteni. A `done/$*Patches` kifejezés `$*` jele az `rc`-stílusú változókiterjesztésre utal, a `done/` a Patches minden eleme elé beszúrásra kerül.

Programcsomag telepítése

Korábban említettük, hogy az `Aap` képes az `rsync` telepítésére, ha az nincs fenn a rendszerünkön. A csomagtelepítő folyamat közvetlenül is hívható. Például az Agide telepítéséhez az `aap --install agide` parancsot használhatjuk. Az Agide az A-A-P GUI IDE, az A-A-P projekt egy másik része, amelyet programok fordítására és hibakeresésre használhatunk a `Vim` és `gdb` segítségével.
Bár még a fejlesztés korai szakaszában van, már most elég jó arra, hogy C programokat fejlesszünk és teszteljünk vele. Számos csomag már most elérhető és az idő múlásával egyre több válik azzá.
A csomagok aktuális listája elérhető a ➲ http://www.a-a-p.org/packages.html oldalon. Az `Aap` maga is telepíthető. A legfrissebb változatra való frissítés az `aap --install aap` paranccsal végezhető el. Ha a rendszerünk rendelkezik csomagkezelővel, valószínűleg jobban járunk, ha azt használjuk.

Összegzés

E cikkből képet kaphattunk arról, hogy milyen feladatokat tehetünk önműködővé az `Aap` segítségével.
A kísérletezés során rengeteg segítséget találhatunk az igen alapos súgóban, amely különböző formátumokban letölthető az `Aap` honlapjáról (lásd a cikkhez kapcsolódó címeket). Ezeken az oldalakon rengeteg olyan dolognak a leírását megtalálhatjuk, amire ebben a cikkben nem tudtunk kitérni, ilyen például a CVS használata a változatkezelésre, az önműködő beállítás és sok más.

Linux Journal 2004. május, 121. szám

Bram Moolenaar az Aap projektvezetője és fő fejlesztője. Leginkább a Vim szerkesztőprogramon végzett fejlesztőmunkájáról ismert. Bram Aap-n végzett munkáját a Stichting NLnet támogatta (www.NLnet.nl). A honlapját a www.Moolenaar.net címen találjuk meg.

udev – állandó eszköznevek a felhasználói térben

Vessünk véget major és minor számok okozta kavalkádnak!

A 2.5-ös rendszermagtól kezdve, a rendszer minden fizikai és virtuális eszköze a felhasználói térből is látható, és a `sysfs` hierarchikus szerkezeten keresztül érhető el. Az */**sbin**/**hotplug*** segítségével figyelmeztetést kaphatunk, ha a rendszer új eszközzel bővül vagy eltávolítottuk valamelyiket. E két lehetőség használatával végre dinamikus, rugalmas eszközneveket használó `/dev` rendszert hozhatunk létre.

Cikkünk témája az `udev` program, amely a `devfs` feladatait vállalja át. Ez a megoldás bármely pillanatban képes a rendszer eszközeihez */**dev*** bejegyzéseket szolgáltatni. Továbbá olyan képességekkel is rendelkezik, amelyeket ezidáig kizárólag a `devfs` segítségével nem lehetett megvalósítani: képes az eszköznevek folyamatos megőrzésére, mialatt az eszköz szabadon mozoghat az eszközfában, rugalmas eszköz elnevezési megoldásra, figyelmeztet ha az eszköz külső rendszerei megváltoznak, valamint a teljes névkiosztási szabályrendszert a rendszermagon kívülre helyezi.

A Linux gépeken a rendszer valamennyi eszközállományát a */**dev*** könyvtárban találjuk. Az eszközállomány adja meg, hogyan érhet el a felhasználói program egy bizonyos alkatrészt vagy függvényt. Például, a */**dev**/**hda*** állományt hagyomány szerint a rendszer első IDE meghajtójának elnevezésére használjuk. A ***hda*** név megfeleltethető egy major és minor számnak, melyek alapján a rendszermag meg tudja állapítani, melyik alkatrésszel vegye fel a kapcsolatot. Jelenleg rengeteg nevet alkalmazunk, melyek a különféle major és minor számoknak felelnek meg.

Minden egyes major és minor számpárhoz valamilyen nevet rendelünk ami az adott alkatrésztípushoz tartozik. Ezt a hozzárendelést a Linux *„**assigned names and numbers authority**”* szervezete (röviden LANANA, magyarul „Linux név és szám összerendelési hivatal”) végzi, melynek jelenlegi eszközlistáját a honlapján találjuk (lásd a hálózati forrásokat).

Ahogy a linux kezdett újfajta eszközöket is támogatni, ezeket is major és minor számtartományokhoz kellett rendelni, hogy aztán a felhasználók a */**dev*** könyvtáron keresztül elérhessék azokat. Alternatív megoldásként az eszközöket elérhetjük a fájlrendszeren keresztül is (lásd a hálózati forrásokat). A 2.4-es és korábbi verzióban a major számok érvényes tartománya 1–255-ig terjedt, míg a minor számok 1–255 között lehettek. A korlátozott tartományok miatt, az új major és minor számok kiosztását a 2.3-as fejlesztési ciklus idejére befagyasztották. Azóta a zárlatot már feloldották és a 2.6-os rendszermag érvényes major számtartományát 4,095-re növelték. Egyazon major számhoz pedig több mint egymillió minor szám rendelhető.

Melyik /dev bejegyzés melyik eszköz?

Amikor a rendszermag egy új eszköztípust talál, általában az eszköztípushoz tartozó következő major/minor párost osztja ki. Így aztán rendszerindításkor az első felderített USB nyomtató kapná a 180-as major számot és a 0-s minor számot, amire a */**dev*** könyvtár alatt a */**dev**/**usb**/**lp0*** hivatkozik. A második USB nyomtató a 180-as major számot és az 1-es minor számot kapná, amit a */**dev*** alatt a */**dev**/**usb**/**lp1*** azonosít. Ha a felhasználó újraszervezi az USB hálózatot, esetleg behelyez egy új USB elosztót, hogy több USB eszközt kezelhessen a rendszeren, a számítógép következő indításakor a nyomtatók USB keresési sorrendje megváltozhat, megcserélve ezzel a két nyomtató minor szám hozzárendelést. Ugyanez érvényes szinte minden eszközre melyet a számítógép működése közben eltávolíthatunk vagy behelyezhetünk. A ***hot-plug*** rendszerű PCI megoldások és buszok megjelenésével (ilyen például az IEEE1394, USB és a CardBus), majdhogynem minden eszköz szembekerül ezzel a problémával.

A 2.5-ös rendszermag megjelenésével jelentősen leegyszerűsödött annak megállapítása, melyik eszköz minor számát rendeltük melyik fizikai eszközhöz. Ha egy rendszerhez két USB nyomtatót csatlakoztattunk, a ***sysfs*** */**sys**/**class**/**usb*** alkönyvtár szerkezete a következőképpen néz ki:

```
/sys/class/usb/
|-- lp0
|   |-- dev
|   |-- device ->
➥ ../../../devices/pci0/00:09.0/usb1/1-1/1-1:0
|   `-- driver -> ../../../bus/usb/drivers/usblp
`-- lp1
    |-- dev
    |-- device ->
../../../devices/pci0/00:0d.0/usb3/3-1/3-1:0
    `-- driver -> ../../../bus/usb/drivers/usblp
```

```
$ cat /sys/class/usb/lp0/device/serial
HXOLL0012202323480
$ cat /sys/class/usb/lp1/device/serial
W09090207101241330
```

Az egyes USB eszközkönyvtárakon belül, amelyekre az ***lp0/device*** és az ***lp1/device*** közvetett hivatkozások mutatnak, számos USB-vel kapcsolatos információt találunk, ilyen például a gyártó azonosítója valamint a (remélhetőleg egyedi) sorozatszám. Amint az a fenti meghatározás állományaiból kiderül, a ***/dev/usb/lp0*** eszközállomány a HXOLL0012202323480 sorozatszámú USB nyomtatóhoz tartozik, míg a ***/dev/usb/lp1*** eszközállomány a W09090207101241330 számú nyomtatónak felel meg. Ha ezeket áthelyezzük, például mindkettőt egy USB elosztó mögé rakjuk, könnyen előfordulhat, hogy átneveződnek, hiszen induláskor más sorrendben próbáljuk őket elérni:

```
$ tree /sys/class/usb/
/sys/class/usb/
|-- lp0
|   |-- dev
|   |-- device ->
../../../devices/pci0/00:09.0/usb1/1-1/1-1.1/
➥1-1.1:0
|   `-- driver -> ../../../bus/usb/drivers/usblp
`-- lp1
    |-- dev
    |-- device ->
../../../devices/pci0/00:09.0/usb1/1-1/1-1.4/
➥1-1.4:0
    `-- driver -> ../../../bus/usb/drivers/usblp

$ cat /sys/class/usb/lp0/device/serial
W09090207101241330
$ cat /sys/class/usb/lp1/device/serial
HXOLL0012202323480
```

Mint a leírásból is látszik, az eltérő próbálkozási sorrend következtében a ***/dev/usb/lp0*** eszköz most a W09090207101241330 sorozatszámú USB nyomtatóhoz tartozik. A `sysfs` segítségével viszont a felhasználó meg tudja állapítani, hogy a rendszermag melyik eszközhöz melyik eszközfájlt rendelte. Ez pedig nagyon hatékony összerendelés, amit azelőtt nem lehetett egyszerűen megoldani. Mindazonáltal a felhasználót általában nem érdekli, hogy a ***/dev/usb/lp0*** és a ***/dev/usb/lp1*** felcserélődött, és most ezért valamilyen beállításfájlban meg kéne változtatni valamit. A felhasználó mindössze annyit szeretne, hogy a helyes nyomtatóra nyomtasson, függetlenül attól, hogy az hol helyezkedik el éppen az USB fán.

Túlméretes /dev

A legtöbb terjesztésben a ***/dev*** könyvtár nem minden eszközállományához tartozik ténylegesen a géphez csatlakoztatott fizikai eszköz. A ***/dev*** könyvtár ugyanis az operációs rendszer telepítésekor jön létre, amikor belekerül az összes ismert név. Egy Red Hat Fedora 1 verziót futtató gépen a ***/dev*** könyvtár nem kevesebb mint 18,000 különféle bejegyzést tartalmaz. Ilyen mennyiségű bejegyzés hamar kényelmetlenné válik, ha a felhasználó meg szeretné állapítani, milyen eszközök vannak éppen jelen.

devfs

A ***/dev*** könyvtárban található számtalan eszközállomány miatt sok operációs rendszer inkább magára a rendszermagra bízza a könyvtár kezelését, hiszen az mindig pontosan tudja, milyen eszközök vannak jelen. Ezt pedig a `devfs` nevű, memória alapú fájlrendszeren keresztül érhetjük el. A Linux is rendelkezik ilyen megoldással, amely idővel több különféle terjesztésben, például Gentoo Linuxban, vált közkedveltté.
A `devfs` sok ember számára jelent megoldást a közvetlen problémákra. Ugyanakkor a Linux-alapú `devfs` megoldásnak máig van néhány megoldatlan problémája. Legnagyobb hiányossága, hogy nem képes állandó neveken létrehozni eszközcsomópontokat.

Az udev céljai

A korábban bemutatott problémák miatt indult el az `udev` projekt. Célja, hogy a felhasználói térben fusson, dinamikus ***/dev*** könyvtárat hozzon létre, egységes eszközneveket használjon ha szükséges, és felhasználó térbeli API-t biztosítson az aktuális rendszereszközök adatainak eléréséhez. Az `udev` és a `devfs` összehasonlításáról bővebben olvashatunk a hálózati források részben.
Az első pontot, a felhasználói térben való működést két lehetőség kihasználásával tudja teljesíteni. Egyrészt, az ***/sbin/hotplug*** eseményeket generál valahányszor egy eszközt eltávolítunk vagy hozzáadunk a rendszerhez, másrészt a `sysfs` bármilyen szükséges információt meg tud adni a kiválasztott eszközről.
A második pontot, azaz a `dynamic /dev` szolgáltatást, úgy éri el, hogy valamennyi ***/sbin/hotplug*** eseményt elfogja, a `sysfs`ből kikeresi a felvett eszközhöz tartozó major és minor számokat, majd az eszközhöz tartozó rendszermag névvel létrehozza a ***/dev*** állományt. Ha az eszközt eltávolítottuk, az eszközhöz tartozó ***/dev*** bejegyzést már nem túl nagy feladat törölni.
Az `udev` e két első célját már 2003 áprilisában képes volt végrehajtani mégpedig egészen apró, lefordított állapotban mindössze 6Kb kód segítségével, annak köszönhetően, hogy a hot-plug események elfogására és a `sysfs` használatára épülő elképzelés jól megvalósítható és egyszerűen elkészíthető volt. A 2003-as szerény kezdeteket követően az `udev` végül valamennyi célját elérte. Felhasználóinak rugalmas szabály-alapú rendszer alapján teszi lehetővé, hogy az eszközökhöz állandó neveket rendeljenek.
Az `udev` szabályait a ***/etc/udev/udev.rules*** állományban találjuk, amely valamennyi olyan eszközt tartalmazza, amelyet a felhasználó az alapértelmezett névtől eltérő módon szeretne megadni.
Nézzünk egy ***udev.rules*** állomány példát:

```
# ha a  /sbin/scsi_id visszatérési értéke
# “OEM 0815” az eszközt
# disk1-nek nevezzük
BUS=”scsi”, PROGRAM=”/sbin/scsi_id”,
➥RESULT=”OEM 0815”, NAME=”disk1”
```

```
# USB nyomtatót lp_color-nak nevezzük
BUS="usb", SYSFS_serial="W09090207101241330",
➥NAME="lp_color"

# Az adott gyártó és modellszámmal ellátott  SCSI
# lemezt boot-nak nevezzük
BUS="scsi", SYSFS_vendor="IBM",
➥SYSFS_model="ST336", NAME="boot"

# A 00:0b.0 azonosítóju PCI hangkártyát dsp-nek
# nevezzük
BUS="pci", ID="00:0b.0", NAME="dsp"

# A második elosztó harmadik kapujára csatlakozta-
# tott USB egeret
# mouse1-nek hívjuk
BUS="usb", PLACE="2.3", NAME="mouse1"

# A ttyUSB1 eszközt mindig pda-nak nevezzük
# két további közvetett hivatkozással
KERNEL="ttyUSB1", NAME="pda",
➥SYMLINK="palmtop handheld"

# USB webkameráinkat és a hozzájuk tartozó
# hivatkozásokat
# webcam0, webcam1, … névvel illetjük
BUS="usb", SYSFS_model="XV3", NAME="video%n",
➥SYMLINK="webcam%n"
```

Az udev szabályok az eszköz tulajdonságai és a kívánt eszköznév közti összerendelést adják meg. Az összerendelés megadásához számos kulcsadatot kérdezhetünk le az eszközről. Ha az ***udev.rules*** állományban nem talál megfelelő bejegyzést, az alapértelmezett rendszermag nevet fogja felhasználni. Az udev által megértett kulcsok listáját alább olvashatjuk :

- BUS: az eszköz busztípusára keres; példák: PCI, USB vagy SCSI.
- KERNEL: a rendszermag által adott névre keres.
- ID: a busz eszközszáma; például a PCI busz ID vagy az USB eszköz ID.
- PLACE: A buszon elfoglalt topológiai helyre keres, mint például a fizikai kapu amelyre az USB eszközt csatlakoztattuk.
- SYSFS_fájlnév, SYSFS{ fájlnév} : segítségével az udev bármilyen sysfs eszközjellemzőt kikereshet, például a címkét, gyártót, az USB sorozatszámot vagy a SCSI UUID értéket. Egyetlen szabályban maximum öt sysfs állományt vizsgálhatunk, melyek mindegyikének teljesülnie kell a szabály érvényesüléséhez.
- PROGRAM: az udev külső programot is meghívhat és ellenőrizheti az eredményt. A kulcs a program sikeres visszatérése esetén lesz érvényes. A program által visszaadott karaktersorozatot aztán tovább vizsgálhatjuk a RESULT kulccsal.
- RESULT: Az utolsó PROGRAM hívás eredményét teszteli. Ezt a kulcsot a PROGRAM hívás utáni bármelyik kulcsban használhatjuk.

A különféle kulcsok után a NAME és az elhagyható SYMLINK szavakat adhatjuk meg. A szabály teljesülésekor az udev a NAME részben megadott nevet használja fel az eszköz neveként, míg a SYMLINK, amennyiben létezik, a szükséges további közvetett hivatkozásokat határozza meg. Szóköz karakterekkel elválasztva egyszerre több hivatkozás is megadható. A */dev* könyvtár egyszerű szerkezetének megtartása érdekében a NAME és a SYMLINK állományok könyvtárneveket is tartalmazhatnak.

Példa

Térjünk kicsit vissza a két nyomtatós példánkhoz. Ha mindkét nyomtatót egyedi névvel szeretnénk ellátni, a következő két udev szabályt használhatjuk:

```
BUS="usb", SYSFS_serial="W09090207101241330",
➥NAME="lp_color"
BUS="usb", SYSFS_serial="HXOLL0012202323480",
➥NAME="lp_plain"
```

A fenti szabályok hatására az udev mindkét nyomtatót megpróbálja megkeresni a sysfs fájlsorozatában, majd a fájl értékétől függően lp_color vagy lp_plain névvel látja el az eszközt. Ezáltal, függetlenül attól, melyiket csatlakoztattuk előbb mindkét nyomtató ugyanazt az állandó nevet kapja. Az sem okoz zavart, ha más USB nyomtatót is csatlakoztatunk.

Különleges udev szabályok

Az udev alatt a ***udev.rules*** állomány NAME, SYMLINK és PROGRAM mezőinek megadásához különféle printf-szerű szöveghelyettesítést is használhatunk. A használható mezők a következők:

- %n: az eszköz rendszermag száma; például az sda3 eszköz rendszermag száma 3.
- %k: az eszköz rendszermag-neve.
- %M: az eszköz rendszermag szerinti major száma.
- %m: az eszköz rendszermag szerinti minor száma.
- %b: az eszköz sín azonosítója.
- %c: a PROGRAM által visszaadott szöveg. A szövegből kiemelhetjük valamelyik szót, ha a módosítót számmal egészítjük ki. Ez a mező magától értetődő módon a PROGRAM mezőben nem működik.
- %%: maga a % karakter.

Ezen kívül bizonyos kulcsokkal héjprogramszerű mintakeresést végezhetünk. A keresőminták a következők:

- *: nulla, egy vagy több karaktert keres.
- ?: egy darab tetszőleges karakter, amely nem lehet nemlétező (nulla hosszú).
- [ ]: a zárójelek közt felsorolt bármelyik karakter; például a tty[SR] minta egyaránt megtalálná a ttyS és a ttyR szöveget. A keresés a - jel használatával támogatja a tartományokat is. Például ha valamilyen számot keresünk, a [0-9] mintát használjuk. Amennyiben a [jelet a ! karakter követi, csak az itt fel nem sorolt karakter lesz érvényes találat.

A fent bemutatott szöveghelyettesítési és szövegkeresési módszerek alkalmazása, valamint az, hogy az udev tetszőleges programot képes futtatni majd eredményét felhasználni, rendkívül rugalmas eszközzé teszik az eszköznév kiosztás terén. E hatékonyság érzékeltetésére nézzük meg a következő szabályt:

```
KERNEL="[hs]d[a-z]", PROGRAM="name_cdrom.pl %M %m",
➥NAME="%1c", SYMLINK="cdrom"
```

A fenti szabály bármely blokkos eszközt megtalál, majd az eszköz major és minor számával meghívja a `name_cdrom.pl` Perl parancsfájlt. Ha a program sikeresen lefutott, az `udev` a kimenet első szavát használja fel eszköznévként, majd létrehozza a `cdrom` közvetett hivatkozást.
A `name_cdrom.pl` parancsfájlt megtaláljuk az `udev` terjesztésben. A parancsfájl feladat a következő: megállapítja, hogy az eszköz CD-ROM eszköz-e. Amennyiben igen, a Free CDDB adatbázisra küldött lekérdezéssel ellenőrzi, hogy az eszközben használt CD-ROM létezik-e az adatbázisban. Amennyiben létezik, ennek alapján nevezi el a CD-t. Például a saját */dev* könyvtáram a szabály alkalmazását követően a következőképpen néz ki:

```
$ ls -l /dev/S* /dev/cdrom
brw-------  1 root root 22, 64 Feb 15 08:26
➥/dev/Samiam-Astray
lrwxrwxrwx  1 root root      8 Feb 15 08:26
➥/dev/cdrom ->
/dev/Samiam-Astray
```

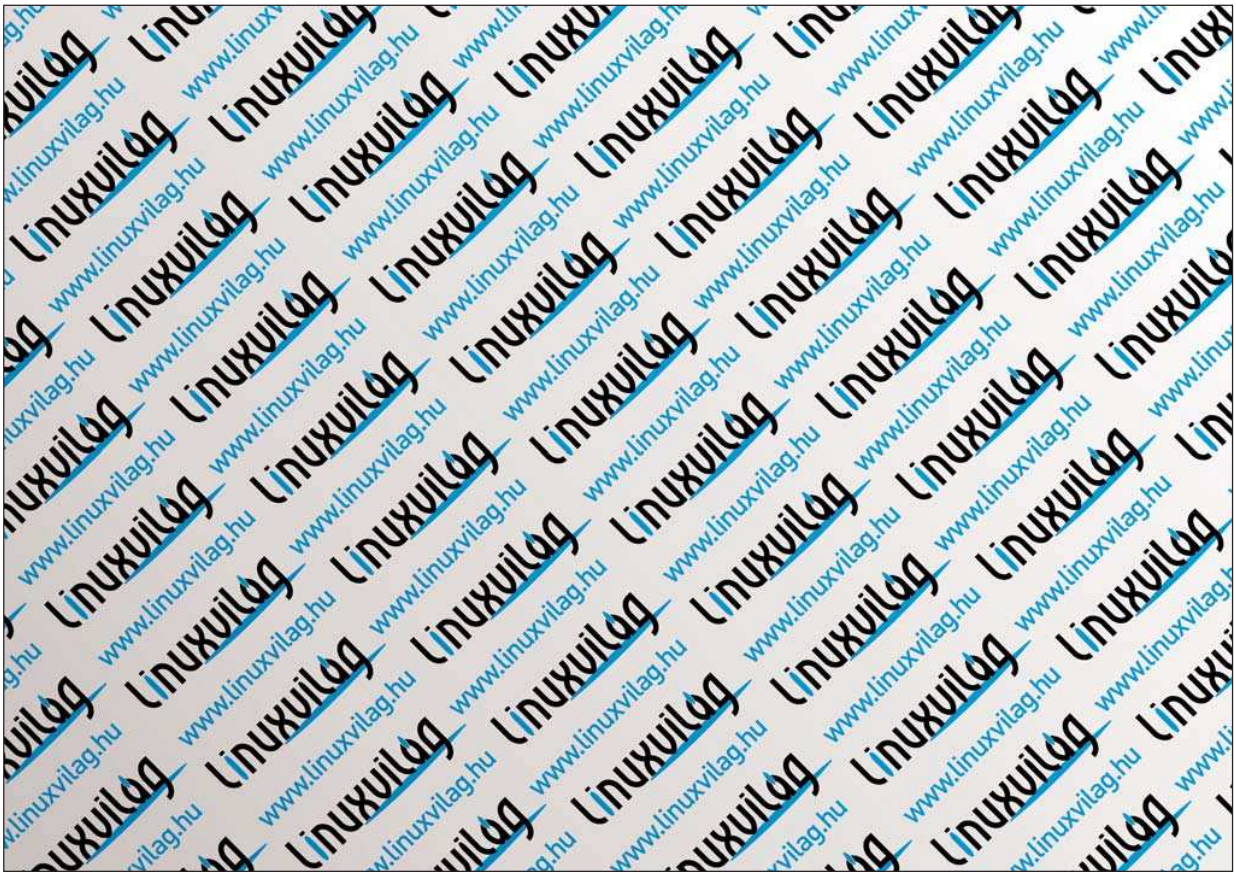
Megfigyelhettük hogyan képes az `udev` internetes adatbázisok alapján elnevezni az eszközünket.
Igen, ez valóban kicsit őrült névhasználati szabály, de jól példázza mire képes és milyen rugalmas tud lenni az `udev`.

Köszönetnyilvánítás

A szerző szeretne köszönetet mondani ***Daniel Stekloff***-nak az IBM munkatársának, aki segített formába önteni az `udev` rendszert. Az ő kitartása nélkül az `udev` nem jöhetett volna létre. Valamint ***Kay Sievers***-nek, aki az `udev` különleges képességeinek kialakításában játszott fontos szerepet, különös tekintettel a karaktersorozat módosítókra és mintakeresésre, melyek nélkülözhetetlenek az `udev` éles használatához. Segítsége nélkül az `udev` közel sem lenne olyan hatékony és használható mint amilyen manapság. ***Pat Mochel*** `sysfs` és meghajtó modell-magja nélkül az udev szintén nem lett volna megvalósítható. A szerző lekötelezve érzi magát, hogy magára vállalta amit a legtöbben megvalósíthatatlannak tartottak és, hogy lehetővé tette másoknak az egységes keretrendszerére való építkezést, lehetővé téve minden felhasználónak a hogy átlássák a rendszermagban használt „belőtt pók szőtte hálót" („Web woven by a spider on drugs", a rendszermag-meghajtókat nyilvántartó adatszerkezet neve – a ford.). A cikk az 2002-es Ottawa-i Linux Symposium udev témájú publikációja nyomán készült (lásd a forrásokat).

Linux Journal 2004. június, 122. szám

Greg Kroah-Hartman jelenleg több különféle meghajtó alrendszer Linux rendszermag gazdája. Az IBM-nél dolgozik, ahol Linux rendszermaggal kapcsolatos dolgokkal foglalkozik és a greg@kroah.com címen érhető el.



Osztott biztonsági rendszer Linux-telepeken

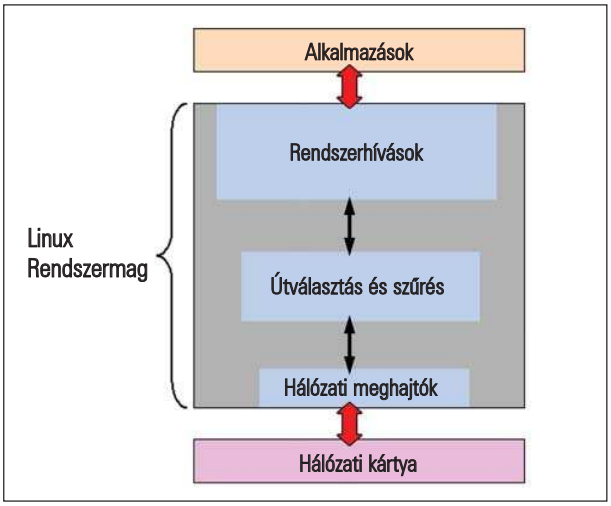
A cikkből megtudhatjuk, hogy milyen rendszermag-szintű eljárást használ az osztott biztonsági modul (DSM) a biztonsági információk IP-üzenetekbe való átlátszó-ágyazásához.

A jelen cikk az osztott biztonsági háttérről (***Distributed Security Infrastructure***, DSI) és az osztott biztonsági modulról (***Distributed Security Module***, DSM) korábban megjelent cikkek folytatása (lásd a ***Linuxvilág 2002 novemberi, 22.*** számában megjelent ***Az osztott linuxos biztonsági modell*** és a ***DSI: biztonságos Linux a távközlésben*** című cikkeket, valamint az eredeti angol nyelvű cikkeket a ***Linux Journal*** honlapján: ***„Linux Distributed Security Module"***, LJ, 2002 októberi szám, ➲ http://www.linuxjournal.com/article/6215, illetve a ***DSI: a New Architecture for Secure Carrier-Class Linux Clusters***, ➲ http://www.linuxjournal.com/article/6053). Jelen cikkben arra fókuszálunk, hogy egy osztott környezetben hogyan használjunk a DSM-ben IP beállításokat biztonsági információk küldésére a folyamatszintű biztonság megvalósításához. Kitérünk a hálózati átmeneti tár kezelésére, a kapcsok rendszermagbeli létrehozására, az IP beállításokra és az IP fejrész módosítására. Ezt követően a DSM hálózati kapcsait tárgyaljuk és bemutatjuk néhány előzetes teljesítményteszt eredményét.

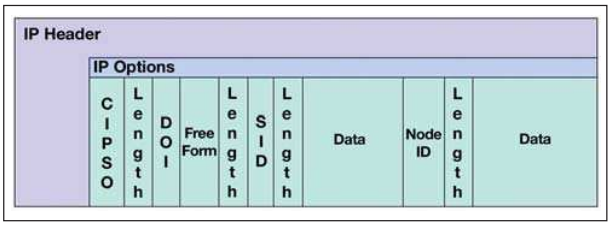
A DSI projekt

Az ***Ericsson Research*** nyílt rendszerekkel foglalkozó laboratóriuma a nyílt forrású DSI projektet azzal a céllal indította útjára, hogy egy olyan programalapú valós idejű kommunikációs rendszert fejlesszen ki, amely carrier-grade osztályú, Linux-alapú távközlési telepeken futtatható. Ezeknek a telepeknek bármilyen gép- vagy programhibától függetlenül folyamatos üzemmódban kell működniük, és lehetővé kell tenniük a karbantartók számára, hogy a programot, a berendezéseket, a rendszermagot vagy az alkalmazásokat működés közben frissítsék. Mindezt úgy, hogy az a kínált szolgáltatásokat ne befolyásolja, és ne legyen semmiféle leállás sem.

A DSI-t eredetileg az olyan carrier-grade jellemzők megvalósítására tervezték, mint a megbízhatóság, méretezhetőség, nagyfokú hozzáférhetőség és nagy teljesítmény. Ezeken felül számos más fontos jellemzővel is rendelkezik. Ilyen az egységes keretrendszer, a folyamatszintű megközelítés, és az, hogy támogatja az időosztásos biztonsági szabályokat, és a menet közben frissíthető biztonsági rendet egyaránt. A DSI fontos jellemzője a folyamatszintű hozzáférés-ellenőrzés. A jelenleg megvalósított biztonsági eljárások a felhasználói jogosultságokon alapulnak, és nem támogatják azokat a hitelesítéseket, amelyek ugyanazon felhasználóhoz tartozó két folyamat kölcsönhatására vonatkoznak akár távoli feldolgozóegységek által létrehozott folyamatok esetén. Távközlési alkalmazások esetén csak kevés felhasználó futtatja megszakítás nélkül hosszabb ideig ugyanazt az alkalmazást. A fenti megközelítés használata a különböző hálózati csomópontokon indított folyamatok mindegyikének ugyanazokat a biztonsági jogokat adja, amely ahhoz vezet, hogy a megosztott rendszeren számos művelet biztonsági ellenőrzés nélkül futhat. Az említett



1. kép A hálózati csomag útja a rendszermagban



2. kép Biztonsági beállítások az IP-fejrészben

1. lista sock_sendmsg()

```
sock_sendmsg
(struck socket *sock, struct msghdr *msg, int
➥size)
{
  int err;
  struct scm_cookie scm;

  err =
    security_ops->socket_ops->sendmsg(sock,
                                      ➥msg, size);
  if(err)
    return(err);
  ...

}
```

2. lista dsi_socket_sendmsg()

```
int dsi_socket_sendmsg(struct socket *sock,
                       ➥struct msghdr *msg, int
                       ➥size)
{
...

inode_security_t *isec;
struck sock sk;
struct ip_options *opt =  NULL;
int optlen = NSID_BASE_LEN + NSID_SSID_LEN +
             NSID_NODEID_LEN; //8 +_6 + 6
unsigned char optptr[optlen];

  ...

  sk = sock->sk;
  opt = sk->protinfo.af_inet.opt;
  dsi_options_fill (isec, optptr, optlen);
  dsi_ip_options_get(&opt, optptr, optlen);
  opt = xchg(&sk->protinfo.af_inet.opt, opt);

...
}
```

biztonsági rendszer alapegysége a felhasználó. A vivőszintű (***carrier-class***) alkalmazások esetén ez a felosztás nem megfelelő, finomabb megközelítésre van szükség, ezért támogatja a DSI is az egyedi folyamatot, mint a felosztás alapegységét.

Az osztott biztonsági modul (DSM)

A DSM a DSI központi összetevője, amely biztosítja a Linux telepen belül a kötelező érvényű hozzáférés-ellenőrzés megvalósítását. A DSM felelős a hozzáférés-ellenőrzés végrehajtásáért, és biztosítja, hogy a telep csomópontjain áthaladó IP-üzenetek el legyenek látva a küldő csomópont és folyamat biztonsági jellemzőit tartalmazó címkével. A DSM az LSM-kapcsokat használó Linux modulként került megvalósításra. A fejlesztés a 2.4.17-es Linux rendszermag használatával kezdődött egy megfelelő LSM-rendszermagfolt alkalmazásával. Ez a megvalósítás az IP-fej módosítását leíró CIPSO és FIPS 188 szabványokra épül.

A DSM megvalósításának fontos eleme az osztott jelleg. Egy telepen belüli csomópontról kiinduló hozzáférés-ellenőrzés vonatkozhat egy olyan erőforrásra, amely egy másik csomóponton helyezkedik el, ezért szükség van arra, hogy létezzen a telepen belüli csomópontok között a biztonsági információk átvitelének a lehetősége. A DSM osztott jellege biztosítja, hogy az erőforrásokhoz való hozzáférés biztonsági szempontból helyfüggetlen legyen.

A hálózati átmeneti tár kezelése

Röviden összefoglaljuk a hálózati átmeneti tár kezelését annak érdekében, hogy jobban megérthessük, hogyan ágyazódik a biztonsági információ a hálózati csomagba. Leírjuk azt is, hogy a rendszermag hogyan kezeli a hálózati tárat a felhasználói rétegtől a hardverrétegig és vissza.

Az *1. kép* a hálózati csomag útját mutatja a rendszermagban. A csomagkezelés két esete a bejövő, illetve kimenő csomagok kezelése. A kimenő csomagok kezelése az alkalmazói rétegtől kezdődően a következőképpen alakul: az alkalmazás előkészíti az adatot a hálózaton keresztül történő továbbításra; az alkalmazás a csomag küldésére vonatkozó rendszerhívást küld a rendszermag felé; a csomag `sk_buff` szerkezetben áthalad a rendszermag szűrő és útválasztó függvényein; végül a csomag a hálózati meghajtóhoz kerül, amely azt a hálózati kártyára (DMA) továbbítja.

A bejövő hálózati csomag útja a hálózati kártyától kezdődően oly módon indul, hogy a kártya begyűjti a saját vagy csoportos (***broadcast***) címével megcímzett csomagot; ezután beolvassa a hálózati memóriába, és egy megszakítást hoz létre. A hardveres megszakítás által kiváltott és a hálózati kártya meghajtóprogramjának részeként a rendszermagban elhelyezkedő megszakítás-kiszolgáló rutin lefoglal egy `sk_buff` területet, majd a hálózati kártya memóriájából (DMA) ebbe az átmeneti tárba mozgatja át az adatokat. Ezt követően a csomag a felsőbb rétegbeli feldolgozás céljából a processzor várakozási sorába kerül, a feldolgozás pedig akkor folytatódik, amikor a megszakítás engedélyezetté válik. Végül a csomagok keresztülhaladnak a szűrőkön és útválasztó függvényeken, és az alkalmazói réteg felé továbbítódnak.

Most, hogy már ismerjük az általános módszert, amivel a Linux rendszermag kezeli a hálózati átmeneti tárat, bemutatjuk, hogy mindezt hogyan használhatjuk fel a rendszermag biztonságának növelésére.

Megvizsgáljuk az IP útválasztó függvényekhez létrehozott kapcsokat, amelyek lehetővé teszik az IP-csomagok befolyásolását és növelik az IP-üzenetek biztonsági lehetőségeit.

Hálózati biztonsági kapcsok létrehozása

A biztonsági modul képes befolyásolni a biztonsági kapcsok megvalósításán alapuló útvonal kiválasztási döntéseket. Mivel az útválasztó kapcsok a rendszermagba érkező és

3. lista ip_options_compile ()

```
int
ip_options_compile (struct ip_options *opt,
                    ➥struct sk_buff *skb)
{
unsigned char *pp_ptr;
unsigned char *optptr;

  ...

  case IPOPT_CIPSO:

    if(security_ops->ip_ops->decode_options(skb,
                              ➥optptr, &pp_ptr)
      goto error;
      break;
  ...
}
```

4. lista sock_queue_rcv_skb ()

```
int
sock_queue_rcv_skb (struct sock *sk,
                    ➥struct sk_buff *skb)
{
int err=0;

  ...

  err=security_ops->socket_ops->sock_rcv_skb
                                    ➥(sk,skb);

  if(err)
    return (err);
  ...
}
```

onnan kimenő csomagok esetén normál rendszermag-függvényként futnak, ezek programozásánál fel kell idéznünk néhány dolgot.
A függvényt regisztráló modulnak rögzítenie kell a függvény kapcson belüli prioritását. A hálózati szűrő kapcsok a rendszermag-kódjából a prioritásuknak megfelelő sorrendben kerülnek meghívásra.
A felhasználói függvények szabadon módosíthatják az IP-csomagokat, és az alábbiak közül kell egy értéket visszaadniuk a hálózati kód számára, hogy az eldönthesse, mi a teendő a csomaggal:

1. NF_ACCEPT: nincs semmi tennivaló, a csomagot átengedi a hálózati vermen.
2. NF_DROP: a csomag eldobása, nincs további feldolgozás.
3. NF_STOLEN: a csomag átvéve, nincs további feldolgozás.
4. NF_QUEUE: a csomag sorbaállítása felhasználói kezelésre.
5. NF_REPEAT: a kapocs ismételt meghívása.

Ez a kód megmutatja, hogy mi történik a csomaggal, mielőtt a rendszerbe vagy onnan kiküldésre kerül.
Azt viszont még mindig nem tudjuk, hogy milyen információkat vagy beállításokat adhatunk a csomaghoz, és hogyan tehetjük azt meg. További kérdés, hogy vajon ezek a változtatások együttműködnek-e a jelenlegi megvalósítással.

IP-beállítások

Az internet-protokoll (IP) egy kevésbé ismert tulajdonsága, hogy egy IP-csomag változó hosszúságú (maximálisan 40 bájt) többletinformációt is tartalmazhat, amely a szabványos 20 bájtos fejrészt követi. Ez a kiegészítés az úgynevezett opcionális terület, amelynek egy része biztonsági információk tárolására van fenntartva.
Jelenleg az Internet Protokoll két biztonsági beállítást tartalmaz. Ezek közül az egyik a ***DoD Basic Security Option*** (alap biztonsági beállítás, 130-as beállítástípus), amely lehetővé teszi, hogy az IP-adatcsomagokat a titkosítás foka szerinti címkével lássuk el. Ez a beállítás 16 titkosítási fokozatot különböztet meg, amelyek különböző számú megszorítást tartalmaznak a csomag kezelésére vonatkozóan. További biztonsági információk kezelését, mint amilyen a biztonsági osztály és szakasz megkülönböztetése, a második biztonsági beállítás teszi lehetővé, amelyre ***DoD Extended Security Option*** (ESO, kiterjesztett biztonsági beállítás, 133-as beállítástípus) néven hivatkozik a szakirodalom. E két beállítás adatterületének értékeit az Információs Rendszerek Védelmi Irodája (***Defense Information Systems Agency***) hivatott nyilvántartani.
A számítógép forgalmazók ma már olyan kereskedelmi operációs rendszereket kínálnak, amelyek kötelező hozzáférés-ellenőrzéssel és többszintű biztonsági beállításokkal rendelkeznek, s ezek a rendszerek már nem kizárólag a védelmi vagy hírszerzési szervezetek számára készülnek, hanem olyan, nyilvánosan hozzáférhető kereskedelmi rendszereknek, amelyeket az állami és civil szféra is elterjedten használ.
A kisszámú ESO formátumkód nem támogatja a kereskedelmi biztonsági beállítások összes lehetséges alkalmazási módját. A BSO és ESO tervezésekor csak az Egyesült Államok Védelmi Minisztériumának támogatása volt a cél.
A különböző egyéb biztonsági módok támogatására a CIPSO (***Commercial IP Security Option***, kereskedelmi IP biztonsági beállítás) került kidolgozásra. Az internetes tervezet biztosítja a kötelező hozzáférés ellenőrzés (***mandatory access control***, MAC) biztonsági rendjéhez szükséges formátumot és eljárásokat.
A mi megvalósításunkban az adatcsomagok címkézésére használt IP-beállítások a FIPS 188 szabványon és a kereskedelmi IP biztonsági beállításon (CIPSO) alapulnak. Az általunk kifejlesztett rendszerben az IP fejrészt ezen szabványok szerint változtatjuk meg, hogy alkalmassá tegyük a biztonsági információk hozzáadására és hálózaton való küldésére.

A DSM IP-beállításai

Az IP-beállítások segítségével továbbítandó két biztonsági információ a biztonsági azonosító (***Security ID***, SID) és a biztonsági csomópont-azonosító (***Security Node ID***, NID). A DSM úgy módosít minden egyes IP-csomagot, hogy IP-beállításként ellátja azokat ezekkel a biztonsági információkkal. A *2. képen* látható ennek a módosított IP fejrésznek a felépítése.

A fejrész beállításainak listája a következő:

- CIPSO: 1 bájt, 134-es értékkel.
- Hossz: 1 bájt, amelynek értéke a teljes hossz, beleértve a típus és hossz mezőket is. A jelenlegi IP-fejrész 40 bájtos korlátja miatt ez az érték nem haladhatja meg a 40-et.
- Az értelmezési tartomány (DOI) azonosítója: előjel nélküli 32 bites egész. A 0 foglalt érték és nem jelenhet meg egy CIPSO beállítás DOI-azonosítójaként sem. A megvalósításoknak el kell fogadniuk, hogy a DOI-azonosítót nem köti semmilyen különleges bájtszintű korlátozás.
- A CIPSO értelmezési tartomány mező vagy a FIPS 188 szerinti biztonsági jelkészlet neve: ez 10001000 hexadecimális értékre van beállítva, ami egy önkényes érték, mivel jelenleg erre a területre nincs érvényes korlátozó szabály.
- Szabad formátum-jel: egy bájt, amely azt jelzi, hogy az ezt követő mezők olyan új mezők, amelyeket a szabvány nem határoz meg (vagyis tetszőlegesek). Értéke 7.
- Hossz: 1 bájt, a címkék teljes hosszát jelöli.
- Címkék (SID, NID): A CIPSO címkecsoportokat használ az IP-csomagban tárolt adatokra vonatkozó biztonsági információk tárolására. Minden címke egy címketípus azonosítóval kezdődik, a címke hosszával folytatódik, majd végül ezt követi a tényleges biztonsági információ.
- SID címke: címkeazonosító: 1 bájt (értéke 3), címkehossz: 1 bájt (értéke 6), címkeadat: a sid 32 bites értéke.
- NID címke: címkeazonosító: 1 bájt (értéke 6), címkehossz: 1 bájt (értéke 6), címkeadat: a nid 32 bites értéke.
- Az általunk használt IP-beállítás a CIPSO. Ezeket a mezőket a szabvány nem határozza meg, így használhatóak az általunk definiált módon.
- A DOI és a FIPS 188 szabvány szerint a következő mezők új mezők, amelyeket a szabvány nem határoz meg, ezért szabadon felhasználhatóak.

DSM hálózati kapcsok

A DSM-ben az LSM biztonsági kapcsokat használtuk fel az IP üzenetek biztonsági címkékkel történő ellátására. A következőkben ezt egy példán keresztül fogjuk szemléltetni, amelyben a program a hálózaton keresztül olyan csomagot küld, amelynek a foglalatát módosította. A program használja az eljáráskönyvtár néhány függvényhívását. Egy ponton egy rendszerhívás jön létre, amely az üzenetet a Linux rendszermagnak továbbítja. A rendszermag foglalat-megvalósításának belépési pontja a `sys_socketcall()` függvény, amely a ***net/socket.c*** fájlban található. A híváslánchan a ***net/socket.c*** fájlban lévő `sock_sendmsg()` függvény *(1. lista)* futtatása történik.

A függvény első tevékenységeinek egyike a biztonsági kapocs futtatása (***security_ops->socket_ops->sendmsg(...)***).
A kapocs a DSM foglalat-kapocsban ér véget, amely ai IP csomagot módosítja, mint az a 2. *listán* látható.
A `dsi_options_fill` függvény elindítja a biztonsági információt az átmeneti tár felé, ahogy azt az előző részben leírtuk. Az ezt követő függvények végrehajtása során kerül ez az információ az IP-üzenet beállítási részében rögzítésre. A SID a foglalat biztonsági azonosítójából származik, a NID pedig az egész csomópontban érvényes érték, vagyis nincs szükség arra, hogy a függvénynek paraméterként átadjuk.
Ezt követően a biztonsági információkat már tartalmazó módosított csomag a normál feldolgozó eljárásnak megfelelően kerül továbbításra a rendszermag felé, majd a hálózatra. A fogadó oldalon a bejövő üzenetek az `sk_buff` szerkezetnek megfelelően tárolódik és különböző függvények és kapcsok sorozata által kerül elő-feldolgozásra.
Az egyik ilyen függvény az `ip_options_compile` *(3. lista)* a ***/net/ipv4/ip_options.c*** fájlban, amely a beállításokat dolgozza fel.
A CIPSO esetén a decode_options biztonsági kapocs kerül meghívásra. Ez után következik a DSM `dsi_decode_options` kapocs, amelyben a bejövő csomag biztonsági paraméterei (SID, NID) kerülnek kiolvasásra és tárolódnak egy az sk_buff formátumhoz kapcsolódó biztonsági formátumban. A biztonsági információkkal feltöltött `sk_buff` tárolók a bejövő foglalatsorhoz csatlakoznak, és itt várjuk, hogy a fogadó program kiolvassa őket.
A kiolvasás érdekében a program egy `sys_socketcall ()` rendszerhívást bocsát ki, ugyanúgy, ahogy a küldött csomag esetén is tette. A hívás ismételten átmegy a DSM biztonsági kapcson, ahol a fogadó foglalat biztonsági azonosító érvényesítésre kerül a bejövő csomag `sk_buff` biztonsági tartalmával. Ha a foglalat a megadott biztonsági azonosítóval nem fogadhatja a csomagokat, akkor azok elvesznek.
A *4. listán* látható az *include/net/sock.h* fájlban lévő rendszermag-függvény.
Ebben láthatjuk, hogy a `sock_rcv_skb` biztonsági kapocs kerül meghívásra, amit a `dsi_sock_rcv_skb` DSM-függvény vált fel a DSM betöltődésekor. Ebben a függvényben történik a biztonsági érvényesítés. A példakódból láthatjuk, hogy milyen műveletekre van szükség a biztonsági címkék kezeléséhez.

Teljesítményvizsgálatok

Számos sebességtesztet végeztük arra vonatkozóan, hogy az IP-fejrész kibővítése a beállításainkkal befolyásolja-e az átfogó teljesítményt, és ha igen, milyen mértékben. Az egyik tesztben egy UDP-csomagot küldtünk át a telep csomópontjai között és mértük a csomag átviteléhez szükséges idő növekedését, ami a küldő oldalon a biztonsági beállítások módosításából, a fogadó oldalon pedig ezek kinyeréséből adódhatott.
A megvalósításunkon alapuló biztonsági információk hozzáadása okozta többletmunka átlagos értéke 30% volt. Ennek nagy részét (mintegy 25%-ot) az IP-csomag IP-beállításain végzett módosítása tette ki, a maradék többletmunka (körülbelül 5%) pedig a Linux rendszermag biztonsági kapocs-rendszeréből származik. Látható, hogy a többlet nagy rész az IP-csomag beállításainak a módosításából származik és csak kisebb rész írható a biztonsági kapcsok rendszerének számlájára.
A jövőbeli erőfeszítéseink arra fognak irányulni, hogy növeljük IP-módosító algoritmusunk hatékonyságát, miközben továbbra is az IP-beállításokat tekintjük a biztonsági információk átviteli eszközének.

Összegzés

Az IP-beállítások megváltoztatásával képesek voltunk arra, hogy a DSM segítségével biztonsági információkat továbbítsunk a telep csomópontjai felé. Az IP-csomagok módosításának optimalizálásával már első próbálkozásra is jelentős teljesítménynövekedést értünk el: a 30%-os teljesítménycsökkenést sikerült 14%-ra leszorítani. Ezek az eredmények igen ígéretesek, több olyan lehetőséget is látunk a további tökéletesítésre, amellyel még kisebb többletmunkával érhetjük el a célunkat. Mindenesetre az eredmények jól mutatják azokat a kihívásokat, amelyekkel egy hatékony osztott biztonsági rendszer fejlesztése során szembe kell néznünk. Reméljük, hogy minél többen próbálják majd ki az általunk kifejlesztett DSI és DSM technológiát és megosztják velünk a tapasztalataikat.

Köszönetnyilvánítás

Köszönjük David Gordonnak a Sherbrooke Egyetem munkatársának a DSM kifejlesztéséhez való hozzájárulását.

Linux Journal 2004. április, 120. szám

Ibrahim Haddad a Linux Journal társszerkesztője, a montreali Ericsson Research & Innovation Unit kutatója. Társszerzője a McGraw-Hill/Osborne által kiadott két Richard Peterson-könyvnek: Red Hat Linux Pocket Administrator és Red Hat Enterprise Linux & Fedora Edition: The Complete Reference (DVD edition).

Miroslaw Zakrzewski az új generációs CDMA rendszerek kifejlesztésén dolgozik az Ericssonnál a kanadai Montrealban. Elérhető a Miroslaw.Zakrzewski@Ericsson.ca címen.

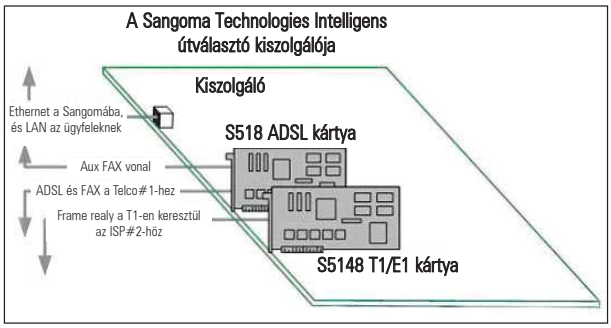
KAPCSOLÓDÓ CÍMEK

A DSI és DSM honlapja:
➲ www.linux.ericsson.ca/dsi
A FIPS 188 szabvány:
➲ csrc.nist.gov/publications/fips/fips188.html
Cikkek a Linux csomagszűrőjéről:
➲ www.linuxjournal.com/article/4852 és
➲ www.linuxjournal.com/article/5617
Az LSM: ➲ lsm.immunix.org
Hálózati átmeneti tárak:
➲ www.linuxjournal.com/article/1312
Az Open System Lab oldala: ➲ www.linux.ericsson.ca
A SE Linux honlapja: ➲ www.nsa.gov/selinux

Intelligens útválasztás: szórakozás és anyagi haszon

Biztosítsuk magunknak a szükséges sávszélességet anélkül, hogy a hónap végén hanyatt esnénk a számlától.

A Sangoma PCI felületű WAN-kártyákat gyártó cég. Bemutató és biztonsági tartalék céljából két különálló szélessávú Internet-kapcsolattal rendelkezünk: egy T1-es ***Frame Relay*** kapcsolattal, amely a PCI-os S5148 T1/E1 modemünket használja, és egy PPPoE protokollos ATM-re épülő ADSL kapcsolattal, amely a PCI felületű S518 ADSL modemünket veszi igénybe. Az ADSL-vonalat a faxgépünkkel osztottuk meg, amely az egyetlen olyan telefonvonal, ami nem kapcsolódik a PBX-ünkre (házi telefonközpont). A vonalakat két külön szolgáltató biztosítja. Az ADSL-t és a faxkészülék telefonvonala a Bell Canada Sympatico szolgáltatása, a T1 Frame Relay kapcsolatot pedig az MCI-tól vesszük igénybe.



1. kép Takarékossági és üzembiztossági megfontolások alapján a kiszolgáló mind T1, mind ADSL kapcsolattal rendelkezik

A sávszélesség és annak költségei

A telepített T1 és ADSL-kapcsolat együttese valóban megbízható kapcsolatot biztosít, viszont a kapott sávszélesség több, mint amire szükségünk van.

A Sangoma honlapjának az atlantai Earthlink ad helyet, így a világhálóval kapcsolatos igényeink nagyjából megegyeznek bármilyen más cég igényeivel. Elsősorban elektronikus levelezéssel és Web-eléréssel kapcsolatosak, ami kiegészül némi FTP-forgalommal. Utóbbi főleg a weboldalunkon lévő FTP-kiszolgálóra feltöltött anyagokkal kapcsolatos. Meglennénk állandó IP-címek nélkül is, de azért hasznosnak találjuk, hogy a T1 kapcsolatunk rendelkezik egy rögzített IP-címtartománnyal.

Az összes Internetes kiszolgálónkon Linux fut. Bár támogatjuk a Windows, FreeBSD, Solaris és még más népszerű operációs rendszereket is, a legfontosabb a Linux, és csak ez az operációs rendszer rendelkezik a számunkra szükséges gazdag forgalomkezelő eszközkészlettel. Az *1. képen* látható a leírt elrendezés felépítése.

Az ADSL-vonal nem kerül sokba, különösen azzal az engedménnyel együtt, amit annak fejében kaptunk, hogy saját ADSL-modemet használunk, ez normális körülmények közt a szolgáltatás részét képezi. A T1-vonal költségei viszont magasak, ha az Egyesült Államokhoz hasonlítjuk: egy korlátlan T1 Internetes kapcsolat ára elérheti a havi 1.900 kanadai dollárt (1.450 amerikai dollár).

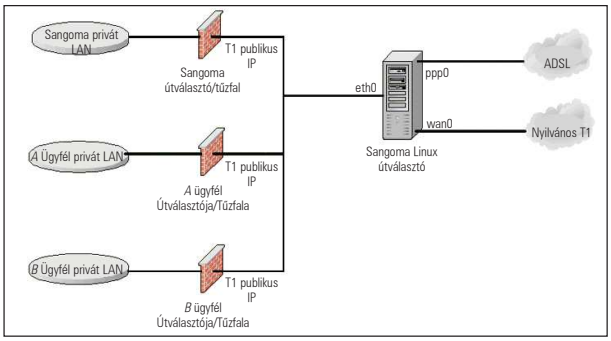
A Sangoma az Internetes hozzáférését a költségek valamelyes ellensúlyozása érdekében tovább értékesíti az épület két másik bérlőjének. A kapcsolatunkat megosztó bérlők Webhely- és VPN szolgáltatásokat nyújtanak, így nekik rögzített IP-címekre és nagy kimenő sávszélességre van szükségük, emiatt a T1-vonal megosztásában érdekeltek. A T1-vonal belső és nyilvános szegmenseit a *2. kép* mutatja.

A Sangoma két Linuxos gépe könnyedén összeolvasztható eggyé. Az így létrejövő útválasztó rendelkezne egy újabb hálózati csatolóval az A és B ügyfél nyilvános hálózati szegmenseinek támogatására, a Sangoma tűzfala pedig a Sangoma belső hálózati szegmense és az összes többi nyilvános szegmens közt helyezkedne el (magában foglalná a Frame Relay T1 kapcsolatot, az ADSL kapcsolatot és a nyilvános Ethernet kapcsolatot).

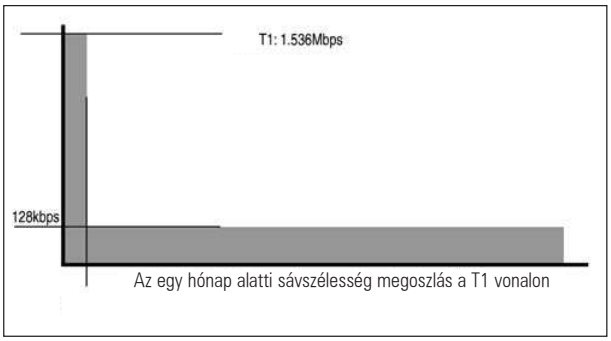
Az ügyfelektől kapott anyagi hozzájárulás nem elegendő arra, hogy kifizessük belőle a T1 kapcsolat teljes kanadai árát. A megoldást az jelentette, hogy a T1 használatán alapuló szolgáltatást választottunk. Ez az úgynevezett ***burstable*** T1 szolgáltatás, amely a teljes sávszélességű T1-nek csak körülbelül a felébe kerül. A T1 korlátlanul használható a teljesen kétirányú 1,536 Mbps sávszélességig.

A számlázás a használt sávszélesség-érték 95 százalékos értékén alapul.

A forgalmat ötpercenként mintavételezik és kiszámolják az öt perc átlagosan használt sávszélességét. A hónap végén ezeket az ötperces értékeket átviteli sebesség alapján csökkenő sorrendbe rendezik. A legmagasabb 5%-ot figyelmen kívül hagyják, a fizetendő díj alapja pedig a következő legnagyobb sávszélesség-érték. Az átvitel határa esetünkben



2. kép Az épület két bérlője vesz igénybe Internet-hozzáférést a Sangoma-tól



3. kép A lehetséges legkisebb költség biztosítása céljából az ideális megoldás szerint a T1 vonal csak az idő 5%-ban üzemel teljes sávszélességen

128 kbps. Ha a 95. százalékértékünk meghaladja ezt a 128 Kbps értéket, a díj legalább 300 dollárral megemelkedik. Az előfizetők nehezen értik meg ezt a bonyolult számlázási rendszert. Az ügyfél számára jó üzletnek tűnik, de a kezelése bonyolult, és nehéz mérni az értékeket.

számlázás alapjaként szolgáló értéket az 5%-os szintnél mérik, ahol a felhasználás görbéjének változása a maximum körül van. Sok felhasználó fizet olyan számlákat, amelyek magas értékét csak egy-két olyan ötperces szakasz okozza, amelyek kilógnak a havi több mint 8500 mérési értékből.

Hacsak nem egyenletesen alacsony a forgalmunk, könnyen azt tapasztalhatjuk, hogy a pótdíjak ilyen rendszere mellett ráfázunk és kifizetjük a teljes T1 árát, holott az átlagos átvitelünk jóval 128kbps alatt volt.

A rendszer legfőbb előnye, hogy a legmagasabb sávszélességű 5% felhasználása minden hónapban „ingyenes”. Ez havonta körülbelül 36 tetszőleges sávszélességű órát jelent mindenféle büntetés nélkül, vagyis egy hónapban majdnem egy hét munkaóráit tölthetjük a teljes sávszélesség kihasználása mellett. A *3. képen* látható az ideális forgalommegosztás, amely az általunk választott fizetési rendszer mellett a legkevesebb díj fizetésével elérhető legnagyobb átvitelt tenné lehetővé. A forgalomirányító logika lényegében 128 kbps sebességre korlátozná a sávszélességet, miután az adott hónapra eső első 36 óra korlátlan sávszélességét felhasználtuk.

Nos, hogyan lehetne megszerezni a csalit anélkül, hogy vele együtt a horgot is bekapnánk? A megfelelő ***policy routingot*** (intelligens útválasztást), IP-fiókkezelést és forgalom ellenőrzést megvalósító parancsfájlok és démonok együttesével intelligensen szabályozhatjuk a hálózati forgalmat, így a saját és társfelhasználóink számára elérhetővé válik a legnagyobb teljesítmény a legalacsonyabb díj mellett.

Intelligens útválasztás IP-táblák és az iproute2 segítségével

Az első lépés, hogy minden olyan forgalmat leveszünk a T1-ről, amely átterhelhető az ADSL-re anélkül, hogy ezzel a szolgáltatás színvonala csökkenne. Az ADSL-vonalunk legnagyobb letöltési sebessége 1.728 kbps, a feltöltésé pedig 800 kbps. A T1 névleges sebessége 1.536 kbps, teljesen kétirányú. Az ADSL-vonal az ATM magas hibajavításból adódó többletterhelés miatt kevésbé hatékony, mint a ***Frame Relay*** elvén működő T1. Vagyis a hasznos átvitel tekintetében a bejövő, azaz a letöltési sebesség a T1 és az ADSL-vonalon hasonló jellemzőkkel bír.

Abban a szerencsés helyzetben vagyunk, hogy az ADSL-vonalunk jó csatlakozástöbblet-aránnyal* bír, ezért a teljesítmény egyenletesebb, mint sok hasonló kapcsolaté (*A csatlakozástöbblet-arány – oversubscription – a szolgáltatók által eladott sávszélesség összegének és a szolgáltató Internetre csatlakozó sávszélességének aránya – a fordító megjegyzése). Ez az arány a központi irodában akár 200-300 szoros is lehet, ami csúcsidőszakban igen gyenge átvitelt eredményez. Még a mi majdnem tökéletes ADSL kapcsolatunk esetén is igaz, hogy a tényleges feltöltési sebesség a T1-ének kevesebb mint fele, ami ésszerűvé teszi a bejövő forgalmat az ADSL csatlakozáson bonyolítani, míg a T1-et inkább megtartani a kimenő adatok számára. A sebességbeli különbségeken túl van még egy lényeges eltérés a Frame Relay T1 és az ADSL-vonal között, mégpedig az, hogy a T1 egy kis fix IP-címtartományt is biztosít, míg az ADSL az IP-címét a DHCP kiszolgálótól kapja. Legalább azoknak a szolgáltatásoknak a T1-vonalon kell lenniük, amelyek számára szükséges, hogy támogassák a rögzített IP-címre érkező kéretlen bejövő forgalmat. Ilyenek például a webkiszolgálók. A nagy mennyiségű adatletöltéssel járó forgalmat nagyrészt a böngészés, levélforgalom és a bejövő FTP-forgalom teszi ki, amelyet a nagy letöltési sávszélességgel rendelkező ADSL sikerrel tud kezelni. A kivétel a kimenő SMTP forgalom, amely ki tudja használni a Frame Relay T1-vonal így felszabaduló sávszélességét.

Az A és B ügyfelek három kiszolgálógéppel rendelkeznek. Ezek közül egy Webkiszolgáló, amely rögzített IP-címmel kell rendelkezzen és főleg kimenő forgalmat generál. A másik egy csekély forgalmat bonyolító VPN-kiszolgáló, amelynek szintén rögzített IP-címre van szüksége. Ennek a két kiszolgálónak minden forgalma a T1 rögzített IP-című vonalát veszi igénybe. A Sangoma forgalomelosztási megoldása több szakaszból álló folyamat, melynek során a kimenő csomagok egy sereg szabályon és intézkedésen keresztülhaladva érik el a legjobb forgalomszétosztást. Csak a kimenő csomagok elosztása történik meg a két vonal között, mivel a bejövő forgalom útvonalát nem tudjuk ellenőrizni. Igaz viszont, hogy ha egy csomag egy meghatározott vonalon, az ADSL-en vagy T1-en elhagyja a rendszert, a rá adott válasz is ugyanazon a felületen fog megérkezni.

1. lista Többszörös útválasztó tábla

```
cat /etc/iproute2rt_tables
#
# reserved values
#
#255    local
#254    main
#253    default
#0      unspec
# local
#1      inr.ruhep
200 adsl
```

A Linux alatt elérhető fejlett útválasztó eszközök és segédprogramok módot adnak arra, hogy a megfelelő hálózatkezeléssel elérhessük céljainkat. A Linux rendszermag támogatja a többszörös útválasztó táblákat, lehetővé téve, hogy minden egyes fizikai kapcsolat saját útválasztó táblával rendelkezzen. Ha már rendelkezünk a két fizikai felületünk számára a különálló táblázatokkal, az `iptables` és `iproute2` programokkal irányíthatjuk a forgalmat a megfelelő útválasztó táblára. Innen a csomagok az alapértelmezett utat követve jutnak a megfelelő fizikai felületre.
Az `iproute2` programcsomag egy beállítófájllal rendelkezik az útválasztó táblák és a Linux útválasztó vermének megfeleltetésére. Alapértelmezésben a `tr_tables` egyetlen útválasztó tábla meghatározást tartalmaz, amelynek neve `main`. Ez az alap útválasztó tábla, amelyet a Linux útválasztó verme használ. Az *1. listában* látható az általunk az ADSL-vonalhoz hozzáadott `adsl` nevű útválasztó tábla bejegyzés. Ehhez az útválasztó táblához szabványos Linux parancsok segítségével adtunk hozzá egyedi útvonalakat. A kimenő csomagoknak az útválasztó bemenete és kimenete között hat szinten kell áthaladniuk.

Az Ethernet-hálózat felől érkező bemenet

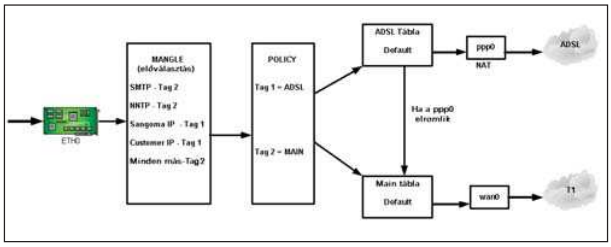
Az első lépés az `iptables` mangle-szabályának alkalmazása, melynek során a forgalom vagy `Tag 1` címkét kap, amely az ADSL-t jelenti, vagy `Tag 2`-t a T1 számára. A Sangoma összes levelének `Tag 2`-vel való megjelöléséhez például a következő szabályt alkalmazzuk:

```
iptables -t mangle -A PREROUTING -i eth0
➥-p tcp -s xxx.xxx.xxx.82 --dport smtp -j t1_line
```

Ezután az `iptables --set-mark` kapcsolóját használjuk a `t1_line` láncon:

```
iptables -t mangle -N t1_line
iptables -t mangle -A t1_line -j MARK --set-mark 2
iptables -t mangle -A t1_line -j ACCEPT
```

Hasonló szabályaink vannak az ADSL-vonalra küldött forgalom számára is.
Az `iproute2 a Tag 1` címkéjű csomagokat az ADSL útválasztó táblájára, a `Tag 2` címkéjűeket pedig a `main` útválasztó táblára irányítja, ami a T1-re irányítja a forgalmat:



4. kép A címkézés és az intelligens útválasztás lehetővé teszi, hogy az ADSL meghibásodása esetén a T1 vonal vegye át a forgalmat

```
ip rule del fwmark 1 table adsl
ip rule add fwmark 1 table adsl
ip rule del fwmark 2 table main
ip rule add fwmark 2 table main
```

Az útválasztó táblák

Az ADSL útválasztó táblájának alapértelmezett útvonala a `ppp0`, amely a PPPoE kapcsolatot képviseli. Az Ethernet ezután ATM-be (EoA) ágyazva folytatja útját, és ezek az ATM-csomagok haladnak keresztül az ADSL kapcsolaton a DSLAM felé.
Amennyiben a `ppp0` csatlakozóval valamilyen gond adódna, az ADSL alapértelmezett útvonalát a rendszermag önműködően eltávolítja, helyébe pedig a `main` útválasztó tábla lép. Így az ADSL kapcsolat meghibásodása esetén az összes ADSL-vonalnak szánt forgalom a feltehetően megbízhatóbb `main` útválasztó tábla felé kerül átirányításra.
Rendszeresen előfordulnak olyan ADSL-üzemszünetek, amelyek az ilyen alacsony árú, ellenőrizetlen sávszélességű szolgáltatások velejárói. Az üzemszünetek hossza néhány másodperctől több óráig is terjedhet, de ez a felhasználók számára nem jelent hátrányt, mert a forgalom észrevétlenül átterhelődik a T1-vonalra.
A T1 felület az ADSL jó biztonsági tartaléka, mindez azonban fordítva már nem igaz. A T1 használatának sok gép esetén az az oka, hogy rögzített IP-címre van szükség, vagyis a változó IP-című ADSL nem alkalmas ilyen szolgáltatás nyújtására. A `main` útválasztó tábla alapértelmezett útvonala a `wan0` (T1). Minden olyan forgalom, amely erre az útválasztó táblára kerül, a T1-re lesz továbbítva.

A kimenő forgalom maszkolása

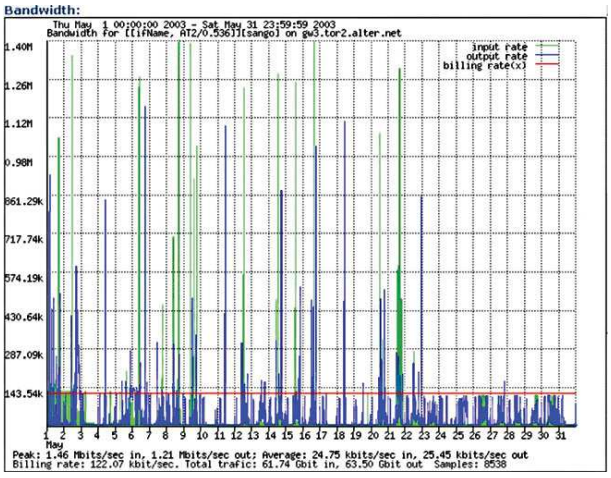
Az ADSL-kapcsolaton keresztül érkező Internet-forgalom olyan kiszolgálókról érkezik, amelyek IP-címe szerepelhet az útválasztásban. Ezeknek a címeknek címfordításon (NAT) kell keresztülesniük, különben a valódi IP címre irányított forgalom a T1-vonalon keresztül visszatér:

```
iptables -t nat -A POSTROUTING -o ppp0 -j
➥MASQUERADE
```

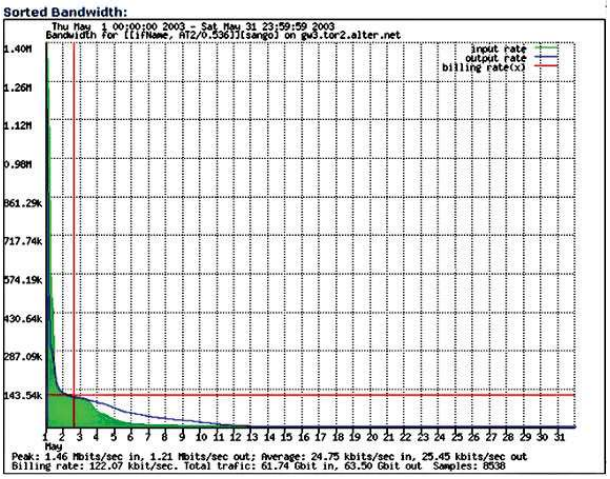
A címkézési és útválasztási eljárásunkat a *4. kép* mutatja.

Az IP-számlálás

Miután a megfelelő forgalmat az ADSL-vonalra irányítottuk át, a T1-re maradó forgalmat úgy kell elosztanunk, hogy a felhasználás határát soha ne lépjük át. A mágikus 95%-os



5. kép A sávszélesség-kihasználtság 2003 májusában 5 perces egységekben mérve



6. kép A sávszélesség-kihasználtság 2003 májusában 5 perces egységekben, sávszélesség szerint rendezve

pont nem haladhatja meg a 128 kbps értéket. Először is az IP-számlálás segítségével mérjük a forgalmat, ami lehetővé teszi számunkra a megadott időintervallumok átlagos átvitelének a számítását.

Az összes T1-vonalon kimenő vagy bejövő csomag áthalad az IP-számlálás szabályrendszerén. Minden ügyfél forgalma külön mérésre kerül az IP-címe és a forgalom iránya alapján. Egy egyénileg fejlesztett démon ellenőrzi az ötperces periódusokban a T1 által használt sávszélességet. Minden olyan esetben, amikor a T1 vonalon átmenő forgalom nagysága meghaladja a 128 kbps értéket az ötperces periódus átlagára számolva, eggyel növeljük az erre fenntartott számláló értékét. A 128 kbps küszöbérték körülbelül 4,5 MB-os átvitelnek felel meg az öt perc alatt.

A számláló 432-es értéke felel meg a havi 36 órának, amely az 5%-os határ elérését jelenti, ekkor lefut a TC (***traffic control*** – forgalomvezérlő) parancsfájl, amely a következő hónap elejéig a 128 kbps érték alá kényszeríti a T1 forgalmát. Az IP-számlálás beállítófájlját a *2. lista* mutatja, amely letölthető a Linux Journal FTP-oldaláról (tp.ssc.com/pub/lj/listings/issue121/7134.tgz).

Forgalomszabályozás a TC-vel

Rendszerint sikerül a hónapot anélkül átvészelnünk, hogy a T1 forgalmát korlátoznunk kellene, de előfordul, hogy felhasználjuk a 36 órás ingyenes keretünket.

Ebben az esetben a forgalomszabályozót (TC) használjuk a sávszélesség korlátozására. A forgalomszabályozásról és a `tc` parancsról szóló leírást a ➲ http://www.lartc.org/manpages címen találunk.

A Qdisc CBQ (***class-based queuing*** – osztályozás-alapú sorbaállítás) szabályait használjuk mind a `wan0` T1 vonal, mind pedig az `eth0` Ethernet szabályozásához. Ezt mindkét kapcsolat mindkét irányú forgalmának vezérlésénél alkalmazzuk:

```
tc qdisc add dev wan0 root handle 10:
➥cbq bandwidth 1500Kbit avpkt 1000
tc qdisc add dev eth0 root handle 20:
➥cbq bandwidth 1500Kbit avpkt 1000
```

Következő lépésként a `Global Class` számára a maximális sávszélességet biztosítjuk a `wan0` és `eth0` számára is, ami mindkét vonal esetén 1500 kbps:

```
tc class add dev wan0 parent 10:0 classid 10:1
➥cbq bandwidth 1500Kbit avpkt 1000 rate 1500Kbit
➥allot 1514 weight 150Kbit prio 8 maxburst 0
tc class add dev wan0 parent 20:0 classid 20:1
➥cbq bandwidth 1500Kbit avpkt 1000 rate 1500Kbit
➥allot 1514 weight 150Kbit prio 8 maxburst 0
```

Létrehozzuk a `User Class` osztályt mindkét vonalon korlátozott sávszélességgel. Az általunk használt sávszélességkorlát 100 kbps és nem 128 kbps, mivel a Linux TC nem teljesen pontos, és próbálgatással azt tapasztaltuk, hogy ha 100 kbps értéknél nagyobbra állítjuk a sávszélesség határát, az átvitel időnként 128 kbps fölé csúszhat:

```
tc class add dev wan0 parent 10:1 classid 10:100
➥cbq bandwidth 1500Kbit avpkt 1000 rate 100Kbit
➥allot 1514 weight 10Kbit prio 8 maxburst 0 bounded
tc class add dev eth0 parent 20:1 classid 20:100
➥cbq bandwidth 1500Kbit avpkt 1000 rate 100Kbit
➥allot 1514 weight 10Kbit prio 8 maxburst 0 bounded
```

Most alkalmazzuk az SFQ sorbaállítási szabályt a `User Class` számára mindkét (`wan0`, `eth0`) vonalon. Ehhez az alapértelmezett ***Stochastic Fairness Queuing*** (SFT) eljárást választottuk ki. Számos más szabály is alkalmazható lenne:

```
tc qdisc add dev wan0 parent 10:100
➥sfq quantum 1514b perturb 15
tc qdisc add dev eth0 parent 20:100
➥sfq quantum 1514b perturb 15
```

Kössük össze a 2-es számú forgalmat a `User Class Queue`-val (felhasználói osztály sor) mind a `wan0` mind az `eth0` esetében. A T1 vonalnak szánt minden forgalom már meg-

kapta a 2-es címkét. A forgalomszabályozás csak a T1 forgalmát korlátozza, az ADSL a teljes fizikai sebességét kihasználva működik:

```
tc filter add dev wan0 parent 10:0 protocol ip
➥prio 25 handle 2 fw flowid 10:100
tc filter add dev eth0 parent 20:0 protocol ip
➥prio 25 handle 2 fw flowid 20:100
```

Az elért eredmények

Az intelligens útválasztás tökéletesen, a programozott tulajdonságoknak megfelelően működik, megfelelően elosztva a forgalmat a T1 és ADSL kapcsolatok között és tartalékvonalat biztosítva egy ADSL-hiba esetére. A T1-en alkalmazott forgalomkezelés kielégítőnek bizonyult, és lehetővé tette, hogy a felhasználóinknak elfogadható szolgáltatást biztosítsunk anélkül, hogy a beavatkozás észlelhető lenne. Természetesen a hónap közben tapasztalt átbocsátóképesség függ attól, hogy az ingyenes sávszélesség milyen gyorsan kerül felhasználásra.

A T1 forgalomkezelésének egy példáját mutatja az *5. kép*, amely a T1 Frame Relay sávszélesség-felhasználásának 2003 májusi értékeit mutatja. A grafikonon látható piros vonal a 128 kbps-os számlázási sávszélesség küszöbértékünket mutatja. Az átvitel korlátozására május 23-tól került sor. Az ügyfeleink egyik kiszolgálóját megfertőzte egy vírus, amely a hónap során igen nagy forgalmat generálva felemésztette az értékes ingyenes sávszélességünk jó részét. Ennek eredményeként ezek az ügyfelek több mint egy hétig 128 kbps sebességkorláttal voltak kénytelenek a T1 vonalon kommunikálni. Az ADSL-forgalmat természetesen mindez nem érintette. A *6. képen* ugyanezeket az adatokat látjuk sávszélesség szerint sorba rakva az ötperces intervallumokat. Ezt összehasonlíthatjuk a cikk elején mutatott ideális sávszélesség-felhasználási grafikonnal. Látható a 122,07 kbps számlázási érték is az ábrán, amely jelzi a forgalomkezelő eljárásunk sikerességét a 128 kbps alatti számlázási értéket illetően.

Összegzés

Bár az intelligens útválasztás, IP-számlálás és forgalomszabályozás egy igen egyszerű megvalósításával ismerkedhettünk meg, képet alkothatunk arról, hogy a Linux fejlett útválasztó eszközei hogyan használhatók fel kifinomult útválasztási stratégiák kialakításához.

Linux Journal 2004. május, 121. szám

David Mandelstam a Sangoma Technologies Corp. elnöke. Az 1984-ben alapított cég WAN-eszközök (hardver és szoftver) fejlesztésére és gyártására szakosodott, ezen belül is kiemelten a PC-ken használható eszközökre. Az általuk kifejlesztett kommunikációs megoldások és útválasztó eszközök minden népszerű WAN-hálózatot, protokollt és PC operációs rendszert támogatnak.

Nenad Corbic a Sangoma technologies Corp. vezető Linux-fejlesztője. (www.sangoma.com)

Biztonságos FTP-szolgáltatás üzemeltetése vsftpd használatával

Ha meg szeretnéd őrizni FTP-helyed biztonságát, akkor csak névtelen hozzáféréseket engedélyezz, és használj minél egyszerűbb FTP-démont.

Gondolná bárki is, hogy az immár közel négy éve futó sorozatban még nem volt szó az FTP-szolgáltatás beállításáról? Most pótolom mulasztásomat, ehhez kedvenc FTP-kiszolgálómat, ***Chris Evans*** kiváló `vsftpd`-jét (Very Secure, nagyon biztonságos FTP démon) hívom segítségül. Mivel korlátozott nagyságú hellyel gazdálkodhatok, és az FTP legjobb használati módja a névtelen FTP, ez alkalommal erre fogunk összpontosítani. A `vsftpd` egyre népszerűbb, hogy újabban a Debian, SuSE, Fedora, Red Hat és egyéb Linux-terjesztésekben is megtalálható. Térnyerésének oka talán az, hogy egyedi módon egyesíti a biztonságot és a kényelmet. Üzembe helyezni mindössze néhány mozdulat, és nem kényszerülünk választani a biztonság és a kényelem között. ***Chris Evans*** fő szempontja a `vsftpd` fejlesztése közben a biztonság megőrzése volt, és eddigi eredményei alapján bizony jó munkát végzett. A `vsftpd` közel négy éve létezik, de komolyabb biztonsági hiányosságot soha nem találtak benne. Mennyire minimalista a `vsftpd`? Teljes forráskódfájának mérete tömörítés nélkül is alig haladja meg az 1 MB-ot. A futtatható `vsftpd` fájl mérete pedig mindössze 80 KB.

A vsftpd letöltése és telepítése

Már említettem, hogy a `vsftpd` sok Linux-terjesztésben alapelemként szerepel. Ha a terjesztésekből telepítjük, a bináris csomagok használatából fakadó szokásos előnyöket élvezhetjük: kényelem, könnyű foltozás, a rendszerprogramok működésének lehető legkisebb mértékű befolyásolása. Debian, SuSE, Fedora és Red Hat alatt logikusan a `vsftpd` nevű csomagot kell telepíteni. Különleges függőségei nincsenek. A legtöbb felhasználó tökéletesen elégedett lesz a saját terjesztésében szereplő `vsftpd` csomaggal.
Ha saját terjesztésünkben a `vsftpd` nem szerepel bináris csomagként, vagy a terjesztésben megtalálhatónál újabb változatot akarunk használni, akkor a ➲http://www.vsftpd.beasts.org címről kell letöltenünk a forrást, majd le kell fordítanunk a programot. A fordítás folyamata határozottan régimódi. Ha még nem tettük volna meg, váltsunk át `root` felhasználóra. Bontsuk ki a ***.tar*** állományt, majd lépjünk át a gyökérkönyvtárába, például így:

```
# tar -xzf vsftpd-1.2.1.tar.gz; cd vsftpd-1.2.1
```

Következő lépésként adjuk ki a `make` parancsot, mindenféle kapcsoló nélkül. Ha futása sikeres, akkor egy futtatható `vsftpd` fájlt kell találnunk az aktuális könyvtárban. Ellenőrizzük, hogy a nobody (senki) felhasználó létezik-e. Ha nem, hozzuk létre, a program ugyanis ennek jogosultságaival fut. Ha még nincs ilyen, hozzuk létre a ***/usr/share/empty*** könyvtárat. Tulajdonosa a `root` legyen, de sem csoport által, sem átfogóan ne legyen írható. Ez lesz a `vsftpd` alapértelmezett `chroot` börtöne (***jail***). Hozzuk létre a névtelen FTP-felhasználó kezdőkönyvtárát. A SuSE hagyományosan a ***/srv/ftp*** könyvtárat használja, más terjesztésekben a ***/var/ftp*** a megszokott, de azt a könyvtárat választjuk, amelyiket csak akarjuk. Ezt a könyvtárat szintén a `root` tulajdonába kell adnunk, senki másnak nem szabad írási jogot kapnia rá. Hozzunk létre egy névtelen FTP-felhasználói fiókot, például ***ftp*** névvel, és ellenőrizzük, hogy kezdőkönyvtára az előző lépésben kiválasztottal egyezik-e. Lehet, hogy a rendszerben már létezik ilyen fiók. A névtelen FTP-felhasználónak ne adjuk írási jogot saját kezdőkönyvtárára, és semelyik fájlnak vagy könyvtárnak nem lehet a tulajdonosa. Most készen állunk arra, hogy a `vsftpd`-t, valamint a ***vsftpd(8)*** és a ***vsftpd.conf(5)*** man oldalakat a megfelelő helyre másoljuk, tehát adjuk ki a ***make install*** parancsot. A minta ***vsftpd.conf*** fájlt kézzel másoljuk a ***/etc*** könyvtárba. Ha a programot önálló démonként szeretnénk futtatni, a ***/etc/init.d*** alatt létre kell hoznunk számára egy indító parancsfájlt. Egyébként az `inetd` vagy az `xinetd` segítségével gondoskodhatunk szükség szerinti indításáról. (Lásd az Önálló démon vagy `inetd`/`xinetd` című részt.)
Ha a `vsftpd`-t önálló démonként futtatjuk, akkor az indító parancsfájlt kétféleképpen tudjuk engedélyezni: RPM alapú Linux-terjesztés alatt a `chkconfig`, Debian GNU/Linux alatt pedig az `update-rc.d` paranccsal. Ha a telepítést RPM vagy deb csomagból végezzük, akkor mindezekre a műveletekre önműködően sor kerül, kivéve talán az utolsót. Mondtam már, hogy bináris csomagokat használni jóval kényelmesebb? Egyes terjesztéseknél az újonnan telepített csomagokat kézzel kell engedélyezni. Például a saját SuSE 9.0 rendszeremen a SuSE vsftpd RPM hiába telepítette

a */etc/init.d/vsftpd* fájlt, engedélyezéséhez ki kellett adnom a `chkconfig --add vsftpd` és a `chkconfig --level 35 vsftpd` parancsot.

A vsftpd leírása

Most nézzük meg, honnan szerezhetünk információt a program működéséről. Először is, a `vsftpd`-hez tartozik egy *EXAMPLE/* könyvtár, amely mintabeállításokat tartalmaz különféle alkalmazási környezetekhez; ide értve az önállóan és `xinetd` alatt történő futtatást, valamint a névtelen és a helyi felhasználók kiszolgálását egyaránt. Ha a `vsftpd`-t forráskódból telepítettük, az *EXAMPLE* könyvtárat a forrást tartalmazó könyvtárban találjuk. Ha a telepítéshez bináris csomagot használtunk, akkor nagy valószínűséggel felkerült egy másolata a gépre, valahova a */user/share/doc* könyvár alá. SuSE rendszereken a */usr/share/doc/packages/vsftpd/EXAMPLE* elérési úton találjuk. Az előző részben már említettem, hogy a `vsftpd`-hez tartoznak man oldalak, a *vsftpd(8)* és a *vsftpd.conf(5)*. Az utolsó megemlítendő forrás, az alapértelmezett (mintaként használható) *vsftpd.conf* maga is rengeteg megjegyzést tartalmaz. Ugyan nem tartalmazza az összes lehetséges beállítást, de a leggyakrabban használtakat igen. Jómagam számos `vstpd`-példányt bírtam már működésre úgy, hogy a minta *vsftpd.conf* fájlban csak elenyésző módosításokat kellett eszközölnöm.

Önálló démon vagy inetd/xinetd

Mielőtt magának a `vsftpd`-nek a beállításait megadnánk, el kell döntenünk, hogy önálló démonként vagy szuperkiszolgálóként, tehát `inetd` vagy `xinetd` segítségével szeretnénk futtatni. A `vsftpd` korábbi változataiban a fejlesztő annak naplózási és hozzáférés-vezérlési szolgáltatásai miatt az `xinetd`-vel történő használatot javasolta. A `vsftpd` 1.2-es és újabb változatai már maguk is képesek biztosítani ezeket a szolgáltatásokat. Éppen ezért *Evans* most már a program önálló démonként való futtatását javasolja. Az `inetd` vagy az `xinetd` használatából természetesen némi teljesítményveszteség is származik. Ez a veszteség semmivel sem ellentételezhető, ha dedikált FTP-kiszolgálót akarunk üzemeltetni, vagy úgy véljük, a rendszer terhelésének jelentős része fog az FTP-szolgáltatás futásából származni. Veszem magamnak a bátorságot, és a továbbiakban feltételezem, hogy önálló démon futtatására rendezkedünk be. A megfelelő leírások részletesen taglalják az `inetd`-vel és `xinetd`-vel való használatot, illetve a `vsftpd` *EXAMPLE* könyvtárában található példabeállítások alapján is sok mindenre rá tudunk jönni. Érdekes módon a SuSE 9 a `vsftpd`-t alapesetben `xinetd`, a Debian 3.0 pedig `inetd` alól futtatja. Utóbbi választás ésszerű, hiszen a Debian 3.0 a `vsftpd` egy régebbi, 1.0.0-s változatát tartalmazza, ám a SuSE 9.0-ban az 1.2-es változat szerepel. A Fedora és a Red Hat terjesztésekhez készült RPM-ek önálló démonként telepítik a `vsftpd`-t. Tulajdonképpen mindegy is, a `vsftpd` `inet`/`xinetd` alóli indításról néhány lépéssel átállítható önálló indulásra. Először is, mint az *A vsftpd letöltése és telepítése* című részben már említettem, ellenőriznünk kell, hogy a */etc/init.d* alatt van-e engedélyezett indító parancsfájl a `vsftpd`-hez. A Fedora Core 1 és a SuSE 9.0 csomagok tartalmaznak és telepítenek is ilyet, SuSE alatt a fájl megvan ugyan, ám az `xinetd` alól történő futtatás miatt alapesetben le van tiltva. Ha a Debian 3.0 `vsftpd` csomagját használjuk, vagy forrásból végezzük a telepítést, akkor magunknak kell elkészítenünk az indító parancsfájlt. Ugyancsak létre kell hoznunk a megfelelő hivatkozásokat azoknak a futási szinteknek a könyvtárában – mint például *rc3.d* és *rc5.d* –, amelyekben futtatni szeretnénk a démont. Végül ki kell adnunk a `chkconfig` vagy az `update-rc.d` parancsot. Az átállás második lépése a `vsftpd` `xinetd` fájljának letiltása a `disable=yes` érték megadásával a */etc/xinetd.d/vsftpd* fájlban, vagy a `vsftpd` sorának megjegyzésbe tételével a */etc/inetd.conf* állományban. Azt is megtehetjük, hogy teljes egészében letiltjuk az `inetd`-t vagy az `xinetd`-t, persze csak akkor, ha a `vsftpd` volt az egyetlen fontos alóla indított dolog. Tudom, felelőtlenség rávenni valakit egy alkalmazás indító parancsfájljának engedélyezésére, míg annak biztonsági szolgáltatásai nincsenek pontosan beállítva. Úgy gondolom azonban, hogy az engedélyezésből önmagában semmi baj nem származhat, ha az illető pontosan követi a javaslataimat, és egyelőre letiltja a szolgáltatást. A harmadik lépés annak ellenőrzése, hogy a */etc/vsftpd.conf* fájlban a `listen` kapcsoló `YES` értéket kapott-e. Ezután továbbléphetünk a tényleges beállításokra.

A vsftpd beállítása névtelen FTP-szolgáltatás biztosítására

Nagy valószínűséggel mást nem is kell tennünk ahhoz, hogy a `vsftpd`-vel biztonságos névtelen FTP-szolgáltatást tudjunk nyújtani. Alapértelmezett beállításai kizárólag ilyen hozzáférést engednek. Sőt mi több, alapesetben semmilyen írási parancs végrehajtását nem engedi, és a `vsftpd` újabb változatai lehetőség szerint `chroot` műveletet hajtanak végre a */usr/share/empty* könyvtárban. Ez az egyik dolog, amiért a `vsftpd`-t szeretem. Igazából az általa nyújtott biztonságot elrontani több munkával jár, mint megőrizni vagy tovább erősíteni. Ha feltételezzük, hogy mindez saját terjesztésünkben sincs másképp, akkor csak annyit kell tennünk, hogy a névtelen FTP-felhasználó kezdőkönyvtárába átmásoljuk a másoknak letöltésre szánt anyagokat. Debian 3.0, SuSE 9.0 és Fedora Core 1 alatt a névtelen FTP-felhasználó alapesetben az ftp fiókot használja, kezdőkönyvtára Debian és SuSE alatt */srv/ftp*, Fedora alatt pedig */var/ftp*. Ha a telepítést forrásból végeztük, a névtelen FTP könyvtára az lesz, amit a névtelen FTP-felhasználó fiókjához kezdőkönyvtárként hozzárendeltünk. Az FTP könyvtárainak fájlokkal feltöltésekor ügyeljünk a tulajdonjogok és az engedélyek pontos beállítására. Előfordulhat, hogy az alapértékek nem megfelelők, ám egy gyors `ls -al` mindent elárul. Ugyan a legtöbb felhasználó számára az alapértelmezett beállítások tökéletesen megfelelnek, tekintsük át a *vsftpd.conf*-ban szereplő, a névtelen FTP-vel leginkább kapcsolatos beállításokat. Alapesetben ez a fájl a */etc* könyvtárban található, bár Red Hat és Fedora rendszereken a */etc/vsftpd/* könyvtárban találjuk. Az *1. kódrészlet* egy minta *vsftpd.conf* fájlt szemléltet. A gyakorlatban a *vsftpd.conf* fájlt senki nem használja az *1. kódrészletben* szereplő formában, ott ugyanis minden beállítás alapértelmezett értékkel szerepel. A kódrészletet inkább hivatkozási alapnak szántam. Lássuk tehát a beállításokat.

- `listen`: Utasítja a `vsftpd`-t, hogy démonként fusson, és ne az `inetd` vagy az `xinetd` által igény szerint, kapcsolatonként indított folyamatként. Alapértéke `NO`.

- `listen_address`: Megadja azt a helyi IP-címet, amelyen a `vsftpd`-nek figyelnie kell a bejövő kapcsolatokat. Alapértéke "" (null), ami az összes helyi IP-címet jelenti. Ha több képzetes FTP-kiszolgálót szeretnénk futtatni, akkor értékét mindegyik képzetes kiszolgáló beállító fájljában meg kell adnunk. (Lásd a Képzetes kiszolgálók című részt.)
- `anonymous_enable`: Alapértéke `YES`, meghatározza, hogy a `vsftpd` fogadja-e a névtelen bejelentkezéseket. Ha értéke `YES` vagy nincs megadva, akkor a `vsftpd` valódi jelszó kérése nélkül fogadja az anonymous (névtelen) és a ftp felhasználók kapcsolatait (a két felhasználó egyenértékű).
- `ftp_username`: A névtelen – vagyis az anonymous és az ftp névvel történő – FTP-bejelentkezésekhez használt felhasználói fiók neve. A fióknak léteznie kell a */etc/passwd* fájlban, és érvényes, nem az adott fiók által tulajdonolt kezdőkönyvtárral kell rendelkeznie. Alapértéke az `ftp`.
- `anon_root`: Az a könyvtár, amelybe a `vsftpd` `chroot` műveletet hajt végre a névtelen bejelentkezéseknél. Az alapérték a névtelen FTP-felhasználói fiók kezdőkönyvtára (lásd az `ftp_username` beállítást), de az `anon_root` segítségével ettől eltérő FTP kezdőkönyvtár is megadható. Bármelyik megoldást is választjuk, a könyvtár tulajdonosa ne legyen a névtelen FTP-felhasználó.
- `write_enable`: Hacsak értéke nem `YES`, egyik felhasználó sem tölthet fel semmilyen fájlt, függetlenül a *vsftpd.conf* egyéb beállításaitól. Alapértéke `NO`.
- `anon_upload_enable`: Ha ennek értéke, valamint a `write_enable` beállítás értéke egyaránt `YES`, akkor a névtelen felhasználók engedélyt kapnak fájlok feltöltésére azokba a könyvtárakba, amelyekre a névtelen felhasználó fiók írási engedéllyel rendelkezik.
- `anon_mkdir_write_enable`: Ha ennek értéke, valamint a `write_enable` beállítás értéke egyaránt `YES`, akkor a névtelen felhasználók engedélyt kapnak könyvtárak létrehozására azokban a könyvtárakban, amelyekre a névtelen felhasználói fiók írási engedéllyel rendelkezik.
- `anon_other_write_enable`: Ha ennek értéke, valamint a `write_enable` beállítás értéke egyaránt `YES`, akkor a névtelen felhasználók engedélyt kapnak könyvtárak átnevezésére és törlésére azokban a könyvtárakban, amelyekre a névtelen felhasználó fiók írási engedéllyel rendelkezik.
- `anon_world_readable_only`: Ha értéke `YES`, akkor a névtelen felhasználók nem tudják letölteni az általános jelleggel nem olvasható fájlokat. Akkor használható jól, ha a névtelen felhasználók engedélyt kapnak fájlok feltöltésére, de nem akarjuk, hogy más névtelen felhasználók ezeket a fájlokat letöltsék.
- `anon_max_rate`: Megadja, hogy a névtelen felhasználók legfeljebb hány bájt/másodperc sávszélességet használhatnak fel. Alapértelmezett értéke 0, ami azt jelenti, hogy semmilyen korlátozás nincs.
- `idle_session_timeout`: A felhasználók legfeljebb ennyi másodpercig adhatnak ki különféle FTP-parancsokat, mielőtt a kapcsolatot a kiszolgáló lezárná. Alapértéke 300, de ha aggódunk a szolgáltatásmegtagadási támadások miatt, csökkentsük.
- `ascii_download_enable`: Ha értéke `YES`, akkor a felhasználók ASCII módú letöltéseket is indíthatnak, nemcsak binárisokat. Alapértéke `NO`, mivel az ASCII mód használatára szinte soha nincs szükség, hatékonysága pedig annyira rossz, hogy kiváló eszköz szolgáltatásmegtagadási támadások indítására.
- `ascii_upload_enable`: Az ASCII módú feltöltés lehetősége bizonyos esetekben jól jöhet, például ha parancsfájlokat akarunk továbbítani. Alapértéke ettől függetlenül `NO`.
- `connect_from_port_20`: Az aktív módú FTP-kapcsolatoknál, ha egy felhasználó letölt valamit, akár csak egy könyvtár fájllistáját, a kiszolgáló új kapcsolatot nyit az ügyfél felé, és ez általában a 20-as számú TCP-kapuról indul ki. Alapesetben azonban a `vsftpd` az ilyen kapcsolatokat magasabb számú kapukról indítja, így nem muszáj `root` ként futnia. Ha az alapértéket meg akarjuk változtatni, mert például a felhasználók ilyen jellegű kapcsolatindításokra fel nem készített proxyk vagy tűzfalak mögül érkeznek, akkor adjunk neki `YES` értéket.
- `port_enable`: Ha `NO` értéket adnunk neki, a `PORT` parancsokat letilthatjuk, amivel gyakorlatilag megtiltjuk az aktív módú FTP-t. Alapértéke `YES`.
- `hide_ids`: Ha `YES` értéket adunk neki, akkor a könyvtártartalmak listázásakor a felhasználók mindenhol `ftp` tulajdonost és `ftp` csoportot látnak. Szerintem nyilvános FTP-kiszolgálókon van értelme egyfajta rejtőzködésre használni, de alapértéke `NO`.
- `log_ftp_protocol`: Ha `YES` értéket adunk neki, engedélyezzük az FTP protokollparancsok részletes naplózását. Ezeket a felhasználói FTP-parancsok indítják ugyan, de azoktól eltérők. Hibakeresési szempontból értéktelen.

1. kódrészlet A vsftpd.conf fájlban szereplő beállítások névtelen FTP üzemeltetéséhez

```
listen=YES
# listen_address=
anonymous_enable=YES
ftp_username=ftp
# anon_root=[$az ftp_felhasználó kezdőkönyvtára]
write_enable=NO
anon_upload_enable=NO
anon_mkdir_write_enable=NO
anon_other_write_enable=NO
anon_world_readable_only=YES
anon_max_rate=0
idle_session_timeout=300
ascii_download_enable=NO
ascii_upload_enable=NO
connect_from_port_20=NO
port_enable=YES
hide_ids=NO
log_ftp_protocol=NO
syslog_enable=NO
max_per_ip=0
# cmds_allowed=
local_root=/usr/share/empty
nopriv_user=nobody
ftpd_banner=(vsFTPd 1.2.0)
```

2. kódrészlet Képzetes FTP-kiszolgáló beállító fájlja (/etc/vsftpd.knusper)

```
listen=YES
listen_on=1.2.3.4
connect_from_port_20=YES
anonymous_enable=YES
anon_root=/srv/ftp/knusper
ftpd_banner=Üdvözöllek a knusper.wiremonkeys.org
➥FTP-helyén. Viselkedj jól!
```

3. kódrészlet Képzetes FTP-kiszolgáló beállító fájlja (/etc/vsftpd.rover)

```
listen=YES
listen_on=1.2.3.5
connect_from_port_20=YES
anonymous_enable=NO
ftpd_banner=Zártkörű FTP a rover.wiremonkeys.org
➥címen. Idegeneket nem fogadunk.
# VIGYÁZAT: Ne használd, amíg nem tudod
➥pontosan, hogy mit is csinálsz!
local_enable=YES
```

- `syslog_enable`: Normál esetben a `vsftpd` a naplóüzeneteket a */var/log/vsftpd.log* fájlba írja. Ha a beállítás értéke YES (alapértéke NO), akkor az üzeneteket a rendszer syslog szolgáltatásának küldi.
- `max_per_ip`: Megadja, hogy egy-egy forrás IP-címről egyszerre legfeljebb hány kapcsolatot lehet indítani. Valamiféle korlátozást bevezetni nem rossz ötlet, bár az alapérték nulla, ami korlátlan kapcsolatszámot jelent. Ha viszont korlátozunk, a NAT/SPAT tűzfalak mögül érkező felhasználókkal kiszúrhatunk, hiszen ők úgy látszanak, mintha egy-egy forrás IP-címről több kapcsolat is eredne.
- `cmds_allowed`: A megengedett FTP-parancsok vesszővel ellátott listája. Alapértéke "" (null), vagyis nincs korlátozás. Csak FTP-protokoll szintű parancsokat lehet megadni, az FTP-ügyfélprogramok által gyakran kezelt parancsokat nem. Ha például az ügyfelek tevékenységét a fájlok listájának lekérdezésére, a munkakönyvtár megváltoztatására és a fájlok letöltésére szeretnénk korlátozni, akkor a következő beállítást kell használnunk: `cmds_allowed=USER,LIST,NLST,CWD,RETR,PORT,QUIT`. A ➲ http://www.nsftools.com/tips/RawFTP.htm oldalon kiváló ismertetőt találunk ezekről a parancsokról.
- `local_root`: Üres, a `root` által tulajdonolt könyvtárat ad meg, a `vsftpd` ide végez `chroot` műveletet, ha éppen a fájlrendszer egyik részéhez sem igényel hozzáférést. Alapértéke */usr/share/empty*.
- `nopriv_user`: Kiemelt jogok nélküli felhasználó, a `vsftpd` ennek jogaival fut, ha lehetséges. Természetesen vannak olyan műveletek, amelyekhez `root` jogokkal kell futnia, ilyen például a 21-es TCP kapuhoz való kötődés. A `vsftpd` a lehető leghamarabb lefokozza önmagát, így próbál hozzájárulni a puffer-túlcsordulásos és az egyéb, `root`-ként eredményesebben kivitelezhető támadások esélyének és sikerességének csökkentéséhez.
- `ftpd_banner`: A csatlakozni próbáló FTP-ügyfeleknek megjelenített üzenet. Az alapértelmezett üzenet bele van drótozva a `vsftpd`-be, az 1.2.0-s változatban ez egyszerűen csak (vsFTPd 1.2.0). Ha saját üzenetet szeretnék használni, akkor a `banner_file` beállításban adjuk meg az azt tartalmazó szöveges fájl nevét.

A ***vsftpd.conf(5)*** man oldal az itt szereplők mellett számos más beállítást is ismertet. Hihető vagy sem, itt csak a felszínt érintettük.

Képzetes kiszolgálók

Egy több IP-címmel is rendelkező gépen több képzetes FTP-kiszolgálót is futtathatunk. Mindössze annyit kell tennünk, hogy több példányban futtatjuk a `vsftpd` démont, mindegyik példányt saját ***vsftpd.conf*** fájllal ellátva, amelyben megadjuk, hogy az adott példánynak melyik IP-címen kell fogadnia a kapcsolatokat, illetve melyik könyvtárat kell névtelen kezdőkönyvtárként használnia. Tegyük fel például, hogy a gépemnek két IP-címe van, az 1.2.3.4 és az 1.2.3.5, ezek rendre a knusper és a rover DNS-névhez tartoznak. Ebben az esetben két `vsftpd` beállító fájlt is használhatok, például */etc/vsftpd.knusper* és */etc/vsftpd.rover* névvel. A *2.* és a *3. kódrészlet* ezeket a fájlokat tartalmazza.
Talán furcsának tűnhet a `local_enable` beállítás használata a *3. kódrészlet*ben. Veszélyes dolog `YES` értéket adni neki, mivel ilyenkor az FTP-bejelentkezéshez használt azonosítók nyílt szövegben kerülnek továbbításra. Elsősorban azért szerepeltetem itt, hogy bemutassam, minden képzetes kiszolgáló saját beállító fájlt használ, és akár merőben eltérő módon is működhet. Megtehetjük például, hogy indítunk egy névtelen felhasználók által is írható képzetes kiszolgálót a nyilvános feltöltések számára, egy másikat pedig egy szigorúan csak olvasható FTP-hely fenntartásához. Ha létrehoztuk a képzetes FTP-kiszolgálók saját beállító fájljait , az indító parancsfájlt is módosítanunk kell. Az én példakiszolgálómon – innen származik a *2.* és a *3. kódrészlet* is – az indító parancsfájlba az alábbi kettőhöz hasonló sorok kerültek:

```
vsftpd /etc/vsftpd.knusper
vsftpd /etc/vsftpd.rover
```

Ha Red Hat vagy Fedora rendszert futtatunk, akkor ezzel a feladattal nem kell foglalkoznunk. Az ezeknek a terjesztéseknek a `vsftpd` RPM-csomagjában szereplő */etc/init.d/vsftpd* parancsfájl önműködően megkeresi a */etc/vsftpd* könyvtárban szereplő beállító fájlokat, feltéve, hogy *.conf* kiterjesztéssel látjuk el őket.

Linux Journal 2004. július, 123. szám

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.

µClinux-ismeretek Linux-programozók számára

Mit szólnál, ha a programjaid memóriakezelés nélküli processzorokon is futnának? Azt, hogy sok munka kellene hozzá? Szó sincs róla.

Az µClinux népszerűsége az utóbbi időben hatalmasat nőtt, és egyre több fogyasztói készülékben láthatjuk viszont. Forgalomirányítókban (*1. kép*), webkamerákban, DVD-lejátszókban való használata önmagában is igazolja sokoldalúságát. Az olcsó, a µClinux futtatására alkalmas 32 bites processzorok terjedése további választási lehetőségeket kínál a µClinux használatán gondolkodó gyártóknak. A µClinux immár a 2.6-os rendszermag része, így borítékolható, hogy ismertsége tovább fog nőni.

1. kép A SnapGear LITE2 VPN forgalomirányítón uClinux fut

Egyre több fejlesztő játszadozik a µClinux használatának gondolatával, ezért egy olyan útmutató, amely összefoglalja a normál Linuxtól való eltéréseit, valamint az alkalmazásánál felmerülő buktatókat és csapdákat, rendkívül nagy értékkel bír. A következőkben áttekintjük, hogy az µClinux használatára magát elszánó fejlesztőnek milyen módosítások elvégzésére kell felkészülnie, valamint maga a környezet hogyan hat a fejlesztési folyamatra.

Memóriakezelés nélkül

Meghatározó és szembeszökő különbség a µClinux és az egyéb Linux-rendszerek között, hogy előbbi nem képes memóriakezelésre. Linux alatt a memóriakezelés virtuális memória (***virtual memory***, VM) használatával folyik, a µClinux viszont olyan rendszerekre készült, amelyek VM-támogatással nem rendelkeznek. Mivel a VM-kezelés általában az úgynevezett memóriakezelő egység (***memory management unit***, MMU) segítségével történik, a µClinux kapcsán sokszor hallani az MMU hiányára utaló NOMMU (nincs MMU) kifejezést.

VM használatakor minden folyamat ugyanarról a – valójában virtuális – címről fut, és a VM alrendszer feladata annak figyelése, hogy ezekhez a virtuális címekhez melyik fizikai memóriaterület tartozik. Lehetséges tehát, hogy az egy folyamat által látott virtuális memóriaterület összefüggő, ám az a fizikai rész, amelyre leképeződik, széttöredezett; sőt, még akár a merevlemezen található csereterületen is lehet. Mivel az önkényesen lefoglalt memória a folyamat címterén belül bárhová leképezhető, már futó folyamathoz minden gond nélkül lehet további memóriát hozzáadni.

VM nélkül minden folyamatot arra a memóriaterületre kell helyezni, ahonnan futni tud. A legegyszerűbb esetben ennek a memóriaterületnek összefüggőnek kell lennie. Bővítésére általában nincs lehetőség, mert előtte és utána más folyamatok lehetnek. Egy µClinux alatt futó folyamat tehát nem tudja a hagyományos Linux alatt futó folyamatokhoz hasonlóan növelni a rendelkezésére álló memória méretét. Ugyan a futtatás érdekében minden programot át kell helyezni, ez a művelet a fejlesztő számára észrevétlen. Szükségessége a VM hiányából fakad, és persze szálka minden µClinux fejlesztő szemében. Közvetlen következmény, hogy semmilyen memóriavédelemmel nem számolhatunk – a rendszermag és a folyamatok a rendszer tetszőleges területét összezavarhatják. Egyes processzortípusok lehetővé teszik ugyan bizonyos be- és kiviteli, utasítás- és memóriaterületek felhasználóktól való védelmét, de garanciát semmire sem kapunk. A rendszer összeomlását okozó hibáknál már csak azok rosszabbak, amelyeket nem veszünk észre, márpedig a folyamatok közötti véletlenszerű memória-felülírások felderítése rendkívül nehéz feladat.

VM hiányában gyakorlatilag csereterületet sem lehet használni. Igaz, a µClinuxot futtató rendszerek esetében ez nem jelent túl nagy gondot, hiszen merevlemezzel vagy a csereterület kihasználásához elegendően nagyméretű memóriával amúgy sem rendelkeznek.

A rendszermag eltérései

A rendszermagot fejlesztők számára a µClinux kevés meglepetést tartogat. Az egyetlen komoly különbség az, hogy az MMU által biztosított lapkezelési lehetőségekről le kell mondani. A gyakorlatban ez csak minimális mértékben érinti a rendszermag működését. A `tmpfs` például nem használható µClinux alatt, mert működése a VM alrendszerre alapul.

Hasonlóképpen le kell mondanunk a szabványos futtatható formátumokról, ezek ugyanis µClinux alatt el nem érhető VM szolgáltatásokat vesznek igénybe. Ehelyett egy új, egyszintű formátumot kell követni. Az egyszintű formátum olyan sűrített futtatható formátum, amely csak futtatható

kódot és adatokat tárol, valamint tartalmazza azokat az áthelyezéseket, amelyek révén a futtatható állomány a memória tetszőleges területére betölthető.
Az illesztőprogramokat sokszor kell módosítani a µClinuxra való áttéréskor, ám nem a rendszermag esetleges eltérése, hanem a támogatandó eszközök jellege miatt. Az SMC hálózati illesztőprogram például támogatja az ISA buszos SMC kártyákat. Ezek általában 16 bites kártyák, és valamilyen 0x3ff fölötti be- és kiviteli címen érhetők el. Az illesztőprogram könnyedén rávehető a lapka nem ISA buszos, beágyazott változatainak támogatására, de 8, 16 vagy 32 bites módban kell futnia, 32 bites be- és kiviteli címet kell használnia, és a megszakítás sorszáma is jóval nagyobb lesz. (Az ISA buszos eszközök megszakítása legfeljebb 16-os lehet.) Tehát az illesztőprogram nagyrészt változatlan marad, a gép jellegzetességei miatt kisebb átültetési munkákat el kell végezni. Elég gyakori, hogy a régebbi illesztőprogramok a be- és kiviteli címeket rövid egészként tárolják, beágyazott µClinux alapú rendszeren viszont, ahol az eszközök leképezett be- és kiviteli címeken érhetők el (memory-mapped I/O addresses), erre nincs lehetőség.
Az `mmap` rendszermagon belüli megvalósítása is merőben eltérő. Ugyan működése a fejlesztő számára nagyjából észrevétlen, mégis ismerni kell a lelkivilágát, különben előfordulhat, hogy a µClinux alapú rendszereken nagyon rossz hatékonyságú megoldásokat fogunk kidolgozni. Hacsak a µClinux `mmap`-ja nem képes közvetlenül a fájlrendszeren belül mutatni a fájlra, biztosítva ezzel annak soros elérését és folytonosságát, memóriát kell foglalnia, és át kell másolnia az adatokat a memóriába. Az `mmap` µClinux alatti hatékony használatához bizony különleges feltételeknek kell teljesülniük. Először is, jelenleg az egyetlen fájlrendszer, amely képes a fájlok összefüggő tárolását biztosítani, az a `romfs`. Mivel a memóriafoglalást el akarjuk kerülni, `romfs`-t kell használnunk. Másodszor, kizárólag a csak olvasható leképezéseket lehet megosztani, vagyis – ugyancsak a memóriafoglalás elkerülése miatt – a leképezések csak olvashatóklehetnek. Éppen ezért µClinux alatt a fejlesztők nem használhatják ki a másolás írás közben (***copy-on-write***) lehetőségeket. A rendszermagnak is figyelembe kell vennie, hogy a fájlrendszernek ROM-ban kell lennie, amely egy csak olvashatónak kinevezett terület lesz a processzor címterében. Erre akkor van lehetőség, ha a fájlrendszer valahol RAM-ban vagy ROM-ban található. Mindkét esetben közvetlen címezhetőséget kell biztosítani a processzor számára. Nulla méretű `mmap` memóriafoglalásra nincs lehetőség, ha a fájlrendszer merevlemezen található, hiába használunk akár `romfs`-t, ugyanis ilyenkor a processzor nem tudja közvetlenül megcímezni a tartalmat.

Memóriafoglalás a rendszermag és az alkalmazások számára

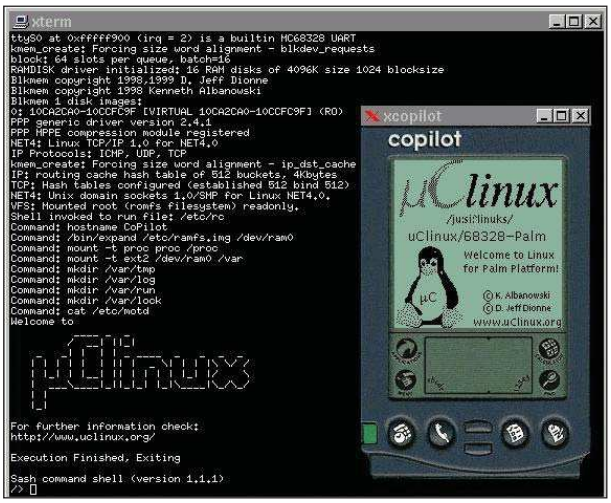
A µClinux kétféle magszintű memóriafoglalót (allocator) kínál. Első látásra talán nem egyértelmű, miért van szükség egy másik memóriafoglalóra is, de a kisméretű µClinux rendszereknél a különbség nyilvánvalóvá válik. A Linux alapértelmezett memóriafoglalója a kettő hatványaira épülő foglalási módszert alkalmaz. Működése ennek köszönhetően gyorsabb, és hamar talál a foglalási kérésnek megfelelő méretű memóriaterületet. Sajnos µClinux alatt az alkalmazásokat arra a memóriaterületre kell betölteni, amit a memóriafoglaló biztosít. A – különösen nagyméretű területek foglalásakor jelentkező – következményeket hamar megértjük, ha veszünk példaként egy 33 KB memóriát igénylő alkalmazást. A memóriafoglalásnál a kettő következő hatványa szerint ez 64 KB memóriát fog kapni. A feleslegesen lefoglalt 31 KB területet nem tudjuk hatékonyan felhasználni. Az ilyen jellegű memóriapazarlás a legtöbb µClinux rendszerben elfogadhatatlan. Az ebből fakadó gondok elkerülésére egy másik memóriafoglalót is készítettek a µClinux rendszermagokhoz. Általában `page_alloc2` vagy `kmalloc2` névvel illetik, a tényleges rendszermagváltozattól függően. A `page_alloc2` a kettő hatványai szerint történő memóriafoglalásból fakadó pazarlást szünteti meg. Az egy lapnyinál (egy lap mérete 4096 bájt, vagyis 4 KB) kevesebb memóriát igénylő alkalmazások számára ez is a kettő hatványai szerint foglalja a memóriát, e felett azonban a lefoglalt terület méretét a következő egész lapméretre kerekíti. Visszatérve az előző példához, egy 33 KB-ot igénylő alkalmazás 36 KB memóriát fog kapni, vagyis egy 33 KB-os alkalmazásnál azonnal megtakarítottunk 28 KB-ot.
A `page_alloc2` a memória töredezése ellen is megteszi a szükséges lépéseket. A két lapnyi (8 KB) vagy kisebb igényeket a memória elejéről, az ennél nagyobbakat a végéről elégíti ki. Ezzel elkerülhető, hogy az átmeneti, például hálózati pufferek számára végzett foglalások miatt a memória feltöredezzen, és később a nagyobb alkalmazások futtatása meghiúsuljon. Részletesebb memóriatöredezési példát lejjebb, az Alkalmazások és folyamatok című részben mutatok. A `page_alloc2` persze nem tökéletes, de a gyakorlatban remekül működik, ugyanis a µClinuxot használó beágyazott készülékeken alkalmazások egy viszonylag állandó csoportja és általában hosszú ideig fut. Miután a kedves fejlesztő túltette magát a rendszermag memóriafoglalási nyűgein, a valódi érdekességeket az alkalmazások terén tapasztalhatja. Itt mutatkozik meg igazán a µClinux VM-kezelésének hiánya. A legnagyobb különbség, ami az alkalmazások µClinux alatti működését gátolja, az a dinamikus verem hiánya. A VM-kezeléssel rendelkező Linuxok alatt, ha egy alkalmazás megpróbál a verem tetején túlra írni, kivételjelzést kapunk, és a vermet újabb memóriafoglalással bővíti a rendszer. µClinux alatt ilyesfajta kényelemre ne számítsunk, a vermet fordítási időben kell lefoglalni. Az a fejlesztő, aki alkalmazásának kapcsán eddig talán nem is törődött a verem használatának témakörével, most kénytelen lesz tisztázni az ilyen vonatkozású kérdéseket. Az első dolog, amivel a fejlesztőknek foglalkozniuk kell, amikor újonnan átültetett alkalmazásuk rejtélyesen összeomlik vagy rendellenesen kezd működni, az a lefoglalt verem mérete. Alapesetben a µClinux 4 KB-ot foglal a verem számára, ami egy korszerű alkalmazás esetében nagyjából a semmivel egyenlő. A fejlesztőnek az alábbi módszerek valamelyikével meg kell próbálnia növelni a verem méretét:

1. Az `FLTFLAGS=-s <veremméret>` hozzáadása és az `FLTFLAGS` exportálása az alkalmazáshoz tartozó `Makefile` állományba fordítás előtt.
2. Az `flthdr -s <veremméret>` parancs futtatása az alkalmazás lefordítása után.

A második jelentős eltérés, ami a µClinux alatti fejlesztést megnehezíti, az a dinamikus kupac (***heap***) hiánya – ez az

a terület, ahonnan a `malloc` és a hasonló C-függvények használatával bejelentett igények kielégítése történik.
A VM-kezeléssel rendelkező Linux-rendszerek alatt az alkalmazások növelhetik folyamatméretüket, vagyis dinamikus kupaccal rendelkezhetnek. Mindennek megvalósítása hagyományosan alacsony szintű `sbrk/brk` rendszerhívásokkal történik, melyekkel növelhető és megváltoztatható a folyamatok címterének mérete. A könyvtári függvényekkel – mint a `malloc` – történő kupackezelés annak a memóriaterületnek a felhasználásával történik, amelynek lefoglalására az alkalmazás nevében meghívott `sbrk()` függvény segítségével került sor. Ha egy alkalmazásnak bármikor több memóriára van szüksége, akkor csak újra meg kell hívnia az `sbrk()` függvényt. A csökkentésre a `brk()` segítségével nyílik mód. Az `sbrk()` a folyamat végén növeli meg a memóriaterületet. A `brk()` önkényesen dönt, hogy közelíti az elejéhez a folyamat végét, vagyis csökkenti annak méretét, vagy kijjebb tolja azt, vagyis növeli a folyamatot.
Mivel μClinux alatt a `brk` és az `sbrk` szolgáltatásai nem valósíthatók meg, egy átfogó memóriakészletből kell gazdálkodni, amely alapjában véve a rendszermag szabad memóriakészlete. Természetesen ennek a megoldásnak is vannak hátulütői. Például egy megszaladó folyamat akár a teljes rendszermemóriát felemésztheti. A rendszerkészletből való foglalás nem egyenértékű az `sbrk` és a `brk` használatával, ezek révén ugyanis a folyamat címterének végét toljuk ki. Egy normál `malloc` megvalósítás tehát nem felel meg, új megvalósításra van szükség.
Az átfogó készletes szemléletnek nyilván előnyei is vannak. Az első, hogy csak a ténylegesen szükséges memóriaterületre tesszük rá a kezünket, nem úgy, mint az előre foglalt kupacot használó beágyazott rendszereknél. Ez a μClinux rendszerek esetében rendkívül fontos tényező, hiszen memória tekintetében általában szűkösen állnak. A második előny, hogy amikor befejezzük egy memóriaterület használatát, azonnal visszatehetjük a készletbe. A megvalósítás során támaszkodhatunk a rendszermag beépített memóriakezelő szolgáltatásaira is, így csökkenteni tudjuk alkalmazásunk kódjának méretét. Az új felhasználók legtöbbször a memóriahiány gondjába futnak bele. A rendszer ugyan nagy mennyiségű szabad memóriával rendelkezik, az alkalmazás mégsem képes adott méretű puffer számára memóriát foglalni. A hiba oka ez esetben a memória töredezése, és jelenleg sajnos minden μClinux alapú megoldás szenved tőle. A VM-kezelés hiánya miatt μClinux alapú környezetben a memória teljes kihasználása a töredezés miatt gyakorlatilag lehetetlen. Legjobb, ha nézünk erre egy példát. Tegyük fel, hogy rendszerünk 500 KB szabad memóriával rendelkezik, és egy alkalmazás betöltéséhez 100 KB-ra van szükségünk. Ez egy egészen hétköznapi eseménynek mondható. Ne feledjük azonban, hogy az igény kielégítéséhez 100 KB-nyi összefüggő memóriaterülettel kell rendelkeznünk. Tegyük fel, hogy a memóriatérkép a következőképpen alakul. Minden karakter körülbelül 20 KB-nyi területet szimbolizál, az X-ek a rendszermag vagy más programok által lefoglalt vagy használt részeket jelzik:

```
0     100   200   300  400   500   600   700  800   900   1000
 -+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+--
  |xxxxx|xxxxx|---xx|--x--|-x---|xx---|-x---|-xx--|-x---|xxxxx|
```



2. ábra uClinux futtatása Xcopilot alatt (Palm emulátor)

Mint látjuk, van ugyan 500 KB-nyi szabad területünk, ám a legnagyobb összefüggő szakasz is mindössze 80 KB-os. Ilyen helyzet többféle módon is kialakulhat. A leggyakoribb ok, hogy egy program lefoglal valamennyi memóriát, majd nagy részét felszabadítja, és ekkor egy kisebb lefoglalt rész marad egy nagyobb szabad blokk közepén. A rövid ideig futó programok ugyancsak befolyásolhatják a memóriafoglalások helyét és módját. A μClinux `page_alloc2` memóriafoglalója rendelkezik egy kapcsolóval, amely pontosan az ilyen gondok elkerülését segíti. Ez egy új `/proc` bejegyzést hoz létre, a `/proc/mem_map`-et, amely a lapokról és foglalási csoportosításukról tájékoztat. Részletes ismertetésétől helyhiány miatt eltekintek, de a `page_alloc2.c`-hez a rendszermag forrásában kielégítő leírást lehet találni.
Gyakori kérdés, hogy miért nem töredezettség-mentesítjük a memóriát, hogy ily módon be tudjuk tölteni a 100 KB-os alkalmazást. A baj az, hogy nincs VM-kezelés, vagyis nem tudjuk áthelyezni a programok által használt memóriaterületeket. A programok általában különféle hivatkozásokat tesznek a lefoglalt területeken belüli címekre, és VM-kezelés nélkül a memóriának mindig a megadott helyen kell elérhetőnek lennie. Ha átmozgatjuk a programokat, egyszerűen összeomlanak. A μClinux ezt a gondot nem tudja megoldani. A fejlesztőknek tudniuk kell róla, és lehetőség szerint kisebb foglalási blokkokat kell használniuk.

Alkalmazások és folyamatok

A VM-kezeléssel rendelkező Linuxok és a μClinux között további fontos különbség a `fork()` rendszerhívás hiánya. Ha egy fejlesztő `fork()` hívást alkalmazó programot szeretne átültetni, bizony sokat kell dolgoznia vele. μClinux alatt az egyetlen lehetőség a `vfork()` használata. Ugyan a `vfork()` sok tekintetben megegyezik a `fork()` függvénnyel, éppen a különbségek számítanak a legtöbbet. Ha valaki nem ismerné a `fork()` és a `vfork()` rendszerhívást: segítségükkel egy folyamat két folyamatra, egy gyermekre és egy szülőre osztódhat. Egy-egy folyamat tetszőleges számú alkalommal osztódhat, ha több gyermeket is létre kell hoznia. Amikor egy folyamat elvégzi a `fork()` hívását, a gyermek minden tekintetben a szülő másolata lesz,

ám semmin nem osztozik vele, és mind ő, mind szülője függetlenül futnak tovább. A `vfork()` esetében más a helyzet. Először is, a szülő felfüggesztésre kerül, futása leáll, amíg a gyermek ki nem lép vagy meg nem hívja az `exec()` függvényt, az új alkalmazás indítására szolgáló rendszerfüggvényt. A gyermek a `vfork()` visszatérése után a szülő vermén fut, illetve a szülő memóriáját és adatait használja. A gyermek tehát akár meg is rongálhatja szülőjének vermét vagy adatszerkezeteit, ami nyilván hibát okoz. A bajt úgy kerülhetjük el, hogy biztosítjuk, a `vfork()` meghívása után a gyermek soha ne térjen vissza a jelenlegi veremkeretből, és munkájának végén az `_exit` függvényt hívja meg. Az `exit` azért nem használható, mert módosítja a szülő adatszerkezeteit. A gyermeknek tartózkodnia kell az átfogó adatszerkezetekben vagy változókban tárolt adatok módosításától, ezek a változtatások ugyanis ellehetetleníthetik a szülő működését. Egy alkalmazás átírása a `vfork` használatára a `fork` helyett a legtöbb esetben vagy gyermekien egyszerű, vagy borzalmasan nehéz. Általánosan elmondható, hogy ha az alkalmazás a `fork()` hívása után nem hívja meg gyakorlatilag azonnal az `exec()` függvényt, akkor a `fork()` – `vfork()` csere elvégzése előtt gondosan át kell vizsgálni. A µClinux egyszintű futtatható formátuma, bár közvetlenül nem érinti az alkalmazásokat és működésüket, enged néhány olyan műveletet, amelyet a normál linuxos ELF futtatható állományok nem. Az egyszintű formátum kétféle változatban létezik, teljesen áthelyezett és helyfüggetlen kód (***position-independent code***, PIC) változatban. A teljesen áthelyezett változat a kódjához és az adatokhoz egyaránt rendelkezik áthelyezésekkel, míg a PIC változat csak az adatokhoz használ néhányat. A beágyazott rendszerek fejlesztői számára az egyik legelőnyösebb szolgáltatás a helyben futtatás lehetősége (***execute-in-place***, XIP). Használatakor az alkalmazás közvetlenül Flash memóriáról vagy ROM-ból fut, valóban csak minimális, adatainak elhelyezésére elegendő memóriát igényelve. Ezzel a megoldással az alkalmazás kódja vagy szöveges részei több példány között is megoszthatók. Az XIP támogatása nem minden µClinuxos gépen megoldott, ugyanis a fordító részéről is megfelelő támogatást igényel, valamint PIC formátumú futtatható fájlok használatát teszi szükségessé. Amíg tehát adott géptípus eszközkészlete képtelen a PIC kezelésére, addig az XIP támogatásáról sem lehet szó. Jelenleg csak m68k és ARM rendszereken létezik az XIP használatához szükséges fejlettségű támogatás az egyszintű formátumhoz. A `romfs` az egyetlen fájlrendszer, amely képes az XIP µClinux alatti támogatására, ugyanis az XIP használatához az alkalmazásokat összefüggően kell tárolni a fájlrendszerben. Az egyszintű formátum az alkalmazások vermének méretét is megadja a fejállományban (***flat header***) mező formájában. Ha növelni kell egy alkalmazás vermének méretét, akkor csupán ezt a mezőt kell átírni. Erre az `flthdr` parancs használható, a következő módon:

```
flthdr -s egyszintű_futtatható_állomány
```

Az egyszintű formátum kétféle tömörítés használatát is lehetővé teszi. A teljes futtatható állományt tömörítve tudunk a legjobban takarékoskodni a ROM-mal. Azzal a kellemes mellékhatással is jár, hogy az alkalmazás teljes egészében egyetlen összefüggő RAM-területre töltődik be. Dönthetünk a csak az adatszegmensre kiterjedő tömörítés használata mellett is. Erre akkor lehet szükség, ha takarékoskodni szeretnénk a ROM-területtel, de XIP-et is használni szeretnénk. Az

```
flthdr -z egyszintű_futtatható_állomány
```

paranccsal teljesen tömörített futtatható állományt készíthetünk, míg a

```
flthdr -d egyszintű_futtatható_állomány
```

paranccsal az adatszegmensre korlátozhatjuk a tömörítést.

Megosztott könyvtárak

A megosztott könyvtárakról csak érintőlegesen szólhatok, de annyit mindenképpen érdemes tudni róluk, hogy µClinux alatt teljesen más a használatuk. A jelenleg elérhető megoldások a fordítóprogram részéről módosításokat, a fejlesztő részéről pedig kiemelt figyelmet igényelnek. Ha megosztott könyvtárat akarunk létrehozni, legjobb, ha egy példával kezdünk. A jelenlegi µClinux terjesztések az ***uC-libc*** és az ***uClibc*** könyvtárakhoz egyaránt kínálnak megosztott könyvtárakat. Megosztott könyvtárat létrehozni nem nehéz, és mindkét könyvtár jó és könnyen érhető példát mutat rá. Mielőtt bárkinek is túl nagy várakozásai lennének, megemlíteném, hogy a GCC `-shared` kapcsolójának használata nem része a megosztott könyvtárak létrehozásának, tehát senki ne gondolja, hogy tapasztalatból meg tudja oldani a feladatot. µClinux alatt a megosztott könyvtárak ugyanolyan egyszintű futtatható állományok, mint az alkalmazások, és tényleges megosztásukhoz helyben futtathatóra kell fordítani őket. A helyben futtatás lehetősége nélkül a megosztott könyvtárak minden őket használó alkalmazáshoz külön példányban indulnak el, ami rosszabb, mint ha befordítanánk őket alkalmazásainkba.

Összegzés

µClinuxra váltani Linuxról sokszor nem csupán a kétféle rendszer közötti különbségek kezeléséről szól. A µClinux rendszerek egyre mélyebben beágyazott rendszerek, sokszor meglehetősen kevés memóriával, kisebb ROM-mal és szokatlan eszközökkel kiegészítve. A merevlemez elmaradása és a szigorú erőforráskorlátok, párosulva a memóriavédelem hiányával és az eltérések szövevényes rendszerével sokszor a vártnál nehezebbé teszik a µClinux világában tett első kirándulást. Kezdetben a legjobb az, ha megismerkedünk a µClinux emulátoraival (*2. ábra*) vagy olcsó eszközöket szerzünk be. Remélem, hogy a fontosabb kérdésekre rámutatva sikerül elérnem, hogy az óvatosabb fejlesztők felkészültebben kezdjenek bele a µClinux megismerésébe, és el tudják kerülni a leggyakoribb csapdákat és félreértéseket.

Linux Journal 2004. július, 123. szám

David McCullough vezető programmérnök és veterán beágyazott-program fejlesztő. A SnapGear és a Lineo előtt a Stallion Technologies alkalmazásában állt, ahol programozási és mérnöki vezető volt, és SCO és BSD UNIX alapú termékek fejlesztésében vett részt.

A Perl és az adatbázisok (1. rész)

Az adatbázisok életútja a szöveges állományoktól az SQL-ig a Perl távcsövén keresztül nézve.

Életünket azóta hálózzák be az adatok kezelésével kapcsolatos teendők, amióta írni tudunk. Az általános iskolában bevett szokássá vált barátság-füzeteink után általában valamilyen katalógus következett, amitől már csak egy lépés volt az üzleti kapcsolatainkat és napi teendőinket tároló határidőnapló. Ezeknek az adatoknak a naprakészen tartása, rendszerezése és nem utolsó sorban a jogosulatlan szemektől való távol tartása sok-sok ismétlődő részfeladat révén okozott unalmas perceket mindannyiunk számára. A számítógép pontosan azokra a feladatokra alkalmazható kiválóan, amik – noha feltétlen pontosságot és mérnöki precizitást igényelnek – a munkavégzés során kevés új ötletet vagy izgalmas megoldást várnak el tőlünk. Röviden szólva a számítógép a mai kor tökéletes rabszolgája. Emiatt jogos igény kényes adataink szervezésével és gondozásával kapcsolatos tennivalóinkat hű szolgánkra bízni. Ahhoz azonban, hogy valóban értékes perceket őrizzünk meg magunknak a szabadidőnkből, gondos tervezés kell. A tervezés első lépése, hogy kiválasztjuk azokat az eszközöket, amelyeket fel szeretnénk használni a munkánk során. Az adatfüggetlenítés elvéből nyilvánvalóan következik, hogy két, egymástól jól elkülönített munkához kell segítőt találnunk. Az egyik kizárólag az adatokra összpontosít, azok tárolásával, felügyeletével és szolgáltatásával törődik. A másik közvetít a felhasználó és az első segítő között, azaz tolmácsolja a felhasználó kéréseit és a megadott módon kivonatolja a temérdek adatból azt, amire kíváncsiak vagyunk. Ebben a cikksorozatban a Perl lesz a közvetítőnk. A továbbiakban feltételezem, hogy nemcsak hallottál erről a hihetetlenül rugalmas parancsnyelvről, de már meg is tetted az első lépéseidet ezen a téren, így nem fogom ecsetelni a nyelv alapjait. A Perl azért jó választás erre a feladatra, mert a legegyszerűbbtől a legösszetettebb adatbáziskezelési feladatokig egyszerű és gyors programozási eszközöket bocsát a rendelkezésünkre. Emiatt nem takarja el a nyelvi megvalósítás módszere az adott megoldás elvi lényegét. Az első segítő személyében pedig cikkről cikkre új adatbáziskezelőt fogunk megismerni. A körutazás során az egyszerűtől az egyre összetettebb rendszerek felé haladunk majd, ami azonban nem jelenti azt, hogy ne lehetne használni akár már a legelső részben bemutatásra kerülő módszereket. Az adatok kezelésénél nagyon könnyű abba a hibába esni, amikor valaki ágyúval lő egy verébre. Mindig fel kell mérni az előre látható igényeket, és azokhoz választani adatbáziskezelő rendszert. A mai adatbázisok legnagyobb hányadának alapjait a táblák képezik. Egy tábla sorok előre nem meghatározott számú, rendezetlen gyűjteménye. Minden sor, más néven rekord, adott számú mezőből épül fel, és minden mezőnek határozott típusa van. Természetesen minden sor ugyanannyi, és ugyanolyan típusú mezőt tartalmaz. A mezők típusának szigorúsága a használt adatbáziskezelő rendszertől függ. Egyes rendszerek különbséget tesznek az egész, és lebegőpontos számok között is, mások csak a szöveget különböztetik meg a számtól, megint mások szövegként kezelnek mindent. Egy adatbáziskezelőnek négy alapvető műveletet kell ismernie. A lekérdezés valamilyen feltételnek eleget tevő rekordok összegyűjtése, illetve ezek mezőinek kiválasztása. A tárolás az adatbázisba történő új rekord felvételét jelenti. A módosítás során egy meglévő rekord egy, vagy több mezője frissül. A törlés pedig egy meglévő rekord eltávolítása az adatbázisból. Az adatbáziskezelő kiválasztása során fontos szempont, hogy az említett műveletek átlagosan milyen gyakorisággal kerülnek végrehajtásra.
Legelső adatbázisunk egy sima, szöveges állomány lesz. Adatbáziskezelőről itt nem beszélhetünk, hiszen még az adatbázis belső szerkezetének épségben tartásáról is magának a programozónak kell gondoskodnia. Egy állomány egy táblát tárol, melyben a rekordokat a sortörés, a rekordok mezőit pedig egy előre meghatározott kitüntetett karakter, többnyire a kettőspont, vagy vessző választja el egymástól. Ennek a módszernek pont az egyszerűségében rejlik a szépsége. Néhány száz sor esetén, ahol a leggyakoribb művelet az új rekord felvétele és a lekérdezés, kevés munkával nagy teljesítmény érhető el. Létjogosultságát jelzi az a tény is, hogy számos komoly rendszer sarokköveit a mai napig ilyen adatbázisok alkotják. Elég csak a UNIX */etc/passwd* állományára gondolni, mely tökéletes iskolapéldája az említett módszernek. A példák során egy olyan adatbázissal fogunk dolgozni, mellyel elsősorban a női szívek meghódítását tehetjük zökkenőmentessé. Az adatbázisban női ismerőseink nevét és telefonszámát fogjuk tárolni, továbbá egy olyan apróságot, amivel levehetjük a leányzót a lábáról. Először is tehát hozz létre egy minta adatbázist, ahol kettőspont választja el a mezőket egymástól. Ehhez tetszőleges szövegszerkesztőt használhatsz.

```
Kata:123-4567:kóla
Évi:765-4321:romantikus vígjáték
Mari:135-2467:állatkert
Mónika:753-6421:virágcsokor
```

Most első lépésként írjunk egy Perl programot az adatok lekérdezéséhez. Az alábbi szkript kiírja az adott állományból az összes olyan lány adatait, akinek a neve illeszkedik a szintén paraméterként átadott mintára.

```
#!/usr/bin/perl -w

use strict;

die "Használat: " . $0 . " <adat fájl> <~lány
➥neve>\n" unless 2 == @ARGV;

my $talalat = 0;
open LANYOK, $ARGV[0] or die "Nem tudom megnyitni: "
➥. $ARGV[0] . "\n";

while ( <LANYOK> ) {
    chop;
    my ( $nev, $telszam, $szereti ) = split
    ➥( /:/, $_ );
    if ( $nev =~ /$ARGV[1]/ ) {
       print "Találat: " . $nev . " a(z) " . $. .
       ➥". sorban.\n\n";
       print " Adatlap: " . $nev . "\n";
       print "===========" . "=" x length ( $nev )
       ➥. "\n";
       print "  Telefonszám                  : " .
       ➥$telszam . "\n";
       print "  Ezzel veheted le a lábáról : " .
       ➥$szereti . "\n\n";
       $talalat++;
    }
}

close LANYOK;
print "Összesen " . $talalat . " találat.\n";
```

Az, hogy a parancsértelmezőt `-w` kapcsolóval hívjuk meg és a `strict` modult használjuk egy lépés a kultúrált, átlátható, továbbá lehetőség szerint hibamentes program írása felé. A `-w` fontos figyelmeztető üzenetekkel lát el, ha kifogásolható nyelvi szerkezettel éltünk, ezért mindenképpen javasolt a használata. A `strict` modul pedig szigorú ellenőrzési módot ír elő az értelmezőnek. A második sorban a programnak átadott paraméterek számát ellenőrizzük. Ha egy tömböt skalárként értelmezünk, akkor a tömb elemeinek a számát kapjuk. Így a megadott, összefűzött karakterlánc jelenik meg a képernyőn, és a hibára jellemző visszatérési értékkel kilép a program, ha az átadott paraméterek száma nem egyenlő pontosan 2-vel. A `$talalat` nevű változóban fogjuk tárolni a találatok számát. Ez csupán ahhoz szükséges, hogy a keresés végeztével megjelenítsünk egy összegző sort. A következő sorban megnyitjuk az első paraméterben szereplő állományt olvasásra, így a továbbiakban `LANYOK` fájlleíróval hivatkozhatunk rá. Amennyiben a megnyitás nem volt sikeres, hibaüzenettel kilépünk.

A főciklusban soronként dolgozzuk fel a bemeneti állományt. Minden körben a `$_` alapértelmezett változó fogja tartalmazni a beolvasott sort. Minden sorról levágjuk a sorvége jelet, majd beszédes változónevekbe kigyűjtjük a rekord mezőit. Ha a név mezőre illeszkedik a paraméterként megadott minta, akkor kinyomtatjuk a képernyőre az aktuális rekord adatlapját, továbbá megnöveljük a `$talalat` értékét. A találatot bemutató jelentést a lehető legízlésesebben jelenítjük meg. Először a `$.` különleges változót felhasználva kiíratjuk, hogy a találat melyik sorban történt. Ezután egy sor kihagyással következik az adatlap. Ennek fejlécét hosszának megfelelően aláhúzzuk, majd kinyomtatjuk az egyes mezők tartalmát. A program végén lezárjuk az adatbázist, utolsó lépésként pedig kiíratjuk a képernyőre a találatok számát. A megvalósítás hibája, hogy nem szerepelhet az elválasztó karakter egyik mezőben sem. Elképzelhető, hogy repülő ékezeteket szeretnél használni az adatbázisban, ekkor azonban a rövid ö betűvel gondjaid támadhatnak. Erre a problémára bevett megoldás a rögzített hosszúságú mezők használata. Ez esetben elválasztó karakterre nincs is szükség, így átvágtad a gordiuszi csomót. A második programban megvalósítjuk a tárolás műveletet. Programozási oldalról ez a lehető legkönnyebb, és az erőforrásokat is ebben az esetben vesszük a legkevésbé igénybe. Önmagában a tárolás művelethez nincs szükség az állomány beolvasására. A program egyszerűen hozzáírásra nyitja meg az adatbázist, és egy új sorral gazdagítja a meglévő állományt.

```
#!/usr/bin/perl -w

use strict;

die "Használat: " . $0 .
    " <adat fájl> <lány neve> <telefonszám>
    ➥<ezt szereti>\n"
    unless 4 == @ARGV;

open LANYOK, ">>" . $ARGV[0] or die "Nem tudom
➥megnyitni: " . $ARGV[0] . "\n";

print LANYOK join ( ":", $ARGV[1], $ARGV[2],
➥$ARGV[3] ) . "\n";

close LANYOK;
print "Az új lány (" . $ARGV[1] . ") felvéve.\n";
```

Egy kifinomultabb alkalmazásban viszont lehetséges, hogy nem megengedhető két azonos kulcs létezése. Ennek az ellenőrzéséhez előbb az első példában bemutatott módszerrel végig kell futni az állományon, és mikor meggyőződtünk arról, hogy nincs azonos kulcs, utána hozzáfűzni az új rekordot. A rekordok módosítása sima szöveges állományban tárolt adatbázis esetén már nehézkes. A fő gond az, hogy miután beolvastuk az adatokat és bemutattuk a megfelelő bűvészmutatványokat a kiszemelt rekordokon, a módosított eredményhalmazt vissza kell írni eredeti helyére. Mindezt természetesen úgy kell megtennünk, hogy nem veszítünk közben adatot. Kétféle megközelítésben foghatunk neki a probléma megoldásának. Az első, hogy beolvassuk a teljes állományt a memóriába, frissítjük a szükséges rekordokat és

visszaírjuk az eredményt. Ennek azonban egy nagyobb adatbázis esetén feltétele a hatalmas memória és egyenes következménye a költséges műveletek okozta teljesítményvisszaesés. Járhatóbb út, ha rekordról rekordra írunk egy átmeneti állományba, végül ezt kicseréljük az eredetivel.

```
#!/usr/bin/perl -w

use strict;

die "Használat: " . $0 .
    " <adat fájl> <lány neve> <uj telszám>\n"
    ➥unless 3 == @ARGV;

open REGILANYOK, $ARGV[0] or die "Nem tudom
➥megnyitni: " . $ARGV[0] . "\n";
open UJLANYOK, ">tmp." . $$ or die "Nem tudok új
➥fájlt létrehozni.\n";

while ( <REGILANYOK> ) {
    my ( $nev, $telszam, $szereti ) = split
    ➥( /:/, $_ );
    if ( $nev eq $ARGV[1] ) {
       $telszam = $ARGV[2];
       print "Módosítás történt a(z) " . $. . ".
       ➥sorban.\n";
    }
    print UJLANYOK join ( ":", $nev, $telszam,
    ➥$szereti );
}

close UJLANYOK;
close REGILANYOK;
unlink $ARGV[0] or die "Nem tudom törölni: " .
➥$ARGV[0] . "\n";
rename "tmp." . $$, $ARGV[0] or die "Nem tudok
➥átnevezni.\n";
print "Kész.\n";
```

Tehát két állományt nyitunk meg a főciklus előtt. Az egyik, melyet a `REGILANYOK` állományleíróval érünk el később, olvasásra nyitjuk meg. Ez az eredeti adatbázisunk, melyet a program végén felváltunk az újjal. A másik, az `UJLANYOK` azonosítójú, egy átmeneti állomány, mely az adott könyvtárban ***tmp.$$*** névvel jön létre, ahol `$$` a programunk folyamat-azonosítója. Ebbe írjuk ki egyenként a rekordokat a `REGILANYOK`-ból, a szükséges módosításokat elvégezve.

A főciklusban sorról sorra dolgozzuk fel az eredeti adatbázist. Minden sort felbontunk és ellenőrizzük, hogy szükséges-e a módosítás. Ha igen, felülírjuk az adott mezőt és erről azonnal tájékoztatjuk a felhasználót is. Végül egyesítjük a szétbontott sort és kiírjuk az átmeneti adatbázisba. Javítható az algoritmus hatékonysága, ha még a felbontás előtt végzünk egy előzetes mintaillesztést a kulcsra. A ciklus végén mindkét állományleírót bezárjuk. Ezután eltávolítjuk az eredeti adatbázist, és átnevezzük az átmenetit. Végül jelezzük a felhasználónak, hogy a folyamat elkészült. Ez a megoldás meglehetősen veszélyes, hiszen ha a törlés sikeres volt, de az átnevezés nem, akkor csak kézi beavatkozással lehet megmenteni az adatbázist.

Már csak a törlés művelet maradt hátra. Ennél egy az egyben ugyanazt az algoritmust követjük, mint az előző esetben, leszámítva azt, hogy a feldolgozás során nem módosítunk a rekordokon, hanem csak a szükségeseket írjuk ki.

```
#!/usr/bin/perl -w

use strict;

die "Használat: " . $0 . " <adat fájl> <lány
➥neve>\n" unless 2 == @ARGV;

open REGILANYOK, $ARGV[0] or die "Nem tudom
➥megnyitni: " . $ARGV[0] . "\n";
open UJLANYOK, ">tmp." . $$ or die "Nem tudok új
➥fájlt létrehozni.\n";

while ( <REGILANYOK> ) {
    if ( /^$ARGV[1]:/ ) {
       print "Törlés történt a(z) " . $. . ".
       ➥sorban.\n";
       next;
    }
    print UJLANYOK $_;
}

close UJLANYOK;
close REGILANYOK;
unlink $ARGV[0] or die "Nem tudom törölni: " .
➥$ARGV[0] . "\n";
rename "tmp." . $$, $ARGV[0] or die "Nem tudok
➥átnevezni.\n";
print "Kész.\n";
```

Mint látható, a költséges `split() / join()` páros helyett egy egyszerű mintaillesztés történt a rekord legelső mezőjére. Ha a főciklusban az átmeneti változóval kapcsolatban hiányérzeted van, képzeld oda a `$_` változót, mely az alapértelmezett minden helyzetben. Máris látni fogod, hogy miért vonatkozik a ciklus elején, az `if` feltétel részében a mintaillesztés a teljes beolvasott sor első mezőjére.

A bemutatott megvalósítások egy további fogyatékossággal is rendelkeznek. Ha például két felhasználó egy-egy folyamattal párhuzamosan akar törölni két különböző rekordot, versenyhelyzet alakul ki. Aki gyorsabb volt, annak a törlés művelete varázslatos módon visszavonásra kerül. Erre megoldás a Perl `flock()` függvénye. Ezzel egy átmeneti zárat helyezhetünk az adatbázisra. A `flock()` két típusú zárat ismer. Az egyik a kizárólagos (***exclusive***), a másik a megosztott (***shared***). Ha egy folyamat feltette a kizárólagos zárat egy állományra, egyetlen másik folyamat sem kaphatja meg, amíg a zár fel nem oldódik. A megosztott zárat párhuzamosan több folyamat is megszerezheti, ám ez alatt kizárólagos zárat nem lehet rátenni az állományra. Többnyire csak olvasó folyamatok használják a megosztott, míg író folyamatok a kizárólagos zárat. Sok minden szerepelt, és nagyon sok más kimaradt ebből az írásból. Viszont, mint mondani szokás, a puding próbája az evés. Kísérletezz, játssz, és egyre többre fogsz rájönni. Sok sikert!

Fülöp Balázs

Újabb GIMP ismeretek

Az előző részben megismerkedtünk a különféle rajzeszközökkel, mintákkal, színekkel és színátmenetekkel, így most megtanulhatjuk, hogyan hozhatunk létre saját igényeinknek megfelelő újabb színátmeneteket. Szó lesz még néhány szűrő alkalmazásáról és az előző részből kimaradt szív is bemutatásra kerül.

Kezdjük tehát a színátmenetek létrehozásával. Amikor új átmenetet választunk az eszközkészletben (a színátmenetet jelző sávra történő kettős kattintás segítségével), akkor a felbukkanó párbeszédablakban találhatunk két gombot. Az egyikkel bezárhatjuk az ablakot, de számunkra most a másik lesz a fontosabb, amelyiken az ***Edit*** feliratot olvashatjuk. Érdemes megjegyezni, hogy az ablakot kevesebb egérmozgatással is előcsalhatjuk, mégpedig a CTRL-G billentyűk segítségével. Nagyon nagy előnyünkre szolgálhat, ha megtanuljuk a billentyűkombinációkat is, mert például ha rajztáblát használunk, akkor szintén egyszerűbb lenyomni két billentyűt, mint pozicionálni a ceruzával, majd a kettős kattintást alkalmazni. Tehát hozzuk elő a párbeszédablakot és máris kezdhetjük létrehozni vagy szerkeszteni a saját színátmeneteinket.

Színátmenet-szerkesztés

Válasszuk ki a ***Edit*** gombot, így eljuthatunk az *1. képen* látható újabb ablakhoz, amiben majd a tényleges szerkesztési műveleteket hajtjuk végre.

A GIMP-ben a színátmenetek szakaszokból állnak, minden szakasznak van egy kezdő- és egy végpontja. A két végpont között található még egy jelölés, ami alapállapotban a szakasz középpontját jelzi. Az *1. képen* az ablak alsó részén látható az aktuális színátmenet, fekete háromszögek jelzik a szakaszok határait. Az egyes részeket a szürke sávon történő kattintással választhatjuk ki. Hozzunk létre egy új átmenetet a ***New Gradient*** gomb segítségével. A név megadása után máris hozzákezdhetünk az átmenet szerkesztéséhez.

Kezdetben a létrehozott átmenet csak egy részből áll, ezt jelzi az átmenet elején és végén található fekete háromszög. Jól látható a szakasz középpontja is, fehér háromszögéről könnyen felismerhető. Alapállapotban a színátmenet kezdő színe fekete-, végpontja pedig fehér színű. Válasszuk ki az egyetlen létező szakaszunkat. A kiválasztott részen jobb gombbal kattintva egy elég bőséges lehetőségeket kínáló menüt kapunk. Itt végezhetünk el minden műveletet, ami a színátmenettel kapcsolatos.

Megváltoztathatjuk a végpontok színét, ha a menüből kiválasztjuk a ***Left endpoint's color*** vagy a ***Right endpoint's color*** menüpontokat. A szokásos színválasztó párbeszédablakban állítsuk be a kívánt színt, majd az ***OK*** gombra kattintva nyugtázzuk a műveletet.

1. kép Színátmenetek készítése

A színeket nem csak közvetlenül állíthatjuk be, mindkét végpont színét másolhatjuk különböző helyekről. A ***Load from*** menüben találhatjuk meg a különféle másolási forrásokat, úgymint a szomszédos szakasz közelebb eső végpontja (***Left/Right neighbor's right/left endpoint***), a szakasz másik végpontja (***Left/Right endpoint***), az aktuális előtérszín vagy az előre meghatározott színek egyike.

Ugyanígy az éppen beállított színt el is tárolhatjuk a ***Save to*** menüben található előre beállított helyekre.

Szintén a jobb egérgombra előbugró menüben állíthatjuk be a színátmenet típusát. A ***Blending function for segment*** almenüben választhatjuk ki, hogy lineáris, görbe által meghatározott, szinuszos, vagy másfajta átmenet legyen

a szakasz két végpontja között. Igazából a típus kiválasztásának a gyakorlatban ritkán van jelentősége (rövid részek használata esetén alig látható különbség), de mindenesetre a fejlesztők erre is gondoltak.

Szakaszok felosztása

A következő fontosabb menüpont, a szakaszok felosztására vonatkozik. Egy kiválasztott részt kettéoszthatunk a középpontjánál a ***Split segment at midpoint*** menüpont segítségével vagy több egyforma részre oszthatjuk a ***Split segment uniformly*** menüpont választásával. Ez utóbbi esetben a program megkérdezi, hogy hány darabra szeretnénk felosztani a kiválasztott átmenetrészt.
Természetesen a színátmenetből törölhetünk is részeket, mégpedig a ***Delete segment*** menüpont választásával, vagy a kiválasztott részek középpontját újra középre állíthatjuk a ***Re-center midpoints in selection*** pont segítségével. Ugyanígy, ha több részt jelöltünk ki, a ***Re-distribute handles in selection*** menüpont választásával a vezérlőpontokat egyenlően eloszthatjuk a kiválasztott területen belül.
Még két hasznos menüpontot találhatunk a helyi menüben, az egyik a ***Selection operations -> Flip selection***, mellyel a két végpont színét felcserélhetjük. A másik pedig a ***Selection operations -> Replicate selection*** menüpont, melynek segítségével ismételhetjük az éppen kiválasztott szakaszokat, miután beírtuk a kívánt ismétlések számát.
Természetesen nem kötelező minden esetben újabb színátmenet létrehozásával kezdenünk a munkát, hiszen a párbeszédablakban más gombok is a segítségünkre vannak. Ha már van egy igényeinknek majdnem megfelelő átmenet, akkor arról a ***Copy Gradient*** gombbal másolatot készíthetünk, és ezt szerkeszthetjük tovább. A ***Rename Gradient*** gombbal átnevezhetjük már meglévő színátmeneteinket, míg a ***Delete Gradient*** gomb alkalmas arra, hogy egy elkészített színátmenetet töröljünk. Mivel a GIMP készítői elég körültekintően tervezték meg a programot, találhatunk itt egy elsőre furcsának tűnő felirattal rendelkező gombot is. A ***Save as PoV-Ray*** gombról van szó, ami arra alkalmas, hogy a színátmenetet a PoV-Ray által értelmezhető formában mentsük el. Később ezt felhasználhatjuk a sugárkövetéssel számított képeinknél, ahogyan azt a *2. képen* is láthatjuk.

Kép, átalakítás

Miután megismerkedtünk a színátmenetek létrehozásának módszerével, érdemes egy kicsit elmélyedni a GIMP képátalakításra szolgáló lehetőségeiben.
A képen a jobb gombbal kattintva elővarázsolhatjuk azt a menüt, melyben a különféle szűrők találhatók. Kezdjük talán az elején, és tekintsük át a ***Blur*** pontban található különféle elmosásra szolgáló megoldásokat.
A ***Motion Blur*** segítségével olyan hatást hozhatunk létre a képen, mintha az mozgás közben készült volna. A mozgás típusa lehet egyenes vonalú (***Linear***), de beállítható olyan hatás is, mintha a kép készítése közben közeledtünk (vagy távolodtunk) volna a tárgyhoz a ***Zoom*** pont választásával, esetleg készíthetünk forgás közben látszó elmosódást is, ha a ***Radial*** módot választjuk. Minden esetben megadhatjuk az elmozdulás mértékét és az irányát, de értelemszerűen forgó mozgás meghatározásakor nincs túl nagy hatása a szög változtatásának.

2. kép A GIMP és PoV-Ray

A ***Blur*** adott sugárral és azonos súlyozással átlagolja a képpontokat, ezzel éri el a kép lágyabb megjelenítését. Például alkalmazhatjuk akkor, amikor több részletből rakunk össze valamilyen képet és az egyes részek közötti éles eltéréseket szeretnénk finomítani.
A ***Gaussian Blur*** hasonló az előbbi szűrőhöz, de ebben az esetben az egyes képpontokhoz tartozó súlyozó tényezőket a program egy Gauss-görbe alapján számítja ki.
A ***Pixelize*** talán nem újdonság, nagyon sok képfeldolgozó programban találkozhatunk hasonló megoldással. A képpontokat adott méretű négyzet alakú területekkel helyettesíti, így ha messziről nézzük a képet, akkor látható az eredeti tartalom durva közelítése.
A ***Selective Gaussian Blur*** a Gauss-görbe szerinti súlyozást az éppen feldolgozott képrészlethez igazítja a GIMP szűrő segítségével, így ahol kisebbek az eltérések, ott nem mosódik el annyira a képrészlet, mint a nagy eltéréseket tartalmazó képterületek esetében. Lényegében ennyit érdemes tudni a különféle elmosásra alkalmas szűrőkről, viszont a most következő szűrési műveletek igen hasznosak lehetnek.
Vegyük sorban a ***Color*** menüben található különféle eszközöket. Az első érdekes szűrő a ***Colorify*** ponton keresztül érhető el. A szűrőnek megadhatunk egy színt, amivel az egész képet vagy a kiválasztott területet átszínezhetjük. A meglévő színek nem vesznek el, csak egy kicsit átalakulnak a megadott színhez közelebbi tartományba.
A ***Value Invert*** a kép világosságértékeinek invertálására használható. A szűrő nem változtatja meg sem a színezetet, sem pedig a telítettséget.
Ebben a menüben, mely a színekkel kapcsolatos szűréseket tartalmazza, a következő hasznos menüpont a ***Map***. Ennek a pontnak szintén több alpontja van, így most ezekkel kell egy kicsit foglalkoznunk.
Az ***Alien Map*** és az ***Alien Map 2*** érdekes ugyan, meglepő hatásokat érhetünk el a kép különféle átszínezésével, de mindeddig gyakorlati hasznát nem láttam.
Ennél sokkal hasznosabb a ***Color Exchange***, mellyel egy adott színnek megadhatunk valamilyen tűrést mindhárom összetevőjéhez, majd az így meghatározott színtartományt másik színnel helyettesíthetjük. Így kereshetünk egy adott színű területet is, vagy csak egyszerűen kicserélhetjük valamilyen tárgy színét.

3. kép Vázlat a Városligetről

A ***Gradient Map*** az éppen aktuális színátmenet színeihez igazítja a kép színeit.
A ***Sample Colorize*** segítségével egy megadott minta színeihez tehetjük hasonlatossá a képünk színeit. A párbeszédablakban a jobb oldalon adhatjuk meg a mintát, ami lehet egy tetszőleges kép, egy színátmenet vagy a kiválasztott átmenet inverze. Ha forrásként egy másik képet adunk meg, azt még a szűrő alkalmazása előtt meg kell nyitni a GIMP-ben. A megnyitott képen ki is választhatunk valamilyen részletet, így a program csak a kiválasztott területen található színeket veszi figyelembe. Miután meghatároztuk a mintaképet vagy átmenetet, kattintsunk a ***Get Sample Colors*** gombra, így a program előnézeti ablakában már láthatjuk a hatást.
A ***Noise*** menüpont almenüin keresztül különféle zajhatásokat adhatunk a képhez.
A ***Hurl*** leginkább a színes televízió zajához hasonlatos eredményt produkál, mintha nem lenne pontosan beállítva a csatorna. A ***Noisify*** beállításainál vörös, zöld és kék csatornánként külön-külön adhatjuk meg a kívánt zaj mértékét.
A ***Pick*** hatására a program a képnek azokat a részein növeli a zaj mértékét, ahol színek és árnyalatok hirtelen változnak. Ilyen helyeken az élkereső algoritmusok éleket feltételeznek, tehát ha a zajt megnöveljük, akkor az élkeresés eredménye is jobban láthatóvá válik. Például ha egy szürkeárnyalatos képen alkalmazzuk a ***Pick***, majd az ***Edge Detect -> Sobel*** és az ***Enhance -> Sharpen*** szűrőket, végül az egész képet invertáljuk, akkor máris láthatjuk némi hasznát a zajkeltésnek, és megcsodálhatjuk első digitális vázlatunkat, ami meglepően hasonlít majd az eredeti képre.

Élek kiemelése

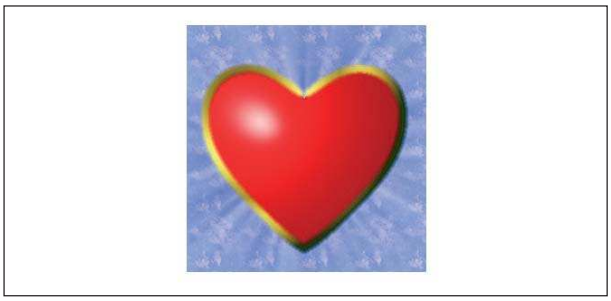
Nézzük most meg, mi is az élkiemelés. A módszer lényege, hogy ahol hirtelen változások vannak a szomszédos képpontok között, ott élt tételezünk fel és ezt más színnel jelöljük. A hirtelen változásokra úgy deríthetünk fényt, hogy a szomszédos pontok színeinek távolságát kiszámítjuk és ezt egy előre meghatározott küszöbértékhez hasonlítjuk. A „színek távolsága” talán némi magyarázatra szorul. Minden képpont egy vörös, egy kék és egy zöld összetevő alapján kapja a végleges színét. Vagyis minden szín ábrázolható egy háromdimenziós koordinátarendszerben, ahol az egyes tengelyeknek a színt alkotó összetevők felelnek meg. Az egyszerűség kedvéért a koordinátarendszer értelmezési tartományának alsó határa legyen 0. A maximális érték, amit mindhárom összetevő felvehet, legyen 1. Így a három dimenziós rendszerben a Pxyz=(0,0,0) pont a fekete színnek felel meg, a Pxyz=(1,1,1) pont pedig a fehérnek. Két szín (melyek adottak r1,g1,b1 és r2,g2,b2 összetevőikkel) távolságát kiszámíthatjuk a következő képlet segítségével:

```
D=SQRT((r2-r1)^2+(g2-g1)^2+(b2-b1)^2)
```

A képletben szereplő `SQRT` függvény a négyzetgyökvonás számítástechnikában alkalmazott megnevezése. Tehát a képen végighaladva megvizsgáljuk a szomszédos képpontok színeinek távolságát és ha az egy adott határ felett van, akkor élként jelöljük a pontot. Ezt a műveletet először elvégezzük vízszintes, majd függőleges irányban haladva az eredeti képen és a kapott eredményeket egy képben egyesítjük. Ez a módszer az egyik legáltalánosabban használt élkeresési eljárás, a GIMP-ben is megtalálhatjuk az ***Edge Detect -> Sobel*** menüpontokat követve.

Képtisztítás

A következő említésre méltó menüpont segítségével különféle zajtípusokat távolíthatunk el a képeinkről. Az ***Enhance*** menüről és annak alpontjairól van szó, ahol az első szembetűnő idegen szó a ***Deinterlace***. Ezzel akkor találkozhatunk, ha valamilyen vízszintesen nagy sebességgel mozgó tárgyról készítünk képet. Bizonyos kamerák érzékelőinek az a sajátossága, hogy a képsorokat nem egymás után továbbítják a feldolgozó egységnek, hanem először a páratlan, majd a páros sorokat olvassák ki. Ezt hívjuk váltott-soros kiolvasásnak és a hátránya az lehet, hogy a nagy sebességgel mozgó tárgyak a páros sorok kiolvasásakor már elmozdulhatnak. A képen, a kiolvasási módnak köszönhetően minden második sorban eltolódást vehetünk észre.
A probléma megoldására különféle eljárásokat dolgoztak ki, így a GIMP-ben is megtaláljuk a lehetőséget, a fent említett menüpont alkalmazásával.
A ***Despeckle*** szűrő olyan esetekben tehet jó szolgálatot, amikor különféle, nem kívánatos foltok jelennek meg a képeinken. A szűrő alkalmazásakor ezek a foltok ugyan nem tűnnek el teljesen, de elmosódottabbá válnak, így nem annyira zavaró a jelenlétük.
Az ***NL Filter*** mindhárom típusa főként természetes képek minőségének javítására alkalmas, zajos képeken látványos javulást érhetünk el a használatával.
Említésre méltó még a ***Sharpen***, ami tulajdonképpen az elmosás ellentéte, hatására a kép élesebbé válik.
Érdekes próbálgatásokra ad lehetőséget a ***Generic*** menüpont alatt található egyetlen szűrő, ahol saját magunk határozhatjuk meg, milyen súlyozással számítsa ki a program a képpontok új értékét. Érdemes vele próbálkozni egy kicsit, ha másért nem is, legalább azért, hogy jobban megértsük a szűrők működését. A táblázatba beírjuk a súlyozáshoz használni kívánt értékeket, majd a GIMP képpontról-képpontra kiszámítja a súlyozott átlagot figyelembe véve az éppen vizsgált képpont környezetét is. A képpont új színe az így kiszámított átlagos érték lesz. Figyeljük meg, hogy ha az súlyozó tényezők összege 1, akkor a kép élesebbé válik



4. kép A szív mely értünk dobog...

5. kép ...és a vagyon

(akár dombormű-szerű hatást is elérhetünk), míg ha egytől különböző pozitív szám, akkor a kép homályosabb lesz. Élkeresésre is használható a szűrő. Például ha a táblázat jobb oldalán csak pozitív számokat adunk meg, a baloldalon pedig ugyanezen számok ellentettjét, akkor a szűrő a függőleges éleket emeli ki.

A szűrő alkalmazása előtt ne felejtsük el a párbeszédablak jobb oldalán található vezérlők segítségével kiválasztani, hogy melyik színcsatornákra szeretnénk alkalmazni a megadott paramétereket.

Nos, egyelőre ennyit mára a szűrők tanulmányozása terén, és most gyorsan készítsünk egy egyszerű képet a GIMP által rendelkezésünkre bocsátott eszközök segítségével.

A gyakorlatban

Sok esetben megkímélhetjük magunkat felesleges modellezési munkáktól, ha jó elboldogulunk a színátmenetek és a kijelölések alkalmazásával.

Nem szükséges ugyanis olyan tárgyakat modellezni, amelyek képét csak egyszer vagy egy nézőpontból nézve használjuk fel. Minden eszköznek megvan a maga alkalmazási területe és ha valamit meg tudunk oldani egyszerűbben, akkor ne bonyolítsuk az életünket.

Ilyen egyszer használatos tárgyak például kétdimenziós játékokban fordulnak elő, de olyan képekről is beszélhetünk, amiket valamilyen dokumentumba illesztünk be, és azok tartalma nem változik. Más kérdés az, ha a játékunkat valamikor majd továbbfejlesztjük és térbeli tárgyakra lesz szükség. Ha ilyen terveink vannak, akkor érdemes elgondolkodni a 3D modellezésen. Ebben az esetben is lehetnek kétdimenziós nézeteink, beilleszthetjük őket dokumentumokba és a későbbiekben a befektetett idő megtérül.

Rajzoljunk szívet!

Lássuk, hogyan állítható elő a *4. képen* látható szív. A kép megtervezésének első lépése legyen, hogy egy egyszerű vázlatot készítünk az elképzelésünkről. Itt akár szöveges megjegyzéseket is használhatunk, a lényeg az, hogy megkönnyítsük a későbbi munkálatokat.

A következő lépésben megpróbáljuk elemeire bontani a képet. Esetemben látható, hogy egy háttérből, egy aranyszínű keretből és egy szívformából tevődik össze a kép. Ezt követően nagyjából körvonalazódik bennünk, hogy mennyi réteget veszünk igénybe. A fentiek alapján legalább hármat. Kérdéses lehet, hogy hogyan állítsuk elő a szív körvonalát. Javaslom a ***Bezier-select*** eszköz használatát, amelyről az előző részben bővebben szóltam.

Hozzunk létre a háttéren kívül még két réteget, és a középsőn rajzoljuk meg a szív egyik felét. Mivel a szívforma tengelyesen szimmetrikus, ezért elég csak az egyik felét megrajzolni. Ezután a kiválasztott területet fessük ki egyszínűre, majd másoljuk le (Ctrl-C) és illesszük be újra a képre (Ctrl-V). Még mielőtt a réteghez csatolnánk, tükrözzük a függőleges tengelyre, hogy a szív másik fele is elkészüljön. Helyezzük el a másik fél mellé úgy, hogy a középvonaluk mentén összeérjenek.

Következhet a keret létrehozása. Válasszuk ki az egész szívet, majd váltsunk át eggyel feljebb lévő rétegre. Itt a kiválasztott területből készítsünk keretet (***Jobb gomb -> Selection -> Border***), és az így elkészített keretet színezzük ki valamilyen tetszőleges színátmenettel. Az én esetemben az átmenet neve ***Golden*** és a GIMP beépített színátmenetei között található meg.

Váltsunk vissza a középső rétegre és válasszuk ki a szívformát. Szintén valamilyen átmenttel kell kifesteni, az én megoldásomban ez egy előtérből-háttérbe típus volt, a színek előzetes beállításával, gömbszerű átmenet alkalmazásával.

Ezek után már csak a tetszőleges háttér létrehozása van hátra, ez bizonyára senkinek nem okoz gondot.

A háttér ízlés szerint díszíthető, ha azonban internetes oldalainkon is szeretnénk használni a képet, akkor érdemes a tervezett weboldal háttérszínének megfelelőre készíteni a háttér széleit. Ugyanez a javaslatom más dokumentumokban való felhasználás esetére is.

Összegzés

Az ilyen és ehhez hasonló egyszerű grafikák elkészítése azonos módszert igényel. Röviden összefoglalva tehát a vázlat alapján rétegekre bontjuk a képet, majd az egyes rétegeket lépésről lépésre felépítjük, kihasználva a szimmetriából származó egyszerűsítési lehetőségeket.

Ugyanezzel a módszerrel készült az *5. képen* látható ikon is.

További kellemes ismerkedést és alkotást kívánok, és a következő számban ismét jelentkezem. Addig is várom az észrevételeket, kérdéseket a dzooli@freemail.hu címen.

Fábián Zoltán (dzooli@freemail.hu)
26 éves, jelenleg oktatóként dolgozik, szabadidejében szívesen foglalkozik Blenderrel, programozással és elektronikai tervezéssel. Szereti a természetet, túrázást, úszást és a kellemes baráti társaságot.

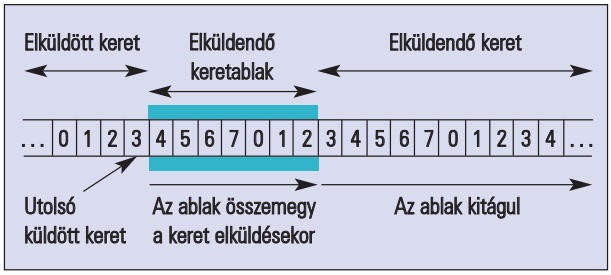
Számítógéphálózatok (9. rész)
Csúszóablakos protokollok, HDLC, SLIP, PPP

Amint azt az előző számban ígértük, ezúttal folytatjuk az adatkapcsolati protokollokkal való ismerkedést, de most már olyanokat is bemutatunk, amelyek a való életben is megállják a helyüket. Például az interneten nagy népszerűségnek örvendő SLIP és PPP.

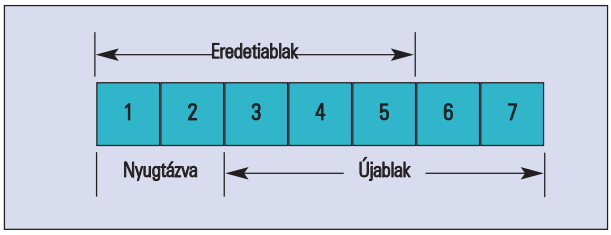
Előző írásunkban elemi adatkapcsolati protokollokkal foglalkoztunk. Sokféle keretátviteli módszert megvizsgáltunk, beleértve a ráültetéses ***(piggybacking)*** módszert, amelynek köszönhetően csökkenthettük a kimenő keretek számát.
Egy csatorna kihasználtságát azonban nem csak úgy növelhetjük, hogy kevesebb keretet küldünk. Az eddig ismeretett összes protokollban az volt a közös, hogy a csatornán egy időben mindig csak egy adatkeret, majd egy azt követő nyugtakeret volt.
Sokkal hatékonyabb lenne, ha a csatornán egyszerre több keret is haladhatna. Az úgynevezett csúszóablakos ***(sliding windows)*** protokollok ezt teszik lehetővé.

Csúszóablakos protokollok

A csúszóablakos protokoll a következő elven működik. Minden elküldött keret tartalmaz egy sorszámot, amely a 0 és valamilyen maximális érték közé esik. A forrás minden elküldött keret sorszámát beteszi egy halmazba. Ez a halmaz az úgynevezett adási ablak ***(sending window)***, amelyhez azok a keretek tartoznak, amelyeket ugyan már elküldtünk, de a vevő még nem nyugtázta őket. Amikor a hálózati rétegtől új csomag érkezik, akkor az adatkapcsolati réteg kiosztja neki a következő sorszámot, majd az ablak felső szélét feljebb csúsztatja.
Vessünk egy pillantást az *1. ábrára*. Itt a kioszthatő sorszámok 0 és 7 közé esnek. Az adási ablakban jelenleg a 4., 5., 6., 7., 0., 1. és 2. keretek vannak. Ezeket már útjukra bocsátottuk és csak a nyugtákra várunk. Amikor egy nyugta megérkezik, akkor az ablak alsó széle eggyel feljebb csúszik, így küldhetünk további kereteket.
A vevő is karbantart egy listát, amelyet vételi ablaknak ***(receiving window)*** nevezünk. Ebben azon keretek sorszámai szerepelnek, amelyeket elfogadhat. Ha egy olyan keret érkezik, amelynek a sorszáma nincs a vételi ablakban, azt azonnal eldobja. Ha az érkezett keret sorszáma mégis benne van az ablakban, a fogadó csak akkor küld nyugtát, ha egyrészt az adott keret még nem került nyugtázásra, másrészt ha minden olyan keretet vettünk, amelyek sorszámai az elsőnek várt és a most érkezett közé esett.



1. ábra Adási ablak



2. ábra Vételi ablak

Szegezzük tekintetünket a *2. ábrára*! Az aktuális ablak első eleme a 3. Ha mondjuk beérkezik az 5. sorszámú keret, csak akkor küldünk róla nyugtát, ha már beérkezett a 3. és az 5. is. Ha pont egy olyan keret érkezik, amelynek sorszáma pont az ablak alsó szélével egyenlő, akkor arról azonnal küldhetünk nyugtát, és az ablakot eggyel elcsúsztathatjuk.
Vegyük észre, hogy a vevőnek nem kell minden keretet nyugtáznia. Visszatérve a *2. ábrára*, ha a forrás megkapja a 6-os kerethez tartozó nyugtát, akkor az egyben a 4-es és 5-ös keretek nyugtázását is jelenti.
Egy másik érdekes dolog, hogy míg az adási ablak mérete folyamatosan változik, addig a vételi ablak mindig ugyanakkora marad. Az adási ablak mérete azonban sohasem mehet a kioszthatő legnagyobb sorszám fölé (ami jelen esetben 7). Ha eléri a maximális méretét, akkor az adatkapcsolati rétegnek „le kell kapcsolnia” a hálózati réteget, azaz nem szabad engedni, hogy az további csomagokat adjon át továbbítás végett.

Mivel a keretek útközben elveszhetnek, a forrásnak az adási ablakba eső kereteket a memóriában kell tartani. Minden kerethez indítania kell egy időzítőt, amelynek lejártáig a nyugtának meg kell érkeznie. Ha ez mégsem történik meg, akkor a keretet ismét el kell küldeni.

Egybites csúszóablakos protokoll

Ez az összes ilyen jellegű protokoll között a legegyszerűbb. Itt minden ablak mérete maximum 1 lehet. A protokoll működése sokban hasonlít az előző részben bemutatott megáll-és-vár protokolléhoz.

Igaz, hogy itt az adatkeretek mindkét irányba mehetnek, de a következő keret csak akkor kerül elküldésre, ha az előzőt a vevő már nyugtázta. A nyugtát azonban hordozhatja egy ellenirányú adatkeret. Mivel egy keret indításának feltétele, hogy az előző nyugtázva legyen, ezért a sorszám értéke csak 0 vagy 1 lehet.

Nézzük, hogyan is működik ez a protokoll.

Legyen a kommunikációban részt vevő két gép neve A és B. Először az A kezdi meg az adást, így elküldi az első adatkeretet B-nek. Az adatkeretbe betesz egy nyugtát, ezzel elhitetve B-vel, hogy az előző keret küldése sikeres volt, így B is elküldi az első keretet (amely egyben A első keretének a nyugtája is). Ezután A veszi B keretét, majd küldi a következőt. De mi történik akkor, ha elveszik egy keret? Például az A elküldte a 0-ás sorszámú keretet B-nek, ami rendben meg is érkezik, a nyugta azonban elvész. Így előbb vagy útóbb lejár A időzítője, és ismét elküldi ugyanazt a keretet. B vételi ablakába azonban már az 1-es keret esik, így a duplikátumot szó nélkül eldobja.

Ezután a B küld A-nak egy keretet, amelynek sorszáma 1, és egyben a 0-s keret nyugtája. Amikor ez megérkezik A-hoz, A elküli a következő keretet. Láthatjuk tehát, hogy hiába veszik el bármelyik keret, a protokoll biztosan nem fog kihagyni egyet sem, és nem fog kettőzött keretet átadni a hálózati rétegnek.

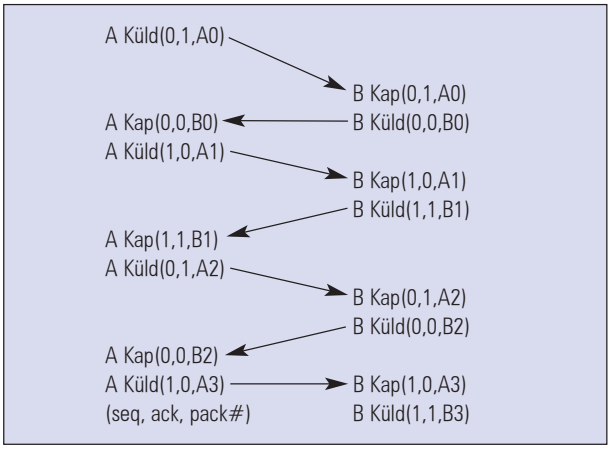
Duplikátumok küldéséhez vezethet azonban az a sajátos eset, amikor A és B az első keretet egyszerre küldik el. Az első keretek ugyanis 1-es nyugtát tartalmaznak a várt 0 helyett, ezért duplikátumok keletkeznek *(3. ábra)*.

Az n-el visszalépő és a szelektív ismétlő protokoll

Az egybites csúszóablakos protokoll csak abban az esetben lehet hatékony, ha a keret és a visszaküldött nyugta célbaérésének együttes ideje elhanyagolható. Ez azonban nem mindig van így. Műholdas átvitel esetén például a keretek átviteli ideje olyan nagy, hogy a fent bemutatott módszerrel a sávszélességnek csupán 4%-át használhatjuk ki (ha 50 kb/sec-os műholdas csatornát használunk).

Ezen úgy tudnánk javítani, ha nem követelnénk meg az adótól, hogy csak akkor küldhet el egy keretet, ha az előzőt a vevő már nyugtázta. Engedjük meg inkább azt, hogy az adó egyszerre több, legfeljebb m darab keretet bocsáthasson a csatornára, majd csak ezután blokkoljon addig, amíg a nyugták meg nem érkeznek.

Ha ezt az m-et ügyesen választjuk meg, akkor az első keret körülfordulási ideje alatt folyamatosan tudunk további kereteket továbbítani anélkül, hogy betelne az adási ablak. (Egy keret körülfordulási idején a keret elküldésétől a nyugta beérkezéséig eltelt időt értjük).



3. ábra Példa az 1 bites csúszóablakos protokoll működésére

Maradjunk az előző példánál, melyben egy 50 kb/sec-es műholdas csatornán szeretnénk kereteket továbbítani. Tegyük fel, hogy egy teljes keret elküldése 20 ms-ig tart, és egy keret körülfordulási ideje 500 ms. Nézzük meg, mi történik ha az előző protokollt használjuk. Ha a kezdő pillanatban elküldjük az első keretet, akkor a 270-edik ms-ban érkezik meg a vevőhöz.

Ha a vevő valamiféle mesebeli masina, amely várakozás nélkül képes a nyugta visszaküldésére, és a nyugta is meglehetősen rövid (legalább annyira, hogy nem kerül időbe a csatornára ráhelyezni), akkor pontosan 520 ms múlva érkezik a forráshoz a nyugta.

Ekkor küldhetjük csak a második keretet, azaz körülbelül félmásodpercenként küldhetünk egyet. (Feltéve ha nem veszik el útközben).

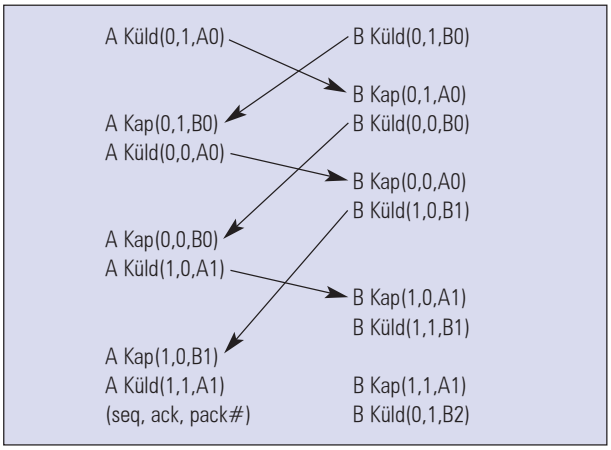
Ha megengedjük, hogy az adó egyszerre több keretet is küldhessen, akkor sokkal jobb eredményt érhetünk el. Ha továbbra is 20 ms-ig tart egy keret elküldése, akkor 520 ms alatt 26 keretet küldhetünk. Mikor a 26. keretet elküldtük, akkor meg is érkezik az első keret nyugtája. Így ha az m értékét 26-nak választjuk, mindig akkor kapjuk meg az engedélyt a következő keret elküldésére, amikor kell.

Mivel mindig 25, illetve 26 nyugtázatlan keret van kint, ha az adási ablak maximális méretét 26-nak szabjuk meg, az sohasem fog betelni (az adó nem fog blokkolni). Így 25-ször nagyobb hatékonyságot értünk el, mint az egybites csúszóablakos protokoll használatával. Ez a techinka csővezetékezés ***(pipelining)*** néven híresült el.

Ez így nagyon meggyőző, de érdemes belegondolni, mi történik akkor, ha a csatorna zajos, és megsérülnek, esetleg el is vesznek egyes keretek.

Az adó csak azután értesül a sérült keretről, miután sok másikat továbbított. Egyértelmű, hogy a vevő egy sérült kerettel nem tud mit kezdeni, de mit csináljon azokkal, amelyek a sérült keret után érkeztek? Ez azért gond, mert az adatkapcsolati réteg a kereteket csak érkezésük sorrendjében adhatja át a hálózati rétegnek.

Erre a problémára két megoldás is létezik. Az első a visszalépés n-el ***(go back n)***, amikor is a vevő eldob minden olyan keretet, amely a sérült keret után érkezett. Mivel nyugtát sem küld vissza róluk, az adó ezeket a kereteket is meg fogja ismételni.



4. ábra 1 bites csúszóablakos protokoll, ha mindkét fél egyszerre kezd el adni. Ilyenkor duplikátumok lépnek fel.

Ha belegondolunk, ez a megoldás nem más, mint egy 1 hosszúságú vételi ablak használata, vagyis a vevő a kereteket kizárólag sorrendben fogadja el. Addig nem foglalkozik a továbbiakkal, amíg az éppen soron következő épségben meg nem érkezik.

Nagyobb átviteli ablak

A másik módszer értelemszerűen nagyobb méretű átviteli ablak használata. Ezt szelektív ismétlésnek ***(selective repeat)*** nevezzük.
Ilyenkor a vevő az összes olyan keretet tárolja a memóriában, amelyek a hibás keret után érkeztek. Az adónak így csak az elveszett vagy megsérült keretet kell megismételnie. Ha az újra elküldött keret már rendben megérkezik, akkor a vevőnek sok-sok hibátlan, ám még nem nyugtázott kerete lesz. Ezeket érkezésük sorrendjében át tudja adni a hálózati rétegnek, majd nyugtázza a legnagyobb sorszámút. Így az adó tudni fogja, hogy az ennél a sorszámnál korábbi keretek hibátlanul célba értek.
Mindkét módszernek megvan a maga hátránya. Az n-el történő visszalépéses stratégiánál ez nyilvánvaló: ha nagyon zajos a csatorna (sok kerethiba lép fel), akkor az a hatékonyságból jelentősen visszavesz.
Az utóbbi módszernél a gond az, hogy minden beérkezett keretet tárolni kell a memóriában, egészen addig, amíg az adott keretnél korábban fogadottak át nem kerülnek a hálózati réteghez.
Hogy a két megoldás közül melyiket célszerű használni, az attól függ, hogy mi a fontosabb számunkra: a sávszélességet a lehető legjobban kihasználni, vagy inkább a memóriával spórolni.

Magas szintű adatkapcsolat-vezérlés

Most bemutatunk egy kicsit öreg, ám ma is elterjedt protokollt, a HDLC-t ***(High-level Data Link Control – magas szintű adatkapcsolat vezérlés)***. Ez egy bit alapú protokoll, amelynek a keretezési eljárását sok más protokoll, például a PPP is átvette (igaz, néhány kisebb különbséggel).
A HDLC-t azokban az időkben fejlesztették ki, amikor a kommunikációban résztvevő két fél általában egy terminál és egy „okos” számítógép volt.
A HDLC (és a hozzá hasonló bitalapú protokollok) az *5. ábrán* látható keretformátumot használják. Minden keretet egy speciális bitsorozat határol, amely a 01111110 (hexadecimálisan 7E). A cím mező olyan kapcsolatoknál érdekes, ahol terminál is csatlakozik a vonalra, és valahogy meg kell határoznunk, hogy az üzenet melyik terminálnak szól.
A vezérlés nevű mezőt sorszámozásra, nyugtázásra és egyéb hasznos dolgokra használjuk, erről majd egy kicsit később. Az adat mező értelemszerűen az átküldendő információt tartalmazza. Ennek nincs meghatározott mérete, csupán egy maximuma. Az ellenőrző összeg a hibakeresésre, illetve a hibajavításra szolgál.
A HDLC olyan csúszóablakos protokoll, amely 3 bites sorszámozást használ. Ez azt jelenti, hogy egyszerre 7 nyugtázatlan keret lehet „kint”.
Háromféle keret létezik: információs, felügyelő és számozatlan. Az 5. ábrán láthatjuk e háromféle keret vezérlés mezejének tartamát. Nézzük először az elsőt! A sorszám mező a keret sorszáma, az utolsó mező pedig a keretre ültetett nyugta. A lekérdezés/utolsó mezőt a HDLC-re épülő protokollok más és más célra használják. Néhol ezzel lehet a másik gépet kényszeríteni arra, hogy azonnal küldje a nyugtát, és ne várjon addig, amíg azt nem tudja ráültetni egy visszafelé menő keretre.
A felügyelőkereteket egymástól a felügyelő kód mező alapján lehet megkülönböztetni. Négyféle felügyelő keret létezik. Az első típus egy olyan nyugta, amelyet nem lehetett ráültetni egy ellenkező irányba menő információs keretre.
A második típus az úgynevezett negatív nyugta, amely arról árulkodik, hogy érkezett ugyan keret, de átviteli hiba miatt sérült volt.
Ilyenkor az adónak újra el kell küldeni az összes keretet a „következő” mezőben lévő sorszámtól kezdődően.
A harmadik típusú felügyelőkeret a „vételre nem kész”. Ez ugyan nyugtázza az összes idáig elküldött keretet, de utasítja az adót, hogy többet ne küldjön. Ez akkor hasznos, ha a vevőnek valamiféle átmeneti problémával kell megküzdenie, például elfogyott a memóriája a további keretek tárolásához.
Az utolsó típus a szelektív elutasítás, amely az adótól csak egy bizonyos keret újraküldését kéri.
Nem szóltunk még a keretek harmadik csoportjától, a számozatlan keretekről. Ezek hordozhatnak adatot és vezérlési információt is. A különbség az előző kettővel szemben az, hogy ezeket a kereteket nem kell nyugtázni.

Az Internet adatkapcsolati rétege

Az Internet önnálló gépek sokaságából áll, amelyeket durván két részre oszthatunk: hostokra és útválasztókra. Ezenkívül az Internethez tartozik még az az infrastruktúra, amely ezeket a gépeket összeköti. Kis távolságok esetén (például egy épület belsejében) általában LAN-okat alkalmaznak. Egymástól messze lévő útválasztók azonban úgynevezett kétpontos összeköttetésben vannak. Ilyen kétpontos összeköttetést létesíthetünk például egy bérelt vonal segítségével.
Kétpontos összeköttetést nem csak útválasztók összekapcsolására használunk. A másik legjellemzőbb felhasználási terület az, amikor az egyéni felhasználók otthoni számítógépüket modem segítségével az internetszolgáltatójuk

(ISP – Internet Service Provider) útválasztójához kapcsolják. Ebben az esetben a felhasználó számítógépe egy „teljes értékű” host lesz, persze csak addig, amíg a kapcsolat él. Legyen szó akár két útválasztóról, amely egy bérelt vonallal van összekapcsolva, vagy egy otthoni PC-ről, amely telefonvonalon keresztül összekapcsolódik az ISP útválasztójával, minden esetben szükség van valamilyen adatkapcsolati protokollra, amely a keretezést végzi. A továbbiakban két az internet világában népszerű adatkapcsolati protokollról lesz szó: a SLIP-ről és a PPP-ről.

SLIP (Serial Line IP – Soros Vonali IP)

Ez egy elég régi protokoll. 1984-ben fejlesztették ki azért, hogy munkaállomásokat tudjanak csatlakoztatni az ARPANET-hez modem segítségével. A SLIP nem is tud semmi egyebet. Nagyon egyszerűen működik. A hálózati rétegtől kapott IP csomagokat szépen sorban átküldi, a keretek végét pedig egy speciális jellel zárja.

Ha az a speciális jel esetleg előfordul a keretben lévő IP csomagban, akkor a régebben bemutatott karakterbeszúrás egyik speciális módszerét alkalmazza: a speciális jelzőbájt helyett egy másik két bájtos sorozatot küld. Egyes SLIP megvalósításokban a keretek is egy speciális bájttal kezdődnek. Az idő előrehaladtával megjelent a SLIP-nek olyan változata is, amely megpróbálja a TCP és az IP csomagok fejlécét tömöríteni. Mivel az ilyen csomagok fejléceinek sok mezője megegyezik (erről majd később lesz részletesen szó), megoldható, hogy a fejlécnek csak egy részét küldjük át.

A SLIP rendkívül népszerű még ma is, sok alkalmazás támogatja, ám sosem volt az Internet szabványos protokollja. Ezért van az, hogy sokféle, egymással nem teljesen kompatíbilis változata létezik. Vannak azonban más súlyosabb problémák, amelyek miatt a SLIP használata erősen behatárolt.

A SLIP Problémái

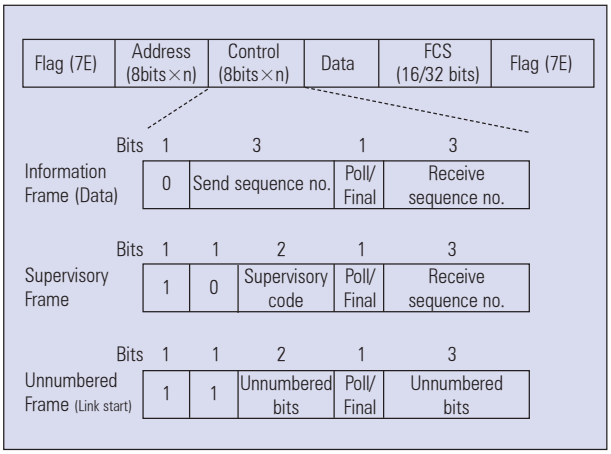
Az első ilyen az, hogy nem tud hibakeresést, illetve hibajavítást végezni. Ezért a felső rétegekre hárul az a hálátlan feladat, hogy ellenőrizzék és kijavítsák a hibákat.

Ezen kívül a SLIP feltételezi, hogy a kommunikációban résztvevő két fél tudja egymás IP címét. Magyarán nincs mód a kapcsolat létrejöttekor az IP cím dinamikus módon történő meghatározására.

Ez nem jó hír az otthonról modemmel csatlakozóknak, hiszen nem kaphat minden ügyfél a szolgáltatójától egyedi IP címet. Ráadásul még a hitelesítésre sincs mód, tehát az egyik fél sosem lehet teljesen biztos abban, hogy valójában kivel is beszél. Az utóbbi két dolog persze két útválasztó bérelt vonalas összeköttetésekor nem jelent gondot. Az viszont már kellemetlen, hogy a SLIP csak az IP alapú hálózatokkal hajlandó együttműködni. Pedig ma már sokféle „színű és szagú” hálózat tartozik az Internethez, és ezek közül nem mind épül az IP-re (például a Novell hálózatok is ilyenek voltak, még az 5-ös változat előtt).

PPP (Point-to-Point Protocol – Pontól – pontig protokoll)

Mivel a SLIP-pel ennyi probléma volt, sokkal kézenfekvőbbnek tűnt egy olyan új protokoll létrehozása, amely kiküszöböli ezeket a hátrányokat, és „méltó” arra, hogy internet-szabvánnyá válhasson. Így született hát a PPP, amely nemcsak képes a hibajelzésre, de többféle hálózatot is támogat, és lehetőséget biztosít a dinamikus IP cím hozzárendelésre, illetve a hitelesítésre is.



5. ábra HDLC protokoll kereteinek felépítése

A PPP legfontosabb feladata a keretezés, vagyis az, hogy a kereteket egymástól egyértelműen elkülönítse és kijavítsa az átviteli hibákból származó sérüléseket. A PPP ezen kívül tartalmaz két alprotokollt is.

Az első az LCP ***(Link Control Protocol – adatkapcsolat vezérlő protokoll)***, amellyel magát a kapcsolatot tudjuk irányítani. Ez alatt olyan dolgokat kell érteni, mint például a kapcsolat bontása, tesztelése, paraméterek beállítása, stb. A másik alprotokoll az NCP ***(Network Control Protocol)***, amellyel a hálózati réteg protokolljának bizonyos beállításait változtathatjuk meg. Az IP esetében ilyen lehet a dinamikus IP-cím hozzárendelése.

PPP a gyakorlatban

Hogy a dolog érthetővé váljon, nézzük meg, mi történik, amikor otthon modemmel csatlakozunk szolgáltatónk útválasztójához. A dolog ott kezd érdekes lenni, amikor a felhasználó és a szolgáltató modeme „összesípolt”, és létrejön a kapcsolat.

Ilyenkor kezdi meg a munkáját a PPP. Először LCP protokoll-csomagok indulnak útnak, amelyek a PPP keretek adatmezejében foglalnak helyet. Az LCP segítségével a kommunikációban résztvevő két fél megállapodik az alkalmazandó PPP paraméterekben.

A második lépés a PC-nk hálózati rétegének beállítása. Több mint valószínű, hogy TCP/IP protokollkészletet szeretnénk majd használni, ezért szükségünk lesz egy IP címre. Mivel a szolgáltatóknak általában kevesebb IP címük van, mint ügyfelük, ezért bejelentkezéskor dinamikusan osztanak ki egyet a felhasználónak. Az aktuális címet az NCP protokoll segítségével fogjuk megkapni.

Nem minden hálózat használ kétpontos összeköttetést. A LAN-ok például adatszóró csatornára épülnek.

A következő részben az ilyen csatornák protokolljaival foglalkozunk.

Garzó András
garzo@interware.hu

Játékos pingvinek

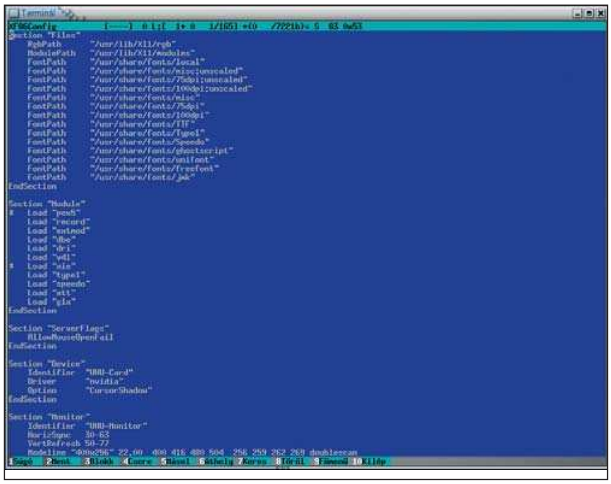
Aki játékok futtatására kívánja használni linuxos gépét, néhány jó ötletet meríthet Zoltán cikkéből.

Nincs más hátra, mint megkoronázni az eddigi munkánkat, és a szórakozást is előtérbe helyezni egy kicsit. A jó munka gyümölcseként, és befejezésként építsünk játékgépet linuxos gépünkből. Mégis, mit érdemes ilyenkor figyelembe venni?

Az alkatrészek

Aki játszani is szeretne a gépén, az szembesül azzal ténnyel, hogy korunk játékainak sokkal nagyobb erőforrás igényei vannak, mint egy szövegszerkesztőnek, vagy egyéb általános programnak. Ez korántsem újdonság, azonban Linux esetében nem árt körültekintőnek lenni. Sajnos még mindig nem jutottunk el odáig, hogy a gyártók végre figyelembe vegyék azokat is, akik nem csak egy ablakon szeretnek nézelődni. Ez úgy tűnik egy darabig még nem fog változni, legalábbis addig, amíg az adott gyártónak olyan konkurenciája nem akad, amely esetében már nem mindegy, hogy kinél az előny. Ekkor szükséges lesz minden felhasználó megnyerése, hisz a késhegyre menő harcban mindenki számít. Ennek lehettünk tanúi az nVidia vs. Ati, vagy a játék piacon az ID Software vs. Epic (quake, doom és Unreal sorozat) esetében. De addig is, amíg ez bekövetkezik, körültekintőnek kell lenni.

Tény, hogy erős gépre lesz szükségünk, de mindez attól is függ, hogy mégis milyen játékot szeretnénk futtatni. Ez ügyben tudom javasolni a Linuxvilág játék rovatát, ahol részletesen megemlítettem a gépigényt is egy adott játéknál. Természetesen minél nagyobb lélegzetű játékkal szeretnénk játszani, annál erősebb gép kell. Ez persze korántsem jelenti azt, hogy a régi, vagy picike játékok ne volnának jók. Sőt, olykor jobbak mint az újak. Aki viszont szeretné majd megvenni a cikk írásakor éppen megjelenés előtt álló Doom 3-at, (ahogy én is) annak tárgytalan a sebességre vonatkozó minden adat, és szempont, ugyanis a kérdés leegyszerűsödik. Azt vedd, ami jelenleg a legerősebb. Viszont a helyzet olykor nem ilyen egyszerű. A Linux jelenleg még mindig nagyon rosszul áll játékvezérlők tekintetében. („hála” a gyártóknak). A botkormányok üzembe



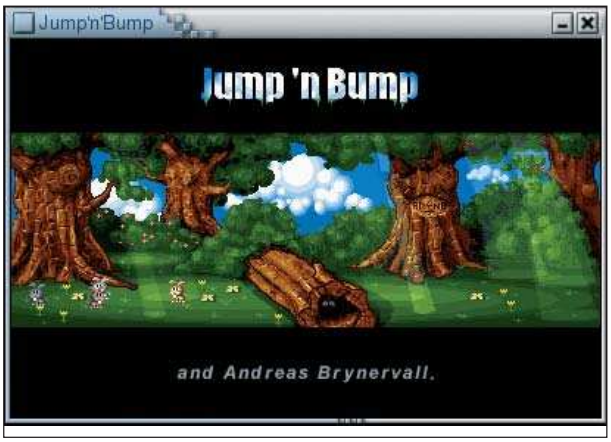
helyezése korántsem lehetetlen, de ez csak egyes típusokra vonatkozik. Gyakorlatilag kizárólag külső cégek által írt meghajtó található meg az interneten, és az sem biztos, hogy rendben van. Én például akkor tettem le a botkormány használatról, amikor kifejezetten egy, a Linux által tá-

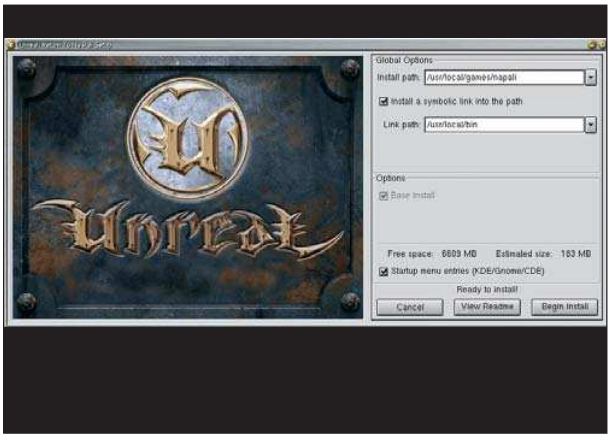
mogatott típust megvettem. A meghajtó lefordult, de alig működött. Sajnos ez könnyen megeshet. Ráadásul alaposabban kell megnézni a típusát, ugyanis a Linux alatti használathoz olykor a változatszám, meg SN, és hasonló azonosságokat kell keresni. (akárcsak a tévékártyánál és webkameránál...) A másik ilyen problémás eszköz a gamepad. Szintén nem egyszerű, de valamivel jobbak az esélyeink. Természetesen sok extráról le kell mondanunk, hisz hiába működik esetleg a gép, ha a ma már alapnak tekinthető programozhatóság nem támogatott. De ugyanez érvényes a kormányra, pedálra is, amelyet autós szimulátorok esetében szoktak beszerezni a játékosok. Az eszköz vásárlásakor feltétlenül meg kell nézni, hogy az adott típus milyen szinten támogatott – ha ugyan támogatott – Linux alatt. Sajnos ha ezt elfelejtjük, rendkívül sok bosszúságot szerezhetünk magunknak. Kiemelendő, hogy ez csak a különleges eszközökre vonatkozik. A többi, játékhoz szükséges eszközre már nem.

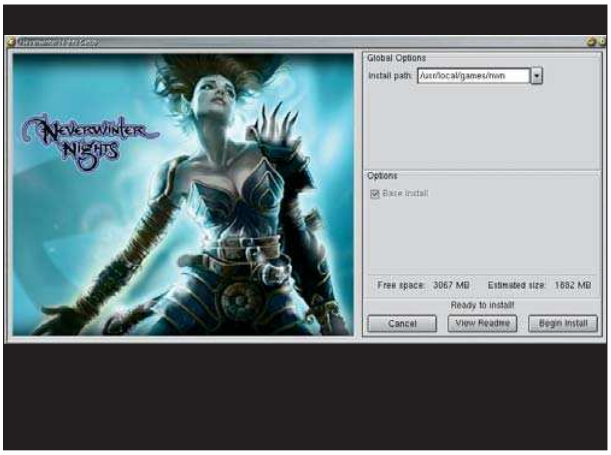
A jelenleg kapható, a játékosok által leginkább ismert kártyák már nem jelentenek problémát. Az nVidia remek módon támogatja a termékeit, kulturált, és ma már kiforrottnak is tekinthető meghajtói vannak. Az Ati is támogatja a Linuxot, csak sajnos nem olyan magas szinten, mint az nVidia. Kicsit bütykölni kell olykor a meghajtót, hogy hajlandó legyen megjeleníteni a 3D-t, ami nem kifejezetten jó pont. Engem is csak a kiforratlan támogatottság tartott vissza az Ati-tól, és azt hiszem egy darabig még ez így is marad. A többi téren viszont semmi fontosra nem kell kitérni, hisz alapértelmezettként is rendben van. Ilyen a hangkártya támogatottság. Ma már egy SoundBlaster Live Audity, vagy Player, esetleg Live 5.1 szinte alapból megy a linux telepítése után, így semmi trükközésre nincsen szükség. Legalábbis a meghajtó szintjén nincsen, azonban a szoftveres részben erre még visszatérünk. Tökéletesen szoktak működni a görgős, optikai, drótnélküli egerek billentyűzetek is. Utóbbi esetben a multimédia-billentyűk szorulhatnak egy kis bütykölésre.

A szoftveres feltételek

A rendszerre telepített meghajtók után lehetőségünk van finomhangolni őket. Ennek mikéntjéről a meghajtó leírása tájékoztat. Dokumentáció tekintetében a két nagy videókártya-gyártó közül ismét az nVidia kerül ki győztesen. Ugyanis lehetőségünk van letölteni az nVidia honlapjáról a leírást, pdf formátumban. Átolvasva rögtön kiderül, hogy mennyi lehetőségünk van a kártya beállítása során. Pl: lehet állítgatni az AGP-hez való viszonyát, skálázni a szín- és mintázatértékeket, a kártyára vagy az operációs rendszerre bízni az egérmutató megjelenítését, esetleg árnyékolását. Rendkívül jó és részletes a leírás, letöltését minden nVidia kártyatulajnak javaslom. A másik, amire érdemes odafigyelni, a GLX. Az nVidia vezérlője a modulok létrehozásakor ezt is elkészíti és telepíti, de ha gyanús, hogy esetleg azzal van a gond, célszerű megnézni, hogy engedélyeztük-e. Igaz ugyan, hogy az UHU-Linux alapértelmezett beállítsa az engedélyezett GLX, de érdemes ránézni. Főleg, hogy ugyanitt tudjuk skálázni a kártyát. Az útvonal: ***/etc/X11/XF86config***







Itt az `nvidia` modult használjuk az `nv` helyett. Közvetlen alatta lehetőségünk van saját beállítást elhelyezni, ha nyitunk egy sort, `Option "érték"` kezdettel. Az értékeket, a leírás tartalmazza. Természetesen minden beállítás érvényesítéséhez újra kell indítani az X-kiszolgálót. A hangkártyánál kicsit más a helyzet. Amit esetleg alapból nem támogatna Linux, ahhoz lehet, hogy van külső vezérlő. Igaz nem túl sok típus ilyen. Ha viszont már támogatott darabunk van, akkor gyakorlatilag semmi akadálya a használatának, a játékok is rögtön megszólalnak. Egyetlen esetben viszont nem: számtalanszor tapasztaltam, hogy a hangok olykor bizony nem hallhatóak. Sokáig kerestem a bűnöst, míg egyszercsak megtaláltam: a KDE alapértelmezett egyik cso-

magja, a KDE-multimédia tartalmazza az „ArtSD” szervert. Egy ez kis hangkiszolgáló, és mixer is egyben, aminek túl sok gyakorlati haszna nincsen, kivéve talán, hogy teljesen lefoglalja a kártyát, így a legtöbb játék nem is tudja használni. És az is meglepő, hogy nem minden kártya esetében fordul ez elő. Az én SB 128 PCI kártyámnál például igen. De kollegáim SB live sorozatú kártyájánál ilyen gond nincsen. A problémát több módon tudjuk megoldani. Ha ragaszkodunk a KDE használatához, akkor a „KDE vezérlőpult/hang/multimédia/hangszolgáltatás” részben ne engedélyezzük a hangokat. Ezzel sajnos még nincsen minden megoldva, mivel a KDE erőlteti a dolgot. A siker érdekében gondosan át kell társítanunk a fájlokat is, hogy mpeg, avi filmeknél, vagy mp3-nál ne a Kabodlee induljon el, ugyanis ő rögtön elindítja az `artsd`-t is. Ekkor külön meg kell ölni szegényt, ha a film után játszani is szeretnénk.

A helyigény

Aki gyakran játszik, ráadásul különböző játékokkal, olykor szembesülhet a helyigény kérdéskörével. Talán első látásra nem is olyan nagy gond ez, hisz a 80-100-200 GB-os merevlemezek korában élünk, de nem mindenki rendelkezik ekkora tárhellyel. A legtöbb gép alapértelmezettként még mindig 20-40 GB hellyel rendelkezik, amin terpeszkedik a rendszer, a külső programok, olykor filmek, dokumentumok. Rutinosabb felhasználók tudják, hogy egy ilyen tárméretű merevlemeznek körülbelül a fele áll majd a játékok rendelkezésére. És ha azt hisszük, hogy a 20 GB hely az nagy, hát gyorsan meglepetés érhet minket. Én például nem szeretem az ilyen kompromisszumokat, hogy mit töröljek le, ha mást is szeretnék feltenni. Így ha belegondolunk, hogy a Neverwinter Nights teljes szett 5 CD, a Conflict Freespace 2 CD, az összes Unreal együtt az szintén 5 GB körül van, és ezekhez még nem számoltam hozzá a mentéseket, akkor beláthatjuk, szükséges lehet egy másik merevlemez üzembe állítása is. Én csak erre a célra raktam a gépembe külön 20 GB-ot, amin a játékok terpeszkednek. Szerencsére a Linux rugalmassága itt is megmutatkozik.

Ugyanis a játékok telepítői kizárólag a fő lemezrészt (/) hajlandók elfogadni, de a telepítés után szabad a kezünk. Sőt, nem egyszer csináltam azt, hogy a telepített játékot becsomagoltam, és ha ráfért egy CD lemezre, rögtön ki is írtam. Ezek után a megoldás már kézenfekvő. A másik merevlemezre másoljuk föl a játékot, így jó darabig lesz helyünk mindenre. Nekem így van elhelyezve közel egy tucat játék. Természetesen a fő lemezrészre történő telepítés után is átmozgathatjuk az állományokat, nem szükséges előtte CD-re írni. Persze a jövőben jól jöhet. Azt is érdemes szem előtt tartani, (főleg maximalistáknak) hogy a merevlemez fizikai adottságai miatt lassul a rendszer, ha megpakoljuk azt. Ennek egyszerű okai vannak. A merevlemez felépítése miatt, ha sok fájl található rajta, megnövekszik az elérési idő, amíg megtalálja amit keres. A nagyon tele levő merevlemeznél (olyan 60%-os kihasználtság esetén) ez már látható jelenség, kiváltképp, ha nagyon töredezett. Ezért szokták javasolni egy rendszer telepítéséhez, hogy lehetőleg a lemez elejére tegyük, (hisz ott kell a fejnek a legkevesebb utat bejárnia), és ha tudjuk kerüljük a nagyszámú kis fájlok használatát. (megfigyelhető, hogy egy 300 MB méretű nagy fájlt sokkal gyorsabban fog átmásolni egy másik lemezrészre, mint 1000 jpg fájlt 300 MB értékben.) A kis fájlok emellett növelhetik a töredezettséget.

A játékok minősége

Sokan hivatkoznak arra, hogy Linux alatt nagyon kevés játék fut. Valóban sokkal kevesebb, mint amennyit kiadnak évente (vagy kiadtak eddig), de a helyzet azért nem ennyire vészes. A számokra lehet hivatkozni, de mint minden számításból ebből is kihagyják az emberi tényezőt. Ugyanis amit eddig átültettek Linuxra, azt azért tették, mert roppant népszerűek. Illetve szeretnék, ha a linuxos felhasználók körében is népszerű lenne. (például az UT200x darabjai, amelyek már „in box” linuxos telepítővel is rendelkeznek.) Magyarán a többség kedvenc játéka már fut Linuxon is. Körülbelül 30 olyan játék átirata létezik, amely valamilyen okból nagy népszerűségre tett szert, illetve többször előkerül a fiókból még manapság is, ha éppen nosztalgiázni támad kedve az embernek. És ez a szám folyamatosan nő. Tehát már régen nem igaz az a meglátás, hogy aki Linuxon szeretne játszani, az valamilyen picike ugrálós, arcade, vagy esetleg konzolos programot kellene használnia. A 3D egyre jobban működik, és a két legnagyobb játékgyártó (ID, Epic) már utat mutatott minden más gyártónak is. (A Bioware például már vette a lapot.) A fejlődés tehát most sem állt meg, és a jövőben is számíthatunk remek átiratokra, vagy külön linuxos fejlesztésekre.

Következtetés

Mint a bevezetőben említettem, a Linux még mindig nem ideális játékfelület. Azonban már messze nem igaz az, hogy olyan rosszul áll a játékok tekintetében. Ha belegondolunk, hogy közel 30 játék már rendelkezik natív Linux átirattal, és körülbelül 100 garantáltam működik a Wine(X) segítségével, az a szókapcsolat, hogy „Linuxos játékgép” már korántsem nevetséges, sőt kifejezetten valóságszagú. De mint mindig, most is a részletekben lakozik a lényeg. A sebesség nem csak a processzortól és a memóriától függ, hanem oda kell figyelni minden egyes alkatrészre, ahogyan azt már a másik rendszer alatt megszokhattuk. A programok beállítása sem jelent túl nagy problémát, főleg ha belegondolunk, hogy a jutalmunk a „linux factor” lesz, amely alapból 5-10% sebesség növekedést jelent, de könnyedén felugrik 15%-ra is, ha jól finomhangoltuk a gépet. Immáron semmi akadálya nincsen annak, hogy jó, minőségi játékokkal pihenjünk, ha végetért a munka.

Nincs más dolgunk hát, mint játszani és játszani és várni az új Doomra, amely (előzetes hírek szerint) szintén elérhető lesz Linuxon is.

Dancsok „strogg” Zoltán (strogg@mail.tvnet.hu) Jelenleg technikai szerkesztőként dolgozik a BME-OMIKK-nál, ahol oktat is. Emellett egyetemi képzésben vesz részt, programozó matematikus szakon. Négy éve foglalkozik Linuxszal. Szabadidejében operációs rendszereket gyűjt és weblapot vezet.

A Perl/Tk

Kölcsönözz felhasználóbarát külsőt a már megírt, hatékonyan működő Perl parancsállományaidnak!

A Linuxvilág hűséges olvasói biztosan emlékeznek még arra, amikor saját IRC-botot írtunk, vagy amikor elkészítettük önálló web pókunkat, hogy azzal gyűjtsük be az információkat a világhálóról. Ezeket a feladatokat azért Perlben oldottuk meg, mert a hálózatkezelés, és a vele járó szövegfeldolgozási feladatok ezen a nyelven tűnhetnek a legegyszerűbbnek. A Perl különlegesen rugalmas és ennek köszönhetően számos feladatra kitűnően alkalmazható. Biztosan feltűnt, hogy eddigi cikkeimben ugyan szó volt szövegfeldolgozásról, könyvtárkezelésről, TCP/IP programozásról, de sosem készült valódi felhasználói felület programjainkhoz. Csak alapvető be- és kimeneti módszereket használtunk, melyekkel parancssorhoz szokott szemmel egész jól el lehetett igazodni az alkalmazások használatában. Ám mindig gondolni kell arra a felhasználóra is, aki először ül a monitor előtt, és általában a grafikus felhasználói felületekért (GUI) rajong. Képes erre a Perl? A válasz természetesen: igen. A Tk-ról van szó, melyet eredetileg a Tcl nevű parancsnyelvhez készítettek el. Később azonban egy Perl modul formájában már ebből a nyelvből is elérhetővé vált ez a remek eszköztár. Mindenekelőtt szerezd be ezt a modult a CPAN oldaláról (➲ http://www.cpan.org). Több mint valószínű, hogy terjesztésed is tartalmazza csomag formájában, ekkor elég ezt telepítened. Debian alatt a `perl-tk` csomagra van szükséged, amely tartalmaz egy elég jó leírást is. A `Tk.pm` egy objektumközpontú felülettel áll a rendelkezésünkre. Látni fogjuk, hogy a grafikus programozás mindössze különféle osztályokból történő példányosításokból fog állni. Ha még nem barátkoztál meg az objektumközpontú programozással, feltétlenül olvasd el a ***perlboot (1)*** súgóoldalt, e nélkül ugyanis nem fogod tudni alkalmazni az itt bemutatásra kerülő módszereket. Vágjunk bele! Első lépésként hozzunk létre egy ablakot. Íme a kód:

```
#!/usr/bin/perl -w

use strict;
use Tk;

my $main_win = MainWindow -> new ();
$main_win -> title ("Hello világ!");

MainLoop ();
```

Bemelegítésként nézzük át sorról sorra, mit is csinál ez a program. Az első sor mindössze arra szolgál, hogy a parancshéjnak meghatározzuk a parancsállományunkat feldolgozó parancsértelmező elérési útját. Emellett megadunk neki egy további kapcsolót is, mellyel arra utasítjuk az értelmezőt, hogy a futtatás során a figyelmeztetéseket is jelenítse meg, ne kizárólag a hibaüzeneteket. A következő sor a szigorú feldolgozást kapcsolja be. Ha nem tudod, mi ez, és jó Perl programot akarsz írni, jobb, ha mindig használod. A harmadik sorban pedig a fentebb már emlegetett Tk modult vesszük használatba. Ezután létrehozunk egy új skalár változót, `$main_win` néven. A `my`, vagy `local` kulcsszóval történő változóbevezetés szigorú módban kötelező. A két kulcsszó a változó láthatóságában tesz különbséget, ám erre itt nem térnék ki. Új változónknak azonnal értéket is adunk. Jelen esetben a `MainWindow` osztály `new` nevű konstruktorát hívjuk meg, mely egy objektummal tér vissza. Ezt csak azért hangsúlyoztam, mert Perlben nincs megkötés egy osztály konstruktorára vonatkozólag, ezért hívhatnák másképp is. Annak, hogy a neve `new`, megvan viszont az az előnye, hogy egy már más programozási nyelvekből is ismerős formában is példányosíthatunk:

```
my $main_win = new MainWindow;
```

Visszatérve a példára, a következő sorban újonnan létrehozott objektumunk egy tagfüggvényét hívjuk meg. A `title ()` egy sztringet vár, és segítségével az ablak címét módosíthatjuk. Az utolsó sorban történő függvényhívás indítja el az eseményciklust. Ez lényegében egy végtelen ciklus, amely a különböző vezérlők (***widget***) eseményeinek lekezeléséért felelős. Ne ijedj meg, ha elsőre nem érted, a példák világossá fogják tenni ennek a rejtélyes függvényhívásnak a mibenlétét. A lényeg, hogy az ablak e nélkül a sor nélkül nem jelent volna meg. Továbbá az eseményciklus végtelen tulajdonsága miatt az ablak mindaddig él (és fut a program), amíg az ablakkezelőn keresztül be nem zárjuk.
Miután létrehoztuk első ablakunkat, semmi sem állíthat meg bennünket, hogy vezérlőket pakoljuk rá. Íme:

```
#!/usr/bin/perl -w

use strict;
use Tk;
```


```
my $main = new MainWindow;
$main -> title ("Hello");

my $label = $main -> Label (
     -text => "Helló világ!",
     -width => 25,
     -height => 3
);

$label -> pack;

MainLoop;
```

A Tk eszközkészletben számos vezérlő áll rendelkezésünkre. Ilyenek a gomb, jelölőnégyzet, rádiógomb, menü, vászon, és még sok más. A címke (*label*) vezérlő egy közönséges, a felhasználó által csak olvasható szövegmező. Mint minden vezérlőnek, ennek is vannak tulajdonságai. Ezek a tulajdonságok már a vezérlő létrehozásakor kaphatnak kezdeti értéket, de vannak alapértelmezett beállítások is. Általában igaz, hogy a vezérlőről a `Tk::vezérlő_neve (3pm)` súgóoldalon kaphatunk egy teljes leírást.

A hatodik sorban hozzuk létre az új címkét. Látható, hogy ehhez mindössze a főablak objektumának a megfelelő tagfüggvényét kell meghívnunk, és ez visszaadja az új címkeobjektumot. A Label tagfüggvény egy asszociatív tömböt vár paraméterként. Ez egy kulcs-érték párokból álló lista, másként fogalmazva egy olyan tömb, ami karakterláncokkal van indexelve. Mi itt egy névtelen tömböt adunk át a függvénynek. A kulcsok rendre egy kötőjellel kezdődnek, jelezve, hogy beállításokról van szó, ám ezek a kötőjelek bizonyos Tk változatokban elhagyhatók. Nagyon fontos, hogy a kulcsok és értékek között => áll és nem ->!

A következő sor nagyon fontos. E nélkül a címke meg sem jelenik az ablakban, hiába hoztuk létre az objektumot. A vezérlőobjektum ugyanis a létrehozás pillanatában független az ablaktól, annak ellenére, hogy a főablak tagfüggvényével hívtuk életre. Szükség van egy geometriakezelőre, amely felpakolja a vezérlőt az ablakra. Számos geometriakezelő létezik, mi most a legegyszerűbbet használjuk, a `pack ()` függvényt. Ezt nem szabad összetéveszteni a szabványos Perl könyvtárban található `pack ()` függvénnyel. Ez a vezérlőobjektum tagfüggvénye. Segítségével nagyon egyszerűen, a már meglévő elemekhez képest a négy irányban bárhova rakhatunk egy új vezérlőt. Paraméterek nélküli híváskor folyamatosan egymás alá pakol.

Nem élet az élet gomb nélkül. Íme:

```
#!/usr/bin/perl -w

use strict;
use Tk;

my $main = new MainWindow (
     -borderwidth => 15
);
$main -> title ("Gombóc");

$main -> Label (
     -text => "Helló világ!",
     -foreground => "blue",
     -width => 30,
     -height => 5
     ) -> pack;

$main -> Button (
     -text => "Kilép",
     -width => 10,
     -height => 2,
     -command => \&exit
     ) -> pack;

MainLoop;
```

Számos újdonságot fedezhetünk fel a már megismert elemek használatában is. Mint láthatod, a főablak objektum létrehozásakor is át lehet adni bizonyos paramétereket a konstruktornak. Itt az ablak belső szegélyének vastagságát adtuk meg képpontokban. Ennek a beállításnak az alapértelmezés szerinti értéke nulla. A félreértések elkerülése

végett, ez nem az ablakkezelő által szolgáltatott ablakkeret! Ez az a szegély, ami körbeveszi az összes általunk felpakolt vezérlőt az ablakban. A címkeobjektumot létrehozzuk, ám egy változónak sem adjuk értékül. Ez akkor nem gond, ha a programunkban később sehol sem akarunk hivatkozni erre a vezérlőre. Ha például később módosítani szeretnénk a szöveget, akkor ezzel a könnyelműséggel nem élhetünk. Ebben az esetben viszont elég arról gondoskodni, hogy az objektum geometriakezelőjét meghívjuk, ezt pedig a fent látható módszerrel megtehetjük. Kurta megoldásunknak nem látjuk kárát, mert a -> operátor balról jobbra értékelődik ki. Megadtuk továbbá a címke `-foreground` beállításában a szöveg színét. A gomb létrehozása ezek után már gyerekjáték. Itt is egy névtelen gombot hozunk létre, és meghívjuk rá a `pack ()`-et. Értelemszerűen a `-text` a gombon olvasható szöveget jelenti. Érdekes újdonság a `-command`. Ez egy olyan beállítás, amelynek egy függvény címét kell adni. Egy változó címét a \ operátorral képezzük, a & pedig a függvénytípust jelöli. A gombra történő kattintás hatására meghívásra kerül a megadott függvény.
A `MainLoop` tehát az ablak kirajzolása után folyamatosan vár a gomb megnyomására, és az eseményt kiváltó kattintást követően meghívja a `-command` függvényét.
Jöjjön végre egy igazi példa.

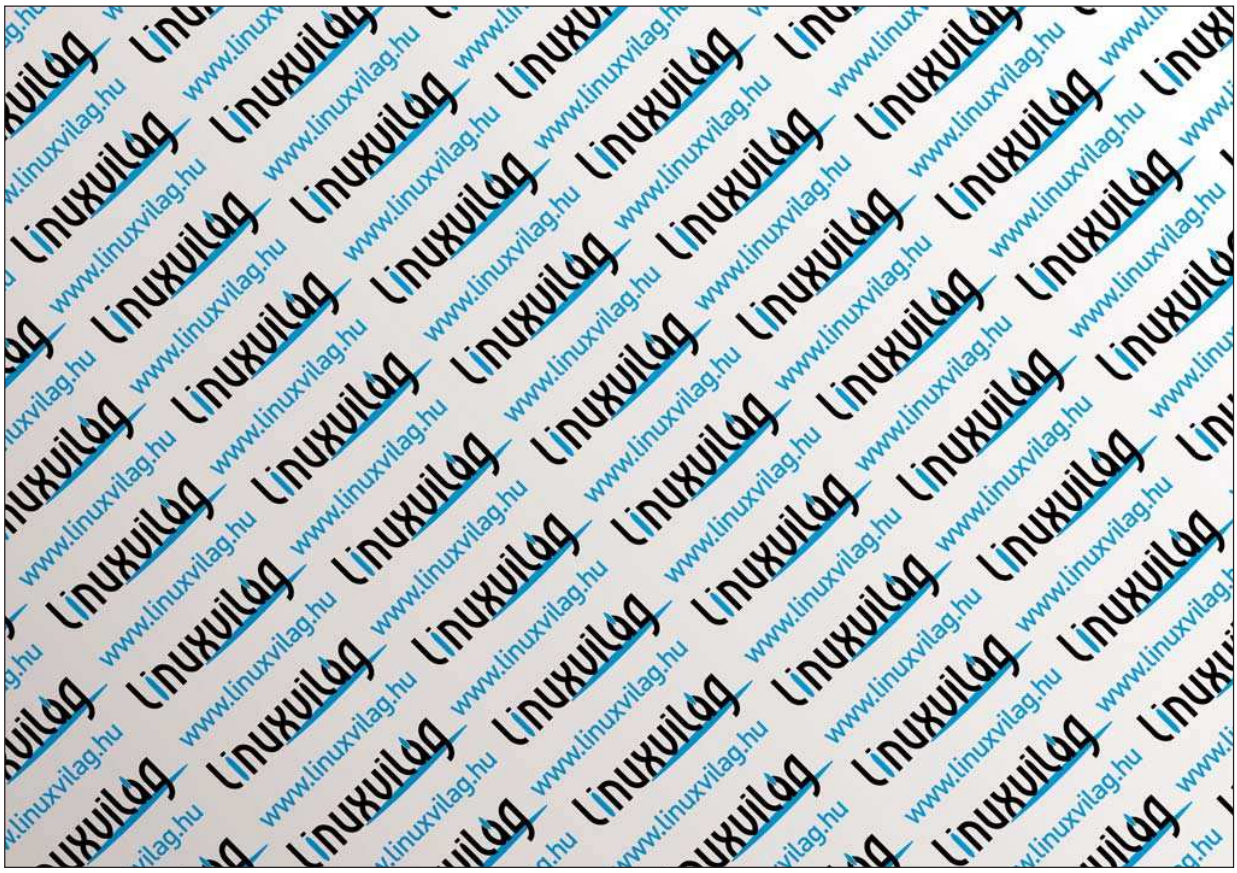
```
#!/usr/bin/perl -w

use strict;
use Tk;

my $main = new MainWindow (
     -title => "Hello",
     -borderwidth => 15
);

my $frame1 = $main -> Frame ( -borderwidth =>
➥15 );

$frame1 -> Label (
     -text => "Nevem",
     -width => 10,
     -height => 1,
     ) -> pack ( -side => "left" );

my $name;
$frame1 -> Entry (
     -textvariable => \$name,
     -width => 20,
     -validate => "key",
     -validatecommand => \&editentry
     ) -> pack ( -side => "right" );

$frame1 -> pack;

my $frame2 = $main -> Frame ( -borderwidth =>
➥15 );

$frame2 -> Button (
     -text => "Szia!",
     -width => 10,
     -height => 2,
     -command => \&printhello
     ) -> pack ( -side => "left" );

my $label;
$frame2 -> Label (
     -textvariable => \$label,
     -foreground => "blue",
     -width => 30,
     -height => 1
     ) -> pack ( -side => "right" );

$frame2 -> pack;

my $quit = $main -> Button (
     -text => "Kilép",
     -width => 10,
     -height => 1
     ) -> pack;
$quit -> bind ("<Button-1>", sub { exit; });

MainLoop;

sub printhello {
     my $hello = "Szia";
     if (defined $name and "" ne $name) {
        $hello .= ", " . $name;
     }
     $hello .= "!";
     $label = $hello;
}

sub editentry {
     $label = "";
     return 1;
}
```

Az első változtatás az, hogy milyen módon adtuk meg a főablak címét. Eddig az objektum `title ()` tagfüggvényét használtuk, most áttértünk a kezdetiérték-adásra. A második a keretek használata. A keret feladata egységbe foglalni a vezérlőket. Ez előnyös lehet nekünk, a programozónak, hiszen logikai egységekre bonthatjuk a vezérlőelemeket. Előnyös lehet a felhasználónak, mert ha láthatóvá tesszük a keret szélét, azzal áttekinthetőbbé tesszük a felületet. Továbbá előnyös lehet a vezérlők elhelyezésekor. Most ez utóbbi miatt alkalmazzuk a kereteket, mivel a `pack ()` nem túl okos. Első megközelítésben nyugodtan gondolhatsz úgy a keretekre, mint kisebb ablakokra a főablakon belül, hiszen nem mások, mint vezérlőtárolók. Ez a program egy nevet kér be, majd kiírja. Úgy építjük fel az ablakunkat, hogy lesz két keretünk egymás alatt, és egy kilépés gomb legalul. A felső keretben egymás mellett van egy címke, és egy szövegbeviteli mező. A címke természetesen utal arra, hogy mit kell beírni a mezőbe. Az alsó keretben pedig van egy gomb, és egy másik címke. A gomb megnyomására a címkében megjelenik egy üdvözlőszöveg, azzal a névvel, amely a felső keretbeli beviteli mezőben szerepel. Mivel a keret olyasmi, mint egy kisebb ablak, úgy is kell használni, mint egy ablakot. Létrehozásakor megadhatjuk a keret vastagságát a `-borderwidth`

beállítás segítségével csak úgy, mint a főablak esetében.
A kereten belül elhelyezésre kerülő vezérlőket természetesen a keretobjektum tagfüggvényeivel kell létrehozni. Ezért dolgoztunk a fenti példában `$frame1`, illetve `$frame2` objektumok függvényeivel. Továbbá nem elég az egyes vezérlőkre meghívni a `pack ()` függvényt, a vezérlő kipakolása után az egész keretre is ugyanúgy meg kell hívni. A következő meglepetés egy szövegbeviteli mező formájában érhet. A beviteli vezérlő (`entry`) egysoros, felhasználó által szerkeszthető szöveges mező. Hasonló beállításai vannak, mint a címkének, ám mivel egysoros, nincs `-height` beállítása.
A `-textvariable` által megadható egy változó címe, mely mindig a mező pillanatnyi értékét fogja tartalmazni, felülírása pedig a mező szerkesztésével egyenértékű. Ám a leírás szerint bizonyos esetekben nem javallott írni ezt a változót.
A `-validate` segítségével megadható egy, a mező szerkesztésével kapcsolatos esemény, amely bekövetkezésekor a `-validatecommand` által tárolt függvény meghívásra kerül. Ha ez utóbbi függvény igazat ad vissza, a szerkesztés végbemegy, ha hamisat, akkor nem. Ezzel szerkesztés közben koppinthatunk a felhasználó orrára, ha például egy csak számokat váró mezőbe betűket pötyögne. Itt csak arra használjuk ezt a lehetőséget, hogy töröljük az üdvözlő üzenetet, ha a felhasználó szerkeszti a nevet.
Azt, hogy az alsó keretben szereplő címkénkben mindig a helyes üdvözlő üzenet szerepeljen, egy gombhoz rendelt függvénnyel oldjuk meg. A függvényünk neve `printhello ()`. Továbbá használunk egy segédváltozót, melyet a `-textvariable` segítségével hozzárendelünk a címkéhez.

A címke először nem tartalmaz semmit, hiszen az objektum létrehozásakor nem adtunk meg alapértéket. Először akkor kaphat értéket, amikor a felhasználó a Szia! feliratú gombra kattint. Ekkor meghívásra kerül a `printhello`, amely, miután ellenőrizte, hogy a felső beviteli mezőben szerepel-e adat, a segédváltozónak értékül adja a megfelelő karakterláncot. A kilépést egy kicsit furcsán oldottam meg. Nem használtam a gomb `-command` beállítását, ehelyett meghívtam a `bind ()` tagfüggvényt. Utóbbi szintén eseményekhez rendel függvényeket, ám sokkal általánosabb. Különböző egérgombokhoz, egyszeres- és kétszeres kattintáshoz, billentyűkombinációkhoz más- és más függvényeket rendelhetünk. Itt az eredmény ugyanaz, mintha a `-command`-ot használtam volna, ám szerettem volna bemutatni, hogyan működik a `bind ()`. Első paraméterként meg kell adni az eseményt. Ez <> jelek közötti egyszerű események sorozata is lehet (gondolj az Emacs-ra). A Button-1 a bal egérgomb (ezt jelenti az 1-es) egyszeri megnyomását jelenti. Második paramétere pedig az a függvény, amelyet az esemény kiváltásakor meg kell hívni.
Rengeteg dolgot hallgattam el, amiért bocsánatot kérek, de sajnos a helyszűke nagy úr. Ajánlom figyelmedbe ***Steve Lidie, Nancy Walsh: Mastering Perl/Tk*** című könyvét, ami egy nagyon jól használható leírás. Az említett súgóoldalak pedig töménytelen információval szolgálnak az egyes objektum tagfüggvényeinek paraméterezéséről.

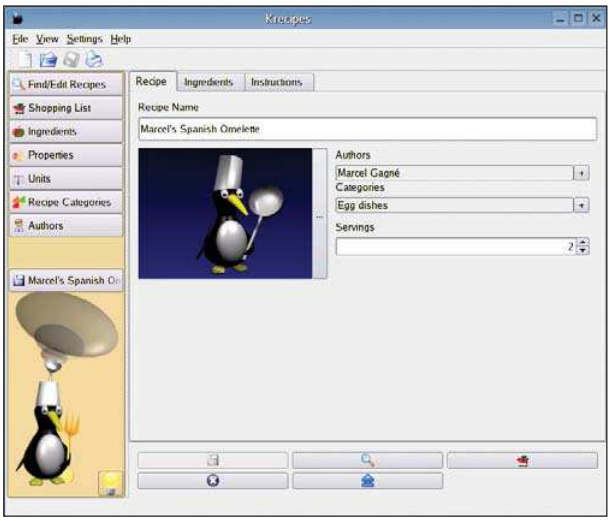
Fülöp Balázs
admin@guardware.com



Tökéletes főzőfülke

Rendszerezzük receptjeinket MySQL adatbázis-kiszolgálóval vagy anélkül. Ízlés szerint.

Semmi különös nincs ebben, ***mon ami***. Egész évben a tökéletes Linux gépről beszélünk, François. Egyfolytában túl akarjuk szárnyalni magunkat grafika, sebesség, memória, lemezterület és egyebek terén.
A tökéletes persze változatlanul egyúttal a legfrissebb és legnagyszerűbb. Az a gondom, François, hogy akármennyire is újak ezek az gépcsodák, voltaképp mind-mind ugyanaz. Aztán felötlött bennem, hogy nem segíthetne-e a Linux a konyhában, ami végre valami igazán új lenne.
Nem, François, nem kell aggódnod, az állásod nem forog veszélyben. Sőt ez pontosan olyasmi, ami a munkádat egyszerűbbé teheti. Tulajdonképpen, nem lennék meglepve, ha vendégeink közül sokan úgy éreznék, ha főzés kerül szóba Linuxot használni teljességgel természetes dolog. Most, hogy a vendégekről beszélek, már látom meg is érkeztek. Üdvözlet, ***mes amis***, Chez Marcelnél. ***Vite***, François! Szaladj a pincébe és hozd fel a 1997-es Brunello di Montalcino-t, ***immédiatement***!
Míg hűséges pincérem felhozza a bort, mesélek egy kicsit a mai menüről. A tökéletes géppel foglalkozó értekezéseink fénypontjaként felfedeztem néhány programot, melyek gépünket képessé teszik a főzésre. Nem lefőzésre célzok a teljesítmény terén, ***mes amis***, hanem főzésre, étellel és borral. Miközben keresgéltem a módját, miképpen tudna segíteni a Linux a konyhában, ráakadtam a Krecipes programra (lásd a hálózati forrásokat!). ***Unai Garro, Jason Kivlighn*** és ***Bosselut Cyril*** nagyon szép nyílt forrású csomagot rakott össze amely gyerekjátékká egyszerűsíti saját receptjeink elkészítését és karbantartását. A Krecipes segítségével elkészíthetjük saját receptjeinket illetve beolvashatjuk őket a legnépszerűbb csereformátumukból (RecipeML, MasterCook, Meal-Master és egyebek). Saját receptjeinket KreML (Krecipes XML formátum) formátumban menti el. Karbantarthatunk kategóriákat, nyomon követhetjük a kalóriákat (rost, zsír és egyéb összetevőket), vásárlási listát készíthetünk a kiválasztott étkek alapján.
Ah, François, hát visszatértél. Kérlek, tölts a vendégeinknek.
A Krecipes adatbázisban tárolja a recepteket, ezért a program lefordítása és használata előtt fel kell telepítenünk a MySQL vagy az SQLite (lásd a forrásokat) programot. Az SQLite apró, programba ágyazható adatbázismotor amihez kevesebb kezelési és karbantartási többletmunka kell mint a MySQL-hez, viszont nem olyan teljesértékű és hatékony. Az ilyesfajta alkalmazásokhoz viszont sokan vonzó lehetőségnek találhatják. Az a szép az SQLite-ban, hogy nincs szükség adatbázis-kiszolgáló futtatására a rendszeren, mégis kihasználhatjuk az SQL adatbázis-tárolási és elérési képességeit.



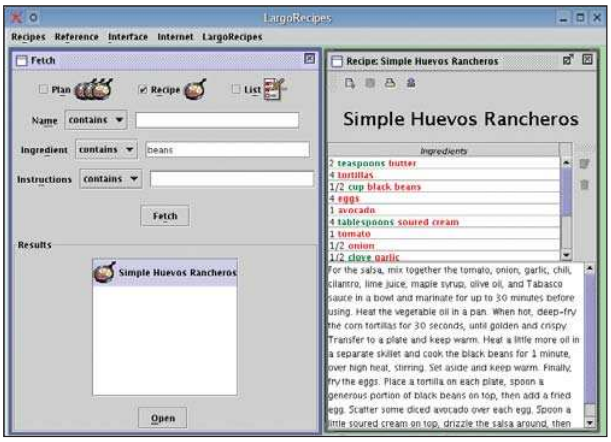
1. ábra Marcel híres Spanyol Omlettjének leírását gépeli be a Krecipes segítségével

Amennyiben MySQL és SQLite is van gépünkön, fordításkor mindkét rendszer támogatása bekerül a programba.
A fordításról csak annyit, hogy klasszikus példája a szokásos öt lépéses fordítási folyamatnak:

```
tar -xzvf krecipes_alpha_0.4.1.tar.gz
cd krecipes
./configure --prefix=/usr
make
su -c "make install"
```

Először a Krecipes varázslóval találkozunk (a program neve egyébként `krecipes`), amely végigvezet bennünket néhány alap beállítási lehetőségen, többek közt, hogy melyik adatbázisban szeretnénk tárolni az adatokat. Amennyiben MySQL és SQLite támogatást is fordítottunk a programba, bármelyiket választhatjuk. Minthogy éttermünkben már

rengeteg MySQL hátterű programról szó esett már, úgy gondoltam nem rossz ötlet kipróbálni a SQLite-ot. Választottam majd a *Next*-re kattintottam.
Miután választottunk, a Krecipes felajánlja a lehetőséget, hogy az adatbázist feltöltse néhány példareceptt el. Ne felejtsük el bejelölni ezt a négyzetet a beállításoknál, így lesz néhány mintánk ami segít megismerkedni a program képességeivel.
Új recept létrehozásához kattintsunk a ***File*** pontra menüben és válasszuk a ***New***-t avagy egyszerűen kattintsunk a ***new recipe*** gombra az ikonsor bal felső sarkában. A recept űrlapjának három füle van. Az első a recept alapadatait írja le – a nevet, a szerzőt, melyik kategóriába soroljuk az étket (új kategóriákat is készíthetünk), illetve hány emberre szól a recept. Erre az *1. ábrán* láthatunk példát.
A másik két fül az összetevők és a leírás számára van



2. ábra Egy kis Huevos Rancheros tálalása a LargoRecipes segítségével

fenntartva. Minden esetben bármikor elmenthetjük a munkánkat vagy később visszatérhetünk ha frissíteni szeretnénk valamit.
Amennyiben inkább az Interneten Meal-Master és RecipeML formátumban megtalálható több ezer receptből szeretnénk válogatni, nagyon könnyen importálhatjuk őket. Ez a sok beszéd az étkekről, ***mes amis***, csak azt bizonyítja mennyire fontos is egy jól feltöltött borospince.
A másik nagyszerű receptkezelő amit érdemes megvizsgálni, ***Douglas Squirrel*** LargoRecipes programja.
A LargoRecipes segítségével (amely egyébként a szerző kutyája után kapta nevét) receptjeinket kezelhetjük, megoszthatjuk őket barátainkkal (weblapokon keresztül), készíthetünk vásárlási listákat, készíthetünk étkezési tervet és így tovább. Itt is importálhatunk Meal-Master és RecipeML formátumú recepteket. A *2. ábrán* működés közben is megtekinthetjük a LargoRecipes-t.
A LargoRecipes telepítéséhez legalább 1.4-es Javára lesz szükségünk, amiből kikövetkeztethetjük, hogy fordításra viszont ez esetben nem lesz szükségünk. Két fájlt kell letöltenünk és elmentenünk a LargoRecipes weblapról (lásd a forrásokat). Az első a largorecipes terjesztés, hamarosan rátérek a második állományra is. Az írás születésekor a 0.9.2.1-es változat volt a legfrissebb. A csomag telepítéséhez mentsük el a csomagot tetszőleges helyre – én egy Largo könyvtárat készítettem a **felhasználói könyvtáramban** – majd hajtsuk végre a következő parancsokat:

```
cd ~/Largo
java -jar largorecipes-0.9.2.1.jar
```

További használatkor is ezeket a parancsokat kell majd kiadnunk. Első futáskor a telepítési ablak jelenik meg. Az összes szükséges adatfájl és könyvtár ott jön létre ahol elindítottuk a telepítést. Az egyik ilyen könyvtár neve ***demo***. Ide kell elmentenünk a letöltött második állományt, a LargoRecipes demo fájlt. Ezt is letölthetjük a LargoRecipes weboldal letöltési lapjáról.
Amennyiben a futás során szeretnénk kipróbálni a példareceptteket, kattintsunk a LargoRecipes pontra a menüsoron majd válasszuk a Demonstration pontot. Ha ezt inkább átugornánk, és egyből el akarjuk kezdeni bevinni a recepteket, kikapcsolhatjuk a LargoRecipes ***RecipeML archive*** négyzetét.
A honlapon 10,000 receptet találunk összezippelve; a hivatkozást a főlapon találjuk.
Aki szeretné másokkal megosztani receptjeit, a LargoRecipes programban weboldal-exportáló pontot is talál. Kattintsunk a menüsor Internet gombjára és válasszuk a Web Page pontot. Az elérhető receptek listája máris megjelenik a a jobb oldali ablakban. Válasszuk ki szimpatikusakat majd az Add segítségével vegyük fel az export listára. Ha mindent kiválasztottunk, adjunk címet a lapunknak, de még ne kattintsunk a Go-ra! Látnunk kell itt egy Include XML Download feliratú jelölőnégyzetet. Ezt ne felejtsük el bekapcsolni, az összes receptlapunkon, így a látogatók RecipeML formátumban letölthetik majd a receptjeinket és kedvenc receptkezelő rendszerükbe importálhatják.
Aki igazán kíváncsi, a források részben megtalálja a RecipeML formátum leírás hivatkozását. Soha sem árt ha az ember tudja, hogyan működnek a dolgok, non? Beírtam egy hivatkozást a Meal-Master weboldalra is. Rengeteg hivatkozást találunk itt, ezeket követve pedig rengeteg receptet találhatunk melyek mind készek rá, hogy a kedvenc receptgyűjtőnkbe importáljuk őket.
Mon Dieu, mes amis, elérkezett a záróra és én csak tovább éheztettem önöket. François talán lesz olyan kedves és újratölti a poharainkat még egyszer. Addig én kihozom a híres dupla vajas briósomat fűszeres, vegyes- bogyós lekvárral. Most hogy ennyi kísértő finomsággal tömtük meg linuxos rendszerünket, az előételek bizony elsőbbséget élveznek. Következő találkozásunkig, mes amis, igyunk egymás egészségére!

A vôtre santé!
Bon appétit!

Linux Journal 2004. augusztus, 124. szám

Marcel Gagné (mggagne@salmar.com)
Mississaguában, Ontario államban él.
Ő a szerzője a Kiskapu kiadásában megjelent Válts Linuxra! – búcsú a kékhaláltól (ISBN 963 9301 76 0) című könyvnek.

Egy igazi örökzöld: Fallout

Talán mindenki emlékszik a nagy klasszikusra, a Falloutra. Anno, mikor megjelent, stílust teremtett, és grafikája – ha nem is volt a kor csúcsa –, nem hagyott kívánnivalót maga után. Játékmenete, stílusa pedig garantálta, hogy egykönnyen ne lehessen végigjátszani, így semmiképpen sem eshetett korunk (sajnos) általános „eldobod úgyis, csak vedd meg tölünk” kategóriájába.

A játék a régi motorosok számára jól ismert, talán csak az újabb játékosgenerációnak jelent újdonságot. Maga a program alapműnek számít minden játékos, és azon belül is a szerepjátékosok számára. Mondhatnám, kötelező darab.
Nem csoda hát, ha felcsillant a szemem, amikor megláttam, hogy életre kelthető Linux alatt is. Így hát semmi sem tarthatott vissza attól, hogy letöltsem a telepítőt, és nosztalgiázzam kicsit.
Azokról az időkről, amikor még a játékot a játék adta el, és nem a marketing, amikor még nem voltak másolásvédelmek, és nem fenyegettek meg nyíltan tizenéves gyerekeket a gyártók.
A hőskor kiemelkedő darabjával van hát dolgunk.

A játékmenet

A játékmenet és a történet jól ismert, de fussuk át kicsit, hátha valaki nem ismeri még a programot. A történet az emberi civilizáció pusztulása után játszódik, ahol kis elszigetelt csoportok élnek elzárva a föld alatti erődítményekben. Ebben a világban élet-halál harc folyik mindenért, ami elősegítheti az életben maradást, vagy a közösségek fenntartását. Ilyen a lőszer, a fegyver, a gyógyszerek és az üzemanyag.
Hősünk kénytelen elindulni a hosszú útra, ki, a sugárfertőzött, pokoli világba, ahol banditák, mutánsok, és mindenféle szörnyeteg les rá. Természetesen mindezt gyengén felfegyverezve, rendkívül alacsony szinten teszi. Az egyik cél a küldetés teljesítése, a másik viszont határozottan az életben maradás.
Ez ugyanis szintén nem egyszerű feladat, igaz, nem is bosszantóan nehéz a játék.
A játékmenet viszont korántsem egyhangú, és nem is válik azzá, ellentétben az Unreal-lel. A harcrendszer ugyan kicsit egyszerűre, de érdekesre sikeredett.
A játékban megtalálható szerepjátékelemek azonban manapság már korántsem korszerűek. Amikor ugyanis a játék készült, még nem létezett a D&D3 rendszere, többek között ez a program is nagymértékben hozzájárult ahhoz, hogy manapság ilyesmik létezhessenek. Inkább a hangulat az, ami vissza-visszavonzza a játékost a jövő eme senkiföldjére, hogy újra, és újra küzdhessen a túlélésért.
Ez a hangulat viszont tökéletes.
A készítők rendkívül jól eltalálták a határt, hogy mennyire legyen az atmoszféra komor és félelmetes, de ne csüggesztő.
Főleg, hogy az alaphangot egy kiváló intro adja meg, amelyet, úgy gondolom, nem ártana egy-két politikusnak is megnéznie, hátha elgondolkodna az emberiség jövőjét illetően.
A feketehumor is lépten-nyomon visszaköszön, így garantált, hogy sosem unatkozunk.

Hangok, grafika, gépigény

A program kiadásakor még nem volt nagyon elterjedt a 3D megjelenítés, főleg ebben a műfajban. A 3D először az FPS-ben kezdett hódítani és lett a Quake után ***de facto*** szabványmegjelenítés. Sok időnek kellett azonban eltelnie ahhoz, hogy ez elterjedjen, és megvesse a lábát a stratégiai és az RPG játékokban. Ezért a Fallout nem is használ 3D-t, így egy régebbi, 3D kártya nélküli gépen is remekül futtatható.

Korának és az akkori technológiának köszönhetően a gépigény sem magas. Az én Duron 800 Mhz-es gépemen a WineX ellenére egy pillanatnyit sem zökkent, tökéletesen futott. A processzor az, ami fontos. Mindez talán köszönhető annak is, hogy program szempontjából is alapos munkát végeztek a készítők, így minimális az ilyen hibák száma.

Mindezt azért említem meg, mert érthetetlen számomra, hogy komoly világcégek hogy nem képesek úgy megcsinálni egy programot, hogy helyenként ne akadjon, vagy az intro ne szaggasson be.

Az ok biztosan nem a gépigényben keresendő, hisz olykor több éves játékoknál látható, egy mai, korszerű vason.

A grafika tehát nem 3D, ennek ellenére tetszetős. Teljesen átlátható a játéktér, nincsenek zavaró elemek, és egy óra játék után a kezelhetőség is teljesen a helyére billen (igaz, hogy addig viszont minden baja van a játékosnak).

A hangulatot fokozó hangok viszont annál érdekesebbek. Teljesen „lefedik" a játékot. A zene kellemes, és olykor határozottan adrenalinemelő, a hangok mai füllel már kicsit porosnak hatnak, de ettől függetlenül teljesen rendben vannak.

Kiemelendő itt, hogy ez a játék kerüli az élesebb hangokat, így sosem ijeszti meg a játékost, és hosszú játék után sem lesz kellemetlen hallgatni. (Bizony előfordul, hogy egy-két játékban olyan hangot adnak a snipernek, ami mindössze egy elviselhetetlen éles csattanásból áll. Tíz lövés után megpróbáltam meglenni nélküle.)

A telepítés

Ez a játék is, mint (sajnos) annyi másik, csak emulátorral hajlandó futni. Így a Fallout is csak WineX-el indul, ennek megléte tehát kötelező.

A WineX elérhető a ➲ http://www.transgaming.com oldalon.

Ha ez megvan, akkor semmi akadálya, hogy telepítsük a programot. Ehhez kell a telepítő, (➲ http://liflg.sourceforge.net/), és a gyári CD lemez. Figyelem, ez a telepítő is egy összedrótozott WineX-es bináris, amelyet a loki által fejlesztett ***game installer*** motor rak fel a gépre. Így az `automount` probléma továbbra is fennállhat (azért kell kikapcsolni, mert egyébként nem látja a CD-t). Természetesen rendszergazdaként hajtsuk végre a telepítést. Ha ez is megvan, akkor gyakorlatilag semmi akadálya sincsen a játéknak. Mint a DeusEx esetében tapasztalhattuk, a fejlesztők kiváló munkát végeztek. Egy telepített teljes értékű WineX megléte esetén, gyakorlatilag azonnal futásra kész a telepített játék, mindenféle beállítás nélkül. Aki a WineX-szel már próbálkozott, az tudja, hogy ez nagy szó.)

Ezért a telepítést merem kezdőknek is ajánlani, hisz a játék problémamentesen felkerül a lemezre, és fut.

Most már mindnyájan élvezhetjük a Fallout komor, kemény, de csodálatos világát, és bekalandozhatjuk Linuxon is ezt a világot.

További kellemes játékot kívánok ezzel az örök klasszikussal. A következő hónapban egy igazi csemegét veszünk szemügyre, a Jedi knightII: Jedi Outcast-ot.

Dancsok „strogg" Zoltán
(strogg@mail.tvnet.hu)
Jelenleg technikai szerkesztőként dolgozik a BME-OMIKK-nál, ahol oktat is. Emellett egyetemi képzésben vesz részt, programozó matematikus szakon. Négy éve foglalkozik Linuxszal. Szabadidejében operációs rendszereket gyűjt és weblapot vezet.

Szoftverjog – barát vagy ellenség?

Bevezető gondolatok egy szoftverjogi sorozathoz.

Alapvetően három magatartásforma létezik, mellyel egy jogász jelenlétét egy informatikusokból álló társaság képes lereagálni:

- utálják
- tartanak tőle
- kíváncsiak rá

Az első kategóriába tartozókat arra kérem, hogy azonnal lapozzanak tovább és pár nap múlva érdeklődjenek egy olyan barátjuknál, aki a két másik csoport egyikébe tartozik, hogy miről is szólt ez a cikk. Azokat, akik fenntartásokkal közelítenek minden doktorhoz – legyen az jogász vagy fogorvos – szeretném megnyugtatni, hogy a cikksorozat megírása során tisztán jószándék vezet. Legfőbb célom hogy a számítástechnika iránt érdeklődőket (kezdőket és profikat egyaránt), kicsit közelebb vigyem a jog világához, annak egy érdekes és mindenki számára érthető oldalát bemutatva. Mellesleg a fogorvosokkal én sem vagyok teljes mértékben kibékülve.
A kíváncsiakat arra bátorítom, hogy küldjenek e-mailt, kérdezzenek, írják meg, hogy miről szeretnének olvasni. Ígérem, hogy a sorozat összeállítása során figyelembe fogom venni ezeket a javaslatokat.

Tervek

A szoftver keletkezésének nemcsak gyakorlati, hanem jogi szempontból is különböző állomásai vannak. Ezért első körben a szoftver fogalmával kell tisztába jönnünk. Ezt követően szó esik majd az alkotásról magáról, arról hogy kik és milyen feltételekkel minősülhetnek szerzőnek, illetve hogy a szerzőket milyen védelem illeti meg.
Kiderül majd, hogy munkaviszonyban illetve azon kívül alkotott programok esetében milyen jogvédő eszközök állnak rendelkezésünkre. Kitartó olvasóink megtudhatják, hogy mi az a felhasználási szerződés és miért hívja a jogászokon kívül mindenki más ezt „licencnek”. Ezek után teszünk egy rövid körutazást, melyben megismerkedhetünk ezen szerződések alaptípusainak jogi hátterével.
Reményeim szerint kitérünk majd arra is, hogy webes felületen milyen feltételek mellett lehet – akár értékesítési, akár más célból – szoftvereket közzétenni.

A szoftver(jog) tárgya

Egyesek szerint a szoftverjog, mint olyan, nem is létezik, hiszen sehol nincs egy önálló jogi kódex, amelyben minden, a szoftverekkel kapcsolatos jogi szabályozás megjelenne. Ráadásul rengeteg olyan jogterület van, melyeket a szoftverekre – azok egyedi megjelenési tulajdonságai miatt – nehéz alkalmazni.
Mások szerint a szoftverekben semmi különleges nincs, semmi olyan, ami miatt külön említést érdemelnének.
Én személy szerint úgy gondolom, hogy a bennünket körülvevő „digitális világ” élő és egyre fejlődő terület, mely nyilvánvalóan fontos részét fogja képezni a jövőnknek. Elég, ha csak az elektronikus kereskedelemre gondolunk. Ennek tulajdonképpen a háttérben működő szoftver a lelke, ahogy az egész virtuális világnak is. A terület különleges voltát pedig, úgy érzem, ezen a fórumon egyáltalán nem kell bizonygatnom.
Az, hogy a szoftverjog tárgya maga a szoftver – evidencia. Az viszont, hogy mi maga a szoftver, már rázósabb kérdés. Vegyük például a szoftver meghatározását. Mikor műszaki könyvekben kerestem e fogalom leírását, mintegy 15 különböző műben 15 eltérő változatot találtam. Ezek után talán nem meglepő, hogy a jogászok is meglehetősen bonyolultan látják a kérdést.
A jog számára elsődlegesen természetesen a jogszabályban, jelen esetben a szerzői jogról szóló 1999. évi LXXVI. törvényben rögzített meghatározás az irányadó.
Eszerint szerzői jogi védelmet élvez „a számítógépi programalkotás és a hozzá tartozó dokumentáció (a továbbiakban: szoftver) akár forráskódban, akár tárgykódban vagy bármilyen más formában rögzített minden fajtája, ideértve a felhasználói programot és az operációs rendszert is.”
Mosolyogva vehetünk egy nagy levegőt, hiszen legalább már van mibe kapaszkodni, aztán persze lelkesedésünk alábbhagy, mikor felötlik bennünk, hogy nem ártana pontosan meghatározni, mi is az a „számítógépi programalkotás”, illetve a „hozzá tartozó dokumentáció”.
A fenti sorokat boncolgatva rájöhetünk, hogy a jogászok tulajdonképpen a következő három csoportba próbálják besorolni a szoftvereket:

1, az operációs rendszerek,
2, a felhasználói programok,
3, az egyéb (ide tartozhatnak például a kiszolgálóoldali alkalmazások)

Az azért egyértelműen kitűnik a felsorolásból, hogy a védelem mindent átfog. Azaz majdnem mindent.
Valamely ötlet, elv, elgondolás, eljárás, működési módszer vagy matematikai művelet nem lehet tárgya a szerzői jogi védelemnek (akkor sem, ha csatolófelület – informatikusul interfész – alapját képezi). A védelem további feltétele még az egyéni, eredeti jelleg. De ha ez az egyéni, eredeti jelleg

már a szoftver félkész állapotában is kirajzolódik, a védelem értelemszerűen e pillanattól kezdve megilleti a művet. Ehhez nincsen szükség semmiféle külön jogi hókuszpókuszra, nincs bejelentési kötelezettség, nem kell lajstromszámot kérni, a szerzőt megilleti a védelem, egyszerűen azért, mert szerző.

Ha tehát az Olvasó legközelebb kitalál néhány sort egy újságcikkhez vagy éppen egy program forráskódjához, majd le is írja, vagy bármilyen egyéb módon dokumentáljajuk azt, akkor kérem, dőljön hátra egy kicsit és jóleső érzéssel a lelkében mélázzon el azon, hogy most szerzővé vált és az alkotása jogilag védett.

Persze azért nem árt a dolgot egy-két példányban lemásolni, és jól eltenni, mert a jog csak szerzői jogi jogsértések ellen véd, az áramszünet vagy merevlemez meghibásodása ellen nem. Ha ilyen módon veszik el szellemi alkotásunk, az bizony a mi hibánk.

A védett sajtpapír esete

Most, hogy már tudjuk, hogy minden sornyi programunk védett, áttérhetünk egy másik érdekes kérdésre. Tegyük fel, hogy reggeli közben – jobb épp nem lévén – a kezünk ügyébe eső sajtpapírra vetjük a kávé és a pirítós közötti villanásnyi szünetben kiötlött világmegváltó gondolatunkat, mondjuk műszaki szervezési ismeretek témakörben. Védett vagy sem ez az alkotás?

Ilyen esetekben elsődlegesen mindig azt kell eldönteni, hogy mekkora darab sajtpapírunk volt és vajon az általunk leírtak meghaladják-e az „ötlet, elv, elgondolás” szintjét.

A szerzői jog keze ugyanis az említett területekre már nem ér el. Ilyenkor ugyanakkor segítségünkre lehet a Polgári Törvénykönyv, amely az alábbi módon rendeli védeni a sajtpapírokat: 86.§ (4) A személyeket védelem illeti meg a vagyoni értékű gazdasági, műszaki és szervezési ismereteik és tapasztalataik tekintetében is. A védelmi idő kezdetét és tartamát jogszabály határozza meg.

Leszögezhetjük azt is, hogy a védelem nem függvénye a terjedelemnek. A szerzői jog ugyanúgy védi Pilinszky Négysorosát, mint Heller Huszonkettes csapdáját. Nincsenek esztétikai követelmények sem, vagyis az egyetemi feladatként összetákolt elégségessel jutalmazott kódunk ugyanúgy védett, mint azé az embertársunké, aki csodálatos módon jelest kapott. Persze ha arról van szó, hogy két különböző tanárhoz kell beadnunk a feladatunkat, s bár azokra ugyanúgy jelest kapunk, mint fent említett embertársunk, ám már egy merőben felületes összehasonlítással is megállapítható, hogy a két mű közötti egyetlen lényeges eltérés a beadó neve, akkor nem szerzői alkotásról, hanem bitorlásról beszélünk, melyet a törvény büntet(het). Ez azonban már egy másik történet.

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

A System Rescue CD

A Kedves olvasókkal most egy nagyon kellemes Linux darabot szeretnék megismertetni, ez pedig nem más mint a System Rescue CD. Ahogyan a neve is mutatja, nem kifejezetten otthoni vagy irodai használatra szánt Linuxról van szó, és a későbbiekben kiderül az is, hogy nem is csupán Linuxról.

Egy CD-ről lesz szó, amit többféle céllal is lehet használni. Egyik nagy előnye, hogy a hivatalos változat mindössze 102MB, amit ezért bárhová könnyen magunkkal cipelhetünk. Eltehetjük akár az irattárcánkba is, és ha éppen unatkozunk, máris kéznél van egy kellemes Linuxos környezet. Annak idején kíváncsiságból kezdtem foglalkozni a System Rescue CD-vel, de lassan megkedveltem és rájöttem, hogy egy nagyon hasznos segédeszközt tudhatok a kezemben; azóta sehová sem indulok el nélküle. Mindezek mellett a rendszer alapvetően rendszergazdai feladatok ellátására készült. Ennek megfelelően rengeteg hasznos segédprogramot tartalmaz, melyekből itt láthatunk egy összefoglalót:

- Linux rendszermag: v2.4.24
- Framebuffer-támogatás, nincs szükség az XWindow rendszerre
- QTParted – grafikus lemezfelosztó program
- parted – a klasszikus GNU lemezfelosztó program
- partimage – biztonsági mentésre
- dump és restore – a klasszikus biztonsági mentésre alkalmas programok
- sfdisk – felosztásitábla-kezelő program
- dar – tömörítőprogram (a `tar`-hoz hasonló)
- clam AntiVirus
- Midnight Commander
- szövegszerkesztők : vim, nano, QTinyEditor
- Samba, NFS és SSH támogatás
- Links szöveges böngésző

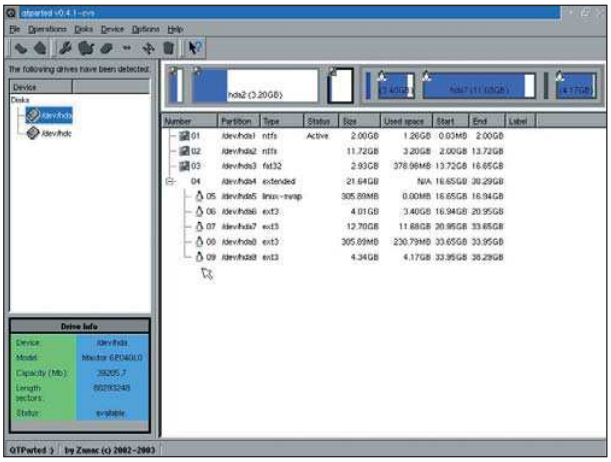
A fenti lista alapján bárki azt mondhatná, hogy „ide vele de hamarost!”, tehát máris indulhat a vadászat, amihez jó kiindulási alapot jelenthet a ➲ http://www.sysresccd.org honlap.

Nézzünk egy kicsit a dolgok mélyére

Amikor a CD elindul, rengeteg beállítási lehetőségünk van. Többféle segédprogram közül választhatunk és nem csak a Linuxot indíthatjuk el róla, de a hardverfelismerő- és információs programot, az aidát is. A lemezen lévő FreeDOS változat alapjában véve alkalmas lemezfelosztásra (a DOS-os fdisk található meg benne) és FAT fájlrendszer formázására. Aki nem ismerné az aida programot, röviden annyit elég tudni róla, hogy 45 képernyőn keresztül a számítógépünk minden egységéről, részletes információkat kaphatunk. A ranish lemezrészkezelő program is hasznos. Ugyan alapvetően csak a FAT16 és FAT32 típusú lemezrészeket képes kezelni, de van két említésre méltó tulajdonsága. Az egyik, hogy teljes merevlemezekről képes másolatot készíteni, a másik pedig, hogy felületi ellenőrzést végezhetünk a segítségével a megadott lemezrészen. A lemez tartalmaz még egy rendszerbetöltő-kezelő programot (`boot-manager`), egy memória ellenőrzőt (`memtest`) és egy segédprogramot, amivel az adatainkat véglegesen letörölhetjük a lemezekről. A fentebb említett programokat a CD elindulása után indíthatjuk a megfelelő lenyomat (***image***) kiválasztásával. Az induláskor egy szokásos LILO parancssort kapunk, ahol az `aida`, a `ranish` vagy a `freedos` beírásával indíthatjuk el a kívánt programot. Térjünk rá végül a CD lényegére, a Gentoo alapú Linuxra. Mint ahogyan eddig is, a megfelelő lenyomat nevének beírásával indíthatjuk el a rendszert, ám a választék igen bőséges. A rendszert indíthatjuk kép-puffer támogatás nélkül, a `nofb` lenyomatot indítva, vagy 800x600 és 1024x768 képpontos felbontásban a `fb800` és a `fb1024` segítségével. Külön lenyomat betöltésére van szükség, ha Intel810-alapú videovezérlővel rendelkezünk (`i810fb640`, `i810fb800` és `i810fb1024` néven érhetők el).

Indítsuk el

A megfelelő rendszermag kiválasztása után további paramétereket adhatunk meg az indításkor. Az első ilyen fontosabb paraméter, a `cdcache`, amellyel rávehetjük az induló rendszert, hogy a teljes CD tartalmát töltse be a memóriába. Ezután a CD-ROM meghajtót saját céljainkra használhatjuk, arra a rendszernek nem lesz szüksége.
A System Rescue CD alkalmas arra is, hogy USB meghajtóról (például pendrive eszközről) indítsuk, azonban ha ilyen módszerrel szeretnénk használni, akkor a rendszermagnak az `usbstick` paramétert is át kell adnunk induláskor. Természetesen a régebbi alaplapokra való tekintettel engedélyezhetjük vagy letilthatjuk az ACPI támogatást, mégpedig az `acpi=on|off|force` paraméterek valamelyikével.
Előfordulhat, hogy az adott számítógépben, amelyen ezt a kellemes eszközt használni szeretnénk, nincsen hálózati



1. kép A QTParted felülete

kártya. Felesleges is lenne tehát az automatikus felismerést használni, adjuk meg induláskor a `nonet` paramétert. Így elkerülhetjük a várakozást egy olyan műveletre, aminek már tisztában vagyunk az eredményével. Ugyanígy megadható a `noscsi` és a `nodetect` kapcsoló. Használatukkal elkerülhetjük a SCSI eszközök felismerését és teljes egészében kikapcsolhatjuk a számítógépben található eszközök felismerését. Szintén hasznos lehetősége ennek az egyszerű rendszernek, hogy képes automatikusan elindítani héjprogramokat. A rendszermag indulásakor megadható az `ar_source` paraméter, amivel azt magyarázhatjuk el az induló Linuxnak, hogy hol keresse az önműködően indítandó programokat. Ilyen programból összesen 10 lehet, elnevezésük a `autorunSZAM` megnevezést követi, ahol a `SZAM` nullától kilencig terjedhet. A programokat egymás után sorban indítja a rendszer, de ennek a sornak a végrehajtása megszakad, amint az egyik program hibakóddal tér vissza.
Az önműködően indítandó programok forrása lehet a CD-ROM gyökérkönyvtára, hajlékonylemezes meghajtó, NFS vagy Samba megosztás, a rendszergazda saját könyvtára, vagy a */usr/share/sys.autorun* program. Az `autoruns` paraméter után vesszővel elválasztva meghatározhatjuk azokat a programokat, amelyeket ténylegesen szeretnénk elindítani. Például ha paraméterként a `autoruns=1,3,6` karakterláncot adjuk meg, akkor a rendszer a megtalált programok közül sorban elindítja az `autorun1`, az `autorun3` és az `autorun6` nevű állományokat. Természetesen ezek a programok nem csak a héj által értelmezhető szöveges programok lehetnek, hanem lefordított, futtatható állományok is. Az utolsó paraméter, ami az önműködő programokra vonatkozik, az `ar_nowait`, amivel elérhetjük, hogy az utolsó program befejeződésével a rendszer ne várakozzon az ENTER billentyű leütésére.

Állítsuk be

Nézzünk meg néhány egyszerű beállítást, amit a rendszer indulása után érdemes elvégezni. Nagyon gyakran van szükség arra, hogy hálózati kártyánkat elindítsuk, amit ebben a rendszerben a `net-setup eth0` parancs segítségével tehetünk meg. A `net-setup` program paramétere a beállítandó hálózati csatoló neve. A parancs segítségével egy `dialog` alapú, szöveges felületen állíthatjuk be a szokásos paramétereket. A rendszer elindulása után vírusellenőrzést is végezhetünk a Clam AntiVirus programmal. Az ehhez szükséges lépések itt olvashatók. Először készítünk egy könyvtárat a vírusmeghatározások számára és az ellenőrizni kívánt fájlrendszer számára a `mkdir /virdefs /mnt/virtest` paranccsal. A következő lépésként a CD-n lévő vírusmeghatározásokat átmásoljuk a létrehozott könyvtárba. (`cp /usr/share/clamav/* /virdefs`). Ezután, ha rendelkezünk internet-hozzáféréssel és azt be is állítottuk, akkor a `freshclam --datadir /virdefs` parancs hatására a Clam AntiVirus frissíti az adatbázisát és mi pedig befűzhetjük az ellenőrizni kívánt fájlrendszert a `/mnt/virtest` könyvtárba. Következhet a vírusellenőrzés. Adjuk ki a `clamscan -i --bell -r -d /virdefs /mnt/virtest` parancsot, majd várakozzunk türelmesen. A víruskereső ilyen paraméterezéssel csak a fertőzött állományokat írja ki a kimenetre és hangjelzéssel jelzi a találatot. A keresés végén ne felejtsük el leválasztani umount (`umount`) a fájlrendszert.
Nos, elérkeztünk ahhoz a részhez, amikor a felhasználóban felmerülhet a kérdés, hogy hogyan is lehetne ezt a hasznos rendszert a felmerülő igényekhez igazítani. Szerencsére a fejlesztők erre is gondoltak, így néhány meglehetősen egyszerű lépéssel teljesen testreszabható a System Rescue CD. Először is szükség lesz egy lemezrészre (lehetőleg valamilyen linuxos fájlrendszerrel), amelyen van 500MB szabad hely. Ezt a fájlrendszert fűzzük be a */mnt/custom* könyvtárba. A következő lépés, az éppen használt rendszer tartalmának kicsomagolása. Ezt könnyedén elvégezhetjük a `sysrescd-custom extract` parancs kiadásával). A parancs hatására a */mnt/custom/customcd/files* könyvtárban elérhetjük a CD tartalmát, és ide másolhatjuk az esetleges önműködően induló programjainkat is, vagy akár beállíthatunk állandó hálózati címet. Egyszerűen szólva igényeinknek megfelelően átalakíthatjuk a rendszert.
Mindezek után következhet az újabb becsomagolás, a CD-lenyomatának (*image*) elkészítése és az új CD írása. A megfelelő parancsok segítségével készítsük el az újabb változatot tartalmazó CD-képet: `sysrescd-custom cloop 210 30000`. A program elkészíti az újabb lenyomat létrehozásához szükséges tömörített fájlrendszert, 210MB mérettel és legfeljebb 30000 állomány tárolására alkalmas formában.
A szabadon beállított változatnak megadhatunk alapértelmezett billentyűzetkiosztást is, mégpedig a `sysrescd-custom setkeymap hu` parancs segítségével. Ebben az esetben az alapértelmezett kiosztás a magyar billentyűzetnek megfelelő lesz. Végül pedig utolsó lépés a CD-lenyomat létrehozása: `sysrescd-custom isogen KOTETCIMKE`. Ahol a `KOTETCIMKE` lesz majd az új CD neve. Az elkészített állományt a */mnt/custom/customcd/isofile/* elérési úton találjuk meg, *sysresccd-new.iso* néven.
A rendszer tartalmazza a cdrecord programot is, tehát a CD-író meghajtónk felismerése után máris kiírhatjuk legújabb életmentő lemezünket.
Sajnos ebben a rövidke leírásban nem tudtam mindent bemutatni, amire képessé válhatunk a System Rescue CD használatával, de szerencsére a rendszer elindulása után a */root/manual-en/index.html* leírásból kiindulva, a `links` böngészővel minden részletre fény derülhet.

Fábián Zoltán