Beköszöntő

Bár szerkesztőségünk számára a 2004-es és is meglehetősen mozgalmas volt, olvasóink ebből viszonylag keveset érezhettek, hiszen az eddigi változások inkább a színfalak mögött zajlottak. 2005-ben terveink és reményeink szerint egy új korszak kezdődik a lap életében, melynek legfontosabb jellemzője a tartalomnak a felhasználói irányba való fokozatos eltolódása lesz. Szeretnénk új témákat, és új arcokat bemutatni a magyar linuxos közönségnek, illetve szeretnénk ennek a közösségnek valamiféle központi fórumává válni, valahogy úgy, ahogy egy-egy szaklap fóruma és elsődleges információforrása lehet a megfelelő tudományterületnek.

A Linuxvilágot tehát mostantól nemcsak olvasni, írni is lehet! Bárkitől szívesen fogadunk a témához kapcsolódó, és közérdeklődésre számot tartó írásokat. Terveink szerint hamarosan egy cikkírói pályázatot is meghirdetünk, amelynek nem titkolt célja a „vérfrissítés", és az új témák és lehetőségek felkutatása.

És most lássuk, mit kínálunk az új esztendő első számában.

E havi számunk talán legérdekesebb témája a videószerkesztés ***Linux*** alatt. Két cikk is foglalkozik ezzel a területtel. ***Marcel Gagné*** a maga jól ismert, egyedi stílusában teszi ezt, két orosz szerzőtől (***Olexiy Tykhomyrov*** és ***Denis Tonkonog***) pedig egészen komoly bevezetőt kapunk a ***Kino*** nevű alkalmazás használatába. Utóbbi cikk tulajdonképpen elegendő információt tartalmaz ahhoz, hogy segítségével berendezzük saját házi ministúdiónkat, hiszen részletesen szól a hardver- és szoftverkövetelményekről, kitér az alkalmazott szabványokra, sőt még a szükséges kábelekre is.

A Linux lelkivilágát mélyebben ismerő, illetve az iránt érdeklődő felhasználóknak szóló cikkek közt akad egy, amelyik egy speciális feladatütemezőt ismertet, a különleges műszaki megoldások kedvelői pedig megtudhatják, hogyan lehet egy épület valamennyi légkondicionáló berendezését központilag vezérleni – természetesen Linux segítségével.

Folytatódnak a korábban megkezdett sorozatok. ***Komáromi Zoltán*** a ***PHP 5*** újdonságait tárgyalja, mégpedig elsősorban az objektumközpontú lehetőségek szemszögéből, ***Auth Gábor*** a külső programok telepítésének ismertetésénél tart a ***FreeBSD*** bemutatásában, ***Dudás Ágnes*** pedig az átdolgozás jogával kapcsolatos részleteket tárgyalja szoftverjogi sorozatának ebben a részében. Folytatódik ***Fábián Zoltán*** lassanként önálló rovatnak tekinthető sorozata, a ***GIMP*** bemutatása is. Ebben és a következő részekben a különböző modulokról és bővítési lehetőségekről esik szó.

Kellemes időtöltést, jó szórakozást kíván a Linuxvilág csapata!

A Linuxvilág jövője

Avagy hova tűnt a Programvadászat rovat és a CD?

Az elmúlt hónapokban gyakori vitatéma volt a szerkesztőségben a Linuxvilág jövője.

- Milyen legyen a kezdőknek, a haladóknak és az abszolút profiknak szóló cikkek aránya?
- Legyen-e CD?
- Egyáltalán van-e igény erre egy olyan korban, amikor a felhasználók többsége egyre nagyobb sávszélességű kapcsolattal rendelkezik, és gyakorlatilag mindent, programot, dokumentációt, oktatóanyagot, híreket a netről szerez be?
- Legyen-e ennek a trendnek megfelelően elektronikus kiadásunk, ami lehetővé teszi, hogy mindenki csak azokat a cikkeket fizesse elő vagy töltse le egyedileg, amelyekre valóban szüksége van?

Talán ezek voltak a leggyakrabban felmerülő kérdések. Két felmérést is végeztünk, amelyek jelentősen segítettek bennünket abban, hogy a magazin új „főcsapását” kijelöljük.
Tekintettel arra, hogy az olvasók, illetve potenciális olvasók többsége nyilatkozott úgy, hogy jobban örülne egy CD nélkül megjelenő, de olcsóbb lapnak, 2005-től a mellékletet megszüntettük, a lap árát pedig jelentősen csökkentettük.
Szeretnénk azonban rögvest leszögezni: ez nem azt jelenti, hogy soha többé, semmilyen formában nem lesz CD vagy DVD. Valószínűleg a gyors és egyre olcsóbb internet korában is számos, a digitális külvilágtól még részben elszigetelt embernek lehet szüksége például arra, hogy lemezen juthasson hozzá egy-egy újabb terjesztéshez, amit nem tud letölteni. Nekik valamilyen formában mindenképpen segítséget kívánunk ehhez nyújtani, hiszen egyik célunk éppen a Linux terjedésének elősegítése.
Hamarosan beindul egy olyan elektronikus előfizetési rendszer is, amelyen keresztül az egyes cikkek egyenként is letölthetők lesznek.
Terveink közül a leglényegesebb azonban a lap „alapstílusának” megváltoztatása. Bár a felmérések azt mutatják, hogy a jelenlegi előfizetőknek legalább a fele a „profi” kategóriába tartozik, úgy érezzük, a példányszám növekedésének – a magas ár mellett – a leglényegesebb fékező ereje az volt, hogy a haladó illetve középhaladó felhasználók nem találtak kellően sok nekik megfelelő cikket.
A jövőben tehát sokkal inkább az alkalmazások bemutatására szeretnénk koncentrálni, és nem a csak informatikai szakemberek számára érthető különleges műszaki megoldásokra és fejlesztésekre. Ezzel párhuzamosan valószínűleg komoly változások lesznek a rovatok szerkezetében is. Újak jelennek meg, egyesek pedig eltűnnek, vagy más néven bukkannak majd fel. Erre az elég komoly stílusváltásra ebben a pillanatban még nem vagyunk teljesen felkészülve, tehát csak fokozatosan, néhány hónap alatt tudunk majd a fent vázolt irányba elmozdulni.
Végül néhány szó a teljesen kezdőkről. Félreértés ne essék, ezt a kifejezést most nem a levelezési listákon megszokott értelemben használjuk. Számunkra teljesen kezdő az, aki még soha nem látott Linuxot, soha nem telepített még önállóan ilyen rendszert, és jó esélye van rá, hogy ha ez sikerülne is neki, akkor se találná meg a módját annak, hogy megjelenítse például egy a meghajtóba helyezett CD tartalmát.
Az ilyen felhasználók problémái jó esetben erősen átmenetiek. Egy jó könyv elolvasása és néhány napi vagy akár pár órányi „játék” után rendszerint minden világossá válik. Ez pedig azt jelenti, hogy ettől a ponttól kezdve jó esélyük van arra is, hogy megértsék a haladóknak és középhaladóknak szóló cikkeket.
Mindebből az következik, hogy a teljesen kezdőknek nem lesz külön rovata a lapban, egyszerűen azért, mert ez nem oldható meg. Értelmetlen volna minden hónapban közölni mondjuk egy-egy terjesztés telepítésének részletes leírását, hiszen a többség ebből semmit nem profitálna, az a néhány „újonc” pedig, aki éppen akkor csöppent bele a Linux világába a következő számig már túl is van a traumán.
Őszintén hisszük, hogy a fent vázolt elhatározásaink jelenlegi és leendő olvasóink örömére szolgálnak majd, és munkánkkal hozzájárulhatunk a szabad szoftverek és a Linux elterjedéséhez Magyarországon.
A változásokkal kapcsolatban természetesen szívesen veszünk minden észrevételt, javaslatot! Igyekszünk minden ésszerű felvetést megfontolni, és minden levélre válaszolni.

A véleményeket a
➲ szerkesztoseg@linuxvilag.hu
címre várjuk.

A Linuxvilág stábja

Wattok garmadája

A japán *SNE* minden eddiginél nagyobb teljesítményű tápegységeket dobott piacra asztali gépekhez. A 950, 900 és 850 W teljesítményű tápok normál *ATX* szabvány szerinti házakba, illetve *EPS12V* rendszerekbe, vagyis munkaállomásokba építhetők be.
A célközönség – noha egy korszerű, csúcsteljesítményű munkaállomás is komoly energiafogyasztással dicsekedhet – egyértelműen a tuning iránt érdeklődők közössége. Ugyanakkor a gyártó arra is felhívja a figyelmet, hogy tápegységei a fenti különlegesen nagy teljesítményt csak rövidebb ideig bírják, huzamosabb működésük során ennél kisebb terhelést viselnek csak el. Az extrém tápok ára rendre 570, 475 és 427 dollár, vagyis 100000 forint körül mozog.
➲ www.sne-web.co.jp

Pofákat vágnak

A kanadai *Sheridan Műszaki és Fejlett Tanulási Intézet* a *Torontoi* és a *Queens Egyetemmel* együttműködve, a *Silicon Graphics Onyx4 Ultimate Vision* megjelenítő megoldására építve a testbeszéd a korábbinál jóval pontosabb elemzésére alkalmas rendszert dolgozott ki. A végeredmény egy szimulált emberi arc, ennek mozdulatait, kifejezéseit a kutatók rendkívül nagy részletességgel tudják szabályozni.
Mindennapi életünk során akarva-akaratlanul mindannyian számos ilyen metakommunikációs elemet használunk, és sokat tudatosan vagy tudattalanul veszünk, értelmezünk is, ám a közöttük fennálló összefüggéseket még nem sikerült részletesen feltárni. A jelenlegi kutatás egyik újdonsága, hogy az arc egyes kifejezései, mozdulatai – az emberi arcoktól eltérően – egymástól teljesen függetlenül szabályozhatók, így önálló jelentésüket is pontosabban meg lehet ismerni.
Természetesen az emberi arc számítógépes modellezése nem újdonság – gondoljunk csak az újabb rajzfilmekre –, ám mindezt a pszichológusok és a nyelvészek igényeinek megfelelő formában megvalósítani korántsem volt egyszerű. Az arcmodellt a kutatók egy emberi arc háromdimenziós letapogatásával, az izomzat jellemzőinek figyelembe vételével alakították ki, ennek valós idejű leképezéséről gondoskodik a *Silicon Image* 32 darab szorosan csatolt grafikai processzora. A vizsgálatok eredményeit több területen, többek közt a kommunikációs nehézségekkel küzdő, például autista személyek kezelése során is fel tudják majd használni.

Solaris 10

Hivatalosan is bemutatta *Solaris 10* operációs rendszerét a *Sun*. A háromezer mérnöki emberóra árán és félmilliárd dolláros fejlesztési költséggel fejlesztett, több mint 600 új szolgáltatást tartalmazó *Solaris 10* állítólag minden idők legfejlettebb *UNIX* operációs rendszere, mely január végére *SPARC*, *x86* és 64 bites *AMD* és *Intel* processzorokra egyaránt elérhető lesz – méghozzá ingyenesen. Az új rendszer legfontosabb újdonságai között szerepel a *DTrace* hibakereső eszköz, a szoftverpartíciók létrehozásának lehetősége, a folyamatok jogosultságainak kezelése, a prediktív öngyógyítás, a linuxos alkalmazások módosítások nélküli futtatása, a hatalmas kapacitású *ZFS* fájlrendszer és a titkosítási keretrendszer.
Természetesen az 1991 óta fejlesztett operációs rendszer ingyenessé tétele miatt sem kell a *Sun*-t a tönkremeneteltől féltenünk. A cég az utóbbi időben egyre inkább a termékeihez fűződő szolgáltatásokból, előfizetésekből próbál fennmaradni, ennek az átállási folyamatnak a része a *Solaris* ingyenessé tétele is.
Ugyanakkor a *Solaris* jövőjét sem akarják csupán az asztali és kiszolgáló gépekre alapozni, a Sun is ki akarja hasítani a maga szeletét a beágyazott készülékek piacáról, amely egyébként a legyártott processzorok 60 százalékát használja fel. Ehhez a *Solarisnak* kisebb teljesítményű gépeken is működnie kell, továbbá magas fokú modularitás biztosításával lehetővé kell tenni a fejlesztők számára, hogy ki tudják választani az adott készülék működtetéséhez szükséges programrészeket. Mondhatnánk, hogy a *Sun* kissé későn ébredt, hiszen a *Linux* vagy a *Windows* átszabott változatainak fejlesztése már évek óta folyik, ám a *Java* révén a cég nem teljesen járatlan ezen a területen, és a *Solaris* későbbi, a különleges igényeknek még jobban megfelelni képes változataival jó esélye van a hangsúlyos szereplővé válásra.
➲ www.sun.com

Mindentudó Opera

Az *Opera Software* megkezdte új, *Extensible Rendering Architecture (Bővíthető Leképező Architektúra, ERA)* nevű leképező-megjelenítő megoldását alkalmazó, asztali gépekre készült böngészőjének tesztelését. A várhatóan a 7.60-as *Operában* megjelenő alrendszer a kis- és közepes méretű, az asztali gépekre jellemző és a tévés képernyők, monitorok kezelésére egyaránt képes lesz. Az *ERA* dinamikusan át tudja méretezni a weboldalak tartalmát, így az oldalakat bármekkora méretű képernyőn képes úgy megjeleníteni, hogy a felhasználónak oldalirányú görgetést nem kell végeznie. Mivel így a teljes tartalmat könnyen át lehet majd tekinteni, könnyebbé, élvezetesebbé válik a használat, és várhatóan újabb alkalmazási területek nyílnak meg az *Opera* előtt.

➲ www.opera.com

Sosem áll meg

Az *NEC Solutions America* magas fokú hibatűrésre képes, *Linux* alapú kiszolgáló gépet mutatott be. A 99,999 százalékos rendelkezésre állású, *Express5800/320Lb* jelzésű számítógép kialakításakor semmit nem bíztak a véletlenre: gyakorlatilag mindenből kettőt vettek.

A számítások végrehajtásáról összesen négy *Xeon MP* processzor gondoskodik, ezeket két modulra osztották el. A modulok külön 4 egység magas fiókokban helyezkednek el, és szinkron működésüknek köszönhetően bármikor képesek átvenni egymástól a végrehajtást. A gépen 2.4.18-as *Linux* rendszermag fut, ezt a hozzá tartozó illesztőprogramokkal együtt a magas rendelkezésre állás követelményeinek megfelelően szabták át az *NEC* mérnökei. Az *Express5800/320Lb* ára 25500 dollártól, vagyis körülbelül ötmillió forinttól indul.

➲ www.nec.com

Kettőslátás

Az *Asus* lett az a gyártó, amely elsőként mutatta be az *NVIDIA* egyes grafikus kártyái által támogatott *SLI (Scalable Link Interface)* megoldás kihasználására alkalmas, *AMD* processzorokat fogadó alaplapját.

Az *A8N-SLI Deluxe* a szokásos egytől eltérően két *PCI Express (PCI-E) x16* foglalatot tartalmaz, így két *VGA* kártya behelyezését is lehetővé teszi. A kártyákat egy apró modullal kell összekötni, ettől kezdve egymással együttműködve képesek megosztani a leképezési feladatokat, amivel gyakorlatilag megkétszerezhető a megjelenítés teljesítménye. Az *Asus* alaplapjának tervezésekor nemcsak az *SLI* modul elhelyezését kellett figyelembe venni, de a nagyteljesítményű *VGA* kártyák áramfelvételét és hőtermelését is. Természetesen, hűen a korábbi *Deluxe* jelzésű alaplapok hagyományához, az *Asus* nem elégedett meg ennyi extrával, az alaplap *AI NOS* szolgáltatása intelligens eljárásokkal, mindig a szükséges pillanatban képes növelni a processzor órajelét, az *AI NET* az operációs rendszer elindítása előtt is képes felismerni a hálózati kábelek hibáit, az összesen nyolc *SATA* csatlakozón akár két különálló *RAID*-rendszert is kialakíthatunk, a hálózati és egyéb kapcsolatok létrehozására pedig két gigabites *LAN* csatoló és két *Firewire* csatoló áll rendelkezésünkre.

➲ www.asus.com

GPL 3.0

A *Free Software Foundation* megkezdte a *GNU General Public License* újabb, harmadik változatának kidolgozását. Az *FSF* keretein belül eredetileg *Richard Stallman* készítette a szerződést, melynek legújabb változata 1991-ben látott napvilágot. Természetes, hogy 14 év alatt a szoftverek világ hatalmasat változott, így a kamaszkorba lépő szerződés képtelen az újabb követelményeknek megfelelni. A legújabb kiadásban a készítők igyekeznek figyelembe venni a szellemi tulajdonnal és a szabadalmakkal összefüggő, a hálózati használat, a webszolgáltatások indítása kapcsán megjelenő és a megbízható számítástechnika várható elterjedése során felmerülő kérdéseket. További kiküszöbölésre váró problémaforrás az egyes országok szabadalmi jogrendszerének, sőt, jogi hagyományainak eltérése, ami önmagában is jelzi a munka nehézségét. A *GPL 3* elkészítése mindettől függetlenül több mint időszerű – ezt önmagában is igazolja az a tény, hogy az idők során több mint ötvenféle féligmeddig szabad szerződést készítettek –, elődje az eredetileg tervezettnél már így is jóval tovább szabályozta a szabad programok világát.

Leckekereső

Google Scholar névvel új kereső szolgáltatást indított a *Google*. A cél a legkülönfélébb területekhez kapcsolódó tudományos jellegű írások, műszaki jelentések, egyetemi weboldalak és könyvek elérésének segítése. Mindezek a tartalmak ugyan felkerülnek a webre, ám túlnyomó részük csak jelszó megadásával, sokszor előfizetés birtokában érhető el, ami rendkívül megnehezíti az adott témában keresgélő látogatók, és persze a keresőmotorok dolgát. A *Google* csapata számos egyetemmel, kiadóval együttműködve valósította meg a keresőt, mely ugyanakkor teljes értékű hozzáférést nem biztosít, ám az írások rövid kivonatait könnyen elérhetővé teszi. A kereső érdekessége, hogy a találatokat – a *Google* normál kereséseihez hasonlóan – nem a látogatottság, hanem a más szerzők által végzett idézések alapján rangsorolja. Felmérések szerint az utóbbi években a kutatók és a diákok egyre inkább a webes források felé fordultak munkájuk és feladataik megoldása során – ezt az irányzatot a keresőknek is követniük kell.

➲ www.google.com/scholar

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

Mi újság a rendszermag fejlesztése körül?

Közel egy évnyi versengés után a *pmdisk* és az *swsusp* újra *swsusp*-ként egyesül. Fejlesztését továbbra is *Pavel Machek* fogja vezetni, amiben az egykori szétválást kezdeményező *Patrick Mochel*, illetve a szoftveres felfüggesztés terén érdekelt fejlesztők segítségére egyaránt számíthat. *Patrick* szívhez szóló bocsánatkérést intézett *Pavelhez* a kód szétágaztatása miatt, s ettől kezdve szépen együtt folytatták a munkát, megosztva a foltokat és átbeszélve a műszaki jellegű kérdéseket. Jó látni, hogy a vita elsimult, ugyanis alapját közös szenvedélyük adta, nevezetesen a kód lehető legjobbá tétele annak érdekében, hogy biztos működésű szoftveres felfüggesztési szolgáltatást valósítson meg, és a lehető legkevesebb gond árán válhasson a hivatalos rendszermag részévé.

A *cryptoloop* nagy valószínűséggel kikerül a 2.6-os sorozatból, hacsak *Andrew Morton* meg nem várja ezzel a 2.7-est. Bárhogy is történik, az tisztán látható, hogy a *cryptoloopnak* mennie kell. A kód, melynek segítségével a felhasználók a helyi hurkon keresztül titkosított fájlrendszereket fűzhetnek be, a jelek szerint nemcsak hibás, de karbantartás nélkül is maradt, ráadásul súlyos biztonsági hibáktól terhes. *Andrew* szempontjából leginkább a biztonsági hiányosságok teszik elfogadhatatlanná a *cryptoloopot*. Jobb, ha nem áll rendelkezésünkre egy lehetőség, mintha egy a biztonságot hamisan ígérő szolgáltatást használnánk. Természetesen, mint arra több fejlesztő is felhívta a figyelmet, az ilyen nagyobb változtatásokat soha nem szabad(na) üzembiztos sorozaton végrehajtani, de így is legfeljebb késleltetni lehet az elkerülhetetlent. Hacsak valaki színpadra nem lép, és el nem vállalja a karbantartást és a biztonsági hiányosságok pótlását, a *cryptoloop* jövője igencsak rövid lesz.

A *Reiser4* fájlrendszer, amely rendkívül messze került a *Reiser3* változattól, az *-mm* rendszermagsorozat (*Andrew Morton* saját fája) része lett, és öles léptekkel halad a hivatalos kiadás részévé válás irányába. A *Reiser*-rajongók természetesen lelkesednek, miközben rohamosan nő a felhasználói tábor, és vele együtt a hibajelentések száma is. A *Reiser4*, mely egyelőre csak a fő szolgáltatásokat tartalmazó változatban jutott el *Andrew*-hoz, állítólag a linuxos világ leggyorsabb fájlrendszere. Persze ilyesmit állítani még jogosan is csak ideiglenesen lehet, azt viszont kár volna kétségbe vonni, hogy az újabb generációs fájlrendszerek kiváló általános teljesítményt nyújtanak, ami főleg az intenzív lemezhasználattal járó alkalmazások futtatásakor fontos.

Andrew Morton, mint a 2.6-os sorozat karbantartója kétségbe vonta néhány hagyomány fenntartásának értelmét, ilyen például az üzembiztos és a fejlesztői rendszermagsorozat fenntartása. Ugyan annyira azért nem rúgná fel a meglévő rendszert, hogy megszüntesse a párhuzamosságot, ám szerinte inkább a terjesztések készítőire kellene bízni az üzembiztosság megteremtését. A hivatalos rendszermagforrások összeállításakor szerinte – az üzembiztosság mellett, és nem helyett – a sebességre és a szolgáltatások bővítésére kellene összpontosítani, és az üzembiztosságra nem kellene olyan kiemelt figyelmet fordítani, mint az utóbbi néhány fő kiadás esetében.

A kérdéssel kapcsolatban sokféle álláspont létezhet és létezik, ám ne feledjük, *Andrew*, *Linus* és a többiek mindig saját ötleteik és a másoktól kapott tanácsok alapján módosítják a fejlesztési modellt. *Andrew* legújabb ötletére némi jóindulattal úgy tekinthetünk, mint a rendszermag fejlesztési munkálataiban segédkezők körének kibővítésére tett kísérletre, amelynek eredményeként a terjesztések készítői is a folyamat teljes jogú, elsődlegesen az üzembiztosságra ügyelő résztvevőivé válnának.

A *DevFS*, mely a *Linux* történetében a legnagyobb viták egyikének tárgya volt, a jelek szerint egyre inkább sodródik afelé, hogy végleg kikerüljön a rendszermagból. *Andrew Morton* úgy nyilatkozott, hogy a 2.8-as rendszermagban már biztosan nem lesz *DevFS*, a végleges eltávolítást 2005 közepére tervezik. Mindig is erősek voltak azok a hangok, amelyek a *DevFS* megtartását követelték, amíg helyettese, az *udev* nem képes a *DevFS* összes szolgáltatását biztosítani. Hibái ellenére a *DevFS*-t bizony nem lesz könnyű leváltani. *Richard Gooch* hatalmas mennyiségű munkát ölt bele, és az *Alexander Viro* és mások részéről érkezett rengeteg bírálat ellenére máig sem sikerült lecserélni – közel két évvel azután, hogy *Richard* felhagyott a tervezettel és a rendszermag fejlesztésével.

Zack Brown

Linux Journal 2004. december, 128. szám

Új termékek

TextMaker Zaurushoz

A német *SoftMaker* cég a *TextMaker* nevű szövegszerkesztője immár elérhető *Zaurus* platformon is. A *TextMaker for Zaurus* konvertálás nélküli közvetlen dokumentumkapcsolatot tesz lehetővé más *TextMaker* platformokkal az eredeti formázás megtartása mellett. A dokumentumcsere terén több *Microsoft Word* verzió, valamint egyéb fájlformátumok között válogathatunk, pél-

dául használhatjuk a *HTML*, *RTF*, *ASCII* és *Unicode* szabványokat. Ezen kívül többnyelvű helyesírás elemző és elválasztás kezelő programot tartalmaz, a vezérlőcsíkok és a billentyűzetkiosztás teljesen testre szabhatóak, valamint beépített dBase rendszerű adatbázis és fájlkezelő rendszerrel rendelkezik. A *TextMaker for Zaurus* bármelyik *Zaurus* módban futhat, kezeli a nagy felbontású képernyőket és fekvő lapállást, valamint a belső memóriába és memóriakártyára is telepíthető.
www.softmaker.co

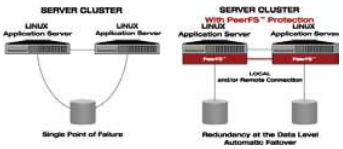
Appro 1U és 4U Quad Opteron Kiszolgálók

Az *Appro* bejelentette két új, négy *Opteron* processzor köré épített, *1U-1142H* és a *4U-4148H* jelű kiszolgálóit. A maximum négy *Opteron* processzorral rendelkező *1U-1142H Quad Opteron Server* párhuzamosan 32-bites és 64-bites számítási képességgel rendelkezik, maximum 32GB *ECC*

333/400 *DDR* memóriát kezel, két *IDE*, *SCSI* vagy *SATA HDD*-t, vékony *FDD* és CD meghajtókat vagy *DVD-ROM* meghajtót tartalmazhat és egy teljes *PCI-X* 64-bit/133MHz hellyel rendelkezik. A *4U-4148H* ezen felül nyolc *SATA* vagy *SCSI HDD*-t, maximum öt *PCI-X* helyet és 3.25" *FDD* illetve 5.25" meghajtóhelyeket tartalmaz CD/DVD-ROM vagy szalagos egységek számára. Mindkét kiszolgálót távoli kiszolgálóvezérléssel és nagy sebességű kapcsolattal szállítják.
www.appro.com

PeerFS Adatmásolat készítés

A *Radiant Data Corporation* kihozta *PeerFS* nevű, Linux alapú üzleti alkalmazásokhoz szánt egyenrangú (peer-to-peer), folyamatos másolatkészítő megoldását.
A *PeerFS POSIX*-szabványú fájlrendszer, mellyel több forrásból származó és több célnak szánt



adatok egységesen és naprakészen tarthatók. Az adat folyamatosan látható és felhasználható bármely helyi alkalmazás számára. A *PeerFS* képes egyidejű műveleteket végezni több kiszolgálón található különböző célokon, szétválasztott de logikailag azonos adattárolókon, megkönnyítve a *MySQL* és más webkiszolgáló összetevők kezelését. Teljes 256-bites AES titkosítást lehet beállítani bármely végponton, továbbá a *PeerFS* a végpontok között tárhely leállás esetén is zökkenőmentes hibakezelést biztosít. A PeerFS bármely alkalmazással vagy adatbázissal képes együttműködni és nem kényes az alkatrészekre.
www.radiantdata.com

Adonis DNS/DHCP eszköz

A *BlueCat Networks Adonis DNS/DHCP Appliance* eszköze egyenesen a vállalati hálózatba épül be, az egységes vállalati IP beállítás és gépnévkezelés érdekében integrálva a *DNS* és *DHCP* szolgáltatásokat. A legfrissebb *Adonis* eszköz már tartalmazza a *crossover high availability (XHA)* nevű szerkezetet amellyel elkerülhető a *DNS* és *DHCP* szolgáltatások leállása; az IP kölcsönzés elosztását táblázatos vagy grafikus formátumban megjelenítő *lease viewer* kezelőfelületet, egy adat ellenőrző rendszert, amellyel a *DNS* és *DHCP* beállítások telepítése előtt felderíthetők a hibák, valamint maximum 100 szintű visszavonási és másol/vág/beilleszt funkciókat kínáló egyszerűsített szerkesztést.
bluecatnetworks.com

BiTMICRO Networks E-Disks

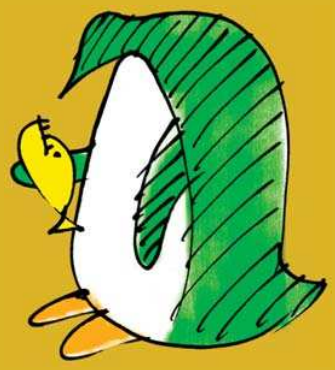
A *BiTMICRO Networks* bemutatta új fajta, *E-Disks* néven futó, homogén, lemez alapú *iSCSI* cél eszközét. Az *E-Disks Fibre Channel*, *Ultra320 SCSI*, *PATA* és *SATA* változatban kapható. Bármely változatot alkalmazhatjuk vállalati környezetben, Internetes kommunikációban és hagyományos ipari alkalmazásokban. Az *E-Disks* szilárd, kezelhető, együttműködő és kezelhető *Flash*-alapú hálózati tároló megoldás kíván nyújtani, amely nagy távolságokra ér el, könnyen összekapcsolható *SAN* rendszerekkel, több felhasználó használhatja párhuzamosan és könnyen boldogul a már létező alkalmazásokkal.

www.bitmicro.com

Ők mondták

Nem önmagában a nyílt forráskód zavarja őket, sokkal inkább az, hogy most már nem ők az olcsó megoldás. Korábban a *Microsoft* volt az alacsony költségű megoldás, így ennek megfelelően alulról támadva próbált versenyezni a drága üzleti rendszerekkel. Most először fordult elő, hogy a szerepek felcserélődtek és a *Microsoftot* támadják alulról alacsony költségű megoldások.

Brad Silverberg,
a *Microsoft* igazgatócsoportjának volt tagja
(➲ www.milestone-group.com/news/04_07/Silverberg.html)



Nincs jó *DRM (digital rights management)*. Pont.

Jean Bedord,
a *Shore Communications* kiadóipari elemzője. Az idézet *David Becker „Have e-books turned a page?"* írásában olvasható a *CNET News.com* oldalon
➲ *news.com.com/Have+e-books+turned+a+page%3F/2100-1025_3-5326015.html?tag=st.pop*

Nyílt szabványok nélkül, a program szabadsága csak illúzió. A nyílt szabványok a programok szabadságának alapját jelentik.

Larry Rosen
a Nyílt forráskódról, Nyílt szabványokról, előszóban

Linux Journal 2004. 128. szám

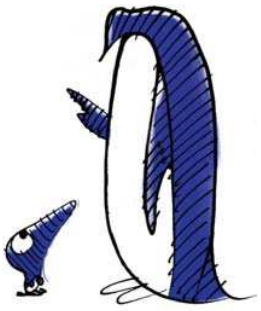
Linux-index

1. A *Linux* előre jelzett növekedési ütemének számjegyei: **2**
2. Azon *UNIX* felhasználók százaléka, akik szívesen váltanának platformot: **4**
3. Azon *Microsoft Windows* felhasználók százaléka, akik szívesen váltanának platformot: **10**
4. Nyílt forráskódú projekteken dolgozó fejlesztők száma *Észak Amerikában*, millióban: **1.1**
5. *Észak Amerikában* legalább ennyi millió fejlesztő dolgozik 64-bites rendszereken: **.5**
6. *Észak Amerikában* közel ennyi millió ember dolgozik grid számítási projekteken: **.25**
7. *Észak Amerikában* közel ennyi millió fejlesztő dolgozik fürtözött megoldásokon: **.5**
8. Az fürtözött megoldásokon dolgozó *Észak Amerikai* számítástechnikai fejlesztők százaléka: **17**
9. Ennyi ezer párhuzamos hanghívást képes kezelni percenként a *Linux* alapú *Készültségi Válaszadó Rendszer (Emergency Response System)*: **10**
10. Ennyi ezer párhuzamos bejövő forródrótot tud kezelni az *ERS*: **30**
11. Ennyi ezer párhuzamos fax hívást kezelhet az *ERS*: **5**
12. Ennyi ezer párhuzamos szöveges üzenetet kezelhet az *ERS*: **5**
13. Az internet százalékos növekedési üteme az elmúlt 12 hónap alatt 2004-ig bezárólag: **26.1**
14. Ez alatt az időszak alatt felvett új gépnevek száma millióban: **10.7**
15. A *Netcraft* által felmért helyek száma 2004 júniusában millióban: **51.635284**
16. Ennyi millió helyen futtatnak *Apache* kiszolgálót: **34.710235**
17. A összes kiszolgálóhoz képest az *Apache* százalékos aránya: **67.22**
18. Az részesedésének növekedése a múlt hónapban: **.17**
19. *Microsoft IIS* rendszert futtató kiszolgálók százalékos aránya: **21.35**
20. A *Microsoft IIS* csökkenése a múlt hónapban: **-0.13**

Források

1-3: *LinuxWorld*, a *Yankee Group* jelentéséből válogatva
4-8: *Evans Data Corporation*
9-12: *Emergency Response Network*
13-20: *Netcraft*

Linux Journal 2004. december, 128. 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

Megjelenítés problémák Compaq Laptopon

Compaq Presario 2500-as gépet és ***Fedora Core 2***-t használok. Az a gond, hogy képernyő állandóan beszűkül, de fogalmam sincs mi okozza. Mit tegyek?

Anonymous,
➲ CC04002@STD.KUKTEM.EDU.MY

Az általad leírt problémához hasonló esetek ismeretesek az ***ATI RADEON*** videókártyákkal kapcsolatban. Valószínűleg a te ***Presario 2500***-asod is ilyet használ. Érdemes megpróbálni az ***/etc/X11/xorg.conf*** állományban megjegyzésbe tenni a `Load dri` sort, majd újraindítani az X kiszolgálót. Előfordulhat azonban, hogy ez az adott gépen váratlan eredményhez vezet.

Felipe Barousse Boué,
➲ fbarousse@piensa.com

ACPI, Sleep és Standby

ASUS L8400B laptopom van, 700MHz P-III processzorral. Nemrég tettem fel a ***Fedora Core 2***-t 2.6.5-ös rendszermaggal. Ez a rendszermag ***ACPI***-t használ a 2.4-es sorozatban alkalmazott ***APM*** helyett, így a laptop működésének felfüggesztése máshogy, elég idegesítően működik.

APM alatt az `apm` paranccsal tudtam ***sleep*** vagy ***standby*** módba váltani. Az első amennyire én tudom, mindent a memóriában tart; a második, gondolom, a lemezre mentené a környezetet majd alvó állapotba lép. A laptop felébresztéséhez mindkét esetben mindössze egy billentyűt kellett megnyomnom a laptopon.

ACPI alatt, ezt látom a ***/sys/power/state*** fájlban:

```
standby mem disk
```

A `disk` szót küldve a ***/sys/power/state***-be semmi sem történik. Bármelyik másik kifejezést küldve a laptop futása felfüggesztődik. Csakhogy nem ébred fel amikor a billentyűzetet csapkodom. Kizárólag akkor ébred fel újra ha a bekapcsológombot használom. Csakhogy ilyenkor a rendszer észleli, hogy megnyomtam bekapcsológombot és azonnal elindítja a rendszerleállítást.

A ***Google***-lel végzett keresés nem sok eredményre vezetett, az ***ACPI FAQ*** úgy tűnik legalább három éves.

Szóval a közvetlen kérdéseim:

1) Támogatott felfüggesztés lemezre funkció? Ha igen, hogy lehet beizzítani?
2) Hogyan kell helyesen felébreszteni a laptopot?
3) Mit kell még tudnom? Eléggé kezdő vagyok ***ACPI*** terén.

Ha lehet kerülném a rendszermag frissítését és hasonlókat, de megcsinálom ha valami javulást hoz.

Arnold Robbins,
➲ arnold@skeeve.com

A felfüggesztés lemezre azon a laptopon a ***BIOS*** egyik szolgáltatása. Ha a ***Linux*** telepítésekor töröltéd a mentési partíciót, akkor újra létre kell hozni. Egy másik ***ASUS*** tulajdonos hasznos tanácsai: ***linux.seindal.dk/item20.html***.

Amennyiben érdekel az ***ACPI*** beállításaid testre szabása, könnyebben megy a hibakeresés ha a megfelelő ***init scriptet*** stop paraméterrel meghívva leállítod az ***acpid*** démont, és a ***-d*** kapcsolóval az előtérben indítod el. Így láthatod, milyen ***ACPI*** események lépnek életbe és módosíthatod a megfelelő parancsfájlokat.

Don Marti,
➲ dmarti@ssc.com

A Daktronics hátrányos megkülönböztetést használ a linuxos vásárlókkal szemben

Miért választja a legtöbb alkatrészgyártó a ***Microsoft*** modellt a ***Linux*** modellel szemben? Miért nem fejleszt mindkettőre? A ***LED*** jelek jelentős fejlesztője, a ***Daktronics*** például kizárólag ***Microsoft*** termékeket használ a fejlesztőkészleteiben. A cégnek van ugyan protokoll útmutatója de semmiféle segítséget nem nyújt a Linux osztott könyvtár modelljéhez.

Én a ***linuxos*** modell szerint fejlesztek ezekhez a kijelzőkhöz, de ennél a munkánál is úgy tűnik a ***Microsoft*** fejlesztés felé billen a mérleg. Hogyan lehetne rávenni a gyártókat, hogy könyvtárakat készítsenek a linuxos fejlesztés meggyorsítására? Továbbra is folytatni fogom a projekt fejlesztését, mivel úgy érzem, hogy a Nyílt Forrású közösség valami használt látja az ilyesfajta kijelzők meghajtóinak. Ha tudnátok valamilyen információt adni hasonló témával foglalkozó emberekről vagy csoportról, szívesen venném.

Chuck Smith,
➲ chucksmith@viawest.net

Bár természetesen pártfogó csoportok is elérhetők, ilyen például az ***OSDL***, a legtöbb gyártó csak egyetlen dologból ért – a vásárlói nyomásból. Csatlakozz a kórushoz, lépj kapcsolatba a cég minden képviselőjével akivel csak tudsz és mondd el nekik, hogy szeretnéd ha felkarolnák a ***Linuxot***. A legtöbb gyártó már elkezdett dolgozni ezen a területen, de sokan nagy kihívásnak találják egy új operációs rendszer alatti fejlesztésbe belevágni, így erős ösztönzésre van szükségük. Türelemre és sok ismétlésre van szükség, hogy hangot adhassunk véle-

ményünknek. Bátorításul; az elkövetkező két évben egyre ritkább lesz az olyan gyártó, aki semmibe veszi a Linuxot.
Chad Robinson,
➲ chad@lucubration.com

Becsülendő, hogy egy projekten szeretnél dolgozni, de ha egy üzleti fejlesztőkészlet licenszéhez vagy kötve, elképzelhető, hogy nem adhatod ki a saját hasonló képességeket nyújtó nyílt forrású kódodat. Érdemes ellenőriztetni egy hozzáértő ügyvédi irodával például a ***rosenlaw.com***-on – valószínűleg jobban jársz, ha olyan projekten kezdesz dolgozni, ahol soha nem kattintottál a kínos ***„I Agree"*** gombra.
Don Marti,
➲ dmarti@ssc.com

Linux kapcsolat a régi Microsoft Exchange-hez?

Remélem létezik válasz erre a kérdésre; mindenfele kerestem a megoldást és szinte semmit sem találtam. Mostanában szeretnénk megjelentetni ***Java Desktop*** rendszereket, de rá kellett ébrednünk, hogy az ***Evolution*** nem támogatja az ***MS Exchange*** naptárazást és egyebeket. Megpróbáltuk működésre bírni, így végül a ***Ximian Connector*** segítségét vettük igénybe, csakhogy az viszont nem működik együtt az ***Exchange 5.5***-el. Biztosan van más is a ***Ximian***-on kívül. Mivel elég új fiú vagyok a ****nix*** területén fogalmam sincs mit lehetne helyette használni. Tudsz esetleg bármilyen más programot amely képes összekapcsolni ***Evolution*** e-mail ügyfeleket ***MS Exchange*** kiszolgálóval?
Monique Marais,
➲ monique.marais@alindigo.com

Olyan régi ***Exchange*** verziód van, amelyet már a ***Microsoft*** sem támogat tovább. Rengeteg megoldás létezik levelezésre és naptárazásra ***Linux*** alatt; vannak köztük ingyenesek és üzleti megoldások is. Vess egy pillantást a ***www.calendarhome.com/clink/web-calendar.html*** listájára ha ilyesmit keresel.
Felipe Barousse Boué,
➲ fbarousse@piensa.com

Előbb vagy utóbb úgyis frissítened kell azt a régi ***Microsoft Exchange*** programot, ha időt akarsz nyerni, esetleg meg lehet próbálni VNC ügyfeleket telepíteni az új Linux rendszerekre és ***VNC*** kiszolgálót a ***Microsoft*** oldalra, így a felhasználók elérhetik a régi alkalmazásokat. A témáról bővebben itt olvashatsz: ***www.linuxmafia.com/faq/Legacy_Microsoft/vnc-and-similar.html***.
Don Marti,
➲ dmarti@ssc.com

Point-of-Sale rendszer?

Egy jó ***POS*** rendszert keresek három alkalmazottat foglalkoztató kis kiállításhoz. ***Red Hat 9***-et használok és ingyenes vagy kedvező árú megoldást keresek.
Randy Freer,
➲ freers@charter.net

Végignéztem jó néhány szép ***POS*** rendszert, amelyek esetleg megfelelhetnek az igényeidnek. Nézz körül a ***www.linux-pos.org*** lapon, ez a lap a kiskereskedelmi ***Linux*** alkalmazásokkal foglalkozik. Itt rengeteg üzleti alkalmazásra szánt szabad programot találsz, többek közt ***POS*** rendszereket is.
Azonban tartsd szem előtt, hogy céged igényeitől függően, elképzelhető, hogy elég sok mindent be kell építeni: pénztárgépet, vonalkód leolvasó rendszert, címke nyomtatót és egyebeket. Ezért, nem rossz gondolat összehasonlítani a teljes birtoklási költséget (beleértve az idődet vagy valamilyen cég felkérését) az üzleti megoldásokkal, még ha ***Linuxról*** és ***Nyílt Forráskódú*** rendszerekről is van szó.
Az üzleti megoldással támogatás és kulcsrakész indulás jár, így nem kell magadnak összeilleszteni, felépíteni és kitalálni mindent.
Felipe Barousse Boué,
➲ fbarousse@piensa.com

Alternatív levelezési megoldás

Walter 2004 augusztusi Legjobb műszaki támogatás kérdésére válaszolva: Régen megoldottam már ezt a problémát, megtekintheted a ***www.trestle.com/linux/trestlemail/bye-trestlemail.html*** címen.

Igaz, ma már a ***Postfix*** kezeli a leveleimet így nincs szükség a ***TrestleMail***-re. Teljesen egyetértek ***Felipe*** válaszával: ameddig többcélú (multidrop) levelet használsz, mindig problémát fognak okozni a rosszul kezelt határesetek (levelező lista levél, bcc-k és egyebek). Ezt nem lehet megkerülni. Ugyanakkor, ez az egyszerű megoldás megbízható, és teljes vezérlést ad neked a teljes levélforgalom felett. Lehet, hogy éppen a ***Trestlemail*** az amire szükséged van. Nyugodtan szólj ha kérdésed akad.
Scott Bronson,
➲ *bronson@rinspin.com*

Linux Journal 2004. december, 128. szám

AEM: saját méretezhető eseménykezelő Linux alá

Tegyük képessé alkalmazásunkat rendszermag visszahívások regisztrálására.

Egy korábbi cikkünkben a telekommunikáció témakörében felvetettük egy saját, általános eseménymechanizmus szükségességét ***Linux*** alá. A ***Linux*** képességeinek javítását célzó legtöbb megoldás kiábrándított bennünket. Mások nem feleltek meg igényeinknek, hiszen egy kommunikációs szintű (carrier-grade) rendszernek komolyabb valós idejű követelményei vannak. A siker érdekében egy ilyen eseménymechanizmusnak szorosan kell kapcsolódnia az otthont adó operációs rendszerhez és ki kell tudnia használni annak képességeit a nagyobb teljesítmény érdekében.

A montreáli ***Open Systems*** laborban (***Ericsson Kutatóintézet***), 2001-ben indítottuk az általános megoldást kereső ***Asynchronous Event Mechanism (AEM)*** projektet. Az ***AEM*** segítségével az alkalmazás bizonyos eseményekhez visszahívható függvényeket adhat meg illetve regisztrálhat, az operációs rendszer pedig az események létrejöttekor aszinkron módon hajtja végre ezeket.

Az ***AEM*** a fejlesztéshez esemény elvű megközelítést kínál. Ezt egy természetes felhasználói csatolófelület bevezetésével értük el, ahol az eseménykezelők paraméterlistájában a végrehajtásukhoz szükséges és a rendszermagnak közvetlenül elküldendő valamennyi adat megtalálható.

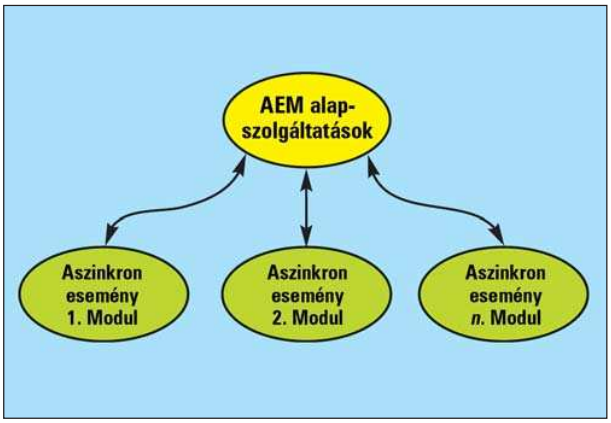
Az ***AEM*** másik mozgatórugója az a tény adta, hogy a többszálú rendszereken futó összetett megoszló alkalmazásokat a kezelő réteg miatt általában igen nehéz több rendszerre fejleszteni vagy azokra átírni. Az ***AEM*** nem pusztán a programfejlesztési idő csökkentését célozza meg, hanem a forráskód létrehozását is leegyszerűsíti, ezáltal megnövelve a különböző rendszerek közti a hordozhatóságot és meghosszabbítva a program életciklusát.

A projekt legnagyobb kihívása egy olyan rugalmas keretrendszer tervezése és kifejlesztése volt ahol a futó rendszerben eseménykezelő megoldásokat tudunk frissíteni és elhelyezni. Alapkövetelmény volt, hogy a rendszerkarbantartás a rendszer újraindítása nélkül is elvégezhető legyen. Az ***AEM*** moduláris felépítésének köszönhetően rendelkezik ezzel a képességgel.

Az ***AEM*** más figyelmeztető mechanizmusok mellett kiegészítő lehetőségként működik. Az egyik nagy előnye, hogy az eseményvezérlet kódot keverhetjük a többi, soros kóddal.

AEM: Szerkezeti áttekintés

Az ***AEM*** magmodulból és a köré épülő betölthető modulokból áll amelyek az adott eseményszolgáltatást nyújtják az alkalmazásoknak. Ilyen például a szoftveres időzítés és az aszinkron ***TCP/IP*** socket csatolófelület (*1. ábra*). Ez a rugalmas szerkezet lehetővé teszi, az AEM képességeinek tetszés szerinti bővítését.



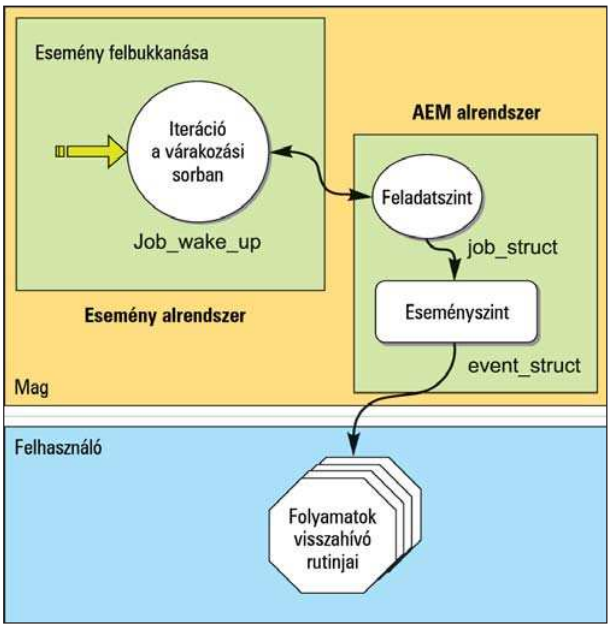
1. ábra Az AEM az alapvető eseményfunkciókat biztosító magmodulból és az alkalmazásoknak aszinkron eseményszolgáltatásokat nyújtó független rendszermag modulokból áll

A modulok által végrehajtott feladatokat száma korlátlan, ugyanis saját, független pszeudo rendszerhívás készletüket adják át az alkalmazásnak. Tulajdonképpen így akár két különböző modul is szolgálhat azonos feladatokat. Érdekes módon éppen ez teszi lehetővé, hogy a továbbfejlesztett változatot a többi alkalmazás zavarása nélkül a rendszerbe illesszük – azok ugyanis továbbra is a régi verziót használják. A felépítés lehetővé teszi, hogy az alkalmazások igénye szerint töltsük be az ***AEM*** modulokat vagy futásidőben fejlesszük a modulokat.

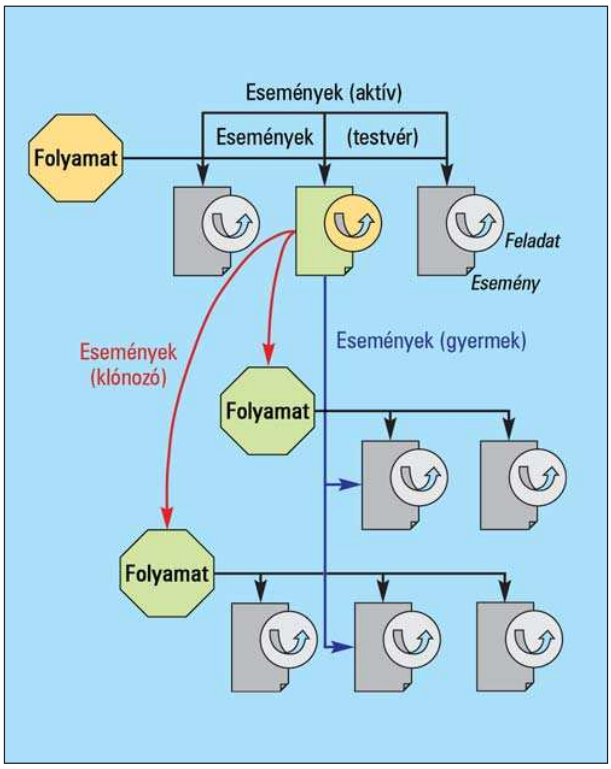
Az ilyesfajta rugalmasság előfeltétele, hogy a rendszermag stratégiai pontján eseményaktiválási pontokat helyezzünk el (*2. ábra*). Minden aktiválási pont egy-egy eseményeket indító AEM sornak felel meg. A következő részben az ***AEM*** belső működését részletezzük.

AEM: Belső szerkezet

Az aszinkronitás elve akkor okoz komoly gondot, amikor a program végrehajtás fő folyama az eseménykezelők alkalmazásakor figyelmeztetés nélkül megszakad. A bemenetei kérelmeket a korábbi bemeneti állapot nélkül kell kezelni.



2. ábra Az AEM esemény aktiválási és alkalmazás-értesítési szerkezete. Az alvó munkafolyamatokat tartalmazó esemény-várakozási sorokat folyamatosan pásztázzuk. Az adott esemény bekövetkeztekor valamennyi érintett feladat felébred. Ezt követően valamennyi ide tartozó folyamathoz rendelt eseményt elindítjuk.



3. ábra A folyamatok és eseménylistáik közötti kapcsolat

Ezeket közvetlenül a rendszermag vagy a megszakítás-kezelő küldi és fogadásukkor nem feltételezhetünk semmilyen sorrendet. Ez a helyzet bizonyos alkalmazásoknál problémákat okozhat, különösen amelyek *TCP/IP* alapon működnek, hiszen ezek működéséhez fontos ismerni a tranzakció állapotát.

Az *AEM* három rétegű szerkezetét az eseménykezelést végző pseudo-rendszer hívások, az esemény sorosítást kezelő és a felhasználói visszahívások érdekében környezetinformációkat tároló folyamatonkénti `event_struct` struktúrák valamint az eseményaktiválásért felelős eseményenkénti `job_struct` szerkezetek alkotják.

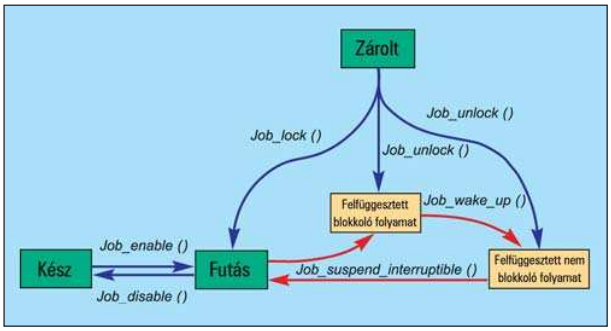
Események

Az *AEM* szemszögéből nézve, az esemény olyan rendszer inger, amely kiváltja az esemény kezelője, azaz a végrehajtó ügynök létrehozását. Ehhez az esemény regisztrációjakor létrehozott `event_struct` struktúra nyújt segítséget, amely egy esemény kezelő végrehajtásához szükséges környezetet tartalmazza. A legfontosabb mezői a felhasználói tér eseménykezelőire hivatkozó mutatók, azok konstruktorai és destruktorai valamint kapcsolatuk más eseményekkel (listaesemények, gyermek események és aktív események – lásd a *3. ábrát*). Folyamatonként annyi regisztrált eseménykezelőt készíthetünk amennyi csak szükséges. Amikor az eseményt észleljük, azt aktiválódásnak nevezzük és hamarosan meghívjuk a felhasználó által megadott kezelő függvényt.

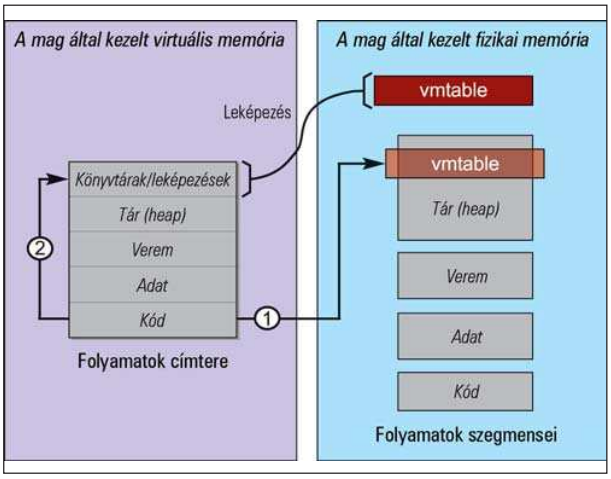
Az események érkezési sorrendben jegyződnek fel a rendszerben. Innentől fogva már az eseménykezelő konstruktorának feladata az adatokat helyesen kezelni és bármiféle sorrendiség feltételezése nélkül sorosítani az egyes folyamatokhoz tartozó eseményeket.

Bizonyos folyamatesemények aktívak és aktív eseménylistához csatlakoznak. Aktiváláskor az esemény folyamatot hozhat létre. Ezeket az eseményeket klónozóknak nevezik. Az események és az általuk létrehozott folyamatok közti kapcsolatot belsőleg tároljuk. A harmadik ábrán megfigyelhetjük a legfelső folyamat által regisztrált eseményt amely két új folyamatot készített. Ezek az eseményhez kapcsolva maradnak és saját eseménylistával rendelkeznek.

Az eseménykezelőket az esemény regisztrálásakor használjuk és a felhasználói szinten kell megvalósítani őket. Megadhatják saját, előre meghatározott számú paramétereiket, hogy az esemény adatok közvetlenül a felhasználói térben futó folyamathoz kerülhessenek. Ezt a műveletet az esemény konstruktorok és destruktorok végzik amelyeket közvetlenül a kezelők előtt és után hívunk meg.



4. ábra Periodikus és reaktív munkafolyamatok állapotátmeneti grafikonja



5. ábra A vmtable felépítése

Az eseménykezelők ugyanabban a környezetben futnak mint a meghívó folyamat. A mechanizmus biztonságos és többszörösen bejárható (re-entrant); a jelenlegi végrehajtás mentésre kerül, végül helyreállítjuk a megszakítás előtti állapotot.

A regisztráció során minden eseményhez valamilyen fontossági szintet rendelünk amely a megfelel annak a sebességnek mellyel az alkalmazás az adott figyelmeztetést fogadni kívánja. Ugyanazon eseményre két különböző fontossági szinten is jelentkezhetünk.

Más valós idejű figyelmeztetési megoldások, például a ***POSIX*** jelzések valós idejű kiterjesztése, az ütemezési döntés során nem veszik figyelemebe a fontossági szinteket. Pedig ez fontos, hiszen így a nagy fontosságú eseményeket fogadó folyamatokat a többi folyamat előtt tudjuk végrehajtani. ***AEM*** alatt az esemény megjelenése a fontossági sorrendnek megfelelően váltja ki a kezelő végrehajtását. Bizonyos szemszögből, az eseménykezelő folyamatnak tekinthető, hiszen van végrehajtási környezete. A folyamat fontosságok dinamikus változtatása akkor okoz igazán komoly gondot, amikor az események beérkezési sűrűsége nagy, hiszen a fontossági szintek ugyanilyen ütemben változnak. Ezt a problémát az esemény fontosságok összességéből számított dinamikus valós idejű érték bevezetésével oldottuk meg. Ez az érték a ***Linux*** ütemező befolyásolása nélkül módosítja az ütemezési döntéseket az alkalmazásoknak pedig valós idejű érzékenységet ad.

Munkafolyamatok

A ***munkafolyamat*** (***job***) egy új rendszermag fogalom amelyet az események figyelmeztető folyamatokat megelőző kiszolgálása érdekében hoztak létre. Ez nem folyamat, bár mind a kettő a végrehajtható elem elképzelésen alapul. A munkafolyamatok által végzett egyik tipikus feladat, hogy elhelyezi magát egy várakozási sorban és itt marad, míg valami fel nem ébreszti. Ekkor gyorsan elvégez valami hasznos dolgot, például ellenőrzi az adat hitelességét vagy elérhetőségét a felhasználói esemény meghívása előtt, majd ismét elalszik. A munkafolyamat azt is garantálja, hogy mialatt valamilyen erőforrást elér, más munkafolyamatok azt nem érhetik el. Egy folyamathoz több munkát is rendelhetünk, de eseményenként csak egy munkafolyamatunk lehet.

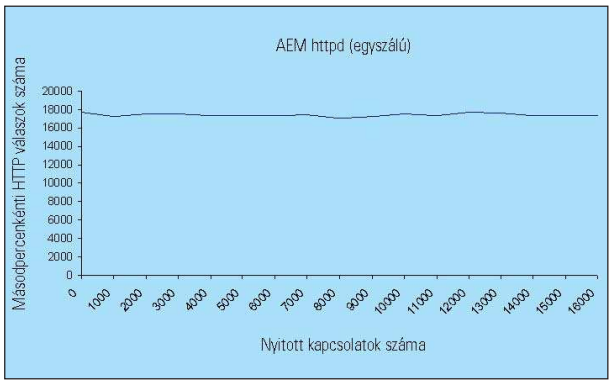
Ez a rendszermag és a felhasználói folyamat közötti elvonatkoztatási réteg nélkülözhetetlen. Nélküle igen nehéz lenne biztosítani a konzisztens adat elérhetőség ellenőrzést vagy a kezelő végrehajtása során azonos eseményekből többet felhalmozni. Amennyiben valami probléma adódik, a folyamat a felhasználói térben pazarolja az esemény kezelésére szánt időt. Az, hogy összevonunk-e több figyelmeztetést általában eseménytől függő és az aktiválás előtt kell végrehajtanunk.

A munkafolyamatok általános megvalósításánál figyelembe kell vennünk a megszakításokat, hogy az esemény bekövetkezése és a folyamat figyelmeztetése közt lehetőleg kevés idő teljen el. Célunk, hogy a folyamatok oldalán végezzük el a műveleteket miközben egy megszakításkezelő és egy rendszermag szál képességeit biztosítjuk, mindezt anélkül, hogy a teljes végrehajtási környezetet magunkkal hurcolnánk.

Két típusú munkafolyamatot alkalmaztunk, a periodikus és a reaktív munkafolyamatokat. A periodikus munkafolyamatokat adott időnként, a reaktív munkafolyamatokat szórványosan, az események észlelésekor hajtjuk végre. A munkafolyamatokat saját ütemező ütemezi. A valós idejű ütemezés elméletet szerint mind a két fajta munkafolyamatot azonos ütemezőnek kell ütemeznie (lásd ***Jeffay és társai*** publikációját a hálózati forrásokban). A mi környezetünkben a munkafolyamat nem preemptív feladat. definíció szerint a feladatoknak nincs megadott határidejűk, bár alacsony szintű jellegükből következően végrehajtási idejük közvetetten mégis kötött. Ez a feltételezés leegyszerűsíti a megvalósítást. Esetünkben előfeltétel, hogy a reagáló munkafolyamatok két meghívás között elhanyagolható idő alatt hajtódjanak végre hogy helyt álljanak folyamátviteli helyzetekben is.

A mi megoldásunk a reagáló és a periodikus esetben is eltérő, hogy a szórványos események esetében jobb átviteli eredményt érjünk el.

A periodikus munkafolyamatokat a ***feladatkiosztó*** (***job dispatcher***) kezeli, a reagáló feladatok ugyanakkor teljesítmény megfontolások miatt saját maguk állítják állapotukat. A *4. ábra* mutatja be a munkafolyamat állapotváltozásokat és az egyik állapotból másikra való áttéréshez használt függvényeket.



6. ábra AEMhttpd, egy szálú HTTP kiszolgáló amelyet az AEM belső megvalósításának skálázhatósági tesztjénél használtunk. Minden mintában 100 aktív kapcsolatot használtunk.

Amint a munkafolyamat aktivizálta a megfelelő eseményt, vagy aszinkron módon végrehajtunk egy folyamatot vagy a felhasználói program jelenlegi futását irányítjuk át a kód másik részére. A felhasználó regisztráláskor dönti el melyik változatot szeretné alkalmazni.

Folyamat aszinkron végrehajtása

Az eseménykezelők végrehajtásához megtörhetjük a fő szál végrehajtását vagy létrehozhatunk egy új folyamatot amelyben az esemény párhuzamosan futhat le. Az esemény mindkét esetben átlátszóan és aszinkron módon fut, és közvetlenül önállóan felügyeli saját futó példányait. Ennek a megoldásnak van néhány fontos következménye ugyanis az alkalmazásnak nem kell idő előtt megőriznie vagy elhasználnia a rendszer erőforrásait. Próbaképpen az ***AEM*** teljesítményméréséhez készítettünk egy egyszerű ***HTTP*** kiszolgálót, amely teljesen egyszálú és az ilyen típusú kiszolgálókhoz képest meglehetősen jó teljesítményű. Cikkünk végén mutatjuk be.

Néha előfordulhatnak olyan helyzetek, amikor az esemény megválaszolásához kénytelenek vagyunk új folyamatot létrehozni. Sajnos a folyamat dinamikus létrehozása elég erőforrás-pazarló; ez alatt az időszak alatt sem az új sem a szülő folyamat nem képes új kérelmeket kezelni. Bizonyos kritikus helyzetekben, például a rendszermemória betelésekor elképzelhető, hogy nincs erőforrás ilyen célra. Ez azért komoly probléma mert a vészhelyzetben szükségünk lehet új folyamatok létrehozására amelyek figyelmesen zárják le a rendszert vagy hívásátadó procedúrákat hajtanak végre.

A probléma orvoslására bevezettük egy új, kapszulának nevezett modellt. A kapszula olyan előre beállított folyamat struktúra amely az egyéb üres kapszulákat tároló tárolóban található. Amikor a folyamat új végrehajtási környezetet akar létrehozni, a kapszulát kicsatoljuk a tárolóból és a jelenlegi folyamat néhány paraméterével alaphelyzetbe állítjuk.

A esemény regisztráláskor külön jelek mutatják hogy a kezelőt a folyamaton belül kell-e végrehajtani. Ha nem adunk meg paramétert vagy 0-t adunk meg akkor a kezelőt a folyamat futásának megtörésével kell végrehajtani. A jelek a következők:

- EFV_FORK: a folyamatot a `fork()` hívással azonos formában kell elvégezni.
- EVF_CAPSULE: a folyamatot a kapszula tárolóból vesszük.
- EVF_NOCLDWAIT: ennek a jelnek a SIG_NOCLDWAIT jelzéssel azonos jelentése van. Amennyiben létezik gyerek folyamat, új szülőt kap a kapszulakezelő szál személyében.
- EVF_KEEPALIVE: megakadályozza a folyamat/kapszula kilépését azáltal, hogy belép egy rendszermag ciklusba (a while(1) utasításhoz hasonlóan) csak felhasználói térben.

Memóriakezelés

Az esemény vezérelt rendszerekben igen fontos kérdés a memóriakezelés, hiszen az eseményeket közvetlenül a rendszermagtól származó alkalmazástérben található visszahívási függvények kezelik. A legegyszerűbb megoldás az lenne, ha csak akkor foglalnánk le a memóriát amikor a folyamat jelentkezik az eseményre. Ugyanakkor gondoljunk bele, mi történik ha rengeteg eseményt kell regisztrálni, vagy ha olyan folyamataink vannak amelyeket újra kell indítani, új eseményeket kell hozzájuk adni, esetleg eseményeket kell eltávolítanunk. Amennyiben az eseménykezelésben hiba történik, a rendszerintegritás sérül. Kevésbé veszélyes az operációs rendszer számára, ha az ilyen erőforrásokat maga kezeli és a memóriát a folyamatok igénylése szerint dinamikusan foglalja le.

Teljesítmény szempontból ezt a memóriát egy előre lefoglalt tárhelyből kell kiosztanunk, hogy elkerüljük a memória feldarabolódását és fenntartsuk a valós idejű jellemzőket. Az általános célú memória foglaló eljárások mint a `glibc malloc()` függvényei, bár általános célokra jól is használhatóak, nem igazán felelnek meg ezeknek a követelményeknek.

Néhány adattípust, például az egészeket, könnyedén át tudjuk adni a felhasználói térbe, az olyan összetettebb adattípusok viszont mint a karaktersorozatok, már külön megvalósítást igényelnek. A folyamatok memóriakezelését a ***glibc*** könyvtár végzi. A dolgok bonyolódnak ha a rendszermagtól szeretnénk memóriát foglalni, mivel arra is ügyelnünk kell, hogy az újonnan lefoglalt memóriaterület jó helyen legyen és a folyamat térben helyezkedjen el.

A felhasználói térből elérhető egyszerű és hatékony memóriafoglalás egyelőre hiányzik a rendszermagból, pedig szükség lenne rá.

Az ***AEM vmtable*** megoldása ezt a memóriafoglalási hiányosságot próbálja orvosolni. A ***„binary buddy allocator”*** egyik változatát valósítja meg (***Knuth*** után, lásd a forrásokat) a felhasználói folyamat memória tárán keresztül. Ennek segítségével majdnem bármilyen előre nem tervezett méretű és típusú adat kezelése megoldható. Kritikus memóriahiány esetén a eseménykezelőkön keresztül a felhasználóra bízza döntést. E képesség segítségével végső megoldásként visszanyúlhatunk glibc változathoz amennyiben valami baj történne.

Az *AEM* úgy készült, hogy amennyiben van még szabad blokk, konstans idő alatt adjon vissza érvényes mutatót. A telekommunikációs alkalmazások esetében amelyek valószínűleg azonos méretet igényelnek azonos alkalmazástípus esetén, általában éppen ilyesmire van szükség. A rövid idő alatt érkező és különböző méretű kérelmek által okozott memória töredezettséget is meg szerettük volna előzni.
A *vmtable* egy érdekes kiterjesztéssel is rendelkezik: felhasználói térből készíthetünk vele alaphelyzetbe állító függvényeket az eszközmeghajtókhoz. Ezt egy mutató használatával tehetjük meg, amelyet az *AEM* alrendszer segítségével a rendszermagtól foglaltunk és visszahívott függvény által adtuk át a felhasználói folyamatnak. A függvény visszatérésekor a terület visszakerül a rendszermaghoz. A visszahívott függvények nem csak események kezelésére jók, hanem nagyon jól használhatóak a rendszermaggal történő biztonságos kapcsolattartásra is.
Ebben a felállásban minden felhasználói folyamathoz rendelünk egy memóriaterületet, amit *vmtable*-nek nevezünk. Az elágaztatott (*forked*) folyamatok és kapszulák *vmtable*-je szülőfolyamatuk *vmtable* táblája alapján automatikusan öröklődik. Két különböző stratégiát alkalmazunk:

1. `VMT_UZONE`: a foglalást a folyamat heap szegmenségben hajtjuk végre. Ez ugyan gyors elérést tesz lehetővé, de a felhasználói folyamat memóriahelyét fogyasztja.

2. `VMT_VZONE`: A foglalást a rendszermag címterében végezzük, majd belapozzuk a folyamat címterébe. Ezzel minimális memóriafogyasztást érhetünk el, de a laphibák kezelése miatt a folyamat több időt vesz igénybe.

Mindkét eljárásnak helyzettől függően vannak előnyei és hátrányai. A kívánt stratégiát az alkalmazás indításakor, futásidőben választhatjuk ki. Az *5. ábra* a vmtable szerkezetét mutatja be.
A jelenlegi megoldásban a *vmtable* alrendszer közvetlenül fizikai lapokat foglal le. Ez azért van így, hogy a foglalások biztosan folyamatosak legyenek és a memóriát biztonságosan használhassák az I/O műveletek. A jövőben hálózati teljesítmény terén is szeretnénk közvetlen ki-/bemenetere felhasználni a *vmtable* rendszert.
A *vmtable* egyszerű és könnyen kezelhető felületet nyújt az *AEM* felhasználóknak és modulfejlesztőknek, a memóriafoglalás összetettségét a rendszermag térbe rejtve. Az átlátszóság kedvéért az *AEM* magmodulba egyszerű memóriafoglalási és felszabadítási rutinokat építettünk. Ez a csatolófelület nagyban leegyszerűsíti a modulok karbantartását és átvitelüket a különféle *Linux* rendszermag kiadásokra.

Skálázhatóság

Végeztünk egy tesztet, hogy megtudjuk miképpen viselkedik az *AEM* két távoli folyamat között zajló egyszerű adatcsere közben. A teszt fő célja annak kiderítése volt, hogy a eseménykezelő környezetváltása miatt fellépő időveszteség nem okoz-e teljesítmény problémákat. Mostanában elvégeztünk egy teljesítménytesztet is amely az *AEM* skálázhatóságát tette próbára valamint ellenőrizte az AEM magvát alkotó munkafolyamatok és várakozási sorok belső megvalósítását.
Könnyen összehasonlítható számokat keresve, úgy döntöttünk, hogy egy már létező webkiszolgálót keresünk és azt alakítjuk az *AEM* csatolófelületéhez. Az *AEMhttpd* egyszerű, egy szálon futó *HTTP* kiszolgáló (lásd a forrásokat.)
Az egyszálú kiszolgáló teljes egészében a folyamat fő végrehajtási szálában fut. Azaz nem hozunk létre sem felhasználói sem rendszermag szálakat a *HTTP* kérelmek kezelésére. A méréseket ilyen típusú kiszolgálóval végeztük és inkább a megvalósításra semmint a kiszolgáló tényleges teljesítményére koncentráltunk.
A *6. ábrán* bemutatott példán 100 aktív tranzakciót futtattunk. Minden tranzakcióban megnöveltük a nyitott kapcsolatok számát ezzel megnövelve a munkafolyamatok számát a socket kapcsolat várakozási sorában. A `select()`-en alapuló hagyományos megoldásban ez megnövelte volna a kérelmekre adott válaszidőt, hiszen a leírókat a rutin sorban nézné végig. Az *AEM* alkalmazásával azonban csak az aktív munkafolyamatok (azaz a kész adatot tartalmazó socketek) hajtják végre a megfelelő eseménykezelőiket. Ezzel a módszerrel az *AEM* képes általános és skálázható megoldást nyújtani.

Összefoglalás

Az iparban széles körben használják a *Linuxot*, amely büszkén nevezheti magát vállalati szintű megoldások választott operációs rendszerének. Nem jár messze attól sem, hogy a következő generációs IP-alapú telefonszolgáltatások választott operációs rendszerévé váljon. A *Linux* képességeit ma már széles körben elismerik, de a skálázhatóságot, teljesítményt és megbízhatóságot váró, következő generációs szolgáltatókat a rendszermag szinten végezett fejlesztések fogják leginkább vonzani.
Az *AEM* erőteljes próbálkozás az aszinkron folyamatértesítés és esemény adatkiegészítés terén. Segítségével kihasználhatjuk az esemény vezérelt programozás előnyeit, amely biztonságos alkalmazásfejlesztést tesz lehetővé. Ezen felül az *AEM* olyan részeket is tartalmaz, melyek célja az alkalmazás megbízhatóságának és hordozhatóságának növelése könnyen kezelhető felülete révén.

Köszönetnyilvánítás

Köszönet az *Open Systems Lab* összes kutatójának az *Ericsson*-tól pedig *Lars Hennert*-nek hasznos tanácsaikért, valamint az *Ericsson Research*-nek, hogy jóváhagyta e cikk megjelenését.

Linux Journal 2004. november, 127. szám

Frédéric Rossi (Frederic.Rossi@ericsson.ca)
Az Ericsson Research Open Systems Lab fejlesztője Kanadában, Montreál városában. Az AEM atyja és a fejlesztés vezetője.

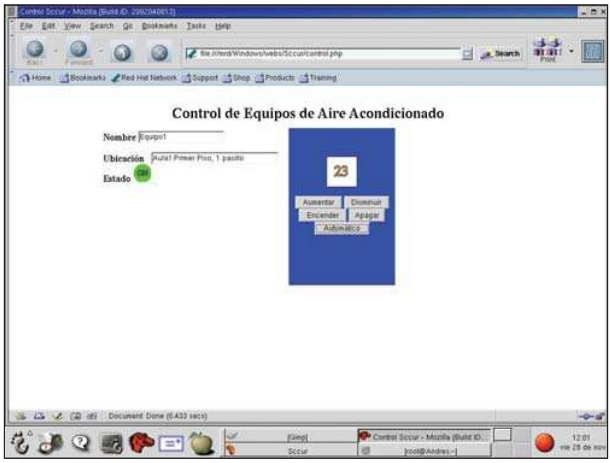
Linux és RTAI Automaták építéséhez

Ez a könnyen telepíthető, web-alapú vezérlőrendszer szabványos telefon vezetékeket használ és bármely távirányítóval rendelkező egységet képes kezelni.

Cikkünkben egy olyan, az épületekben található különféle légkondicionáló berendezések központosított működtetését végző vezérlőrendszer fejlesztéséről és tervezéséről írunk, amelyet ***Linux*** illetve valós idejű ***Linux*** alkalmazás csatolófelület (***RTAI***) segítségével hoztunk létre. Az épületben elszórtan elhelyezkedő légkondicionálók saját infravörös távirányítóval rendelkeznek. Célunk az volt, hogy a légkondicionálók vezérlését központosított, számítógéppel vezérelt megoldással váltsuk fel, amely képes ki- és bekapcsolni a készülékeket, valamint a kívánt hőmérsékletet és fordulatszámot beállítani. A projekt ötlete akkor merült fel, amikor a helyi egyetemnek, szerény keretből megvalósítható, központosított és rugalmas a légkondicionáló vezérlési megoldásra volt szüksége. Hasonló célokra léteznek üzleti alkatrészek és programok, de általában túlságosan költségesek és gyártófüggőek.
Projektünk alkatrész megvalósítása egy ***Linuxot*** futtató központi vezérlő-számítógépből és a hozzá csatlakoztatott ***RS-485***-ös mikrovezérlő hálózatból áll. A mikrovezérlők a közeli felszereléseket irányító infravörös jelekkel parancsokat tudnak küldeni a megfelelő légkondicionálónak.
A rendszer programterve két valós idejű feladatot (a fő vezérlő feladatot és a ***RS-485*** hálózat vezérlő feladatot) valamint további, nem valós idejű feladatokat (a webkiszolgálót és az adatbázist) tartalmaz. A felhasználói felületért a webkiszolgáló felel így az egyetem számítógép-hálózatán bármely böngészőről elérhető. Adataink tárolására a ***PostgreSQL*** adatbázist választottuk. Megvalósításunk olyan alacsony költségű megoldás, amely igény szerint rugalmasan bővíthető. Ezen kívül gyártófüggetlen és bármely légkondicionálóval működik, amely képes infravörös távirányítót kezelni. Valamennyi légkondicionáló önállóan működik, saját hőmérséklet-vezérlő rendszerével. Az eszközök működésének ellenőrzésére a hálózat valamennyi mikrovezérlőjét hőérzékelővel láttuk el amely regisztrálja az aktuális teremhőmérsékletet és jelenti a központi számítógépnek.

Felhasználói felület

A teljes felhasználói felület weblap alapú. A kezdőlapon az összes légkondicionáló állapotának összefoglalóját soroljuk fel. A megjelenített adatok között találjuk a légkondicionálót azonosító szöveget, az épületben elfoglalt helyét, a jelenlegi teremhőmérsékletet valamint, hogy van-e az eszköznek



1. ábra A vezérlőfelületről a hitelesített felhasználó beindíthatja a légkondicionálót, új hőmérsékletet állíthat be vagy kikapcsolhatja azt

valamilyen előreprogramozott működési módszere, amely alapján éppen dolgozik. Erről a lapról valamennyi légkondicionáló külön kezelőfelületére mutat egy-egy hivatkozás. Mielőtt erre a lapra juthatnánk, a rendszer felhasználónevet és jelszót kér. Ezen a szinten lehetőségünk nyílik utasításokat adni a rendszernek, közvetlenül vezérelni a légkondicionálót illetve megváltoztatni vagy létrehozni az automatikus működés valamilyen programját (*1. ábra*).
A rendszer legérdekesebb része a programozott működés. Például a rendszer képes az órák kezdete előtt minden nap automatikusan bekapcsolni a a légkondicionálókat, minden szobában előre megadott hőmérsékletértékekkel. Aztán este, amikor már nincs élet az épületben vagy az adott teremben, a rendszer lekapcsolja a légkondicionálókat.

Alkatrész felépítés

Az alkatrészkészlet a központi számítógépből és a légkondicionáló gépeket irányító mikrovezérlőkből áll. Valamennyi mikrovezérlőt ***RS-485*** kéteres hálózatra kapcsoltuk. Megvalósításunkban az ***Intel*** származék ***ATMEL AT89C2051***-es jelzésű vezérlőt alkalmaztuk. Húsz lábú DIP csomagban kapható, 128 byte adat memóriával és 2KB ROM kód hellyel, egy aszinkron soros kapuval és 14 független digitális I/O kapuval rendelkezik.

A mikorvezérlő lapját és elemeit a 2. *ábrán* mutatjuk be. Az infravörös jeleket a mikrovezérlőben található program hozza létre az egyik digitális kimeneti kapu segítségével. A soros kapu kapcsolja a mikrovezérlőt az ***RS-485*** hálózatunkhoz. Valamennyi mikrovezérlő lapot felszereltük a ***Dallas Semiconductor DS1620*** típusú hőmérőjével. A mikrovezérlő 3 eres, digitális szinkron soros felületen kommunikál a hőérzékelővel. A mikrovezérlőben nincs alkatrész szintű támogatás a szinkron soros felület kezeléséhez, ezért a vezérlést programozottan oldottuk meg a normál I/O kapukon keresztül.

Alkalmazásunkban azért esett az ***RS-485*** hálózatra a választás, mert könnyen telepíthető, olcsón megvalósítható és könnyen összekapcsolható vele hasznos mennyiségű csomópont. A csomópontokat mindössze egyetlen 3-as kategóriájú telefon osztályú kábel köti össze. Az alkatrész meghajtó korlátai miatt a lehetséges csomópontok száma 32, de ezt a számot könnyedén megnövelhetjük hálózati ismétlők segítségével. A vezérlő-számítógép és az első ismétlő, vagy két ismétlő közötti maximális kábelhossz 1200 méter.

A fizikai kábel elérést mester-szolga elven vezéreljük. A ***Linuxot*** futtató számítógép a mester, amely előre meghatározott sűrűséggel kérdezi le a csomópontokat. A mester minden lekérdezéskor parancsot küld a csomópontnak; a lekérdezett csomópont adattal válaszol a mesternek avagy üres csomagot küld, jelezve, hogy a csomópont élő. Az ilyesfajta élérési megoldás hátrányára akkor derül fény, amikor a mester áll le. Ilyenkor ugyanis a teljes hálózat is csődöt mond.

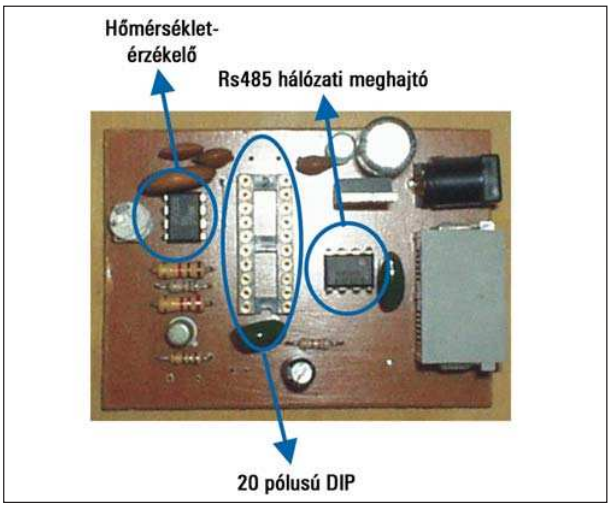
A mikrovezérlők korlátozott számát figyelembe véve a 9. bit protokoll segítségével állapíthatjuk meg, hogy a hálózaton átküldött csomag az adott vezérlőhöz tartozik-e. A hálózaton átküldött minden egyes bájthoz tartozik egy 9. bit is. Ez a kiegészítő bit kizárólag a csomag cél cím esetében lesz magas rétékű. A mikrovezérlő ***UART (universal asynchronous receiver and transmitter)*** alapértelmezés szerint csak akkor vált ki megszakítást, ha a fogadott érték 9. bitje magas. A megszakítás szolgáltatás rutinja ezután a fogadott bájtot összehasonlítja a csomópont címével. Amennyiben megegyeznek, a megszakítás átprogramozza az ***UART***-ot, hogy a 9. bit állapotától függetlenül, valamennyi bájtot fogadja a csomag végéig. Amennyiben a cél címe nem egyezik meg a csomópont címével a megszakítás rutin visszatér.

A központi számítógép ***UART***-ja, lévén PC alkatrész, nem támogatja közvetlenül a 9. bit protokollt. E hátrányt a meghajtó a paritás bit segítségével szimulálva próbálja meg kiküszöbölni. A bájt átvitele előtt a meghajtó úgy állítja be a paritást, hogy a cím bájtok esetében egy legyen a többi bájt esetében pedig nulla.

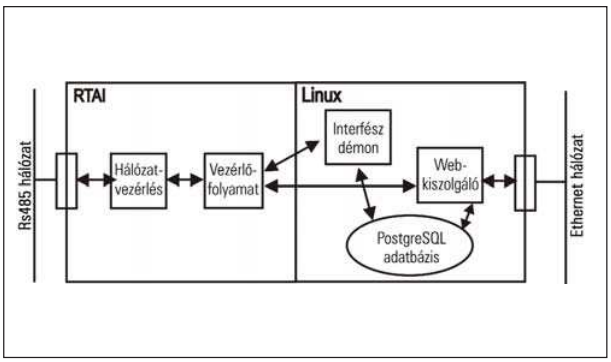
A 3. *ábra* a rendszer feladatait és a köztük élő kommunikációs vonalakat mutatja be.

Valós idejű feladatok

A fő vezérlő feladat, a a hálózati elérést vezérlő feladat és az ***RS-485*** hálózat fizikai rétegének program meghajtója valós időben futó feladatok. Az ***RTAI***-hoz kifejlesztettünk egy ***RS-485*** meghajtóját. A meghajtó az alkalmazásban felhasznált és korábban bemutatott 9. bit protokoll kivételével hasonló a többi soros meghajtóhoz.



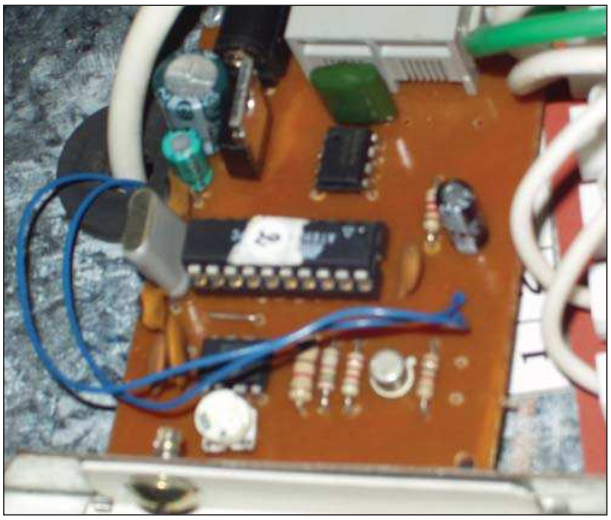
2. ábra A mikrovezérlő lapja a mikrovezérlő eltávolítása után



3. ábra A rendszerfeladatok közötti kommunikáció

A másik valós idejű feladat a hálózati elérést vezérlő feladat, mely meghatározott időnként csomagokat küld az egyes hálózati csomópontoknak. A csomag lehet az infravörös jelet bekapcsoló parancs vagy lekérdezés amivel kideríthetjük, hogy a csomópont aktív-e, valamint az aktuális teremhőmérsékletet lekérdező parancs. Az első két csomagtípus esetében a csomópont visszaigazolással, az utolsó esetében a teremhőmérséklettel válaszol. A fő vezérlő folyamat valamennyi csomópont aktuális állapotát ismeri, és értesíti a felhasználói felületet ha valamelyik csomópont meghibásodott.

A fővezérlő feladat, az adatbázisban tárolt, előre programozott információk alapján működteti az épület légkondicionáló berendezéseit. A faladat képes parancsokat fogadni a felhasználói felületről, amelyek két ***RT-FIFO*** segítségével felülbírálhatják az előreprogramozott beállításokat. Az ***RT-FIFO*** folyamatközi kommunikációs rutinok feladata a valós idejű és normál ***Linuxos*** feladatok párbeszédének lehetővé tétele. A ***PostgreSQL*** adatbázissal való kommunikációhoz Linux démont kellett kifejleszteni. A démon két másik ***RT-FIFO*** csatornán keresztül beszélget a fővezérlő folyamattal. A démon másik fontos feladata a rendszer időt és dátumot elküldeni a fővezérlőnek, ugyanis ennek kiolvasására nem létezik megoldás ***RTAI*** alatt.



4. ábra A teljes mikorvezérlő lap a telepítés után

A kifejlesztett rendszer parancsokat küld a légkondicionálóknak, így nincs szükség helyi távirányítókra. A légkondicionálók hőszabályzó rendszerébe nem avatkoztunk bele és nem változtattuk meg a belső kapcsolási sémákat sem. Minden légkondicionáló saját beépített hőszabályzó rendszerét használja, a mikrovezérlőkbe épített hőérzékelők pedig ellenőrzik, hogy az eszköz helyesen működik-e. A *4. ábra* a telepített mikrovezérlőt mutatja be.

Linux feladatok

A ***Linux*** feladatok célja a felhasználó felület megjelenítése a fő adatforrásként használt ***PostgreSQL*** adatbázis futtatásával a webkiszolgálón keresztül. Mint korábban leírtuk, a másik Linux oldali feladat a démon, amelyet a ***RTAI*** fővezérlő feladat használ a rendszer idő/dátum és az adatbázis eléréséhez.
A felhasználói felület egyszerű. Az első lapon az egyes légkondicionálók aktuális állapotát láthatjuk. Ezt a lapot minden felhasználó elérheti. A program megváltoztatásához vagy egy adott légkondicionáló utasításához a rendszer felhasználói nevet és jelszót kér. Az adatbázisból lekért információ dinamikus megjelenítéséhez a ***PHP*** nyelvet használtuk. A rendszer a ***PostgreSQL*** adatbázisban tárolja az általános információkat (***BTU***, elhelyezés, márka, mikrovezérlő hálózati azonosító) programozott műveleteket és az egyes kondicionálók működtetéséhez használt infra parancsokat.

IRC parancsfelület

A rendszer fontos eleme az a modul, amely a légkondicionáló távirányítójának jeleit olvassa be és a megfelelő eszközhöz rendelve az adatbázisban tárolja az kapott azokat, hogy később a hálózatba kötött mikrovezérlőkkel a jeleket újra létre lehessen hozni. Ezt a modult csak olyankor használjuk amikor egy új márkájú és/vagy vagy eltérő parancskészletű légkondicionálót veszünk fel a rendszerbe.
A modul két részből áll: az első az infra jeleket beolvasó valós idejű feladat. A hálózati forrásokban felsorolt ***LIRC Projekt*** valamint a ***Ripoll and Acosta*** tanulmány részletes információkat tartalmaz az infravörös távirányítókról. Ugyanitt normál ***Linux*** és ***RT-Linux*** rendszerekhez készült példaalkalmazásokat is találunk. A modul másik feladata a ***Linuxon*** futó felhasználói felület. A két feladat ***RT-FIFO*** kapcsolaton keresztül kommunikál.
A mikrovezérlőkben használható kis mennyiségű RAM és a hosszú infravörös jelhossz miatt, a program fontos funkciója, hogy segít a felhasználónak megszerezni a különféle infra távirányítók gombokhoz vagy gombkombinációkhoz rendelt ismétlődő parancsokat. Ezeket a mintákat aztán a mikrovezérlő belső programjába (***firmware***) beégetve újra elő tudjuk állítani az eszközt vezérlő jelet. Például amennyiben tíz különféle minta létezik, a megfelelő mikrovezérlőhöz a hálózaton keresztül küldött parancs a következő lesz: ismételd meg az egyes számú jelet tízszer, aztán a kettest háromszor és így tovább, míg a teljes parancsot meg nem kapjuk. A technika előnye hogy kevesebb erőforrást használ a jel ismételt előállításához. Hátránya, hogy valahányszor új légkondicionálót vásárolunk, a mikrovezérlő programját meg kell változtatni, hogy az új eszköz parancskészletét kezelje.

Költségek

Jelen pillanatban kilenc légkondicionálót kezelünk a leírt rendszerrel; valamennyi ugyanabban az épületben található. Hamarosan további tizenötöt veszünk fel. Az egyes mikorvezérlők alkatrészköltsége 60 dollár, a központi vezérlő számítógép költsége 500 dollár. További költségek az ***RS-485*** hálózat kiépítése és természetesen a rendszer kifejlesztésének és megvalósításának költsége.

Összefoglalás

A megvalósított rendszer megfelel az aktuális felhasználó igényeinek. A projekt sikere miatt az egyetem által vásárolt valamennyi további légkondicionálónak együtt kell működnie a rendszerrel. Ennek egyetlen követelménye, hogy valamennyi új légkondicionálónak rendelkeznie kell infravörös távirányítóval.
Az RTAI-nak hála a rendszer fővezérlő feladata független a Linux alatt futó felhasználói felület folyamataitól. Még abban a valószínűtlen esetben is, ha a Linux leállna, a rendszer tovább folytatná az előre beprogramozott műveleteket. A jövőben a rendszer könnyedén kiterjeszthető az épület világításának, riasztóinak, korlátozott elérhetőségű területeinek és egyéb rendszereinek vezérlésére.

Linux Journal 2004. november, 127. szám

Andres Benitez (adorego@conacyt.org.py) Villamosmérnöki diplomáján dolgozik az Asuncioni Katolikus Egyetemen, Paraguaiban. A cikkben bemutatott megoldás e diplomamunka végső feladata.

Vicente Gonzalez (vgonzale@uca.edu.py) építőmérnök. Automatizálásból szerzett MSc diplomát a Madridi Polytechnic Egyetemen. Jelenleg az Asuncioni Katolikus University Egyetem elektronikai és számítástudományi tanszékén dolgozik tanársegédként.

Pont-pont kapcsolatok megvalósítása Linux segítségével

Egy pénzügyi cég úgy döntött, felépíti saját redundáns WAN útválasztóit. Most megismerhetjük a trükkös részleteket és hogy hogyan is működik az egész.

Múlt nyáron a befektetési cég, amelynél dolgozom úgy döntött, létrehoz egy katasztrófa elhárító helyet. Itt, *New Yorktól* 40 mérföldre tükrözné az összes *Manhattan* belvárosában zajló műveletet. Projektünkben a lehető legtöbb helyen szerettünk volna *Linuxot* használni, mégpedig a következő okok miatt:

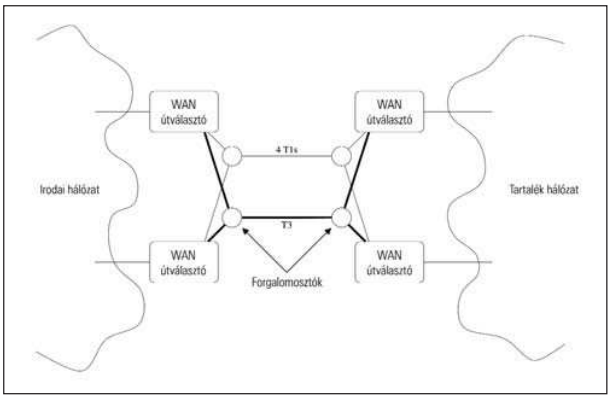
1. Egyébként is főként *Linuxon* dolgozunk, így használhatnánk a meglévő tapasztalatainkat.
2. Tetszőlegesen testre szabhatnánk a beállításokat, hiszen minden nyílt forráskódú lenne.
3. Azt reméltük a *Linux* megoldások olcsóbbak mint más cégek, például a *Cisco* megoldásai.

Ebben a cikkben azt fogom bemutatni, minként használható nagy kiterjedésű hálózatokban (*Wide Area Network*; *WAN*) a *Linux* útválasztóként. A *WAN* útválasztó saját meghatározásom szerint az a rendszer, amely két WAN kimenettel rendelkezik. Ezek lehetnek például T1 és T3 vonalak, de az is elképzelhető, hogy a rendszer egy helyi, például egy 100baseT szabványú hálózathoz is csatlakozik és a két hálózat közt továbbítja a csomagokat.

A hálózati megtervezése

Külön vonalakat vásároltunk, hiszen katasztrófa elhárító helyről volt szó, így a lehető legmegbízhatóbb vonalakra volt szükségünk. Számításaink szerint egy T3 (45Mb/sec) és négy T1 (egyenként 1,544Mb/sec) vonal elegendő sávszélességet biztosítana a műveleteink számára. Végül a T3-as vonalat használtuk elsődleges kapcsolatként és a T1-eseket meghagytuk kötött 5,7Mb/sec sebességű kisegítő vonalnak. A *WAN* kapcsolatokat a hálózati tervünk határozta meg. Biztosítékként két *WAN* útválasztót állítottunk szolgálatba mindkét oldalon. Az útválasztók azonosak és a T1 és T3 vonalakhoz egyaránt megfelelő alkatrészekkel rendelkeznek. Egy osztóalkatrész beiktatásával szerettük volna az összes *WAN* kapcsolatot az összes útválasztóval összekötni, ahogy azt az *1. ábrán* láthatjuk. Végül azonban kiderült, hogy ezt a megoldást elképesztően nehéz megvalósítani az alább ismertetett nehézségek miatt.

A *WAN* csatolásokon felül a távoli gépet a szolgáltatást nyújtó cég gerincvezetékén keresztül az internethez is csatlakoztattuk. A „több kapcsolat jobb mint kevés” elvet követtük és ez igen hasznosnak bizonyult később a hálózat tervezésénél. Amikor az ember egy félregépelt paranccsal véletlenül leállítja a T3-as vonalat, akkor tudja igazán értékelni, hogy az útválasztókhoz beépített egy hátsó ajtót az internet felől.



1. ábra Redundáns WAN csatornák

Alkatrész igény

A szolgáltató cégnél egy darab szabványos szekrény hely állt rendelkezésünkre. Következésképpen a helyigény elsődleges szemponttá vált, hiszen mi sok kiszolgálót szerettünk volna telepíteni. Ezért a *WAN* útválasztóinkhoz 1U rendszereket választottunk. Nehéz volt meghozni ezt a döntést, hiszen az alkatrészellátással kapcsolatos lehetőségeink így erősen korlátozottá váltak. Visszatekintve sokkal egyszerűbb lett volna 2U rendszerként elkészíteni a *WAN* útválasztóinkat.

A következő lépés a T1 és T3 csatolófelületet biztosító kártyák kiválasztás a volt. A fő kérdést itt annak eldöntése jelentette, hogy olyan kártyát használjunk-e, amely integrált csatorna szolgáltatási egységgel / adat szolgáltatási egységgel rendelkezik (integrated channel service unit/data service unit azaz *CSU/DSU*) és ezt közvetlenül csatlakoztassuk a bejövő *WAN* körre, vagy egy nagy sebességű soros csatlakozóval rendelkező kártya mellett döntsünk önálló *CSU/DSU* egységgel kiegészítve. Helybéli kötöttségeinket

figyelembe véve az integrált megoldás tűnt előnyösebbnek. Korábbi *WAN* telepítéseinkben, *Cisco 2620*-es útválasztó dobozokat alkalmaztunk T1 kártyákkal kiegészítve. Ebben a projektben azonban ez már nem volt elegendő, hiszen több T1 és T3 vonalat szerettünk volna alkalmazni.
Hosszú keresgélés után egyetlen gyártónál találtunk olyan eszközt amely T3 és több T1 kártyát is kezelni tudott, az *SBE Inc* termékét. Az ilyen kártyák piaca nem túl nagy, a gyártók sincsenek túl sokan. *WAN* kártya keresésnél érdemes elkezdeni a technikai támogatást nyújtó részleget kérdésekkel bombázni és figyelni, hogyan válaszolnak. Egyúttal figyelmesen olvassuk át a meghajtó és alkatrész adatokat mielőtt valamelyik gyártó mellett letennénk a voksunkat.

Az útválasztók fizikai tervezése

Az *SBE* T3-as és a négy T1-es kártyájához két darab teljes magasságú, fél hosszúságú *PCI* helyre volt szükségünk. Végül a *Tyan S5102* alaplapok mellett döntöttünk egyetlen *Pentium 4 Xeon 2.4GHz* processzorral. Memóriának 256MB ECC RAM-ot alkalmaztunk a maximális megbízhatóság érdekében.
A rendszerleállás esélyének minimalizálása érdekében Flash-alapú IDE eszközöket alkalmaztunk. A *SimpleTech* cégnél találtunk egy olyan eszközt, amely hagyományos merevlemezként csatlakoztatható és vezérelhető. 256MB méretű eszközök mellett döntöttünk, mivel úgy véltük elegendő lesz a *Fedora Core 1* működéséhez.
A teljes számítógéprendszert (a *WAN* kártyáktól eltekintve) fehér doboz (white box system) forgalmazótól vásároltuk. Ez kicsit problémásnak bizonyult, mivel a forgalmazó nem volt képes négy teljesen azonos rendszert összeállítani. A rendszerek *CPU* hűtő gyártók és memória sebesség tekintetében tértek el egymástól.
Az egyetlen terület ahol a forgalmazó hasznosnak bizonyult a helyes doboz kiválasztása volt. A számtalan rendszergyártó közül akikkel beszéltem csak egyetlen egy tudott olyan alaplap doboz kombinációt kínálni amely két teljes méretű *PCI* kártyát képes befogadni. Reméltük, hogy a *Delltől* vagy *IBM*-től tudunk valamilyen alapgépet vásárolni, de egyik nagy név sem tudott a követelményeinknek megfelelni.

Vezetékek és kábelezés

Létfontosságú, hogy az irodát több vezeték is összekösse a háttérhellyel. Tudjuk meg meg ki szolgáltatja a rendszerünket és a háttérhelyet több különböző szolgáltatótól rendeljük meg. Irodánk fizikailag két szolgáltatóhoz kapcsolódik, így végül a T3 vonalat az egyiktől a T1-eseket a másiktól rendeltük meg. Ha nem nézünk utána figyelmesen melyik szolgáltató rendelkezik tényleges fizikai vonallal a helyünkhöz, végül úgy járhatunk, hogy valamelyik ponton egyetlen gyártó vonalán fut át minden adatunk.
T1-ek szabványos RJ-45 kábeleken keresztül jönnek be. Általában a szolgáltató a demarkációs pontnál (demarc) fejezi be a T1-eseket – és egyúttal a garancia vállalását is. A demarkációs pont általában az a hely, ahová az összes telefonkapcsolatunk is megérkezik. Innen indítva egyszerű dolog *Ethernet* kábeleket vezetni a szekrényhez.
T3-asok már kicsit több odafigyelést igényelnek. A fizikai kapcsolat két koaxiális kábelen keresztül történik, az egyiken az adás a másikon a vétel. A T3-as RG-59A jelű kábelt használ *BNC* csatlakozókkal. A T3 szolgáltató felhívta a figyelmünket, hogy a kiszolgáló termünk túl messze van a vevőtől az épületünkben, így T3 ismétlőre lesz szükségünk. Ez 4U helyet és 120-voltos csatlakozót igényelt a szekrényünkben. Szerencsére ez a távolsági probléma a társzolgáltatói oldalon nem ismétlődött meg.

Megosztás és vezetékek

Az volt a célunk hogy a valamennyi vezetéket az összes *WAN* útválasztóval összekötjük (1. ábra), a tartalék rendszeren pedig mindkét végen kikapcsolva hagyjuk a őket. Mindkét oldalon egy-egy kiszolgáló lenne a T3 mester, míg a másik a T1-eseket szolgálná ki. Ha bármelyik útválasztó kiesik, a kört fel lehet hozni a másik útválasztón.
Kutatásaink, de különösen a *Cisco* felső kategóriás telekommunikációs felszerelése alapján tudtuk, hogy a vezetékeket meg lehet osztani. A legfőbb feltétel az, hogy mindkét oldalon egy időben csak egy rendszer adhat vagy fogadhat adatot. Mint kiderült ez komoly probléma, mivel az *SBE* alkatrészeket nem úgy tervezték, hogy képesek legyenek inaktívak lenni mialatt a vonalra csatlakoznak. A kritikus probléma az volt, hogy a T3 kártya transmittere automatikusan bekapcsolt ha a kártya áramot kapott. Így ha két rendszer közt élő T3 kör kapcsolatunk van, egy-egy mindkét oldalon, és beüzemeljük a tartalék rendszert, a T3 leáll, mivel egy oldalon két rendszer próbál adatot küldeni. Ez részlegesen kezelhető ha a kártya transmitterének shutoff parancsot küldünk. Ez azonban azonban a gép elindulása és az OS telepítése előtt nem lehetséges, ami néhány perces potenciális késést jelent.
Azt is felfedeztük, hogy a koaxiális kábeleken a T3 jelek impedanciájának meg kell egyeznie. A T3-as kábelek impedanciája 75 ohm. Ha egyszerűen megosztjuk a kapcsolatot, a két eredményül kapott kábel impedanciája 37.5 ohm lesz, ami az alkatrészeinktől függően vagy elég vagy nem. A T3-as kábelek megosztásának helyes módja ha erősített megosztót alkalmazunk, ahol egy transzformátor mindkét kapcsolaton 75 ohm-ra egyenlíti ki az impedanciát. Mi a *Micro Circuits, Inc* passzív erősített osztóit használtuk.
A T1-esek megosztása sokkal egyszerűbben ment. Itt elegendő egy RJ-45-ös elosztóval az egyetlen bejövő kábelt két kimenő kábellé alakítani. Továbbá az *SBE 4T1* kártya nem kapcsolja be a transmittert a meghajtó betöltése eljött, így a kapcsolatot megoszthatjuk a rendszerek közt.
Valamennyi megosztott kapcsolatot sikerült kialakítanunk. Ugyanakkor a T3 kártyáknál tapasztalt indulási és egyéb problémák miatt a megosztóink jelenleg nincsenek telepítve. Amennyiben ezt a lépést is használni szeretnénk, először meg kell győződnünk arról hogy minden sziklaszilárdan működik és csak akkor álljunk neki a megosztásnak. Máskülönben a hibakeresések során egyfolytában le kell szednünk a megosztást, mivel nem vagyunk biztosak az összeállításunk helyességében.

Saját terjesztés összerakása

A 256MB-os flash tárolóegység tömör OS telepítést követelt meg. A *Telemetry*-nél valamennyi *Linux* helyen a *Fedora Core 1* változatot szabványosítottuk. Így kényelmes megoldás volt az útválasztókon is *FC1*-et telepíteni. A két célunk:

1. A *Fedora Core 1*-hez hasonló dolgot készíteni ami ráfér az apró meghajtóra.
2. A rendszerbeállítások megváltoztatása, hogy elkerüljük a felesleges lemezírásokat.

Ez azért fontos mert a flash meghajtók élettartama véges, így naplófájlokat helyezni rájuk nem túl jó ötlet.
Mint kiderült, saját *Fedora* rendszert egészen könnyen ki lehet alakítani, különösen a korábbi *Red Hat* kiadásokhoz képest. Az egész dolog kulcsa, a saját rendszer image kialakítása egy másik gépen amelyen friss *RPM* adatásbázis található, majd ezt az image fájlt át kell vinni az útválasztóra. Az *Linux Journal* FTP lapjáról letölthető 1 listában (➲ *ftp.ssc.com/pub/lj/listings/issue126/7661.tgz*) bemutatjuk hogyan készíthetjük el az alap rendszer image fájlt. A folyamat során először egy új *RPM* adatbázist készítünk az építő rendszeren, felrakjuk a minimális *RPM* készletet felépítve a rendszert majd telepítjük az összes kívánt *RPM*-et. Én a `--aid` kapcsolóval utasítottam az *rpm*-et, hogy automatikusan oldja fel az összes függőségi problémát a megadott könyvtárban keresve ahová előzőleg az összes *Fedora RPM*-et feltettem. Így nem kellett kézzel kikeresnem az össze függőséget. Miután a rendszert felépítettük, másoljuk a útválasztóra teszteléshez. A flash meghajtón elérhető 256MB tárhelyből mindössze 171 MB-ot felhasználva működő rendszert tudtunk kialakítani.

Az útválasztó beállításainak finomhangolása

A *Linux* útválasztó használatának célja, hogy a lemezírás és olvasást minimalizáljuk. Ez azért fontos mert a Flash meghajtó csak korlátozott számú újraírást tesz lehetővé (ez általában néhány száz vagy ezer alkalmat jelent). Az ilyen műveletek számát úgy csökkentettük, hogy az útválasztót laptopként kezeltük. Először is a rendszermagban bekapcsoljuk a laptop módot, ahogy azt a *Linux Journal 2004 szeptemberi* számában olvashattuk. (A cikk magyarul a *Linuxvilág 2004 októberi* számában olvasható.) Ezzel az írásokat elhalaszthatjuk a következő olvasási kérelemig így azok nem kerülnek azonnal a lemezre.
Másodszor, a fájlrendszerünk becsatolási opciói között szintén állítsuk be az írások elhalasztását. Az *ext3* esetében állítsuk a jóváhagyássi (*commit)* időintervallumot 60 másodpercre. Majd a fájlrendszert a *noatime* kapcsolóval csatoljuk be így a fájlok olvasása nem fogja kiváltani a az elérési időpont megváltoztatását.
Harmadszor, valamennyi naplófájlt mozgassunk át a RAM-alapú *tmpfs* fájlrendszerre. A 2. listában bemutatjuk, hogyan szerkeszthetjük újra a fájlrendszerünket és mozgathatjuk a */var* összes naplóállományát a */var/impermanent* alatt található *tmpfs* fájlrendszerre. Hogy mindez igazán hatékony legyen, egy parancsfájlra is szükségünk lesz (3. lista) amely a naplófájlokat rendszer leállításakor tar labdaként elmenti, majd rendszerindításkor visszatölti (a 2. és 3. lista az Linux Journal FTP helyén szinten elérhető). Ezt a parancsfájlt rendszerindításkor a lehető leghamarabb kell meghívni a */etc/rc.d/rc.sysinit*-ben, leálláskor pedig a lehető legkésőbb a */etc/init.d/halt*-ban.

A WAN csatlakozók beállítása

WAN csatlakozók megtévesztőek! Például a T3 és T1 meghajtók a rendszermag *HDLC* vermének különböző verzióit használják. Ez azt jelenti, hogy a *sethdlc* program két verzióját is használnunk kell, egyenként a megfelelő *hdlc* veremhez lefordítva ha a *WAN* vezeték protokollját be szeretnénk állítani.
Elég sok beállítási lehetőségünk van a T3 vagy T1 körök esetében – külső vagy belső időzítés? *CRC* méret? *HDLC* mód? És így tovább. Szerencsére, az *SBE* támogatási részlege nagyon segítőkész volt és rengeteg beállítási és hibakeresési tippet adott.
A négy T1-est egyetlen csatolt körré alakítottuk a *teql* segítségével. Ez ugyan jól működött, de ha az egyik T1-est eltávolítottuk a teljesítmény borzalmas volt még akkor is ha újrakapcsolódott. Munkatársam *Bill Rugolsky*, megtalálta a hiba forrását: a csatlakozás állapotának lekérdezése hiányzik. Az *SBE* kártya jelenti, hogy a csatlakozás élő-e vagy sem, de ez az üzenet a veremig már nem jut el. Így a *teql* megpróbálta elküldeni a csomagokat a működésképtelen csatlakozáson keresztül. *Bill* az *SBE* meghajtó foltozásával és mások által készített foltok felrakásával végül megoldotta a *teql* és *linkwatch* figyelmeztetések problémáját. A meghajtó foltokat elküldte az *SBE*-nek, reméljük következő meghajtó verziójukba már benne lesznek a javítások.
Az OSPF-et főnökünk, *Andy Schorr* állította be a *WAN* csatolásokon keresztüli útválasztás kezeléséhez. A szükséges keretrendszert a nyílt forráskódú *Quagga*, a *Zebra* utódja biztosította. Amennyiben az egyik csatlakozás leáll (emlékezzünk két csatlakozás van, a T3 és a virtuális csatlakozás az összekapcsolt T1-eseken keresztül), a *Quagga* ezt észreveszi és a csomagokat másik csatolófelületen keresztül küldi tovább. A pont-pont (point-to-point) kapcsolatokat hagyományosan a másik csatolófelület címének kölcsönzésével állítjuk be, amely általában az *eth0* lesz. Mi ezzel szemben minden point-to-point csatlakozásnak külön dedikált alhálózatot készítettünk. *Andy*-nek módosítania kellett a forráskódot, hogy a *Quagga* helyesen működjön ilyen beállítások mellett is.
Kénytelenek voltunk kidolgozni néhány *iptables* szabályt is, hogy a *Quagga* helyesen működjön az összekapcsolt T1-eseken. A *teql* eszköz csak küldeni tud, így a csomagok soha nem jelennek meg rajta. Emiatt a *Quagga* eldobta a csomagokat, hiszen azok nem jó csatolófelületről érkeznek.
A problémát néhány *iptables* szabállyal oldottuk meg. Használatukkal az összes T1 csatolófelületen (hdlc0-tól hdlc3-ig) érkező csomag úgy tűnik mintha a *teql0*-ról jönne:

```
iptables -t mangle -A PREROUTING -i hdlc\+ -j
➥TTL --ttl-inc 1
iptables -t mangle -A PREROUTING -i hdlc\+ -j
➥ROUTE --iif teql0
```

Összefoglalva a *WAN* csatolások készítése trükkös dolog, elég sok tanulást és csiszolgatást igényel. Ne várjuk, hogy a dolgok maguktól működni fognak csak mert összedugtuk a kábeleket.

Akadályok az út során

Számos problémát kellett megoldanunk mialatt a *WAN* útválasztóinkat beállítottuk. Néhány ezek közül a *WAN* meghajtókkal kapcsolatos amint azt korábban említettük. Mialatt ezt a cikket írtam, felfedeztük, hogy a T1 teljesítmény

erősen leromlott és a *ping* idők erősen változnak – a szokásos 10ms helyett 1 másodperc is előfordult. A nyomok az egyik *WAN* kártyához vezettek amely nem hozott létre megszakításokat; meglazult a *PCI* foglalatban. Az erősen változó csomagidőtartamok azért jöttek létre, mert az azonos megszakítási vonalat használó többi eszköz (*eth0*) viszont küldött megszakításokat. Ezek aztán felébresztették az *SBE* meghajtót és feldolgozták a megszakítást. Az ilyen típusú nem egyértelmű hibák rávilágítanak a csatlakozás minőségfigyelésének fontosságára.

A jövő

Az alaprendszerrel elégedettek vagyunk, de számos fejlesztést kel még végrehajtanunk. A többszörös T1 összetett csatolófelülettel kapcsolatos bosszantó problémákra tekintettel jelenleg tervezzük, hogy a T1-eseket egy második T3-asra cseréljük. Miután ezt megtettük a teljes vezetékosztást elfelejthetjük. A vezetékek megosztása egy teljesen új bonyolultsági szintet visz a rendszerbe és nem vagyunk meggyőződve róla, hogy megéri.

További fejlesztéseket kell végeznünk a két vonal állapot és minőség megfigyelése terén. Elég nehéz vezetékszolgáltatóknál panaszt emelni a teljesítményre ha nincsen semmiféle adatunk amivel állításunkat alátámaszthatnánk.

Kényelmes lenne szabványos kiszolgálókat alkalmazni az útválasztó gépnek. Folyamatosan figyeljük a nagyobb gyártók 1U szekrénybe illeszthető gépeit, de valamilyen ok miatt egyik sem megfelelő. Az a gond, hogy a *BIOS* nem engedélyezi bármilyen *IDE* eszközről a rendszerindítást. A gyártó ismeri ezt a korlátot, de nem javítja ki a *BIOS*-t. Így aztán előre láthatóan magunknak kell elkészítenünk a rendszerünket a jövőben. A *LAN* forgalom kezelésére további belső útválasztókat fogunk üzembe helyezni, a kifejlesztett *WAN* útválasztó modellt felhasználva, Flash meghajtóval rendelkező 1U rendszereken futó minimális *Fedora* rendszermaggal.

Összefoglalás

Bár a projekt még nem teljes, úgy érzem eleget haladtunk hogy értékelni lehessen az eredményeket. A kulcskérdés az, vajon újból nekivágnánk-e a dolognak? A válasz egyértelműen igen. A *WAN* útválasztóink redundáns kapcsolatot biztosítanak az iroda és a háttérhely között. A *WAN* vezetékek megosztása a redundancia növelésére nem igazán volt sikeres, hiszen túlbonyolítja a rendszert.

A projekt lényegesen hosszabb időt vett igénybe mint eredetileg terveztük, ami általános jellemzője a saját rendszer kifejlesztésének. A válaszok ott vannak, csak sokáig tart megtalálni őket. Egy ügyes, szakértő csoport (mint amilyen nekem volt) elengedhetetlen az ilyen munkához. Csak ne felejtsünk el egy kis többletidőt biztosítani az apró bosszantó problémákra, melyek szinte bizonyosan megjelennek majd.

Linux Journal 2004. október, 126. szám

Phil Hollenback a Telemetry Investments rendszergazdája New York-ban. Amikor éppen nem Linux gépeket frissít vagy görkorcsolyázik, Phil www.hollenback.net címen olvasható weblapjának fejlesztésével tölti idejét.

FM rádió hallgatása programmal lépésről lépésre

Kezdjük a szoftveres rádiók témáját egy olyan projekttel, amely két FM állomást képes egyszerre befogni.

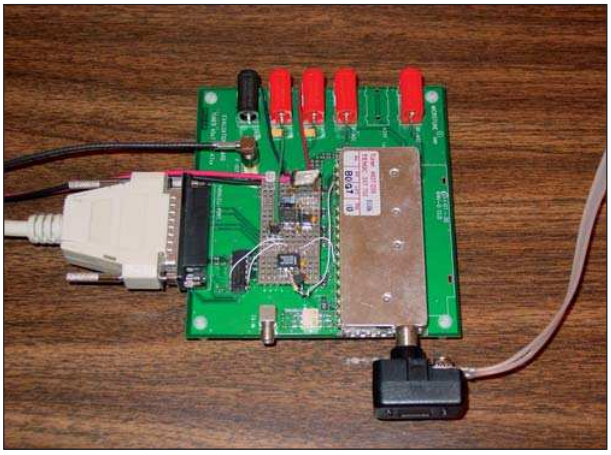
A *Linux Journal* 2004 júniusi számában megjelent *„GNU Radio: A rádiófrekvenciás világ felfedezésének eszközei”* című cikkemben (*Linuxvilág, 2004 július 68-72. oldal*) áttekintést adtam a *GNU* rádiórendszerek működéséről és ismertettem néhány hardveres módszert a rádiófrekvenciás (RF) jelek számítógépes digitalizálására. Ebben a cikkben azt vizsgálom, miképpen használhatjuk a *GNU Radiot* FM műsorok hallgatására.

Az *1. ábrán* a cikkhez használt eszköz felépítése látható. Ez a nyers és minden sallangtól mentes megközelítés tökéletesen alkalmas a működés magyarázatára. A cikk későbbi részében összefoglalom az ***USRP*** (***Universal Softvare Radio Peripherial***, univerzális programrádió-periféria) legfontosabb tudnivalóit is.

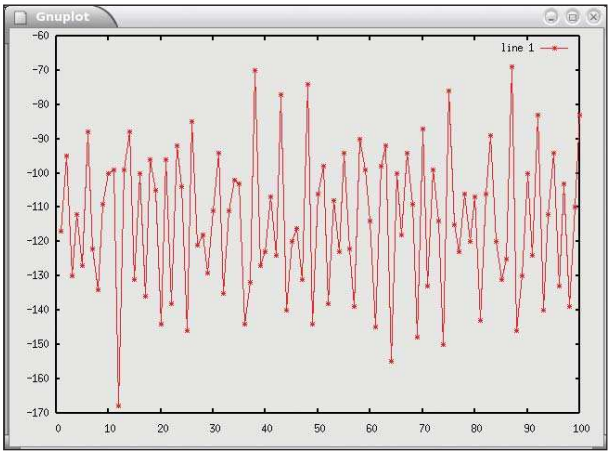
A készülékünk egy közönséges FM dipólantennából, egy próbatáblára szerelt kábelmodemes vevőegységből és egy 20 millió jel/másodperc mintavételi sebességű analóg-digitális átalakítást végző PCI kártyából áll. A vevőegység bemenetére az antenna csatlakozik. A vevőegység IF-kimenete egy darab koaxiális kábellel csatlakozik a számítógép hátlapján lévő ADC-bemenetre. A vevőegység próbatáblája a PC párhuzamos kapujára van kötve, így lehetőségünk van az egység vezérlésére is.

A használt egységek: a ***Microtune 4937 DI5 3X7702*** kábelmodemes vevőegység és egy ***Measurement Computing PCI-DAS 4020/12 ADC*** kártya. Ez a vevőegység-típus már nehezen beszerezhető, de mások – mint például a ***Sharp Microelectronics*** által gyártottak – is jó szolgálatot tesznek (lásd a kapcsolódó címeket tartalmazó részt).

A kábelmodemes vevőegység tölti be az RF előtag szerepét és ez felelős azért, hogy a számunkra fontos rádiófrekvenciás jeltartományt olyan frekvenciasávba konvertálja, amit az ADC-egységünk kezelni tud. Esetünkben az egység az 50 MHz-800 MHz-ig tartó frekvenciatartomány egy választható 6 MHz-es szeletét alakítja át egy 5,75 MHz frekvenciaközéppel bíró 6 MHz-es szeletre. Az elvről részletesebben az említett júliusi lapszámban megjelent cikkben olvashatunk.



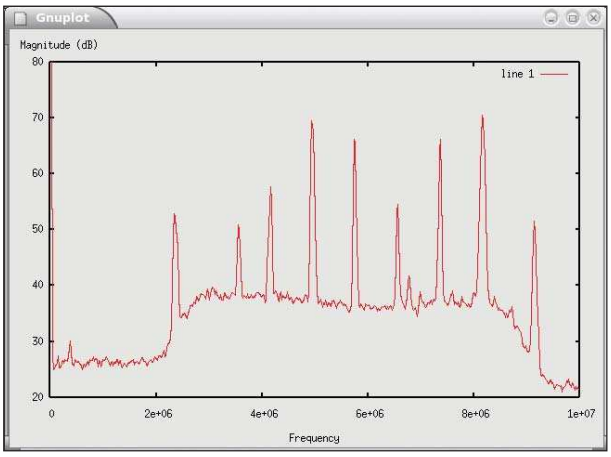
1. ábra A kábelmodemes vevőegység rádiófrekvenciás előtagja



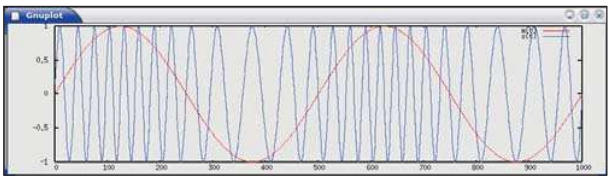
2. ábra Az ADC-minták képe az időtartományban

Vágjunk bele!

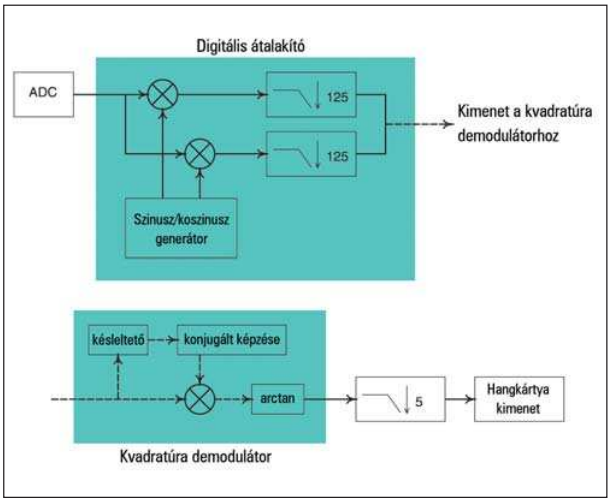
Elsőnek vizsgáljuk meg, mi történik, ha az előtagunkat az FM frekvenciasáv közepére, mondjuk 100,1 MHz-re hangoljuk. A *2. ábra* mutatja a kapott mintákat az idő függvényében. Ez az időtartományban mutatott nézet az, amit egy oszcilloszkópon is láthatunk. Ebből sok mindent nem tudunk kiolvasni, az mindene esetre bíztató, hogy a minták a 170-től 70-ig tartó intervallumba esnek. Ideális esetben a jelek a null értékhez képest szimmetrikusan helyezkednének el, de a céljaink szempontjából ez az eltolás nem lényeges. A frekvenciatartománybeli vizsgálat további információkat szolgáltat. Ebben az esetben egyazon időben 1024 mintát veszünk és ennek számítjuk ki a Fourier-transzformáltját, a gyors Fourier-transzformáció (FFT) algoritmusának alkal-



3. ábra Az FM sáv gyors Fourier-transzformáltja kilenc állomással



4. ábra Egy egyszerű frekvenciamodulált jel



5. ábra Az FM vevő blokkdiagramja

mazásával. Ezzel képet kapunk azokról a frekvencia-összetevőkről, amelyek a bemeneti jelben szerepelnek. A *3. ábrán* láthatjuk a kapott spektrumot. Az x-tengely a frekvenciát jelöli, az y-tengelyen pedig a teljesítmény decibelben kifejezett (10 * log10 teljesítmény) értékei szerepelnek. Az alsó határ nulla Hz, a felső pedig 10 MHz, a mintavételi frekvenciánk fele.

A 3. ábrán látható tüskék mindegyike egy-egy rádióállomást jelent. A programunk elsőre látja mindegyiket!

Egy állomás hallgatásához módot kell találnunk arra, hogy az állomás jeleit elválasszuk az összes többi állomásétól, alakítsuk át az alapsávra (egyenáram, 0 Hz) és alakítsuk vissza a frekvenciamoduláció hatását. Lépésről lépésre végig fogunk haladni ezeken a folyamatokon, de mindezek előtt nézzük, mit is takar az FM rövidítés.

1. lista A kvadratúra-demodulátor fejállománya

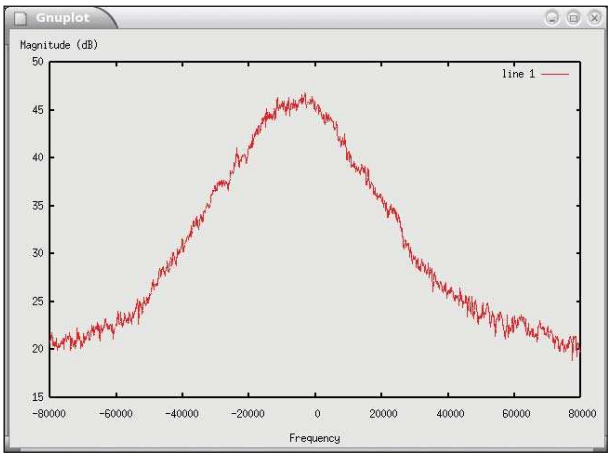
```
#ifndef INCLUDED_GR_QUADRATURE_DEMOD_CF_H
#define INCLUDED_GR_QUADRATURE_DEMOD_CF_H

#include <gr_sync_block.h>

class gr_quadrature_demod_cf;
typedef boost::shared_ptr<gr_quadrature_demod_cf>
  gr_quadrature_demod_cf_sptr;

gr_quadrature_demod_cf_sptr
gr_make_quadrature_demod_cf (float gain);

/*
 * quadrature demodulator: complex in, float out
 */
class gr_quadrature_demod_cf : public
gr_sync_block
{
  friend gr_quadrature_demod_cf_sptr
    gr_make_quadrature_demod_cf (float gain);
  gr_quadrature_demod_cf (float gain);

  float        d_gain;

 public:

  int sync_work (
        int noutput_items,
        gr_vector_const_void_star &input_items,
        gr_vector_void_star &output_items);
};

#endif /* INCLUDED_GR_QUADRATURE_DEMOD_CF_H */
```

Mi az a frekvenciamoduláció?

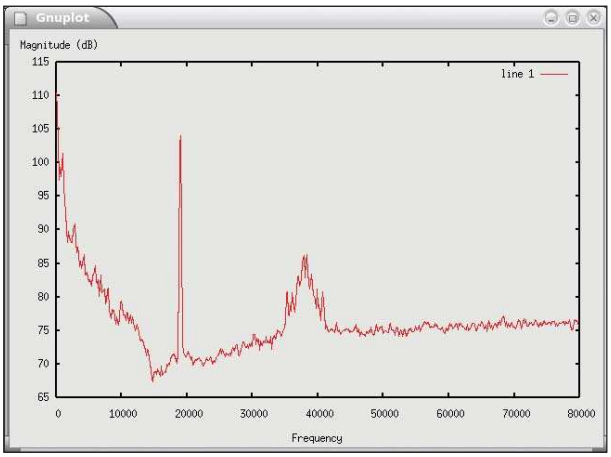
Ahhoz, hogy megértsük egy FM vevő működését, nem árt ismernünk az FM jelek keletkezésének módját. A frekvenciamoduláció (FM) során a vivőfrekvencia pillanatnyi értékét a bemeneti jel frekvenciájának a függvényében változtatjuk. A *4. ábrán* láthatjuk az m(t) bemeneti jelet (beszéd vagy zenei jel), és a létrejövő s(t) modulált jelet. Matematikai precizitással felírva, a pillanatnyi frekvencia mindenkor az alábbi összefüggés szerint számolható:

```
f(t) = k*m(t) + fc
```

Az ***m(t)*** jelöli a bemeneti jel frekvenciáját, a ***k*** egy állandó érték, amely a frekvenciaérzékenységet szabályozza, az ***fc*** pedig a vivőjel frekvenciája (például 100,1 MHz). Emlékezzünk rá, hogy a frekvencia mértékegysége radián/másodperc, így a frekvenciát tekinthetjük forgási gyorsaságnak is. Ha integráljuk a frekvenciát, fázist vagyis szögértéket kapunk. Megfordítva pedig a fázis idő szerinti differenciálása frekvenciát ad eredményül. Ezek azok a kulcsmotívumok, amiket a vevő megépítésekor felhasználunk.



6. ábra Gyors Fourier-transzformált a digitális frekvenciaváltó kimenetén



7. ábra A demodulált FM jel gyors Fourier-transzformáltja

A blokkdiagram

Az *5. ábrán* látható az FM állomások hallgatására szolgáló módszerünk. Ha eltávolítjuk a vivőhullámot, egy olyan alapsávú hullámunk marad, amelynek pillanatnyi frekvenciája arányos az eredeti jel ***m(t)*** értékével. Vagyis az előttünk álló feladat nem más, mint eltávolítani a vivőhullámot és meghatározni a pillanatnyi frekvenciát.

Az első rész nem okoz nehézséget, a vivőhullámtól a programmal megvalósított digitális frekvenciaváltó (DDC) segítségével szabadulunk meg, amelyet `freq_xlating_fir_filter_scf` elnevezéssel illettünk.

Az egység egy numerikus vezérlésű oszcillátorból áll, amely azokat a szinusz és koszinusz hullámokat állítja elő, amelyeket ki szeretnénk oltani, valamint egy keverőből (ez a programunk számára egy szorzónak felel meg), és egy tizedelő véges impulzusválaszú szűrőből. Az ***scf*** utótag jelzi, hogy ez az egység a bemenetén ***short*** (rövid) jeleket fogad, a kimenetén komplex jelfolyamot állít elő, a szűrő megadásához pedig lebegőpontos mintákat használ.

A digitális frekvenciaváltó azt a trigonometrikus azonosságot használja ki, hogy amikor két ***f1*** és ***f2*** frekvenciájú szinuszhullámot összeszorzunk, az eredmény két másik szinuszhullámból áll elő, amelyek frekvenciái ***f1+f2*** illetve ***f1-f2***. Esetünkben a bemenő jelet szorozzuk a vivőjel frekvenciájával. A kimenő jel két összetevőből áll, az egyik a vivőjel kétszerese, a másik pedig egy nullszintű jel. A kétszeres komponenstől egy aluláteresztő szűrővel szabadulunk meg, s így megmarad az alapsávú jelünk.

A számítási gyorsaság a lényeg!

A digitális frekvenciaváltó egyenes szoftveres megvalósítása rendkívül számításigényes feladat. Ekkor a teljes bemenő frekvencián elvégeznénk a szinuszos és koszinuszos jelek előállítását és ezek összeszorzását. Egy ***Pentium 4*** processzorral a szinusz és koszinusz jelek számítása 150 utasításciklust igényel. Egy 20 millió jel/másodperc sebességű bemenettel számolva ez 20e6 * 150 = 3e9 ciklust jelentene másodpercenként csupán a szinusz és koszinusz kiszámítására! Ez nyilvánvalóan járhatatlan út. Jó hír azonban, hogy a DDC szoftveres megvalósításának létezik egy jobb módszere is. ***Vanu Boseet*** a ***„Virtual Radios”*** (***Virtuális rádiók***) című írásában – lásd a kapcsolódó címeket – vázolt eljárás szerint a műveletek sorrendjének átszervezésével és a valóságos együtthatók helyett a frekvenciaspecifikus komplex-szűrő együtthatóinak használatával elegendő az összes számítást tizedakkora frekvencián elvégezni. Ennek a módszernek az eredménye meghatározó, ezt már valós időben is el tudjuk végezni!

Kvadratúra-demoduláció

A következő feladat az alapsávú jel pillanatnyi frekvenciájának meghatározása. Erre a `quadrature_demod_cf` blokkot használjuk. A fázist közelítő módszerrel választjuk le a szomszédos minták közti szögeltérés meghatározásával. Emlékezzünk vissza, hogy a frekvenciaváltó blokk kimeneteként komplex számok álltak elő. Egy kis trigonometriai ismeret felhasználásával, két egymást követő minta közti szögeltérést meghatározhatunk oly módon, hogy az egyiket megszorozzuk a másik komplex konjugáltjával és az eredménynek vesszük az arkusz tangensét. Az *1.* és *2. listán* láthatjuk a `quadrature_demod_cf` blokk megvalósítását. Ha már tudjuk, hogy mit akarunk, nem kell sokat kódolnunk. A jelfeldolgozás zömét a `sync_work` ciklusában lévő három sor végzi. Mindezeket kipróbálva a *6. ábrán* láthatjuk a digitális frekvenciaváltó kimenetét, a *7. ábra* pedig a kvadratúra-demodulátor kimenetét mutatja. A 7. ábrán láthatjuk az FM hullámforma összes összetevőjét. A 0-tól körülbelül 16 kHz-es frekvenciáig terjedő rész a jobb és bal oldal (L+R) együttes hangtartománya. A 18 kHz-nél látható tüske a sztereo pilotjel. A bal és jobb oldal különbségjele (L-R) a pilotjel kétszeres frekvenciájának értékére középpontozva jelenik meg AM-modulált jelként az FM felső tartományában. Időnként további segéd-vivőfrekvenciákat is találhatunk az 57 kHz és 96 kHz közé eső frekvenciatartományban. Hogy ne bonyolítsuk túl a dolgokat, a kvadrát-demodulátor kimeneti jelét 16 kHz-es frekvenciánál levágjuk, amivel egy monó jelet kapunk, s ezt a hangkártya kimeneteire kapcsoljuk.

Egy többcsatornás vevőegység

A ***LinuxJournal*** FTP kiszolgálójáról (lásd a kapcsolódó címeket) letölthető 3. lista egy általános vevő megvalósításának Python-kódját mutatja. Valójában arra is képes, hogy két FM állomást fogjon egy időben, egyiket a bal oldali hangszórón keresztül, másikat a jobb oldalin. Nem állítom, hogy ennek különösebben nagy hasznát vesszük, mindenesetre

2 .lista A kvadratúra-demodulátor megvalósítása

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <gr_quadrature_demod_cf.h>
#include <gr_io_signature.h>

gr_quadrature_demod_cf::
  ➥gr_quadrature_demod_cf (float gain)
  : gr_sync_block (
       "quadrature_demod_cf",
      gr_make_io_signature(1,1,sizeof
(gr_complex)),
      gr_make_io_signature(1,1,sizeof (float))),
    d_gain (gain)
{
  set_history (2);    // provide 1 sample look
                      // ahead
}

gr_quadrature_demod_cf_sptr
gr_make_quadrature_demod_cf (float gain)
{
  return gr_quadrature_demod_cf_sptr (
      new gr_quadrature_demod_cf (gain));
}

int
gr_quadrature_demod_cf::sync_work (
    int noutput_items,
    gr_vector_const_void_star &input_items,
    gr_vector_void_star &output_items)
{
  gr_complex *in = (gr_complex *) input_items[0];
  float *out = (float *) output_items[0];
  in++;         // ensure that in[-1] is valid

  for (int i = 0; i < noutput_items; i++){
    gr_complex product = in[i] * conj (in[i-1]);
    out[i] = d_gain * arg (product);
  }

  return noutput_items;
}
```

jól mutatja a programmal megvalósított rádióban rejlő lehetőségeket. A több csatorna egyidejű vételének ez az ötlete felhasználható rádiós ***TiVo***-szerű eszközök alapjaként.
A kód három függvényre bomlik. A main kezeli a paraméterellenőrzést, irányítja az RF-előtagot és vezérli a fő jelfeldolgozó ciklust. Ha csak egyetlen állomást veszünk, az RF előtagot arra utasítjuk, hogy az állomást tegye a vevőegység kimeneti frekvenciájának közepére (az IF-re). Ha két állomást fogunk, azoknak egymástól 5,5 MHz-es távolságon belül kell lenniük. Ennek a megszorításnak a kábelmodemes vevőegységben lévő ***SAW***-szűrő az oka. Ez egy 5,75 MHz-re beállított sávátereszto szűrő, melynek értéke a 6 Mhz-es érték – az északamerikai TV-csatornák sávszélességének – közelében van. Ebben az esetben a különbséget megfelezzük, és az előtagot pontosan a két állomás közé hangoljuk. A `build_graph` az általános jelfeldolgozó blokkokat tartalmazza és kapcsolja egymáshoz. Mind az egy, mind pedig a több csatorna vételi eljárásánál egyetlen nagy sebességű analóg-digitális átalakítót használtunk a bemeneten és egy hangkártyát a kimenet számára. Minden egy időben hallgatni kívánt csatorna számára külön digitális frekvenciaváltó, kvadratúra-demodulátor és aluláteresztő szűrő került megvalósításra.

Többféleképpen is megvalósítható!

Az egyszerre fogható adók száma a számítógépünk sebességének függvénye. Még eme ötletes megvalósításunkra is igaz, hogy a processzorciklusok nagy része a `freq_xlating_fir_filter` blokkokkal terhelt. Az általunk most ismertetett módszer a nyers ADC-módszer nevet kaphatná. A számítási igény csökkentésének egyik módszere a digitális frekvenciaváltás végrehajtásának hardveres megvalósítása lenne. Több gyártó, így a ***Texas Instruments***, ***Intersil*** és ***Analog Devices*** cégek is kínálnak olyan céláramköröket (***ASIC***), amelyek képesek e feladat betöltésére.
A ***Universal Software Radio Peripherial (USRP)*** által követett módszer a digitális frekvenciaváltó ***Verilog*** hardverleíró nyelven való lekódolása, majd a keletkezett bitfolyam ***USB***-kapun keresztül az ***FPGA***-ba való áttöltése. Ezzel egy olyan kombinált megoldást kapunk, amely megőrzi a maximális rugalmasságot, ugyanakkor lehetővé teszi a számításigényesebb részek végrehajtásának hardverre történő átterhelését. Az ***URSP*** vonatkozásában további információk a ***GNU Radio Wikin*** érhetők el.

Összegzés

A cikkben egy egyszerű, de teljes funkcionalitású többsávos FM vevőt vizsgáltunk meg, miközben sikerült egy több ezer dolláros hardvert két ötdolláros tranzisztort tartalmazó rádióval egyenértékűvé tennünk, és képet kaptunk a feldolgozó folyamatokról is. Akiket mélyebben is érdekel az FM rádiózás, a GNU Radio kódtárában egy lényegesen jobb minőségű FM vevőre (***hifi_fm.py***) és egyéb finomságokra bukkanhat.
A ***GNU Radio*** terén mostanra rengeteg értékes munka született, amelyek némelyike a mozgó ideiglenes hálózatokat veszi célba, mások az amatőr rádiózás hagyományait követik vagy a szoftveres ***GPS*** megvalósításán fáradoznak, egy csoport pedig lázasan dolgozik a következő generációs föld-űr amatőr műholdas kommunikációs rendszer megteremtésén. Habár a ***GNU Radio*** eszközkészlet nagymértékben független a ki- és bemeneti eszközöktől, ezeknek a törekvéseknek a nagy része az ***USRP***-t használja, vagy tervezi használni a rádiófrekvenciás egységek és a számítógép közti csatolófelületként.

Linux Journal 2004. október, 126. szám

Eric Blossom a GNU Radio Project alapítója. Évekig dolgozott a biztonságos telefonszolgáltatások területén mielőtt programalapú rádiózással kezdett foglalkozni. Ha éppen nem programalapú rádiót bütyköl, akkor valószínűleg jógázással vagy ju-jitsuval tölti idejét. Az eb@comsec.com címen érhető el.

Alkalmazások teljesítményének növelése a folyamatok szinkronizálásával HPC rendszereken

Hahó, fürtcsomópont, most ne dolgozz azon a karbantartási munkán – mindannyian arra várunk, hogy befejezd az MPI munka rád eső részét! Ismerjünk meg egy hatékony fürtkezeléshez használható ütemezési rendszabály.

Azt várhatnánk, hogy ha megduplázzuk az alkalmazásra jutó számítási erőt az alkalmazás teljesítménye is duplájára nő vagy a futásidő felére csökken. Sajnos azonban a *HPC* felhasználók jól tudják ez bizony távol áll az igazságtól, ugyanis az aktuális hatékony teljesítmény akár a rendszer elméleti csúcsteljesítményének 5%-ára is visszaeshet. A *HPC* kutatók és alkalmazás fejlesztők rengeteg időt fordítottak és fordítanak ma is arra, hogy megtalálják az ilyen teljesítményvesztés forrását és feltornásszák az alkalmazás teljesítményét. Amikor elkezdtünk a *Cray XD1* rendszeren dolgozni, magunk is csatlakoztunk a problémával foglalkozó kutatók sorába. Cikkünk arról szól, hogyan tanultunk az előttünk járóktól, és hogyan építettünk erre a tudásra kifejlesztve egy Linux-ütemező alapú megoldást, amely lényegesen megemelheti a *Linux HPC* rendszerek alkalmazásainak teljesítményét.

A legtöbb kutató a *HPC* alkalmazások szerkezetére koncentrált. Különféle kutatócsoportok próbálták növelni a gyorstárazás hatékonyságát, keresték a processzorközi kommunikáció csökkentésének lehetőségeit és vizsgáltak hasonló elemeket, de egyik stratégia sem hozott néhány százaléknál komolyabb javulást. Egy másik kutatási terület azonban sokkal ígéretesebbnek tűnik. Ha megértjük milyen kapcsolat van a *HPC* alkalmazás és a rendszer háttér folyamatai között, megpróbálhatjuk megváltoztatni ezt a kölcsönhatást megnövelve a teljesítményt.

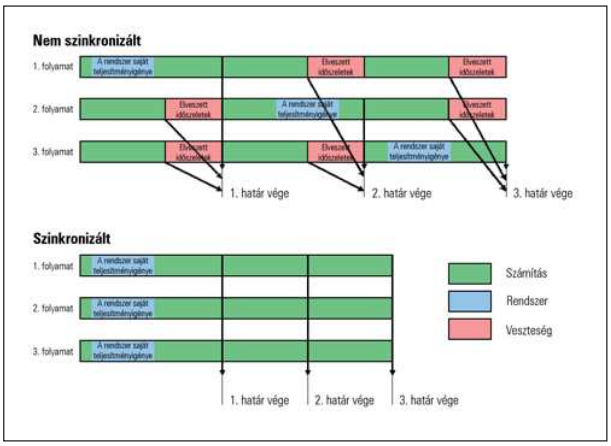
Hova tűnik a teljesítmény?

Ez irányú kutatásaikat ismertető ötletadó írásukban *Petrini*, *Kerbyson* és *Paking* a *Los Alamos National Laboratory* kutatói (lásd a hálózati forrásokat) azt állítják, az alkalmazások teljesítményét egy általuk „zajnak" nevezett tényező okozza – nevezetesen a nagy több ágon futó *MPI* munkafolyamatok és a háttérfolyamatok közötti kapcsolat. Megfigyelték, hogy a karbantartási feladatok, azaz a zaj miatt egyes processzorok nem tudták elérni az *MPI* korlátot (szinkronizálási pontok az alkalmazásban) emiatt az összes folyamatnak rájuk kellett várakozni, míg a processzor végre befejezte a karbantartást. Ez az összes többi processzoron elpazarolt ciklusokhoz vezetett.

Az *1. ábra* felső része ezt a kapcsolatot és az általa okozott teljesítményvesztést mutatja be. A bemutatott folyamatok egy párhuzamos munka részei, valamennyi külön processzoron fut, melyek időnként az *MPI* korlátok segítségével szinkronizálnak. A számítás első felében az 1. folyamat késlekedett, mivel a csomópont ütemezője leállította a folyamat végrehajtását, hogy a minden *Linux* vagy *UNIX* csomóponton megtalálható szokásos háttérfolyamatok egyikét futtassa. A 2. és 3. csomópont szintén késlekedik. E minta megismétlődése jelentősen csökkentheti alkalmazás teljesítményét. A hatás nagyságrendje a korlátok felbukkanásának sűrűségétől és a processzorok számától függ.

Petrini és kollégái ezt a teljesítményvesztést egy az összenyomható fluidumok *Euler* típusú hidrodinamikai viselkedését leíró kód, a *SAGE* futtatásakor figyelték meg *ASCI Q* nevű *HPC* rendszerükön. Az *ASCI Q* egy 2,048 darab *HP ES45* csomópontból álló fürt, ahol minden csomópont maga is négy utas *SMP* rendszer. *Petrini* és társai megfigyelték, hogy amennyiben több mint 256 csomópontot bevontak a számításba, jobb teljesítményt értek el a *SAGE*-el, ha az *SMP* rendszereken a négyből csak három processzort futtattak. Feltételezték, hogy az eltérést a zaj okozza, majd elképzelésüket bizonyították a zajforrások egy részének kiküszöbölésével, amely érzékelhető teljesítménynövekedést okozott.

A kutatások rámutatnak, hogy a folyamat szinkronizálás hiánya és várakozási idő a bűnös, amely az egyébként finoman hangolt és erősen párhuzamosított *HPC* alkalmazások potenciális teljesítményének akár 50%-át (vagy még többet) is elrabolhatja. Sajnos az ilyen tolvajlás kiküszöbölése nem éppen kézenfekvő. A bűnös azonosítására *Petrini* és társai által alkalmazott módszer – a rendszer szabadságfokainak korlátozása a karbantartási munkák végzése terén – a legtöbb *HPC* alkalmazásban nem igazán járható út. A legtöbb *HPC* tulajdonos számára nem igazán kívánatos eljárás, hogy a rendszer processzorainak egynegyedét karbantartási munkák kössék le. Ráadásul a sok háttérfolyamatot nem lehet eltávolítani, így ezzel a megközelítéssel elérhető megtakarítás erősen korlátozott.



1. ábra Példa folyamatok szinkronizált és aszinkron végrehajtására

A hiányzó teljesítmény visszaszerzése

Ha rászánjuk magunkat egy új nagy teljesítményű számítógép összeállítására, valahogy ki kell találnunk, hogyan akadályozhatjuk meg ezt a teljesítménytolvajlást. Kidolgoztunk egy új eljárást, amely a ***Linux*** ütemező segítségével próbálja meg szinkronizálni az ***MPI*** és a karbantartási munkafolyamatok ütemezését. Korábbi munkáink és kutatásaink azt sugallják, hogy az új szinkronizált ütemezési megközelítés akár 50% vagy még komolyabb teljesítménynövekedést is okozhat egyes 32 vagy több processzoron futó, finom hangolt, párhuzamosított alkalmazásnál.

Szinkronizált ütemezési rendszabályok bevezetése

Elképzelésünk alapja egy új ***Linux*** ütemezési rendszabály bevezetése volt. A kívánt előnyök elérése érdekében a rendszabálynak a ***Linux HPC*** rendszer összes gépén szinkronizálnia kell az ütemezőket biztosítva, hogy az ***MPI*** folyamatok az összes folyamattal párhuzamosan futnak ugyanakkor a ***Linux*** karbantartási folyamatok is végrehajtódnak. Így az ütemezőnek valamilyen módon meg kell tudnia valósítani az 1. ábrán bemutatott globális szinkronizálást. A globális szinkron elérése érdekében a kommunikációért felelős processzorhoz terveztünk egy kiegészítést amely az összes feldolgozó csomóponton szinkronizálja az órát. Az új ***Linux*** ütemező rendszabály 128 hellyel rendelkező ütemezési keretet szab meg amelyből 120 az alkalmazás végrehajtására, nyolc pedig a karbantartási munkákra van fenntartva. A különböző processzorok ütemezői a globálisan szinkronizált órához igazíthatják saját ütemező munkájukat, amely a rendszer csomópontjai közt legfeljebb mikroszekundum alatti eltéréseket jelenthet. Bármely időpillanatban az összes processzor vagy a ***HPC*** alkalmazást futtat vagy a karbantartási folyamatokat végez (az 1. ábra alsó fele).
A folyamatszinkronizálás ilyen megközelítése nagy számú processzorra is jól skálázható, hiszen az ütemezési döntést minden csomópont maga végzi el. Az eljárás a korlátokra várakozó ***CPU*** ciklusok megszüntetésével jelentősen megnövelheti a megnyirbált teljesítményt.
A szinkronizált ütemezőt új rendszabályként készítettük el a ***Linux*** kernel ütemezőjéhez már korábban elkészült három létező rendszabály kiterjesztéseként. A ***Linux*** ütemező akkor lép működésbe ha a végrehajtott folyamat elakad, önként átadja a ***CPU***-t, illetve ha a processzor megszakítást érzékel vagy a 10 milliszekundumos időszelet lejár. Az ütemező a következő futtatandó folyamatot az ütemezési rendszabály alapján választja ki a folyamat jellege és fontossága alapján. Az új ütemezési rendszabállyal felvértezett ***Linux*** két valós idejű és a két hagyományos időosztásos folyamatokat támogató ütemezési rendszabály közül választhat. Ezek fontossági sorrendben a következők:

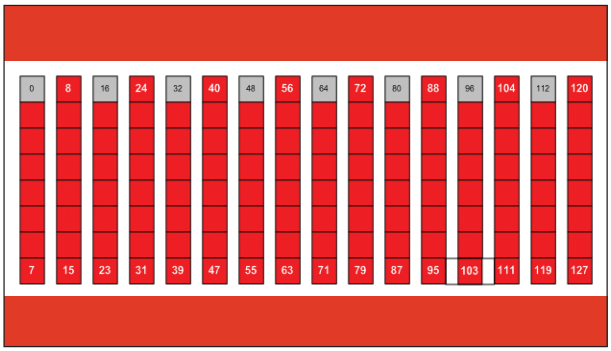
1. ***FIFO (első be, első ki)***: A ***FIFO*** jelölésű folyamat addig fut, míg el nem ereszti a ***CPU***-t. Ezt a prioritást csak rövid ideig használjuk a valós idejű rendszerfolyamatoknál. A ***FIFO*** folyamatok mindig elsőként futnak le.

2. ***Körbejáró (Round robin)***: Az ilyen rendszabályt használó folyamatok sorjában megkapják a 10 milliszekundumos időszeletüket. Valós idejű feldolgozáshoz használatos.

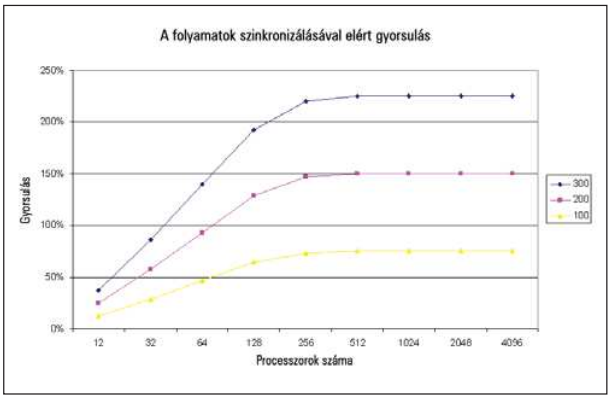
3. ***Szinkronizált***: A szinkronizált rendszabályt azért adtuk a rendszerhez, hogy a kötegelt többprocesszoros munkánál lehetővé tegyük a szinkronizált feldolgozást. A terheléskezelő rendszer minden folyamatot ezzel a rendszabállyal jelöl meg indításkor. A folyamatok és gyermekeik kihasználhatják a szinkronizált ütemezés előnyeit.

4. ***Előjogokon alapuló (Priority)***: Az előjogokon alapuló ütemezés nem más mint a ***Linux*** felhasználók számára jól ismert időmegosztási módszer. Azok a folyamatok melyek ezt a rendszabályt alkalmazzák előjogokkal rendelkeznek és előjogaiknak megfelelően kapnak időszeleteket. Lényegében minden felhasználói és rendszer folyamat ez alatt a rendszabály alatt fut. Az ütemező a legmagasabb fontossági szinten álló osztályból választja ki a következő futtatandó feladatot. A ***FIFO*** és körbejáró rendszerfolyamatok következnek legelőször. A szinkronizált végrehajtásra jelölt feladatok pedig előbb következnek mint a hagyományos előjogos ütemezőben megadottak.

Az új szinkronizált ütemező rendszabály létrehoz egy ütemező keretet, amely eldönti mikor lehet végrehajtani a kötegelt és az egyéb felhasználói illetve rendszer folyamatokat. A keretben előre megadott számú hely van, amelyek sorrendben egymás után következnek. Egy időhely10 milliszekundumot jelent, ***Linux*** alatt ugyanis ennyi egy „rendszerpillanat” (tick), ami alatt a időhelyhez rendelt folyamat futhat. Jelenlegi megvalósításunkban 128 időhely van, amelyből 120 a kötegelt munkák végrehajtására szolgál és 8-at tartunk fenn a rendszerfolyamatok részére. Ez utóbbi időhelyek alatt a szinkronizált ütemező jelzi, hogy nincs futtatandó kötegelt folyamat és így a karbantartási munkákhoz használható hagyományos ütemezési rendszabály lép érvénybe. Amennyiben nincs kötegelt munkafolyamat, a ***Cray*** ütemező megkülönböztethetetlen a megszokott ***Linux*** ütemezőtől.



2. ábra Időhelyek (128) nyolc fenntartott időhellyel



3. ábra Folyamatszinkronizálással elérhető elméleti gyorsulás 1.5% Dćmon CPU foglaltság esetén

Az ütemezési keretben található helyek száma állítható, azonban kötelező kettő hatványnak lennie. A többi folyamattal szemben a kötegelt folyamatok részére fenntartott mennyiség szintén beállítható. A 2. *ábra* egy tipikus ütemezési keretet mutat be, ahol a kötegelt munkák részére fenntartott rész vörös, a karbantartási munkák időhelyei pedig szürke színűek.

Az ütemező keret a csomóponton elindított első kötegelt folyamattal egy időben jön létre. Ilyenkor az összes kötegelt időhely ehhez a folyamathoz tartozik. A további kötegelt folyamatok hozzáadása során az időhelyek egyenletesen eloszlanak a folyamatok közt. Ha ***n*** számú kötegelt folyamatot hozunk létre, az első kötegelt folyamat a helyek 120/***n***-ed részét kapja, a második a következő 120/***n*** időszeletet és így tovább. Így a szinkronizált ütemező olyan kötegelt munkákat is ki tud szolgálni a processzorokon amelyek egynél több folyamatot igényelnek.

A kötegelt folyamat a megkapott idő végéig futhat, feltéve, hogy nincs akadályozva és nincs ***CPU***-t átadó rendszerhívás. Amennyiben a kötegelt folyamat átadja a ***CPU***-t, például mert blokkoló rendszerhívást végez, egy másik kötegelt folyamat kerül végrehajtásra. Amennyiben nincs futtatható kötegelt munka, a vezérlés a hagyományos ütemezőhöz kerül ahol a karbantartási feladatok lefuthatnak. Természetesen a kötegelt folyamat visszakapja a ***CPU***-t amint a megszakítás kezelése után kioldódik.

Ütemezési keretek kiosztása a processzorok közt

Ez idáig kizárólag az egy csomóponton végzett kötegelt munka és rendszerfolyamat ütemezéssel foglalkoztunk.

A teljesítmény elorzásának megakadályozásához, az ütemezőnek valamennyi processzort be kell vonnia. Itt egy rendkívül fontos tervezési feladatba botlunk, amely végső soron lehetővé teszi a szinkronizált ütemezést, nevezetesen a globális időszinkronizálás kérdésébe. Megoldásunkban a globális időszinkronizálást a ***HPC*** rendszerbe tervezett kommunikációs processzorok végzik. Ezek a processzorok a kommunikációs feldolgozás terhét veszik le a többi alkalmazás processzorról. A kommunikációs processzorok az egységes időzítés érdekében egyúttal időszinkronizációs feladatokat is ellátnak. A pontos időszinkronizációt azért érhetjük el, mert a kommunikációs processzorok vezérelni tudják a csomagidőzítést és ugrálást, és így bármely két processzor közötti időeltérés kevesebb lesz mint 1 mikroszekundum. Azzal, hogy a szinkronizációs feladatokat a kommunikációs processzorokkal végeztetjük, további előnyökhöz jutunk, hiszen az alkalmazás terhelést kiszolgáló processzorokról leveszünk bizonyos mennyiségű munkát, így több idő marad az alkalmazás kiszolgálására valamint a megszakítások száma is csökken az alkalmazás processzorokon.

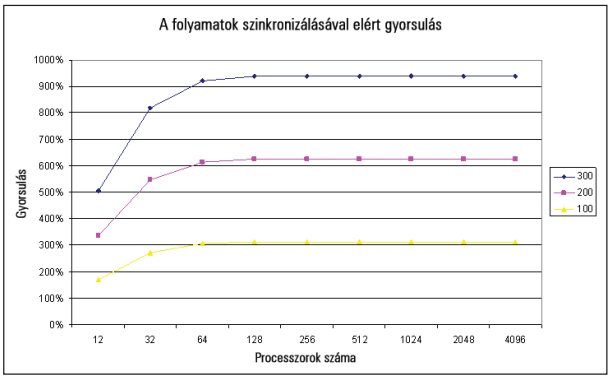
Az idő szinkronizációs protokoll kiegészítő helyeket tartalmaz az időhely kiosztáshoz. A protokoll mester szolga elgondolás szerint működik, ahol az egyik csomópont szolgál idő és időhely információ forrásként az összes többi csomópont pedig a mester csomópont órájához igazodik. A mestertől kapott időszinkronizálási csomagok azonosítják a végrehajtandó időhelyet és az időhely indítása óta eltelt időt, így a teljes ***HPC*** rendszerben precízen megadhatóak az ütemező keretek.

Teljesítmény kérdések

A szinkronizált ütemező a párhuzamos alkalmazások folyamatainak szinkronizált végrehajtását teszi lehetővé. Hogy mekkora teljesítménycsökkenést lehet elkerülni, vagy mekkora potenciális teljesítmény érhető el, az az alkalmazás által használt korlátok és gyűjtőműveletek számától, a rendszer karbantartó folyamatok által elhasznált időmennyiségtől és az alkalmazásban felhasznált processzorok számától függ.

Kutatásaink azt mutatják, hogy a módszerrel jelentős gyorsulást lehet elérni. A 3. és 4. *ábra* a szinkronizált ütemező által elérhető elméleti gyorsulást mutatja a hagyományos előjogos ütemezővel szemben. A 3. ábrán feltételeztük, hogy a háttérfolyamatok a processzor 1.5%-át foglalják le, a 4. ábra szerint pedig a háttérfolyamatok a ***CPU*** 6.25%-át igénylik – ezek egyébként a legtöbb valódi fürtön tapasztalható értékek. A bemutatott görbék másodpercenkénti 100, 200 és 300-as korlátszámnak felelnek meg.

Ahogy a processzorok száma növekszik, a szinkronizált ütemező nyújtotta előnyök aszimptotikusan közeledik a maximum felé. Ez azt a tényt tükrözi, hogy a hagyományos ütemezővel sem csökken akármeddig a teljesítmény. Bizonyos processzorszám elérése után annak a valószínűsége, hogy legalább egy processzor karbantartási munkát végez, közelít a 100%-hoz. További processzorok hozzáadása lényegesen már nem növeli a korlátoknál észlelhető alkalmazás csúszást.



4. ábra Folyamatszinkronizálással elérhető elméleti gyorsulás 6.5% Dćmon CPU foglaltság esetén

Összefoglalás

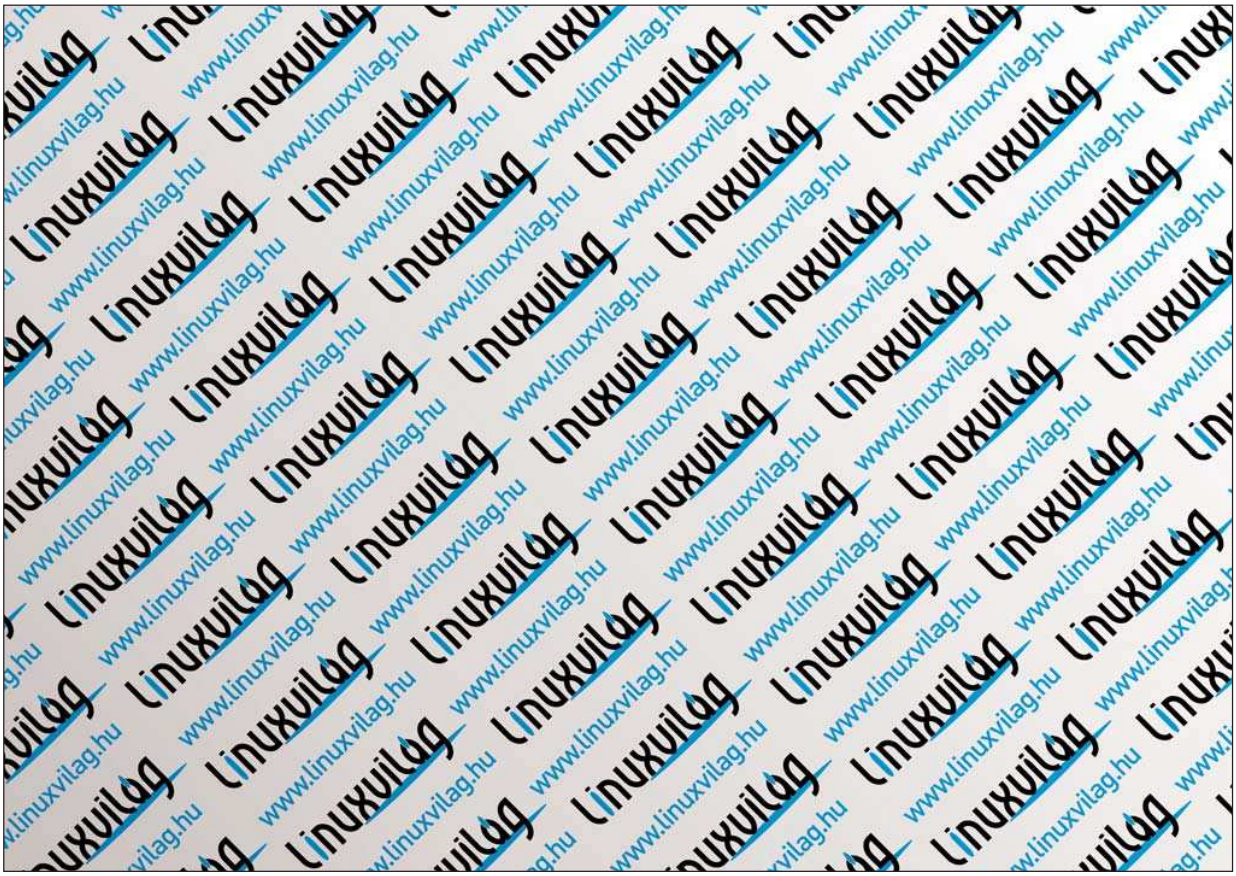
A *HPC* alkalmazás és a rendszer háttérfolyamatai közötti kölcsönhatásra fokuszálva, a *HPC* kutatók rátaláltak a párhuzamos alkalmazások teljesítménycsökkenésének egyik fő okára. További kutatások e tolvajlás megakadályozásának lehetőségére is fényt derítettek, de ez egyelőre még egyetlen valódi működő megoldás sem létezik. A *Linux* ütemezőjét használó globális folyamat szinkronizálás megszünteti a zaj várakozási időt és jelentős teljesítménynövekedéssel kecsegtet. A *HPC* rendszer egyéb szabályai és az alkalmazáson túllépve úgy hisszük, rábukkantunk egy valódi, komolyan használható megoldásra. Globális óraszinkronizálást alkalmazó *Linux* folyamatszinkronizálással és az egyes csomópontokon futtatott *Linux* rendszerekkel a *Cray* megoldása az összes processzoron képes biztosítani az alkalmazás folyamatainak párhuzamos futását miközben a karbantartási folyamatok kötött időben, szintén párhuzamosan futnak le. Folyamatszinkronizáló megoldásunk megelőzheti a teljesítményvesztést és akár 50%-al is megnövelheti a finom felbontású erősen párhuzamosított alkalmazások teljesítményét a 32 vagy több processzort alkalmazó rendszereken.

Linux Journal 2004. november, 127. szám

Dr. Paul Terry a Cray által 2004 áprilisában felvásárolt, korábban OctigaBay Systems néven ismeretes, Cray Canada vezető műszaki szakembere. Az új számítási szerkezetek technológiai stratégája, aki a cég műszaki elképzeléseinek és vezető helyzetének megalapozásáért felelős.

Amar Shan a Cray termelésirányítási igazgatója, a Cray élvonalbeli műszaki újításaiért és kreatív üzleti megoldásaiért felelős. Több mint 20 éves tapasztalattal rendelkezik a számítási és telekommunikációs ipar termékmenedzsmentje, fejlesztése és architektúra szabályai terén.

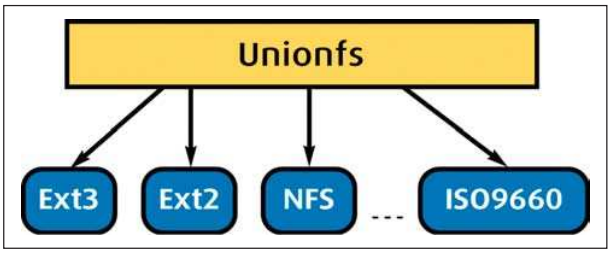
Pentti Huttunen a Cray teljesítménymérési szakértője aki a párhuzamos számítási eljárások kidolgozásáért és az alkalmazások optimalizálásért felelős. Munkájának fő célja az, hogy ezek az alkalmazások a Cray különféle gépein a lehető leghatékonyabban fussanak.



Unionfs: eggyé váló fájlrendszerek

Az Unionfs több könyvtárat is képes egyetlen nézetben egyesíteni. A továbbiakban az Unionfs alkalmazásait, valamint működésének néhány érdekes kérdését tárgyaljuk.

Az egymáshoz kapcsolódó, ám különböző fájlokat a könnyebb kezelés miatt sokszor érdemes elkülönítve tárolni. A felhasználók viszont inkább együtt szeretnék látni őket. Ilyenkor a rendszergazda egy egyesítéssel elérheti, hogy fizikailag elkülönítve maradjanak, logikailag azonban mégis egyetlen nézetben legyenek elérhetők a fájlok. Az egybevont könyvtárak halmazát uniónak nevezzük, minden fizikai könyvtár ennek egy ága. Amint az *1. ábrán* is látható, a ***Unionfs*** egyszerre több fájlrendszer felett vagy adott fájlrendszer több könyvtára felett alkot újabb réteget. Ezt a rétegezéses megoldást vermelésnek nevezzük, az internetes források között bővebben is lehet olvasni róla. A ***Unionfs*** a rendszermag felé egy fájlrendszer felületet mutat, míg az alatta üzemelő fájlrendszerek őt a rendszermag ***VFS***-ének látják. Mivel a ***Unionfs*** fájlrendszerként látszik a rendszermag számára, bármely felhasználói alkalmazás, illetve a rendszermag felől az NFS-kiszolgáló is igénybe veheti szolgáltatásait. A ***Unionfs*** elfogja az alacsonyabb szintű fájlrendszerekkel végzett műveleteket, és ezek módosításával képes az egységes nézet biztosítására. A korábbi vermelhető fájlrendszerekkel ellentétben a ***Unionfs*** valóban legyezőként terült szét az alsóbb rétegek felett, és nagy számú alsóbb szintű ágat képes közvetlenül elérni.



1. ábra Egy unió számos alsóbb szintű ágból áll, amelyek tetszőleges típusú fájlrendszerrel üzemelhetnek

Az Unionfs szemantikája és használata

Unionfs alatt minden ágnak van egy elsőbbségi szintje, precedenciája. A magasabb elsőbbségi szintű ágak elnyomják az alacsonyabb szintűeket. A ***Unionfs*** könyvtárakkal dolgozik. Ha egy könyvtár két ágban is megtalálható, akkor a ***Unionfs***-ben megjelenő könyvtár tartalma és jellemzői a két könyvtár kombinációjából tevődnek össze. A ***Unionfs*** magától eltávolítja a kettős könyvtárbejegyzéseket, így a felhasználókat nem zavarják meg a kettős fájl- és könyvtárnevek. Ha egy fájl két ágban is jelen van, akkor a ***Unionfs*** alatt látható fájl tartalma és jellemzői a magasabb elsőbbségi szintű ágban találhatóval egyeznek meg, az alacsonyabb szintű ágban lévő fájlt a rendszer elnyomja.
Példaként tegyük fel, hogy két könyvtárat egyesítünk, ezek a */**Gyümölcsök*** és a */**Zöldségek***:

```
$ ls /Gyümölcsök
Alma Paradicsom
$ ls /Zöldségek
Répa Paradicsom
$ cat /Gyümölcsök/Paradicsom
Növénytanilag gyümölcs vagyok.
$ cat /Zöldségek/Paradicsom
A kertészek zöldségként kezelnek.
```

A ***Unionfs*** használatba vételéhez először le kell fordítanunk a ***Unionfs*** modult, majd be kell töltenünk. Következő lépésként, mint minden más fájlrendszert, a ***Unionfs***-t is be kell fűzni. Az egyéb fájlrendszerekkel ellentétben a ***Unionfs*** nem egy eszköz, hanem a befűzésekor megadott könyvtárak fölé helyezkedik el. Unió létrehozásához a következőképpen kell befűznünk a ***Unionfs***-t:

```
# mount -t unionfs -o dirs=/Gyümölcsök:/Zöldségek
➥> none /mnt/egészséges
```

A fenti példában a befűzéskor átadott `dirs` értékkel adjuk meg az uniót alkotó könyvtárakat. Mivel a ***Unionfs*** eszközt, meghajtót nem használ, a `none` (semmi) helyőrzőt használjuk. Utolsóként a `/mnt/egészséges` értéket adjuk meg, amely az egyesített nézet helye. Most a */**mnt/egészséges*** könyvtár három fájlt tartalmaz: `Alma`, `Répa` és `Paradicsom`. Mivel a `/Gyümölcsök` könyvtárat a `/Zöldségek` előtt adtuk meg, a `/mnt/egészséges/Paradicsom` fájl tartalma a „`Növénytanilag gyümölcs vagyok`”. Ha a `dirs=` átadott érték elemeit megfordítjuk, akkor a `/mnt/egészséges/Paradicsom` tartalma az „`A kertészek zöldségként kezelnek`” lesz. (Egyébként ez utóbbi felel meg az ***USA Legfelsőbb Bírósága*** a témában 1893-ban hozott döntésének.)
A folyamat rekurzív. Ha a `Gyümölcsök` könyvtárban lenne

egy Zöld nevű alkönyvtár, amely tartalmazná a Zöldcitrom fájlt, a Zöldségek könyvtárban pedig volna egy szintén Zöld nevű alkönyvtár, benne a Saláta fájllal, az eredmény a következő lenne:

```
$ ls /mnt/egészséges
Alma Répa Zöld/ Paradicsom
$ ls /mnt/egészséges/Zöld
Zöldcitrom Saláta
```

A *Unionfs* számos különböző célra használható. Alkalmas például arra, hogy több kiszolgálóról egyesítsük a kezdőkönyvtárakat, vagy több *ISO* képfájlt összevonva egységes képet alkossunk egy terjesztésről. Hasonlóan, az *Unionfs* másolás írás közben szemantikával alkalmas CD-lemezek tartalmának foltozására, amivel forráskód kezelésére vagy pillanatfelvételek készítésére is megfelel.

Egyesített kezdőkönyvtárak

Egy-egy ügyfélgép gyakran több különböző *NFS*-kiszolgálóról fűzi be a kezdőkönyvtárat. Szerencsétlen megoldás, hiszen ilyenkor mindegyik kiszolgálóhoz külön befűzési pont tartozik, ami kényelmetlen a felhasználó számára. A legjobb az lenne, ha mindegyik kezdőkönyvtárat ugyanarról a helyről, például a */home* könyvtárból lehetne elérni. Az automatikus befűzők egy része szimbolikus hivatkozásokat használ, az egyesítés látszatát keltve. *Unionfs* alatt ilyen hivatkozásokra nincs szükség. Két különálló könyvtár teljesen egyszerűen egyesíthető egyetlen nézetben. Tegyük fel, hogy két fájlrendszerünk van, ezek /alcid és /pingvin névvel vannak befűzve. A */home* könyvtárban a következő módon egyesíthetjük őket:

```
# mount -t unionfs -o dirs=/alcid,/pingvin
➥> none /home
```

Így a /alcid és a /pingvin kezdőkönyvtára egyaránt elérhető a /home könyvtárból.
Az Unionfs támogatja a többes olvasási-írási ágakat, vagyis a felhasználói fájlokat nem kell egyik könyvtárból átmozgatni a másikba. Ebben is eltér a korábbi egyesítő megoldásoktól, például a 4.4-es *BSD*-ben található *Union Mounts* rendszertől, ezek általában csak egy olvasási-írási ágat támogatnak.

Terjesztések ISO képfájljainak egyesítése

A legtöbb *Linux*-terjesztés *ISO* képfájlok és különálló csomagok formájában egyaránt elérhető. Az *ISO* képfájlok kényelmes megoldást nyújtanak, hiszen közvetlenül fel lehet írni őket CD-lemezre, letölteni és külön tárolni csak néhány fájlt kell. Ha viszont hálózaton keresztül kell telepítenünk egy gépet, sokszor jó volna, ha az egyes csomagokat egyetlen könyvtárban látnánk. A hurokeszköz segítségével az *ISO* képfájlokat ugyan be tudjuk fűzni különböző könyvtárakba, ám a hálózati telepítéshez ez az elrendezés nem felel meg, hiszen az összes fájlnak egyetlen faszerkezetben kell lennie. Éppen ezért sok helyen az *ISO* képfájlokból és az egyes csomagfájlokból egyaránt fenntartanak egy másolatot, ami viszont a tárhely és a sávszélesség pazarlását, valamint a felügyeleti feladatok szaporodását eredményezi.

A *Unionfs* segítségével egy könnyed huszárvágással oldhatjuk meg a problémát, ha képzetesen egyesítjük az *ISO* képfájlok csomagokat tartalmazó könyvtárait.
Az alábbi példában két könyvtárat fűzünk be, ezek a /mnt/lemez1 és a /mnt/lemez2. A befűzési parancs a következő:

```
# mount -t unionfs -o dirs=/mnt/lemez1,/mnt/lemez2
➥> none /mnt/egyesített-terjesztés
```

Ezután a /mnt/egyesített-terjesztés könyvtárat *NFS*-en vagy *FTP*-n keresztül használhatjuk a hálózati telepítésekhez.

Másolás írás közben típusú uniók

Az iménti példában az unió minden ága csak olvasható volt, ennél fogva magát az uniót is csak olvasni lehetett. A *Unionfs* képes csak olvasható és írható-olvasható ágak egyesítésére is. Ilyenkor az unió maga írható-olvasható lesz, és az *Unionfs* másolás írás közben megoldást használva annak a látszatát kelti, hogy a csak olvasható ágak fájljainak és könyvtárainak módosítására is van lehetőségünk. Ezzel a módszerrel például CD-lemez tartalmát tudjuk frissíteni, foltozni. Ha a CD-ROM-meghajtót a /mnt/cdrom alá fűztük be, valamint van egy /tmp/cdfolt könyvtárunk is, akkor a következő parancsot adjuk ki:

```
# mount -t unionfs -o dirs=/tmp/cdfolt,/mnt/cdrom
➥> none /mnt/foltozott-cdrom
```

A /mnt/foltozott-cdrom könyvtáron keresztül nézve úgy tűnik, mintha írni lehetne a CD-lemezt, de természetesen az írások a /tmp/cdfolt könyvtárba történnek. Egy csak olvasható ág írása egy fölémásolásnak nevezett műveletet eredményez. Ha egy csak olvasható fájlt írásra nyitunk meg, akkor a fájlrendszer átmásolja egy magasabb elsőbbségi szintű ágba. Szükség esetén a *Unionfs* magától létrehozza a szülőkönyvtárak szerkezetét is.
A CD-lemezes példában az alacsonyabb szintű fájlrendszer kötelező érvénnyel betartja a csak olvashatóságot, az *Unionfs* pedig figyelembe veszi ezt. Lehetnek olyan esetek is, amikor az alacsonyabb szintű fájlrendszer ugyan írási-olvasási hozzáférést enged, ám az *Unionfs* számára meg akarjuk tiltani az ág módosítását. Lehet például egy águnk, amely eredeti rendszermag forráskódokat tartalmaz, és dönthetünk úgy, hogy emellett egy másik ágat használunk a helyi módosítások tárolására. A *Unionfs*-en keresztül az eredeti ágat csak olvashatóvá is tehetjük, hasonlóan az előző példa CD-lemezéhez. Ehhez csak az =ro kapcsolót kell hozzáadjunk a dirs befűzési átadott értékhez. Tegyük fel, hogy a /home/cpw/linux könyvtár üres, és a /usr/src/linux tartalmazza a Linux rendszermag forráskódjának könyvtárait. Az alábbi paranccsal a *Unionfs*-t forráskód-változatkövető rendszerré alakíthatjuk:

```
# mount -t unionfs -o
➥> dirs=/home/cpw/linux:/usr/src/linux=ro
➥> none /home/cpw/linux-src
```

A parancs kiadása után úgy fogjuk érzékelni, mintha egy teljes *Linux* forráskód lenne a /home/cpw/linux-src

könyvtárban, ám bármilyen módosítást is hajtunk végre rajta, a megváltozott vagy új fájlok valójában a `/home/cpw/linux` könyvtárba kerülnek.
Egy egyszerű módosítással elfedő befűzést is választhatunk, ekkor a `/home/cpw/linux` könyvtárat egységes nézetben látjuk:

```
# mount -t unionfs -o
➥> dirs=/home/cpw/linux:/usr/src/linux=ro
➥> none /home/cpw/linux
```

Megvalósítás

Unionfs alatt a legtöbb fájlrendszerbeli művelet a magasabb elsőbbségi szintű ágak felől az alacsonyabb elsőbbségi szintűek felé halad. A ***LOOKUP*** (keresés) például a szülőt tartalmazó legmagasabb elsőbbségi szintű ággal kezd, innen halad az alacsonyabb szintűek felé. A keresések alatt az ***Unionfs*** a későbbi műveletek céljaira gyorstárazza az adatokat.
A ***CREATE*** (létrehozás) a szülőkönyvtárat tartalmazó legmagasabb elsőbbségi szintű könyvtárban kísérli meg a fájl létrehozását. A ***CREATE*** művelet a gyorstárazott keresési adatokra támaszkodik, így biztosítva, hogy közvetlenül a megfelelő ággal dolgozzon, így lényegében a magasabb ágak felől mozog az alacsonyabbak felé.
A ***Unionfs*** számos módszert alkalmaz a csak olvasható ágak látszólagos módosíthatóságának fenntartására, miközben a közönséges, ***UNIX***-ra jellemző szemantikát is megőrzi. Ha egy fájl létrehozása közben hiba történik, akkor hibakezelést kell végezni. A hibakezelés a legalacsonyabb elsőbbségi szintű ág felől a magasabbak felé halad. A szülőt tartalmazó legmagasabb elsőbbségi szintű ággal kezdve az ***Unionfs*** mindegyik magasabb szintű ágban megkísérli a fájl létrehozását. Végül, ha a művelet a legmagasabb szintű ágban a teljes unióra nézve sikertelen, az ***Unionfs*** hibát jelez a felhasználó felé.
A ***CREATE*** művelettel ellentétben az ***UNLINK*** (leválasztás) mindig a legalacsonyabb elsőbbségi szintű ágtól halad a legmagasabb felé. Mivel az utolsó leválasztandó objektum a legmagasabb fontossági szintű objektum, a felhasználó szemszögéből semmi sem változik, amíg az ***Unionfs UNLINK*** művelete teljesen be nem fejeződik. A legbonyolultabbak a részleges hibák. Ha egy köztes fájlt nem sikerül eltávolítani, és az ***Unionfs*** egyszerűen törli a legmagasabb fontossági szintű fájlt, akkor az alacsonyabb fontossági szintű fájl válik láthatóvá a felhasználó számára. A hibahelyzetek kezelésére az ***Unionfs*** egy különleges, magas elsőbbségi szintű fájlt használ, ennek neve ***whiteout*** (kb. kihagyás). Ha az ***Unionfs*** kihagyás fájllal találkozik, akkor úgy viselkedik, mintha a kérdéses fájl egyik alacsonyabb elsőbbségi szintű ágban sem létezne. Az ***Unionfs*** belső folyamatai például az ***F*** nevű fájlhoz a kihagyás fájlt egy nulla bájtos ***.wh.F*** nevű fájl formájában hozzák létre.
Visszatérve a leválasztás művelethez: ha egy köztes leválasztás végrehajtása sikertelen, akkor a legmagasabb elsőbbségi szintű fájlt az ***Unionfs*** nem törli, hanem a megfelelő kihagyás névre kereszteli át.
A műveletek gondos sorrendezése két dolgot eredményez. Az első, hogy a unixos szemantika még hibák felmerülésekor és csak olvasható ágak kezelésekor is sértetlen marad. A felhasználó által látott állapot mindaddig érintetlen marad, amíg a művelet el nem ér a legmagasabb elsőbbségi szintű ágig. A művelet sikerességét vagy sikertelenségét az ebben az ágban kapott eredmény határozza meg. A kihagyás fájlok használatával adott fájlt még akkor is lehet törölni, ha az csak olvasható ágban található. A második fontos hatás, hogy ha nem történik hiba, akkor a fájlok és könyvtárak azokban az ágakban maradnak, ahol eredetileg is voltak. Ez fontos szempont, hiszen a ***Unionfs*** egyik fő célja a fájlok különböző helyeken tartása.

A fájlok törlésének szemantikája

Alapesetben az ***Unionfs*** az összes ágból törli a megadott fájl vagy könyvtár példányait, ezt a módot ***DELETE_ALL*** (teljes törlés) módnak nevezzük. A teljes törlés mellett az ***Unionfs*** további két törlési módot is támogat, ezek a ***DELETE_WHITEOUT*** (kihagyás törlése) és a ***DELETE_FIRST*** (első törlése). A kihagyás törlése kívülről szemlélve az alapmódhoz hasonlóan működik, ám az unió összes vonatkozó fájljának törlése helyett egy kihagyás fájlt hoz létre. Ennek az az előnye, hogy az alacsonyabb elsőbbségi szintű fájlok az alsóbb szintű fájlrendszereken keresztül elérhetők maradnak. Az első törlése mód szakít a hagyományos unixos szemantikával, és csak az unió legmagasabb elsőbbségi szintű bejegyzését távolítja el, aminek eredményeként az alacsonyabb elsőbbségi szintű bejegyzések láthatókká válnak. Mindezeket a módokat a ***RENAME*** (átnevezés) művelet is segítségül hívja, ez ugyanis egy létrehozásból és egy azt követő törlésből áll.
Az első törlése művelet használatához a felhasználónak ismernie kell az unió összetevőit. A mód elsősorban akkor használható jól, ha a ***Unionfs***-t a korábbi rendszermag forráskódfás példához hasonlóan forráskód változatainak követésére használjuk. Ha a ***/home/cpw/linux*** könyvtárban megváltoztatunk egy fájlt, akkor a rendszer felmásolja a magasabb elsőbbségi szintű ágba. Ha a fájlt a normál teljes törléssel távolítjuk el, az ***Unionfs*** egy kihagyás fájlt hoz létre a legmagasabb elsőbbségi szintű ágban – a csak olvasható alacsonyabb elsőbbségi szintű ágat ugyanis nem tudja módosítani. Az eredeti, az alacsonyabb elsőbbségi szintű ágban lévő forrásfájl ezzel elérhetetlenné válik, így a forrásból kell bemásolni az unióba, ez viszont aligha fogadható el egy változatkövető rendszertől. Pontosan ezek azok a helyzetek, amikor az első törlése mód jól jön. A törlési módot befűzéskor adhatjuk meg, például:

```
# mount -t unionfs -o
➥> dirs=/home/cpw/linux:/usr/src/linux=ro,
➥> delete=first none /home/cpw/linux
```

Most, ahogy korábban is, ha módosítunk egy a `/home/cpw/linux` könyvtárban lévő fájlt, a `/usr/src/ linux` könyvtár tartalma érintetlen marad. Ha meggondoljuk magunkat, és nem akarjuk megtartani a változásokat, akkor egyszerűen töröljük a fájlt, ekkor újra láthatóvá válik az eredeti változat.

Unionfs pillanatfelvételek

Az ***unionctl*** segédprogrammal az ***Unionfs*** ágait üzem közben is meg lehet változtatni. Az unió bármelyik pontjára új ágat csatolhatunk, tetszőleges ágat kivehetünk, vala-

mint írható-olvasható ágakat csak olvashatóvá változtathatunk, illetve fordítva. Rugalmasságának köszönhetően az ***Unionfs*** a fájlrendszer pillanatfelvételének elkészítésére is használható. A következő példában a `/usr` könyvtárról készítünk pillanatfelvételt, miközben telepítünk egy új csomagot:

```
# mount -t unionfs -o dirs=/usr none /usr
```

Ezen a ponton a ***Unionfs*** egyetlen írható-olvasható ággal rendelkezik, és ez a `/usr`. Minden művelet továbbkerül az alsóbb szintű fájlrendszerhez, minden úgy zajlik, mintha a ***Unionfs*** jelen se lenne.
A pillanatfelvétel elkészítése két lépésből áll. Az első a pillanatfelvétel fájlok helyének megadása, amely egy ág – ez legyen például a `/pillfelv/0` – hozzáadásával történik:

```
# unionctl /usr –add /pillfelv/0
```

Ekkor a ***Unionfs*** a `/usr` könyvtárba szánt új fájlokat a `/pillfelv/0` könyvtárban hozza létre, miközben a `/usr` könyvtár alkönyvtáraiba mentett fájlok továbbra is a `/usr` könyvtár alá kerülnek. A látszólagos ellentmondás abból a szabályból fakad, hogy a fájlokat a legmagasabb elsőbbségi szintű olyan ágban kell létrehozni, amelyben a szülő létezik. Az unió gyökérkönyvtárának, vagyis a `/usr` könyvtárnak a fájljai esetében a szülő mindkét ágban létezik. Mivel a `/pillfelv/0` a magasabb elsőbbségi szintű ág, az új fájlok és könyvtárak fizikailag a `/pillfelv/0` könyvtárban jönnek létre. A `/pillfelv/0` viszont üres, ezért, ha létrehoznánk egy fájlt a `/usr/local` könyvtárban, akkor a legmagasabb elsőbbségi szintű szülő a `/usr` ágban lenne.
Az áttelepítés teljessé tételéhez az eredeti `/usr` ágat csak olvashatóvá kell tenni. Ismét az `unionctl`-t használjuk az ágak módosítására:

```
# unionctl /usr –mode /usr ro
```

Most, köszönhetően annak, hogy az ***Unionfs*** a `/usr` könyvtárat csak olvashatónak látja, minden írási művelet ténylegesen a `/pillfelv/0` könyvtárra irányítódik. Több pillanatfelvételt is készíthetünk, ha létrehozunk egy újabb ágat, például `/pillfelv/1` névvel, és a `/pillfelv/0` ágat csak olvashatóként jelöljük meg.
Az első pillanatfelvételt a `/usr` könyvtáron keresztül látjuk. Mindegyik pillanatfelvétel egy alapkönyvtárból és további, az eltéréseket növekményi formában tartalmazó könyvtárakból áll. Ha megadott pillanatfelvételre vagyunk kíváncsiak, akkor csupán egyesítenünk kell az első pillanatfelvételt és a növekményes változásokat. Ha például a `/usr` és a `/pillfelv/0` könyvtárak tartalmából összeálló pillanatfelvételt akarjuk elérni, az ***Unionfs***-t a következőképpen kell befűznünk:

```
# mount -t unionfs -o ro,dirs=/pillfelv/0:/usr
➥> none /mnt/pillfelvétel
```

Ebben az esetben az ***Unionfs*** maga is csak olvashatóként kerül befűzésre, az alsóbb szintű könyvtárakat tehát nem lehet módosítani.

A pillanatfelvételen végrehajtott módosítások helyességének ellenőrzése után a következő lépés gyakran a pillanatfelvétel beolvasztása az alapba. A ***Unionfs*** letölthető csomagjában egy `snapmerge` nevű parancsfájl is szerepel, ez – a pillanatfelvétel könyvtárában lévő fájlokat rekurzívan az alapkönyvtárba másolva – a növekményes ***Unionfs*** pillanatfelvételeket képes összeolvasztani az alapkönyvtárral.
A másolási eljárás befejezése után az új és a módosított fájlok hiánytalanul rendelkezésünkre állnak. A végső lépés a fájltörlések kezelése, amely a kihagyás fájlok listájának létrehozásával és a megfelelő fájlok törlésével történik. Maguk a kihagyás fájlok szintén törlésre kerülnek, így nem terhelik feleslegesen a fát.

Összefoglalás

A ***Unionfs*** rekurzívan egyesít akár nagyobb számú könyvtárat vagy ágat is egyetlen képzetes nézetben. Hatékony, legyezőszerű szerkezete révén a ***Unionfs*** számos alkalmazási területen állja meg helyét. A ***Unionfs*** például ***ISO*** képfájlok egyesítésére, összevont ***/home*** könyvtár biztosítására és rengeteg további célra használható. Az ***Unionfs*** másolás írás közben szemantikájának köszönhetően változatkövetésre, pillanatfelvételek készítésére és CD-lemezek foltozására is alkalmas. Az ***Unionfs*** teljesítményét 2.4-es ***Linux*** alatt vizsgáltuk. Fordítási feladatoknál egynégy ág kezelésekor az ***Unionfs*** miatti többletterhelés mindössze egy-két százalék volt. A nagy mennyiségű be- és kiviteli műveletet végző folyamatoknál a többletterhelés egy ág esetén 10, négy ág esetén pedig 12 százalék körüli volt.

Köszönetnyilvánítás

Hálával tartozom ***Puja Gupta***, ***Harikesavan Krishnan***, ***Mohammad Zubair*** és ***Jay Dave*** barátomnak, akik szintén a ***Unionfs*** fejlesztői csapatába tartoznak. Külön szeretnék köszönetet mondani ***Mohammadnek***, amiért segített összeállítani az írásom alapjául szolgáló szoftverkörnyezetet.

Linux Journal 2004. december, 128. szám

Charles P. Wright (cwright@cs.stonybrook.edu) a Stony Brook Egyetem informatikus doktorandusz hallgatója. Charles operációs rendszerekkel kapcsolatos kutatásokat vezet, amelyek fő témái a fájlrendszerek, a biztonság és a bővíthetőség. A Stony Brook Linux Felhasználói Csoport (LUGSB) aktív tagja.

Erez Zadok (ezk@cs.stonybrook.edu) a Stony Brook Egyetem informatikai karán dolgozik, a Linux NFS and Automounter Administration (Sybex, 2001) című könyv szerzője, a FiST vermelhető sablonrendszer létrehozója és karbantartója, továbbá az Am-utils (más néven Amd) automatikus befűző fő karbantartója. Erez operációs rendszerekkel kapcsolatos kutatásokat vezet, ezek fő témái a fájlrendszerek, a biztonság, a sokoldalúság és a hordozhatóság.

Postfix kiszolgáló bővítése Clam Antivirussal

Linuxos levélkiszolgálónkat nagyteljesítményű védelmi vonallal felszerelve megvédhetjük a bajtól a sérülékeny rendszereket.

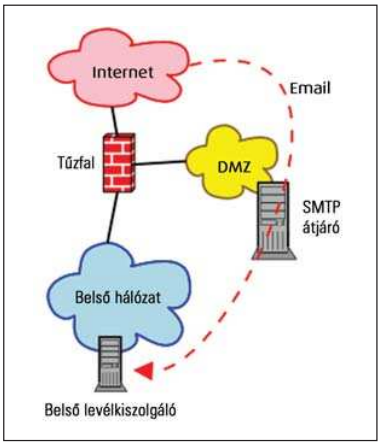
A *Linux Journal* 2004-es *Szerkesztői Díját* a biztonsági eszközök között a *ClamAV*, egy teljes mértékben szabad és nyílt forrású víruskereső kapta, amely ugyan jellemzően Linuxon fut, ám számos különböző géptípus vírusait képes felismerni. (Lásd: ***Linuxvilág,*** 2004. szeptemberi szám) Mint ***Reuven Lerner*** a díjakról szóló cikkben megjegyezte, a ***„ClamAV miatt az üzleti víruskereső programok tényleg futhatnak a pénzük után”***.
Ez alkalommal szeretném megmutatni, hogy ***Postfix*** alapú elektronikus levél átjárónkon hogyan használhatjuk ki a *ClamAV* tudását. Eközben szó esik majd az ***Amavisd-new***-ről, erről a sokoldalú, elektronikus levelek feldolgozására használható démonról, amely létfontosságú összekötőként szolgál a levélkiszolgálók, mint a ***Postfix*** és a ***Sendmail***, valamint a leveleket ellenőrző eszközök, mint a *ClamAV* és a ***SpamAssassin*** között.

A rendszer felépítése

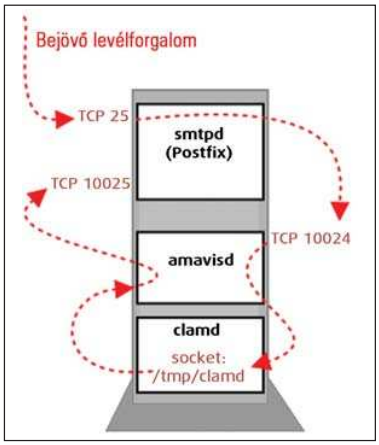
A továbbiakban ismertetendő összeállítás természetesen nem az egyetlen lehetséges vagy jó lehetőség a *ClamAV* használatára. Ez ugyanakkor a legelterjedtebb megoldás – mondjuk úgy, jellegzetesnek nevezhető. Tegyük fel, van egy ***SMTP*** átjárónk, az internet felől ez fogadja a saját cégünk, szervezetünk számára címzett elektronikus leveleket, a célunk pedig az, hogy az ***SMTP*** átjáró előzetesen ellenőrizze, a levelek nem tartalmaznak-e vírust. (*1. ábra*) Az átjáró feladata lehet az, hogy a leveleket a helyi postaládákban helyezze el, de belső levélkiszolgálónak való továbbításra is beállíthatjuk. A következőkben foglaltak függetlenek a tényleges levéltovábbítási módszertől.
Ha nagy forgalmú rendszert üzemeltetünk, akkor lehetséges, hogy az ***SMTP*** átjáró helyett a víruskeresést inkább egy különálló géppel érdemes elvégeztetnünk – az itt ismertetett eszközök ekkor is működni fognak. Az egyszerűség kedvéért azonban, illetve igazodva a szokásokhoz, most tegyük fel, hogy a víruskereső magán az ***SMTP*** átjárón fut. ***Levéltovábbító ügynökként (Mail Transfer Agent, MTA)*** ***Postfixet*** használunk, ez ugyanis egy népszerű és biztonságosan futtatható program, továbbá a *ClamAV*-vel is gond nélkül képes együttműködni. Csakhogy a ***Postfix*** nem tud közvetlenül kapcsolatba lépni a *ClamAV*-vel, vagy legalábbis ez a kapcsolat nem megbízható. A *ClamAV* sem jeleskedik az elektronikus levelek elemzésében, inkább adatfolyamokkal tud dolgozni. Ezért van szükségünk a kisegítő démonra, az ***Amavisd-new***-ra. Az ***Amavisd-new*** szintén ingyenes és nyílt forrású eszköz, létének egyetlen célja az ***MTA***-k, mint a ***Postfix*** és a ***Sendmail***, valamint a víruskereső és a szemétszűrő programok, mint a *ClamAV* és a ***SpamAssassin*** közötti tranzakciók közvetítése. Az ***Amavisd-new*** egyebek mellett kiválóan teljesít az elektronikus levelek ***MIME*** mellékleteinek hagyományos, a keresők által is érhető adatfájlokká alakításában.
Az ***Amavisd-new*** démonja, az ***amavisd*** számos protokollon keresztül képes kommunikálni, ide értendő az ***SMTP*** és az ***LMTP*** elektronikus levéltovábbító protokoll mellett a ***UNIX*** foglalatok rendszere is. Ebben az esetben az ***amavisd***-t úgy állítjuk be, hogy a leveleket ***SMTP***-n keresztül, a 10024-es számú ***TCP*** kapun át fogadja, annak helyi ***UNIX*** foglalatán keresztül lépjen kapcsolatba a *ClamAV*-vel, majd a levelet és a víruskeresés eredményét a 10025-ös ***TCP*** kapun adja tovább a ***Postfixnek***. A leveleknek az ***SMTP*** átjárón keresztüli mozgását a *2. ábra* szemlélteti.

A program beszerzése és telepítése

A *ClamAV* és az ***Amavisd-new*** egyaránt ***Perlben*** íródott, és számos *Perl*-modultól függ. Mindenkinek javaslom tehát, hogy a két eszköz valamelyik újabb változatának saját terjesztéséhez készült bináris csomagját használja. Még egyszerűbb, ha az ***apt-get***, a ***Yum*** vagy az ***up2date*** szolgáltatásaira hagyatkozunk, így nem kell foglalkoznunk a függőségek kézi telepítések során felmerülő gondjával.
A *ClamAV* webhelye, amellett, hogy itt lelhető fel a legújabb *ClamAV* forráskód is, tartalmaz egy oldalt, amelyen a *ClamAV* különféle Linux-terjesztésekhez és egyéb operációs rendszerekhez készült bináris csomagjainak elérhetőségét gyűjtötték össze. A ***Red Hat*** vagy ***Fedora*** terjesztést használók ***Dag Wieers*** oldalán (lásd az internetes forrásokat) találnak ***Yum*** ágakat és ***up2date*** forrásokat, *ClamAV*-t és ***Amavisd-new***-t egyaránt. Az ***Amavisd-new*** weboldalon további ***Amavisd-new*** csomagok forrásaira mutató hivatkozásokat is találunk, illetve a legújabb ***Amavisd-new*** forráskódot is innen tölthetjük le. A *ClamAV* a ***sarge*** kiadása óta a ***Debian*** alapcsomagja, az ***Amavisd-new*** pedig például a ***SuSE*** rendszerekben a 9.1-es kiadás óta szerepel.
Ha bármelyik összetevőt forráskódból vagy önálló csomagból telepítjük, akkor a ***Yum***, ***up2date*** vagy ***apt-get*** alapú



1. ábra Példa levelezőrendszer



2. ábra A Postfix, az Amavisd-new és a ClamAV közötti adatáramlás

megoldással ellentétben nekünk kell ügyelnünk arra, hogy az ***Amavisd-new***-hoz tartozó telepítési leírás ***Prerequisites*** (előfeltételek) című részében foglaltakat teljesítsük. Sajnos a ***ClamAV*** telepítésének feltételeit eléggé hiányosan foglalták össze. Ha kétségeink támadtak, semmibe sem kerül saját ***ClamAV RPM***-ünk nevével kiadni az

```
rpm –test -iv clamav_csomagnév.rpm
```

parancsot, így láthatjuk, hogy mi hiányzik még a gépünkről. Ha van egy kis szerencsénk, akkor terjesztésünkhöz elérhetők a ***ClamAV*** és az ***Amavisd-new*** használatához szükséges ***Perl***-modulok csomagjai. Amit mégsem találunk, azt a ***CPAN***-ról vagy az egyéb, a saját terjesztésünkhöz készített csomagokkal foglalkozó weboldalak valamelyikéről tölthetjük le.

A ClamAV beállítása

A ***ClamAV*** és az ***Amavisd-new*** telepítése után nekiláthatunk a beállítások megadásának. A ***ClamAV***-vel kezdünk, ez az egyszerűbb. A ***ClamAV*** beállító fájlja a */etc/clamav.conf*. Nyissuk meg a kedvenc szövegszerkesztőnkkel. Az 1. ábrán azok a beállítások láthatók, amelyeket a legtöbbeknek azonnal meg kell változtatniuk.

Az első sor ártatlannak tűnik, de mindenképpen tegyük megjegyzésbe. Ha ezt elmulasztjuk, a ***clamd*** nem fog futni. A két `LogFile...` beállítás alapesetben megjegyzésként szerepel. Ha engedélyezni akarjuk a naplózást, vegyük ki őket megjegyzésből. A naplófájlt mi választjuk ki, maximális mérete `LogFileMaxSize`, ennek elérésekor a fájl felülírásra kerül. A `DatabaseDirectory` kulcsfontosságú beállítás. A ***ClamAV*** itt tárolja a vírusaláírások adatbázisait, mondhatnánk, az agyát. Az általam telepített ***ClamAV RPM***-ben található ***clamd*** démon úgy volt lefordítva, hogy a */usr/share/clamav* könyvtárat használta erre a célra, miközben a hozzá mellékelt példa ***clamav.conf*** fájlban a */var/lib/clamav* érték szerepelt, igaz, megjegyzésbe téve. Úgy döntöttem, kiveszem megjegyzésből a sort, és */usr/share/clamav* értékre módosítom, kerülendő a félreértéseket.

A `LocalSocket` beállítás azt határozza meg, hogy a ***clamd*** melyik foglalaton keresztül tartja a kapcsolatot a külvilággal, jelen esetben az ***Amavisd-new***-val. Ha használjuk ezt a beállítást, márpedig én ezt javaslom, akkor ügyeljünk arra, hogy a `TCPSocket` és a ***TCPAddr*** beállítás megjegyzésben maradjon. Saját ***Genco*** csomagomnál a `LocalSocket` elérési út alapértéke `/tmp/clamd` volt, amivel az a baj, hogy a /tmp bárki által írható-olvasható. Helyette javaslom a `/usr/share/clamav/clamd.sock` elérési út választását, a */usr/share/clamav* engedélyeit pedig állítsuk `rwxrwx` értékre, vagyis vonjuk meg az egyéb felhasználók olvasási, írási és futtatási jogát. Az *1. kódrészlet* utolsó beállítása a `User`, ez annak a felhasználónak a neve, amelynek fiókjával a ***clamd***-nek indulása után futnia kell. A ***clamd***-t a rootnak kell elindítania, de ha ezt a beállítást kivesszük megjegyzésből, majd értéket adunk neki, akkor a ***clamd*** indulás után lefokozza önmagát.

A legtöbbünk számára elég ennyit ismerni a */etc/clamav.conf* fájlból. A ***clamd*** indítása előtt ellenőrizzük, hogy van-e rendszerünkben fiókja a ***clamav***-nek, és a */etc/clamav.conf* fájlban szereplő elérési utakra vonatkozó engedélyeket megfelelően beállítottuk-e. A fiók csoportjaként szintén érdemes a ***clamav***-t választani. Amint a következő részben látni fogjuk, így könnyebben meg tudunk osztani bizonyos erőforrásokat a ***clamd*** és az ***amavisd*** között.

A */etc/passwd* bejegyzése a ***clamav*** fiókhoz a következő:

```
clamav:x:52:52:ClamAV Daemon:/:/bin/false
A /etc/group fájl clamav csoportfiókja pedig a következő:
clamav:x:52:
```

Ha a ***clamd*** beállításán is túlestünk, elindításához csupán a ***clamd*** parancsot kell kiadnunk. Ha a ***ClamAV***-t bináris csomagból telepítettük, */etc/init.d* névvel valószínűleg bekerült rendszerükbe a ***clamd*** indító parancsfájlja is. Ha így van, akkor ne feledkezzünk el az engedélyezéséről, így a ***clamd*** már a rendszer betöltésekor elindul. Ha a fájl hiányzik, akkor magunknak kell gondoskodnunk a létrehozásáról.

Az Amavisd-new beállítása

Ahogy a ***clamd***, úgy az ***amavisd*** beállításait is egyetlen fájl tárolja, ez a */etc/amavisd.conf*. Ezzel azonban egy kicsit több munkánk lesz. A *2. kódrészlet* saját */etc/amavisd.conf* fájlom legfontosabb elemeit tartalmazza.

A 2. kódrészlet első két beállítása a `$daemon_user` és a `$daemon_group`, ezek az ***amavisd*** futtatásához használt felhasználói és csoportfiókot adják meg. Értéküket rendre legyen ***amavis*** és ***clamav***. Mint korábban említettem, szerintem érdemes közös csoportba rendelni az ***amavisd*** és a ***clamd*** fájljait, így tehát van értelme az amavisd-t is ezzel a csoporttal futtatni. Az ***amavishoz*** tartozó */etc/passwd* bejegyzés így néz ki:

```
amavis:x:53:52:Amavisd-new
Daemon:/var/amavis:/bin/false
```

A `$mydomain` szervezetünk tartománynevét adja meg. A `$MYHOME`, amelyet az ***amavis*** fiókjának kezdőkönyvtárára

kell állítani, az *Amavisd*-new fájljainak gyökérkönyvtárát határozza meg, ez általában a */var/amavis*. Erre a könyvtárra csak a root kapjon írási jogot, és a tulajdonjogot is neki adjuk. A `$QUARANTINEDIR` annak a könyvtárnak az elérési útja, amelybe az *amavisd*-vel a karanténba helyezett elektronikus leveleket el szeretnénk helyeztetni. A könyvtár tulajdonosa az *amavis* felhasználói fiókja legyen, és kizárólag ez kapjon írási jogot hozzá.
A `$db_home`, melyet lehetséges, hogy ki kell vennünk megjegyzésből, azt határozza meg, hogy az *amavisd* hol tárolja adatbázisait, például a gyorstárazott keresési eredményeket.
A `$helpers_home` az a könyvtár, amelybe az *amavisd* saját `SpamAssassin` beállításait írja, és néhány további apróságot helyez el. Lehetséges, hogy alapesetben a `$helpers_home` megjegyzésként szerepel. A `$db_home` és a `$helpers_home` könyvtárat az *amavis* felhasználói fiókja birtokolja, és kizárólag általa legyen írható.
A `$pid_file` és a `$lock_file`, melyek szintén megjegyzésként kerülhetnek a kezünk alá, az *amavisd* folyamatazonosító és zárolási fájljának helyét adják meg.
A `$log_level` szabja meg, hogy az *amavisd* naplóüzenetei mennyire legyenek részletesek. Itt 0 - 5 közötti értéket adhatunk meg, ahol az 5 jelenti a legrészletesebb naplózást. Az alapérték 0, személyes tapasztalatom szerint a 2-es szint elegendő adatot szolgáltat, de a naplófájl sem hízik kezelhetetlen méretűre. Alapesetben az *amavisd* naplóüzeneteit `mail` erőforrásként a *syslog*-nak küldi el, vagyis az *amavisd* és a *Postfix* naplóüzenetei ugyanarra a helyre kerülnek.
A következő négy beállítás az *amavisd* által vírus vagy levélszemét felismerésekor küldött levelekre vonatkozik.
A `$virus_admin` az az elektronikus levélcím, amelyre a vírusokról szóló értesítőket kapni fogjuk. Érvényes címnek kell lennie, ha az itt szereplő érték még nem szerepelne benne, akkor gondoskodjunk a helyi `aliases` fájl frissítéséről. A gyakorlatban itt jellemzően a rendszergazda, vagyis a saját címünk szerepel.
Arra is van lehetőség, hogy az *amavisd* az egyes levelek eredeti címzettjeinek vagy küldőjének küldjön értesítést, ám ezzel csak bosszúságot fogunk okozni másoknak, hiszen a levélszemetek és a vírusos levelek feladójának címe szinte mindig hamis. Jobb tehát, ha lemondunk erről a lehetőségről.
A `$mailfrom_notify_admin` és a `$mailfrom_notify_spamadmin` rendre azt a feladócímet adja meg, amellyel az *amavisd* a vírusokról és a levélszemetekről szóló értesítő leveleket feladja.
Ezen a ponton végre elérkeztünk az *amavisd.conf* lényegéhez: a *ClamAV*-hez tartozó víruskereső beállításokhoz. Nálam az alapértelmezett */etc/amavisd.conf* fájl a teljes részt megjegyzésbe téve tartalmazta, tehát azzal kezdtem, hogy töröltem a # karaktereket a sorok elejéről. Szükség esetén tehát ne feledkezzünk meg erről a lépésről.
Mindenképpen ellenőrizzük a *clamd* foglalatának ebben a részben megadott elérési útját. A 2. kódrészletben az alapértelmezett */var/run/clamav/clamd* helyett a */usr/share/clamav/clamd.sock* elérési út szerepel, ugyanaz, mint amit a */etc/clamav.conf* fájlban adtunk meg.
Ha megfelelően átírtuk a */etc/amavisd.conf* fájlt, és beállítottuk az *amavisd* könyvtáraira vonatkozó engedélyeket, akkor az *amavisd* parancsot – kapcsolók nélkül – kiadva indíthatjuk el a programot. Ahogy a *clamd*, úgy lehetséges, hogy

1. kódrészlet Nem alapértelmezett beállítások a /etc/clamav.conf fájban

```
# Példa
LogFile /var/log/clamd.log
LogFileMaxSize 5M
DatabaseDirectory /usr/share/clamav
LocalSocket /usr/share/clamav/clamd.sock
User clamav
```

az *amavisd* számára is létre kell hoznunk egy indító parancsfájlt. Javaslom, hogy elsőként a *clamd*-t indítsuk. Így elérhetjük, hogy mire az *amavisd* elindul, addigra a *clamd* foglalata már jelen legyen.
A clamd-hez hasonlóan az amavisd-t is a rootnak kell indítania. A program ezt követően az amavisd.conf fájlban megadott felhasználó és csoport jogosultságaival fut tovább.

A Postfix beállításai

ClamAV és *Amavisd-new* démonunk megkapta a kellő beállításokat, és el is indult. Még ne dőljünk hátra, némi tennivalónk még akad, ugyanis be kell állítanunk a *Postfixet* a tartalomszűrésre, továbbá frissítenünk kell a *ClamAV* vírusadatbázisát.
Egy fontos megjegyzés: a továbbiakban feltételezem, hogy a *Postfix* már be van üzemelve, és képes ellátni normál fogadó/továbbító feladatait.
Először nyissuk meg kedvenc szövegszerkesztőnkkel a */etc/postfix/master.cf* fájlt, majd – amennyiben még nem szerepelnek ott – fűzzük a *3. kódrészletben* látható sorokat a fájl végére.
Az *smtp-amavis* rész a *Postfix* kimenő, az *amavisd*-vel *SMTP*-n keresztül létesített kapcsolataira vonatkozik.
Ide a következő sor tartozik, ezt kell hozzáadnunk a */etc/postfix/main.cf* fájlhoz, illetve átírnunk, ha már szerepel benne:

```
content_filter = smtp-amavis:[127.0.0.1]:10024
```

Ezzel a sorral arra utasítjuk a *Postfixet*, hogy a *master.cf* fájlban megadott *smtp-amavis* felületen keresztül minden bejövő levelet küldjön ki a `127.0.0.1` címre, vagyis a helyi rendszernek, mégpedig a 10024-es *TCP* kapun, az *amavisd* alapértelmezett *SMTP* figyelő kapuján át. Az *amavisd* figyelő kapuját a */etc/amavisd.conf* fájl `$inet_socket_port` beállításának átírásával változtathatjuk meg.
A 3. kódrészlet második szakasza azt a bejövő felületet adja meg, amelyen keresztül a *Postfixnek* fogadnia kell az *amavisd* által visszaadott üzeneteket. A *Postfix* tehát a helyi hurokfelület, a `127.0.0.1`-es IP-cím 10025-ös TCP kapuján hallgatózik, ez az a kapu, amelyre alapesetben az *amavisd* az értesítéseket és a továbbított üzeneteket küldi. Az *amavisd* értesítési és továbbítási címét és kapuját rendre a */etc/amavisd.conf* fájlban szereplő `$notify_method` és a `$forward_method` beállítások átírásával változtathatjuk meg. A *master.cf* és a *main.cf* módosítása után a *Postfixet* újra kell indítani.

2. kódrészlet A /etc/amavisd.conf fontosabb beállításai

```
$daemon_user  = ´amavis´;
$daemon_group = ´clamav´;
$mydomain = ´pelda.org´;
$MYHOME   = ´/var/amavis´;
$QUARANTINEDIR = ´/var/virusoslevelek´;
$db_home   = "$MYHOME/db";
$helpers_home = "$MYHOME/var";
$pid_file  = "$MYHOME/var/amavisd.pid";
$lock_file = "$MYHOME/var/amavisd.lock";
$log_level = 2;
$virus_admin = "mick@$mydomain";
$mailfrom_notify_admin = "antivirus@$mydomain";
$mailfrom_notify_spamadmin = "antivirus
➥@$mydomain";

### http://www.clamav.net/
[´ClamAV-clamd´,
  \ &ask_daemon, ["CONTSCAN { } \ n",
"/usr/share/clamav/clamd.sock"],
  qr/\ bOK$/, qr/\ bFOUND$/,
  qr/^.*?: (?!Infected Archive)(.*) FOUND$/ ],
```

3. kódrészlet A /etc/postfix/master.cf fájlhoz hozzáadandó sorok

```
smtp-amavis unix -    -     y     -     2  smtp
     -o smtp_data_done_timeout=1200
     -o disable_dns_lookups=yes

127.0.0.1:10025 inet n  -     n     -     -  smtpd
     -o content_filter=
     -o local_recipient_maps=
     -o smtpd_client_restrictions=
     -o smtpd_helo_restrictions=
     -o smtpd_sender_restrictions=
     -o
smtpd_recipient_restrictions=permit_mynetworks,
        ➥reject_unauth_destination
     -o mynetworks=127.0.0.0/8
     -o strict_rfc821_envelopes=yes
     -o smtpd_error_sleep_time=0
     -o smtpd_soft_error_limit=1001
     -o smtpd_hard_error_limit=1000
```

A rendszer ellenőrzése

Mielőtt bármilyen további műveletnek nekifognánk, ellenőrizzük a rendszert. A legegyszerűbb módszer az ellenőrzésre az, hogy küldünk magunknak egy levelet, amelybe az alábbi karakterláncot helyezzük el. Ez nem egy valódi vírus, hanem az *Eicar* tesztaláírásnak nevezett karakterlánc:

```
X5O!P%@AP[4\ PZX54(P^)7CC)7} $EICAR-STANDARD-
ANTIVIRUS-TEST-FILE!$H+H*
```

Ha minden rendben működik, az *amavisd* küldött az *amavisd.conf* `$virus_admin` beállításában megadott címünkre egy levelet, eredeti üzenetünk pedig az *amavisd.conf* `$QUARANTINEDIR` beállításában megadott karanténkönyvtárba került.
Az ellenőrzés idejére erősen ajánlott a levelezési naplófájl végének elkülönítése. Adjuk ki tehát a

```
tail -f /var/log/mail
```

parancsot, így elkülöníthetjük a *Postfix* és az *amavisd* naplóüzeneteit. Tapasztalatom szerint ez a leggyorsabb megoldás az esetleges hibák felismerésére, főleg akkor, ha a korábban javasolt módon növeltük az *amavisd* naplójának részletességét.
Ne feledjünk legalább egy tiszta próbalevelet küldeni, ezzel ellenőrizhetjük, hogy a *Postfix* továbbra is képes-e a szűretlen levelek fogadására és továbbítására.

A ClamAV adatbázisainak frissítése

Már csak egyetlen, de nem kevésbé fontos dolog van hátra, mégpedig a *ClamAV* vírusaláírásokat tartalmazó adatbázisainak frissítése, illetve az ezt a műveletet napi rendszerességgel végrehajtó *cron* munka megadása. A *ClamAV*-hez tartozik egy pontosan ilyen célra készült, *freshclam* nevű segédprogram.
Mivel a *freshclam* az egész rendszer legegyszerűbb eleme, és gyakorlatilag a helyből is kifutottam, mindenkinek meghagyom az élményt, hogy maga fedezze fel a *freshclam(1)* és a *freshclam.conf(5)* súgóoldalakat. Annyit azért gyorsan elmondanék, hogy a hétköznapok során csupán a

```
freshclam -l /naplófájl/elérési/útja
```

parancsot kell használnunk, ahol a `/naplófájl/elérési/útja` azt a fájlt adja meg, amelybe a *freshclam* a naplóüzeneteit írja.
A *freshclamet* néhány óránként érdemes lefuttatni. A legegyszerűbb az, ha a *freshclamet* a -c és a -d kapcsolók segítségével démon módban használjuk. További tudnivalókat a *freshclam(1)* súgóoldalon találni.

Összefoglalás

Munkánk eredményeként egy *ClamAV* alapú védelemmel felszerelt *SMTP* átjárót kaptunk, vagy legalábbis elindultunk az úton ennek megvalósítása felé. Akinek további kérdése maradtak, az az internetes források között *Postfix* és *Amavisd-new* oktatóanyagot egyaránt talál. Sok szerencsét!

Linux Journal 2004. december, 128. szám

Mick Bauer biztonsági szakember, a Linux Journal biztonsági témákkal foglalkozó szerkesztője, biztonsági tanácsadó a Minnesota állambeli Minneapolisban. A Building Secure Servers With Linux című kötet szerzője (O'Reilly & Associates, 2002).

Cikkgyűjtés az Atom segítségével

Szeretnénk mindenkinek udvarias értesítést küldeni a weboldalunkon megjelent új tartalomról? Valósítsuk meg a legújabb cikkgyűjtő szabvány támogatását, és újabb eszközt nyerünk látogatóink visszacsalogatására.

A szervezett bűnözés világában a szindikátus együtt dolgozó bűnözők csoportját jelenti. A sajtó világában a szindikátus feladata az információk terjesztése az előfizetők felé, miközben minden kiadó testreszabhatja a kapott információkat. A karikatúrákat, híreket és publikációkat gyakran szindikátusok terjesztik, így a szerzők nagyobb közönséget érhetnek el, az olvasók pedig nagyobb tartalomválasztékból csemegézhetnek.
Az elmúlt néhány évben a webes fejlesztők is átvették a szindikátus kifejezést, és egyből igeként és főnévként is használni kezdték. Szerencsére a weben a szindikátus sokkal inkább a sajtó világában megszokott dolgot takarja, és nem sok köze van a mackósokhoz. Az viszont tény, hogy miként a szervezett bűnözés világában, a kapcsolódó nyilvános viták során is jónéhány ember kapott súlyos sebeket (igaz, nem fegyverek, inkább szavak által), ami megosztottsághoz és sok keserűséghez vezetett a webes szindikálás, cikkgyűjtés terén.
A megosztottság eredménye az ***Atom*** nevű új cikkgyűjtő formátum, amely sokban egyezik az ***RSS***-sel (Az ***RSS a „rich site summary"*** rövidítése. Használatos még az ***RDF site summary*** kifejezés is attól függően, hogy melyik változatról van szó és kit kérdezünk meg.) Én úgy vélem, hogy az Atom számos előnyt kínál az ***RSS*** bármely változatával szemben, és az Atom cikkek létrehozásának egyszerűsége egyértelműen előnyösebb választássá teszi az ***RSS***-hez képest. Figyelembe véve, hogy a legtöbb webnapló termék ***RSS*** cikkek előállítására képes, kijelenthetjük, hogy a két tábor békésen megfér egymás mellett. Érdemes viszont mindkét megoldás működését átlátni, hiszen így megalapozott döntést hozhatunk arról, hogy melyik – esetleg mindkettő – szabályrendszert kövessük.

Egy kis történelem

Mint a múlt hónapban láttuk, az ***RSS*** valójában két különböző formátum, vagyis inkább két különböző formátumcsalád. Az ***RSS 0.9x*** és az ***RSS 2.0*** ugyanabba a családba tartoznak, és kiválóan szemléltetik a webes cikkgyűjtés fejlődését. Az ***RSS 2.0***-t elsősorban a ***Userland***-től ***Dave Winer***, a ***scripting.com*** és újabban a ***Harvard University*** tartotta, tartja karban. ***Winer*** átadta a szabvány tulajdonjogát a ***Harvardnak***, ám azt is kijelentette, hogy a 2.0 lesz az utolsó változat. Van is benne valami, hiszen az RSS 0.9x és az RSS 2.0 együttesen széles körben elterjedt, üzembiztos, jól ismert – és a kissé zavaros – protokollt jelentenek a webes tartalmak összegyűjtésére.
Létezik egy másik ***RSS***-fajta is, neve a megtévesztő ***RSS 1.0***, amely az erőforrás-fejlesztési keretrendszerre (***resource development framework, RDF***) épül, és amelyet a ***World Wide Web Consortium (W3C)*** gondoz. Az ***RDF***-et arra tervezték, hogy a számítógépek képesek legyenek megérteni a webhelyek tartalmát, ennek alapján képesek legyenek kapcsolatokat teremteni közöttük, pontosan úgy, ahogy az emberek is teszik ezt ösztönösen. Az ***RSS 1.0*** összegzései a többi ***RSS***-változattal nem használhatók, viszont az ***RDF*** alapján szabványos leírást nyújtanak a webhely tartalmáról. Abból, hogy az ***RSS 1.0*** is az ***RSS*** nevet kapta, rengeteg vita és ellenségeskedés származott, sokan a legváltozatosabb módokon szidták ***Dave Winert***, az ***RSS*** előírások bizonytalanságát és az Atom elődjének alkotóit. Végül néhány vezető személyiség – ***Tim Bray***, ***Mark Pilgrim*** és ***Sam Ruby*** vezetésével –olyan cégek támogatásával, mint például a ***Six Degrees*** (ez adja ki a ***Movable Type*** webnapló programot) nekikezdett egy új szabálygyűjtemény kidolgozásának, amelyet eleinte ***PIE***-nak majd ***Echonak*** hívtak, és amely az ***RSS*** gyengeségeinek kiküszöbölését célozta.
Az ***Atom*** fejlesztése eltartott egy ideig, ugyanis első lépésként meg kellett határozni, hogy napjaink webje esetében pontosan mit is jelentsen a cikkgyűjtés. Az ***RSS***-t immár nemcsak eredeti alkalmazói, a híroldalak használják, de sok webnapló és nem szöveges jellegű tartalmat szolgáltató oldal is támogatja. A fejlesztők úgy döntöttek, hogy a többnyelvűségnek kiemelt szerepet szánnak, vagyis azt célozták meg, hogy a cikkeket tetszőleges nyelven lehessen szolgáltatni. Ugyancsak nagy hangsúlyt kapott a bővítmények fejlesztése, vagyis az ***Atomot*** úgy lehet új szolgáltatásokkal bővíteni, hogy a fő Atom szabályokhoz nem kell hozzányúlni.
Cikkem írásakor (2004. augusztusának közepén) az ***Atom*** szabálygyűjtemény 0.3-as változatban létezik, továbbá tartozik hozzá egy szabványos ***API*** is a hálózaton keresztül végzett tartalomszerkesztéshez. Az ***Atom*** elindult az ***IETF (Internet Engineering Task Force***, ez a szervezet állítja össze és teszi közzé az internetes szabványokat) szabvánnyá válás útján, vagyis hamarosan általánosan elfogadott szab-

vány lehet, akár csak a *TCP/IP*, az *SMTP* vagy a *HTTP*. Kétségtelen, hogy az *Atom* növekvő érdeklődésre számíthat azon szervezetek részéről, amelyek már csak az *IETF* jóváhagyó pecsétjére várnak.
Az *Atom* továbbra is kezdeti állapotban van, számos elemnek – például a bővítményeknek – hiányzik a nyilvános leírása. Készítői ugyanakkor máris elmondhatják, hogy készítettek egy szabványt, amelynek összetettsége közel áll az *RSS 0.9x* és *2.0* előírásgyűjteményéhez, jól érhető és áttekinthető, a webes cikkgyűjtő közösség komoly támogatását élvezi, valamint szemléletében messze túlmutat a puszta webes cikkgyűjtésen.

Atom cikk létrehozása

Míg az *RSS*-t elsősorban híroldalak és webnaplók bejegyzéseinek összegyűjtésére tervezték, az *Atom* inkább általános célú megoldásnak készült. Lehet például olyan rendszert építeni, hogy egy üzem gépei állapotjelentésüket *Atom* formátumban készítik el, majd egy cikkgyűjtő ezek alapján jelzi a meghibásodásokat. Megoldható, hogy a könyvtárak Atom cikkeket készítsenek legújabb beszerzéseikről, a különféle témákkal kapcsolatos könyvek elérhetőségét pedig intelligens cikkgyűjtők figyeljék. További felhasználási terület a faxgépek faxmodemekre cserélése, majd a faxképek Atom alapú terjesztése a megfelelő személyek vagy csoportok felé.
Az *Atom* akár szerkesztőségi rendszer összeállítására is alkalmas lehet, ahol a tudósítók írásaikat nem elektronikus levélben, hanem *Atom* cikkek formájában küldik el. Mindegyik szerkesztő összegyűjtené az irányítása alá tartozó tudósítók *Atom* cikkeit, majd a szerkesztési munka befejezése után kimenő *Atom* cikkekbe másolná őket. A cikkgyűjtés utolsó fázisa a tördelő részleg lenne, ahol előkészítenék a szövegek nyomtatását. Az újság tartalma tehát *Atom* cikkek áramlása révén állna össze, a végső cikk maga az újság lenne.
Atom cikket előállítani rendkívül egyszerű, ha *Perl*-t vagy más magas szintű nyelvet használunk, amelyhez létezik *Atom* könyvtár. A *Perl* esetében például az *XML::Atom* modullal tudunk dolgozni, amely a *CPAN*-ról tölthető le. Nekem Fedora Core 2 alatt, a *Perl 5.8.3*-as változatával volt némi gondom az *XML::Atom* telepítésével, ezt úgy tudtam megoldani, hogy az elhagyható *DateTime* modult kivettem a telepítésből. Éles környezetben ezt a megoldást nem javaslom.
Bár a teljes csomag neve `XML::Atom`, az *Atom* cikkeket előállító programok valójában az `XML::Atom::Feed` és az `XML::Atom::Entry` modult használják. Példaként lássunk egy rövid *Perl* programot, amely egy egyszerű cikket állít elő, és amely részben az `XML::Atom::Feed` internetes perldoc leírásában szereplő példaprogramra alapul:

```
#!/usr/bin/perl

use strict;
use diagnostics;
use warnings;

use XML::Atom::Feed;
use XML::Atom::Entry;

# Új Atom cikk létrehozása
my $feed = XML::Atom::Feed->new;
$feed->title(´Webnaplóm´);

my $entry;
# A cikk első bejegyzésének létrehozása
$entry = XML::Atom::Entry->new;
$entry->title(´Első írás´);
$entry->content(´Első írás törzse´);
$feed->add_entry($entry);

# A cikk második bejegyzésének létrehozása
$entry = XML::Atom::Entry->new;
$entry->title(´Második írás´);
$entry->content(´Második írás törzse´);
$feed->add_entry($entry);

# Az XML kimenet előállítása

my $atom_feed_xml = $feed->as_xml;

# Az XML kimenet megjelenítése
print $atom_feed_xml, "\ n";
```

A fenti program az alábbi cikket állítja elő, amelyet a könnyebb olvashatóság kedvéért kicsit átformáztam:

```
<?xml version="1.0"?>
<feed xmlns="http://purl.org/atom/ns#">
<title>
    Webnaplóm
</title>
<entry
xmlns:default="http://www.w3.org/1999/xhtml">
    <title>
    Első írás
    </title>
    <content mode="xml">
    <default:div
        ➥xmlns="http://www.w3.org/1999/xhtml">
        Első írás törzse
    </default:div>
    </content>
</entry>
<entry
xmlns:default="http://www.w3.org/1999/xhtml">
    <title>
    Második írás
    </title>
    <content mode="xml">
    <default:div
        ➥xmlns="http://www.w3.org/1999/xhtml">
        Második írás törzse
    </default:div>
    </content>
</entry>
</feed>
```

Mint láthatjuk, létre kell hoznunk egy `XML::Atom::Feed` objektumot, amibe az `XML::Atom::Entry` egy vagy több példánya kerül. Az Atom cikken belül minden bejegyzés

objektum egy-egy <entry> címkével párosul, ezek pedig egy-egy üzenetet jelenítenek meg webnaplónkból vagy éppen az üzem felügyeleti rendszeréből.
Az Atom szabálygyűjtemény szerint a cikk számos jellemzőt és alelemet is magába foglalhat, ilyen például a nyelv, a webnapló vagy webhely leírása, a szerzői jogi megjegyzések és a származási hellyel kapcsolatos egyéb tájékoztatások. Az egyes bejegyzések ugyanakkor saját elemkészlettel rendelkeznek, mint például cím, létrehozás időpontja és összegzés. Minden Atom elemnek van egy *MIME* típusa, amely a tartalom jellegéről tájékoztat, hasonlóan a *HTTP*-válaszokhoz és az elektronikus levelek mellékleteihez.
Természetesen cikk létrehozására a fenti módon csak akkor van szükség, ha új *Atom*-képes alkalmazást írunk, vagy szeretnénk ilyen irányba bővíteni meglévő webnapló alkalmazásunkat. A legtöbb webnapló termék már most is képes Atom cikkek előállítására, akár a normál csomag részeként, akár valamilyen beépülő modul vagy egyéb bővítmény révén. A *Blosxom Weblog* rendszerhez például beépülő modul készült, segítségével könnyedén megoldható az Atom cikkek előállítása. A beépülő modult egyszerűen csak be kell másolni a *plugins* könyvtárba, és ettől kezdve bárki kaphat Atom cikkeket a kérdéses webnaplóról.
Remélem, nem okozott meglepetést, hogy mindez ennyire egyszerű, ugyanis a *Blosxom Perlben* íródott, a *Perl* pedig kiváló eszközöket kínál az *XML* kezelésére, a beépülő modulnak pedig csak annyi a dolga, hogy összesítse és újraírja a webnapló legújabb bejegyzéseinek tartalmát. Abból, hogy a *Blosxom* ennyire könnyűvé teszi a beépülő modulok számára a főoldal módosítását (ahogy az *Atom* cikk megjelenésének jelzését is) és a tartalom lekérdezését (a beépülő modul *API*-n keresztül), egyenesen következik, hogy alóla rendkívül könnyű az Atommal dolgozni. Mivel a legtöbb webnapló program magas szintű nyelven, tehát *Perlben*, *Pythonban* vagy *PHP*-ban készül, az *Atom* kezelésének megvalósítása még nulláról kezdve sem okozhat nehézséget.

Atom cikk feldolgozása

Az *Atom* cikkek feldolgozására rengeteg módszer közül választhatunk, akár cikkgyűjtőt szeretnénk írni, akár *Atom* alapú alkalmazást szeretnénk készíteni. A legegyszerűbb megoldás a cikkek felfedezésére és lekérdezésére továbbra is az `XML::Atom::Feed` használata. Például:

```
#!/usr/bin/perl

use strict;
use diagnostics;
use warnings;

use XML::Atom::Feed;

# A www.diveintomark.org Atom cikkeinek
# lekérdezése
my @uris =
    XML::Atom::Feed->find_feeds(
        “http://www.diveintomark.org/”);

    # Az egyes Atom cikkekhez tartozó URI-k
    # kiírása
    foreach my $uri (@uris)
    {
    print “uri = ´$uri´\ n”;
    }
```

A fenti példával egyszerűen meg tudjuk jeleníteni az *URI*-kat. Most, hogy már tudjuk, hol keressük az egyes cikkeket, elő tudjuk állítani a bennük szereplő hivatkozások listáját, majd a hivatkozásokat *XML* formára tudjuk hozni:

```
#!/usr/bin/perl

use strict;
use diagnostics;
use warnings;

use XML::Atom::Feed;

# Atom cikk lekérése
my @uris = XML::Atom::Feed->
➥find_feeds(“http://www.diveintomark.org/”);

foreach my $uri (@uris)
{
my $feed = XML::Atom::Feed->new(URI->new($uri));

my @links = $feed->link();

foreach my $link (@links)
{
    my $link_xml = $link->as_xml();
    print “link = ´$link_xml\ n”;
}
}
```

Természetesen *XML*-t nem állítunk elő és nem jelenítünk meg, a lényeg a hivatkozások kinyerése, amelyeket elektronikus levélben el tudunk küldeni az előfizetőknek, hozzá tudunk adni a megfelelő adatbázishoz, de akár el is hagyhatjuk közülük az adott feltételeknek meg nem felelőket.
Mivel az *Atom* cikkek szabályos felépítésűek, illetve internetes szabványokra alapulnak, mint az *XML*, az *Unicode* és a *MIME*, biztosak lehetünk abban, hogy a cikkből kiemelt tartalmat egyszerű módszerekkel tudjuk kezelni. A különféle tartalomtípusokat különféle kezelőprogramoknak továbbíthatjuk, különféle módszerekkel dolgozhatjuk fel (akárcsak a korábban említett szerkesztőségi rendszerben) vagy új cikkekbe másolhatjuk őket, így akár egy „szupercikkgyűjtőt” is létre tudunk hozni.
Aki szeretne cikkgyűjtőt készíteni, vagy szeretne pontosabb képet alkotni arról, hogy a milliófféle *RSS*- és *Atom*-változatokkal hogyan tud dolgozni, annak érdemes megismernie *Mark Pilgrim* cikkgyűjtőjét. A program *Python* alapú, szerzője folyamatosan frissíti, és talán ez a legjobb leírással ellátott nyílt forrású motor a cikkgyűjtő rendszerek anyagainak kezelésére.

RSS vagy Atom?

A fő kérdés: weboldalunk – vagy éppen webnaplónk – az *RSS* vagy az *Atom* formátumú cikkgyűjtést támogassa, esetleg mindkettőt? Számomra egyértelmű, hogy az *Atom* a leg-

jobb a napjainkig kidolgozott kettő (pontosabban három) cikkgyűjtő formátum közül. ***Dave Winer RSS*** formátumai megjelenésükkor komoly újdonságnak és előrelépésnek számítottak, ám túl sok hibájuk volt ahhoz, hogy egy teljes értékű, akár vállalati környezetben is használható szabvány alapját adják. A ***HTML*** és a ***JavaScript*** korai változatainak példáján láthattuk, hogy a félkész szabványok csak kínlódást hoznak, és figyelembe véve, hogy a cikkgyűjtés a jövőben jó eséllyel rendkívül fontos kommunikációs módszerré válik, a teljesség és az egyértelműség alapkövetelménynek számít.

Hasonlóan fontos a különféle nyelvek támogatása, valamint azt sem szabad elfeledni, hogy az emberek nem csak szövegeket szeretnének közzétenni. Az ***Atom*** egyértelműsége a különleges karakterek kezelése tekintetében szintén hangsúlyos tényező, ennek köszönhetően biztosak lehetünk abban, hogy ha elhelyezünk a webnaplónk szövegében egy < and > elemet, akkor ezzel nem fogjuk megzavarni a cikkgyűjtés működését. A legfontosabb talán mégis is, hogy a tervezett bővítési lehetőségek révén az Atom bármely különleges csoport vagy alkalmazás igényeinek képes lesz megfelelni úgy, hogy magát az alapot nem kell majd módosítani.

Az ***Atom*** rendkívül sokoldalú, használata mégis egyszerű, ennek fejlesztése során is nagy figyelmet szenteltek. Egy új ***API***-t létrehozni nem egyszerű, különösen, ha a lehető leginkább általános jellegűnek kell lennie.

Végül megjegyeznék annyit, hogy az ***RSS*** változatszámok kavalkádja, amely kicsinyes és szinte már a politikára emlékeztető vitákat szült – miközben maga a kavalkád is ezeknek a vitáknak köszönhetően alakult ki – gyakorlatilag senkinek nem vált hasznára. Az ***Atom*** különböző nevet kapott, és bár ennek a névnek homályos a jelentése, legalább nem súlyosbítja a fejlesztők és a felhasználók oldalán az ***RSS*** miatt kialakult félreértéseket.

Összefoglalás

Az ***Atom*** révén esélyt kaptunk az ***RSS*** kapcsán felmerült gondok megoldására, illetve arra, hogy a cikkgyűjtés újfajta, internetes alkalmazások között folyó, magas szintű adatcsere megoldások alapjává válhasson. Az ***Atom*** némileg bonyolultabb ***Dave Winer RSS***-változatainál, ám – legalábbis első változatát tekintve – egyszerűbb, mint a webhelyek leírását és tartalmuk összesítését az ***RDF*** alapján végző ***RSS 1.0***. Az ***Atom*** cikkek kezelését leegyszerűsítő, könnyen használható segédeszközök, a bővíthetőség és a szerzőknek az internetes szabványokra fordított figyelme egyértelművé teszik, hogy az ***Atom*** fontos szerepet fog játszani a webes kommunikáció jövőjében.

Linux Journal 2004. november, 127. szám

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

Filmszerkesztés a Kino-val

Napjaink olcsó videokamerái kiváló felvételeket készítenek. Ha igazi filmeket szeretnénk előállítani, akkor csupán a megfelelő szerkesztőeszközökre és persze gyakorlatra van szükségünk.

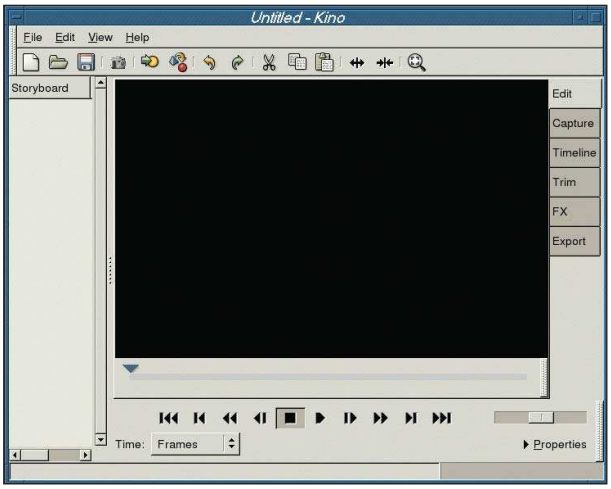
Minden videokamera-tulajdonosban felvetődött már a gondolat, hogy nyers felvételeit átszerkesztve valamilyen igazi filmet is készíthetne. Maga a kamera hiába tud számtalan különleges hatást, szerkesztőeszközökre mindenképpen szükségünk lesz. Szerencsére a manapság kapható nagyteljesítményű számítógépekkel, a ***Linuxszal*** és a ***Kino*** nevű alkalmazással könnyedén összeállíthatjuk saját kis hollywoodi stúdiónkat.

A ***Kino*** egy nemlineáris szerkesztőprogram. Képes a nyers felvételek rögzítésére, szerkesztésére és átrendezésére, valamint a végeredmény átalakítására és mentésére. A további ingyenes beépülő modulok révén a ***Kino*** minden olyan eszközt a kezünkbe ad, amelyre jó filmek szerkesztéséhez szükségünk lehet.

A filmkészítés általában a nyersanyag kamerás felvételével kezdődik. Ezzel a résszel nem foglalkozunk, a munkát a nyers felvétel számítógépre másolásának fázisától tárgyaljuk. A korszerű kamerák a számítógépekkel ***IEEE 1394*** felületen keresztül kommunikálnak, ezt az ***Apple FireWire***, a ***Sony*** pedig ***i.Link*** névvel is illeti. Az átmásolás után lehetőségünk nyílik a mozgókép szerkesztésére, valamint feliratokat és hatásokat adhatunk hozzá. A filmszerkesztés a hangok hozzáadására, keverésére és lecserélésére is kiterjedhet. A ***Kino*** az összes ilyen feladat elvégzésére alkalmas. Ha elkészült a film, felírhatjuk ***DVD***-lemezre, így különálló ***DVD***-lejátszóval is meg tudjuk nézni, de ***MPEG4*** fájlt is készíthetünk belőle. A tömörített fájlok minősége elmarad az eredeti felvételekétől, ha tehát valóban kiváló minőségre vágyunk, és digitális kameránk tudását tökéletesen ki szeretnénk használni, akkor a filmet a ***Kino*** és a kamera segítségével ***DV*** szalagra is visszaírhatjuk.

A szükséges hardvereszközök

A videófolyam kamera és számítógép közötti továbbításához mindkét oldalon egy ***IEEE1394*** csatolóra van szükség. Nézzük át gépünket, sok korszerű számítógép, ide értve a hordozhatókat is, rendelkezik ilyen csatolóval. Ha nincs ilyen a gépünkben, különálló ***IEEE1394*** kártyát kell vásárolnunk, ilyet számos gyártó választékában találunk. Nyilván egy digitális videokamerára is szükségünk van, ez ***Digital8*** (röviden ***D8***) vagy ***MiniDV*** rendszerű egyaránt lehet.



1. ábra A Kino főablaka indításkor

A számítógép és a kamera összekötésére kell még egy kábel. A kamerákon általában négy érintkezős mama csatlakozó van, a számítógépeken pedig hat érintkezők csatlakozókat találunk. Vessünk tehát egy pillantást a két készülékre, majd, ha még nincs ilyenünk, vegyünk egy 4-4 vagy 4-6 kábelt.

Mondani sem kell, számítógépünknek nagy kapacitású merevlemezzel kell rendelkeznie, amin nagy mennyiségű szabad helyet kell biztosítanunk. Ha például egy 60 perces ***MiniDV*** kazetta tartalmát szeretnénk a számítógépre másolni, akkor 12 GB helyre lesz szükségünk. A szerkesztéshez, a képek és hangok tárolására 15 GB-tal számolhatunk, vagyis egy órányi nyers felvételből 27 GB-nyi helyen tudunk filmet vágni. Attól függően, hogy mit szeretnénk végeredménynek, további 14 GB-ra lesz szükségünk, ha a filmet egy új ***.dv*** fájlba szeretnénk kiírni. Az átmásolás idején a számítógépnek másodpercenként nagyjából 3,5 MB adatot kell rögzítenie, tehát a merevlemez a lehető leggyorsabb legyen. Megfelelő beállításokkal bármelyik korszerű ***Ultra-DMA*** meghajtó megfelel.

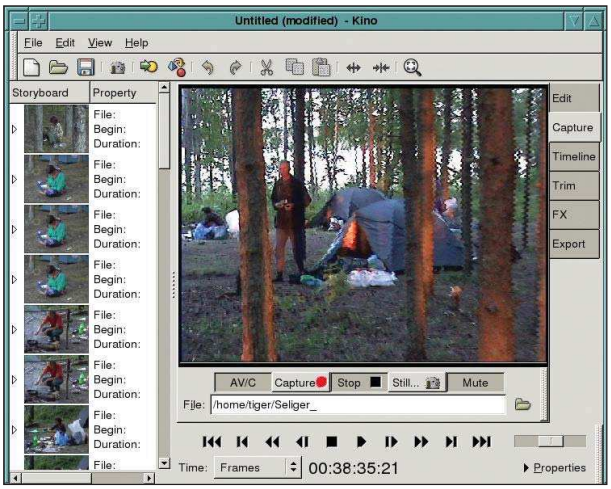
1 GHz órajelű processzor és 128 MB memória nélkül neki se kezdjünk a műveletnek, de ha kényelmesen akarunk dolgozni, akkor több memóriával és erősebb processzorral számoljunk.

A szükséges programok

A programok listája a ***Kino***-val és az általa igényelt könyvtárakkal kezdődik, de itt korántsem ér véget. A ***Kino*** csak az alapvető szerkesztőeszközöket biztosítja, és csupán néhány hatást támogat. ***Tim Shead*** és ***Dan Dennedy*** fejleszti a ***timfx***-et, mely további hatások beépítését segítő ***Kino*** beépülő modulok gyűjteménye. A feliratok, címek hozzáadásához egy további beépülő modul szükséges, az ***Alejandro Sierra*** által fejlesztett ***dvtitle***.

Mivel a ***Kino*** további programokat és könyvtárakat is igényel, forrásból való telepítése bizony nem egyszerű feladat. A ***Kino*** egységes csomagként számos terjesztés – ***Debian***, ***SuSE*** és ***Fedora*** – alá érhető el. A megfelelő csomag telepítése jóval egyszerűbb, mint a forrással bajlódni.

A ***Kino*** fejlesztése rendkívül jó tempóban halad. Cikkünk írásakor a legújabb változat a 0.7.3. A ***Debian 3.0*** üzembiztos kiadásban a ***Kino 0.5.0***, a ***Debian 3.1***-ben (fejlesztői ág)



2. ábra A Kino ablaka rögzítés után

a 0.7.3, míg a ***SuSE 9.1***-ben a 0.7.0 változat szerepel. A programok naprakészen tartásához minden szükséges csomag rendelkezésünkre áll.

Nyers felvétel átmásolása számítógépre

Miután telepítettük a programokat, az ***IEEE1394*** felületen keresztül csatlakoztassuk kameránkat a számítógéphez, majd tetszőleges X terminálon vagy a ***KDE*** vagy a ***GNOME Alt-F2*** párbeszédpaneljéről adjuk ki a `kino` parancsot.

A ***Kino*** nyitóképernyője jelenik meg, mely az *1. ábrán* láthatóhoz fog hasonlítani.

A grafikus felületet könnyű átlátni. Mozgókép rögzítéséhez válasszuk a jobb oldalon lévő ***Capture (Rögzítés)*** fület. A felvétel elindítása előtt azonban érdemes ellenőrizni a beállításokat. Az alapértelmezett beállításokat az ***Edit->Preferences (Szerkesztés->Beállítások)*** paranccsal érhetjük el.

A ***Normalisation (Normalizálás)***, ***Audio (Hang)*** és ***Aspect Ratio (Képarány)*** beállítást a kamerának megfelelően állítsuk be. A pontos beállítások a kamera mellett attól is függenek, hogy a világ mely tájékán élünk. Az ***USA***, ***Kanada*** és ***Japán*** területén az ***NTSC***, míg a világ többi részén a ***PAL*** szabványt használják. A kamerák általában kétféle hangmódot ismernek, a 16 és a 14 biteset. A legtöbb esetben az előbbi az alapbeállítás. A felvétel elindítása előtt ezt is állítsuk 16 bitre, így jobb minőséget kapunk.

A felvett mozgóképet háromféle formátumba menthetjük, ***.dv*** és kétféle ***.avi*** formátumot választhatunk. Általában mindegy, melyiket használjuk, de inkább maradjunk a nyers ***DV***-nél.

Ha végeztünk a beállítások megadásával, kattintsunk a ***Capture*** gombra, majd – kiterjesztés nélkül – írjuk be a felveendő mozgóképet tároló fájl nevét. Ha gondoljuk, kapcsoljuk be Az ***Auto Split Files (Fájlok önműködő vágása)*** szolgáltatást, ez az általa felismert jelenetváltásoknál új, az általunk megadott névvel és egy sorszámmal elnevezett fájlt kezd. A többi beállításnál nyugodtan meghagyhatjuk az alapértéket.

Következő lépésként rá kell vennünk a kamerát a vezérlőjelek fogadására. A kamerák ezt jellemzően lejátszás módban tudják, de ha bizonytalanok vagyunk, olvassuk át a kamera leírását. Próbáljuk ki, hogy tudjuk-e a ***Kino***-val vezérelni a kamerát. Próbáljuk meg előre-hátra csévélni a szalagot. Ezután kattintsunk a program ***Play (Lejátszás)*** gombjára. A lejátszásnak el kell indulnia, a felvételnek a ***Kino*** főablakában és a kamera kijelzőjén is meg kell jelennie. Figyeljük az ***Eldobott képkockák (Dropped frames)*** mező értékét. Elméletileg itt nullának kell megjelennie, de akkor sincs baj, ha az első egy-két képkocka elvész. Ha viszont az eldobott képkockák száma folyamatosan növekedik, akkor vagy lassú a számítógép, vagy elállítottunk valamit. Ha a kamerát nem tudjuk vezérelni a ***Kino***-val, akkor próbáljuk kézzel betölteni a ***raw1394*** illesztőprogramot. Ehhez a `modprobe raw1394` parancsot kell kiadnunk.

Ha gépünk túl lassúnak bizonyul, akkor próbáljuk ki a `dvgrab` nevű, digitális videó rögzítésére alkalmas parancssori programot. A ***dvgrab*** a ***Kino*** honlapjáról érhető el. Használata előtt lépjünk ki az X-ből, majd kövessük a hozzá tartozó man oldalon foglalt utasításokat. A nyers felvétel rögzítése után indítsuk el a ***Kino***-t, és az alábbiakat követve töltsük be szerkesztésre a felvételt.

Ha a lejátszás jól működik, nekiláthatunk a rögzítésnek. Léptessünk 1-2 másodperccel a rögzíteni kívánt rész elé, majd kattintsunk a ***Capture*** gombra. A ***Kino*** elindítja a kamerát, és rögzíti a felvételt. A rögzítést bármikor leállíthatjuk, ha rákattintunk a ***Stop*** gombra. Az átvitt mozgókép a jelenetlistában tűnik fel, a ***Kino*** ablakának baloldali részén. Az ***AutoSplit*** szolgáltatás időnként hibásan működik, ám ezeket a hibákat később ki tudjuk küszöbölni. A ***Kino*** ablakának rögzítés utáni állapotát láthatjuk a *2. ábrán*.

Amint végeztünk a rögzítéssel, válasszuk a ***File (Fájl)*** menü ***Save (Mentés)*** parancsát, és egy ***SMIL (Synchronized Multimedia Integration Language, szinkronizált multimédia integrációs nyelv)*** fájl formájában mentsük el tervezetünket. Az *1. kódrészlet* egy ***SMIL*** példát tartalmaz.

A `<seq>` és a `</seq>` címke között mozgóképek leírásai találhatók. Mindegyik egy egyszerű vagy összetett jelenetet ad meg. Az egyszerű jeleneteket egy-egy `<video...>` címke írja le, ez a filmben felhasználandó szakasz első és utolsó képkockájára mutat. Az összetett jelenetek egyszerű jelenetek halmazai. A jelenetek első képkockája a ***Kino Storyboard (Forgatókönyv)*** bal oldalán jelenik meg.

1. kódrészlet Példa .smil filmtervezetre

```
<?xml version="1.0"?>
<smil xmlns:smil2="http://www.w3.org/2001/SMIL20/Language">
  <seq>
    <video src="/mnt/RAW FILES/Paris001.dv" clipBegin="0" clipEnd="441"/>
  </seq>
  <seq>
    <video src="/mnt/RAW FILES/Paris002.dv" clipBegin="0" clipEnd="368"/>
  </seq>
  <seq>
    <video src="/mnt/RAW FILES/Paris003.dv" clipBegin="28" clipEnd="761"/>
    <video src="/mnt/RAW FILES/Paris004.dv" clipBegin="567" clipEnd="967"/>
    <video src="/mnt/RAW FILES/Paris008.dv" clipBegin="28" clipEnd="761"/>
  </seq>
  <seq>
    <video src="/mnt/RAW FILES/Paris004.dv" clipBegin="26" clipEnd="234"/>
  </seq>
</smil>
```

Ha a filmet több különböző szalagról állítjuk össze, akkor helyezzük be a következőt a kamerába, majd ismételjük meg a fenti lépéseket.

A forgatókönyv az egyes jelenetek első képkockája mellett további hasznos adatokat is tartalmaz, így például a jeleneteket tároló fájlok nevét, továbbá a jelenetek kezdő időpontját és hosszát.

Rögzítés közben a ***Kino*** az eredeti forrás szerinti időt jeleníti meg, szerkesztés közben viszont a futó film ideje látható. Az idő kijelzése többféle formátumban is történhet. A legegyszerűbb a keret formátum, ez egy nulláról induló keretszámlálót mutat. Az emberi gondolkodáshoz közelebb áll a másodperc és perc alapú mérés, természetesen ilyet is választhatunk. Ha gondoljuk, használhatjuk a ***Society of Motion Picture and Television Engineers (SMPTE, Mozifilm és Televíziós Mérnökök Szervezete)*** időkód formátumát is, amely egy pontosvesszővel elválasztva az időt és a keretszámot egyaránt tartalmazza. Ha például a ***00:07:40;15*** időkijelzést látjuk, akkor 7 percnél, 40 másodpercnél és 15 képkockánál járunk. A nekünk tetsző időformátumot a ***Time (Idő)*** legördülő menüből választhatjuk ki.

A film szerkesztése

Mozart állítólag soha nem készített vázlatokat, ám nem lehet mindenki ***Mozarthoz*** hasonló. Ha a rögzítést túl korán kezdjük, netán túl későn állítjuk le, akkor a jelenetet meg kell vágni, ellenkező esetben hibás, szükségtelen, a kívánt időn túlnyúló részek lesznek a filmben. Egy-egy jelenet hosszát általában a cselekmény határozza meg. Ha nincs cselekmény, akkor a jelenet ne legyen 4-6 másodpercnél hosszabb. Az ennél rövidebb jelenetek villanásoknak hatnak, a hosszabbak viszont unalmasak.

A szerkesztés megkezdéséhez tekintsük át a jeleneteket. Ha egy jelenetben csak néhány felesleges kockát találunk, akkor egyszerűen töröljük őket. Ha az önműködő vágás hibázott, akkor a kérdéses részt egy másik jelenethez is áthelyezhetjük. Ha el akarunk távolítani egy jelenetet, kattintsunk a jobb oldalon található ***Edit (Szerkesztés)*** fülre, a forgatókönyvben mutassunk a törlendő jelenetre, majd kattintsunk rá. A főablakban a jelenet első képkockája tűnik fel. Kattintsunk az eszköztár olló ikonjára. A jelenet törlődik a forgatókönyvből, és a következő jelenet első képkockája tűnik fel a főablakban.

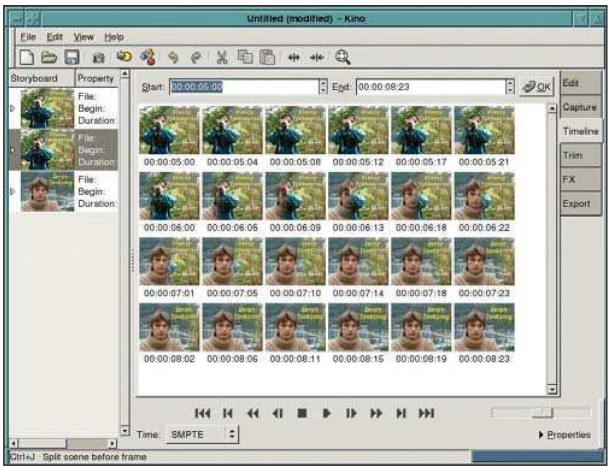
Sokszor szükség támad arra, hogy áttekintsük egy-egy jelenet egyes képkockáit. A ***Kino*** használatakor ezt az ***Idővonalra (Timeline)*** kattintva tehetjük meg. A *3. ábrán* a ***Kino*** idővonala látható. Ha egy jelenetnek csak egy részét akarjuk kivágni, akkor szükséges és szükségtelen részekre kell szétosztanunk. Ezeket az átvitelvezérlő vonallal kereshetjük meg és tekinthetjük át. Ha az eszközkészlet szétvágás ikonjára kattintunk, az éppen kiválasztott képkockával új jelenet kezdődik. Ügyeljünk arra, hogy az aktuális képkocka a ***Cut (Kivágás)*** paranccsal törölni kívánt jelenetben legyen. A módszer elsősorban akkor használható jól, ha egy jelenet középső részét szeretnénk eltávolítani. Ha pontosan be akarjuk állítani egy jelenet kezdetét vagy végét, akkor inkább a ***Trim (Vágás)*** módot használjuk.

Metszés

A metszés rendkívül hasznos művelet. Alapvetően egy-egy jelenet első és utolsó képkockájának pontos kivágására használható. A metszés használatához válasszuk ki a jelenetet a forgatókönyvben, majd kattintsunk a ***Trim*** fülre. Egy csúszka és két szövegdoboz jelenik meg, a bal oldali az ***In (Bemenet)***, a jobb oldali pedig az ***Out (Kimenet)*** felirattal, ahogy az a *4. ábrán* is látható. A dobozokban a mozgókép-fájl aktuális jelenetének kezdő és záró pillanatához tartozó képkockák száma jelenik meg. A vezérlőgombokkal vagy a csúszka segítségével mozogjunk az új kezdő pozícióra. Az egy-egy képkockával dolgozó funkciók révén tökéletes pontossággal léptethetünk. Ha megvan az új kezdő pillanat, a pozícióváltáshoz kattintsunk a vonal alatti háromszögre. Az ***In*** szövegdoboz megváltozása visszajelzi a kiválasztást. A jelenet végét hasonló módszerrel jelölhetjük ki, végleges beállításához kattintsunk a jobb oldali háromszögre. Ha elégedettek vagyunk az eredménnyel, kattintsunk a piros színű ***Apply (Alkalmaz)*** gombra.

A program akkor is életbe lépteti a módosításokat, ha kilépünk metszés módból vagy másik jelenetet választunk ki a forgatókönyvből, ezért legyünk óvatosak szerkesztés közben.

Az ***In*** mutató alatt egy szövegmező jelzi az aktuális metszési módot. Kétféle ilyen mód van, az ***Insert (Beillesztés)*** és az ***Overwrite (Felülírás)***, ezek a szövegszerkesztőknél megszokott módon használhatók. A felülírás használatakor a forgatókönyv éppen kiválasztott jelenetének helyét az új veszi át, a beillesztés választásakor pedig az új jelenet az éppen



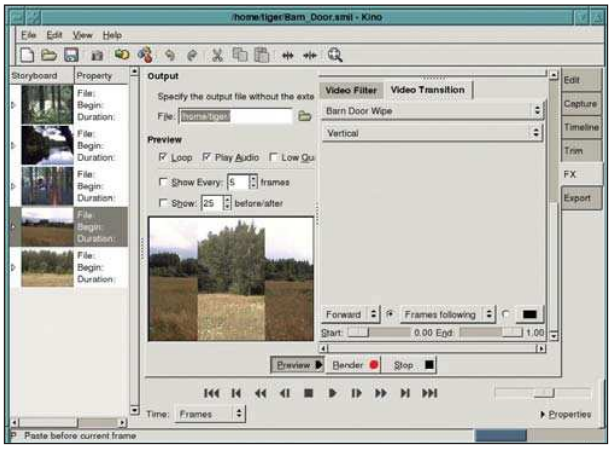
3. ábra Jelenet áttekintése az idővonal segítségével

A ***Commands/Insert Movie (Parancsok/Film beillesztése)*** gombbal a kiválasztott filmet az aktuális pozícióra illeszthetjük be, a ***Commands/Append Movie (Parancsok/Film hozzáfűzése)*** gombbal pedig a végére. A beillesztett fájlokat a program kérésünkre azonnal jelenetekre osztja.

Jelenetek összekapcsolása hatásokkal

Ha összeállítottuk a jeleneteket, akkor a film mozgókép része majdnem kész, ám, ha gondoljuk, a ***Kino*** vagy a ***timfx*** hatásainak egy részét is beledolgozhatjuk. A hatásokkal bánjunk módjával, ha túlságosan sokat szórunk el a filmben, az zavarja a nézőket. Nem muszáj az összes jelenetváltásnál hatást alkalmazni. Gondoljunk arra, hogy ha készítünk egy dokumentumot, annak sem írjuk minden bekezdését másik betűtípussal. A legegyszerűbb ***Kino*** hatás a ***Barn Door (Kétszárnyú ajtó)***. Próbáljunk vele összekapcsolni két jelenetet.

4. ábra A metszés használata



5. ábra A Kino kétszárnyú ajtó hatásának használata – a hatás előnézete egy külön ablakban jelenik meg

kiválasztott elé vagy mögé kerül. A beillesztés módnak nincs ***Apply*** gombja, csak ***Before (Elé)*** és ***After (Utána)*** ikonja. A módosítások érvénybe léptetéséhez a megfelelő gombra kell kattintani.

A jelenetek sorrendezése

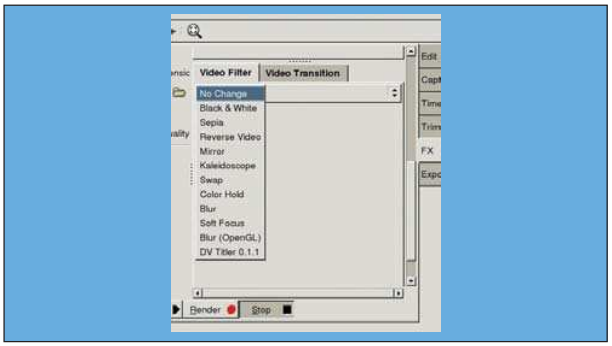
Ha az összes jelenetről és epizódról levagdaltuk a felesleges részeket, megfelelő sorrendbe kell rendeznünk őket. Ezt legegyszerűbben a forgatókönyv segítségével tehetjük meg. Először kattintsunk az ***Edit (Szerkesztés)*** fülre, majd kattintsunk a forgatókönyv áthelyezni kívánt jelenetére, végül a ***Cut (Kivágás)*** parancsra. Válasszuk ki, hogy melyik jelenet legyen a kivágott mögött, majd kattintsunk a ***Paste (Beillesztés)*** parancsra. A kivágott jelenet a kiválasztott előtt jelenik meg. Még könnyebb a jelenetek átrendezése a fogd és vidd módszer használatával. Szerkesztés módban kattintsunk az áthelyezendő jelenetre, tartsuk lenyomva az egér gombját, majd húzzuk a forgatókönyv kívánt pozíciójára a jelenetet. Az egérgomb felengedése után a jelenet ezen a helyen tűnik fel. Húzás közben figyeljük az aktuális célpozíciót jelölő vékony vonalat. A fogd és vidd módszer nem működik, ha a cél pontozott téglalapként jelenik meg. Folyó filmtervezetünkbe korábban rögzített mozgóképeket is beilleszthetünk.

Kattintsunk az ***FX->Video Transition*** (***FX->Videoátmenet***) parancsra. A legördülő menüből válasszuk a ***Barn Door Wipe (Kétszárnyú ajtó átúsztatás)*** elemet. A másik mezőben válasszuk ki a kívánt ajtótípust. Az *5. ábrán* egy függőleges típus látható. A forgatókönyvben megjelenik az aktuális, a következővel összefűzendő jelenet első képkockája. Válasszuk a ***Frames following (Következő képkockák)*** és a ***Forward (Előre)*** parancsot. A leképezés előtt a ***Preview (Előnézet)*** gombbal nézhetjük meg az eredményt. Játsszunk el a hatás beállításaival is, és vizsgáljuk meg a leendő végeredményt. Ha elégedettek vagyunk, kattintsunk a ***Render (Leképezés)*** gombra.

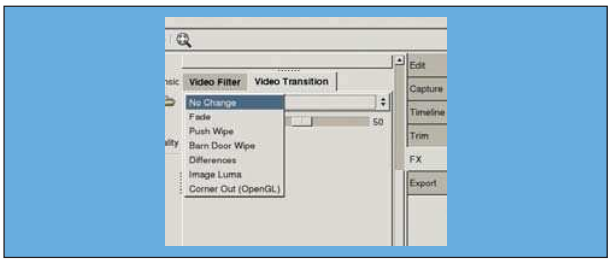
A ***Render*** gomb piros jelölése is fontosságát jelzi. Leképezés közben a ***Kino*** egy új fájlt állít elő. Minden eddigi módosítás az eredeti fájlokat érintetlenül hagyja, csak a ***SMIL*** fájlt érinti. Most azonban új fájl jön létre, ami azonnal be is kerül a tervezetbe.

Az ajtós hatást a ***Kino*** maga tartalmazza, további hatásokat a ***timfx*** révén érhetünk el. A *6. ábra* a hatások teljes listáját tartalmazza. Ezek jelentős része széles körben ismert, ilyen a feliratok kapcsán hamarosan szóba kerülő is.

A ***Kino*** a film sebességét is képes megváltoztatni. Ezzel a lehetőséggel vicces hatásokkal bővíthetjük a filmet, illetve



6. ábra A Video Filters (Videoszűrők) menü

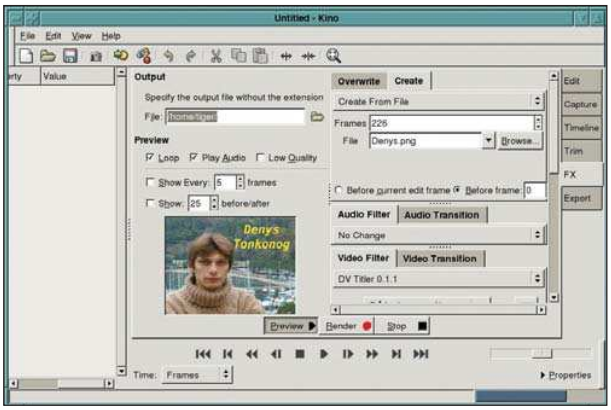


7. ábra A Video Transition (Videoátmenetek) menü

egyes pillanatokat meghosszabbíthatunk. A sebesség módosításához használjuk az ***Advanced Options (Különleges beállítások)*** alatt található csúszkát, illetve a ***Speed (Sebesség)*** gombot. A ***Reverse (Visszafelé)*** paranccsal a végéről indulva is lejátszhatjuk a filmet, egészen a kezdő pontig. A sebesség módosítása és a visszafelé lejátszás az átmenetekkel és a szűrőkkel együtt is használható.

Címek, feliratok készítése

Ha szöveget szeretnénk egy-egy epizódhoz hozzáadni, a ***dvtitler*** lehet segítségünkre. Példaként készítsünk egy egyszerű albumot, amely a filmünkben játszó karaktereket ismerteti. Megfelelő képszerkesztő, például a ***GIMP*** segítségével az albumba akár fényképeket is illeszthetünk. Először készítsünk képeket az egyes személyekről, majd méretezzük át ezeket. ***PAL*** szabványnál a képméret ***720×576***, ***NTSC***-nél pedig ***720×480*** képpont.
Példánkban feltételezzük, hogy az első személyt öt másodpercig akarjuk mutatni, ezután négy másodpercet hagyunk a váltásra, majd újabb öt másodpercet hagyunk a következő színészre is.
Az ***FX*** menüből válasszuk a ***Create (Létrehozás)*** parancsot; (5+4)sec*25=225 képkockára lesz szükségünk. A képkockák számát párosan tartjuk, ezért a Create From File (Létrehozás fájlból) paranccsal 226 képkockát illesztünk be, amint az a *8. ábrán* is látható. Ebben a lépésben egyben a ***„Denys Tonkonog”*** címet is megadjuk. Ez két középre rendezett sorból áll, és kezdetben a képkocka jobb felső részében helyezkedik el. Mivel végső helyzete szintén ez, a felirat a képkockákon belül nem mozog. Az X és az Y eltolás (offset) abban segít, hogy a teljes címet a képkockákon belül tartsuk, még akkor is, ha a filmet tévékészüléken vetítik majd, amelynek képmérete korlátozott lehet. A cím előnézete a *9. ábrán* látható.
Ugyanezzel a módszerrel készítünk egy ***„Olexiy Tykhomyrov”*** címet is. Ezután visszalépünk szerkesztés



8. ábra A fájlból létrehozott képkockák előnézete

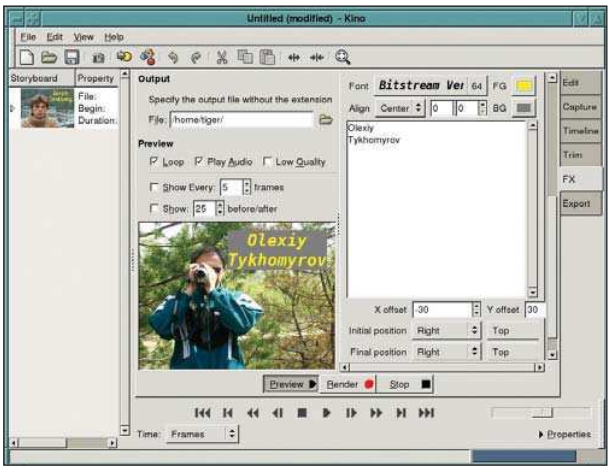
módba, majd a hatás beillesztéséhez a megfelelő helyeken szétvágjuk a filmet. Nem két címjelenetet készítünk, hanem négyet, ezek hossza öt, négy, négy majd újra öt másodperc. A két középső – négy másodperces – jelenetet az ***Image Luma (Világosság)*** hatással egyesítjük.
FX módban kattintsunk a második jelenetre a forgatókönyvben. A videóátmenetek közül válasszuk az ***Image Luma*** hatást. Következő lépésként a „világosság” (luma) képet kell megadnunk. Példánkban (*10. ábra*) egy normál, balról jobbra átmenetet tartalmazó fájlt választottunk (left_to_right.png). A módot ne feledjük beillesztésről felülírásra változtatni. Ha végeztünk a leképezéssel, négy helyett három jelenetünk lesz. A 3. ábrán az egymást váltó címek idővonala látható. A világosságot különféle fájlokkal használva sok érdekes képi hatást állíthatunk elő.

Hang

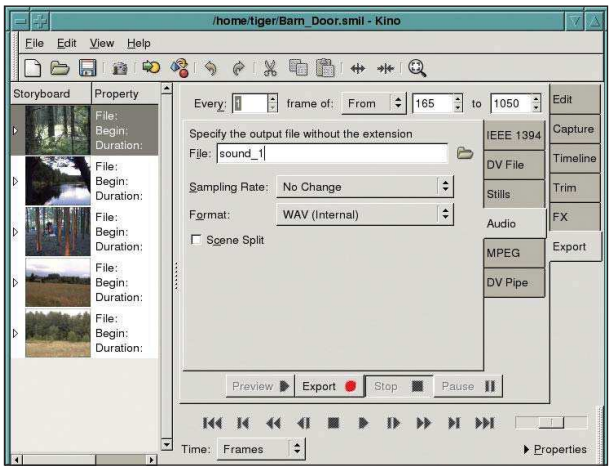
A ***Kino*** segítségével teljesen átalakíthatjuk a jelenetekhez tartozó hangot, hangokat adhatunk hozzájuk, zenét keverhetünk alájuk, de természetesen akár érintetlenül is hagyhatjuk őket. Az alákeverésre vagy hangcserére leginkább megfelelő zene megtalálása talán a munka legnehezebb része. Ha nem üzleti célra készítjük a filmet, látogassunk el a ***Creative Commons*** weboldalra.
A ***Kino*** a hangokat ***WAV*** formátumban, nem ***MP3***-ban kezeli. Ha ***MP3*** formátumban találtuk meg a filmünkhöz megfelelő hangot, zenét, akkor először WAV formátumra kell alakítanunk. Használjuk például a ***mpg123***-at:

```
mpg123 –wav pelda.wav pelda.mp3
```

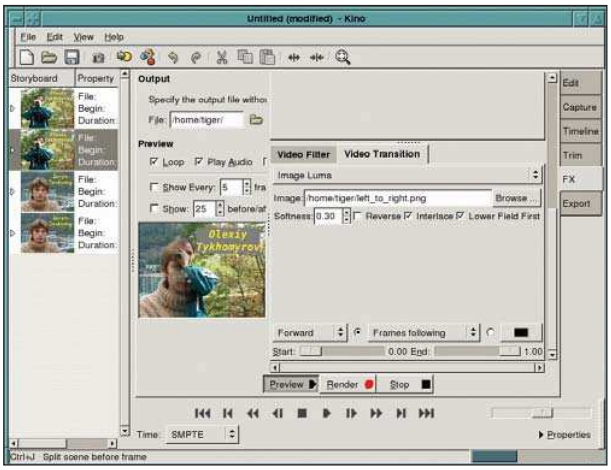
Ha élő hangot használunk, akkor jobb, ha kimásoljuk a hangsávot a ***.dv*** forrásból, a megvágott jelenetekkel ugyanis a hangok elvesznek. Hangszerkesztésre az ***snd***-t használjuk. Keressük meg a jelenetnek azt a kezdő és záró pontját, amelyek közé be szeretnénk illeszteni a hangot, majd váltsunk ***Audio (Hang)*** módba. Válasszuk ki a szükséges beállításokat, ide értve a ***From (Honnan)*** és a ***To (Hová)*** beállítást is. (*11. ábra*) Ne feledjük, hogy a kimeneti fájl nevét kiterjesztés nélkül kell megadnunk.
A ***Split (Vágás)*** szolgáltatással a hang beillesztése előtt egyesítsük az érintett jeleneteket. Ellenőrizzük, hogy a forgatókönyvben kiválasztottuk-e azt a jelenetet, amelyhez a hangot csatolni akarjuk. Váltsunk ***FX*** módba, majd válasszuk az ***Audio Transition (Hangátmenet)*** lapot. A ***Dub***



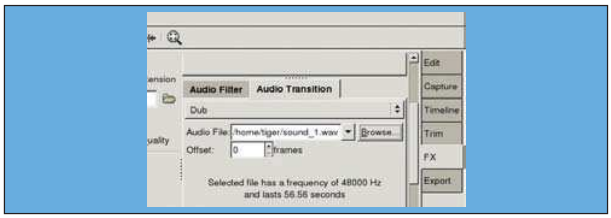
9. ábra Az „Olexiy Tykhomyrov” cím elkészítése



11. ábra A hangsáv kimásolása jelenetből



10. ábra A címek összefűzésének előnézete



12. ábra Az Audio Transition (Hangátmenet) menü

(Szinkronizálás) paranccsal a fájlban lévő hangot csatolhatjuk, a ***Mix (Keverés)*** paranccsal pedig keverhetjük azt a jelenet meglévő hangjával. Adjuk hozzá a hangot, majd indítsuk el az előnézetet. Szükség esetén játszadozzunk el a beállításokkal, majd kattintsunk a ***Render*** gombra. A leképezés eltarthat egy ideig.

A film kimentése

Miután minden összeállt, már csak az eredmény kimentése várat magára. Ugyan számos kimeneti formátum közül választhatunk, ha meg akarjuk őrizni az eredeti minőséget, akkor inkább írjuk vissza a filmet szalagra, így a merevlemezhellyel is takarékoskodni tudunk. Válasszuk az ***Export (Kimentés)*** lapot, majd a menü ***IEEE1394*** elemét. A kimentés megkezdése előtt hívjuk életre a ***dv1394*** eszközt, és töltsük be az illesztőprogramot. Először – kameránk típusától függően – az alábbi parancsok valamelyikét kell kiadnunk, ezzel hozzuk létre a ***dv1394*** eszközt.

PAL rendszernél a parancs a következő:

```
mknod -m 666 /dev/dv1394 c 171 34
```

NTSC rendszernél pedig az alábbi:

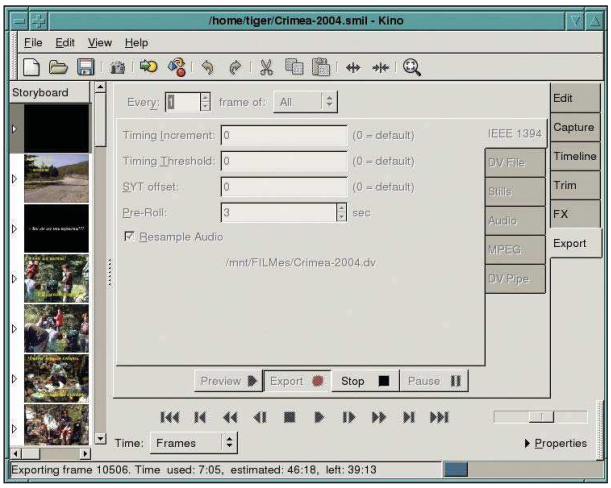
```
mknod -m 666 /dev/dv1394 c 171 32
```

Továbbra is rootként az alábbi parancsot kiadva töltsük be az illesztőprogramot:

```
modprobe dv1394
```

Ezután normál felhasználóként válasszuk a ***Kino Preferences->IEEE1394 (Beállítások->IEEE1394)*** parancsát. A *13. ábrán* láthatóhoz hasonló ablak jelenik meg. Ha a rögzítést a ***Kino***-val végeztük, akkor a ***DV Capture (DV rögzítés)*** részben a megfelelő adatok jelennek meg. A program mint ***VCR (AV/C)*** eszközt ismeri fel a kamerát. A kamerának bekapcsolva kell lennie, és értenie kell a vezérlőjeleket. (Ehhez általában lejátszás módban kell lennie.)
Mivel az ***IEEE1394*** felületen akarjuk kiküldeni a kimenetet, válasszuk ki a ***/dev/dv1394*** eszközt. Az ***OK*** gombra kattintva zárjuk be a beállító menüt. Kattintsunk az ***Export (Kimentés)*** elemre, majd válasszuk az ***IEEE1394***-et. Amit látunk, a *14. ábra* tartalmához fog hasonlítani.
Vegyük észre a 14. ábra alján látható információs sort. A ***Kino*** itt jelzi a ***dv1394*** eszköz elérhetőségét. Ha hibát jelez, várjunk 20-40 másodpercet. Ha a hiba ezután is fennáll, akkor a ***Linux1394*** honlapon olvassunk utána az eszköz működésének.
Következő lépésünk a ***Preview (Előnézet)*** kiválasztása. A filmnek meg kell jelennie a kamera kijelzőjén. Most állítsuk a kamerában lévő szalagot arra a pozícióra, amelytől a felvételt el szeretnénk indítani, majd a szalagra írás elindításához kattintsunk az ***Export*** gombra. Kézikönyve alapján ellenőrizzük a kamera jelzéseit. Valószínűleg valami olyat ír ki, hogy ***„DV Input” (DV bemenet)***, és lehet, hogy további adatokat is megjelenít.
Ha minden rendben van, állítsuk le a kamerát, tekerjük pontosan a kívánt helyre a szalagot, majd kezdjük meg

13. ábra A Kino beállítása a film kimentésére, IEEE1394 felületen keresztül, kamerára

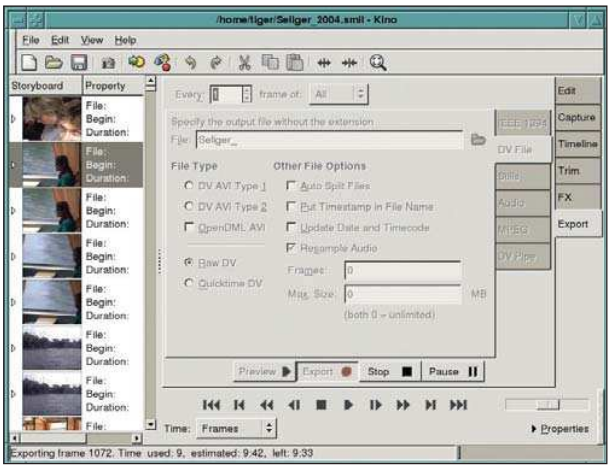


14. ábra A film visszaírása a kamerára

a kimentést. A szükséges idő hossza pontosan megegyezik a film játékidejével. Kimentés közben a ***Kino*** az ablak aljára írja ki az eltelt és a hátralévő időt.
A filmet fájlként vagy fájlok csoportjaként is elmenthetjük. Más programokkal kiegészítve a ***Kino*** a következő célokra használható még:

- ***DV*** fájl előállítása, amely a filmet új formátumban tartalmazza, és amely a ***Kino*** közreműködése nélkül további tömörítésre vagy kimentésre használható
- ***Stills (Állóképek)*** – A film képkockáinak kimentése *JPEG* formátumú fájlok sorozataként
- ***MPEG*** – A film ***MPEG*** vagy ***DivX*** formátumba kódolása
- ***Audio (Hang)*** – A hangsáv kimentése
- ***DV pipe (DV csővezeték)*** – Elegáns eszköz saját módszerek létrehozására a filmek kimentésére. Olyan alapszolgáltatásokat is támogat, mint a kimentés ***MPEG1*** (video CD készítéséhez) vagy ***MPEG2*** formátumba.

Ne feledjük, a kimentési szolgáltatások nem működnek a gépünkön, amíg a szükséges programokat nem telepítettük.
A ***DV*** fájlba való kiírás az ***IEEE1394*** felületen keresztül, kamerára való kimentéshez nagyon hasonló módon működik.
Ha a filmet egyetlen ***.dv*** fájlba szeretnénk kiírni, válasszuk



15. ábra Film kimentése .dv fáljba

az ***Export/DV File*** parancsot. Tiltsuk le az ***Auto Spilt Files (Önműködő vágás)*** szolgáltatást, majd az ***Export Range (Kimentési tartomány)*** beállításhoz válasszuk az ***All (Minden)*** értéket, ezzel a jelenetlista összes tagját ki tudjuk írni.
A ***Type File (Fájltípus)*** beállításnál a ***Raw DV (Nyers DV)*** értéket válasszuk. A ***Frame per File (Képkocka/fájl)*** és a ***Max File name (Max. fájlnév)*** mezőbe írjunk nullát, ahogy az a *15. ábrán* is látható. Kattintsunk az ***Export*** gombra. A ***Kino*** megkezdi a kiírást, miközben jelzi a folyamat befejezéséig hátralévő időt.

Záró megjegyzések

A ***Kino*** jelenleg is fejlesztés alatt áll, ezért ne lepődjünk meg, ha a szerkesztés vagy a metszés használata közben időnként összeomlik. Ügyeljünk filmünk rendszeres mentésére. Bár a ***Kino*** figyelemmel követi az általa létrehozott fájlokat, összeomlás esetén elveszítheti a fonalat. Merevlemezünk tisztán tartása érdekében ne felejtsük el törölni a felesleges ***.dv*** fájlokat.

Köszönetnyilvánítás

A szerzők szeretnék hálájukat kifejezni ***Paul Bartholdi*** professzornak, az olaszországi ***Nemzetközi Elméleti Fizikai Központ Genfi Obszervatórium*** munkatársának, amiért valódi internetkapcsolatot biztosított nekik, ami a ***Dnepropetrovski Nemzeti Egyetemről*** elérhetetlen lett volna.

Linux Journal 2004. december, 128. szám

Olexiy Tykhomyrov (tiger@ff.dsu.dp.ua) 1994 óta használja a Linuxot. A Dnepropetrovski Nemzeti Egyetem Kísérleti Fizika tanszékén dolgozik, fizikát és kommunikációt oktat. Imádja Misha nevű fiát, aki Tigrisnek hívja őt, amiért a diákjainak egy része fél tőle. Tigris imád úszni és utazni.

Denis Tonkonog, Tigris korábbi hallgatója, szintén a Dnepropetrovski Nemzeti Egyetem munkatársa. Kedvenc időtöltése az utazás és a fegyveres horgászás. Barátai Fekete Macskának hívják, de senki nem árulja el, miért. A denis@ff.dsu.dp.ua címen érhető el.

Bővítsük a GIMP-et

Miután megismerkedtünk a GIMP legfontosabb képességeivel, a használat során bizonyára felmerült bennünk a kérdés: tud-e ennél többet a program, vagy lehet-e valamilyen módszerrel a gyakran ismétlődő műveletek végrehajtását felgyorsítani. Természetesen a készítők már a kezdet kezdetén gondoltak erre a lehetőségre, így a válasz mindkét kérdésre igen!

A *GIMP* kétféle megoldást is kínál a bővítésre, mindkettőt egy-egy programnyelven keresztül. A korábbi változatokban a ***scheme*** programozási nyelvet használhatjuk, amelynek azonban kissé nehezen érthető a szintaxisa. A következő oldalakon az egyszerűbb megoldással fogunk foglalkozni, mégpedig a ***Python*** programnyelven készített beépülő modulok készítésével.

A modulok elkészítéséhez azért is célszerű a ***Python*** nyelvet választani, mert véleményem szerint ez az egyik legkönnyebben elsajátítható nyelv, mely egyszerűsége ellenére is alkalmas összetett programok elkészítésére. A nyelv egyik nagy előnye az is, hogy könnyedén illeszthető más programkönyvtárakhoz, így napjainkban már használhatunk például ***GTK+*** vagy ***Qt*** alapú grafikus felületet is a ***Python*** programjainkban.

Az alábbiakban a nyelv alapelemeit szeretném bemutatni, mindössze olyan mélységben, amennyire a ***GIMP*** modulok készítéséhez szükségünk lesz rá.

Minden programnyelvben találhatunk programvezérlési szerkezeteket, mint amilyenek a ciklusok és a feltétel vizsgálatra szolgáló utasítások. A ***Python***-ban egy számláló ciklus az *1. listán* látható módon készíthető el. Szintén az 1. listán látható az elöl tesztelő ciklus és a feltételvizsgálat formailag helyes megadása is. Ha jól megfigyeljük a listát azonnal látható a formátum egyszerűsége. A ciklus vagy vizsgálat kezdetét a `:` karakter jelzi, majd az utasítások következnek.

Az utasítások beírásánál figyelni kell arra, hogy a ***Python*** értelmező a tabulátorok vagy szóközök alapján határozza meg az utasításblokk végét. Tehát a második listán látható utasítások mindegyike helytelen, mert nem kezdtük beljebb az utasításblokkba tartozó sorokat. Aki írt már egyszerűbb vagy összetettebb programokat, annak nem lesz nehéz megszoknia a ***Python*** formai követelményeit sem, hiszen már az áttekinthetőség kedvéért sem írjuk folyamatosan, mindenféle tagolás nélkül a sorokat egymás alá.

A ***Pythonban*** természetesen lehetőségünk van a korszerű programozási nyelvekben megszokott függvények és osztályok használatára is. A *3. listán* látható egy függvény, mely megfelel a nyelv formai követelményeinek. A függvényünk nem csinál mást, csupán kiírja az első paraméter értékét, majd az első és a második paraméter összegét. A ***Pythonban*** nem szükséges sokat foglalkozunk a változók típusával, elegendő csak arra figyelni, hogy azonos típusú változókkal végezzünk műveleteket. Alapvetően három típussal dolgozhatunk. Számokkal, szövegekkel és listákkal. Természetesen nem adhatunk össze egy számot egy szöveggel és listával sem.

1. lista

```
# számláló ciklus
for a in range(1,10):
        utasítások

# elöl tesztelő ciklus
i = 0
while i<10:
        i = i + 1

# feltételvizsgálat
if FELTÉTEL:
        utasítások
else:
        utasítások
```

2. lista

```
# számláló ciklus helytelen
for a in range(1,10):
print a

# elöl tesztelő ciklus helytelen
i=0
while 1:
print i
if i==10:
break
```

3. lista

```
# Függvény megadása
def fuggvenyneve(parameter1, parameter2):
        utasításblokk
        print parameter1
        print parameter1 + parameter2
```

4. lista

```
#!/usr/bin/env python

from gimpfu import *

# A fo fuggveny, ami a munkat vegzi
def anigif(image, drawable, neww, newh,
filename):
    newimage = pdb.gimp_file_load(filename,
    ➥filename)
    newlayer = newimage.active_layer
    newlayer.scale(neww, newh, 0)
    pdb.gimp_selection_all(newimage)
    pdb.gimp_edit_copy(newimage.active_layer)
    mylayer = gimp.Layer(
            ➥image, "from " + filename,
            ➥neww, newh,
            ➥image.base_type, 100, NORMAL_MODE)
    image.add_layer(mylayer, 0)
    mylayer.set_offsets((image.width-neww) / 2,
            ➥(image.height-newh) / 2)
    image.active_layer = mylayer
    pdb.gimp_edit_paste(mylayer, 1)
    pdb.gimp_floating_sel_anchor(
       image.floating_selection
    )

# Modul bejegyzese
register(
    "python_fu_anigif",
    "Animated GIF creator",
    "Create animated GIF image from single
    ➥images",
    "Zoltan Fabian",
    "Zoltan Fabian",
    "2004",
    "<Image>/Python-Fu/GifAnim",
    "RGB*",
    [
       ( PF_INT, "neww", "New width", 100),
       ( PF_INT, "newh", "New height", 100),
       ( PF_FILE, "filename", "Filename", "")
    ],
    [], anigif)

main()
```

5. lista

```
register (  modul_neve, "modul leírása",
➥"modul bővebb leírása",
➥"készítő_neve", "copyright információk",
➥"dátum",
➥"modul_helye_a_menüben", "KÉPTÍPUSOK",
➥paraméterek[], visszatérési_értékek[],
➥a_modul_fő_függvénye
 )
```

Mindössze ennyi elegendő a modulok készítésének megértéséhez, tehát a most egy példán keresztül bemutatom kedves olvasóimnak, hogyan is lenne célszerű elkészíteni egy *GIMP* kiegészítő modult. A *4. listán* látható a példa, igyekeztem minél rövidebben használható rutint írni, de természetesen a továbbiakban bővíteni is fogjuk a programunkat. A *GIMP* alapértelmezésben a megadott könyvtárakban keresi a kiegészítő modulokat, tehát a beállítások párbeszédablakában meg kell határoznunk a modulokat tartalmazó könyvtárakat. Amennyiben ezt a lépést kihagyjuk, a program a felhasználó saját könyvtárában lévő ***.gimp-2.0/plug-ins*** könyvtárban vizsgálja az állományokat. A ***Python*** nyelven készült modulok írásakor két fontos dolgot kell megjegyeznünk. Az egyik, hogy a `from gimpfu import *` sornak szerepelnie kell a modul forrásában, mert így használhatjuk a *GIMP* beépített lehetőségeit. A másik fontos dolog pedig, a `register` függvény meghívása a megfelelő paraméterekkel. A függvény paraméterezése az *5. listán* olvasható.
A paraméterek közül bővebb magyarázatra szolgál a modul helyét meghatározó `'modul_helye_a_menüben'` paraméter. Ez a paraméter határozza meg, hogy a *GIMP*-ben milyen menüpontokon keresztül találjuk meg a kiegészítő modult. A megfelelő formátum az alábbi példákból jól látható:

```
"<Image>/Python-Fu/AnimGif"
vagy
"<Toolbox>/Xtns/Python-Fu/AnimGif"
```

A *GIMP* indulásakor alapértelmezésben nincs megnyitva semmilyen kép, tehát amennyiben a *GIMP* eszköztárának menüpontjai között szeretnénk találkozni az elkészített modullal, akkor a második megoldás szerint a az eszköztárban az ***Xtns*** menüben a ***Python-Fu*** pontot választva találhatjuk meg az ***AnimGif*** modult, melynek forrásával a 4-es listán ismerkedhettünk meg. A másik fontos paramétere a `register` függvénynek, a `KÉPTÍPUSOK` nevet viseli. Ezzel a paraméterrel határozhatjuk meg, hogy a *GIMP* milyen képekre engedélyezze a modul használatát. A típus lehet *'RGB*'*, *'GRAY*'* vagy *'INDEXED'* de természetesen a * karakter helyett használhatjuk a pontos értéket is, mely meghatározza az elfogadható színmélységet. Például, amennyiben csak 24-bites képekre használható a modul, akkor a megfelelő érték az `RGB24` lesz. Több értéket megadhatunk a karakterláncban, vesszővel elválasztva.
Moduljaink készítése során előfordulhat, hogy a modulnak paramétereket szeretnénk átadni a *GIMP*-ből. Meg kell jegyezni, hogy minden modul fő függvényének első

6. lista

```
PF_INT
PF_FLOAT
PF_STRING
PF_INTARRAY
PF_FLOATARRAY
PF_STRINGARRAY
PF_COLOR
PF_REGION
PF_IMAGE
PF_LAYER
PF_CHANNEL
PF_TOGGLE
PF_BOOL
PF_FONT
PF_FILE
PF_BRUSH
PF_PATTERN
PF_GRADIENT
```

két paraméterében a *GIMP* átadja az éppen aktuális rajzolásra használt felületet és az aktuális képet, tehát a fő függvénynek legalább két paraméterrel kell rendelkeznie a definícióban is. E két alapértelmezett paraméteren kívül a `paraméterek[]` paraméterben adhatunk meg továbbiakat. Ahogyan a 4. listán is látható, a szögletes zárójelben (python lista) vesszővel elválasztva felsoroljuk a szükséges paramétereket. A paraméterekről is háromféle információt szükséges a *GIMP* tudtára adni, mégpedig a paraméter típusát, a párbeszédablakban megjelenő szöveget és a paraméter alapértelmezett értékét. A leggyakoribb paraméterek típusa a *6. listán* olvasható. A négyes listán látható példában könnyen nyomon követhető a paraméterek meghatározásának módja.

Végül a modul használatba vételéhez a regisztráció elvégzése után meg kell hívni a `main()` függvényt, mely a *GIMP* beépített modulkezelő függvénye, tehát semmi más tennivalónk nincs a modul működtetéséhez.

Tehát amikor a *GIMP* elindul, végignézi az elérési utakat ahol modulokat és egyéb kiegészítőket tárolhatunk, majd a saját készítésű moduljainkat megtalálva meghívja az abban található `register` függvényt. Ezzel készen is van a modul alaphelyzetbe állítása és a megadott néven, a megfelelő menüpontokon keresztül megtalálhatjuk a *GIMP* menüjében.

A programozási alapismeretek elsajátítása után, térjünk rá első modulunk tárgyalására. A modul célja, hogy az animált *GIF* képek elkészítését megkönnyítsük. A kiegészítő modulok használata nélkül minden képet, melyet a végeredményen látni szeretnénk, meg kellene nyitnunk, majd az átméretezés után mindent kijelölni és a vágólapra helyezni. A másolás után átváltunk az készülő képre, majd ott beillesztjük a másolt képrészletet és új rétegként létrehozzuk a következő képkockát. Ez a folyamat ugyan csak néhány billentyűlenyomást igényel, kis gyakorlással már észre sem vesszük az mozdulatokat, de mégis célszerűnek tartom egy modulba foglalni az ismétlődő műveleteket.

1. táblázat *Változók és metódusok*

Változók	
`bpp`	A réteg színmélysége
`height`	Magasság
`width`	Szélesség
`image`	A kép, melyhez a réteg tartozik
`is_floating_selection`	Nem 0 az értéke, ha a réteg egy kiválasztott rész
`mask`	A rétegmaszk, ha létezik
`mode`	Kirajzolás módja
`name`	A réteg neve
`opacity`	A réteg átlátszósága

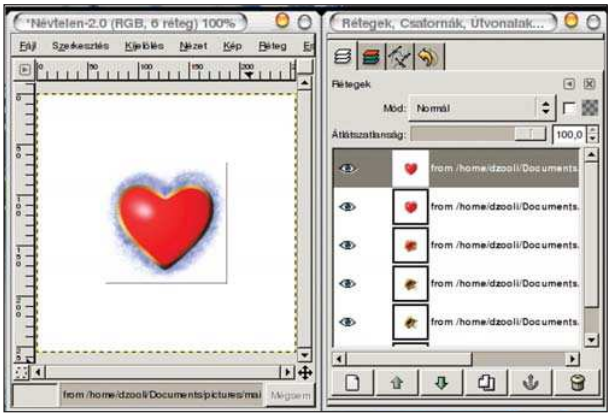
Metódusok	
`add_alpha()`	alfa csatorna létrehozása
`copy()`	másolat a rétegről
`create_mask(típus)`	rétegmaszk létrehozása
`resize(w,h,x,y)`	méret változtatása [w,h] méretre
`scale(w,h,közép)`	átméretezés
`set_offsets(x,y)`	eltolás a [0,0] ponthoz képest
`translate(x,y)`	eltolás az aktuális helyzethez képest

Tekintsük át a 4. listán látható programot. A program elején látható, hogy a forrást a *Python* értelmező fogja végrehajtani és az is, hogy a *GIMP* modulkiegészítőit is használni szeretnénk. A következő fontos dolog, maga a műveleteket végrehajtó függvény. Az `anigif` függvény a *GIMP*-től megkapja az aktuális rajzterületet és a képet, majd a `register` eljárás meghívásakor bejegyzett egyéb változókat.

Első lépésként betölti a `filename` paraméterben kapott képet, mégpedig a *GIMP* beépített függvényének segítségével. A betöltés után a képről megállapítja az aktív réteget (ez maga a kép lesz, amennyiben nem *Photoshop* vagy *GIMP* formátumú képet töltünk be) és azt átméretezi a kívánt méretre. A szükséges méreteket szintén a modul indulásakor megjelenő párbeszédablakban állítjuk be, majd a `neww` és `newh` paramétereken keresztül a fő függvényben használhatjuk. Az átméretezés után szintén a GIMP beépített függvényadatbázisából választjuk ki a megfelelő függvényt, melynek segítségével az aktív réteget teljes egészében kijelöljük és a vágólapra másoljuk.

A következő lépés célszerűen az lenne, hogy a célképen a beillesztendő képnek létrehozunk egy új réteget. A *GIMP* minden működése elérhető a *Python* felületen keresztül is, így létrehozhatunk új réteget is a `gimp.Layer` objektum létrehozásával. Az objektum létrehozásához hét paraméterre van szükség, melyek sorrendben a következők: első a kép, melyhez a réteg tartozik. A második a réteg neve, amely

1. kép Animált GIF-et készítő modul



2. kép A modul használat közben

majd a GIMP réteg-kezelőjében is megjelenik. A harmadik és a negyedik határozza meg a kép méretét, míg az ötödik a réteg típusát. A réteg típusa lehet `RGB_IMAGE`, `RGBA_IMAGE`, `GRAY_IMAGE`, `GRAYA_IMAGE`, `INDEXED_IMAGE` vagy `INDEXEDA_IMAGE`. A hatodik paraméter a réteg átlátszatlanságának mértékét határozzuk meg. 100-as értéknél a réteg egyáltalán nem lesz átlátszó, míg nulla értéknél teljesen átlátszó réteget kapunk. A hetedik paraméter pedig a réteg kirajzolási módját adja meg, amit most `NORMAL_MODE`-ként határozunk meg, tehát a réteggel semmilyen műveletet nem kell végezni a kirajzoláskor.
Az *1. táblázatban* egy rövid összefoglaló található a `gimp.Layer` objektum változóiról és metódusairól.
Az új réteg létrehozása után már csak annyi dolgunk maradt, hogy azt a képhez hozzáadjuk és beállítjuk a megfelelő eltolást és átméretezést, hogy az cél-kép közepére kerüljön a beillesztett kép. A beillesztéshez ismét a *GIMP* egyik menüpontjához tartozó eljárását hívjuk meg, mégpedig a `pdb.gimp_edit_paste` metódust. Itt látható, hogy minden beépülő modul vagy elérhető függvény használható a *PDB (Plug-in DataBase)* adatbázison keresztül, tehát itt is érvényes a bővíthetőség és az újra felhasználhatóság elve. A réteg beillesztése után a fő függvényünkben az utolsó feladat már csak a kijelölés megszüntetése, vagyis a beillesztett tartalom rögzítése.
A modul további sorai tartalmazzák a *GIMP* adatbázisába történő felvételt (a `register` függvény segítségével) és a modul elindítását.
Végül az egyes rétegek beillesztése után még egy egyszerű feladatot kell megoldanunk, hogy az animált *GIF* kép ne tartalmazzon üres képkockát. A két legalsó réteget egyesíteni kell a réteg-kezelőben a *Merge Down* menüpont kiválasztásával. Mindezek után máris menthetjük *GIF* formátumba az elkészült képet. A *GIMP* érdeklődni fog a mentés formátuma iránt, itt válasszuk ki az animált *GIF* lehetőséget és készen is vagyunk első modulunk tesztelésével. Az *1-es képen* látható a modul beállító párbeszédablaka, amelynek felületét a `register` eljárás meghívásakor hozzuk létre.
A *2-es képen* a modul segítségével beillesztett rétegekből álló kép részletei tekinthetők meg.
Amint látható, nem túl bonyolult dolog egy *GIMP* modult elkészíteni, köszönhetően annak, hogy a *Python* nyelv támogatása is belekerült a *GIMP* nyelvezetébe. Felmerülhet azonban az a kérdés kedves olvasóimban, hogy honnan tudhatjuk meg, milyen lehetőségek rejlenek a háttérben, milyen feladatokat végezhetünk el egy-egy rövid programsor segítségével.
Mivel a *GIMP* tartalmaz egy modul- és függvényböngésző párbeszédablakot, így könnyedén utána járhatunk a meglévő modulok képességeinek.
A *GIMP* fő ablakában található az ***Xtns -> Python-FU -> PDB Browser*** menüpont, melynek segítségével minden beépülő modulról vagy függvényről bővebb információkhoz juthatunk. Amint megnyitjuk ezt a párbeszédablakot, a rendelkezésre álló függvények listáját láthatjuk és kedvünkre válogathatunk közöttük. Egy függvényt kiválasztva az ablak jobb oldali részén annak rövid leírását, paramétereit és használatát segítő szöveges leírását olvashatjuk. Az ablakban lehetőségünk van a függvények közötti keresésre is, tehát igazán néhány óra olvasgatással hamarosan otthon érezhetjük magunkat a *GIMP* moduljainak körében.
A másik lehetőség az ismeretek bővítésére szintén a fő eszköztár menüjén keresztül érhető el. Az ***Xtns -> Python-Fu -> Console*** menüpontok kiválasztásával megjelenik egy szövegbevitelre alkalmas párbeszédablak, ahol írhatunk különféle *Python* nyelven értelmezhető parancsokat, melyeknek kimenete az ablak felső részét elfoglaló mezőben azonnal láthatóvá válik. Első lépésben mindig töltsük be a ***gimpfu*** modult, hogy a *GIMP* saját függvényei is elérhetővé váljanak. Ezt az `import gimpfu` utasítás begépelésével érhetjük el, ahogyan a modul írásakor is tettük. Ezután már a Python nyelvben használatos `dir` függvény alkalmazásával minden objektumról megjeleníthetjük a metódusokat és az objektum változóit. Bővebb leíráshoz juthatunk, ha a `import pydoc` utasítással a dokumentációt segítő modult is betöltjük majd a `pydoc.doc(VALAMI)` függvény használatával kérünk segítséget a kérdéses objektumról. A konzol azonban nem csak ismerkedésre alkalmas, hiszen segítségével a modulokat még a tényleges elkészítés előtt részleteiben is ellenőrizhetjük, továbbá kipróbálhatjuk a meglévő függvényeinket is.
Meg kell jegyeznem, hogy amikor a *GIMP*-et kiegészítjük valamilyen modullal, a regisztráció után más modulokban is használhatjuk a saját függvényeinket, tehát megtalálhatjuk a *PDB Browser* listájában is.
A következő részben folytathatjuk az ismerkedést bonyolultabb feladatok megoldásával, addig is bizonyára az érdeklődő olvasók sikerrel veszik a kezdeti nehézségeket.
Addig is mindenkinek kellemes ünnepeket és Boldog Új Évet kíván a szerző.

Fábián Zoltán

A felhasználói viselkedés vizsgálata (3. rész)

A sorozat előző részében megvizsgáltuk az adatgyűjtési módszereket. Most már ideje használni is ezeket, hogy a legyen mit elemeznie a Nagy Testvérnek.

A legjobb megoldás

A legfrissebb adatok alapján az internet legelterjedtebb webkiszolgálója az ***Apache***, a maga közel 67%-os részesedésével. Éppen ezért döntöttem úgy, hogy a modellezést ennek a programnak az eseménynaplói alapján fogom bemutatni. A választást az is megerősíti, hogy a témával kapcsolatban a ***Google*** közel 160 olyan szoftvert talált, amelyek az eseménynapló valamiféle feldolgozásával, statisztikák készítésével és modellezéssel foglalkozik.
Az ***Apache*** előnye, hogy teljesen nyílt forrású, rengeteg platformon és operációs rendszeren fut (***Unix***, ***Linux***, ***BSD***, ***Microsoft Windows***, ***Novell Netware***), széleskörűen támogatott, és rengeteg kiegészítő modul érhető el hozzá. Beállítása egyszerű, egészen szélsőséges terhelési viszonyok között is használható, kiegészítő modul használatával pedig akár elosztottan is futtatható. Az internetes közösség folyamatosan jelentkezik újabb és újabb modulokkal. (Van például adatbázisba naplózó modul is.)

Apache naplózás

Az ***Apache*** naplózó modulja a ***mod_log_config*** (➲ ***http://httpd.apache.org/docs/mod/mod_log_config.html***), amely alapból feltelepül minden fent említett rendszeren. Az eseménynaplóba kerülő bejegyzések részletessége igen változatos lehet, a modellezés leghatékonyabb támogatásához azonban érdemes a legrészletesebben naplózni.
Az ***Apache*** webszerveren a `LogFormat` direktíva segítségével adható meg, hogy milyen adatok és milyen sorrendben kerüljenek az eseménynapló egy bejegyzésébe. A lehetséges direktíva értékek:

- `%a`: az ügyfél IP címe,
- `%A`: a kiszolgáló (helyi) IP cím,
- `%b`: az elküldött adat nagysága (Common Log Format formátum),
- `%c`: a kapcsolat státusza a kérés teljesítése után,
- `%f`: a lekért erőforrás (fájl) neve a helyi fájlrendszeren,
- `%h`: az ügyfél neve,
- `%H`: a kérés protokollja,
- `%l`: a távoli eseménynapló neve (általában üres),
- `%m`: a kérés metódusa,
- `%q`: a lekérdezést tartalmazó karakterlánc (***query string***), kiegészítve egy kérdőjellel ha létezik, egyébként üres, (például: `?action=register&step=3&sess=12345678`)
- `%r`: a kérés első sora, például: "`GET /modul.php?action=register ➥&step=3&sess=12345678 HTTP/1.0`",
- `%s`: a kérés kiszolgálásának státusza (HTTP protokoll státusz kód),
- `%t`: időbélyeg (***Common Log Format***),
- `%T`: a kérés kiszolgálására fordított idő, másodpercben,
- `%u`: távoli felhasználó (azonosításból származik),
- `%U`: a kért URI a ***query string*** nélkül, például: "`/modul.php`",
- `%v`: a kiszolgáló szerver neve.

Amint azt már a sorozat korábbi részeiben említettem a fenti adatok alapján nem lehet egyértelműen elkülöníteni az egyes felhasználókat, főleg ha azok azonos proxy vagy tűzfal mögül látogatják az elemezni kívánt portált.
A modul lehetőséget nyújt az ügyfél felől érkező kérésekben található változók értékének eseménynaplóba írására is. Ennek segítségével tudjuk naplózni az ügyfél böngészőjének típusát és a „előző oldal" vagy küldő oldal (***referer***) értékét is. Ehhez a következő direktívákat kell használnunk:

- `%{Referer}`: a referer értéke,
- `%{User-Agent}`: a böngésző típusa.

A direktíva-értékek tetszőlegesen kombinálhatóak.
Íme néhány példa:

- Általános: "`%h %l %u %t \"%r\" %>s %b`"

Példa:

```
217.20.135.85 - - [08/Apr/2004:18:06:47 +0200]
➥"GET / HTTP/1.0" 200 16897
```

- Összetett: "`%h %l %u %t \"%r\" %>s %b ➥\"%{Referer}i\" \"%{User-agent}i\"`",

Példa:

```
62.165.209.144 - - [08/Apr/2004:18:30:55 +0200]
➥"GET / HTTP/1.1" 200 16937
```

```
"http://www.oktaton.hu/index.php?inc=portal"
➥"Mozilla/4.0 (compatible; MSIE 6.0; Windows
➥NT 5.1)"
```

- Egyedi: "%h %l %u %t \"%r\" %>s %b
  ➥\"%{Referer}i\" \"%{User-Agent}i\"
  ➥\"%{Cookie}n\" %T %v \"%U\" %c %f",

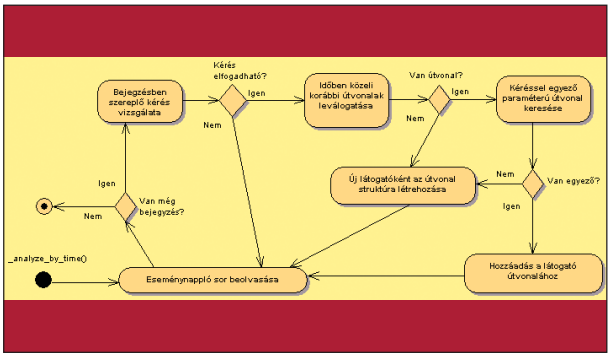
Példa:

```
42.165.229.144 - - [08/Apr/2004:18:30:55 +0200]
➥"GET / HTTP/1.1" 200 16937 "http://www.oktaton.hu/
➥index.php?inc=portal" "Mozilla/4.0 (compatible;
➥MSIE 6.0; Windows NT 5.1)"
➥"42.165.229.144.38811081441855459" 0
➥www.jegyzet.com "/" -
➥/srv/www/www.jegyzet.com/index.php
```

Látható, hogy a felhasználói viselkedés modellezésének pontossága jelentősen növelhető a ***referer*** és a böngésző típusára vonatkozó adatok felhasználásával. Az egyes weblapok (különböző tartalom szerinti) elválasztását a %q és %U direktívák értékének felhasználása könnyíti meg, ugyanis azonos %U értékek esetén a %q értékek különbözősége más-más weboldalt jelent. Ugyanakkor elmondható, hogy az általánosan használt eseménynapló bejegyzés formátumok a modellezés szempontjából felesleges adatokat is tartalmaznak (például: elküldött adat mérete, távoli eseménynapló neve), azonban az általánosan szokásos biztonsági naplózás miatt ezeket általában nem szokás eltávolítani.
A legjelentősebb probléma a fenti eseménynapló szerkezetekkel, hogy nem tartalmaznak a felhasználókat egyértelműen megkülönböztető bejegyzéseket: a %u paraméter értéke általában üres, mert a felhasználói azonosítást a legtöbb helyen nem a webszerver végzi, hanem a portál alkalmazása. Gyakori, hogy egyes erőforrásokat a webszerver által biztosított hitelesítési mechanizmussal (is) védenek, jellemző azonban, hogy az egyes felhasználók ugyanazt a felhasználónév és jelszó párost használják, azaz az így keletkező adat nem használható a felhasználók megkülönböztetésére.

Egyedi felhasználói azonosító

Szerencsére van megoldás, ugyanis az Apache webszerverhez létezik egy modul, amely segítségével minden egyes felhasználóhoz egyedi azonosító rendelhető.
Ez a ***mod_usertrack*** modul (➲ ***http://httpd.apache.org/docs/mod/mod_usertrack.html***).
A modul segítségével minden felhasználóhoz egy munkamenet azonosító rendelhető, függetlenül a kiszolgált, futtatott portál szoftverétől.
A modul működése igen egyszerű: a webszerver minden kérés esetén megkapja a ***HTTP*** fejlécekben a korábban az adott weboldalra juttatott sütiket. A modul megvizsgálja, hogy létezik-e a munkamenet azonosító süti, és ha úgy találja, hogy az létezik, és még ráadásul érvényes is, akkor egyszerűen visszajuttatja a klienshez. A visszaküldést a webkiszolgáló automatikusan végzi a HTTP fejlécben, a süti kezelés módszereinek megfelelően.
Amennyiben nem talál munkamenet azonosító sütit, akkor a modul automatikusan generál egyet. Minden munkamenet azonosítónak egyedinek kell lennie, ami egy nagy (sok ezer lekérés percenként) forgalmú portál esetén nem olyan triviális feladat, mert a szoftveres véletlenszám generátorok hajlamosak ismételni (szekvenciába esni). A modul az egyedi munkamenet azonosítót a következőképpen konstruálja meg:



1. ábra A kérések idő alapú elkülönítésének algoritmusa

- prefix karakterlánc (tetszés szerint beállítható),
- az ügyfél IP címe, vagy a helyi gép (***host***) neve,
- a kérést kiszolgáló folyamat (***process***) azonosítója,
- a kérés időbélyege másodperc pontossággal,
- a kérés időbélyege mikromásodperc pontossággal.

A modul a fenti adatokat összefűzi, és az így előálló karakterlánc lesz az egyedi munkamenet azonosító.
A modul megbízhatósága általában véve a szerver oldali adatgyűjtésnél ismertetett problémák miatt nem 100%-os. A kliens oldalon a sütik elfogadásának letiltásával a felhasználó minden egyes letöltésekor új munkamenet azonosító keletkezik, hiszen a böngésző nem tárolhatja a sütiket, azaz nincs hol tárolni az azonosítót. Az újabb böngészőkben lehetőség van arra, hogy a munkamenet azonosító sütik ne kerüljenek letiltásra. Ezt a modul megfelelő konfigurációjával segíthetjük elő.
A modul a következő beállításokkal vezérelhető (a beállításokat az ***Apache*** kiszolgáló beállítási fájljában kell elhelyezni):

- `CookieDomain`: a süti érvényessége, egyes böngésző beállítások esetén a böngésző csak az éppen használt szervertől fogad el sütit, ezért explicit módon meg kell adni, hogy melyik tartományhoz (***domain***) tartozik a süti.
- `CookieExpires`: ezzel az értékkel tulajdonképpen azt mondjuk meg, hogy mennyi ideig lehet a felhasználó egy látogatáson belül inaktív (nem kér le újabb oldalt a szervertől). Ha letelik a megadott idő, akkor a következő oldallekérés esetén a felhasználót új látogatóként kezeljük. Könnyen elképzelhető, hogy közben másvalaki ült le a géphez.
- `CookieName`: opcionális, a süti elnevezése.
- `CookiePrefix`: az azonosító elejére kerülő tetszőleges karakterlánc, opcionális.
- `CookieStyle`: a létező süti szabványoknak megfelelő típusú süti küldés beállítása (nagyon régi böngészők használata esetén lehet szükséges a módosítása).

- `CookieTracking`: mivel a modul betöltése esetén nem kerül automatikusan használatra, ezért explicit módon kell bekapcsolni.

Egy tipikus konfigurációs példa:

```
CookieDomain ".jegyzet.com"
CookieExpires "30 minutes"
CookieName "www-jegyzet-com-user-tracking"
CookieStyle "Cookie"
CookieTracking "on"
```

A modul beállítási lehetőségei közül a legfontosabb a `CookieExpires`, mert ennek értéke akár jelentősen is befolyásolhatja a felhasználói modellezést. A paraméter értékét minden esetben az adott portál felhasználási viszonyaihoz kell megállapítani figyelembe véve a látogatók áramlását és szokásait.
A modul egyetlen feladata tehát az eseménynapló bejegyzések kiegészítése egy munkamenet azonosítóval, amely azonosító a feldolgozás során a felhasználók szétválogatását, megkülönböztetését szolgálja.

Felhasználók elkülönítése

Ha valamilyen oknál fogva nem tudjuk használni a ***mod_usertrack*** modult, akkor is van lehetőség a felhasználók elkülönítésére.
Ebben az esetben a látogatók elkülönítése és útvonalaik összegyűjtése elsődlegesen az IP cím, távoli eseménynapló, távoli felhasználó adatok alapján történik, de az ismertetett problémák miatt ez önmagában nem elegendő az egyértelmű megkülönböztetéshez. További segítséget nyújt a ***referer*** és a böngésző adatok jelenléte. Azonban a legegyszerűbb szerkezetű eseménynaplók nem tartalmaznak az IP címen és a kérés időpontján kívül több olyan adatot, amely segítené a látogatók elkülönítését, vagy egy nagyvállalati hálózatból érkező látogatók esetén jellemzően a böngésző adatok is azonosak.
Az idő alapú elkülönítés során (*1. ábra*) az éppen vizsgált kérést a megadott időintervallumon belül eső, kéréssel rendelkező látogatóhoz próbáljuk meg csatolni, oly módon, hogy a fent említett adatok közül a lehető legtöbb egyezzen. Teljes egyezés esetén a vizsgált kérést az adott látogatóhoz kapcsoljuk, ha nem találtunk teljes egyezést, akkor mint új látogató kezeljük, és egy új útvonal tömb bejegyzést nyitunk neki.

Folytatás

A sorozat következő részében az elkülönített felhasználóink útvonalait fogjuk részletesen kielemezni és elkészítjük az átlagos viselkedést bemutató modelleket.

Beszédes Balázs (beszedes@ei.hu)
24 éves, az e-Média Informatikánál mérnök-informatikus. Hobbija a kerékpározás és a kirándulás.

FreeBSD – a szomszéd vár (3. rész)

A FreeBSD alaprendszer és a külső programok telepítése.

Hajdanán bőven megelégedtünk volna programok oly bő választékával, amit egy ***FreeBSD*** alaprendszer nyújt: programozható parancssor (burok), szövegszerkesztő programok, fordítóprogramok (GNU C), hálózati munkát segítő programcsomag, s még sok száz kis programocska, amelyek megkönnyítik a munkánkat. Manapság ezen programok megléte teljesen természetes, s egyben az „igazi" munkához kevés.

Az alaprendszer telepítése

Ha minden igaz, akkor rendelkezünk egy friss és naprakész alaprendszer forrással; sőt, talán még a helyes beállítások is már régen megtörténtek. Nincs más teendő, mint lefordítani, majd a helyére másolni az új rendszermagot és az új alaprendszert. Ez egyszerűnek hangzik, de valójában is egy egyszerű művelet. Eleinte azonban bonyolult és összetett tevékenységnek gondoljuk, de hónapok alatt rutinná, majd „szertartássá" válhat.

Ha valaha fordítottunk már mások által írt alkalmazásokat, akkor nem lesz ismeretlen a `make` program használata. Az alaprendszer fordítását és telepítését egy – a fejlesztők által írt – ***Makefile*** vezényli le, feladatunk ennek a folyamatnak az elindítása, illetve (nem túl megterhelő) megfigyelése. Felmerülhet a kérdés, hogy ha az alaprendszer és a rendszermag együtt biztosít megfelelő működést, akkor mi módon tudjuk a lehető legkevesebb ideig tartó kiesést elérni. Megnyugtatok mindenkit, erre megvannak a megfelelő módszerek, nézzük:

```
$ cd /usr/src/
$ make buildworld
$ make buildkernel KERNCONF=SAJAT
$ make installkernel KERNCONG=SAJAT
```

Az első sor önmagáért beszél, a többiről már írnom kell. A `buildworld` hatására készül el az új alaprendszer bináris tükörképe, amelyet egyelőre a */**usr**/**obj**/* könyvtárban találhatunk meg. Ennek a fordításnak az ideje a gép tudásától függően 10-15 perctől (többprocesszoros nagyszámítógép) akár több napig is eltarthat (egy olcsó 486-os PC). A fordítás sikeres végét arról ismerhetjük meg, hogy visszakaptuk a parancssort, és az utolsó egy-két képernyőnyi szövegben nem látjuk az „*error*" szót. Hasonló feladatot lát el a `buildkernel` is, amelynek van egy paramétere, mégpedig annak a konfigurációs állománynak a neve, amely az új rendszermag beállításait tartalmazza. Ezt el is hagyhatjuk, ekkor ***GENERIC*** rendszermagot fogunk fordítani, amely az esetek nagy részében jó kompromisszum. A rendszermag fordítása nagyjából negyedannyi időt vesz igénybe, mint az alaprendszeré. Az utolsó parancs a lefordított rendszermagot telepíti a helyére, ezáltal a következő rendszerindításkor már az új rendszermag fogja működtetni a gépünket. Ezen kívül semmi lényeges módosítás nem történt a rendszerben, minden úgy működik, ahogy eddig tette: az alaprendszer változatlan, az új rendszermag pedig nem aktív, hiszen ehhez újra kell indítanunk a gépet. A fordítások befejeztével kiválasztunk egy kellemes és ***tervezett*** időpontot az újraindítás elvégzéséhez, amikor képesek vagyunk a konzolhoz fizikailag hozzáférni, mivel az elkövetkező pár perces munkát mindenképpen az úgynevezett ***singleuser*** módban ***kell*** elvégeznünk. Természetesen lehetőségünk van ***multiuser*** módban is elvégezni, azonban ez nem támogatott a fejlesztők részéről, így előfordulhatnak nem várt komplikációk. A ***singleuser*** módot (jelenleg) a negyedik menüponttal választhatjuk ki, amikor újraindítjuk a gépünket.

```
$ cd /usr/src
$ mergemaster -p
$ make installworld
$ mergemaster
```

Új fogalom jelent meg: ez a `mergemaster`, amely összetett feladatot hajt végre, az új alaprendszer által megkívánt változtatásokat fésüli össze az aktuális beállításokkal. Nem árt a program futtatása előtt a */**etc**/* könyvtár archiválása, bár még nem találkoztam olyan programhibával, amely bármi kárt tett volna (de mint tudjuk: a ***FreeBSD*** ördög nem alszik :). Az első `mergemaster` elmenti az aktuális állapotot, amelyet az `installworld` módosítani fog. A második `mergemaster` feladata, hogy felderítse és elvégezze a beállítások módosítását, persze a rendszergazda jóváhagyásával. Ezen utóbbi tevékenység nem vesz el pár percnél több időt, így egy éles rendszer kiesése mindössze ennyi idő lesz. Siker esetén – egy megismételt újraindítás után – már az új, friss és ropogós FreeBSD alaprendszert vehetjük használatba.

Programok telepítése

Ha többet szeretnénk kihozni a ***FreeBSD*** rendszerből (és miért ne szeretnénk), mint az alaprendszer használata, akkor programokat kell telepítenünk. Erre sok lehetőségünk van, amelyek között szabadon választhatunk belátásunk szerint.

A sysinstall használata

Legegyszerűbb módja a programtelepítésnek a `sysinstall` nevű mindenes használata. Mint már említettem, a *„Configure”* menüpont *„Packages”* almenüjében, a megfelelő telepítési médium kiválasztása után, egyszerűen kijelöljük a telepítendő programcsomagokat. Ezek előre le vannak fordítva (a használt architektúra „leggyengébb” láncszemére), ezért azonnal telepíthetők. A módszer előnye az azonnali használatbavétel, hátránya az optimalizált fordítás hiánya, amely 5-50% teljesítményveszteséget is okozhat. Kezdőknek bátran ajánlom ezt a módszert.

A pkg_add program

Ha nem szívleljük a menürendszer által vezérelt programok használatát, akkor a `pkg_add` programot nekünk találták ki. Ezzel ugyanis azt végezhetjük el, amit a `sysinstall` is megtenne, de lassú hálózati kapcsolaton át sokkal hatékonyabb és gyorsabb lehet a munkánk. A program használata egyszerű, bár ismernünk kell a telepítendő csomag pontos nevét:

```
$ pkg_add -r mozilla
```

„Letöltjük, kicsomagoljuk, lefordítjuk, feltelepítjük” módozat

Ez a módszer a legősibb programtelepítési mód *UNIX* rendszerek esetén. A programok nagy része *C/C++* nyelven megírt forráskód formájában áll rendelkezésre, amelyeket használat előtt le kell fordítani. Ez több okból alakult így, részben a *UNIX* rendszerek változatos hardverei miatt, részben a nyílt forrás okán. Egy programot több tíz architektúrára előfordítani időigényes és felesleges munkának bizonyult, a programok forrása pedig többnyire rendelkezésre állt. Így célszerűen letöltötték-megszerezték minden szükséges program forrását, ezeket lefordítottak a használat helyén, majd „feltelepítették”. Sajnos ez a módszer egyre összetettebbé vált a programok méretének növekedésével, illetve egyre nehézkesebbé – programozásban kellően nem gyakorlott – rendszergazdák számára is. Ez utóbbi esetben nem a letöltés vagy a kicsomagolás okozott gondot, hanem a forráskód lefordítása. Ehhez ugyanis magas szinten érteni kellett a programok fordításához, illetve a *C/C++* nyelvhez, ha egy nem telepített programkönyvtárra vagy forrásfejlécre hivatkozott az aktuális program.
A probléma megoldására olyan programot fejlesztettek ki, amely felderíti és leellenőrzi a telepítendő program igényeit, s ezeket hiányát közérthető módon jelzi, ez a `configure`.
Jó esetben egy ilyen telepítés csak pár utasításunkba kerül:

```
$ cd /usr/local/src/program/
$ ./configure
$ make
$ make test
$ make install
$ make clean
```

Sajnos sok olyan programmal találkozni újabban, amelyek nagyon erőteljesen hozzá vannak láncolva a *Linux* konvenciókhoz. Ezek sok szenvedés árán fordíthatók le *FreeBSD* alá, mivel olyan dolgokat keresnek olyan helyen, amelyek *FreeBSD* esetén egyáltalán nincsenek, vagy azon a helyen nem léteznek. Remélhetőleg idővel a fejlesztőknek sikerül közös nevezőre jutniuk.

A ports „adatbázis” használata

A *ports adatbázis* lényege, hogy a programcsomagok karbantartói a telepítendő programokról kiegészítő információkat gyűjtöttek egybe, amelyek jelentős segítséget adhatnak a fordítás során. Az előkészületek azonosak az alaprendszer karbantartásához: az adatbázist letöltjük a CVS kiszolgálóról, ehhez el kell készítenünk egy *sup* állományt, amely a *ports* frissítéséért felelős (ennek neve lehet például *ports-sup*):

```
*default host=cvsup.hu.freebsd.org
*default base=/usr
*default prefix=/usr
*default release=cvs tag=.
*default delete use-rel-suffix
*default compress

ports-all
```

Általában az éles szeműeknek is csak egyetlen különbség tűnik fel az *src-sup* állományhoz képes: *src-all* helyett *ports-all* van az állomány végén. Azonban érdemes megnézni a *tag* opciót is, ahol most egy darab pontot láthatunk csak, ugyanis a *ports* független a FreeBSD alaprendszer verziójától! A többinek már mennie kell, mint a karikacsapásnak: ki kell adnunk a `cvsup ports-sup` parancsot, amely a */usr/ports/* könyvtár alá szinkronizálja a legfrissebb adatbázist. Jogos kérdéseket vélek hallani: *„De mire jó ez a 300 megabájt letöltött adat?!”* A kérdés jó!
A *ports adatbázis* (jelenleg) közel 11500 program adatait tartalmazza, amelyeket felkészítettek a FreeBSD alatti komplikációktól mentes fordításra. Több tíz kategória közül választhatunk, amelyek sok száz – a kategóriába tartozó – programot tartalmazhatnak. Például a *BitchX IRC* kliensprogram feltelepítéséhez bele kell lépnünk a */usr/ports/irc/bitchx* könyvtárba, ahol körülnézve alig látunk néhány állományt. Ez így van rendjén, ugyanis csak az összefoglaló információkat tudhatjuk meg a kiválasztott programról (*pkg-descr*), a feltelepítendő állományok nevét és helyét (*pkg-plist*), valamint egy *Makefile* is ott árválkodik. A program forrásának nyoma sincs, és ez így van rendjén. Gondoljunk csak bele, hogy 11500 program forrása mekkora helyet igényelne! A `make` parancsot kiadva tudjuk a program telepítését megkezdeni, s a gépezet működni kezd: letölti a megadott helyek egyikről a program forrását:

```
>> ircii-pana-1.1-final.tar.gz doesn't seem to
➥exist in /usr/ports/distfiles/.
>> Attempting to fetch from ftp://ftp.bitchx.org/
➥pub/BitchX/source/
```

Amint láthatjuk, először a */usr/ports/distfiles/* könyvtárban keresi a letöltendő állományt, amely könyvtár egyfajta gyorsítótár funkciót valósít meg. Ha elfeledkeznénk erről a könyvtárról, könnyen előfordulhat, hogy valami rejtélyes oknál fogva elfogy a hely a */usr/* fájlrendszeren. Ilyen esetben elsősorban erre a könyvtárra gyanakodjunk (második gyanúsítottunk a */usr/obj/* lehet, amely az alaprendszer lefordított állományainak ad helyet).
A letöltött forrás kicsomagolásra kerül, s innentől azonos módon járunk el, mint az előző módszer: lefordítjuk és fel-

telepítjük a kívánt programot. Látható, hogy a felmerült problémák nagy részét sikerült megoldani, mivel nem kell keresgélni és letölteni a programok forrását, illetve teljesen biztosan működni fognak ***FreeBSD*** alatt. Továbbá már nem kell kézzel vadászni a szükséges komponenseket, mivel a ***ports*** adatbázis már tartalmaz információkat a függőségekről, így rekurzív úton automatikusan feltelepítésre kerül minden szükséges csomag.

A portinstall használata

Szinte hallom a morgolódást, hogy *„Ennyi? Mi ebben a rendszergazdabarát?!”* Igen, én is ezt kérdezgettem magamtól, amíg rá nem találtam az „egyetlen igaz útra”: a `portinstall` használatára. Aki egyszer ráérez az ízére, soha nem feledi. A `portinstall` a */usr/ports/sysutils/portupgrade/* könyvtárból telepíthető fel (mivel azonos a `portupgrade` programmal), természetesen az előző részben megismert módon.
A programok telepítésének (eddigi) legkényelmesebb módja a `portinstall` program használata, amely jelentősen megkönnyíti ezt a tevékenységet. Első használatkor a program elkészíti a saját adatbázisát a feltelepített és a feltelepíthető csomagokról, ez sok ideig is eltarthat. Alapvetően könnyű a használat, egyszerűen meg kell neveznünk a kívánt program nevét, `portinstall apache`, ezzel elindul a megnevezett program telepítése. Feltéve, ha van ilyen, illetve csak egyetlen ilyen nevű programot talál a portinstall. Telepíthető program hiányában egy hibaüzenetet kapunk, amely arról tájékoztat, hogy nincs ilyen nevű program a ports adatbázisban. Több találat esetén sorban megkérdezi a feltelepítendő program pontos nevét, és verziószámát:

```
-->  Found 2 ports matching 'apache':
        www/apache13
        www/apache2
Install 'www/apache13'? [yes] n
Install 'www/apache2'? [yes] y
```

A telepítés innentől teljesen azonosan történik a ***ports adatbázis*** használatával, eltekintve attól, hogy nem kell más parancsot kiadnunk, a `portinstall` mindent megold és elvégez. Érdemes a telepítéskor mindig rákérdeztetni a telepítendő programokra, ezzel sok felesleges munkától szabadulunk meg (nem kívánt program eltávolítása); ezügyben a `-i` opciót kell használnunk. Fontos lehet a -P opció ismerete is, amely kombinálja az előfordított csomag telepítését a `portinstall` előnyeivel: a program megpróbálja letölteni a kívánt csomag előrefordított állományát, s csak akkor használja a ***ports adatbázist***, ha nincs ilyen előfordított állomány. Ezzel időt tudunk megspórolni, ha gyorsan kell egy programot telepíteni.

A portupgrade használata

A telepítések mellett nagyon fontos a telepített programok naprakészen tartása. Ehhez alapvetően a ***ports adatbázis*** frissítése szükséges, amelyet a `cvsup ports-sup` parancs kiadásával tehetünk meg. Ekkor a frissítendő programot el kell távolítani, majd a helyére feltenni az újabb verziót. Feltéve, ha van újabb, viszont ennek eldöntése kicsit „favágó” munka, mert egyenként végig kell lépkednünk a telepített programokon, és megnézni, hogy van-e újabb.

Ezt az idegőrlő munkát teszi átláthatóvá és egyszerűvé a `portupgrade` program, ugyanis végignyálazza helyettünk a feltelepített programokat; majd utánanéz, hogy van-e újabb verzió. Gyakorlatilag csak a `portupgrade -a` parancsot kell kiadni, óvatosabbak a `-i` opciót is használhatják `portupgrade -a -i`.
Általános esetekben némi molyolás után listát kapunk a feltelepített és a frissíthető programokról. Előfordulhat azonban, hogy a programcsomagok függőségei a frissítés során (aljas módon :) megváltozhatnak. Ekkor egy (első látásra riasztó) üzenetet láthatunk, mint például

```
Stale dependency: intlfonts-1.2.1 -> XFree86-
➥3.3.6_11 – manually run 'pkgdb -F' to fix, or
➥specify -O to force.
```

Nem kell megijedni, csak futtatnunk kell a `pkgdb -F` programot, amely a segítségünket igénybe véve kijavítja a hibát.

```
Stale dependency: intlfonts-1.2.1 -> XFree86-
➥3.3.6_11 (x11/XFree86):
New dependency? (? to help):
```

A megoldás egyszerű, az ***Xfree86*** csomag helyett már ***Xorg*** néven kell a grafikus felületre hivatkozni:

```
New dependency? (? to help): xorg-6.7.0_1
Fixed. (-> xorg-6.7.0_1)
```

Ugye, nem is volt olyan bonyolult? Figyelni kell arra, hogy a függőség megadásánál pontosan nevezzük meg a programot, annak minden verziószámával együtt. Sajnos ez a „hiba” ritka, és ennél fogva zavarba ejtheti a gyanútlan ***FreeBSD*** rendszergazdát. Ennél is ritkább, ha szinte egy komplett nyomozati apparátusra van szükségünk, ha fel szeretnénk deríteni a hiba okát, mivel igencsak rejtélyes lehet a függőségek szövedéke. Érdemes gyakran frissíteni a ***ports adatbázist***, majd azonnal futtatni a `pkgdb -F` parancsot, mivel hosszabb idő (hónapok) kihagyása után esélytelen lehet a függőségek kibogozása.
A fenti problémák nélkül egy listát kapunk a telepített programokról, amelyeknek nagy részét nem kell frissíteni, mert nincs belőlük újabb verzió. Természetesen felkérhetjük a `-f` opcióval, hogy minden telepített programot fordítson újra: előrefordított programok esetén megtehetjük, ha rengeteg felesleges időnk van.

```
** No need to upgrade 'xpdf-3.00_3' (>= xpdf-
➥3.00_3). (specify -f to force)
** No need to upgrade 'urwfonts-1.0' (>= urwfonts-
➥1.0). (specify -f to force)
```

Néhány csomag esetén viszont megkapjuk a kérdést:

```
-->  Upgrade of java/jdk14 started at: Wed, 03 Nov
➥2004 19:39:06 +0100
-->  Upgrading 'jdk-1.4.2p6_5' to 'jdk-1.4.2p6_6'
➥(java/jdk14)
OK? [yes]
```

Elég egy ENTERT ütni, s elindul a kiválasztott csomag frissítése, amely során a `portupgrade` elkészíti az újabb verzió telepítésre kész csomagját, kiveszi a csomaglistából a régi programot, elmenti a régi csomag minden telepített állományát, majd megpróbálja az újat telepíteni. Ha sikerül, akkor nincs már semmi gond, ettől kezdve az újabb programot használhatjuk. Probléma esetén visszakerül a helyére a régi program, így a hibák nem okoznak felesleges leállást.

Program keresése a ports adatbázisban

Gyakran előfordul az a helyzet, amikor egy bizonyos feladatra keresünk megfelelő programot. Egyik lehetséges megoldás, hogy a számunkra érdekes kategóriát nézzük át, és bízunk a szerencsénkben: hátha belebotlunk a keresett programba. Sokkal célravezetőbb megoldás, ha a */usr/ports* könyvtárba lépve utasítást adunk erre a keresési feladatra. Ez egyszerűen egy megfelelő `make` parancs kiadása lesz:

```
$ make search key=open | less
```

Ezzel az utasítással a program végignyálazza a teljes ports adatbázist, és minden olyan csomagot kiír, amely neve, rövid vagy bővebb leírása tartalmazza a megadott kulcsot, amely jelen esetben az „open” szó. A „less” programot azért érdemes futtatni, mert sok száz találat is lehet, amelyek hamar felfutnak a képernyőn. Előfordulhat, hogy olyan találatot is olvashatunk, amely látszólag nem tartalmazza a kulcsszót:

```
Port:   lzo-1.08_1
Path:   /usr/ports/archivers/lzo
Info:   Portable speedy, lossless data compression
        ➥library
Maint:  ports@FreeBSD.org
```

Nem kell azonban aggódni, a keresés kiválóan működik, a program bővebb leírása tartalmazza az „open” kulcsszót, egyszerűen csak ez a bővebb leírás nem jelenik meg. A kapott listát végigkövetve hamar megtalálhatjuk azt a programot, amelyet fel szeretnénk telepíteni.

A telepített programok adatbázisa

A telepített programok leírásai egy speciális adatbázisban foglalnak helyet, amelyet a */var/db/pkg/* könyvtárban találunk. Nem kell hozzá speciális program, ugyanis egy program – egy könyvtár elven találjuk meg az összes regisztrált (telepített) program jellemzőit.
Természetesen létezik külön program, amely kényelmesebbé teszi ezen adatbázis használatát, ez a `pkg_info`, amelynek annyi a dolga, hogy a paraméterében megkapott csomagról információkat szolgáltasson:

```
$ pkg_info hu-openoffice-1.1.3
Information for hu-openoffice-1.1.3:

Comment: Integrated wordprocessor/
➥dbase/spreadheet/drawing/chart/
➥browser
```

Programok törlése

Eljöhet az idő, amikor egy telepített program feleslegessé válik, bár én ritkán találkozom olyan esettel, amikor programok törlése valóban szükséges lépésnek bizonyul (gondoljunk csak a merevlemezek szinte határtalan kapacitására). Az egyik törlési megoldás igazából nem törlés, hanem a program frissítése során alkalmazott technikai megoldás, amely azonban önmagában is használható: `pkg_deinstall`. Ekkor a telepített állományokat nem töröljük a helyükről, csak a telepített csomagok adatbázisából kerülnek ki, így azonos program újabb verziójának telepítésénél nem lesz akadálya. Nem szép megoldás, ezért ne nagyon használjuk függőségi problémák megoldására.
Véglegesen a `pkg_delete` parancs segítségével tudjuk törölni a kívánt programot, ekkor a telepített programok adatbázisa alapján a törlődnek a nem módosított állományok. Ez utóbbi azért fontos, mert különben a programok beállításai is törlődnének:

```
$ pkg_delete lsof-4.73.1
```

A programnak hibátlan lefutás esetén nem lesz kimenete, ha szeretnénk, hogy részletesen számoljon be a tevékenységéről, akkor használnunk kell a `-v` paramétert is. Ha nem tudjuk pontosan a törlendő program nevét ***verziószámával*** együtt, akkor használhatjuk a megszokott helyettesítő karaktereket is:

```
$ pkg_delete lsof*
```

Mind a kettő program azonos paramétereket fogad el, amelyek közül elsődlegesen a `-f` lehet érdekes számunkra, amely megadásakor a törlés akkor is végrehajtódik, ha függőségi problémák merülnének fel. Érdemes a `-r` és a `-R` paraméterre is gondolnunk, amelyek hatására a törlés rekurzívan hajtódik végre. A `-R` esetén mindazon csomagok törlésre kerülnek, amelyekre a megadott csomagnak szüksége van, ezzel biztosak lehetünk benne, hogy nem maradnak olyan telepített csomagok, amelyekre igazából már nincs más csomag felől hivatkozás (így igazából feleslegesek).
A `-r` ennek ellentéte: az összes olyan csomag törlődni fog, amelyeknek a megadott csomagra szüksége van; ezzel óvatosan érdemes bánni, mert egy alapvetően szükséges csomagot megadva akár az összes telepített csomag törlődni fog.

Auth Gábor (auth.gabor@enaplo.hu)
Egy pécsi középiskolában informatikát és programozást oktat. Tíz éve botlott először a UNIX rendszerekbe, 7 év Linux használat után kapta el a FreeBSD lázat, amiből máig nem tudott kigyógyulni.

KAPCSOLÓDÓ CÍMEK

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

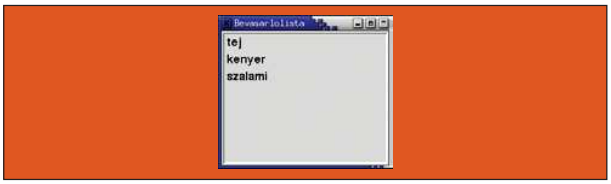
Listaablakok röptében a Tk::HList segítségével

A bevásárlólistától a /etc/passwd-ig grafikus külsőt kaphat bármi, ha Perl alatt kipróbálod a HList elemet.

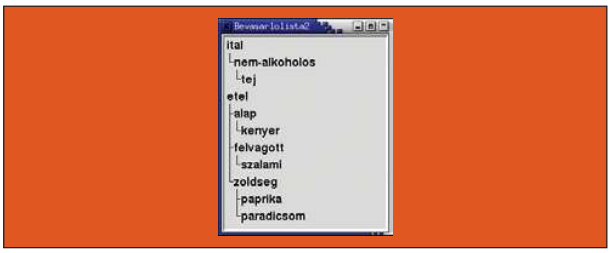
Ezúttal rendhagyó módon szeretnék eltekinteni az unalmas történelmi bevezetőtől. Miután teljesen nyilvánvaló, hogy a kedves Olvasó előbb a képernyőfotókat nézi meg, majd azonnal a kódra kíváncsi, térjünk rögtön a lényegre. Elöljáróban csak annyi feltételezéssel élek, hogy olvasóimnak nem teljesen új az a módszer, amivel *Perl/Tk*-ban egy ablakot létre lehet hozni . Ha ez nem jelent gondot, akkor kattogjon az a klaviatúra, mert máris jön az első lista.

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

use strict;
use Tk;
use Tk::HList;

my $foablak = new MainWindow (-title =>
              ➥"Bevasarlolista");

my $lista = $foablak -> HList (-width => 25);
$lista -> pack;

my @vasarlas = ("tej", "kenyer", "szalami");
my $dolog;

foreach $dolog (@vasarlas) {
    $lista -> add ($dolog, -text => $dolog);
}

MainLoop;
```

A szkript három modult használ. A már jól ismert `strict` szigorú ellenőrzési módot ír elő a parancsértelmezőnek. A *Tk* a grafikus eszközkészletet (*ToolKit*) teszi elérhetővé egy jól átgondolt objektum-orientált felületen keresztül. Végül a `Tk::HList` a *Tix* (*Tk Interface Extension*) `HList` nevű elemét biztosítja. Ez a kiegészítés minden *Tk* csomag része, így ennek a használatával jó eséllyel nem veszítünk a felület-függetlenségből.
Első lépésben példányosítjuk a `MainWindow` osztályt, és a konstruktornak egy – jelen esetben egyetlen párból álló – névtelen asszociatív tömböt adunk át. Ezzel úgy hozzuk létre az ablakot, hogy azonnal be is állítjuk a címét. A kapott

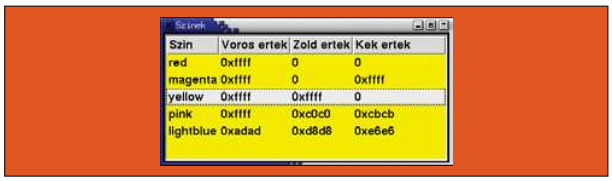


1. ábra A bevásárlólista

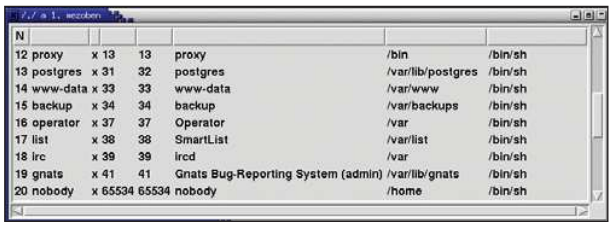


2. ábra Bevásárlás több szinten...

`$foablak` objektum `HList` elemfüggvényével hívunk életre egy `HList` ablakelemet, melynek szintén adunk kezdőértéket, ezáltal meghatározzuk a szélességét. Ezután az új, `$lista` objektumra meghívjuk a `pack` elrendezés-kezelőt. Semmi gond nem származik abból, hogy anélkül hívtuk meg a `$lista pack` elemfüggvényét, hogy az akár egyetlen listaelemet is tartalmazna. Az ablak, és minden, ami benne van, csak akkor jelenik meg, amikor a `MainLoop`-ot meghívjuk. A lista feltöltése a `@ennivalok` tömbből fog történni. Egy `foreach` szerkezetben a $dolog változó sorra felveszi ezen tömb elemeit. A ciklus minden lefutásában hozzáad egy új tételt a bevásárlólistához.
Ezen a ponton álljunk meg egy pillanatra. A `HList` egy igen sokoldalú ablakelem, és sokkal többre képes egy egyszerű lista megjelenítésénél. Kialakíthatunk egy hierarchiát az adatelemeink között, és akár egy faszerkezettel is elkápráztathatjuk felhasználóinkat. Az adattagokat a hierarchiában egy egyedi név azonosítja, amely ponttal elválasztva tartalmazza az elem elérési útját. Például az `etel.zoldsegek.paprika` azt jelenti, hogy a paprika a harmadik szinten helyezkedik el, és a közvetlen szülője a `zoldsegek`, ami az `etel` alatt található.
Természetesen nem kötelező kihasználnunk ezt a lehetőséget, tehetjük az összes elemünket egy szintre. Ebben a pél-



3. ábra Színes oszlopok



4. ábra Szöveges adatbázis megjelenítése

dában pontosan ez történik. Az add elemfüggvény első paramétereként megadjuk az elérési utat, továbbá kulcs-érték párral az elemhez tartozó megjelenítendő szöveget. Ez a két dolog jelen helyzetben ugyanaz, de ez nem szükségszerű. Egy többszintű ábránál szinte biztos, hogy nem is így van. Lássunk erre is egy példát!

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

use strict;
use Tk;
use Tk::HList;

my $foablak = new MainWindow (-title =>
              ➥"Bevasarlolista2");

my $lista = $foablak -> HList (
                        -width => 25,
                        -height => 15
                        ) -> pack;

my @vasarlas = (
    ["ital", "nem-alkoholos", "tej"],
    ["etel", "alap", "kenyer"],
    ["etel", "felvagott", "szalami"],
    ["etel", "zoldseg", "paprika"],
    ["etel", "zoldseg", "paradicsom"]
);
my $dolog;

foreach $dolog (@vasarlas) {
  for (my $elem = "", my $i = 0; $i < 3; $i ++) {

      $elem .= "." unless ("" eq $elem);
      $elem .= $$dolog [$i];

        unless ($lista -> infoExists ($elem)) {
          print "Elem: " . $elem . " (" . $$dolog
          ➥[$i] . ") ";
          $lista -> add ($elem, -text => $$dolog
          ➥[$i]);
          print "hozzaadva.\n";
        }

    }
}

MainLoop;
```

A $foablak HList elemfüggvényének meghívásakor egyrészt megadtunk egy további magasság paramétert (height), másrészt a pack-et közvetlenül a létrejövő objektumra hívtuk meg. Mivel a nyíl operátor balról jobbra értékelődik ki, ez az állítás teljesen egyenértékű az eredetivel, és talán egy kicsivel rövidebb. A @vasarlas tömbünk itt már névtelen, három elemű tömbök referenciáit tartalmazza. Ezeket a referenciákat veszi fel sorra a $dolog változó a foreach ciklusban.
Minden egyes lefutásban három lépést teszünk. A második tömbreferencia feldolgozásakor például előbb az etel, majd az etel.alap, végül az etel.alap.kenyer elemet vizsgáljuk. Erre szolgál a belső for ciklus. Minden lépésben az infoExists elemfüggvény segítségével megnézzük, hogy az elem létezik-e, és ha nem, létrehozzuk. Itt látható, hogy az elérési út különbözik a megjelenített névtől. A végén adódó fa egyes ágai nem bezárhatóak. Ha ilyesmire lenne szükséged, nézz utána a Tree elemnek, amely a HList-ből származik.
A következő szkriptben egy több oszlopból listát láthatunk, sőt, még azt is megtanulhatod, hogyan lehet eseménykezelést rendelni az ablakelemhez.

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

use strict;
use Tk;
use Tk::HList;

my $foablak = new MainWindow (-title => "Szinek");

my $lista = $foablak -> HList (
                        -columns => 4,
                        -header => 1,
                        -width => 25,
                        -height => 15,
                        -command => \&hatter_beallit
);

$lista -> headerCreate (0, -text => "Szin");
$lista -> headerCreate (1, -text => "Voros ertek");
$lista -> headerCreate (2, -text => "Zold ertek");
$lista -> headerCreate (3, -text => "Kek ertek");

$lista -> pack (
          -expand => 1,
          -fill => "both"
);

my @szinek = ("red", "magenta", "yellow", "pink",
➥"lightblue");
my $szin;
```

```
foreach $szin (@szinek) {

    my ($voros, $zold, $kek) = $foablak -> rgb
                               ➥($szin);

    $lista -> add ($szin);
    $lista -> itemCreate ($szin, 0, -text =>
              ➥$szin);
    $lista -> itemCreate ($szin, 1, -text =>
              ➥sprintf “%#x”, $voros);
    $lista -> itemCreate ($szin, 2, -text =>
              ➥sprintf “%#x”, $zold);
    $lista -> itemCreate ($szin, 3, -text =>
              ➥sprintf “%#x”, $kek);

}

MainLoop;

sub hatter_beallit {
    my ($szin) = @_;
    $lista -> configure (-background => $szin);
}
```

Ez a szkript már valamivel összetettebb az előzőekben bemutatottaknál. Színek egy rögzített halmazát jeleníti meg táblázatos formában. Minden szín vörös, zöld és kék összetevőjét közvetlenül a szín neve mellett 16-os számrendszerben mutatja. Ha a felhasználó duplán kattint valamelyik elemen, a teljes lista az adott háttérszínt veszi fel. A színek közismert angol nevét használjuk a feladat egyszerűsítése végett.
A főablak objektum `HList` elemfüggvényének paraméterezése már közel sem olyan kurta, mint a legelső példában. A `-columns` kulccsal lehetőségünk van az oszlopok számának beállítására. Ez alapértelmezésben 1, ezért hagyhattuk el eddig. Új opció továbbá a `-header`, mely azt határozza meg, hogy legyen-e fejléce az egyes oszlopoknak. Alapértelmezésben 0, vagyis nincs fejléc, ezt bíráljuk felül az 1-es értékkel. Végül a `-command` egy függvényt tartalmaz, melyet a `MainLoop` minden egyes alkalommal meghív, ha bármely elemen kettős kattintás történik.
A fejléc tartalmának létrehozására a lista objektum `headerCreate` elemfüggvénye szolgál. Paraméterezése önmagáért beszél. Először az oszlop sorszámát kell megadnunk, ahol az első oszlop a 0. indexű. Utána egy már jól ismert `-text` kapcsolóval beállíthatjuk a megjelenítendő szöveget. Ezúttal a `pack`-et sem üres paraméterlistával hívtuk meg. A megadott két beállítás hatására az ablakelem önműködően kitölti a főablakot, azaz a felhasználó bárhogy módosítja az ablak méretét, a listánk széltében és hosszában is követi a változtatásokat.
A lista feltöltése ismét egy `foreach` szerkezettel történik, ám most fontos, hogy a `@szinek` tömb szabványos angol színneveket tartalmaz. A ciklus hasának első utasításában ugyanis a tömb sorra következő elemével meghívjuk a főablak `rgb` függvényét. Ez utóbbi a paraméterként kapott szín három összetevőjét tartalmazó tömbbel tér vissza. Így a `$voros`, `$zold` és `$kek` változók a megfelelő értéket kapják.
Ezután az add elemfüggvénnyel csak létrehozzuk az elemet, szöveget még nem rendelünk hozzá. Ezt a feladatot ugyanis az `itemCreate` függvényre hárítjuk, amely egy meglévő listaelem celláinak kitöltésére használható. Első paramétere a listaelem azonosítója, második az oszlop sorszáma, utána pedig bármi, ami az add-nél is használható. C-hez szokott szemeknek nem meglepő, hogy a színösszetevők 16-os számrendszerbeli értékének előállításához a `sprintf` függvényt használtuk.
Végül térjünk ki egy kicsit az eseménykezelésre. Mint említettem, még a `HList` ablakelem létrehozásakor meghatároztunk egy eseménykezelőt a `-command` kapcsolóval. Ezt a `MainLoop` hívja meg amennyiben a felhasználó duplán kattintott. Azon elem azonosítóját, melyen a kattintás történt, paraméterként megkapja a függvényünk. Vagyis a `hatter_beallit` függvény első utasításában a `@_` nevű, paraméterlistát tartalmazó tömb első elemének kinyerésével annak a színnek a nevéhez jutunk, melyre a felhasználó kettőt kattintott. Ezután egy `configure` függvénnyel érvényre juttatjuk a kívánt hatást.
Utolsó példánkban egy általános, szöveges alapú adatbázisok megjelenítésére használható szkriptet szeretnék bemutatni.

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

my $hatarolo = “:”;

die “Hasznalat: “ . $0 . “ {kulcs} {regkif}\n”
unless @ARGV == 2;

use strict;
use Tk;
use Tk::HList;

my @talalatok = (); my $mezoszam = 0;

while (<STDIN>) {
    chomp; my @mezok = split $hatarolo;
    if ($mezok[$ARGV[0] - 1] =~ /$ARGV[1]/) {
      push @talalatok, $. . $hatarolo . join
      ➥$hatarolo, @mezok;
      $mezoszam = @mezok if ($mezoszam < @mezok);
    }
}
$mezoszam ++;

my $foablak = new MainWindow (
    -title => “/” . $ARGV[1] . “/ a “ . $ARGV[0] .
    ➥“. mezoben”, -borderwidth => 2

);

my $lista = $foablak -> Scrolled (
    “HList”,
    -scrollbars => “se”,
    -columns => $mezoszam,
    -header => 1,
    -width => 25,
    -height => 15,
    -command => \&duplakatt
);
```

```
$lista -> headerCreate (0, -text => "N");
$lista -> pack (
          -expand => 1,
          -fill => "both"
);

foreach (@talalatok) {
    my @mezok = split $hatarolo;
    $lista -> add ($mezok[0]);
    for (my $i = 0; $i < $mezoszam; $i ++) {
      $lista -> itemCreate ($mezok[0], $i, -text =>
      ➥$mezok[$i]);
    }
}

MainLoop;

sub duplakatt {
    my ($azonosito) = @_;
    for (my $i = 1; ; $i ++) {
      print $lista -> itemCget ($azonosito, $i,
      ➥-text);
      last if ($i == $mezoszam - 1);
      print $hatarolo;
    }
    print $/;
    exit;
}
```

A szkript a határoló karaktert egy változóban tárolja, így az egyetlen sor módosításával változtatható. A szabványos bemenetről olvassa az adatokat, és önállóan meghatározza az oszlopok számát. Az adatsorok közül kizárólag azokat jeleníti meg, melyeknek adott sorszámú mezője illeszkedik a szintén adott szabályos kifejezésre. Az oszlopok sorszámozása 1-től indul. A grafikus megjelenítés után a felhasználó duplán kattinthat bármely elemen. Amennyiben így kiválasztott egyet, az megjelenik a szabványos kimeneten, és véget ér a program.

A `use`-al kezdődő soroktól kezdve nézzük át a program működését. Két fő ciklus köré szervezzük a működést. Előbb beolvassuk az összes adatot egy tömbbe, majd egy következő ciklusban megjelenítjük valamennyi elemet. Az első lépésben két változót töltünk fel: a `@talalatok` tömböt és a `$mezoszam` változót. A `@talalatok` a keresés eredményét tartalmazza, minden találatot kiegészítve egy sorszámmal. A `$mezoszam` a találatok legtöbb mezőt tartalmazó sora mezőinek számát tartalmazza.

A `while` ciklus addig olvas a szabványos bemenetről, amíg talál ott adatot. Minden lefutásban egy sort dolgozunk fel. Először levágjuk a sorvége jelet, majd a határoló karakterek mentén felszeleteljük a sort, és a mezőket kigyűjtjük egy `@mezok` tömbbe. Ezután ellenőrizzük, hogy a rekord kielégíti-e az általunk támasztott feltételeket. A szkript első paramétereként kapott mezőszámot eggyel csökkentjük, és a `@mezok` ezen indexű elemét vizsgáljuk.

Ezáltal ha a paraméter 1 volt, az itt 0-át jelent, ha 2, akkor 1-et, stb. Azaz a felhasználó számára a mezők sorszámozása 1-től indul, és pont ez volt a cél. Tehát a `@mezok` megfelelő elemének kiválasztása után mintaillesztést végzünk rá a második paraméterrel. Ha ez sikeres volt, veremként használva a `@talalatok` tömböt, betesszük a visszaépített sort, az elején kiegészítve a találat helyével. Erre azért van szükség, mert ez biztosan egyedi azonosítója az adott rekordnak.

A `$mezoszam`-ot minden sikeres mintaillesztés után hozzáigazítjuk a mezők számához, amennyiben az szükséges.

A ciklus végén pedig megnöveljük az értéket eggyel, hiszen minden rekordot kiegészítettünk egy, az eredeti állományban elfoglalt sor számát tartalmazó mezővel, így a legszélesebb rekord is biztosan eggyel szélesebb lett. Ezzel beolvastuk az összes adatot, jöhet a megjelenítés.

Először is létrehozzuk a főablakot. A címben jelezzük a szkript paramétereit, vagyis a keresés szempontjából kulcs mező számát és a reguláris kifejezést. Továbbá a `-borderwidth` kapcsolóval meghatározzuk az ablak széle és a hozzá legközelebb eső ablakelem közti távolságot képpontokban. Vagyis jelen esetben az ablakban használt egyetlen elem, a lista és az ablak széle között lesz egy 2 képpont vastag üres keret.

Mivel görgetősávokat is szeretnénk létrehozni, ezúttal nem a `HList` elemfüggvényt hívjuk meg. A `Scrolled` segítségével az áhított görgetősávokkal kiegészített elem jön létre, mindössze azt kell meghatároznunk, hogy hol helyezkedjenek el a sávok. Az elem neve után a `-scrollbars` kapcsolóval égtájak angol nevének kezdőbetűivel adhatjuk meg a pozíciót. Jelen esetben a dél (south) és a kelet (east) beállítással élünk. A további paraméterekkel már korábban megismerkedtünk. Csak az első oszlopnak adunk nevet, ez jelzi az adatok forrásában elfoglalt sor számát.

Ezután egy `foreach` szerkezettel végiglépkedünk a `@talalatok` tömbön. A kevesebb változó használata érdekében az alapértelmezett változót, a `$_`-t használjuk futó változóként, ahogy ezt a `while` ciklusnál is tettük. Minden lefutásban újból szétbontjuk a rekordot, hozzáadjuk a listaelemhez, és kitöltjük a cellákat. Ezáltal a listánk pontosan a keresésnek megfelelő elemekből fog állni.

Végül a `duplakatt` függvény meghatározása következik. Ezen eseménykezelő a szkript élete során jelen esetben csak egyszer fut le, hiszen egy kettős kattintás után a rekordot kiírja a szabványos kimenetre, és kilép. A kiíratásnál már nem jelenítjük meg a korábban felhasznált sorszámot, ezért indul 1-től a `for` ciklus. Az `itemCget` segítségével kinyerjük a megadott cella tartalmát, és az utolsó lefutást leszámítva egy elválasztókarakterrel együtt írjuk ki. Legvégül egy sorvége karakter nyomtatását követően kilépünk.

Remélem, hogy a példák kellően érzékletesek voltak ahhoz, hogy a `HList` nyújtotta lehetőségeket megfelelő részletességgel bemutassam. Kisebb jártasság megszerzése után valóban percek alatt lehet grafikus külsőt készíteni egyszerűbb szkripteknek, s mivel a ***Perlt*** alapvetően szöveges feladatok megoldására készítették, a `HList` igen sok helyen alkalmazható. Sok sikert a további kísérletezéshez, akinek pedig kérdése van, írjon bátran!

Fülöp Balázs (admin@guardware.com)
21 éves, imádja a Túró Rudit, a Debian Linuxot és a teheneket. Kedvenc írója Slawomir Mrožek. Leginkább a számítógépes hálózatok biztonsága érdekli. A BME VIK műszaki informatikus szak hallgatója.

PHP5 – Új generáció (2. rész)

...avagy hogyan használjuk okosan az osztályokat és objektumokat PHP 5-ben.

Cikksorozatom előző részében képet kaphattunk arról, hogy valójában mik is azok az objektumok, milyen tulajdonságaik, *PHP* vonatkozású különlegességeik vannak, illetve néhány példaprogramon keresztül megismerkedhettünk a konkrét használatukkal is. Ebben a részben központi szerepet kap az objektumközpontúság savát-borsát adó öröklődés, az ezzel kapcsolatos elvont (***abstract***) osztályok és felületek (***interface***) létrehozása, alkalmazása, valamint egy-két különleges tagfüggvény használata.

Vágjunk bele – mi is az az öröklődés

Az objektumközpontú programozás egyik ismérve a nagyfokú újrahasznosíthatóság. Ezt egyrészt annak köszönheti, hogy ezek a jól beburkolt, jól felépített objektumok komponensekként viselkednek, remekül lehet velük LEGO-zni. Másrészt ezeket az objektumokat egymással rokoni kapcsolatba állíthatjuk. A gyakorlatban ezt hívják öröklődésnek. Ha egy objektum egy másik (szülő)objektumtól örököl (gyermekobjektummá válik), akkor megkapja annak minden tulajdonságát és tagfüggvényét – a láthatóság által megfogalmazott feltételek mellett természetesen.
Az örökölt tagfüggvények teljes értékűen használhatók, felülírhatók, átalakíthatók, s új tagfüggvényeket adhatunk az osztálynak, egyszóval programozóként igen nagy szabadságot élvezünk, s mindemellett az ős összes képességét kihasználhatjuk. Lássunk erre egy példát, vegyük az előző epizódban bemutatott négyzet osztályt, kissé átalakítva

```
class Negyzet {
    protected $oldal = 0;

    public function oldalHossztBeallit($ertek) {
        $this->oldal=$ertek;
    }

    public function terulete() {
        echo $this->oldal*$this->oldal;
    }
}
```

Közben a programunkban szeretnénk speciális négyzetekkel, rombuszokkal foglalkozni. Tudjuk, hogy a négyzetek és a rombuszok hasonlóak abban, hogy minden oldaluk egyenlő, de a rombusz esetében fontos az oldalak által bezárt szög, s a területe is másként számítandó. Használjuk ki a hasonlóságokat, és csak a különbségeket valósítsuk meg.

```
class Rombusz extends Negyzet {

    protected $szog = 0;

    public function szogetBeallit($szog) {
        $this->szog=$szog;
    }

    public function terulete() {
        echo $this->oldal*sin($this->szog)*
            ➥$this->oldal;
    }
}
```

Nos, ebben a példában felülírtuk a területszámító algoritmust, és az új tagfüggvény hozzáadásával elintéztük az oldalak által bezárt szög kezelését. Ha jól megnézzük, megspóroltuk az oldalak kezeléséért felelős függvények megírását, az osztályunk mégis teljes értékű. Fontos tény továbbá, hogy megmaradt az eredeti négyzet osztályunk, s nincs a kódunkban ismétlés.
Az így megspórolt munka természetesen nem túl sok, de ez a példa igen egyszerű. Említést érdemel még a változók előtti `protected` módosító: Most már láthatjuk, hogy mire jó: az ilyen változók elérhetők az örökös osztály belsejéből, de csak onnan.

A konstruktorok, destruktorok, és az öröklődés

PHP 5-ben ha az ősosztályunknak volt egy konstruktora, majd az abból örököltettünk másik osztály esetében nem határoztunk meg ilyen tagfüggvényt, akkor a példányosítás során az ősosztály konstruktora automatikusan meghívódik, lefut, elvégzi a szükséges lépéseket.
Ha azonban az örököltetett osztály is kapott konstruktort, akkor az ősosztály konstruktorának meghívásáról magunknak kell gondoskodnunk (ha szükséges). Ez azért van így, mert ellenkező esetben jelentősen meg volna kötve a kezünk a származtatásokat illetően. Lássunk erre is egy példát, módosítsuk az ősosztályt. Az előző cikkben is ismertetett módszer szerint adjunk hozzá egy konstruktort, amely beállítja az oldalhosszt.

```
class Negyzet {
    protected $oldal = 0;

    public function __construct($oldal) {
        $this->oldalHossztBeallit($oldal);
    }

    public function oldalHossztBeallit($ertek) {
        $this->oldal=$ertek;
    }

    public function terulete() {
        echo $this->oldal*$this->oldal;
    }
}
```

Ilyenkor ugye a példányosítás után nem kell bíbelődni az oldalhossz beállításával, elintézhetjük a dolgot egy lépésben is. Na de mi van ilyenkor az előbbiekben bemutatott örökössel? Szeretnénk, ha az is egy lépésben elvégezné ezeket a módosításokat, de mint tudjuk, az oldalhossz kezelésével nem is foglalkoztunk, az az ősosztály feladata. Nézzük!

```
class Rombusz extends Negyzet {

    protected $szog = 0;

    public function __construct($oldal,$szog) {
        parent::__construct($oldal);
        $this->szogetBeallit($szog);
    }

    public function szogetBeallit($szog) {
        $this->szog=$szog;
    }

    public function terulete() {
        echo $this->oldal*sin($this->szog)*$this->
        ➥oldal;
    }
}
```

Látható, hogy az oldalkezelés továbbra is az ősé maradt. Mi csupán annyit tettük, hogy megkértük az örökös belsejéből, hogy foglalkozzon az ő érdekeltségébe tartozó adatokkal. Erre szolgál az a bizonyos `parent` előtag a `scope (::)` operátorral: az ősosztály függvényeire hivatkozhatunk vele. Ez természetesen csak akkor érdekes, ha felülírunk egy függvényt, ugyanis ha ezt nem tesszük, a `$this->függvénynév()` módon elérhetjük, mint az osztály saját tagfüggvényét. Ha azonban az öröklés során felülírjuk, és úgy szeretnénk hivatkozni, az előbbi módszerrel egy végtelen rekurzív függvényt kapunk, ami bizonyos, hogy senkinek sem jó.
A `parent::__construct()` hívás helyett jelen esetben a `$this->oldalHossztBeállít()` tagfüggvényt is használhattuk volna, ám úgy logikailag összefolyna a két objektum. Ez most még egy sokadrangú döntés, amely bonyolultabb esetekben azonban életet menthet.

Függvénytúlterhelés (overloading)

Számos – főként erősen típusos nyelvekben – létezik ez a fogalom. Ez nem is annyira az objektumokhoz kötődik, hanem úgy általánosságban létezik, ám most az öröklődés és a felülírás kapcsán érdemes néhány szót ejteni róla a félreértések elkerülése végett.
A függvénytúlterhelés azt jelenti, hogy több ugyanolyan nevű, de más paramétereket fogadó (esetleg más visszatérési típussal rendelkező) függvényt is definiálhatunk, s a meghívás során az a függvény hajtódik végre, melynek paraméterei (száma, típusa) illeszkednek a hívó paraméterekre. Ezzel lehet megoldani, hogy egy hasonló funkciójú függvényt különböző típusú és számú esetben is alkalmazni lehessen.
A *PHP*-ben ez mindkét okból szükségtelen. Mint tudjuk a *PHP* egy gyengén típusos nyelv, egy változó (paraméter) értéke lehet egész, lebegőpontos, karaktersorozat, tömb, akármi. Így tehát mindegy, hogy az adott paraméter gyanánt milyen típust adunk át.
A másik ok a paraméterek száma volt. Ez a lehetőség azért válik feleslegessé, mivel megadhatunk függvényparaméterként alapértelmezett értékeket a meghatározásban. A *PHP* tudja, hogy ha a meghívás során nem adunk át paramétert, akkor behelyettesíti a definícióban megadott alapértelmezett értéket.
Ennek folyományaként a *PHP*-ben sehol sem engedélyezett a függvénytúlterhelés, ne is keressük.

Tagfüggvény felülírásának megakadályozása

Számos esetben előfordulhat, hogy ki szeretnénk kötni néhány metódus számára, hogy azokat az örökösök bizony ne írhassák felül, ne változtathassák meg az algoritmust. Erre olyankor lehet szükség, ha több programozó által használt ősosztályt készítünk, és szeretnénk, ha a mi szabályaink szerint kódolnának – mert az úgy egységes, ellenőrizhető, konzisztens, és így tovább.
A problémára a `final` módosító kínál megoldást, amelyet tagfüggvények definíciói előtt használhatunk.

```
class OsOsztaly {
    final public function teszt() {
        echo 'Ősosztály teszt() metódusa lefutott';
    }
}

class GyermekOsztaly {
    public function teszt() {
        echo 'felülírt teszt() metódus lefutott';
    }
}
```

Ha ilyet szeretnénk csinálni, programunk futása végzetes hibával megszakad.

Elvont osztályok

Az eddigiekben tárgyalt objektumaink, még ha rokonságba is állíthatók egymással, meglehetősen szétszórt szerkezetet alkothatnak, s ennek csakis a programozó szabhat határt. Mint tudjuk, ez messze nem elégséges feltétel. Sokszor szükség lenne arra, hogy egy jól átgondolt hierarchiát építsünk, s a fejlesztés során, mint karácsonyfáról a szaloncukrot, csak

leakasszunk egyet a megfelelő helyről. Ezen szabályozott öröklés, hierarchia felépítését segítik az elvont osztályok és a felületek, nézzük most ez előbbit.

PHP 5-ben lehetőségünk van elvonatkoztatott osztályok, elvont tagfüggvények létrehozására. Ezek általában olyan tagfüggvények, amelyeket az adott osztály nem tud megvalósítani, mondjuk mert nem ismeri a megoldás miként jét, de azt pontosan tudja, hogy ilyen szolgáltatásra szükség lesz majd valamikor, és azt is tudja, hogy a gyermekosztályok már képesek lesznek a függvény megvalósítására.

Az absztrakt osztályoknak tehát csak a definíciója ismert, nincs is lehetőségünk a konkrét algoritmus megvalósítására az adott osztályon belül.

Az az osztály, amely majd megvalósítja az absztrakt metódust, legfeljebb olyan erős láthatósági paraméterrel rendelkezhet, mint maga az absztrakt metódus. Ha tehát ez a bizonyos elvont tagfüggvény `protected` besorolású, a megvalósító gyermekosztályban csak `protected`, vagy `public` lehet, `private` nem. Ennek az az oka, hogy az elvonatkoztatott osztályt készítő programozónak valószínűleg jó oka volt arra, hogy az adott láthatósági paramétert választotta, s ha ezt szűkítenénk az öröklés során, a további öröklések folyamán megváltozna a tagfüggvény jellege – esetleg teljesen el is tűnne.

Fontos megjegyezni, hogy ha egy osztálynak van legalább egy elvont tagfüggvénye, akkor az osztálynak is elvontnak kell lennie, továbbá az ilyen osztályok nem példányosíthatók, csak a gyermekosztályaik.

Az érthetőség kedvéért íme egy összetett példa: A geometriánál maradva szeretnénk objektumokkal modellezni a szabályos sokszögeket, s elég egyértelmű, hogy a valóságban ezek egy igen egyszerű hierarchiába szervezhetők, próbáljuk ki a programunkban megalkotott világunkban is – az eddigi példáktól teljesen függetlenül!

```
abstract class SzabalyosSokszog {
    protected oldalakSzama = 0;
    protected oldalHossz = 0;

    public function
        ➥__construct($oldalHossz,$oldalakSzama) {
        $this->oldalHossztBeallit($oldalHossz);
        $this->oldalakSzamatBeallit($oldalakSzama);
    }

    public function oldalHossztBeallit($oldalHossz)
    ➥{
        $this->oldalHossz =$oldalHossz;
    }

    public function oldalakSzamatBeallit
    ➥($oldalakSzama) {
        $this->oldalakSzama=$oldalakSzama;
    }

    public function oldalakAltalBezartSzog() {
        return 360/$this->oldalakSzama;
    }

    abstract public function terulet();
}
```

```
class NegySzog extends SzabalyosSokszog {
    public function __construct($oldalHossz) {
        parent::__construct($oldalHossz,4);
    }

    public function terulet() {
        return $this->oldalHossz*$this->oldalHossz;
    }
}


class HatSzog extends SzabalyosSokszog {
    public function __construct($oldalHossz) {
        parent::__construct($oldalHossz,6);
    }

    public function terulet() {
        return 3*$this->oldalHossz*$this->
        ➥oldalHossz*sin(60);
    }
}
```

Kicsit hosszúra sikeredet, de mint látszik, annál egyszerűbb. Íme a `SzabalyosSokszog` osztályunk. Jól látható, hogy vannak olyan szolgáltatások, amelyek minden gyermekre ugyanúgy vonatkoznak. Ezek az oldalhossz beállítása, oldalak számának beállítása, oldalak által bezárt szög kiszámítása. Ez minden sokszög esetében ugyanaz.

De ott van a terület, melynek kiszámítási módját az alaposztály nem ismeri, de tud róla, hogy minden szabályos sokszögnek van területe. Ekkor ezt megjelöli, de nem valósítja meg.

Felületek

A fenti elvont osztályos példa remek akkor, ha olyan tagfüggvényeket szeretnénk készíteni, amit nem feltétlenül kötelező minden örökösnek megvalósítani. Felmerülhet a kérdés, hogy ezeken a gyermekosztályokon valamilyen közös műveleteket végezzünk, mondjuk mindnek kiszámítsuk a területét. Ilyenkor elengedhetetlen az, hogy minden gyermekosztály rendelkezzen a `terulet()` tagfüggvénnyel, különben programhibával le fog állni a futás. A megoldás a felületek alkalmazása.

A felületek olyan osztálydefiníciók, amelyek tartalmazzák, hogy az őket megvalósító osztályoknak pontosan milyen tagfüggvényeket kell megvalósítaniuk. Hasonló a függvénydefinícióhoz, csak ez az osztályokra vonatkozik. A felületek az absztrakt osztályokkal ellentétben nem tartalmazhatnak semmilyen megvalósítást, csak az osztály minimális „tervét”. Nem tiltott, hogy a megvalósító osztály az előírtnál több tagfüggvényt tartalmazzon, ám ha nem készít el minden megadott metódust, a program végzetes hibával leáll. Egy ilyen felület minden metódusa nyilvános (`public`) kell legyen – ez a felületek természetéből adódik.

Felületeket tényleg csak akkor használjunk, ha minden egyes metódusra szükségünk van. Így az osztályaink egymásnak megfelelők (`compatible`) lesznek. Sok esetben nincs szükség az ilyesmire, azon esetekben inkább az elvont osztályokat használjuk. Nézzünk egy példát ismét a `SzabalyosSokszog` osztálytól függetlenül.

```
interface SikidomTulajdonsagok {
    public function terulet();
    public function kerulet();
    public function oldalakSzama();
    public function tengelyesenSzimmetrikus();
    public function kozeppontosanSzimmetrikus();
}

class Teglalap implements SikidomTulajdonsagok {
    //implementálni kell minden tagfüggvényt, de
    //ezen túl készíthetünk konstruktort, oldalbe-
    //állítót,stb. Az a későbbi feldolgozást
    // nem érinti
}
```

Előfordulhat, hogy több felületnek is meg kell felelnünk, ekkor az `impements` kulcsszó után vesszővel elválasztva kell megadni az egyes felületek nevét, amit megvalósítunk.
A felületek megvalósítása természetesen nem zárja ki azt, hogy az osztályunk örököljön is.
Módosítsuk a `SzabalyosSokszog` példánkat úgy, hogy kötelező legyen a `terulet()` tagfüggvény használata.

```
interface Teruletes {
    public function terulet();
}

abstract class SzabalyosSokszog implements
➥Teruletes {
    protected oldalakSzama = 0;
    protected oldalHossz = 0;

    public function __construct
    ➥($oldalHossz,$oldalakSzama) {
        $this->oldalHossztBeallit($oldalHossz);
        $this->oldalakSzamatBeallit($oldalakSzama);
    }

    public function oldalHossztBeallit
    ➥($oldalHossz) {
        $this->oldalHossz =$oldalHossz;
    }

    public function oldalakSzamatBeallit
    ➥($oldalakSzama) {
        $this->oldalakSzama=$oldalakSzama;
    }

    public function oldalakAltalBezartSzog() {
        return 360/$this->oldalakSzama;
    }

}
```

Ekkor a gyermekosztályokhoz nem is kell hozzányúlni, de minden későbbit úgy kell elkészíteni,hogy tartalmazz a `terulet()` tagfüggvényt. Felmerülhet a kérdés, hogy miért nem jelez hibát a PHP a fenti esetben, holott az osztályban nem is valósítottuk meg azt a bizonyos `terulet()` metódust. Ennek az az oka, hogy ez egy elvont osztály, tehát nem példányosodhat, ennek értelmében biztosan nem fogja megsérteni a szabályt. Megsértik viszont azok az örökösök, akik elmulasztják eme tagfüggvény megvalósítását. Jelen esetben tehát csak ennyi szerepe van osztályunk elvont voltának, mert mint láthatjuk, nem tartalmaz egyetlen elvont metódust sem. Az is egy megoldás lett volna továbbá, ha a szülőt változatlanul hagyjuk és a gyermekosztályoknál az öröklés után megmondjuk, hogy ezek a `Teruletes` nevű felületet valósítják meg. (A megoldás hátránya többek között az, hogy így minden sokszögfajtára le kell ellenőriznünk a használat során, hogy implementálják-e a várt felületeket)
A gyakorlatban remekül lehet kombinálni az öröklést a felületeket és az elvont osztályok alkalmazását. Sok esetben vezet igen-igen érdekes eredményre. Ha jobban megnézzük, a fenti esetben is ezt alkalmazzuk. Természetesen a sok osztálynak, melyek ugyanazt a felületet valósítják meg, nem kell feltétlenül egy helyről öröklődniük, lehetnek teljesen függetlenek, látni fogjuk később, hogy lehetőségünk van ellenőrizni egy osztályról, objektumról, hogy megvalósítja-e a szerintünk szükséges felületet, felületeket.

Néhány mágikus tagfüggvény

Az összes ilyen különleges metódusnak közös tulajdonsága, hogy ezeket nem mi, programozók hívjuk, hanem a *PHP*. Hogy tiszta legyen a kép: ide tartoznak a konstruktorok és destruktorok, ám azok ismertetése elengedhetetlen volt az alapvetésnél. Most a `__set()`, `__get()` és a `__call()` tagfüggvényeket vizsgáljuk meg közelebbről. Megjegyezném, hogy a *PHP 5* ezen kívül is tartalmaz még olyan függvényeket, amelyek __-rel kezdődnek (ilyen előtag azonosítja ugyanis a különleges tagfüggvényeket), ám ezek tárgyalása a szűkös terjedelmi keretek miatt most elmarad.

A __set() tagfüggvény

Ha egy osztály, objektum tartalmazza ezt a metódust, akkor minden olyan esetben lefut, ha mi az adott objektum nem létező tulajdonságának adunk értéket. Ez olyankor lehet hasznos, ha egy dinamikusan bővülő adathalmaz alkotja osztályunk tulajdonságait. Példaképp, ha a program futása során különböző gyümölcsök színét szeretnénk összegyűjteni. Célszerű ilyenkor a gyümölcsöket nem külön-külön felvenni, mint osztálytulajdonságot, hiszen az fix, ehelyett jó lenne dinamikusan bővíteni. Mivel azért a nyelv nem teszi lehetővé, hogy futásidőben ily módon piszkáljuk az osztályokat, osztálytulajdonság gyanánt használjunk struktúrát, asszociatív tömböt a megoldásra. Ez kellően rugalmas. Az esetleges beállító tagfüggvényekkel ugyanez a helyzet: nem elég rugalmasak. Itt jön a képbe a `__set()` metódus, nézzük, hogyan:

```
<?php
class Gyumolcsok {
    private $gyumolcsok = array();

    public function __set($name,$value) {
        $this->gyumolcsok[$name]=$value
    }
}
$deliGyumolcsok = new Gyumolcsok();
$deliGyumolcsok->narancs="sarga";
```

```
$deliGyumolcsok->citrom="citromsarga";
$deliGyumolcsok->banan="erdekesen sarga";
?>
```

A __set() tagfüggvény első paramétere a változó neve (ami a ->után szerepel), a második paraméter pedig az érték, ami az egyenlőségjel után szerepel.
A feladat, tehát megoldva. Bármilyen gyümölcsöt beletehetünk, kötöttségek nélkül, az érték nem fog elveszni. Vigyáznunk kell azonban, hogy semmilyen gyümölcsnév ne szerepeljen osztálytulajdonságként, mert ha teszem azt van $narancs nevű osztályváltozó, akkor a második hívás annak értékére fog vonatkozni, nem fut le a __set() metódus.

A __get() metódusa

Ha már van egy ilyen szabadon feltölthető osztályunk, nem ártana, ha legalább ilyen szabadon hozzáférhetnénk. A __set() párja, a __get() siet ilyenkor a segítségünkre. Teljesen analóg módon: ez akkor fut le, ha olyan változó értékére vagyunk kíváncsiak, amely nem szerepel az osztálytulajdonságok között. A fentiek ismeretében bővítsük tovább az osztályunkat.

```
<?php
class Gyumolcsok {
    private $gyumolcsok = array();

    public function __set($name,$value) {
        $this->gyumolcsok[$name]=$value
    }

    public function __get($name) {
        if (array_key_exists($name,
           ➥$this->gyumolcsok))
            return $this->gyumolcsok[$name];
        else
            echo 'Nincs ilyen gyümölcs!';
    }
}
$deliGyumolcsok = new Gyumolcsok();
$deliGyumolcsok->narancs="sarga";
$deliGyumolcsok->citrom="citromsarga";
echo $deliGyumolcsok->narancs;
echo $deliGyumolcsok->banan;
?>
```

A példa az első esetben kiírja, hogy sárga, a második esetben, hogy Nincs ilyen gyümölcs. A megoldás egyértelmű: a __get() paramétere tartalmazza, hogy milyen változóra hivatkoztunk. Megnézzük, hogy létezik-e az adott kulccsal érték a tömbben. Ha igen, akkor visszaadjuk azt, különben kiírjuk, hogy nincs olyan.

A __call() metódus

Ez a tagfüggvény is illeszkedik az előző kettőre, ám ez akkor fut le, ha olyan tagfüggvényt hívunk meg az objektumunk esetében, amely nem létezik. Az előző kettő esetben a program semmilyen hibát nem ír ki, a __get() ill. __set() metódusok hiányában sem, ha olyan változóra hivatkozunk, ami nem létezik. A tagfüggvények hívása esetén ez nincs így, ezért még akár hibaelnyelés céljából is hasznos lehet. A metódusnak két paramétere van. Az első a meghívott tagfüggvény neve, a második pedig a híváskor átadott paraméterek tömbje. Ennek segítségével bizonyos hívásokat elkaphatunk, s különböző műveleteket végezhetünk, mielőtt visszatérnénk. (A visszatérési érték természetesen ugyanúgy viselkedik, mintha valóban létezne az a metódus). A cél itt is az volt, hogy az objektumok futás idejű dinamikus viselkedését növeljék.

Egy kakukktojás – az automatikus betöltés

Bár nem kapcsolódik szorosan az objektumközpontú viselkedésmódhoz, de mindenképp hasznos és érdekes móka az egyes osztályok igény szerinti betöltésének lehetősége.
A *PHP* számára eddig is előnynek számított, hogy maga a forráskód futásidőben alakítható volt az include, require parancsokkal – ha például feltételhez kötöttük őket. Így ugyanis elérhettük, hogy mindig csak a minimálisan szükséges méretű kódot kellett a *PHP*-nak értelmeznie, lefordítania, amivel jelentős sebességnövekedést eredményezett.
Az osztályokat gyakorta helyezzük el külön-külön fájlokban, ami átláthatóvá teszi kódunkat, ám elég kényelmetlen, hogy minden egyes példányosítás előtt be kell töltenünk az adott fájlt. Erre nyújthat megoldást az __autoload() nevű különleges függvény, amely minden olyan esetben meghívódik, ha nem definiált osztályt szeretnénk használni, példányosítani. A függvény meghívása után a *PHP* még egyszer megpróbálja használni azt a bizonyos osztályt, s ha ekkor sem sikerül, hibaüzenettel leáll. A gyakorlatban mindez így nézhet ki:

```
<?php
function __autoload($className) {
  include_once($className.'.php');
}

$obj = new ProbaOsztaly();
$obj2 = new MasikProbaOsztaly();
?>
```

Ez természetesen nem az egyetlen felhasználási mód, egész sor automatikát vihetünk általa programunkba.

Végszó

Nos, még mindig nem értünk a téma végére, de már közel járunk. Hátra van még a nagy újításnak számító ***Refleciton API***, amely az objektumok, osztályok részletes elemzésére szolgál, de nem beszéltünk még a típus előírásról (***type hinting***), a kivételkezelésről és az objektumok másolásáról sem. A cikksorozat következő részében ezekre lehet tehát számítani.

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

Hálózatok (13. rész)
Nagy sebességű és műholdas hálózatok, hálózati réteg

Nem szóltam még a nagy sebességű LAN-okról és a műholdas hálózatokról. Ebben a részben pótolom ezt, majd belekezdek a következő nagyobb témakörbe, a hálózati réteg feladatainak ismertetésébe.

Ebben a részben befejezzük az adatkapcsolati rétegről történő értekezést, és rátérünk a következő nagy témakörünk, a hálózati réteg bemutatására. Mielőtt azonban ezt megtenném, törlesztem két adósságomat: szólok pár szót a 10 Mb/s feletti sebességű *LAN*-okról és a műholdas hálózatokról.

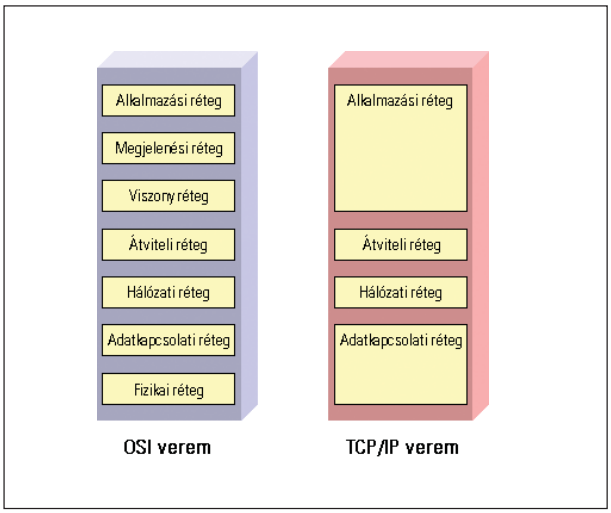
Gyors Ethernet

Eddig többnyire olyan *LAN*-okkal foglalkoztunk, amelyeknek a lelke egyetlen rézvezeték volt. Erre a rézvezetékre kapcsolódtak az állomások, akik állandóan versenyre keltek egymással a csatorna használati jogáért. Amíg az állomások közötti távolság kicsi, és a felhasználók nem álmodoznak nagyobb átviteli sebességről, addig az ilyen hálózatok minden igényt kielégítenek. Ha azonban felmerülnek ehhez hasonló óhajok, akkor vagy lecseréljük a rézvezetéket üvegszálas kábelekre, vagy több egymással párhuzamos rézvezetéket létesítünk az állomások között. A 90-es évek elején felmerült tehát az igény egy új gyorsabb hálózati szabvány kifejlesztésére, így az *IEEE* fel is kérte erre a *802.3* bizottságot. Két koncepció látott napvilágot. Egyesek úgy tartották, hogy el kellene vetni teljes egészében a 802.3-as szabványt, és egy teljesen elölről kezdeni mindent, vadonatúj protokollokkal és szolgáltatásokkal (például digitális hangátvitel). Mások úgy gondolták, hogy fontosabb, ha az új szabvány kompatibilis marad a *802.3* szabvánnyal, a különbség csak a sebességben mutatkozna. Ez a két egymástól homlokegyenest eltérő meggondolás remek alapot szolgáltatott éjszakába nyúló vitákhoz. A bizottság végül az utóbbi elv (a kompatibilitás megtartása) mellett voksolt, amelybe a másik tábor nem tudott beletörődni, így létrehozták saját szabványukat (amely a *802.12* alapja lett).
Valóban nehéz kérdés, hogy a kettő közül melyik elgondolás a célravezetőbb, hiszen az *Ethernet*-el is sok probléma van (erről már részletesen beszámoltam egy régebbi epizódban), viszont a már meglévő több ezer (esetleg millió?) hálózat miatt fontos a kompatibilitás, arról nem is beszélve, hogy az *Ethernet* már egy jól működő dolog volt. Az újabb protokollok és szabványok bevezetése mindig magukban rejthetnek olyan problémákat, amelyekre a tervezés során senki sem gondolt.
1995 júliusában létrejött tehát az új *IEEE* szabvány, amely a *802.3u* kódnevet kapta, de leginkább *gyors Ethernet (fast Ethernet)* néven híresült el. Mivel ennek hátrafelé kompatibilisnek kellett maradnia, ezért nem volt szabad az interfészeket, illetve a keretformátumokat megváltoztatni. A bitidőt azonban csökkenteni lehetett a tizedrészére, így elvileg tízszer nagyobb sebesség érhető el.
A gyors *Ethernet* kábelezése viszont nem engedi meg a *10Base-5*, illetve a *10Base-2* által használt kábelezéseket. Nem használhatunk tehát *BNC* és vámpír csatlakozókat. A *10Base-T* kábelezését azonban szimpatikusnak találták, így erre alapoztak. Ha visszaemlékszünk, ez az a kábelezés, ahol csavart érpárokat is elosztókat *(hub)* használunk. Az ilyen gyors *Ethernet* hálózatokat *100Base-T*-nek nevezzük.
Lehetőség kínálkozik arra is, hogy csavart érpárok helyett üvegszálas kábeleket használjunk (*100Base-FX*). Ilyenkor a hub-tól az állomáshoz két, ellentétes irányú üvegszálas kábelt helyezünk el, ezzel biztosítva van a duplex üzemmód. Az üvegszálas kábelek egyik nagy előnye, hogy az állomás és az elosztó közötti távolság jóval nagyobb (akár 2 km) lehet, mint a csavart érpár esetén *(200 méter)*. Az ilyen hálózatok kiépítése költséges, hiszen az olyan elosztók, amelyek nagyobb számú porttal rendelkeznek, elég borsos árúak. Mindezek ellenére megéri a beruházás, hiszen nem kell tartanunk az ütközésekről, az összes állomás bármikor adhat, illetve vehet. Ez a tény már önmagában több nagyságrenddel nagyobb sávszélességet eredményez. Szintén megdobja a költségeket az is, hogy az állomások hálózati csatolójának is támogatnia kell a gyors *Ethernet* szabványt.

Műholdak

Sokféle adatszórásos hálózattal foglalkoztunk már, a műholdas rendszerekre azonban még nem tértünk ki részletesen.

1. ábra Geostacionárius pályák



2. ábra Az OSI és a TCP/IP verem rétegei

Manapság pedig nagyon nehéz lenne az élet műholdak nélkül, amelyek fontos szerepet töltenek be a hírközlésben és az adatátvitelben. De vajon pontosan mi is a feladatuk, miként is működnek, és milyen típusai repkednek a fejünk felett?

Távközlési szempontból az a legnagyobb baj a ***Földdel***, hogy gömbölyű. Ezért például rádióhullámok segítségével nem lehet két egymástól nagyon távol elhelyezkedő ponttal kapcsolatot kiépíteni. Persze átjátszó állomások segítségével megoldható a probléma, csak hát ezeket meg kell építeni, karban kell tartani, és fohászkodni, hogy ne sérüljön meg például valamilyen természeti katasztrófa következtében.

Másik megoldás, hogy keresünk egy égitestet, amely visszaveri az általunk küldött elektromágneses jeleket. Ilyen például a ***Hold***. Az 50-es években folytak is kísérletek egy olyan navigációs rendszer kifejlesztésére, amelynek alapja a ***Hold*** által visszavert elektromágneses hullámok érzékelése volt. A ***Hold*** azonban nem képes a visszavert jelek felerősítésére. Nem úgy mint a távközlési műholdak, amelyeknek ez a legfontosabb feladatuk.

A műholdakat legkönnyebben keringésük pályája szerint csoportosíthatjuk. Az *1. ábrán* láthatunk pár kitüntetett keringési pályát. Kékkel jelöltük az úgynevezett ***geostacionárius*** pályát, amelynek síkja egybe esik az Egyenlítőével. Ha az egyenlítő felett 35792 km-es magasságban helyezünk el műholdakat, akkor azok keringési ideje meg fog egyezni a ***Föld*** tengely körüli forgásának idejével, amely majdnem 24 óra. Ez azt jelenti, hogy a ***Föld*** felszínéről nézve a műhold az égbolton egy helyben áll. (Persze ez csak látszólagos nyugalom, a műhold valójában 3064 km/s-os sebességgel száguld).

A pálya ezen tulajdonságát leginkább a televíziós műsorszórással megbízott műholdak tudják kihasználni. A ***Földön*** a TV előtt bambuló felhasználóknak sincs más dolguk, minthogy egyszer irányba állítani antennájukat. Egy ilyen műhold a Föld felszínének körülbelül 38%-át képes besugározni, így könnyedén kiszámíthatjuk, hogy három darab műholdat tartalmazó hálózattal az egész bolygót lefedhetjük.

Az úgynevezett egyenes vagy direkt műholdpályák az egyenlítő síkjával 0 – 90 fokot zárnak be. Ezen többnyire navigációs, felderítő műholdak közlekednek, többnyire alacsonyabb röppályán. A napszinkron pálya esetében a pálya síkja és a Nap iránya által bezárt szög minden esetben állandó. Ez azt jelenti, hogy itt a műholdak „együtt járnak a nappal", azaz egy adott pont fölé mindig ugyanabban az időpontban ér.

Nem minden műhold lakik olyan magasan, mint a geostacionárius műholdak. Manapság egyre nagyobb szerep jut az úgynevezett alacsony röppályás műholdaknak is. Ezekkel a műholdakkal az a gond, hogy csak kevés ideig vannak látótávolságban. Ezt a hátrányt úgy akarták orvosolni, hogy amint hatótávolságon kívülre ér az egyik műhold, egy másik pont akkor lép be. Ez volt az alapja az ***Iridium*** nevű projektnek is. Itt eredetileg 77 (végül már csak 66) darab műholdat szerettek volna 750 km-es magasságba állítani abból a célból, hogy egy olyan világméretű hálózatot alakítsanak ki, amelyek kézi eszközök között teremtene kapcsolatot. Sajnos a projekt nem hozta meg a várt sikert, valószínűleg a magas percdíjak és a ***GSM*** hálózat elterjedése miatt.

Csatornakiosztás műholdas hálózatok esetén

A műholdas hálózatok az egyetlen olyan ***WAN***-ok, amelyek a ***LAN***-okhoz hasonlóan osztott csatornát használnak. A földi állomások a műholddal az úgynevezett ***felfelé irányuló (uplink)*** frekvencián küldenek kereteket a műhold felé. A műhold feladata a hub-okéhoz hasonló: mindent meg kell ismételniük, amit hallanak. Így a beékezett jelet felerősítve szétszórják a ***lefelé irányuló (downlink)*** frekvencián. Fontos, hogy az ***uplink*** és ***downlink*** frekvenciák különbözőek, így nem zavarják egymást a beérkező és a kimenő hullámok. Mivel a ***downlink*** frekvencián csak a műhold beszél, ezért versenyhelyzet csak a felfelé irányuló csatornán alakulhat ki.

A kérdés már csak az, hogy miként oszthatjuk fel a közös csatornát az állomások között. Most kicsit más a helyzet mint a *LAN*-ok esetében, ugyanis a távolságok jóval nagyobbak. Igaz, hogy a jelek fénysebességgel terjednek, de a műholdak olyan messze vannak, hogy még így is számolnunk kell az átlagosan 270 ms-os jelterjedési idővel. Ez pedig azzal a sajnálatos következménnyel jár, hogy nem megoldható a vivőjel érzékelés (azaz nem tudjuk megállapítani, hogy ad-e éppen valaki rajtunk kívül). Ha egy állomás belehallgat a lefelé irányuló csatornába, akkor csak azt tudja meg, hogy 270 ms-al ezelőtt valaki adott, amely a számítógépek számára a távoli múlt. Arra esély sincs, hogy megtudjuk, mi történik a jelenben.

Problémánkra a legegyszerűbb megoldás a ***lekérdezés (polling)***. Amikor egy közös csatornát meg akarunk osztani több állomással, akkor azt úgy is megtehetjük, hogy körbe kérdezünk mindenkit, van-e épp elküldendő adata. A műhold persze ezt nem teheti meg, mivel a 270 ms-os jelterjedési idő miatt rendkívül nagy késleltetési idővel kéne számolni. Ha az állomásokat azonban össze tudjuk kötni egy kis sávszélességű földi hálózattal, akkor logikai gyűrűbe szervezhetjük őket. Innentől kezdve csak az adhat, akinél a vezérjel van. Kis állandó számú állomások esetén ez egy hatékony módszer lehet.

Műholdak esetében is használható a már régebben bemutatott réselt *ALOHA*, *FDM* és *TDM* csatornaosztásos módszerek. Mi most helyhiányra hivatkozva csak a *TDM*-et mutatjuk be műholdas környezetben.

A *TDM* esetében meg van határozva pontosan, hogy melyik állomás mikor adhat. Ehhez fel kell osztanunk egy időintervallumot egyenlő részekre, úgynevezett időrésekre. Minden időrést hozzárendelünk egy-egy állomáshoz.

Az állomások csak akkor adhatnak, amikor éppen a saját időrésükben vannak. Ehhez élből két problémát is meg kell oldanunk: egyrészt egyeztetni kell az állomások óráját, hiszen mindenkinek tudnia kell, hogy mikor kezdődik a saját időrése. Másrészt az időréseket el kell tudnunk osztani az állomások között.

Nézzük először az első problémát. Az állomások szinkronizációjához használhatjuk magát a műholdat. Egy speciális földi állomás, a referenciaállomás szabályos időközönként egy jelet bocsát a műhold felé, amely szétszórja azt az állomások között. Ez lesz az úgynevezett óraindító jel, ehhez képest fogják kiszámolni az időrések kezdetét.

Ha például egy időrés *T* időpillanatig tart, akkor az *n*-edik időrés kezdete az óraindító jeltől számított *k*T* idő múlva következik be.

Az időréseket ki lehet osztani statikusan és dinamikusan. Kis állandó számú állomások esetén megfelelő az első módszer, de ezt az esetet leszámítva mindig a dinamikus kiosztás a célravezető. Erre több módszer is kínálkozik.

Az első módszer él azzal a feltételezéssel, hogy legalább annyi időrés van, mint állomás. Ez azt jelenti, hogy minden állomás rendelkezik egy „saját bejáratú” időréssel. Ha egy állomás ki van kapcsolva, vagy nincs küldendő adata, akkor az időrése kihasználatlanul marad. Erről tudomást szerez a többi állomás (a lefelé irányuló csatorna figyelésével), így

legközelebb az állomások versenyezhetnek ezért az időrésért. Ha az időrés tulajdonosának mondanivalója akad, és vissza szeretné szerezni saját időrését, akkor nem kell mást tennie, mint egy keretet elküldenie. Ilyenkor ütközés következik be, ami után (a tulajdonost kivéve) egyik állomás sem fogja használni az adott időrést. Ha megfigyeljük, ez egy hatékony módszer abból a szempontból, hogy az állomások viszonylag rövid időn belül szóhoz jutnak, viszont alacsony csatornakihasználtság esetén nem a legjobb hatásfokot produkálja. Nem beszélve arról, hogy csak akkor használható, ha előre ismerjük az állomások számát. Ha ismeretlen, sőt mi több, változó számú állomásokkal van dolgunk, az időréseknek nem lehet állandó tulajdonosa.
A másik módszer erre az esetre kínál megoldást. Az időrésekért az állomások a réselt ***ALOHA***-nál bemutatott módszerhez hasonlóan versenyeznek. Ha egy állomás sikeresen forgalmazott egy keretet (nem történt ütközés), akkor legközelebb is adhat ugyanabban az időrésben. Ezzel elméletileg egy állomás bármeddig adhat, amíg csak van elküldendő adata. A gyakorlatban azonban az állomások figyelnek arra, hogy ne adjanak túl sok időrésen keresztül, és másokat is hagyjanak szóhoz jutni.

A hálózati réteg

Befejeztük végre az adatkapcsolati réteg, illetve a közegelérési alréteg tárgyalását. Most egy magasabb szintre lépünk: a hálózati réteg szintjére. Hasonlóan, ahogy eddig is tettük: először megfogalmazzuk, hogy mi is pontosan a réteg feladata, illetve milyen problémákra kell megoldást nyújtania. Ezek után következik csak a gyakorlati megvalósítás ismertetése.
A hálózati réteg feladata a csomagok célbajuttatása. Mivel a forrás- és a célállomás gyakran különböző hálózatokban van, ezért a csomagoknak akár több útválasztón is át kell haladniuk. Láthatjuk, hogy ez a feladat mennyire elkülönül az adatkapcsolati réteg feladatától. Az utóbbi szinten csak kereteket kellett továbbítanunk a csatorna egyik végéről a másikra. Az adatkapcsolati réteg tehát két szomszédos, vagy legalábbis azonos csatornát használó állomások között valósította meg az átvitelt. A hálózati réteg viszont két tetszőleges végpont között teremt kapcsolatot. Eddig nem volt olyan rétegünk, amely erre képes lett volna, tehát a hálózati réteg a legalacsonyabb olyan szint, ahol a hálózati végpontok kommunikálhatnak egymással.
Fontos megérteni, hogy a hálózati réteg nem foglalkozik közvetlenül az átvitel megvalósításával, hiszen arra ott vannak az adatkapcsolati réteg szolgálatai. Két útválasztó (***router***) között az átvitel általában pont-pont kapcsolattal van megvalósítva, amelynek kezelését az adatkapcsolati réteg végzi. A hálózati réteg igazi feladata a csomag útbaigazítása. Mivel a hálózati réteg ismeri a kommunikációs alhálózat (a routerek hálózatának) felépítését, ezért csak ő tudja, hogy a forrástól milyen útvonalon juthat el a csomag a célhoz. Ráadásul mindezt úgy kell megvalósítania, hogy elkerülje egyes útválasztók terheltségét, a torlódások kialakulását. A forgalmat a lehető legegyenletesebben kell elosztania.
A hálózati réteg a szállítási réteg ***(transport layer)*** számára kínál szolgálatokat. Ugyanúgy, mint a többi már ismertetett réteg esetében, a hálózati rétegnek el kell rejtenie a megvalósítás kérdéseit. A szállítási réteget nem fogja érdekelni a kommunikációban résztvevő alhálózatok száma, topológiája, típusa. A szolgálatoknak függetlennek kell lennie az alhálózatok felépítésétől. A szállítási réteg mindig csak egy címet szeretne mondani, ahová a csomagot szánja. Nem tudja, és nem is érdekli, hogy az adott cím fizikailag merre is található. Ehhez persze az is szükségeltetik, hogy legyen egy olyan egységes címzési rendszer, amelyek a szállítási és a többi felsőbb réteg egyértelműen azonosíthatja a végpontokat, a hálózat kiterjedésétől függetlenül (azaz ugyanazt a címzést lehessen használni ***LAN***-ok és ***WAN***-ok esetében is).
Más kikötés nincs előírva a hálózati réteg szolgálatai számára. Ez azt jelenti, hogy a hálózatok tervezői viszonylag nagy szabadságot kaptak a megvalósításban. Rajtuk áll például az is, hogy a hálózati réteg összeköttetés alapú vagy összeköttetés nélküli szolgálatot alakítsanak-e ki. Sorozatunk első részében sokat foglalkoztunk e kér szolgálat típus tulajdonságaival. Ott az összeköttetés alapú szolgálatot a telefonrendszerrel szemléltettük: ahhoz, hogy kommunikálhassunk valakivel, először fel kell hívnunk őt, azaz fel kell építeni a kapcsolatot. Miután mindent megbeszéltünk, a kapcsolatot végül le kell bontani. Az összeköttetés nélküli szolgálatra a legjobb példa a postai úton történő levelezés.
Az internet is ezt a megoldást alkalmazza. Ő a hálózati réteget egy egyszerű postásnak tekinti, akinek a feladata kimerül a csomag kézbesítésében. Ezért a felsőbb rétegek fel sem tételezhetik azt, hogy a célhoz a csomagok feladásuk sorrendjében, sértetlenül fognak megérkezni, továbbá nem várhatnak el olyan bonyolult szolgáltatásokat, mint a forgalomszabályozás. Egyszerűbben úgy fogalmazhatjuk meg, hogy a hálózati réteg interfészében két primitív található: a csomag küldése és a csomag fogadása primitív. (A primitívek a felsőbb rétegek által meghívható elemi műveletek).
Nem minden hálózat követi ezt az elvet, az ***ATM*** hálózatok például összeköttetés alapúak. Itt a kommunikáló két fél először felépít egy kapcsolatot, a csomagok sorrendhelyesen érkeznek meg, és még a forgalomszabályozás is biztosítva van, tehát az adó nem adhat gyorsabban, mint ahogy azt a vevő fel tudja dolgozni.
A két szemlélet között az igazi különbség az, hogy a hálózati modell mely rétege végezze a nehéz, összetett munkát. Az első megközelítésben ezt a szállítási réteg végzi. Ő fogja majd a csomagokat sorrendbe helyezni, figyelni a hibákra és a forgalomirányításra. Ezt legtöbbször az operációs rendszer szintjén valósítják meg, tehát maguk a gépek (***host***) fogják a munka számításigényes részét átvállalni. A másik szemlélet szerint a felhasználókat meg kell kímélni attól, hogy saját számítógépeiken futtassanak összetett számításokat igénylő protokollokat.
Mindkét megoldásnak megvan a maga előnye és hátránya. Erre részletesen kitérünk majd a következő részben, ahol foglalkozunk majd a hálózati réteg belső felépítésével, és megismerkedünk pár forgalomirányító algoritmussal.

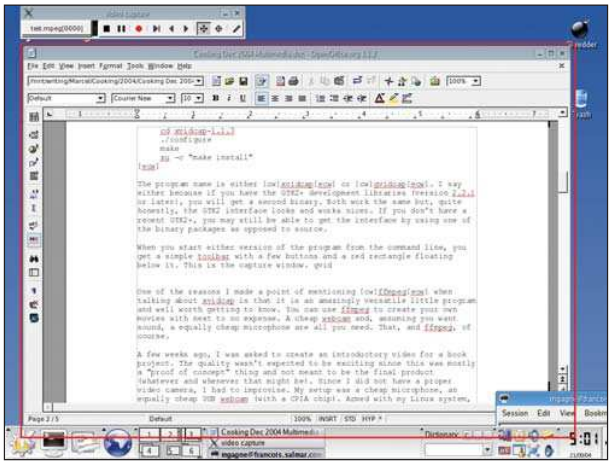
Garzó András (garzoand@interware.hu)
Körülbelül három éve foglalkozik Linux- és más Unix-rendszerekkel. Legjobban az operációs rendszerek lelkivilága érdekli, de nyitott egyéniség. Kedvenc étele a palacsinta, és van egy Richard nevű macskája. Minden észrevételt, megjegyzést, levelet szívesen fogad.

Lámpák...Kamera...Tessék!

Elég egy webkamera, egy mikrofon, néhány nyílt forrású csomag, és máris összeáll egyszerű videóstúdiónk.

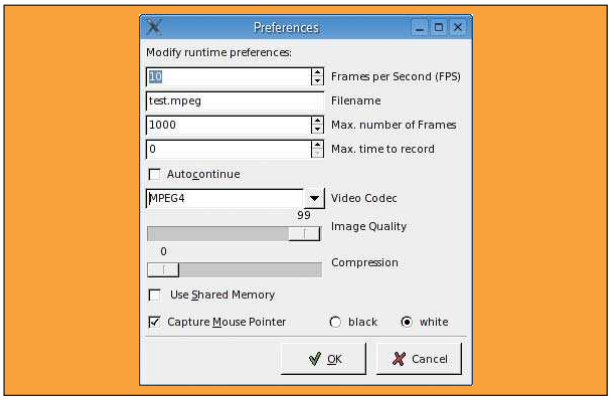
Attól tartok, igaz, ***François***. A kamera nem hazudik. Így nézel ki, miközben bort szolgálsz fel. Ó, dehogy, ***mon ami***, eszembe sem jutott, hogy kitegyem a filmet a vendéglő weblapjára. Hogy miért csinálom? Aaaa…! Te még nem láttad a mai ajánlatunkat? Multimédia és szórakozás a jelszó, ***mon ami***, és mint magad is látod, ez egy roppant szórakoztató kis filmecske. Ne izgasd fel magad, ***François***, a végsőkig tiéd a tiszteletem. Különben is, marakodásra most nincs idő, vendégeink bármelyik pillanatban itt lehetnek. Már ott is toporognak az ajtóban! Üdvözöl benneteket, ***mes amis***, a ***Chez Marcel***! Olyan jó itt látni benneteket, a világ legjobb linuxos vendéglőjében, és egyben a világ egyik legpompásabb borospincéjében. Foglaljatok helyet, helyezzétek magatok kényelembe. ***François*** éppen most indult a pincébe. A 2001-es ***Santa Lucia Highlands Pinot Noir*** szinte itatja magát. Rikító borfajta, a sztárok világát idézően sziporkázó – kiválóan illik mai menünkhöz.
Talán már nektek is feltűnt, hogy manapság minden munkaállomás fel van szerelve kamerával és mikrofonnal. Kézenfekvő ötlet, hogy mi – vagy valamelyik társunk – legyünk saját filmünk sztárjai, ám a középpontba ezúttal a programok kerülnek. Végül is, a ***Chez Marcel*** menüjén már nem egy alkalmazás ragyogott fel, ***non***? Itt jön a képbe ***Karl Beckers xvidcap*** nevű programja. Az ***xvidcap*** kiváló kis eszköz az X munkaasztalunkon történtek rögzítésére, amihez csupán az ***ffmpeg*** nevű programot használja segítségül, ami viszont valószínűleg már fent van a gépünkön. Eredményként egy ***MPEG*** filmet kapunk munkaasztalunk egy részéről vagy egészéről.
Azt kérditek, mire jó ez? Például oktatási célú filmeket készíthetünk vele, amelyen megmutathatjuk másoknak, hogyan használjanak egy-egy alkalmazást, esetleg közhírré tehetjük, milyen jók vagyunk a kedvenc lövöldözős játékunkban. Az ***xvidcap*** beszerzéséhez látogassunk el a hivatalos weboldalra. (Lásd a kapcsolódó címeket.) Az oldalról forrás és bináris csomagokat is letölthetünk, utóbbiak ***RPM*** és ***DEB*** kiterjesztéssel szerepelnek. Ha saját terjesztésünkhöz nincs előre lefordított csomag, akkor az ***xvidcap*** fordítását a megszokott öt lépést követve végezhetjük el:

```
tar -xzvf xvidcap-1.1.3.tar.gz
cd xvidcap-1.1.3
./configure
make
su -c "make install"
```



1. ábra A felvételi ablak beállítása xvidcap/gvidcap alatt

A program neve `xvidcap` vagy `gvidcap`. Azért említek két nevet, mert ha rendelkezünk a ***GTK2+*** fejlesztői könyvtárral (2.2.1-es vagy újabb változat), akkor egy második bináris fájlt is kapunk. Mindkét példány azonos módon működik, de őszintén szólva a ***GTK2+*** felület sokkal jobban néz ki. A felületet akkor is elérhetjük, ha nincs fent a gépünkön valamelyik újabb ***GTK2+***, de ilyenkor ne a forrást, hanem valamelyik bináris csomagot válasszuk.
A program bármelyik változatát is indítjuk el a parancssorból, egy egyszerű eszköztárat fogunk látni néhány gombbal, alatta egy vörös téglalappal. Ez a rögzítési ablak, ezt kell a felvenni kívánt területre mozgatnunk. Az alapértelmezett ablak meglehetősen kis méretű. Ha meg szeretnénk változtatni a felvétel tárgyát képező területet, kattintsunk a célkeresztet ábrázoló ikonra (jobbról a második az eszköztáron), majd kattint és húz módszerrel állítsuk be a kívánt területet. A vörös téglalap átméreteződik. Az eszköztáron megtaláljuk a felvétel elindításához, leállításához, szüneteltetéséhez stb. szükséges gombokat. Ha kicsit elidézünk a gombok felett, megjelennek a vonatkozó tanácsok.
Ennyi, máris filmre vehetjük a vörös téglalapon belül történő eseményeket. (*1. ábra*)



2. ábra A gvidcap beállító párbeszédpanelje

A beállítások révén megváltoztathatjuk a másodpercenként felvett képkockák számát, az alkalmazott mozgókép-kodeket, valamint megadhatjuk, hogy az egérmutató szerepeljen-e a filmben. Kattintsunk az egér jobb gombjával az eszköztár bal szélső gombjára, majd válasszuk a ***Preferences (Beállítások)*** parancsot. (*2. ábra*) Ha tettünk már néhány próbát, a `man xvidcap` paranccsal azt is kideríthetjük, hogyan használhatjuk grafikus felület nélkül a programot. Külön érdekesség, hogy a program hangrögzítésre is alkalmas, ami különösen oktatási célú filmek készítésekor hasznos.
Az ***ffmpeget*** azért említettem meg már az ***xvidcap*** bemutatásakor is, mert lenyűgözően sokoldalú program, érdemes megismerkedni vele. Az ***ffmpeg*** segítségével szinte ingyen készíthetünk saját filmeket. Elég egy olcsó webkamera, ha hangot is akarunk, akkor kell egy egyszerű mikrofon, de másra nincs is szükségünk, kivéve persze az ***ffmpeget***.
Néhány hete felkértek, hogy készítsek egy könyvhöz egy bevezető videót. A minőség nem volt szempont, az egész egy kísérleti dolog volt, messze nem a végleges megoldás. Mivel megfelelő videokamera nem volt kéznél, rögtönöznöm kellett, és azt hiszem, a világ legolcsóbb videóstúdióját sikerült összeállítanom. (*3. ábra*) A felszerelés egy olcsó mikrofonból és egy szintén alsó polcos, ***USB***-s, ***CPIA*** lapkával felszerelt webkamerából állt. ***Linuxos*** rendszeremmel felfegyverkezve bármire készen álltam. A kérdés csak az volt: „Hogyan?”
Az ***ffmpeg*** segítségével felvettem a filmet, aztán kísérletezgettem egy kicsit az időzítéssel, a képfrissítéssel és a többi beállítással – végül egész meggyőző eredményt kaptam. Az alábbi paranccsal egy ***AVI*** formátumú, tíz képkocka/másodperc frissítésű filmet vettem fel:

```
ffmpeg -vd /dev/video -ad /dev/sound/dsp -r 10 \
-s 352x288 video_teszt.avi
```

Ennyi. A bemeneti videóeszköz (ezt a `-vd` kapcsoló adja meg) a `/dev/video`, a hangforrás (`-ad` kapcsoló) pedig a `/dev/sound/dsp` volt. Természetesen más gépeknél az eszközök neve ettől eltérő is lehet. Az egyik másik gépemen például az ***USB***-s videóeszköz a `/dev/video0`. Ha a parancsot ebben a formában adjuk ki, akkor a felvétel egészen a merevlemez megteléséig folytatódik. Ha korlátozni akarjuk a felvétel hosszát, például 10 másodpercre, akkor a `-t` kapcsolót kell használnunk:

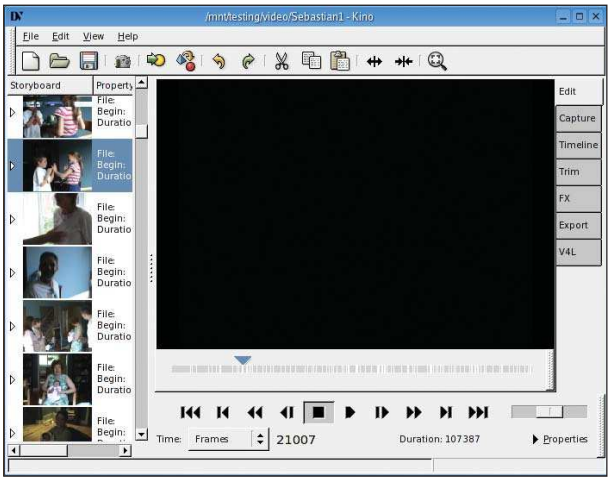
3. ábra A világ legolcsóbb videóstúdiója?

```
ffmpeg -vd /dev/video -ad /dev/sound/dsp -r 10 \
-s 352x288 -t 10 video_teszt.avi
```

A létrejövő filmet melynek neve `video_teszt.avi` lesz, ***Xine***, ***Kmplayer***, ***MPlayer*** vagy egyéb hasonló program segítségével játszhatjuk le.
Nos, ***mes amis***, azt hiszem, sikeresen betörtem a költséghatékony videostúdiók piacára. Amikor elküldtem a felvett filmet, megkértek, hogy inkább ***MPG*** formátumban adjam oda. Természetesen nem forgattam újra a filmet, inkább az alábbi paranccsal átalakítottam a fájlt:

```
ffmpeg -i video_teszt.avi video_teszt.mpg
```

Amilyen sokoldalú az ***ffmpeg***, biztosan jó néhány percet el tudunk időzni súgóoldalának tanulmányozásával. Ígérem, rengeteg érdekes szolgáltatást fogunk felfedezni.
Azok számára, akik rendelkeznek digitális videokamerával, nem okozhat gondot a jó minőségű felvételek elkészítése. Ez esetben a kihívást inkább a felvétel számítógépre juttatása jelenti, hiszen szerkesztését, tömörítését és másoknak is elküldhető lemezre írását leginkább így tudjuk elvégezni.
Nekem egy ***Sony Handycam*** kamerám van. Van rajta ***USB*** kapu, de támogat egy ennél jóval gyorsabb soros buszt is, amit a legtöbben ***FireWire*** névvel ismernek. A műszaki témák iránt fogékonyabbak sokszor ***IEEE1394***-nek hívják. Számítógépünk oldalán szintén szükségünk van egy ilyen portra vagy csatolókártyára, ami ugyan költséget jelent, ám az ***IEEE1394*** busz teljesítménye messze kárpótol ezért.
Ha a kapott felvételt értelmes módon akarjuk szerkeszteni, akkor meg kell kaparintanunk egy megfelelő szerkesztőprogramot, például ***Arne Schirmacher Kino***-ját. (Lásd a kapcsolódó címeket.) A ***dvgrab*** program, mely szintén a ***Kino*** weboldaláról érhető el, a ***Kino*** része, így ezt nem kell külön letöltenünk. Látjátok, ***mes amis***, az ***xvidcaphez*** hasonlóan a ***Kino*** is rejt parancssori eszközöket a csinos felület alatt. Ezek egyike jó öreg barátunk, az ***ffmpeg***.
Mára ***Arne*** kiváló fejlesztőket gyűjtött maga köré, akik rendkívül csinos, könnyen használható linuxos videoszerkesztő alkalmazást állítottak össze. A ***Kino*** egyik szépsége abból fakad, hogy az ***IEEE1394***-et támogató videokame-



4. ábra Házi felvétel szerkesztése a Kino-val

ránkról képes közvetlenül átmásolni a szerkesztendő felvételt. Ha rendelkezünk ilyen ***FireWire*** kábelen keresztül csatlakoztatható kamerával, akkor még vezérlését is a ***Kino***-ra bízhatjuk. Mielőtt továbblépnénk, hadd ejtsek néhány szót az ***IEEE1394*** támogatásáról. Az újabb ***Linux*** terjesztésekből nem hiányozhat az ***IEEE1394*** – betölthető rendszermagmodulok révén megvalósuló – támogatása, ám például az én tesztrendszeremen az illesztőprogramok nem töltődtek be önműködően. Sebaj, betöltöttem őket kézzel:

```
modprobe ohci1394
modprobe raw1394
chmod 666 /dev/raw1394
```

A digitális felvétel átmásolása az ***IEEE1394*** kábelen keresztül a számítógépre a parancssorból is könnyedén elvégezhető a ***dvgrab***-bel. Tulajdonképpen a ***Kino*** is ezt a módszert követi. Bár a másolást egyszerűen a `dvgrab` paranccsal is elindíthatjuk, inkább használjuk a `-i` kapcsolót, mely interaktív módba állítja a programot. Ezt követően vezérelni tudjuk a kamerát, és egyszerű gombnyomásokkal irányíthatjuk a rögzítést.

```
dvgrab -i
Going interactive. Press ´?´ for help.
q=quit, p=play, c=capture, Esc=stop, h=reverse,
➥j=backward scan,
k=pause, l=forward scan, a=rewind, z=fast forward,
➥0-9=trickplay,
<space>=play/pause
Capture Started
```

Igaz, hogy a legtöbben megszoktuk a grafikus felületek használatát, de ne féljünk, mindez sokkal egyszerűbb és elegánsabb, mint ahogy hangzik. A létrejövő fájl neve alapesetben `dvgrab-XXX.avi` lesz, ahol az `XXX` három számjegyet takar.
A ***Kino*** felhasználói felülete letisztult, könnyen érthető és kezelhető. (*4. ábra*) A főablak jobb oldalán lévő fülekkel érhetjük el a különféle szolgáltatásokat, kezdve a rögzítéstől, a szerkesztésen keresztül egészen a késztermék kiírásáig. A munka túlnyomó részét az ***Edit (Szerkesztés)*** lapon végezzük el; persze csak a rögzítés után. Itt tudjuk egyesíteni és szétvágni a jeleneteket, de további fájlokat is beilleszthetünk filmtervezetünkbe. A ***Timeline (Idővonal)*** fül az éppen kiválasztott jelenetet képkockákra bontja, így a film tetszőleges pontjára ugorhatunk úgy, hogy nem kell az egészet lejátszanunk.
Az ***FX*** lapon különleges hatásokat adhatunk hozzá a filmhez, ilyen például a visszirányú lejátszás, a szépia hatás, a tükrözés és a kaleidoszkóp. A hanghatások között elhalkulást, felerősödést és némítást találunk.
Ha ráéreztünk a digitális videózás ízére, valószínűleg néhány percesnél hosszabb felvételeket is készítünk majd. Mindezt azért hozom szóba, mert amikor a ***Kino***-val átmásoljuk a felvételt a számítógépre, több nagyméretű, körülbelül 820 MB-os fájl is létre fog jönni, ezek tartalmazzák a film egyes részeit. Nyilvánvaló, hogy végeredményül nem ilyen fájlokat várunk. Pontosan ezért kell megbarátkoznunk az ***Export (Kimentés)*** lappal. Ezen – allapok egy csoportján – különféle lehetőségeket találunk, például külön kimenthetjük ***WAV*** fájlokba a hangsávot. Nekem nagyon tetszett az egyes képkockák kimentésének lehetősége is. Az utóbbi időkben a legtöbbet mégis a ***DV pipe (DV csővezeték)*** szolgáltatást használom, ahol az ***ffmpeg*** újfent szerephez jut. Amikor minden szerkesztéssel és vágással végeztem, az ***ffmpeg*** segítségével videó CD ***AVI*** fájlokba vagy DVD formátumban mentem ki a munkámat. Az így kapott, például ***VCD AVI*** formátumú fájlok jóval kisebbek és könnyebben kezelhetők. Egy órányi digitális mozgókép körülbelül 9 GB lemezhelyt foglal, a kimentett film ugyanakkor kényelmesen elfér egyetlen 700 MB-os CD-lemezen.
Mint látjátok, ***mes amis***, linuxos rendszeretek kiváló eszközöket biztosít ahhoz, hogy ti magatok – vagy alkalmazásaitok – sztárokká válhassatok. Olyan jól elbeszélgettünk, hogy közben elérkezett a záróra, és borfakasztotta mosolyotokat nézve úgy látom, sikerült néhány emlékezetes felvételt készítenetek. Na jó, csak vicceltem. ***François***, légy oly kedves, töltsd még egyszer újra vendégeink poharát. Lehet, hogy érdemes volna megörökítenünk az utókornak a búcsúzó pohárköszöntőt?... A következő alkalomig igyunk egymás egészségére, ***mes amis***! A ***votre santé! Bon appétit!***

Linux Journal 2004. december, 128. szám

Marcel Gagné (mggagne@salmar.com) Ontarioban, Mississaugában él. Legújabb, immár harmadik könyve a Moving to the Linux Business Desktop (ISBN 0-131-42192-1, Addison-Wesley). A rendszerintegrálással és hálózati tanácsadással foglalkozó Salmar Consulting, Inc. igazgatója. Hobbipilóta, tudományos-fantasztikus írások szerzője, jelenleg egy kisebb origami T-Rexen dolgozik.

KAPCSOLÓDÓ CÍMEK

ffmpeg: ➲ ffmpeg.sourceforge.net
Kino és dvgrab: ➲ kino.schirmacher.de
xvidcap/gvidcap: ➲ xvidcap.sourceforge.net

Az átdolgozás joga, avagy a szoftver és a zene

Vagyoni jogi jogosultságok.

Bármennyire hihetetlennek tűnik is első hallásra, jogi szempontból a szerzői jogi védelmet élvező művek közül a szoftverek a zeneművekhez állnak a legközelebb. Legalábbis, ami az átvétel, átdolgozás, feldolgozás, idézés problematikáját illeti[1]. Ugyanis mindegyik alkotásnak van két egymás mellett létező megjelenési formája. A zene, amit hallunk és a szoftver, amit használunk, valamely, a háttérben megbúvó, hétköznapi emberek számára érthetetlen vagy felismerhetetlen dokumentáción nyugszik. Ahogy csak szavakat értünk egy forráskódból, a gépi kódból pedig azt sem, ugyanúgy nehézséget okozhat meglátni vagyis inkább „kihallani" kedvenc dallamunkat az öt vonal közé préselt hangjegyekből. Míg a képzőművészeti alkotás maga a mű, az irodalmi pedig adott esetben maga a könyv, addig a zenéhez és a szoftverekhez valamilyen közvetítő közegre van szükség. Legyen az egy vonós- vagy egy Pentium négyes.

Alapok

Az átdolgozás jogát a magyar törvények „3+1" lényeges helyen szabályozzák. A törvény 4. § (1) bekezdésében rendelkezik az eredeti mű szerzőjének azon jogosultságáról, hogy az ő jogait az átdolgozás, feldolgozás, fordítások készítése esetében is tiszteletben kell tartani, ugyanakkor kimondja, hogy az említett módokon keletkezett szerzői mű szintén jogosult a védelemre, amennyiben egyéni eredeti vonásokat hordoz. A második pont a felhasználási engedélyek pontja, mely szerint ez a felhasználási szerződés (licenc) keretében szabályozott jogosultság kizárólagosan a szerzőtől nyerhető el, a mű bármilyen formában történő felhasználására[2], ugyanakkor külön nyomatékosító szabályként kiemeli a törvény, hogy az átdolgozás csak kifejezett kikötés esetében tekinthető a felhasználási szerződés részének[3].
A harmadik kategóriát a személyiségi jogokat is érintő integritásvédelem[4] jelenti, melynek ellenpólusaként a felhasználót védő rendelkezés áll, mely szerint – amennyiben a szerző hozzájárulását adja a mű felhasználásához, köteles azt „felhasználhatóvá tenni", azaz elvégezni rajta olyasféle, a mű lényegét nem érintő változtatásokat, melyek elengedhetetlenek illetve nyilvánvalóan szükségesek. Amennyiben erre nem hajlandó vagy nem képes, a felhasználó ennek más módon való megvalósításáról gondoskodhat[5]. Ehhez kapcsolódik a törvény speciálisan szoftverekre vonatkoztatott része, ahol a többszörözés és fordítás kötelező engedélyeztetése alól mentesítést biztosít, amennyiben az említett esethez hasonlóan a szoftver működtetéséhez ez szükséges. Valamint érdemes azt is megjegyezni, hogy harmadik személyek „beavatkozását" – ezen lényegi részt nem érintő változtatás eszközölése céljából – abban az esetben is lehetővé teszi, ha ezt előtte a szerzőtől nem is kérték.

Az ami, majdnem az, ami...

A „+1" kicsit kakukktojás részlet, ugyanis szintén ezen a ponton kívánom tárgyalni a jogos felhasználásnak a szabad felhasználás körébe tartozó részleteit, azaz az idézést, és átvételt[6].
Átdolgozás esetében egy alapul szolgáló mű mellett egy származékos mű is megjelenik. Az pedig, hogy valóban átdolgozásról, vagy egy – az eredetitől független – egyéni eredeti alkotásról van-e szó, a két mű között fellelhető kapcsolat léte és mélysége alapján döntendő el. A mérlegelés mindannyiszor egyedi eset, és szakértő rendelése mellett történik. Csupán az a körülmény, hogy bizonyos elemek mindkét alkotásban – akár visszatérően – megjelennek, nem alapozza meg az átdolgozás tényének megállapíthatóságát. Ahogy a zenében annak az esélye, hogy két zeneszerző egymástól függetlenül pontosan ugyanazt a dallamot találja ki, majdnem olyan kicsi, mint hogy egy adott probléma megoldására két programozó ugyanazt a forráskódú programot hozza létre. Azonban az azonos dallam kitalálás esélye nő, ha a tizenkét fokú komponálás alapelveit követve hozzák létre. Ahogy szoftver esetében bizonyos irányvonalat ad már magának a programnak, hogy melyik programozási nyelvet választja a szerző alapul.

[1] Az elemzés támpontja Gyenge Anikó: Zeneművek átdolgozása a szerzői jogban c. tanulmánya in: Iparjogvédelmi és szerzői jogi szemle 2003/10. http://www.hpo.hu/ipsz/200206/zenemuvek.htm 2003. december 21. 9:00
[2] Szjt 16.§ (1)
[3] Szjt. 47§ (1)
[4] Szjt 13.§
[5] Szjt 50.§
[6] Szjt 34§ (1) és (2)

Részlet kérdések

További nevesített, ám ki nem fejtett fogalmakat rejt a törvény, mikor szintén szerzői jogosultságokat rendel azon személy részére, aki a művel kapcsolatosan az átdolgozáson túl, feldolgozással vagy fordítással nyújt egyéni, eredeti teljesítményt. Szoftverek és zeneművek esetében sem lehet igazán értelmezni a fordítás fogalmát. Kifejezetten fordításnak ugyanis egyik értelemben sincsen helye, hiszen a zene minden nyelven beszél, a szoftverek esetében pedig a fordítás alatt azt az elengedhetetlen folyamatot értjük, mikor a forráskódból gépi kódot készítünk, ezen eljárás pedig mentes minden egyéni, eredeti jellegtől, hiszen egy másik (fordító)program készíti el számunkra a megkívánt formát. Így hát a következőkben tárgyalt alapfogalmak két fő kategóriába sorolhatók, egyrészt a jogszerű, a szerző által engedélyezett (bizonyos esetekben ezt meg sem kívánó) átvétel, másrészt a műnek illetve egy részletének másik műben való jogszerűtlen megjelenése.

Hasonló tulajdonsága a zenének és a szoftvernek, hogy az átdolgozása csak műfajon belül valósul meg. Előfordulnak persze extrém esetek, mikor valaki megfest egy szimfóniát vagy egy extrém tárlaton bemutat egy gépi kódokból álló képet. Valamivel gyakoribb az, mikor egy zeneműhöz utólagosan készül szöveg, ám ezeket az eseteket leszámítva ezen műfajtákra nem jellemző az átdolgozásokkal elérhető műfajváltás, ahogy például egy filmből regény vagy regényből film lesz.

A feldolgozás fogalmának tisztázásánál azt a halvány elválasztó vonalat kell megtalálnunk, melyet voltaképpen a szabályozásnál maga a törvény sem követ szisztematikusan, hiszen az átdolgozásra vonatkozó szabályok alkalmazását rendeli a feldolgozások tekintetében is.

Amennyiben nem adjuk fel a kutatást a konkrét elhatárolási pontokat illetően, egy 1973-ból származó (bár a szoftverekről még tudomást sem vevő) szerzői jogi kézikönyvében találhatunk némi eligazítást[7].

„Az átdolgozás mindig egy műfajon belüli beavatkozás."[8]

Míg a feldolgozott művek általában elhagyják eredeti címüket és műfajukat, de megtartják az eredeti legjellegzetesebb tartalmi és gondolati egységét. Ami azonban a jelenlegi szabályozást illeti, a fent nevezett éles elhatárolás félrevezető lehet, hiszen ***„a törvényi felsorolás azt tükrözi, hogy az átdolgozásnak és az eredetiséget el nem érő, de ahhoz hasonló formai változtatásoknak a különböző műfajokban, a gyakorlatban sokféle elnevezése alakult ki... Ezek használata egyáltalán nem következetes a jogi tartalmat illetően."***[9]

Az elhalványodott egyéni vonás átvétele esetén nem engedély nélküli többszörözésről, hanem feldolgozásról beszélhetünk. Ha követjük a zenei irányvonalát[10] és átdolgozás alatt a javító célú, rendszerint kisebb beavatkozásokat értjük, akkor ide sorolható a szabad szoftverek alkotásának jónéhány lépése, azaz, mikor egy profi programozó egy már más által megkezdett, de félbehagyott, vagy nem tökéletesen megalkotott programot forráskódsorok beépítésével, vagy éppen cseréjével sokkal gördülékenyebben működővé, hibamentesebbé tesz.

A zenében feldolgozásnak tekintett rész (mely szerzői jogi szemüvegen át továbbra is az átdolgozás részlete) valójában egy tágabb kört ölel fel, a lexikonokban meghatározott zenei alapanyagon végzett átalakító tevékenységek összességét. Az ***arrangement*** kifejezés (a mű más berendezésű apparátusra való alkalmazása, mint amelyre az eredetileg készült) a szoftverek hardverhez igazításával lehet rokonítható, mikor egy asztali gép alkalmazásait egy ***Palmra*** viszik át, azaz a speciális körülményekhez igazítják, mindamellett, hogy az eredeti mű teljes mértékben felismerhető marad.

A modernizálás ugyanúgy, ahogy a régi zene életre keltése, megjelenik a szoftverfejlesztésben is, hiszen ahogy a népdalok szabadon feldolgozhatóak, ugyanúgy az eredeti, még ***Public Domain*** alatt álló, vagy más okból szabad szoftverek forráskódja, megismerhető és feldolgozhatóak. Ugyanakkor egyes régebbi programokban feleslegessé vált programrészek elhagyása (tekintettel a technika fejlődésére) lehetségessé válik, ám az ilyen kis mértékű változtatás nem engedi áttörni a szerzői minőséghez szükséges egyéni, eredeti jelleget sem, ahogy ***Mahler*** zenei változtatásai sem nyertek átdolgozásként elismerést[11].

Hasonlóan problémás úgy a zenében, mint a szoftverben az idézés kérdése, hiszen még zenében nem lehet „lábjegyzetelni", esetlegesen egyes kottarészletekhez lenne fűzhető, hogy az átvétel mely szerző mely művéből való, az a zene megszólaltatását követően többé nem válna ismertté, hacsak nem műértő füllel hallgatják. Szoftvereknél hasonló a helyzet, hiszen a program futtatása során nem derül már ki, hogy a forráskód részletei kihez és milyen mértékben köthetőek. Kereskedelmi szoftverek esetében még a szerzők feltüntetésére is ritkán kerül sor, hiszen őket általában a forráskódban őrzik, melyet nem hoznak nyilvánosságra, gondolhatjuk tehát, ha az eredeti szerzők név feltüntetéshez való joga ilyen módon sérül, milyen elbánásban részesülhetnek az idézett művek szerzői. A szerzői jogi törvény idézésére vonatkozó rendelkezései[12] sajnálatos módon mindkét esetben elenyésznek.

Természetesen a szoftvereknek vannak egyéb, „specifikus" feldolgozási formái is, ám ezekre a következő számban térünk ki.

Dr. Dudás Ágnes

[7] A szerzői jog kézikönyve, szerk: Bernárd Aurél, Tímár István, Közgazdasági és Jogi Kiadó, Budapest 1973
[8] im. 236. o.
[9] Kommentár 166. o.
[10] http://www.hpo.hu/kiadv/ipsz/200206/zenemuvek.htm
[11] „Mahler karmesteri tapasztalataiból és a közönség ízlésének ismeretéből kiindulva Beethoven és Schumann-szimfóniákat is „kijavított". Egyes helyeken a vonósok eredetileg megkívánt kettőzését megszüntetve, másutt éppen előírva, szünetjeleket átírva, staccato-jeleket beillesztve „átszínezte" a partitúrákat. A változtatásoknak azonban ugyanúgy oka volt Mahler érzékeny hallása, mint a kor általános zenei ízlése. Az eredeti darabokat viszont nem alakította át olyan mértékben, hogy azokon a leghalványabban is Mahler hatása, egyéni-eredeti alkotótevékenysége lenne felismerhető, amelynek alapján a műveket átdolgozásoknak kellene tartanunk." Gyenge Anikó: i.m. Zenei feldolgozások a gyakorlatban http://www.hpo.hu/kiadv/ipsz/200206/zenemuvek.htm
[12] Szjt 34.§ (1)