VIII. GNU/Linux Szakmai konferencia

2006. november 24.

A kiadvány tördelése a TEX 3.14159, a pdfTEX 1.10b
és a Ghostscript 8.53 verziójával készült,
GNU/Linux operációs rendszeren.
Helyesírás-ellenőrzés: Hunspell 1.1.4 és Magyar Ispell 1.1.1
Szóelválasztás: Huhyphn3 2006-09-06

A címlapképet rajzolta:
Macsinka Zsolt

A borítót tervezte:
Kecskeméti László

Szerkesztette:
Szabó Péter *szabo.peter@szszi.hu*

Lektorálták:
Hajdú Gábor, Hazai Géza, Horváth Róbert Ervin,
Jenei Gabriella, Koblinger Egmont, Korn András,
Kovács László, Szabó Péter, Szervác Attila,
Szabó Zoltán és Tímár Gábor

Segítettek:
Balázs Tibor, Nagy Jelena, Szabadi Edina,
Tomka Gergely és Zelena Endre

Az LME logóját rajzolta:
Moldvai Dezső E.

ISBN 978-963-06-1184-8

Linux-felhasználók Magyarországi Egyesülete
1395 Budapest 62, Pf. 432.
Weboldal: *http://www.lme.hu/*
E-mail: *info@lme.linux.hu*

Tartalomjegyzék

A konferencia támogatói

Arany fokozatú támogatók:

Adverticum Zrt.

Alerant Zrt.

Novell Kft.

TMSI Kft. Tőzsér és Máriás Szoftver Iroda, mint a Sophos hazai Primary Partnere

Ezüst fokozatú támogató:

Magyar Telekom

Fő médiatámogató:

Prim Online

Mégiatámogató:

Linuxvilág, a magyar Linux-barátok magazinja

Előszó

Ismét ősz, ismét GNU/Linux konferencia. Nem az első, és remélhetőleg nem is az utolsó. De mint minden, az LME éves rendes konferenciája sem marad változatlan. Idén két újdonsággal szeretnénk kívánatosabbá tenni rendezvényünket.

Első újításunk a konferencia célközönségének változásával jár együtt. A kezdeti pólós, loboncos, egyetemista közönség mellett megjelentek, és egyre nagyobb szerepet kapnak a kisebb-nagyobb vállalatok szakemberei is. Ez együtt jár a GNU/Linux rendszerek terjedésével, és távolról sem ok a borongásra. De a „dolgozó" embereknek fontos a hétvége, ezért most először hétköznapra tettük a konferenciát.

Második újításunk egy új előadói kört céloz meg. Ahogy az eredeti közösség egyre magasabbra hágott a szakmai fejlődés lajtorjáján, úgy lett az előadások színvonala is magas, olykor talán túlzottan is magas. Ez sajnálatos módon ellentétben áll a GNU/Linux rendszerek terjedése során a Linux felé forduló kezdők érdekeivel: nekik más igényeik vannak. Ezért, az eddigi mélyen szakmai előadások mellé idén több, a hétköznapi gyakorlatot bemutató előadást is beválogattunk a repertoárba.

Reméljük, mind a két változás sikeres lesz, és a jövő évi, kilencedik konferenciánkon már ez lesz a hagyomány. Az ideire pedig sok érdekes előadást, és még több hasznos új ismeretséget kívánunk:

LME Elnökség

Professzionális GPL alapú térinformatika: a GRASS Linux környezetben

Bugya Titusz
<titusz@gamma.ttk.pte.hu>

PTE TTK Földrajzi Intézet
Térképészeti és Geoinformatikai Tanszék
Pécs, 7624, Ifjúság útja 6.
Tel.: 72/503-600/4526

Kivonat

Előadásomban a GRASS térinformatikai programrendszerről adok áttekintést. A GRASS felhasználási területének bemutatása után sorra veszem fejlesztésének fontosabb lépéseit, a programrendszer főbb tulajdonságait és összetevőit. A GRASS használatának mikéntjéből néhány példán keresztül adok ízelítőt, felvázolva a rendszer kapcsolódási pontjait más programcsomagok felé. Az előadás végén a GRASS hazai felhasználási lehetőségeiről, az oktatási tapasztalatokról ejtek szót.

A GRASS a GPL keretei között terjeszthető, professzionális térinformatikai szoftverrendszer. Professzionalitása abban áll, hogy a felhasználható eljárások tartománya igen széles, valamint számtalan egyéb programmal illetve programrendszerrel képes együttműködni, adatokat és fájlokat cserélni. Igaz továbbá az is, hogy nem oktatási célzattal fejlesztették illetve fejlesztik, hanem különböző tudományterületek, gyakorlati problémák igényeinek kielégítése a cél. A GRASS-ban végzett műveletek és a megjelenítés pontossága, a műveletek elvégzésének sebessége nem marad el más térinformatikai rendszerekben elvárttól.

Programrendszer mivolta abban áll, hogy nem egy vagy néhány nagyméretű, összetett programfájl futásán keresztül működik. Ehelyett minden kis részprobléma megoldását másmás részprogram végzi, melyek kicsik és hatékonyak, hiszen csupán egy vagy néhány feladat elvégzésére írták őket, de arra nagyon precízen lettek optimalizálva. Mindezek fényében a GRASS-ról elmondható, hogy fejlett térinformatikai eljárásaival, szabad hozzáférhetőségével kiemelkedő alternatíva térinformatikai feladatok megoldása, térinformatikai rendszerek létrehozása, fejlesztése esetében.

A GRASS a *Geographical Resource Analysis Support System* rövidítése. Ez magyarra legszerencsésebben talán *Földrajzi Forráselemzést Támogató Rendszer*ként fordítható. A program fejlesztését az Amerikai Egyesült Államok Mérnökhadtestének Környezeti Osztálya indította el az 1980-as évek elején, 1991-től pedig a rendszert szabadon felhasználhatóvá tették. Így egyrészt széles körben elterjedtté vált, másrészt többen, illetve mások is bekapcsolódhattak a fejlesztésébe.

A GRASS felhasználói köre igencsak széles, ám Magyarországon még mindig csak elszórtan, egy-egy felhasználó számítógépén lehet vele találkozni. Külföldön ugyanakkor, különösen Olaszországban, Németországban és természetesen az USA-ban meglehetősen sokan, és széles körben alkalmazzák: felhasználói között számos egyetem mellett említhetjük az az Egyesült Államokban a központi adminisztráció és a kutatási szféra szereplőit, például

a Nemzeti Talajvédelmi Szolgálatot, a Nemzeti Parkok Szolgálatát, a NASA-t, az Egyesült Államok Geológiai Szolgálatát és az Egyesült Államok Választási Irodáját.

A GRASS Linux és MacOS X környezetben teljes körűen rendelkezésre áll. A Microsoft Corporation által forgalmazott operációs rendszereken a GRASS csak bizonyos kerülőutakon, a Cygwin csomag feltelepítése után futtatható. A program forráskódja szabadon letölthető, így elvileg problémamentesen fordítható valamennyi UNIX rendszerre is. Mivel a működéséhez szükséges programkönyvtárak és egyéb állományok ugyancsak a GPL hatálya alá tartozóak, ezek hiánya sem okozhat problémát. Magának a programrendszernek a használata operációs rendszertől függetlenül, ugyanúgy lehetséges, csupán a könyvtárak, fájlok elérési útja különbözik. A legelterjedtebb Linux-disztribúciókhoz, MacOS X-hez és Windowshoz kész csomagok formájában is letölthető a weboldaláról. Beszerzése, telepítése (tapasztalatom szerint) Debian, MacOS X és Poseidon Linux környezetben a legegyszerűbb, de SUSE, Fedora és UHU-Linux alatt is gond nélkül fut, forrásból fordítva.

A fentiek fényében úttörő jelentőségűnek tekinthető, hogy a Pécsi Tudományegyetem Természettudományi Karán már két éve nem csak kutatási, de oktatási téren is szerepet kap. Egyetemünkön a graduális képzésben környezettan szakon, számítástechnika szakon, valamint térinformatika specializáció alatt vezettük be oktatását. Az eredmények pozitívak. Nem csupán arról van szó, hogy ingyenesen van lehetőség egy, a maga kategóriájában valóban professzionális rendszer használatára. Fontos szempont, hogy így a hallgatók más operációs rendszert is használnak, más számítástechnikai gondolkodásmóddal szembesülnek, szélesebb szakmai horizontra tesznek szert. Mindemellett fontosnak tarjuk, hogy növelheti elhelyezkedési esélyeiket, ha egy olyan rendszerrel is megismerkednek, amely különösebb anyagi ráfordítások nélkül áll rendelkezésre.

A GRASS hazai felhasználása ugyanis potenciálisan igen jelentős. Bár nagyon kevesen ismerik, még kevesebben használják, sok helyen lenne létjogosultsága. Egyrészt az előírások értelmében minden magyarországi önkormányzatnak rendelkeznie kell(ene) működő, naprakész térinformatikai adatbázissal. Ennek teljes körű megvalósulása egyelőre illuzórikusnak nevezhető. Nem utolsó sorban azért, mert kevés a hozzáértő szakember, nagyon sok az elvégzendő munka, valamint drágák a feladat elvégzésére alkalmas szoftverrendszerek. Ez utóbbi probléma rögtön kiesik a GRASS választásával. Könnyítené a megfelelő számú szakember képzését is a szélesebb körű elterjedés: mivel szabadon terjeszthető, minden hallgató szabadon telepítheti otthoni gépére és kedve, illetve szükséglete szerint gyakorolhatja használatát. Fontos szempont továbbá, hogy egy-egy térinformatikai szakterem felszerelésekor nem kell az adott intézménynek súlyos milliókat, esetleg tízmilliókat költenie a megfelelő színvonalú szoftverek beszerzésére.

Másrészt, egyre több a térinformatikai eljárásokat igénylő feladat a magán- és vállalkozói szférában is. Ilyenek a jármű-navigáló berendezésektől a közlekedésirányításon át a szállítási, logisztikai feladatok, de terjed a térinformatika használata a gazdaságirányításban, a mentésszervezésben és még számos egyéb területen. Úgy vélem, ha a leendő szakembereket megtanítjuk egy szabadon felhasználható, professzionális térinformatikai rendszer használatára, javítja elhelyezkedési, vállalkozói esélyeiket is. Ennek megfelelően üdvös lenne a GRASS oktatását mind több felsőoktatási intézményben bevezetni. Ehhez első lépésként elkészült egy 60 oldalas, magyar nyelvű könyvfejezet [1], mely a GRASS használatába nyújt bevezetést. Ennek folytatásaként tervezem egy magyar nyelvű komplett kézikönyv összeállítását, valamint a GRASS magyarításának elkezdését is.

Hivatkozások

[1] Bugya Titusz: Magyar nyelvű könyvfejezet a GRASS-ról.
URL `http://foldrajz.ttk.pte.hu/titusz/grass`.

Egyéb nyílt operációs rendszerek

Dancsok Zoltán

Kivonat

A cikkben néhány kevésbe ismert, aktív fejlesztés alatt álló, nyílt forráskódú operációs rendszert mutatunk be. Az előadás célja felhívni a szabad szoftvert és a nyílt forrású megoldásokat kedvelők figyelmét arra, hogy e törekvések zászlóshajója, a Linux mellett egyéb operációs rendszerek is használatosak. A bemutatott rendszerek (OpenBeOS (Haiku), ReactOS, Visopsys, plan9, OpenDOS, FreeDOS, GNU/DOS és MenuetOS) nem közvetlen versenytársai a Linuxnak: szolgáltatásaik és törekvéseik is lényegesen mások.

1. Bevezető

Ugyan a Linux valóban olyan operációs rendszer, amely minden további nélkül munkára fogható a legspeciálisabb feladatok esetében is – de korántsem az egyetlen ilyen rendszer. Nyílt forrású keretek közt számos olyan fejlesztés jött létre, amely egy-egy feladatra gyorsabb, pontosabb, kisebb, könnyebb, kulcsrakészebb megoldás lehet. A cikk ezen rendszerekből nyújt ízelítőt.

2. Alternatív nyílt forrású operációs rendszerek

2.1. OpenBeOS (Haiku)

Az OpenBeOS a Be Inc. által tervezett és létrehozott BeOS operációs rendszer nyílt forrású verziója. A Be Inc. több, mint 1 évtizedig fejlesztette a BeOS-t (ejtsd: bí-ó-esz), amely egészen a 4-es verzióig nem is futott x86 platformon, csak PowerPC-s gépeken. Sőt, egy időben saját PPC-alapú hardvert is készítettek hozzá; ez volt a BeBox. Sajnos a Release 5 (R5) után, 1999 és 2000 során a cég anyagi gondokkal szembesült, csődbe ment, és végül a Palm felvásárolta.

A BeOS felhasználóbarát, gyors, pontos és fejlett rendszer volt. Fénykorában több területen megelőzte korát – és magánvéleményem szerint a Linuxnak lenne mit tanulnia tőle. Sajnos sosem nyitották meg a forráskódját, és nagyon sok szimpatizánsnak ezzel mintha a szívét tépték volna ki.

2001-ben a BeOS néhány külső fejlesztője (vagyis nem a Be Inc. alkalmazottja) egy új, nyílt forrású operációs rendszer fejlesztésébe kezdett. A rendszer kezdetben OpenBeOS névre hallgatott, de később átkeresztelték Haikura[1]. A Haiku idén ünnepelte az 5. születésnapját; az 1. ábrán egy tipikusnak mondható képrenyőképet láthatunk. A BeOS eredeti formájában sajnos az örök cybermezőkre költözött, ám a Haiku a nyílt, a Zeta pedig zárt fejlesztés keretében folyamatosan fejleszti tovább a rendszert.

Mitől is volt jó a BeOS? A Be Inc. szerint a BeOS „média-oprendszer” volt, vagyis ki-

[1] „A haiku a japán költészet egyik jellegzetes versformája, amely a XX. század elejétől egyre több nyelv irodalmában megjelent. Három sorból áll, melyek rendre 5, 7 és 5 morásak (a fordításokban szótagosak). A haikut nagyon erős zeneiség jellemzi, részben a szimmetrikus forma ritmusa, részben a magán- és mássalhangzók hangulati értéke miatt, amelyekre a vers rövidsége miatt a befogadó is nagyobb figyelemmel van.” – írja a Wikipedia.

1. ábra. Haiku (OpenBeOS) képernyőkép

2. ábra. Zeta 1.1 képernyőkép (az rdesktop ablakon belül távoli Windows fut)

fejezetten nagy terhelésre és pontosságra volt kihegyezve, ami létszükséglet a médiaállományok lejátszása és szerkesztése során. A BeOS erőforrás-beosztása rendkívül jó. Az R5 verzió hardverigénye egy 166 MHz-es processzor és 32 MB memória. Ezen a „vason” viszont már egyszerre tudunk internetezni, levelezni, dokumentumokat írni, méghozzá remek sebességgel. A 2. ábrán jól látható a rendhagyó grafikus felület (az ábra a BeOS zárt forrású utódán, a Zeta rendszeren készült képernyőkép). Ez, bár kicsit szokatlannak tűnik, egy félelmetesen jól átgondolt felület, amelynek használatát elsajátítani csupán percek kérdése, sebessége pedig egyedülálló. A grafikus felület során komolyan figyelembe vették a felhasználók szokásait, a

billentyűparancsok tervezésekor pedig az ergonómiát. Az átgondoltsága minden tekintetben elüt a Windows- és Linux-rendszerekétől, és a legapróbb részletekig végigkíséri a rendszert, beleértve a kényelmes C++ API-t is.

A BeOS fájlrendszere, a BFS, egy 64 bites naplózó fájlrendszer. Ez a maga korában (1998) személyi számítógépen teljesen egyedülálló volt. További hasznos szolgáltatása a BFS-nek, hogy indexeli a fájlneveket (tehát az adott mintára illeszkedő fájlneveket egy pillanat alatt megtálja).

A BeOS-nél találkozhattunk először igazán jól működő *plug & play*-jel: a rendszer indításkor felismeri a hardvert, és betölti az illesztőprogramját. Ez annak idején nagy újdonság volt, és kevés rendszer implementálta maradéktalanul. A *plug & play*-nek köszönhető például, hogy hangkártyacsere után már induláskor hallhatjuk a bejelentkező zenét.

A BeOS-ben jelent meg a *replikáns technológia* nevű érdekes eljárás is. A lényege az, hogy egy-egy program több helyen is futhat egy időben, akár egy másik programon belül is. Ez a gyakorlatban azt jelenti, hogy a BeOS pici óraprogramját bele tudjuk hajítani egy dokumentumba, amelyet mentve, majd később megnyitva újra megtaláljuk benne az órát; sőt, mi több, a pontos időt fogja mutatni.

A BeOS fő hátrányai a következők: sosem volt széles körben támogatott rendszer; habár sok apró program létezik, ami rendkívül használhatóvá teszi, több, a saját korában elvárt dolgot sem tudott, a mai igények közül pedig még többet nem elégít ki. Például még csak most készülnek korrekt meghajtóprogramok a 3D grafikus kártyákhoz.

A BeOS és a Haiku bátran ajánlható kis gépekre, nagyon régi laptopokra, ahol más rendszer már nem tudna olyan gördülékenyen futni, mint ezek. Ilyen gépeken problémamentesen végezhetünk el alapvető feladatokat anélkül, hogy túlterhelnénk a gépet. Az [1] weboldalon számos apró programocska található BeOS-hoz, továbbá innen letölthető a magyar változat (Hungarian Edition 3.0), amely a Personal Editionre épül.

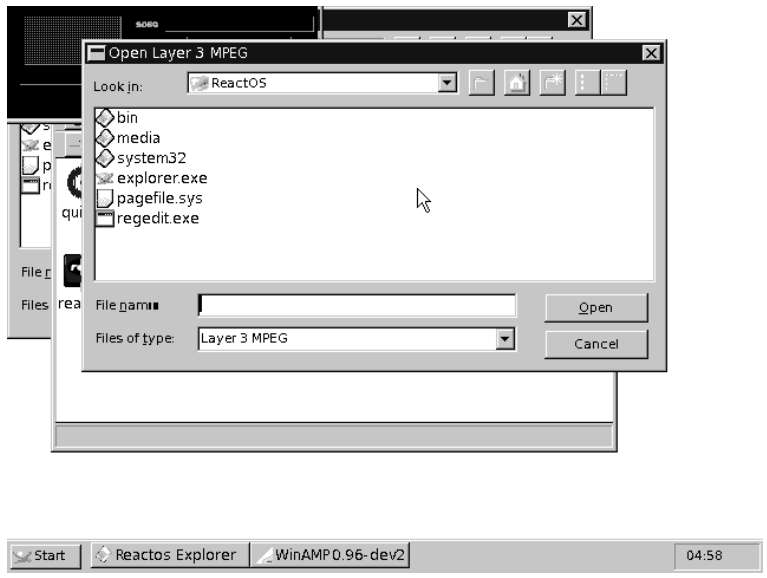
2.2. ReactOS

A ReactOS egy nagyon ígéretes és nagyon érdekes projekt. Közel tízéves története során szinte semmit nem lehetett róla hallani, még linuxos körökben is újdonságként hat. Pedig a projekt célja kiemelkedő, fejlettsége alapján pedig komolyan esélyes a sikerre. A cél: egy olyan, a Windows 2000-rel/XP-vel kompatibilis operációs rendszer létrehozása, amelyre minden probléma nélkül telepíthetőek a fenti Windows verziókra írt programok és eszközmeghajtók. A képeken jelenleg a ReactOS 0.3.0-as verziója látható.

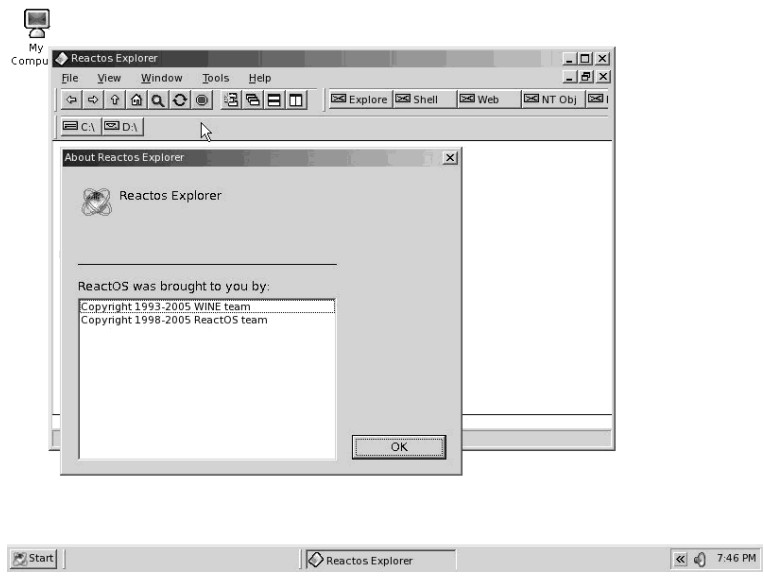
A ReactOS jelenleg rendelkezik egy gyors és egyszerű telepítővel, amely a `C:\ReactOS` mappába képes telepíteni a rendszert, de csak a FAT32 fájlrendszert támogatja. A grafikus felület (3. ábra) természetesen a Windowsra hasonlít, az intéző viszont már kissé eltér a Windowsétól. (4. ábra). A telepítés során létrejön a regisztrációs adatbázis, a `system32` mappa és számos olyan dolog, amit a másik rendszer alatt már megszokhattunk.

A lista, amelyben vezetik, hogy mely programokkal és meghajtókkal kompatibilis, folyamatosan bővül, és a lokalizáció is folyamatosan követi a fejlesztést. A weboldalán található képernyőképek igen ígéretesek; a képek alapján az 1.0-s verzióban egy „OpenWindows” várható.

A ReactOS hardverigénye nagyobb, mint az előbb említett BeOS-é. Futtatásához mindenképpen egy 500 MHz-es gép javallott, és legalább 128 MB memória. A stabilitásával még természetesen lehetnek gondok. Ezenkívül azt is korai lenne elvárni tőle, hogy telepítés után bármely, a másik rendszerre íródott program gond nélkül telepíthető és futtatható legyen – de a helyzet egyre jobb.



3. ábra. A ReactOS fájlmegnyitó ablakának képernyőképe



4. ábra. A ReactOS Intézőjének képernyőképe

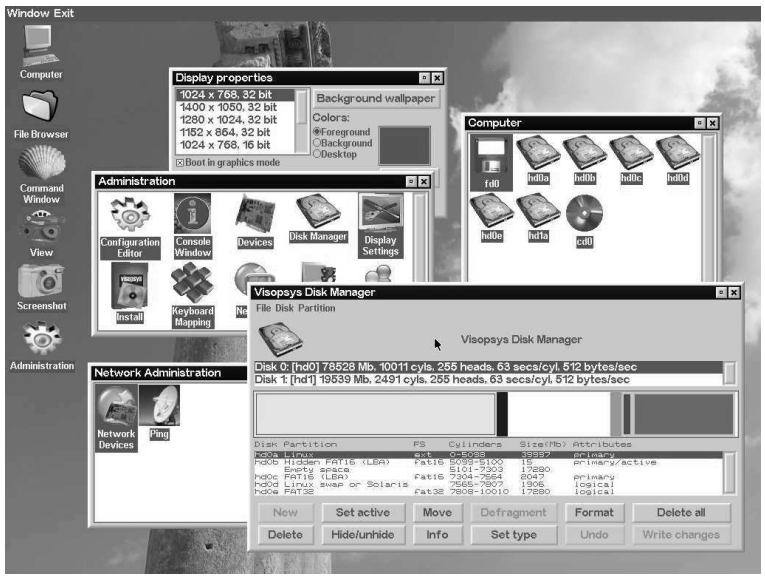
2.3. Visopsys

A Visopsys egy nyílt forrású alternatív operációs rendszer. A fejlesztés célja ennyi, és nem több. A fejlesztők szerint mindig jól jön egy nagyon pici, ám barátságos és gyors rendszer a lemezre vésztartalékként: ha a nagy rendszerünkkel baj történik, akkor a kis Visopsys segítségével képesek lehetünk elhárítani a hibákat.

A Visopsys fejlesztése 1997 óta zajlik. Jelenleg rendkívüli kicsi mérete ellenére teljes grafikus felületet nyújt (lásd az 5. és 6. ábrákon), valódi, ám egyszerű multitasking és virtuálismemória-kezelés mellett. Hardvertámogatása minimális, tehát alapszinten kezeli a SCSI és az IDE lemezeket, a képernyőt képes meghajtani VESA módban, valamint kezeli a billentyű-

zetet és az egeret. A letölthető verzió `.iso` képfájlja mindössze 9,1 MB (!), és ez egyszerre Live CD és telepíthető verzió is. A sebességére igazán nem lehet panasz; az erőforrásoknak csak töredékét használja ki, és a feladatát is jól elvégzi. Főleg Windows mellé lehet értelme feltenni, mivel csak FAT fájlrendszerekre tud írni (olvasni tudja még az ext2-t, az ext3-at és az ISO9660-ot). A partícionáló csak alapfeladatokra képes, ám azokat remekül ellátja. A Visopsys hardverigénye minimális, egy Pentium 1-es már elég neki. További felhasználási területei lehetnek az oktatásban, hiszen ezzel a számítástechnika sokkal közelebb hozható már a kisiskolás gyerekekhez is.

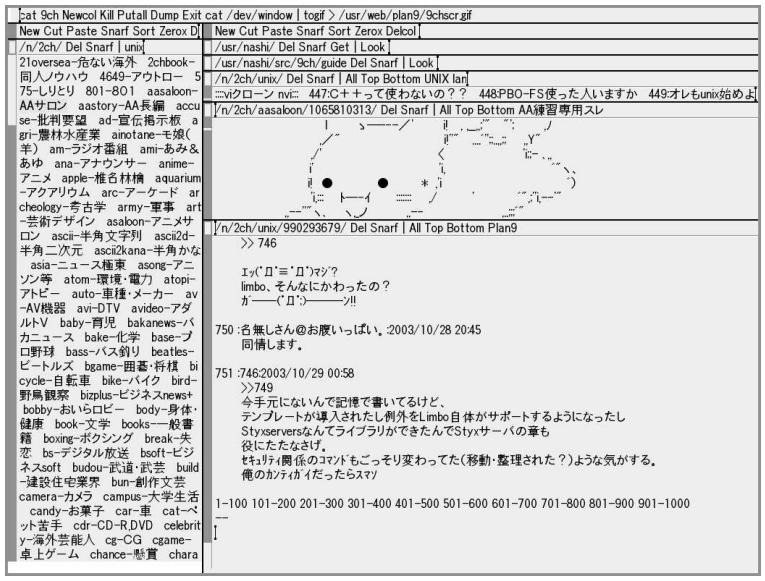
5. ábra. A Visopsys bejelentkező-képernyője



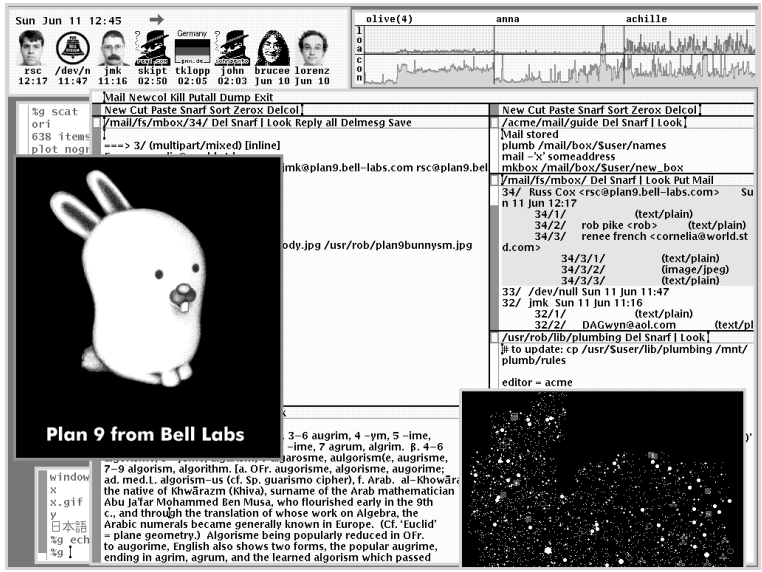
6. ábra. Visopsys munkában – képernyőkép

2.4. plan9

Időutazás. A plan9 a Bell laboratórium által fejlesztett Unix, amely külsőleg évtizedes lemaradásban van, ugyanakkor belülről igencsak korszerű, és rengeteg kellemes meglepetést tartogat az érdeklődőknek. Jelenleg a 4. kiadása számít frissnek. Ám mielőtt ebbe belemennénk, fontos megjegyezni, hogy a plan9 nem a kezdők rendszere. Önmagamat meglehetősen tájékozottnak tartom, legalábbis ami az operációs rendszereket illeti, ugyanakkor ez a rendszer nekem is feladta a leckét, nem is kicsit. De nézzük csak, milyen érdekességeket lehet felsorolni nagy hirtelen. Az `.iso` fájl 235 MB méretű, de ez ne tévesszen meg senkit sem, ettől még nagy tudású a rendszer. Az első meglepetés akkor ér minket, amikor a CD-n több



7. ábra. A plan9 többablakos szövegszerkesztés közben – képernyőkép



8. ábra. A plan9 munkában – képernyőkép

példányban is megtaláljuk a rendszert, így x86 mellett telepíthető többek között ARM és PPC rendszerekre is. Jó tudni, hogy a letöltött CD képes Live CD-ként is működni. Telepítése odafigyelést igényel, de több év linuxos vagy unixos tapasztalattal azért minden további nélkül kivitelezhető. Talán annyit fontos megjegyezni, hogy saját fájlrendszere van, a fossil; fontos tulajdonsága, hogy lehet hálózaton éppúgy, mint a helyi gépen.

A rendszer és a fájlrendszer három védelmi réteget biztosít, tehát ahhoz, hogy a felhasználó hozzáférhessen az adataihoz, a kérésnek három rétegen kell keresztülverekednie magát, az adat ezek mögött található. Ez a modell igen biztonságos rendszerek kialakítását teszi lehetővé. Érdekes lenne elbeszélgetni valakivel, aki tört már fel plan9-et…

Parancsai teljesen egyediek, úgymond rettentő módon célorientáltak, például egy kép megjelenítésre az `image` parancs használható, természetesen a hozzá tartozó opciókkal. A képeken látható grafikus felület a *rio,* míg az ablakok elrendezéséért a *glenda* nevű program felel. Képernyőképek a 7. és 8. ábrákon. Található még benne pár örök klasszikus is a Unix hőskorából, ilyen a *sam* és a *pine.* Nem meglepő módon azonban e-mail küldésére/fogadására a `mail` parancs az alapértelmezett. A rendszer tehát felidézi a régi Unix hangulatát, de belülről már a mai gépekhez van idomítva. Gépigénye meglehetősen alacsony, és remekül képes alkalmazkodni a kapott hardverhez. Például 512 MB memória esetén közli, hogy 307 MB-ot használhatnak a felhasználók, a többi a kernel számára van fenntartva. A képernyőt 3D-gyorsítás nélkül, VESA és VGA módban képes kezelni (tökéletesen működött az nForce 3-as chipsettel szerelt laptopon, ezt a készletet alapból támogatja), és több hálózati kártyát ismer (a laptopban levő rtl8139-est rögtön detektálta).

A plan9 mindenkinek javasolható, aki nyitott az új iránt. A unixos örökség miatt javasolható az „öreg motorosoknak”, hiszen feltámad a múlt, a fiataloknak pedig azért, hogy egy kicsit nagyobb rálátásuk legyen a régmúltra, amelyből már kimaradtak. Ebbe a körbe tartozom jómagam is. Véleményem szerint oktatásban és a szakemberképzésben is nagy szerepe lehet, hiszen megtakaríthatók vele a kereskedelmi Unix-változatok súlyos licencköltségei.

2.5. OpenDOS

A DOS visszatér. Meglepő, hogy hány helyen használják még. Talán nem is hinnénk, hogy most, amikor a 3D-s ablakkezelők (*xgl, aero, metisse, aiglx, beryl)* lassan szinte felváltják a hagyományos 2D-s felületeket, még mindig van, aki DOS-t fejleszt, és „hivatalos” kiadások is készülnek. Az OpenDOS, a FreeDOS és a rá épülő GNU/DOS közül kezdjük az OpenDOS-szal. Sokáig a Caldera fejlesztette, de mára kicsit elöregedett, mivel a Caldera utódcégének, az SCO-nak a pereskedés mellett az OpenDOS-ra nem maradt ideje. 1999 óta nem jelent meg új verzió, és a termék már nem is elérhető a cég honlapjáról. (Megemlítjük, hogy vannak aktívan fejlesztett forkok, például [2]. E leágazás azonban felhasználóknak nem ajánlható, mert letölthető telepítőkészletet nem tartalmaz.) Ha az OpenDOS-t le akarjuk tölteni, kedvenc webkeresőnkhöz kell fordulnunk. A 7 MB-os csomag telepítése után egy „99 %-ban” MS-DOS-kompatibilis rendszert kapunk. Véleményem szerint az OpenDOS oktatási célokra éppoly tökéletes, mint bármely egyéb feladatra, ahol DOS-t használnak. Ám azt ne feledjük el sohasem, hogy régi fejlesztésről van szó, így lassan eljár fölötte az idő.

A 9. ábra egy szöveges módú képernyőképet mutat.

2.6. GNU/DOS és FreeDOS

A GNU/DOS neve – a GNU/Linuxhoz és a GNU/Hurdhöz hasonlóan – arra utal, hogy GNU felhasználói programok futnak DOS kernelen. Pontosabban: FreeDOS kernelen, ezért is tárgyaljuk e két szoftvert együtt.

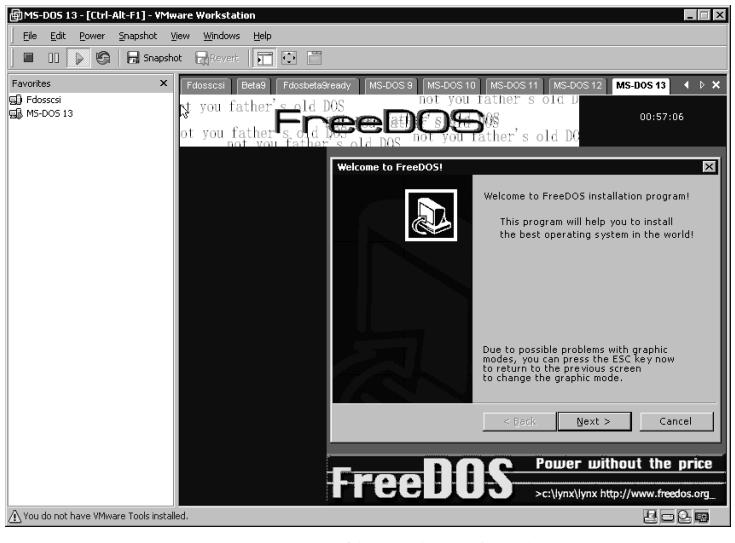
A GNU/DOS elképzelése az, hogy vegyünk egy minimális DOS alaprendszert (ahogyan a DOS-t elképzeljük), és portoljunk rá annyi GNU-s és GPL licencű programot, amennyit csak

```
C: DOSKEY   EXE : XCOPY    EXE : EXE2BIN  EXE : DISKOPT  EXE : XDEL     EXE
C: XDIR     EXE : EDIT     COM : FASTOPEN COM : FC       COM : FILELINK EXE
C: FIND     EXE : GRAFTABL COM : GRAPHICS COM : JOIN     EXE : LABEL    COM
C: MEM      EXE : MOVE     EXE : NETWARS  EXE : NWDRAW   EXE : NWDRAW   HLP
C: NETWARS  ICO : README   NW  : NWCACHE  EXE : NWCACHE  OV1 : NWCACHE  OV2
C: DBLBUF   SYS : PASSWORD EXE : LOCK     EXE : DELWATCH EXE : DELPURGE EXE
C: DISKMAP  EXE : UNDELETE EXE : RECOVER  COM : BACKUP   COM : RESTORE  COM
C: PRINT    COM : RENDIR   EXE : REPLACE  EXE : SCRIPT   EXE : SHARE    EXE
C: SORT     EXE : SUBST    EXE : TOUCH    EXE : TREE     COM : VDISK    SYS
C: UNINSTAL EXE : SERNO    EXE : MEMMAX   COM : EMMXMA   SYS : CREATE   COM
C: VIPX     386 : DCONVERT COM : DCONVRT2 EXE : PREVIEW  EXE : SCONVERT EXE
C: SDEFRAG  COM : SCREATE  SYS : SCREXEC  EXE : UNSTACK  COM : STACFM   DLL
C: STACFM   HLP : STACHIGH SYS : STACLOAD BIN : SDEFRAG2 EXE : STACKER  EXE
C: STACKER  COM : UNSTACK2 EXE : WINSWAP2 EXE : DEVSWAP  COM : NWCDEX   EXE
C: EMM386   EXE : DPMI     EXE : TASKMGR  EXE : RPLDOC   TXT : DPMS     EXE
C: README   TXT : OPENDOS  INI : TASKMGR  INI : SETUP    BAK : SETUP    INI
      125 File(s)2054422528 bytes free

OPENDOS C:\OPENDOS>ver

Caldera OpenDOS 7.01
Copyright (c) 1976, 1997 Caldera, Inc.
All rights reserved.

OPENDOS C:\OPENDOS>
```

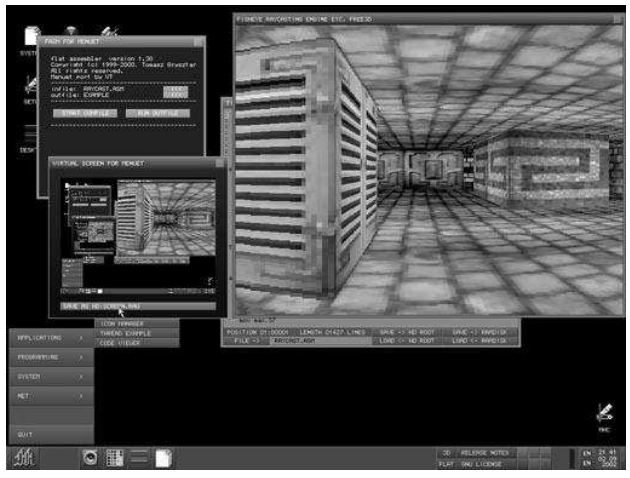
9. ábra. Képernyőkép az OpenDOS-ról

```
FreeDOS Beta9 ("Methusalem") INSTALL

  1 - Boot with (IDE/ATAPI) CD-ROM driver (default)
  2 - Boot with (IDE/ATAPI) CD-ROM driver and support XMS (386+)
  3 - Clean Boot (No CD-ROM driver loaded)

  4 - LSI/NCR/SYMBIOS-chip based SCSI CDROM install
      \DRIVER\ASPI8XX.SYS
      \DRIVER\SYMDISK.SYS
      \DRIVER\SYMCD.SYS /D:FDCD0001



      (press 0 to run Memtest86 - physical memory test program)

      FreeDOS is a trademark of Jim Hall 1994-2003


Singlestepping (F8) is :OFF - please select a Menu[012345] (default=1) - 50
```

10. ábra. A FreeDOS karakteres telepítőjének rendszerindító menüje



11. ábra. A FreeDOS grafikus telepítője – képernyőkép



12. ábra. A MenuetOS egy labirintusdemót futtat – képernyőkép

lehet. Ennek eredményeképpen kapunk egy olyan telepített DOS-t, amely tartalmazza a Bash, az Apache és az XFree86 programokat is, sok kisebb, a GNU rendszerek alatt megszokott kiegészítővel egyetemben. Természetesen a forráskódot is mellékelik a CD-n, de létezik mini CD is, amely valamivel több, mint 50 MB-nyi területen csak a fontos bináris programokat és a telepítőt tartalmazza. Egyébként hosszú évek fejlesztése után idén, 2006-ban jelent meg az 1.0-s verzió. A GNU/DOS – bár neve alapján egyesek ezt nem várnák – kompatibilis az MS-DOS rendszerrel, a fejlesztők ígérete szerint 100 %-ban.

Fontos tudni, hogy a telepítő nem tud a CD-ről elindulni, hanem egy hajlékonylemezről kell indítani a gépet, becsatolni a CD-t és úgy telepíteni a rendszert. Remélhetőleg a fejlesztők ezt a hibát hamarosan javítják, hiszen korunkban szinte teljesen kihalt a floppy. Ritka a floppymeghajtó, és még ritkább, hogy valakinek lenne a keze ügyében hajlékonylemez. A fejlesztők érvelhetnének azzal, hogy aki GNU/DOS-t szeretne telepíteni, az valószínűleg régi, elavult gépre tenné, amiben valószínűleg van még floppymeghajtó. Aki GRUB bootmenedzsert használ, megpróbálhatja a telepítő floppy image-et a SYSLINUX részét képező MEMDISK-kel bebootolni.

A FreeDOS más szemléletet követ. Először is nem csak a régi 486-os gépekre gondol. Telepíthető közvetlenül CD-ről is, és elsősorban nem a GNU programokra építkezik, hanem a saját kis DOS-os programjaira, bár természetesen több GNU-fejlesztés is megtalálható benne.

A FreeDOS-ban található GNU-fejlesztésekre jó példa a Seal2 GUI. A fejlesztők igyekeznek egy emberbarát, igen jól felszerelt és könnyen kezelhető DOS-t létrehozni. Ennek tudható be, hogy habár a telepítés karakteres felületen kezdődik (10. ábra), hamarosan átvált a VGA üzemmódú grafikus telepítőbe (11. ábra). Mint említettük, grafikus felületet is tudunk telepíteni. Rendszerigénye a karakteres módban olyan, mint az OpenDOS-é és GNU/DOS-é: egy 486-os gép és 4 MB (!) memória általában elég neki. Egyes felhasználói programok 8 MB memóriát igényelnek, és ha a Seal2 grafikus környezetet szeretnénk használni, akkor bizony a 16 MB is elkél. Véleményem szerint egy 4 MB memóriával rendelkező 486-os gépre a legjobb választás az OpenDOS az Arachne Desktop grafikus felülettel.

2.7. MenuetOS

Végére maradt egy igazi gyöngyszem. Aki elmélyülten foglalkozott már assembly-programozással, vajon gondolt-e már arra, hogy írjon egy operációs rendszert assembly nyelven? Nem? Nos, a MenuetOS fejlesztői igen. Joggal merül fel a kérdés, hogy mit tudhat egy ilyen rendszer? A floppys verzió nem túl sokat: 12-féle médiaformátumot ismerő lejátszó, mini as-

13. ábra. A MenuetOS távol-keleti menüjének képernyőképe

sembler, sima szövegszerkesztő és néhány beállítási lehetőség, mindez grafikus felületen. A merevlemezre telepített CD-s változaton viszont már a Quake 1 is fut (12). Nem csak az angol nyelvet ismeri, a menüt is átállíthatjuk távol-keletire (13. ábra). Van benne CD-lejátszó és jó teljesítményű MP3-lejátszó (nem lehetett könnyű az egészet assemblyben megírni...). Úgy találtam, hogy a MenuetOS lényegesen „okosabb”, mint a már ismertetett Visopsys. Erőforrásigényével a középmezőnyben foglal helyet: 300–400 MHz-es Pentiumon már tökéletesen fut, és elkél a 64 MB rendszermemória. Sajnos nagyon érzékeny a hardverre, ne számítsunk rá, hogy minden eszközünket felismeri.

Sok szolgáltatást kínál, de messze nem annyit, mint szeretnénk. A MenuetOS funkcióinál sokkal érdekesebb a forrása: ez már egy olyan komplex rendszer, melynek assemblyben történő megírása a lehetetlen határát súrolja, van tehát mit tanulni a forrásból, ha assemblyprogramozásra adjuk a fejünket. Sajnos a 64 bites verzió forrása nem hozzáférhető, ebből kereskedelmi termék készül. A 32 bites alkalmazások közül sem mindegyiknek tölthető le a forrása.

3. Zárszó

Jelenleg a nyílt forrású operációs rendszerek kínálata rendkívül bő, e cikkben csak ízelítőt adtunk belőle. Említhetnénk még a beágyazott rendszereket és a tárgyalt rendszerek kisebb mellékágait is. Vannak évek óta stagnáló projektek is, pl. a AtheOS, a Blue Illusion és a plan9 mintájára létrehozott Inferno. A zárt forrású alternatív operációs rendszerek közül nagy kort ért meg a valaha nyílt SkyOS és a borsos árú, valósidejű QNX rendszer. Zárt rendszereken is megfigyelhető a nyílt forrású alkalmazások terjedése: a legtöbb rendszerre van Mozilla, Firefox és GIMP. A QNX-közösség például folyamatosan karbantartja a Linux-világból ismert szoftverek (Apache, X11-alkalmazások, ablakkezelők, böngészők) QNX-es portját.

Hivatkozások

[1] BeOS-os szoftverek letöltési helye. URL `http://www.bebits.com/`.

[2] The DR-DOS/OpenDOS enhancement project (egy szabad OpenDOS-leágazás). URL `http://drdosprojects.de/`.

Vékonykliensek használata az LTSP segítségével

Gergi Miklós
<mgergi@math.bme.hu>

Budapesti Műszaki és Gazdaságtudományi Egyetem
Matematika Intézet

Kivonat

Mit kezdjünk az elavult, régi számítógépeinkkel? Hogyan csináljunk magunknak csendes számítógépes munkahelyet? Mindkét kérdésre jó válasz: használjunk vékonyklienseket.

A BME Matematika Intézetében 2004 őszén határoztuk el, hogy géptermeink felújításánál vékonykliens megoldást fogunk használni. Jelenleg mindkét géptermünkben teljesen csendes, mozgó alkatrészek nélküli gépek vannak elhelyezve, a felhasználók egyetlen gombnyomással, újraindítás nélkül választhatnak, hogy a Linuxot vagy Windows 2003-at futtató terminál-szerverünket szeretnének használni. Mindkét rendszer alatt lehetőségük nyílik USB-portos adathordozók használatára, és akár zenét is hallgathatnak. A géptermek kialakítása mellett egyre több kiöregedett számítógépet is kliensként használunk.

A cikkem a rendszer kialakítását, a felmerült buktatókat és az ezekre talált megoldásokat mutatja be.

Tartalomjegyzék

1. Bevezetés

A BME Matematika Intézetében 2004 végén döntöttünk arról, hogy a hallgatói géptermeinket felújítjuk. Ennek során jelentős szempont volt, hogy minél értékállóbb megoldást találjunk, így mozgó alkatrész nélküli vékonykliens gépek beszerzése mellett döntöttünk. A mozgó alkatrészek hiánya várhatóan csökkenti a meghibásodások gyakoriságát, és további előny, hogy a géptermek teljesen csendessé váltak, és a hőtermelés is „elviselhetővé" vált (korábban a felhasználókat a nyári időszakban igencsak megviselte az a néhány plusz °C, amennyivel a gépteremben melegebb volt a többi helyiséghez képest).

Kulcsrakész vékonykliens megoldásokat sok cég kínál, hogy mi végül miért az LTSP rendszert választottuk, annak a legfőbb okai a következők:

1. Lehetőség van más beszerzésből származó, más gyártótól származó gépekkel bővíteni a kliensek körét, így gyártófüggetlenné válunk.
2. A régi, kiöregedett gépek is klienssé alakíthatóak.
3. Tud kapcsolódni Linuxhoz és Windowshoz is, egyetlen billentyűkombinációval, egy másodperc alatt válthatunk a két operációs rendszer között.
4. Csak a saját tudásunk szab határt a megvalósítható lehetőségeknek.

Természetesen hátránya is van ennek a megoldásnak:

1. Nem egy kész, működő megoldást kapunk, hanem magunknak kell létrehoznunk a kívánt rendszert.
2. Ha nem üzemel, nincs hol reklamálni, legfeljebb a fórumokon kereshetünk segítséget.
3. A dokumentáció rosszul karbantartott, hiányos, nem követi a frissülést. Sok esetben nem lehet tudni, hogy amit olvasunk, az éppen melyik verzióra igaz, és melyikre nem.

Az LTSP (Linux Terminal Server Project) egy Linux alá fejlesztett programcsomag, amit arra találtak ki, hogy megkönnyítse a kisteljesítményű számítógépek, vékonykliensek csatlakoztatását Linuxot és/vagy Windowst futtató terminál szerver gépekhez. A projekt fejlesztését 1999-ben kezdte Jim McQuillan és Ron Colcernian. Az aktuális verzió, valamint dokumentációk, wiki stb. elérhetőek a projekt honlapján [2].

2. Telepítés

2.1. Hardver- és szoftverigény

Kliens-hardver ♦ A fejlesztők szerint már egy 16 MB memóriával rendelkező 66 MHz-es 486-os gép is elegendően erős ahhoz, hogy kliensgépként használjuk, 24 MB memóriánál és P133 processzornál jobbra pedig nincs is szükség. Processzor szempontjából ez valóban így is van, viszont ha egy gépet egyidejűleg Linux és Windows kliensként is akarjuk használni, hanggal, nyomtatóval, és ha nem akarunk swapot használni (amihez használhatunk helyi merevlemezt, de lehetőség van hálózaton keresztüli swapolásre is), akkor szükség lehet 64 MB memóriára is.

A vásárlás előtt több gyártó többféle termékét kipróbáltuk, és még a kapható leggyengébb modellek is elegendőnek bizonyultak. Problémák csak az az AMD Geode processzorral szerelt kliensgépeknél merültek fel, ahol a legfrissebb X.Org nem támogatja az integrált monitorvezérlőt, a gyártó pedig csak egyetlen, régebbi X.Org verzióhoz adott ki patchet [1].

Eddig csak x86 architektúrájú gépeket teszteltünk kliensként. Néhány további architektúrához található ugyan régebbi LTSP-változat, például PPC-hez vagy SPARC-hoz [3], a több vékonykliens által is használt AMD Alchemy processzorcsaládot (MIPS architektúra) az LTSP még nem támogatja.

Szerver-szoftver ♦ Ha helyi tároló nélküli *(diskless)* klienseket akarunk létrehozni, akkor szükség van a PXE (Preboot Execution Environment) bootoláshoz egy DHCP- és egy TFTP-szerverre. Debian Linux alatt több TFTP szervercsomagot találunk, sajnos pont az alapértelmezett *tftpd* csomag nem megfelelő az LTSP számára, helyette a *tftpd-hpa* csomagot kell használnunk. A *tftpd-hpa* tud futni daemonként, vagy meghívhatja az inetd. Mivel viszonylag ritkán van szükség a futására, ezért használhatjuk nyugodtan az inetd által meghívva. Telepítenünk kell egy nfs-szervert is, erre a célra mi az *nfs-kernel-server* csomagot alkalmazzuk. Kell még egy X Display Manager, amit tetszőlegesen választhatunk (*gdm*, *kdm*, *wdm*, *xdm*...). Szükség lehet még egy fontszerverre, különösen, ha olyan alkalmazásaink is vannak, amelyek saját fontokat használnak, mint pl. a Mathematica. Erre a célra tökéletesen megfelel az *xfs* csomag. (Megjegyezzük, hogy az újabb, Xft-s fontkezelést alkalmazó szoftvereknek, pl. Mozilla, GIMP, Gnome, KDE, nincs szükségük fontszerverre.) Ha hangokat is szeretnénk hallani a kliensünkön, akkor a terminálszerverünkre az *esound-clients* csomagot telepítsük.

Szerver-hardver ♦ Alapértelmezett telepítés során elegendő egy darab Linux rendszert futtató gép, ami egyrészt ellátja a kliensgépek működtetéséhez szükséges feladatokat, valamint terminálszerverként üzemel, vagyis valójában erre jelentkeznek be a felhasználók, ezen futtatják a programjaikat. Ennek méretezésekor gondolnunk kell arra, hogy ezt a gépet egy időben több felhasználó is használja, így ennek megfelelő mennyiségű memóriával lássuk el. Hasonló okokból többprocesszoros gép ajánlott.

A felsorolt feladatokat azonban érdemes két gépre szétbontani, amiből az egyik egy kisteljesítményű gép, és pusztán a kliensek működtetéséhez szükséges feladatokat (DHCP, TFTP, NFS, syslog, swap) látja el, a másik gép pedig csak terminálszerverként működik. Ebben az esetben a linuxos terminálszerver meghibásodása esetén még mindig tudjuk a kliensgépeinket Windows-kliensként használni. Sőt, mivel az első szervergépünk egy egyszerű, olcsó gép, így kevés ráfordítással tudunk belőle HA-klasztert építeni, ami tovább javítja a vékonykliensrendszerünk megbízhatóságát.

A Linuxos szervereink mellé beállíthatunk egy Windows 2003 vagy Windows 2000 szervert is, amihez szintén tudnak csatlakozni a klienseink. (Windows XP azért nem megfelelő, mert az – licenc okokból – egyidejűleg csak egy aktív felhasználót engedélyez.)

2.2. Telepítés az `ltspadmin` és a `ltspcfg` segítségével

Ha az XDM-szerver, és a klienseket egyéb szempontból kiszolgáló (DHCP, TFTP, NFS) szerver ugyanaz a gép, akkor egy kényelmes telepítési mód az *ltsp-utils* csomagban található `ltspadmin` parancs használata, ami letölti és kicsomagolja az LTSP rendszer fájljait, majd az `ltspcfg` programot meghívva annak bekonfigurálásában is segít. Fontos, hogy mindig a legfrissebb *ltsp-utils* csomagot használjuk! Ha a disztribúciónk által felkínált csomag nem a legfrissebb, akkor töltsük le az új verziót az LTSP honlapjáról!

Telepítsük fel csomagkezelőnkkel a szükséges programokat (*dhcpd*, *tftpd-hpa*, *nfs-kernel-szerver*).

Az `ltspadmin` parancsot elindítva a menüből válasszuk ki a *Configure the installer options* pontot, ahol megadhatjuk, hogy honnan kívánjuk letölteni az ltsp csomagokat, melyik könyvtárat akarjuk a kliensek gyökérkönyvtáraként használni, és beállíthatjuk a proxynkat is, ha van.

Ezt követően indítsuk el az *Install/Update LTSP Packages* menüpontot, amiben kiválaszthatjuk, hogy az LTSP mely komponenseit telepítsük. Általában az *ltsp_core*, *ltsp_kernel*, *ltsp_libusb*, *ltsp_localdevs*, *ltsp_rdesktop*, *ltsp_x_core* komponensekre lesz szükségünk. Ha nem használunk fontszervert, akkor az *ltsp_x_addtl_fonts*-ot is érdemes telepíteni. Jó tudni, hogy *xfs* konfigurálásában az *ltspcfg* nem nyújt segítséget!

Végül indítsuk el a *Configure LTSP* menüpontot. Ekkor az *ltspcfg* leteszteli, hogy fut-e az összes szükséges szolgáltatás, majd a *Configure the services manually* pont alatt egyenként beállíthatjuk ezeket.

2.3. Telepítés kézzel

Ha kézi telepítést választunk, akkor egyenként kell a különböző konfigurációs fájlokat megszerkesztenünk. Ez egy kicsit több tapasztalatot igényel, mint az *ltspcfg* használata, de nagyobb rálátást nyújt a rendszer működésére, és finomabb beállításokra ad lehetőséget.

DHCP ♦ A DHCP daemon konfigurációs fájljában, a `/etc/default/dhcpd` fájlban tudjuk megadni, hogy melyik hálózati interfészen fusson a DHCP szolgáltatás. Ezután pedig a `/etc/dhcpd.conf` fájlt kell létrehoznunk. Ahhoz, hogy a klienseink távolról is egyértelműen azonosíthatóak legyenek, fontos, hogy fix IP-címet kapjanak. Egy jó konfigurációs fájl például a következő:

```
option subnet-mask         255.255.255.0;
option broadcast-address   192.168.1.255;
option routers             192.168.1.1;
option domain-name-servers 192.168.1.1;
option domain-name         "ltsp";
allow bootp;
allow booting;
use-host-decl-names         on;
subnet 192.168.1.0 netmask 255.255.255.0 {
  range 192.168.1.200 192.168.1.250;
  next-server               192.168.1.1;
  filename                  "lts/2.6.17.8-ltsp-1/pxelinux.0"
  option root-path          "192.168.1.1:/usr/local/ltsp/i386"
}
host labor1 {
  hardware ethernet 00:40:63:DA:C3:76;
  fixed-address 192.168.1.11;
}
host labor2 {
  hardware ethernet 00:40:63:DA:C3:76;
  fixed-address 192.168.1.12;
}
```

Az első sorokban beállítjuk a hálózat alapvető paramétereit. A `bootp` és a `booting` a hálózatról való bootoláshoz szükséges jogosultságok. A mostani DHCP-szerverekben alapértelmezésben engedélyezve vannak, de a régi rossz tapasztalatok miatt jobb határozottan engedélyezni. A `use-host-decl-names` bekapcsolása azért fontos, mert így a fix IP-című kliensgépek a nevüket is megkapják DHCP-n keresztül.

Létrehozunk egy subnetet, amiben a 192.168.1.200 és a 192.168.1.250 tartományban dinamikusan osztunk IP-címeket. Az új, fix IP-címet még nem kapó kliensek ebből a tartományból fognak IP-t kapni. A `next-server` a TFTP-szerver címét addja meg, a `filename` pedig azt, hogy mit is kell arról a gépről bebootolni. A `root-path` paraméterben pedig az nfs-en exportált gyökérkönyvtár címét állítjuk be.

TFTP ♦ Mivel a tftpd-t az inetd indítja, ezért a beállítását a `/etc/inetd.conf` fájlban lehet elvégezni az alábbi sorral:

```
tftp dgram udp wait root /usr/sbin/in.tftpd /usr/sbin/in.tftpd -s /tftpboot
```

NFS ♦ Az NFS-szerver konfigurációját a `/etc/exports` fájlban módosíthatjuk. Mindössze egyetlen sorra van szükség benne, mit osztunk meg, melyik gépeknek, és milyen paraméterekkel:

```
/usr/local/ltsp 192.168.1.1/255.255.255.0(ro,no_root_squash,sync)
```

XDM ♦ Ha GDM-et használunk,akkor a `/etc/gdm/gdm.conf` fájlban kell megkeresnünk az `[xdmcp]` fejezetet és ott:

```
[xdmcp]
Enable=true
MaxSessions=50
```

Az utóbbi sorral az egyidejűleg engedélyezett kapcsolatok számát növeljük meg, ha az kevesebb a szükségesnél.

Ötödik lépésként konfiguráljuk a fontszerverünket. Ennek a beállításai a `/etc/X11/fs/config` fájlban találhatóak. Itt a legfontosabb feladatunk a TCP porton való hallgatózás tiltásának megszüntetése, valamint a fontokat tartalmazó könyvtárak felsorolása:

```
#no-listen = tcp
catalogue = /usr/lib/X11/fonts/misc/,
     /usr/lib/X11/fonts/100dpi/:unscaled,
     /usr/lib/X11/fonts/75dpi/:unscaled,
     /usr/lib/X11/fonts/Type1/,
     /usr/lib/X11/fonts/Speedo/,
     /usr/lib/X11/fonts/100dpi/,
     /usr/lib/X11/fonts/75dpi/,
     /usr/local/Wolfram/Mathematica/5.2/SystemFiles/Fonts/AFM/,
     /usr/local/Wolfram/Mathematica/5.2/SystemFiles/Fonts/BDF/,
     /usr/local/Wolfram/Mathematica/5.2/SystemFiles/Fonts/Type1/
```

Végül vegyük fel a kliensgépeinket a `/etc/hosts` fájlba, és az *ltspadmin* programot használva az előző fejezetben leírt módon töltsük le és csomagoljuk ki az LTSP szükséges komponenseit a `/usr/local/ltsp` könyvtárba. Ellenőrizzük, hogy a kernel, az initramfs, és minden a hálózatról bootoláshoz szükséges fájl az `/tftpboot/lts` könyvtárban, a megfelelő helyen található.

2.4. Konfigurálás

A kliensek konfigurálása a NFS-en exportált gyökérkönvtárban levő `/etc/lts.conf` szerkesztésével történik. A konfigurációs fájlban megadhatunk globális, minden kliensre vonatkozó beállításokat, mint például a terminálszerverek, vagy a fontszerver IP-címe, és megadhatunk kliensfüggő beállításokat, mint pl. a monitor felbontása, vagy a klienshez csatlakozó egér típusa.

A kliensgépeket nevük, IP-címük, vagy MAC-címük alapján azonosíthatjuk. Egy egyszerű konfigurációs fájl az alábbi módon néz ki:

```
[Default]
  SERVER             = 192.168.1.1
  RDP_SERVER         = 192.168.1.2
  XSERVER            = auto
  USE_XFS            = Y
  X_MODE_0           = 1280x1024
  X_MOUSE_PROTOCOL   = ImPS/2
  SCREEN_01          = telnet
```

```
  SCREEN_02          = telnet
  SCREEN_03          = telnet
  SCREEN_04          = telnet
  SCREEN_05          = telnet
  SCREEN_06          = rdesktop
  SCREEN_07          = startx
[192.168.1.18]
  XSERVER            = ati
  X_HORZSYNC         = 30-83
  X_VERTREFRESH      = 56-75
  X_MODE_0           = 1024x768
  SCREEN_01          = shell
```

Értelmezzük ezt a konfigurációt! Az alapértelmezett szervergép minden kliens számára a 192.168.1.1. Ezt a gépet használják XDM-szervernek, fontszervenek, NBD-szervernek, erre küldik a syslogot, és a telnetek is ehhez a géphez csatlakoznak. Mivel az RDP (Windows terminál-) szervert külön megadtuk, így az rdesktopok a 192.168.1.2 géphez próbálnak csatlakozni.

Az `XSERVER=auto` beállítással megadtuk, hogy az induló X automatikusan próbálja meg eldönteni, melyik driver való a kliens monitorvezérlő kártyájához. Ez általában jól működik, de ha valamelyik kártyánál mégis hibásan dönt, akkor megadhatjuk közvetlenül a driver típusát, mint ahogy ezt a 192.168.1.18 gépnél tettük. A `USE_XFS=Y` jelzi, hogy szeretnénk fontszervert használni. Az `X_MODE_0` értéke adja meg, hogy mi a monitor elsődleges felbontása. Ha valami szokatlan felbontást szeretnénk használni, akkor az `X_MODE_0` értékének megadhatunk egy teljes `ModeLine` sort is. További felbontásokat a `X_MODE_1` és az `X_MODE_2` változókkal definiálhatunk. Mint a 192.186.1.18 gépnél láthatjuk, ha gondban vagyunk a monitor beállításával, akkor a vertikális és horizontális szinkron frekvenciákat is beállíthatjuk. Az `X_MOUSE_PROTOCOL` az egér típusát határozza meg az X konfigurációjához. Az ImPS/2 a legtöbb mai kétgombos–egygörgős egérnek megfelel.

A beállítások talán leglényegesebb része a `SCREEN` változók megadása. Ezek a változók határozzák meg, hogy melyik virtuális terminálon milyen program fog elindulni. Négy lehetőség közül választhatunk mindegyik screen esetében.

shell ♦ Az adott virtuális terminálon egy root-shell fog elindulni. Általában hibakeresési célra használjuk, és csak egy-két kliensgépen engedélyezzük.

telnet ♦ Ez egy telnet kapcsolatot nyit a `TELNET_HOST` változóban, vagy annak hiányában a `SERVER` változóban megadott gépre.

startx ♦ Csatlakozik az `XDM_SERVER` változóban, vagy annak hiányában a `SERVER` változóban megadott Linux terminálszerverhez.

rdesktop ♦ Csatlakozik az `RDP_SERVER` változóban, vagy annak hiányában a `SERVER` változóban megadott Windows terminálszerverhez.

A `/etc/lts.conf.readme` fájlban további konfigurációs lehetőségeket is találunk, azonban ez a fájl nem követi az LTSP fejlődését, így egyrészt vannak benne időközben megszűnt konfigurációs változók, és hiányoznak bizonyos újabb lehetőségek. Ahhoz, hogy megismerjük az összes kínálkozó lehetőséget, talán a leghatásosabb módszer az, ha végignézzük, hogyan is indul egy a kliens, és az indítófájlokban megtaláljuk a „titkos” változókat.

3. Hogyan működik?

1. A kliens TFTP-n keresztül megkapja a kernelt és a kezdeti ramdisket (initrd). (Ha a hálókártya nem támogatja a hálózatrol bootolást, akkor a kernelt és az initrd-t elhelyez-

hetjük egy lokális meghajtón is, és valamilyen bootmanagerrel betölthetjük.) Ezekkel elindul a bootolás. Gyökérkönyvtárként az NFS-en exportált könyvtárrendszert csatolja fel.

2. Lefut az initial ramdiszken levő `init` fájl. Ez többek között létrehoz egy ramdiszket, ami a továbbiakban a rendszer gyökérkönyvtára lesz, ennek a `/nfsroot` könyvtárába felcsatolja az exportált könyvtárrendszert, majd linkeket csinál a legfontosabb könyvtárakra.

3. Lefut a `/etc/rc.early_sysinit`, ami felcsatolja a `/proc` és a `/sys` könyvtárakat, elindítja a *udev* daemont, és létrehozza az `lts.conf` alapján az `/etc/inittab` fájlt.

4. Elindul az init program. Az init indítja a `/etc/rc.sysinit` szkriptet.

5. A `/etc/rc.sysinit` az `lts.conf`-ban beállítottak alapján elindítja a hálózaton keresztüli swapolást, betölti a szükséges kernelmodulokat, létrehoz sok szükséges könyvtárat, elindítja a syslogot, az xinetd-t, végrehajtja a `/etc/rc.local` fájlt, elindítja az *snmp, sound,* és *localdev* szkripteket.

6. Az init elindítja az ltspinfod daemont, a screeneket, és nyomtatószervert. Ha ezek közül bármelyik leáll, azt azonnal újraindítja.

4. Extrák

4.1. Billentyűzet

Furcsának tűnhet, hogy a billentyűzetet az extra opciók között látjuk, de ha Windows Terminal Serverhez is szeretnénk csatlakoztatni a klienseinket, akkor sajnos billentyűzet területén is van tennivalónk. Az LTSP rendszerben lévő rdesktop nem támogatja ugyanis a magyar billentyűzetkiosztásnál szinte elengedhetetlenül szükséges AltGr módosító billentyűt, és ezt a problémát sajnos nem is tudjuk pusztán konfigurálással megoldani. (Pontosabban: az eredeti rdesktop azt feltételezi, hogy Windows alatt menet közben nem váltunk billentyűkiosztást, és emiatt nem állítható be úgy, hogy a billentyűzet-események AltGr módosítóbitjeit váltakozó magyar és angol kiosztás mellett is megfelelően továbbküldje. A felhasználó viszont elvárja, hogy tudjon kiosztást váltani, és emiatt ne kelljen ki-bejelentkeznie.) Egy Szabó Péter által írt folt alkalmazásával újra kell fordítanunk az rdesktopot, és az új rdesktop csomaggal már lehetőség van a magyar billentyűzet használatára is [6]. A részletes megoldás megtalálható a 6.2. szakaszban a 31. oldalon.

4.2. Nyomtató

A klienshez csatlakoztatott nyomtatókat hálózati nyomtatóként tudjuk használni a terminálszervereinkről, vagy egyéb gépekről. Ennek a beüzemeléséhez az `lts.conf` fájlt kell módosítanunk.

```
PRINTER_0_DEVICE = /dev/lp0
PRINTER_0_TYPE = P
PRINTER_0_PORT = 9100
PRINTER_1_DEVICE = /dev/usb/lp0
PRINTER_1_TYPE = U
PRINTER_1_PORT = 9101
```

A fenti példában két nyomtatót konfiguráltunk be a kliensen, a printer_0 egy párhuzamos portra kötött (`TYPE=P`) nyomtató, amit a 9100-as tcp porton keresztül lehet elérni, a második nyomtatónk pedig egy USB porton csatlatozó (`TYPE=U`) nyomtató, amit a 9101-es tcp porton tettünk elérhetővé.

Ha egy nyomtatót megosztunk a fent leírt módon a 192.168.1.28 gép 9100-as portján, és azt CUPS-on keresztül szeretnénk elérni, akkor a Device URI megadásakor a `socket://192.168.1.28:9100` értéket kell beállítani.

4.3. Hang

Az LTSP 4.2-es verziójában áttértek a 2.6-os kernelre, aminek következtében az eddigi OSD hangrendszer helyett az ALSA hangrendszert támogatja a kernel. Nem kerültek bele viszont az LTSP-be az ehhez szükséges programok, (alsa-utils, alsamixer, stb...), így ezeket külön kell telepítenünk a klienseket kiszolgáló nfs-szerveren. Ehhez az *ltsp-esd-alsa* csomagra lesz szükségünk, amit a [4] címről lehet letölteni. A letöltött csomagot kitömörítve az install paranccsal telepíthetjük fel.

Ezek után az `lts.conf` fájlban a megfelelő klienshez a `SOUND=Y` bejegyzést kell beírni.

A Windows 2003 szerver alapbeállításban tiltja a hang átirányítását a kliensre. A tiltást a Microsft Management Console (mmc.exe) segítségével kapcsolhatjuk ki a biztonsági házirend megfelelő beállításával *(Számítógép konfigurációja | Felügyeleti sablonok | Windows-összetevők | Terminálszolgáltatások | Ügyfél/kiszolgáló adatai átirányításának tiltása | Hangátirányítások engedélyezése)* [8].

Az rdesktop a `-r sound:local` kapcsoló megadásával veszi át a hangot a terminálszervertől. Ezt az `lts.conf` fájlban az `RDP_OPTIONS` változóban adhatjuk meg.

A Windows hangerőszabályzója nem működik az rdesktopon keresztül, ezért érdemes egy külön screent létrehozni, amiben az *alsamixer*t futtatjuk.

4.4. USB-tárolók

Internet-hozzáféréssel nem rendelkező kollégák számára igen fontos dolog, hogy az eléjük rakott kliensgép alkalmas legyen valamilyen módon az otthonról hozott állományok feltöltésére, és a terminálszerveren létrehozott állományok lementésére. Ennek megoldására az LTSP fejlődése során rengeteg ötlet és próbálkozás volt (például NFS-en vagy Samba-n megosztották az automatikusan felcsatolt adathordozókat). Az LTSP jelenlegi megoldása egy saját hálózati fájlrendszer, az ltspfs, és hozzá kapcsolódó L-BUS rendszer.

Az *ltspfsd* (LTSP FileSystem Daemon) a kliens gépen fut, és annak könyvtárait elérhetővé teszi a terminálszerver számára. Az terminálszerveren az *ltspfs* paranccsal csatolhatjuk fel a klienshez csatlakoztatott adathordozót. Az *ltspfsd* igyekszik a helyi adattárolókat lecsatolt állapotban tartani. Ha 2 másodpercen keresztül semmiféle fájlművelet nem történik, akkor az adathordozót leválasztja, majd ha újabb fájlműveletre kap utasítást, akkor újra mountolja. Ezzel megkönnyíti a behelyezett írható adathordozók eltávolítását, hiszen a fájlműveletek idejét kivéve többnyire szinkronban van a fájlrendszer, sőt fel sincsen csatolva.

Az L-BUS rendszer két daemonból áll, az egyik a kliensen futó *lbuscd* (L-BUS Client Daemon), a másik a terminálszerveren a felhasználó nevében futó *lbussd* (L-BUS Server Daemon). Egy új eszköz csatlakoztatásakor a *lbuscd* jelez az *lbussd*-nek, ami elindítja a `lbus_event_handler.sh` szkriptet. Ez a szkript létrehoz a felhasználó home könyvtárán belül egy könyvtárat az új adathordozónak, és az *ltspfs* segítségével oda felcsatolja. Ha az ablakkezelő támogatja, akkor még egy ikont is feldob az asztalra. Az adathordozó eltávolításánál is ugyanilyen módon hajtódik végre az umount, a célkönyvtár törlése és az ikon eltávolítása.

Mit kell tennünk, hogy mindez működjön?

Először is kapcsoljuk be a helyi adathordozók támogatását a klienseken. Ehhez a `/etc/lts.conf` fájlba kell beírnunk a `LOCAL_DEVICES=Y` sort. Ellenőrizzük, hogy a kliensen a `hostname` parancsot kiadva ugyanazt a nevet kapjuk, mint amit terminálszerverre a kliensről belépve a `$DISPLAY` változóban találunk. Ha a két név nem egyezik, akkor nézzük át a `/etc/hosts` fájlt és ellenőrizzük a dhcpd konfigurációját, hogy átadja-e a klienseknek a hosztnevet.

Ha ezzel megvagyunk, akkor a kliensünk már készen is van. A terminálszerveren szükségünk van a kernelben a FUSE támogatására, és szükségünk van a *fuse-utils* csomagra. Adjuk hozzá a felhasználóinkat a *fuse* csoporthoz. Telepítsük a *libx11-protocol-perl* csomagot, amire a *lbussd*-nek van szüksége, majd töltsük le és telepítsük az *ltsp-localdev-support* csomagot, amit az [5] címen találunk *ltsp-server-pkg* néven. Ez a csomag tartalmazza az *lbussd*-t, és az ltspfs-t.[1] Végül ellenőrizzük, hogy a `/etc/X11/Xsession.d` könyvtárban létrejött-e a `51lbus-start` fájl, ami arra szolgál, hogy bejelentkezéskor automatikusan elindítsa a felhasználónak az *lbussd* daemont. Ha ez a fájl hiányzik, mert például csak felmásoltuk a csomagot, akkor az `xsession-lbus-start` állományt linkeljük be ezen a néven.

4.5. Monitor lekapcsolása

A kliensgépek, különösen a kis fogyasztású modern vékonykliensek általában napi 24 órában futnak, viszont ennek jelentős részében üresjáratban vannak. Jogos elvárás tehát, hogy a monitor energiagazdálkodási lehetőségeit kihasználják. Ennek beállításához ismét az `lts.conf` fájlt kell szerkesztenünk:

```
X_DPMS = Y
X_DPMS_STANDBYTIME = 3
X_DPMS_SUSPENDTIME = 6
X_DPMS_OFFTIME = 10
```

Ezekkel a beállításokkal, ha a klienst nem használjuk, a monitor 3 perc után standby, 6 perc után suspend módba kapcsol, majd további 4 perc múlva kikapcsolja magát.

4.6. Saját screen típusok kialakítása

Az LTSP négy beépített screen típussal rendelkezik (a már tárgyalt shell, telnet, startx és rdesktop), ami mellé saját egyedi típusokat írhatunk. A hangkezelésnél láttuk, hogy hasznos, ha a felhasználók tudják használni a kliens *alsamixer* programját, vagy kirakhatunk egy *top*ot valamelyik virtuális terminálra, vagy indíthatunk egy *TCD*-t [7], amivel az audio CD-ket lehet lejátszani.

Az alsamixerhez hozzunk létre egy `alsamixer` nevű fájlt a `/etc/screen.d` könyvtárban az alábbi tartalommal:

```
#!/bin/bash
/bin/alsamixer
/bin/sleep 1
```

A `sleep`-re azért van szükség, hogy ha kilépnek a programból, akkor az kis várakozással induljon újra, és ne lehessen leterhelni a rendszert a túlságosan gyakori újraindítással.

Ugyanez a top-hoz:

```
#!/bin/bash
/bin/cp /nfsroot/root/.toprc /.toprc
```

[1] Ebben a csomagban található meg a network block device-on keresztüli swapoláshoz szükséges *ltspswapd* program is.

```
/usr/bin/top -s
/bin/sleep 1
```

Érdemes megfigyelni, hogy a *top* a gyökérkönyvtárban keresi a konfigurációs állományát, amit oda kell másolni a számára, valamint, hogy a *top* parancsot `-s` paraméterrel indítjuk, így, bár rendszergazda tulajdonú folyamat, mégsem lehet leállítani vele egy process futását, vagy módosítani annak prioritását.

Az itt említett *top* és *tcd* programok sajnos nem szerepelnek az LTSP-ben, ezért vagy statikusan fordított binárisként kell elhelyeznünk a `/usr/bin` könyvtárban, vagy saját csomagot kell fordítanunk belőlük.

4.7. Monitorozás, távoli adminisztrálás

ltspinfo ♦ A kliensgépek állapotát távolról is nyomon követhetjük, sőt megfelelő beállítások esetén még némi beavatkozásra is lehetőségünk nyílik. Az LTSP erre kifejlesztett alapértelmezett eszköze a kliensen futó ltspinfod daemon, és az ehhez kapcsolódó *ltspinfo* program, ami az *ltsp-utils* csomag része. Az *ltspinfo* paranccsal ellenőrizni tudjuk a kliensgép által használt konfigurációs változókat:

```
ltspinfo -h 192.168.1.32 -cfg XSERVER
ltspinfo -h 192.168.1.14 -cfg RDP_OPTIONS
```

Lehetőség van a kliensgép `/proc` könyvtárában lévő fájlok kiíratására:

```
ltspinfo -h 192.168.1.32 -proc cpuinfo
ltspinfo -h 192.168.1.32 -proc meminfo
```

Kikapcsolhatjuk, vagy újraindíthatjuk a kliensgépet:

```
ltspinfo -h 192.168.1.11 -s
ltspinfo -h 192.168.1.12 -r
```

A `/proc` könyvtár távoli olvasásához az `ALLOW_PROCREAD=Y`, a távoli leállításhoz és újraindításhoz pedig az `ALLOW_SHUTDOWN=Y` megadása szükséges az `lst.conf` fájlban.

SNMP ♦ A monitorozás egy másik lehetséges módja, ha a kliensgépünkön SNMP daemont indítunk. Ehhez az `lts.conf` fájlba kell beírnunk, hogy `SNMPD=Y`. Az elindított daemonhoz csatlakozhatunk közvetlenül az *snmpwalk* programmal, vagy készíthetünk a kinyert adatokból grafikont az *mrtg* vagy a *munin* használatával.

Például a kliens *load average* értékeinek lekérdezése így történhet:

```
snmpwalk -Os -c public -v 1 192.168.1.32 laLoad
```

ssh ♦ Az LTSP tartalmaz egy sshd daemont is, viszont az alapbeállításokkal ez csak akkor indul el, ha a `LOCAL_APPS=Y` beállítást használjuk az `lts.conf` fájlban. Ez elindít mindent, ami a kliensen helyileg futó alkalmazások használatához szükséges (NFS-home, NIS, stb.), ezekre viszont nekünk nincsen szükségünk. A megoldás a `/etc/rc.locale` fájl átírása. Ez a fájl alapértelmezésben nem is létezik, most azonban hozzuk létre, és töltsük fel az alábbi tartalommal:

```
# Start sshd
if [ "${SSHD}" = "Y" ] ; then
  /bin/cp /nfsroot/root/.ssh/ssh_host_*sa_key /tmp/
  /bin/chmod 600 /tmp/ssh_host_*sa_key
  echo "Starting sshd..."
  sshd
fi
```

Módosítanunk kell még a kliensek `etc/ssh/sshd_config` állományát is:

```
HostKey /tmp/ssh_host_rsa_key
HostKey /tmp/ssh_host_dsa_key
SyslogFacility AUTH
LogLevel INFO
PermitRootLogin yes
StrictModes yes
PubkeyAuthentication yes
AuthorizedKeysFile      %h/.ssh/authorized_keys
PasswordAuthentication no
KeepAlive yes
```

A kliensgép által használandó `ssh_host` kulcsokat másoljuk be az exportált gyökérkönyvtár `/root/.ssh` könyvtárába. A `/tmp`-be való kimásolásra azért van szükség, mert a fájl jogosultságai csak így állíthatóak be a szükséges értékre.

Ezzel elértük, hogy az `lts.conf` fájlban az `SSHD=Y` beállításával a kliensgépen is tudunk SSH daemont indítani. Mivel a kliens rendszergazdáját nem lehet jelszó alapján autentikálni, ezért kulcs alapú autentikációt kell megvalósítanunk. Ehhez helyezzük el a nyilvános SSH-kulcsunkat a kliensgép `/root/.ssh` könyvtárában az `authorized_keys` fájlba.

5. Biztonság

Biztonsági szempontból az LTSP sok támadási pontot nyújt. Csak tűzfallal szeparált lokális hálózatban használjuk. Szükséges a megfelelő házirend megalkotása és betartása, bizonyos esetekben pedig akár egyes szolgáltatások leállítása is indokolt lehet.

Javaslom a szervergépek valamelyikén futtatni az *arpwatch* programot, aminek segítségével azonnali értesítést kapunk, ha a hálózatban új IP-cím–hardvercím pár jelenik meg. A bejövő leveleket *procmail*lel feldolgozva némi szkripteléssel akár azonnali védelmi folyamatot indíthatunk ebben az esetben, például módosíthatjuk a tűzfal beállításait.

Vegyük szemügyre az LTSP fő biztonsági kockázatait:

telnet ♦ A biztonság egyik legégetőbb pontja a telnet használata. A loginnév/jelszó páros kódolatlanul utazik a hálózaton, nagyon könnyen lehallgatható. Ha nem tudjuk biztosítani, hogy a hálózatban csak általunk felügyelt gépek legyenek, akkor inkább ne is használjuk.

XDMCP ♦ Bár kevésbé rettegett protokoll, mint a telnet, valójában az xdmcp is ugyanúgy lehallgatható mint telnet esetén, a jelszavunk szintén kódolás nélkül utazik!

nyomtatás ♦ A klienseink 9100-as, 9101-es és 9102-es tcp portján várják a nyomtatni valót. Mivel semmiféle autentikálás nincsen, ezért bárki, aki ezekre a portokra adatot tud küldeni, az tud nyomtatni a klienshez csatlakozó nyomtatóra. Ha például PostScript formátumot hardveresen tudó nyomtatónk van, akkor pusztán a cat és a telnet parncsokkal is nyomtathatnak az ügyesebb felhasználók:

```
cat print.ps | telnet 192.168.1.28 9100
```

hang ♦ az *esd* démon a kliensünk 16001-es TCP portján figyel, és a nyomtatáshoz hasonlóan itt sincsen semmiféle autentikáció. Bárki, aki erre a portra adatot tud küldeni, az „megszólaltathatja” a kliensünket. Ez néhány esetben akár hasznos is lehet, például a saját mikrofonunkon bejövő jelet a kliens hangfalára továbbíthatjuk, így meglephetjük

az órai munka helyett mással foglalkozó diákot. (A diák meglepésére egyébként csendes módok is kínálkoznak, például feldobhatunk neki egy figyelmeztető ablakot – csak a `DISPLAY` változót kell beállítanunk a kliensgépére.) Általában ha a klienseink külön tűzfallal védett hálózatban vannak, akkor sem a hang, sem a nyomtató nem okoz gondot, amíg felhasználóink megbízhatóak.

ltspinfod ♦ Nagyszerű dolog, hogy kliensgépeinket távolról leállíthatjuk, vagy újraindíthatjuk, de ha ezt az ltspinfo használatával akarjuk megoldani, akkor számítanunk kell arra, hogy ez sem autentikált módon történik, így gyakorlatilag bármelyik felhasználónk, vagy bárki, aki az adott portot (tcp/9200) eléri és kellő ismerettel rendelkezik, szintén leállíthatja vagy újraindíthatja a klienseinket. Ezt az opciót kapcsoljuk ki; ha nagyon szükséges az újraindítás, használjuk az SSH-t!

SSH ♦ Az előző fejezetekben láttuk, hogyan lehet megoldani, hogy a kliensgépeinken SSH daemon fusson. Azt is láttuk, milyen trükközéssel járt, hogy a daemon biztonságosan letároltnak gondolja a `host_key` fájlokat. Természetesen ez egyáltalán nem így van, és mivel a `host_key` gyakorlatilag szabadon elérhető, ha egy idegen gép felveszi a valamelyik kliensünk IP-címét, és ezt a `host_key`-t használja, akkor fel sem tűnik, hogy más gépre sikerült bejelentkeznünk.

6. Készítsünk saját LTSP-t!

Előbb vagy utóbb minden komolyabb LTSP-üzemeltető eljut arra a pontra, hogy további lehetőségekkel szeretné bővíteni a kiépített vékonykliens rendszerét. Ilyenkor először statikusan linkelt bináris programokkal rakja teli az exportált `/bin` és `/usr/bin` könyvtárakat, majd egyre inkább library-ket is másol a fájlrendszerbe, végül elérkezik arra a pontra, hogy ezt így már nem lehet folytatni, és saját LTSP-t készít. Szerencsére, ez nem is olyan nehéz feladat, mint hinnénk.

6.1. Az LBE telepítése és használata

Ha további programokat szeretnénk a klienseken futtatni, vagy az alap LTSP rendszer valami miatt nem felel meg az igényeinknek (például patchelni kell az rdesktop programot a magyar billentyűzetért, vagy egy speciális videokártya miatt egy X.Org pathcre van szükségünk), akkor lehetőségünk van saját LTSP-t fordítani az LBE (LTSP Building Environment) segítségével. Ez az eszköz elsősorban az LTSP-fejlesztők számára készült, de mi is használhatjuk saját LTSP rendszerünk létrehozására.

Bár az LBE-t letöltve és szétnézve a létrejött könyvtárrendszerben örömmel látjuk a `crosscomp-src` könyvtárat, sajnos a cross-compilation még nem működik, vagyis csak x86 architektúrán tudunk x86 alapú kliensek számára LTSP-t fordítani.

A fordításhoz szükségünk lesz kb. 5 GB helyre, egy 3.2 és 3.4 közötti verziójú *gcc*re, továbbá *make*re, *flex*re, *bison*ra, *autoconf*ra, *zlib1g-dev*-re és esetleg még néhány programkönyvtárra, ami fordítás közben úgyis kiderül.

Saját LTSP rendszer elkészítéséhez először is töltsük le az LBE-t:

```
cvs -d :pserver:anonymous@cvs.ltsp.org:/usr/local/cvsroot checkout lbe
```

Ha ezzel megvagyunk, lépjünk be az `lbe` könyvtárba, és töltsük le az aktuális LTSP forrást:

```
./build_all --fetch
```

Amikor ezzel is végeztünk, jöhet a fordítás. A gyorsabb fordítás érdekében adjuk meg a processzorok számát is:

```
./build_all --cpus=2
```

És itt most hátradőlhetünk, vagy elmehetünk ebédelni. Végül készítsük el a lefordított programokból az *ltsp-** csomagokat:

```
./build_all --makepkg
```

Az új csomagokat az `lbe/packages` könyvtárban találjuk, és akár rögtön innen telepíthetjük is őket az *ltspadmin* használatával.

6.2. Régi programok módosítása, új programok létrehozása

Az eddigiekben láttuk, hogyan tudjuk előállítani forrásból az eddig is használt LTSP rendszerünket, most végre foglalkozhatunk azzal a kérdéssel, hogy hogyan tudunk módosítani rajta, hogyan tudunk új programokat felvenni bele.

Nézzünk bele az `lbe/ltsp-src` könyvtárba! Minden egyes LTSP programcsomaghoz találunk itt egy-egy saját kis könyvtárat. A program felépítését a `package.def` szabályozza.

Első példaként módosítsuk az rdesktop programot, hogy alkalmas legyen a magyar billentyűzetkiosztásban fontos szerepet kapó AltGr billentyű kezelésére. Lépjünk be az `rdesktop` könyvtárba, és töltsük le a szükséges patch-eket:

```
wget http://www.inf.bme.hu/~pts/pts-rdesktop-1.4.1-xkeymap-altgr-localstate.patch
wget http://www.inf.bme.hu/~pts/pts-xmodmap-raw-us.txt
wget http://www.inf.bme.hu/~pts/pts-rdesktop-keymap-raw.txt
```

Módosítsuk az `rdesktop.wrapper` fájlt az alábbi módon:

```
#!/bin/sh
/usr/X11R6/bin/xmodmap /usr/share/xmodmap.raw-us || exit
while true; do
  /usr/bin/rdesktop -k pts-raw "$@"
done
```

A `packages.def` fájlban be kell állítanunk, hogy az *altgr-localstate* patchet is szeretnénk használni, valamint az `INSTALL` változót kell módosítanunk, hogy a `xmodmap.raw-us` és a `pts-raw` keymap fájlok is belekerüljenek az LTSP csomagba:

```
PREPATCH2 = pts-rdesktop-1.4.1-xkeymap-altgr-localstate.patch
INSTALL = make DESTDIR=${LTSP_ROOT} prefix=/usr install && \
  cp ../rdesktop-screen ${LTSP_ROOT}/etc/screen.d/rdesktop && \
  cp ../rdesktop.wrapper ${LTSP_ROOT}/usr/bin/rdesktop.wrapper && \
  cp ../pts-xmodmap-raw-us.txt ${LTSP_ROOT}/usr/share/xmodmap/xmodmap.raw-us && \
  cp ../pts-rdesktop-keymap-raw.txt ${LTSP_ROOT}/usr/share/rdesktop/pts-raw
```

Természetesen lehetőség van egyetlen program újrafordítására is. Ehhez lépjünk az `lbe/ltsp-src` könyvtárba, és futtassuk le a következő parancsokat:

```
./build --clean --only=rdesktop
./build --only=rdesktop --cpus=2
cd ..
./build_all --makekpkg
```

Ha egy új programból akarunk LTSP csomagot készíteni, akkor hozzunk létre számára egy könyvtárat az `lbe/ltsp-scr` könyvtárban, hozzuk létre a `package.def` fájlt a régebbi csomagoknál látható módon, és fordítsuk le a fent leírt parancsok segítségével a programunkat.

Hivatkozások

[1] AMD Geode™ Processor Linux Drivers. URL http://www.amd.com/us-en/ConnectivitySolutions/TechnicalResources/0,,50_2334_2452_11363,00.html.

[2] Linux Terminal Server Project. URL http://ltsp.org/.

[3] LTSP clients on Ultra Sparc. URL http://math.univ-lille1.fr/ltsp-sparc/.

[4] Az *ltsp-esd-alsa* csomag lelőhelye. URL http://prdownloads.sourceforge.net/symbiont/LTSP-esd-alsa.tgz?download.

[5] Az *ltsp-server-pkg* csomag lelőhelye. URL http://ltsp.mirrors.tds.net/pub/ltsp/utils/.

[6] Szabó Péter: rdesktop AltGr javítások, 2006. szeptember. URL http://www.inf.bme.hu/~pts/pts-rdesktop-altgr-fixes.txt.

[7] TCD: karakteres alapú audió CD-lejátszó alkalmazás. URL http://www.nongnu.org/tcd/.

[8] Microsoft Windows Server 2003 TechCenter: Ügyféleszköz-hozzárendelések beállításainak megadása, 2005. január. URL http://www.microsoft.com/technet/prodtechnol/windowsserver2003/hu/library/ServerHelp/17d44d9a-cf4b-4a6a-94ec-093cb5f8b2b7.mspx?mfr=true.

Az openSUSE összeállító (build) szolgáltatása – automatikus csomaggenerálás különböző disztribúciókra és hardverekre

Hargitai Zsolt

Kivonat

Az alábbiakban bemutatjuk az openSUSE összeállító (build) szolgáltatását, melynek célja a szoftverfordítási és csomagkészítési folyamat hatékony támogatása és a lehető legnagyobb fokú automatizálása. Más rendszerekkel szemben ez nem egy adott disztribúcióra koncentrál, hanem képes többet is kiszolgálni (SUSE Linux változatok és egyéb disztribúciók). Úgy készült, hogy egyszerűen lehessen elágaztatni egy csomagot például a módosítások kipróbálásához, vagy eltérő csomagkonfiguráció használatához. A rendszer automatikusan újraépíti a csomagokat, ha azt észleli, hogy a tartalmuk megváltozott.

Tartalomjegyzék

1. Bevezető

A nyílt forráskód előnyei a rendelkezésre álló szoftverek nagy száma és a gyors frissítési ciklusok. Ezeknek az előnyöknek azonban ára van: a szoftverek szerzőinek garantálniuk kell, hogy a programjuk futni fog a legfrissebb rendszereken is. A felhasználók pedig elvárják a bináris változatokat, hiszen a fordítás sokáig tart, és sokszor nehézkes. Minden bináris változat állandó naprakészen tartása azonban roppant időigényes. Néha szinte lehetetlen, ha a szoftver szerzője nem fér hozzá a kívánt architektúrához vagy disztribúcióhoz.

A SourceForge-hoz hasonló projektgyűjtő szerverek hozzáférést biztosítanak egy különböző architektúrákat használó gépekből álló „fordítási farmhoz”, különböző verziójú disztribúciókkal. A csomagok összeállítását azonban továbbra is kézzel kell elvégezni, ha a disztribúció megváltozik. Más rendszerek csak egy adott disztribúcióhoz képesek előállítani a csomagokat, vagy pedig kézi közreműködésre van szükség az ismételt összeállításhoz.

Az openSUSE [1, 2, 3] összeállító (build) szolgáltatása [4] éppen arra készült, hogy a szoftverek szerzőinek kényelmes eszközt nyújtson csomagjaik többféle disztribúcióhoz való előállításához, és lehetővé tegye az automatikus újrafordítást, ha a disztribúció megváltozik. Legfontosabb jellemzői:

1. nem koncentrál egyetlen disztribúcióra (de csak RPM-alapú disztribúciókkal működik);
2. egyszerűen kezelhetőek a csomagok különböző verziói;
3. automatikusan végrehajtható az újrafordítás.

Minden szoftverkészítő szabadon felhasználhatja az összeállító szolgáltatást, hogy elkészítse szoftverének csomagjait különféle disztribúciókhoz, és azokat automatikusan naprakészen tarthassa. Akik szeretnek kísérletezni, készíthetnek egyéni csomagváltozatokat is. Akik egy adott csomagot keresnek, végignézhetik az összeállító kiszolgálón, hogy van-e az igényeiknek megfelelő változat.

Az alábbi dokumentumban az openSUSE összeállító szolgáltatásának felépítését írjuk le.

2. Projektek

A *projekt* az openSUSE összeállító szolgáltatás alapvető építőkockája. A projektek forrásfájlokat tartalmazó csomagokból állnak, és ezek a forrásfájlok szolgálnak az egyes csomagok bináris RPM-jeinek az előállítására.

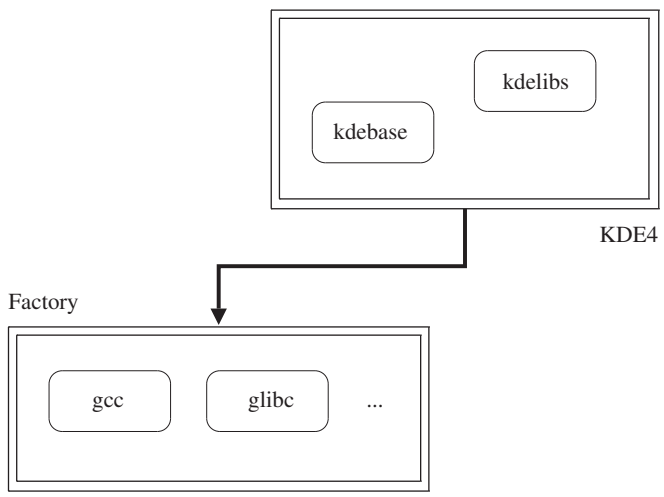
2.1. Hozzáférés-vezérlés

Egy projekt létrehozásakor a létrehozó automatikusan a projekt adminisztrátorává válik. Az adminisztrátor felvehet és eltávolíthat csomagforrásokat, és módosíthatja a projekt konfigurációját. A konfigurációnak része az adminisztrátorok listája, vagyis egy adminisztrátor felvehet más felhasználókat az adminisztrátorok listájába.

2.2. A projekt rétegei

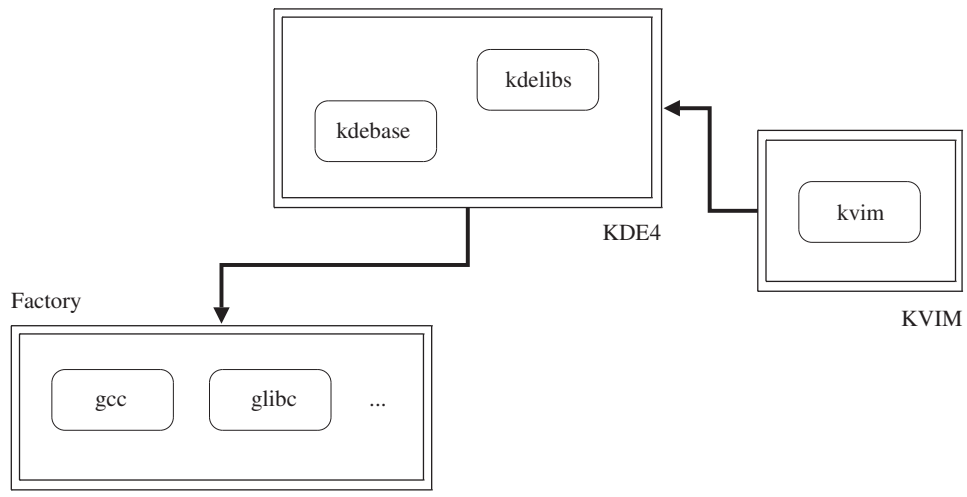
Az összeállítási folyamatnak el kell készítenie egy összeállítási rendszert, amelyben az összes RPM telepítve van. Mivel az egyes projektek csak a bennük található csomagok RPM-jeit tartalmazzák, megadható egy olyan projekt, amelyik más projektek felett helyezkedik el (azaz rájuk épül). Egyszerű példaként vegyük a KDE 4 projektet, amelyik a következő KDE-kiadás csomagjait tartalmazza (1. ábra). A csomagok összeállításához más RPM-ekre (például fordítóprogramokra és egyéb eszközökre) is szükség van. Vagyis a KDE 4 csomagot egy másik

olyan projekt tetejére kell elhelyezni, amelyik már tartalmazza ezeket az RPM-eket – ilyen például az openSUSE Factory disztribúció.



1. ábra. A KDE 4 projekt a Facoryprojektre épül

Vegyünk egy másik projektet (2. ábra): a KVIM csomag a kvim programot, a népszerű szerkesztő grafikus változatát tartalmazza. A KVIM fejlesztői, akik biztosítani kívánják csomagjukat KDE 4-hez, a KDE 4 csomag fölé helyezik el a projektjüket. Nem kell megadniuk, hogy a KVIM-nek a Factory csomagokra is szüksége van, ugyanis ez a rétegzés automatikusan öröklődik. A KVIM csomag összeállításakor a rendszer be lesz állítva a KDE 4 projekt KDE csomagjaival és a Factory projekt alapcsomagjaival.



2. ábra. A KVIM projekt közvetlenül a KDE 4-re, közvetetten a Factory-ra épül

2.3. A szükséges csomagok megadása

A SUSE Linuxban található *build* program használói tudják, hogy jelenleg egy csomag által igényelt RPM-ek megadásához az összeset be kell írni a `BuildRequires` sorokba. Mivel ez

nem túlságosan rugalmas megoldás, az openSUSE összeállító szolgáltatása automatikusan, tranzitív módon kibővíti a `BuildRequires` feltételeket az egyes RPM-ek *requires* listájával. Ennek eredményeképpen drasztikusan csökken a megadandó RPM-ek száma. Alapesetben csak néhány **-devel* csomagot kell megadni, például a *cups* csomaghoz `BuildRequires` feltételként a következőket: `gcc-c++ libpng-devel libtiff-devel openslp-devel openssl-devel pam-devel tcpd-devel`. Ez a csökkentés megnöveli annak esélyét is, hogy a csomag változtatások nélkül, problémamentesen lefordul más disztribúciókon is. Az említett `BuildRequires` következtében a `neededforbuild` el fog avulni.

2.4. Rétegek és megbízhatóság

Fontos tudni, hogy mely projektek szolgáltak egy csomag összeállítására, hiszen a projektek csomagjai olyan rosszindulatú kódot is tartalmazhatnak, amely tönkreteheti az összeállítás eredményét. Alapszabály, hogy egy projekt csomagjai nem épülhetnek olyan projektekre, melyek náluk kevésbé megbízhatóak.

2.5. Összeállítás többféle architektúrához

Egy projekt megadhatja, hogy a csomagjai többféle architektúrákhoz is elkészüljenek. Továbbá ki is hagyhatók bizonyos architektúrák a csomagokból, hogy a reménytelen összeállításokkal ne kelljen feleslegesen próbálkozni.

Ne feledjük, hogy a rétegzés korlátozza a használható architektúrák számát: ha a rétegek bármelyike nem támogat egy adott architektúrát, akkor lehetetlen lesz RPM-et készíteni ahhoz az architektúrához, hiszen az összeállítási rendszer kialakításához szükséges csomagok nem állnak rendelkezésre.

2.6. Összeállítás többféle disztribúcióhoz

Az openSUSE összeállító szolgáltatás fontos funkciója, hogy nemcsak többféle architektúrához, hanem többféle disztribúcióhoz is csomagokat tud készíteni. Lehet szó akár a SUSE Linux vagy a SLES régebbi változatairól, de akár idegen disztribúciókról is, mint a Fedora vagy a Mandriva.

Többféle disztribúcióhoz történő összeállítás során mindegyikhez ki kell alakítani az összeállítási környezetet. A projekt tulajdonságai – mint például a rétegzés vagy az architektúrák – ténylegesen az összeállítási környezet tulajdonságai. Egy másik környezetnek más jellemzői lehetnek.

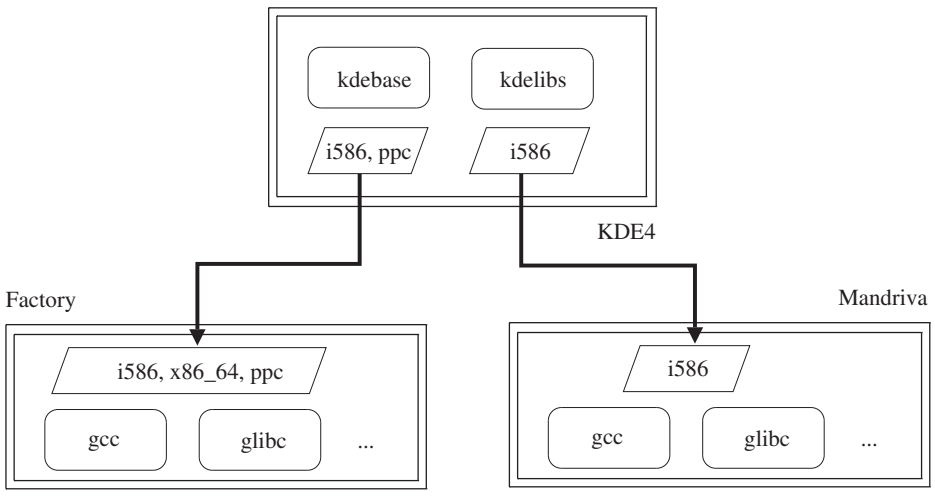
Vegyük ismét példának a KDE 4 projektet (3. ábra). Jelenleg a Factory projektre épít az i586 és ppc architektúrák esetén. A fenntartók úgy döntöttek, hogy a Mandrivához is készítenek csomagokat. Ezért fel kell venniük egy új összeállítási környezetet, amely a Mandriva disztribúció tetejére épül. A példánkban a Mandriva csak i586 RPM-eket tartalmaz az openSUSE összeállító szolgáltatásban, vagyis csak i586 KDE 4 RPM-ek készíthetők hozzá.

3. Csomagforrások

Egy csomag forrásának három fajtája van: teljes forrás, hivatkozás egy másik projekt csomagjára, valamint egy hivatkozás egy javítókészletre.

3.1. Teljes forrás

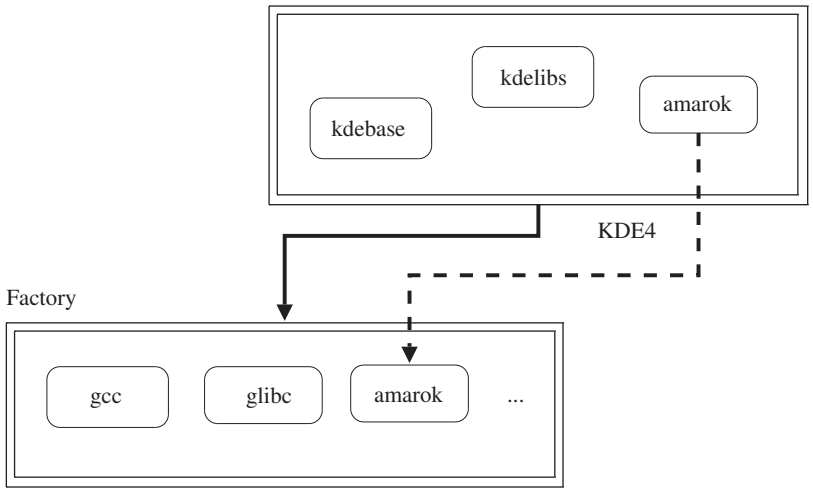
Ez a forrás legegyszerűbb formája: a csomag előállításához szükséges összes fájl benne van.



3. ábra. Példa projektek architektúrafüggő egymásra épülésére

3.2. Forráshivatkozások

Pontos másolat helyett néha hasznos lehet létrehozni egy hivatkozást egy másik projekt csomagjára. A hivatkozás előnye, hogy a forrás változásai automatikusan átvezetődnek. Vegyük ismét példának a KDE 4 projektet (4. ábra). Tegyük fel, hogy a rendszergazdák egy olyan *amarok* csomagot szeretnének, amelyikben már az új KDE 4 függvénytárak találhatók. Ehhez létrehoznak egy hivatkozást az openSUSE Factory projekt *amarok* csomagjából. Ha az *amarok* csomag módosul a Factory projektben, akkor automatikusan elkészül egy frissített KDE 4-verzió.



4. ábra. Példa forráshivatkozásra: egy csomag hivatkozik egy másik projektben levő csomagra

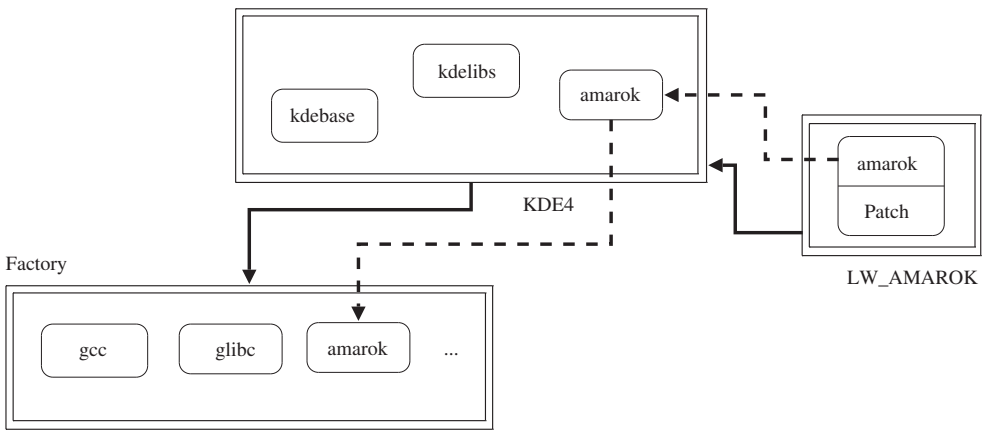
3.3. Forráshivatkozások javításokkal

Gyakran csak egy kis részletet kell módosítani a forrásban, például egy konfigurációs paramétert. Ebben az esetben preferált megoldás: hivatkozás egy sor javítására. Ez úgy viselkedik,

mint egy normál forráshivatkozás, de a csomag összeállítása előtt a javítások automatikusan alkalmazásra kerülnek. A normál hivatkozáshoz hasonlóan a csomag változásai átvezetődnek, vagyis a forrásmódosítás hatására új összeállítás készül. Ismét csak egy példa: tegyük fel, hogy a KDE 4-es fiúk szeretnének egy „könnyített” *amarok* csomagot kapcsolni a KDE 4 függvénytárakhoz. Ehhez létrehoznak egy új, LW_AMAROK nevű projektet, ráhelyezik a KDE 4-re, és létrehoznak egy hivatkozást az *amarok* csomag egy javítására.

4. Csomagok átadása más projekteknek

Tegyük fel, hogy a KDE 4 projekt óriási siker lett, és minden problémamentesen működik. A projekt karbantartói azt szeretnék, hogy a Factory disztribúció vegye fel az ő forrásukat. Ehhez egy áthelyezési kérést nyújtanak be a Factory projekthez. A Factory karbantartói kilistázzák az összes nyitott kérést, és készítenek egy különbséglistát a benyújtott csomagok és a disztribúciójukban található csomagok között. Ha úgy gondolják, hogy a csomagok megfelelnek a befoglalási irányelveknek, akkor átmásolják a forrásfájlokat saját projektjükbe. A folyamatot az 5. ábra szemlélteti.



5. ábra. Példa projektek közötti csomagátadásra

4.1. Áttekintők

A csomagok forrásait csak a projekt adminisztrátorai módosíthatják; ők felelősek azért, hogy ellenőrizzék a különbségeket a források között, és eldöntsék, hogy a benyújtott csomag elfogadható-e vagy sem. Mivel ez nagy projektek esetén meglehetősen sok munka, megadhatók áttekintők *(reviewers)* a projekt bármely csomagjához. Ha egy másolási kérés érkezik egy olyan csomagra vonatkozóan, amelyhez van kijelölve áttekintő, akkor a kérés először az áttekintőhöz kerül. Az áttekintő megvizsgálja a forráskód változásait és megjegyzéseket fűz hozzá. Ha az áttekintő javasolja az átmásolást, akkor ő újraküldi a kérést a projekt adminisztrátorainak, akik az ő megjegyzései alapján dönthetnek.

5. Csomagok összeállítása

Miután létrehoztak egy csomagot egy projektben, automatikusan összeállításra kerül az openSUSE összeállítási farmon.

5.1. Elszigetelés (sandboxing)

Mivel egy felhasználó szabadon felhasználhatja az összeállító szolgáltatást bármilyen forrásfájlokhoz, vigyázni kell a rosszindulatú kódokkal. Ez nem annyira a tényleges összeállítás problémája (ami ugye nem root felhasználóként történik), hanem az összeállítási rendszert alkotó RPM-ek telepítéséé, hiszen ez a művelet a root felhasználó nevében zajlik. A folyamat biztonságossá tételéhez a Xen segítségével egy virtuális gépet készítettünk az összeállításhoz. További előnyként a rosszindulatú folyamatok nem férnek hozzá a hálózathoz, és az összes folyamat automatikusan leállításra kerül az összeállítás után.

5.2. Automatikus újraépítések

Az openSUSE összeállító szolgáltatás automatikusan újraépíti a csomagokat, ha azok lejártak. Egy csomag lejárt, ha

1. a forrása módosult;
2. hivatkozás esetén a hivatkozott forrás módosult;
3. javításos hivatkozás esetén a javított forrás vagy a javítások egyike módosult;
4. a csomag összeállításához használt RPM-ek valamelyike lejárt és újra lett építve;

Ne feledjük, hogy a lejáratot figyelő algoritmus csak a forráskódok változását nézi, vagyis nem indul el végtelen összeépítési ciklus függőségi kör kialakulása esetén sem.

5.3. Kiadási számok

Az openSUSE összeállító szolgáltatás automatikusan kiadási számokat rendel az összeállított csomagokhoz. A kiadási szám két részből áll: az első a forrás módosításszámlálója, amelyet egy újraépítési számláló követ. Minden egyes alkalommal, amikor a csomag forrása módosul, a forrásváltozás-számláló megnő, és az újraépítés-számláló visszaáll 1-re. Ha tehát a régi kiadásszám *4.5* volt, és a forrás módosul, akkor a következő összeállítás kiadása *5.1* lesz, a következő újraépítésé (ha a forrás nem változott) *5.2* és így tovább.

A kiadásszámozás egy projekten belül helyinek számít. Ez azt jelenti, hogy két projekt kiadásszámainak összehasonlítása értelmetlen. A nagyobb kiadásszám nem feltétlenül jelenti azt, hogy a csomag forrása újabb. Kivételt képeznek persze a hivatkozott/kapcsolt csomagok, ahol a hivatkozott kiadásszáma mindig magasabb, mint a hivatkozás forrásáé.

Hisszük, hogy két projekt csomagjainak összehasonlítását egyszerű számozási renddel nem lehet szabályozni. Ezt más szinten kell kezelni, irányelveket kell rá kidolgozni. A csomagkarbantartó szoftvernek kell valamilyen eszközt nyújtania az adminisztrátor számára, hogy beállíthassa a két projekt közötti rendezést, ha a projekt saját kiadásszámainak vagy korszakszámainak összehasonlítása nem jár eredménnyel. Az a vélemény sem alaptalan, hogy még a verziószámokat sem szabad összehasonlítani, mivel azok nem helyiek, hiszen „feljebb” kerülnek meghatározásra.

6. Összefoglalás

Az openSUSE összeállító szolgáltatás úgy készült, hogy a szoftverszerzők és csomagkészítők számára lehetővé tegye azt, hogy szoftvereik bináris csomagjai mindig naprakészek legyenek. Ehhez létre kell hozniuk egy, az összeállítandó csomagokat tartalmazó projektet, és ezt rá kell helyezniük egy vagy több disztribúcióra. Forráshivatkozások és javításos hivatkozások

használhatók a csomagok változásainak egyszerű nyomon követéséhez. A rendszer automatikusan összeállítja (és újraépíti) a szoftvert úgy, hogy a felhasználóknak mindig a legfrissebb verzió álljon a rendelkezésére. Az összeállítási folyamat a Xen hipervizor rendszert használja a különböző összeállítások elszigeteléséhez.

Hivatkozások

[1] Hasznos leírások az openSUSE-ről.
URL http://hu.opensuse.org/Dokument%C3%A1ci%C3%B3.

[2] Magyar openSUSE weboldal. URL http://hu.opensuse.org/.

[3] Az openSUSE projekt weblapja. URL http://opensuse.org/.

[4] Az openSUSE összeállító szolgáltatás.
URL http://en.opensuse.org/Build_Service.

Mi mind egyéniségek vagyunk! Cfengine bevezetése közepes méretű szerverparkon

Hirling Endre

Kivonat

A *Cfengine* nagy számú szerver konfigurációjának és működésének ellenőrzésére szolgál, a rutinfeladatokat automatizálja. Legfontosabb képességei: konfigurációs fájlok szerkesztése, verziókövetése; futó programok leállítása, nem futó programok elindítása; információgyűjtés, állapotfigyelés- és elemzés, káosz esetén riasztás; csomagok telepítése, a rendszer frissítése. Mindezt különösebb intelligencia nélkül, a rendszergazda által készített konfigurációs fájlokban előírt módon teszi.

Szüksége lehet rá annak, akinek sok (egynél több) többé-kevésbé egyforma Unixot kell üzemeltetnie; aki arra számít, hogy a közeljövőben sok Unixot fog üzemeltetni, és szeretné, ha valaki megcsinálná helyette a telepítés utáni rutinfeladatokat; aki lusta (a jó rendszergazda lusta) vagy szorgalmas, de kényelemszerető.

Tartalomjegyzék

1. Bevezető

1.1. Kezdetben volt A Rendszergazda, Akinek Szervere Van

Egy rendszergazda munkája során sokféle feladattal találkozik, melynek nagy része rendszeresen és/vagy minden üzemeltetett gépen elvégzendő. Ilyen ismétlődő feladat pl. a telepítés után a házi szabványoknak megfelelő beállítások elvégzése, vagy a gépek rendszeres ellenőrzése (fut-e a program, van-e elég hely a diszkeken stb.). Ezek a feladatok egy gépnél viszonylag kevés ráfordítással elvégezhetőek, és rendes helyeken rendelkezésre áll a megfelelő kőtábla is a szervertelepítés, ill. -ellenőrzés tízparancsolatával, így tehát a szórakozott rendszergazda ellen is biztosítottuk magunkat. Nem biztos azonban, hogy ez a jövőben is hatékony módszer marad.

1.2. Szaporodjatok és sokasodjatok!

Minden rendszergazda életében eljön az idő, amikor az egy szerverből kettő, majd húsz, kétszáz vagy még több lesz, és ezek időnként újabbakkal szaporodnak. Ekkor már a rendszer testreszabása, valamint a rendszeres ellenőrzések jelentős időt vesznek igénybe. Számoljunk csak utána: ha egy géppel egy nap csak két percet foglalkozunk, akkor az 60 gépnél már két óra, vagyis, a napi munkaidőnk egynegyede. Már ennyit dolgoztunk, és még tulajdonképpen nem csináltunk semmit. Ez tarthatatlan állapot, hiszen ha egyszer nem csinálunk semmit, akkor azt az időt meredt monitorbámulás helyett kávézással is lehetne tölteni.

1.3. A rendszerek túlerőben vannak! Erősítést kérünk!

Számos eszközt találhatunk készen a monitorozási, automatizálási feladatok nagy részére, sőt a megfelelő mennyiségű idő és szaktudás birtokában magunk is készíthetünk ilyet. Ez utóbbi megoldás azonban jóval több és sokrétűbb tudást igényel, mint maga az üzemeltetés. A saját fejlesztésű eszközt folyamatosan karban is kell tartani, jó eséllyel megismételve azt a munkát, amit adott esetben mások már elvégeztek.

1.4. Itt a Cfengine! Az ellenállás értelmetlen, mindenkit asszimilálunk!

A *Cfengine* egy ilyen, automatizálást és központosított felügyeletet segítő eszköz. Saját nyelvezete segítségével könnyen leírhatjuk az egyes géptípusok vagy egyedi hostok elvárt állapotát. Ezután nincs más dolgunk, mint a felügyelni kívánt hostokra is feltelepíteni a *Cfengine*-t, elindítani, hátradőlni, és nézni, ahogy helyettünk dolgozik. Szerzője Mark Burgess, az Oslói Egyetem *Hálózat- és rendszerfelügyelet*-tanára, aki tudományos szemszögből is körbejárta a témát, és számos cikket is írt a számítógéprendszerek működéséről és adminisztrációjáról. Lássuk tehát, miként állíthatjuk szolgálatunkba, hogy – miután szervereink szépen lassan behódoltak – rutinfeladatok helyett hasznos dolgokkal tölthessük az időt.

2. A Cfengine beállítása

2.1. A rest kétszer fárad. . .

Ahhoz, hogy a *Cfengine* egyáltalán megmozduljon, telepítés után még két konfigurációs állományt kell kitölteni. A *cfservd.conf*-ra a szerveren van szükség, ebben határozzuk meg, hogy mely kliensek kapcsolódhatnak a szerverhez, valamint a terjesztendő fájlok elérhetőségét. A kliens az *update.conf*-ból tudja, honnan kell letöltenie a tényleges konfigokat, ill. azok megsérülése esetén ennek alapján tudja újra letölteni a jó változatot.

A legegyszerűbb *cfservd.conf* így nézhet ki:

```
control:
  AllowConnectionsFrom = ( 1.2.3.4/24 2.3.4.5 )
admit:                  # "Beengedő" szekció
  /etc/cfengine *       # a konfigot odaadjuk minden
                        # szóbajöhető kliensnek
```

A hozzá tartozó *update.conf* pedig így:

```
control:
  policyhost = ( cfserver )
  cfe_dir = ( /var/lib/cfengine2 )
  actionsequence = ( copy )
copy:                           # másolunk
  any.!$(policyhost)::          # akinél a mesterpéldány van, nem másol
    $(cfe_dir)/inputs           # innen...
          dest=$(workdir)/inputs  # ... ide.
          recurse=inf
          mode=600
          type=binary
          server=$(policyhost)    # megadjuk a forrásszervert is
          exclude=*.lst           # valamint a nem terjesztendő fájlokat
          exclude=*~
          exclude=*.swp
          exclude=#*
```

Az összes konfigurációs állomány tartalmaz egy ún. *control* szekciót, ahol a szükséges változókat láthatjuk el kezdőértékkel. A változók között vannak konfigurációs paraméterek, melyek közvetlenül befolyásolják a *Cfengine* működését, de igény szerint létrehozhatunk sajátokat is, melyeket később még felhasználunk.

A *cfservd* és a kliensek egymást hostnév, IP-cím, valamint egymás publikus kulcsa alapján azonosítják, ezért feltétlenül szükséges, hogy gépeink DNS-bejegyzései rendben legyenek, és a *Cfengine* telepítésekor meg kell oldani az egymáshoz kapcsolódó hosztok kulcsainak kölcsönös telepítését is. (Több címmel rendelkező gép esetén oda kell figyelni, hogy ahhoz a címhez rendeljük a kulcsot, amelyiken a *Cfengine* kommunikálni fog.) Ha a fentiek teljesülnek, a *Cfengine* működésre kész, s elkezdhetjük beletölteni az eszünket. Ezt majd a *cfagent.conf* állományba tesszük, de előtte nézzük meg, mitől egyéniségek a hosztjaink.

2.2. Sötétkékek balra, világoskékek jobbra – osztályba sorolás

A *Cfengine* sokoldalúságának egyik alapköve, hogy osztályokba sorolja a hosztokat, ahol fut. Ezek az osztályok lehetnek előre definiáltak (mint pl. a hoszt neve, IP-címe, az operációs rendszer típusa), és mi is definiálhatunk újakat. Minden akció, vagy azok valamilyen kombinációjára korlátozható osztályokra, amint ezt már láthattuk az *update.conf*-nál is, ahol az `any.!$(policyhost)` minden olyan gépet jelent, amely nem tartozik semmilyen, a *policyhost* nevével megegyező osztályba. Mivel a *Cfengine* nem tudja, hogy mi csak és kizárólag hosztnévre gondoltunk, különösen oda kell figyelni a név választásakor, hogy lehetőleg az ne egyezzen meg egyetlen beépített osztály nevével sem. Mint később látni fogjuk, saját osztályokat definiálhatunk megadott feltételek teljesülése alapján, de azt is megtehetjük, hogy egyszerűen kézzel felsoroljuk az egy csoportba tartozó gépeket:

```
groups:
  mailhosts = ( mail mx0 mx1 )
  mysqlhosts = ( mysql )
```

Hogy egy adott hoszt milyen osztályokba tartozik, azt a *cfagent -p -v* paranccsal kérdezhetjük le. Mintakimenet:

```
Defined Classes = ( 192_168_1 192_168_1_31 192_168_2 192_168_3 195_70_33
  195_70_33_24 32_bit Day17 Hr00 Hr00_Q1 Min00_05 Min02 October Q1 Tuesday
  Ubuntu_6_06_LTS_ VMware Yr2006 any cfengine_2 cfengine_2_1 cfengine_2_1_20
  compiled_on_linux_gnu debian dusk dusk_interware_hu dusk_iw hu i686
  interware_hu ipv4_192 ipv4_192_168 ipv4_192_168_1 ipv4_192_168_1_31
  ipv4_192_168_2 ipv4_192_168_2_253 ipv4_192_168_3 ipv4_192_168_3_253
  ipv4_195 ipv4_195_70 ipv4_195_70_33 ipv4_195_70_33_24 linux linux_2_6_16_28
  linux_i686 linux_i686_2_6_16_28
  linux_i686_2_6_16_28__1_SMP_Thu_Aug_31_13_21_43_CEST_2006
  net_iface_eth0_5 net_iface_eth0_6 net_iface_lo )
```

Amint látható, van választék. Hogy egy adott műveletnél mely szempontokat vesszük figyelembe, azt a művelet típusa alapján magunk határozhatjuk meg.

2.3. Egységben az erő. Vagy egy bolond százat csinál?

Lássuk a *cfagent.conf* születését. Első menetben itt is ki kell tölteni a *control* szekciót, hiszen anélkül nem élet az élet.

```
control:
  actionsequence = ( editfiles directories processes alerts )
  policyhost = ( cfserver )
```

Valójában az egyetlen kötelezően kitöltendő dolog az *actionsequence;* ez mondja meg ugyanis a cfagentnek, hogy mit, és milyen sorrendben kell csinálnia. Érdemes itt megadni azt a szervert is, ahonnan letöltögetjük a központilag karbantartott fájlokat, hiszen ezt sok helyen fogjuk használni.

Lássuk tehát néhány morzsáját a *Cfengine*-be töltött okosságnak. Először is, ma már szinte mindenki használ az SSH-hoz kulcsalapú autentikációt. Természetesen minden szerverre fel kell telepíteni a megfelelő kulcsokat, adott esetben többfélét is, és ezeket folyamatosan karban kell tartani, ahogy az emberek és beosztások cserélődnek. Tartsunk tehát karban egy (vagy több) központi példányt, a többit bízzuk az imént megidézett csordaszellemre. Először is definiálunk egy mindenki által elérhető könyvtárat, ahol a terjesztendő publikus kulcsokat tároljuk:

```
control:
  pubdir = ( /usr/local/cfengine-public ) # így később majd lehet rá hivatkozni
admit:
  $(pubdir) *                             # ezt is mindenki elérheti
```

E *control* és *admit* szekciók tartalmát természetesen a korábban már létrehozott azonos nevű szekciókhoz kell hozzáadni. Ezután már csak a terjesztés módját kell részletesen leírni:

```
directories:
  any::
    /root/.ssh mode=700 owner=root

copy:
  any::
    $(pubdir)/authorized_keys
      dest=/root/.ssh/authorized_keys
      mode=600
      owner=root
```

```
      group=root
      verify=true
      server=$(policyhost)
      type=sum
```

Ennek eredményeképpen minden gépen létrejön a root `.ssh` könyvtára, amibe bele is kerül a központilag karbantartott kulcskészlet. Természetesen a megfelelő osztályok használatával elérhető, hogy egyes gépcsoportokra (pl. a fejlesztői szerverekre) más kulcskészlet kerüljön telepítésre, sőt akár ideiglenesen érvényes készletet is létrehozhatunk.

Lássunk egy izgalmasabb példát. Nem minden rendszergazda szereti, ha felhasználó nélküli szervereken rotálódik a `wtmp` állomány, ellenben ezt Debianék belegyógyították a `/etc/logrotate.conf`-ba, ahelyett, hogy külön konfigot csináltak volna neki a `logrotate.d` alatt. Ki kell tehát kommentezni az idevágó részt:

```
editfiles:
  debian::
    { /etc/logrotate.conf
      LocateLineMatching "^ */var/log/wtmp *\{"
      CommentToLineMatching "^ *}"
      CommentNLines "1"
    }
```

Ennyi. A fentiek hatására az összes Debianon megszűnik a `wtmp` rotálása.

Kicsit összetettebb, amikor nem tudjuk előre, mit kell az adott hoszton csinálni. Remek példa, hogy általában *openntpd*-t használok időszinkronizálásra, de szerveren és tűzfalon *ntpd* (régebbi neve: *xntpd*) van fent. El kell dönteni tehát, hogy melyiket találtuk, és annak megfelelően eljárni a konfig kitöltésénél. Szerencsére a két csomag máshova teszi a konfigfájlját, így könnyű dolgunk van:

```
classes:
 openntpd   = ( IsPlain(/etc/openntpd/ntpd.conf) )
 xntpd      = ( IsPlain(/etc/ntp.conf) )

alerts:
 !xntpd.!openntpd::
 "Nincs NTPd!"

processes:
 xntpreconf::
 "ntpd" signal=term restart "/etc/init.d/ntp-server start" useshell=true
 openntpreconf::
 "ntpd" signal=term restart "/etc/init.d/openntpd start" useshell=true
```

Fent az újraindítgatásról is gondoskodunk: küldünk egy TERM szignált (mindenféle érvényes szignált küldhetünk), majd lefuttatjuk az initszkriptet. Figyeljük meg, hogy a `restart` után nincs = jel, de attól még van neki paramétere. A beállítások így folytatódnak:

```
editfiles:
  openntpd::
    { /etc/init.d/openntpd
      DeleteLinesContaining "set -e"
    }
    { /etc/default/openntpd
      AppendIfNoSuchLine "DAEMON_OPTS='-s'"
    }
```

```
    { /etc/openntpd/ntpd.conf
      BeginGroupIfNoLineContaining "195.70.32.4"
        EmptyEntireFilePlease
        Append "server 195.70.32.129"
        Append "server 195.70.32.3"
        Append "server 195.70.32.4"
        DefineInGroup "openntpreconf"
      EndGroup
    }
  xntpd::
    { /etc/ntp.conf
      BeginGroupIfNoLineContaining "195.70.32.4"
        EmptyEntireFilePlease
        Append "server 195.70.32.129"
        Append "server 195.70.32.3"
        Append "server 195.70.32.4"
        DefineInGroup "xntpreconf"
      EndGroup
    }
```

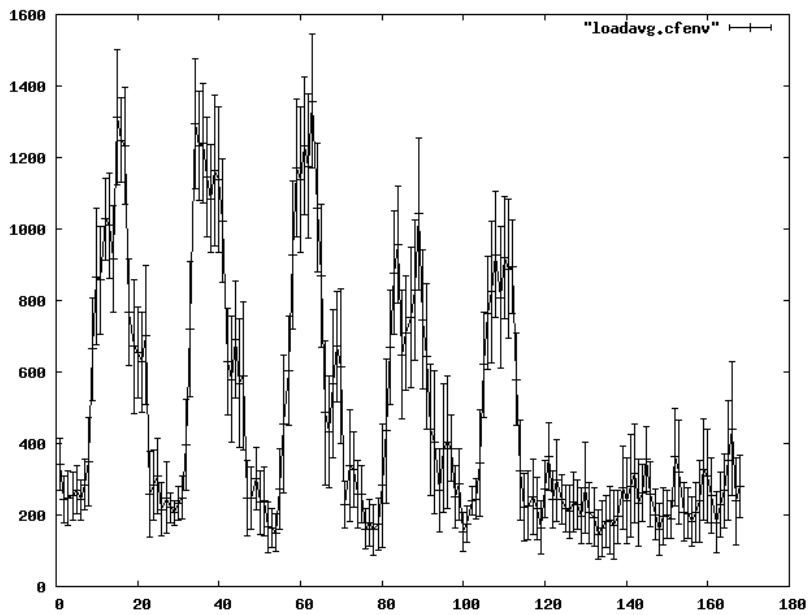
Mint látjuk, ha kellett valamit módosítani a konfigokon, újra is indítja nekünk az *ntpd*-t. Kis kényelmi szolgáltatásként, ha nem talált felismerhető változatot belőle, figyelmeztető üzenetet jelenít meg, így viszonylag hamar pótolhatjuk, mielőtt szanaszét mászkálnának a rendszeróráink.

3. Értékelés

3.1. Kb. mínusz húsz fok és eső várható – időjárásjelentés cfenvd módra

A *Cfengine* legtudományosabb darabját mutatnám be, ez a *cfenvd*, ami működési adatokat gyűjt a rendszerről, és ezekből készít olyan statisztikát, amit utána a *cfagent* felhasználhat a döntéseihez. Sajnos nem teljesen kiforrott, és a félbemaradás benyomását kelti az, hogy ha nagyjából bármit akarunk konfigurálni rajta, ahhoz a forrásba kell belenyúlni, ami ugyan nem bonyolult, ámde mégis kevésbé elegáns. Mindazonáltal úgy, ahogy van, el lehet kezdeni használni, és ha ráéreztünk az anomáliadetekció ízére, testre szabhatjuk, ha pedig nem, félre is tehetjük, a szoftver többi része e nélkül is működőképes marad.

A *cfenvd* azon feltételezésből indul ki, hogy a számítógépek terhelési adataiban (1. ábra) megfigyelhető egy egyhetes periódus. Megfigyelhetők ugyan más periódusok is, de kb. az egyhetes a legkisebb, amivel már érdemes dolgozni. Az ábrán egy szerver terhelési adatai láthatók: a vízszintes tengelyen az órák, a függőlegesen a *load average* %-os értéke, a szórással együtt. A *cfenvd* működési elve – komoly matematikai alapozással, természetesen – nagyjából az, hogy ennek az egyhetes periódusnak minden pillanatára kiszámolja az adott érték hosszú távú átlagát és szórását, majd az átlagtól való számottevő (egy, két vagy három szórásnyi) eltérést tekinti szokatlan viselkedésnek. A mért értékek és ezen szokatlansági mutatók alapján újabb osztályokat definiál a cfagent számára, amely így reagálni tud a bekövetkezett változásokra.



1. ábra. A cfenvd által gyűjtött statisztika a számítógépek terhelési adatairól

3.2. Hogyan tovább?

A *Cfengine* roppant sokoldalú eszköz, tudásának csak egy része került itt bemutatásra, azok a funkciók, amelyekre általános szerverüzemeltetés során szinte biztosan igény lesz. További funkciók, tippek, trükkök, publikációk találhatók a weboldalon [1] és a wikiben [2].

Hivatkozások

[1] Cfengine. URL `http://www.cfengine.org/`.

[2] A Cfengine wikije. URL `http://cfwiki.org/`.

Elosztott webalkalmazások kialakítása Linux és Java alapokon

Karóczkai Krisztián

Kivonat

A web meghódította a világot, ez vitathatatlan. A megnövekedett felhasználói igényeknek mind szolgáltatás, mind teljesítmény oldalról meg kell felelniük korunk webalkalmazásainak. Napjainkban a legtöbb nagyszámú felhasználót kiszolgáló webalkalmazás már egy elosztott, több rétegű rendszer, amely fizikailag is különálló szervereken fut. Az ilyen nagy teljesítményű webes rendszerek kialakítására, a környezeti szolgáltatásokat is figyelembe véve, a mérések alapján az egyik legalkalmasabb a Linux–Java páros.

Az előadás során kialakításra kerül egy Java webes környezet a rendelkezésre álló legelterjedtebb ingyenes eszközök használatával. A létrehozott webes rendszer fokozatos bővítésével képessé tesszük a rendszerünket különféle elosztott rendszerekkel való kapcsolatfelvételre, távoli metódushívásokra. Az így kialakított webes frontend az előadás végére képes lesz XML-RPC, SOAP, RMI, RMI-IIOP (EJB) alapokon különféle műveleteket végezni távoli gépeken, majd az eredményeket megjeleníteni egy webes felületen szabványos technológiák alkalmazásával. Az előadás során kialakításra kerül továbbá többféle webszolgáltatás-implementációt használó webalkalmazás, RMI és EJB konténer. Bemutatásra kerülnek a szükséges konfigurációs beállítások, telepítési módok, és néhány Java nyelvi elem, amelyek segítségével hozzá tudunk férni ezekhez a szolgáltatásokhoz webalkalmazásunkból.

Tartalomjegyzék

1. Miért web, és miért Java?

Korunkban a webes alkalmazások egyre nagyobb népszerűségnek örvendenek. Egyes alkalmazási területeken már szinte csak web alapú megoldásokkal találkozunk. Mára már ezek a webes rendszerek megjelentek olyan felhasználási területeken is (pl. irodai alkalmazások), amiket igazából a vastagklienses desktop alkalmazások világának gondolna az ember.

A webes rendszerek nem csak a felhasználók, hanem ezen rendszerek üzemeltetői körében is nagy népszerűségnek örvendenek, mivel ezeket könnyebb karbantartani, frissíteni és üzemeltetni. Elég a funkciókat egy helyen, a kiszolgálón elvégezni.

Mivel ezek központosított rendszerek, nagyon fontos a teljesítmény; ugyanis egy időben nagyszámú felhasználót kell kiszolgálniuk, ezért megvalósításukkor olyan platformot kell választani, ami kellőképpen nagy terhelhetőséget, méretezhetőséget biztosít. Az operációs rendszerek közül nagyvállalati környezetben is a legnépszerűbb platform a Unix/Linux, köszönhetően teljesítményének és megbízhatóságának. Alkalmazásfejlesztési platformok terén azonban már korántsem ilyen egyszerű a döntés.

Kezdetben a Java nyelvet nem szerveroldali alkalmazások készítésére szánták, de mára már a Java servlet a webes világ egyik legjobb teljesítményt nyújtó kiszolgálóoldali technológiájává nőtte ki magát. Kellett ehhez a nyelvi és a futtatókörnyezetben bekövetkezett fejlődés, és a Java kapcsán már-már túlzottan is hangoztatott platformfüggetlenség: ahhoz, hogy a kész program futtatható legyen, különböző felépítésű számítógépeken, a fordítóprogram a forrás programot nem a processzor gépi kódjára, hanem egy virtuális gép (JVM) gépi kódjára fordítja le. Az így létrejött közbülső kódot az alkalmazott virtuális gép hajtja végre.

A korai, egyszerű JVM implementációk általában interpreteres megoldást alkalmaztak, modern változataik azonban a jóval nagyobb futtatási sebesség elérését lehetővé tevő JIT (*just in time*, fordítás közvetlenül a futtatás előtt) technikát használják. A JIT-es fordítók az interpretált nyelveken készült alkalmazások futtatását gyorsítják fel olyan módon, hogy végrehajtás előtt elvégzik a program forráskódról tárgykódra történő átalakításának folyamatát. Így a program a tényleges végrehajtás során már natívan, gépi kódban működik, és a végrehajtásához nincs szükség az időigényes értelmezési folyamatra. A másik gyorsítási lehetőség a Java Hotspot technikája. Ez a technológia a felhasználó „szokásait” veszi figyelembe. Egy adaptív algoritmussal tehát vizsgáljuk a kódrészletek használatát, és a leggyakrabban használt kódrészt és környezetet feltételezve optimalizálunk.

A fejlesztési környezet kiválasztásánál fontos szempont, hogy az adott környezethez milyen programkönyvtárakat kapunk, valamit a nyelv és a környezet milyen támogatásokat biztosít a fejlesztők számára munkájuk mielőbbi, minél kevesebb hibával történő implementálásához. Jelenleg a legtöbb fejlesztési projekthez irreálisan kevés időt adnak a megvalósításra, ezért sok nagyon kiváló C és C++ programozó dönt a platform elhagyása mellett, és képezi át magát Java vagy .NET fejlesztőnek, hogy meg tudjon felelni a minőségi munka mellett a rövid fejlesztési határidőknek is.

2. Egy fejlesztői/futtatói Java környezet kialakítása

A virtuális gépet telepíteni kell a programot futtató gépre a rendszer könyvtárakkal együtt. Bár ez a rendszer nem nyílt forrású, és nem is szabad, de ingyen letölthető [2].

Vannak olyan Java fordítóprogramok, amelyek natív gépi kódra fordítják le a forráskódot, ilyen például a GCJ [1], így valamelyest felgyorsítják a futtatást, de ugyanakkor a lefordított program elveszti hordozhatóságát. Továbbá a GCJ nem teljesen kompatibilis a Sun-féle JVM-mel.

Ahhoz, hogy egy Java-alkalmazást el tudjunk indítani a saját gépünkön, csak:

1. A futtatókörnyezetet kell letölteni.

2. Futtatási jogot adni rá.
3. Elindítani, ekkor automatikusan kitömöríti magát.
4. A `JAVA_HOME` környezeti változót beállítani arra az értékre, ahová telepítettük a környezetet.
5. És végül a `PATH`-hoz hozzá kell adni a `$JAVA_HOME/bin`-t.

Ha fejleszteni szeretnénk vagy szervereket üzemeltetni, akkor nem elég a futtatókörnyezet, mert az nem tartalmazza a fordításhoz szükséges eszközöket. Ebben az esetben a J2SE-t (Java 2 Standard Edition) kell letölteni. A fejlesztéshez szükségünk lesz egy fejlesztői környezetre is. Ebből több is a rendelkezésünkre áll, a legelterjedtebb a NetBeans, amit a többi komponenshez hasonlóan csak le kell tölteni, és el kell indítani a telepítéshez.

Bár a NetBeans alapból tartalmaz egy webszervert is a fejlesztés támogatáshoz, éles alkalmazásaink futtatáshoz a szerverünkön nekünk kell kialakítanunk ezt a környezetet. Java alapú webes szolgáltatásokhoz egy Java-képes webszervert kell telepítenünk. Jelenleg a legelterjedtebb nyílt forrású implementáció a Tomcat. A Tomcat egy Java servlet-konténer, ami képes futtatni minden olyan Java webes alkalmazást, ami a servlet technológiára épül. Használatbavételéhez letöltés után:

1. Ki kell csomagolni.
2. A `JAVA_HOME` környezeti változót a J2SE telepítési könyvtárára állítani.
3. A `CATALINA_HOME` környezeti változót arra könyvtárra kell állítani, ahová a Tomcat-et kicsomagoltuk.
4. A Tomcat alatti `bin` könyvtárban levő `startup.sh` állománnyal már el is lehet indítani.

3. mod_jk: Tomcat és Apache összekapcsolása

Általában azonban nem az az elterjedt megoldás, hogy egy szerveren egy Tomcat van telepítve – mivel az csak Java servlet és statikus webtartalom kiszolgálására alkalmas – hanem az, hogy egy általánosabb webszerver, mint például egy Apache. Az Apache nem egy Java servlet-konténer, tehát alapból nem képes servletek futtatására. Létezik egy mod_jk nevű modul az Apache-hoz, ami képes együttműködni egy, már a gépen lévő Tomcattel. A mod_jk segítségével az Apache megadott szintaktikára illeszkedő URL-ekre irányuló kéréseket a Tomcathez fogja iránytani feldolgozásra. Ehhez egy APJ protokollt fog használni. Mivel a mod_jk forrásban érhető el, azt nekünk le kell fordítani, amihez az Apache oldaláról szükség lesz a *-devel csomagok telepítésére, melyek segítségével Apache-modulok fordítására leszünk képesek.

A mod_jk fordítása a szokott módon történik. A forrás kicsomagolása után:

```
cd native
./configure --with-apxs=/usr/sbin/apxs
make
su -c 'make install'
```

A mod_jk-hoz szükség van egy konfigurációs állományra. Ebben az állományban kell definiálni a Tomcat és a Java-környezet elérési útját. Ezenkívül itt van lehetőségünk ún. *worker*ek definiálására, és ezekhez a workerekhez fog az Apache kapcsolódni. Egy iworkernek van hostja, portja, képes cache-t kezelni és kérésekhez időtúllépést is tud kezelni. Példa konfiguráció:

```
workers.tomcat_home=/opt/tomcat
workers.java_home=/opt/java
worker.list=worker1
worker.worker1.type=ajp13
worker.worker1.host=localhost
worker.worker1.port=2709
worker.worker1.lbfactor=50
worker.worker1.cachesize=10
worker.worker1.cache_timeout=600
worker.worker1.socket_keepalive=1
worker.worker1.reclycle_timeout=300
```

A Tomcat-hez tartozó `server.xml` fájlba fel kell vennünk a AJP-t, mint connectort. A connector portnak meg kell egyeznie a mod_jk konfigurációjában definiált porttal. Például:

```
<Connector port="2709" enableLookups="false"
  redirectPort="8443" protocol="AJP/1.3" />
```

Ezek után már csak az Apache `http.conf` fájljában kell beállítani az APJ használatát az elkészített konfigurációs állomány megadásával. A naplózási formátum és szint definiálása is hasznos funkció, amit szintén itt tudunk beállítani. A `JkMount` direktíva szolgál azon URL-ek definiálására, melyekre illeszkedő kéréseket a Tomcathez fog küldeni az Apache. Példa beállítás:

```
LoadModule    jk_module extramodules/mod_jk.so
JkWorkersFile /opt/tomcat/conf/jk/workers.properties
JkLogFile     /var/log/apache2/mod_jk.log
JkLogLevel    debug
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
JkRequestLogFormat     "%w %V %T"
JkOptions     +ForwardKeySize +ForwardURICompat -ForwardDirectories
JkMount  /*jsp worker1
JkMount  /*portal worker1
JkMount  /*service* worker1
```

Ha a Java-alapú webalkalmazásaink könyvtárai elérhetőek az Apache-on belül is, akkor biztonsági okok miatt érdemes letiltani az `*-INF` könyvtáraink elérését:

```
<Directory WEB-INF>
  Deny from all
</Directory>
<Directory META-INF>
  Deny from all
</Directory>
```

4. A Java webes lehetőségei kiszolgálóoldalon

A Java nyelv sokszínűségét bizonyítja, hogy webalkalmazásaink többféle nyelvi szabványt megvalósítva, akár teljesen különböző eszközök és eljárások alkalmazásával is implementálhatják ugyanazokat a műveleteket. Legyen szó egyszerű HTML-kimenetet generáló osztályokról, komponensalapú fejlesztésről, vagy akár szabványos portálrendszerekről. Ezek elkészítése és Tomcat alatti futtatása némi gyakorlással könnyedén elsajátítható.

4.1. Servletek

A Java servletek olyan speciális Java osztályok, amik egy servlet-konténeren belül futtathatóak, és annak képességeit egészítik ki. A servlet-osztályok szorosan együttműködnek a konténerükkel, velük egy folyamatban, külön szálban futnak. Ez a szoros együttműködés az egyik oka a teljesítményüknek. Ugyanis minden információ a folyamaton belül rendelkezésére áll a kiszolgáló servletnek. Ráadásul mindez akár azonos virtuális gépen belül is megvalósítható. Ha a webszerverünk egy olyan kérést kap, amit egy servletnek kell kiszolgálni, akkor megvizsgálja, hogy van-e szabad, már létrehozott servletpéldány, ami ezt a kiszolgálást el tudja végezni. Ha van, akkor továbbítja felé a kéréshez tartozó paramétereket. Ha nincs, akkor előbb létrehoz egy új példányt. A webalkalmazásban elérhető servleteket a webalkalmazásunk `web.xml` konfigurációs állományában tudjuk beállítani. A beállítás két lépésben történik. Az elsőben egy álnevet definiálunk a servlet-osztályunkhoz:

```
<servlet>
  <servlet-name>linuxconfservlet</servlet-name>
  <servlet-class>hu.linux.conf.servlet</servlet-class>
</servlet>
```

A következőben pedig a megadott álnevet egy URL-hez csatolunk:

```
<servlet-mapping>
  <servlet-name>linuxconfservlet</servlet-name>
  <url-pattern>/servleturl</url-pattern>
</servlet-mapping>
```

Akkor célszerű servleteket alkalmazni,

– ha az általa megvalósított funkciók egyáltalán nem, vagy csak nagyon ritkán változnak;
– ha a webes megjelenítési funkciók minimálisak, vagy teljesen lényegtelenek, statikusak és az alkalmazás funkcionalitása kimerül a servletek által megvalósított funkciókban.

4.2. JSP

A servletek nagy hátránya, hogy a generált HTML-kimenethez meg kell változtatni a forrást; lefordítani, feltölteni, és a változásokat a webalkalmazásunkban aktualizálni. A JSP-lapok a servlet-technológiára épülnek, azonban egyszerű szöveges formátumban vannak jelen a webszerveren, és az első hívás alkalmával a Tomcat automatikusan legenerálja belőle a servletforrást és lefordítja azt, majd ezek után már csak a servletet futtatja. Ha a JSP-fájl tartalmában változás történt, akkor a Tomcat megint automatikusan elvégzi a már említett teendőket. Így a JSP-lapok kezelése sokkal kényelmesebb a servletekénél. Ráadásul a JSP-lapjainkon használhatunk ún. tag-könyvtárakat, amik megadott tevékenységeket végeznek el. Az elnevezés abból adódik, hogy a tag-könyvtár új XML tageket definiál, melyeket a JSP-lapon elhelyezve elérhetjük a könyvtár funkcióit. Gyakori, hogy a tag-könyvtár és a JSP-lap szerzője különbözik. Internetről is tölthetünk le tag-könyvtárakat.

A tag-könyvtárakat szintén be kell állítanunk a `web.xml`-ben:

```
<taglib>
  <taglib-uri>/linuxconftld</taglib-uri>
  <taglib-location>/WEB-INF/tlds/linuxconf.tld</taglib-location>
</taglib>
```

JSP-t akkor célszerű használni, ha a webes megjelenítés teljesen leválasztható az alkalmazás logikájáról, az alkalmazás elemei elérhetőek JavaBeanek vagy tag-könyvtárak segítségével.

4.3. JSF

A JSF webes rendszerekben is magas fokon képes támogatni az MVC (modell–nézet–vezérlő) modellt és a komponensalapú fejlesztési nézetet. Ennek köszönhetően segíteni tudja a fejlesztők munkáját az így könnyebben létrehozható, testreszabható magas szintű integrált fejlesztőkörnyezetek (IDE) alkalmazásával.

A JSF egy szerveroldali, komponens alapú felhasználóifelület-keretrendszer, webes és általános környezetre. Felépítése független a kliensalkalmazástól, azaz ugyanazt az alkalmazáslogikát használhatjuk webes, mobil és vastag kliens alkalmazásban is. A felületi elemek (címkék, szövegmezők, gombok, jelölőmezők) állapottal rendelkeznek a szerver oldalon, amely állapot az események feldolgozásakor az MVC-ben elvárt módon mindig a megfelelő lesz. A felületi komponensek állapota, eseménymodellje és a megjelenítési környezet jól specifikált. A JSF használatának előnye leginkább bonyolultabb oldalak szerkesztése során jön elő [3].

A JSF-környezetet a webalkalmazásunk alá kell telepíteni. A `WEB-INF/faces-config.xml` fájlban konfigurálhatjuk a JSF-et. Lehetőségünk van ellenőrzések, navigációs szabályok és beanek definiálására, melyeket a JSF-oldalainkon tudunk használni. Példa beállítás:

```
<validator>
  <validator-id>azonosító</validator-id>
  <validator-class>osztály</validator-class>
  <property>
    <property-name>tulajdonság</property-name>
    <property-class>tulajdonság típusa</property-class>
  </property>
</validator>
<navigation-rule>
  <navigation-case>
    <from-action>művelet</from-action>
    <from-outcome>kimenet</from-outcome>
    <to-view-id>megjelenő oldal</to-view-id>
    <redirect/>
  </navigation-case>
</navigation-rule>
<managed-bean>
  <managed-bean-name>név</managed-bean-name>
  <managed-bean-class>osztály</managed-bean-class>
  <managed-bean-scope>érvényesség</managed-bean-scope>
</managed-bean>
```

A JSF-et ott érdemes használni, ahol JSP-lapok nagy számban kerülnek alkalmazásra, nem kizárólag webes klienseket kiszolgáló rendszerek esetén. Ha az alkalmazott fejlesztőkörnyezetünk támogatja, akkor ezáltal lerövidíthető a fejlesztési idő.

4.4. Portlet

A webes keretrendszerek nem csak Java, hanem más programozási környezetekben is nagy népszerűségnek örvendenek. Az ilyen rendszerek alkalmazásával az ember kap egy jól működő alapot saját problémájának webes megoldására. Sok esetben a keretrendszerekhez kapott kiegészítő modulok segítségével el is végezhető a szükséges feladatok nagy része. Azonban ezek a modulok csak az adott rendszeren belül használhatóak, s ezért nem vagyunk képesek a moduljaink alatt lecserélni a keretrendszert, ha esetleg nagyobb teljesítményre lenne szükségünk. Ráadásul az ilyen modulokat minden keretrendszerhez másképpen kell illeszteni, felépíteni. Így könnyen előfordulhat, hogy ugyanazt a problémát eltérő keretrendszerek alkalmazása esetében részben, vagy egészben újra kell implementálni. A Java-s világban erre

adnak megoldást a szabványos JSR-168-as portletek. Ez egy szabvány, amit sorra implementáltak az egyes portálkonténerekbe, és ezáltal a webes portálok moduljai szabványosan épülhetnek fel. Amikor létrehozunk egy portletet, használat előtt el kell helyezni róla egy definíciós bejegyzést az őt tartalmazó webalkalmazás `portlet.xml` fájljába. Például:

```
<portlet>
  <description xml:lang="hu">Portlet leírás</description>
  <portlet-name>portletneve</portlet-name>
  <display-name xml:lang="hu">Portlet display</display-name>
  <portlet-class>portlet osztály</portlet-class>
  <expiration-cache>cache time out</expiration-cache>
  <supports>
    <mime-type>text/html</mime-type>
  </supports>
  <supported-locale>hu</supported-locale>
  <portlet-info>
    <title>címke</title>
    <short-title>rövid cím</short-title>
    <keywords>kulcsszavak</keywords>
  </portlet-info>
</portlet>
```

Ezt követően már a portletkonténer ismeri a portletünket, a portálunkon való elhelyezéséről egy konténerspecifikus állományban rendelkezhetünk.

Portletek alkamazása ajánlott nagy bonyolultságú portálok, webalkalmazások esetében, melyek funkciói könnyen kisebb szerkezeti egységekre bonthatóak.

5. Elosztott rendszerek Java alapon

A webes technológiák fejlődésének köszönhetően a webalkalmazások egyre okosabbak, sokoldalúbbak és felhasználóbarátabbak lettek. Ennek következtében egyre több alkalmazást ültetnek át webes felületre, és ezeket mind több felhasználó használja. A komolyabb rendszerek már annyi felhasználót szolgálnak ki, hogy egy szerver nem is bírja elvégezni a feladatokat. Ebben az esetben az alkalmazásunknak is képesnek kell lennie több gépen futni. Ez akkor kivitelezhető, ha az alkalmazás egyes részei, funkciói elkülöníthetőek a teljes rendszertől, mégpedig úgy, hogy a felhasználó ebből semmit sem vesz észre. Ebben az esetben azonban már egy elosztott rendszerről beszélünk. Jelenleg több olyan technológia is van, amely segítségével ezeken rendszerek részegységei egymással kommunikálhatnak, adatokat, objektumokat cserélhetnek. A fejlesztők előtt rejtve marad a hálózati kommunikációt, tehát csak az alkalmazáslogikával kell foglalkozniuk.

5.1. RMI

Az RMI egy, csak Java környezetben létező megoldás távoli metódusok hívására. A módszer lényege, hogy készítenünk kell egy Java-s interfészt aminek elérhetőnek kell lennie mind kliens, mind szerver oldalon. Szerveroldalon szükségünk lesz egy osztályra, amely implementálja az interfészt. Ezt egy adott névvel be kell jegyeznünk az rmiregistry-be. Az rmiregistry-t a Java részeként kapjuk meg. A kliensoldalon csak azt kell megadnunk az alkalmazásunkban, hogy melyik host milyen néven bejegyzett objektumára lenne szükségünk. A kliensoldalon az interface alapján létrehozott objektum metódusait meghívva kiváltódik a szerver oldalon a hívás, majd az eredmény a kliensoldalra másolódik. A módszer nagy előnye, hogy különböző adattípusok(objektumok) is átadhatóak a hívás során, illetve kaphatóak vissza is.

Az RMI alkalmazása ajánlott Java alapú elosztott rendszerek esetében.

5.2. EJB 2.x

Az EJB-t komponensalapú üzleti alkalmazások támogatására találták ki. Használatához egy külön EJB-konténer szükséges, ezen szerveren belül jönnek létre az *enterprise bean*ek, kérésre itt hajtódnak végre. Egy enterprise beannek tartalmaznia kell a specifikációban meghatározott metódusokat és a kliensek felé jól definiált interfészeket kell nyújtania. Ezek telepítéséhez a specifikáció definiál egy `ejb-jar.xml` fájlt, amiben le kell írnunk a bean struktúráját, típusát és a konténertől igénybe vett szolgáltatásokat. Ezek a beállítások különböző beantípusok esetében eltérőek. Példa beállítás:

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC
  '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
  'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
  <enterprise-beans>
    <!-- A minimal session EJB deployment -->
    <session>
      <ejb-name>PostingEJB</ejb-name>
      <home>ejbs.PostingHome</home>
      <remote>ejbs.Posting</remote>
      <ejb-class>ejbs.PostingBean</ejb-class>
      <!-- or Stateless -->
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  <assembly-descriptor></assembly-descriptor>
</ejb-jar>
```

Ezek után már csak a konténerspecifikus állományokat kell definiálni, amik sajnos konténerenként eltérőek. Egy ilyen konténert (esetünkben OpenEJB) webes felhasználásra össze tudunk kapcsolni a Tomcatünkkel, ha a webalkalmazásunk `web.xml` fájljába elhelyezzük az alábbi servlet-bejegyzést:

```
<servlet>
  <servlet-name>loader</servlet-name>
  <servlet-class>org.openejb.loader.LoaderServlet</servlet-class>
  <init-param>
    <param-name>openejb.loader</param-name>
    <param-value>tomcat-webapp</param-value>
  </init-param>
  <init-param>
    <param-name>openejb.home</param-name>
    <param-value>...define OPENEJB_HOME here...</param-value>
  </init-param>
  <load-on-startup>0</load-on-startup>
</servlet>
```

Nemcsak Java alapú elemeket tartalmazó nagy bonyolultságú elosztott rendszerek esetében ajánlott EJB-t alkalmazni, hanem olyan esetekben is, amikor az RMI már nem képes megfelelő szolgáltatásokat nyújtani, például nagy mennyiségű adatokkal dolgozó „üzleti” rendszerek esetében.

5.3. EJB 3.x

Az EJB 3.0 célkitűzése az volt, hogy leegyszerűsítse a programozási feladatokat. Ennek érdekében új nyelvi elemek kerültek bevezetésre. Ezeket az új elemeket azonban még nem minden konténer ismeri, számuk azonban folyamatosan növekszik.

Bárhol, ahol EJB 2.x-et alkalmaznánk, ajánlott az EJB 3.x, de rendelkezésünkre kell állni egy olyan kiszolgálói környezetnek, ami ismeri a 3.x-es specifikációt.

5.4. Apache XML-RPC

A legegyszerűbb webszolgáltatás-implementáció. A megvalósításánál végig az egyszerűség volt a mérvadó, ez az egyszerűség vezetett a nyújtandó szolgáltatások, átvivendő adattípusok minimális számához. Az alapkönyvtárat be kell építenünk egy webalkalmazásba, amihez még egy plusz servletet is készíteni kell, ha webkonténerünkből akarjuk használni.

Olyan elosztott rendszerek esetében javasolt használni, ahol csak HTTP-alapú adatátvitel valósítható meg, a hálózaton csak alapvető objektumokat akarunk átvinni, illetve fontosabb a kommunikációs protokoll teljesítménye, mint az általa nyújtott szolgáltatások.

5.5. Apache Axis 1.x

A webszolgáltatások körében a bonyolultabb műveletek elvégzésére SOAP-alapú megoldások is használhatók. A SOAP előnye az XML-RPC-vel szemben, hogy többféle típust támogat, és nem csak HTTP környezetben használható. Ezen plusz szolgáltatásokért azonban teljesítményvesztéssel kell fizetnünk. A legelterjedtebb SOAP implementáció az Axis. Az Axist egy webalkalmazásként tudjuk telepíteni a webkonténerünkbe. A letöltés után csak ki kell csomagolni a `webapps` könyvtárba. Az Axis mögött is servletek lapulnak. Példa konfiguráció:

```
<servlet>
  <servlet-name>AxisServlet</servlet-name>
  <display-name>Apache-Axis Servlet</display-name>
  <servlet-class>org.apache.axis.transport.http.AxisServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>AxisServlet</servlet-name>
  <url-pattern>/servlet/AxisServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>AxisServlet</servlet-name>
  <url-pattern>*.jws</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>AxisServlet</servlet-name>
  <url-pattern>/services/*</url-pattern>
</servlet-mapping>
```

Axis alatt többféleképpen lehet szolgáltatásobjektumot definiálni. Az egyszerűbb változat, amikor az objektum forrását `.jws` kiterjesztéssel elhelyezzük a webalkalmazásunk publikusan elérhető részén. Az első hívás alkalmával az Axis lefordítja a forrást. A másik lehetőség, amikor mi helyezünk el lefordított osztályt a `WEB-INF` könyvtáron belül, ekkor azonban vagy webes vagy konzolos felületen az Axis tudtára kell adnunk, hogy az adott osztály egy webszolgáltatás-objektum.

A konzolos beállítás esetében egy egyszerű WSDD-állományt kell szerkesztenünk. Ebben az állományban kell megadnunk a szolgáltatásra vonatkozó adatokat:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
 xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="szerviz neve" provider="java:RPC">
    <parameter name="className" value="osztály"/>
    <parameter name="allowedMethods" value="elérhető_metódusok"/>
    <parameter name="scope" value="request"/>
  </service>
</deployment>
```

A fájlt az AdminClient segítségével tudjuk az Axisba bejegyezni:

```
java org.apache.axis.client. AdminClient \
  -lhttp://host:port/webapp/servlet/AxisServlet service.wsdd
```

Kliensoldalon a hívó osztályban csak az URL-t, a hívandó metódust és a paramétereket kell megadnunk.

Olyan helyeken célszerű az Axist alkalmazni, ahol már az XML-RPC nyújtotta szolgáltatások és adattípusok kevésnek bizonyulnak, de nem igényelünk még EJB-szintű megoldásokat.

5.6. Axis 2.x

Az Axis 2-es verziója az 1-eshez hasonlóan SOAP-alapú, a két verzió között elsősorban programozástechnikai és szolgáltatásbeli különbségek vannak. Természetesen ebben az esetben is a több szolgáltatást nyújtó 2.x-es verzió a lassabb; nagyságrendileg feleakkora teljesítményre képes, mint az 1.x-es verzió. A két verzió fejlesztése párhuzamosan halad, tehát a 2.x-es nem váltotta fel az 1.x-est.

Az Axis 2.x ajánlott nagy bonyolultságú webes szolgáltatások, hitelesítéses fájlmozgatás esetében, vagyis olyan helyeken, ahol nem a teljesítményen, hanem a fejlesztést megkönnyítő szolgáltatásokon van a fő hangsúly.

5.7. JAX-RPC

A JAX-RPC szintén SOAP- és WSDL-alapú távoli metódushívásra alkalmas eljárás. A rendszer teljesen elrejti a SOAP-ot a fejlesztők elől. A JAX-RPC a webszolgáltatásokat statikus stubokkal (távoli szolgáltatásokat reprezentáló objektumok) és dinamikus proxykkal (futásidőben generált objektumok) is igénybe tudja venni. A JAX-RPC-t a WSDP (Web Service Developer Pack) csomagban érhetjük el legkönnyebben. Ebben a csomagban egy előre konfigurált Tomcatet kapunk a JAX-RPC használatához.

A JAX-RPC ajánlott nagybonyolultságú webszolgáltatások esetében, melyek már szinte EJB-szintű megoldásokat tartalmaznak.

5.8. Hessian

A fő eltérés az eddig tárgyalt webszolgáltatás-implementációktól, hogy míg azok szöveges alapú protokollt használnak, addig a Hessian binárisat. A Hessian használatához is szükségünk van egy webkonténerre. A konténerünk webalkalmazását a szükséges `.jar` fájlok telepítésével és a `web.xml` módosításával tudjuk képessé tenni ilyen kommunikációra. Egy erőforrásként kell definiálni az objektumot, amit szervizként akarunk megosztani Ez a Resin webszerver alatt így néz ki:

```
<resource jndi-name="service/greeting" type="example.GreetingImpl">
  <init>
    <greeting>Hello, web.xml</greeting>
```

```
  </init>
</resource>
<servlet url-pattern="/hessian/greeting"
         servlet-class="com.caucho.hessian.server.HessianServlet">
  <init>
    <home>${jndi("service/greeting")}</home>
    <home-api>example.GreetingAPI</home-api>
  </init>
</servlet>
```

Kliensoldalon hasonlóan működik a rendszer, mint a már ismertett webszolgáltatások.

Ha mindenképpen valamilyen webszolgáltatást kell használnunk, de valamilyen okból nem alkalmazhatunk szöveges alapú kommunikációt (pl. azért, mert fontos, hogy kis mennyiségű adat utazzon a hálózaton), akkor a Hessian ajánlott.

6. Összegzés

Amint látható, a Java nyelv kiváló kiindulási alapot jelent elosztott webes alkalmazásaink fejlesztéséhez, hiszen nem csak maga a nyelv, hanem az elérhető eszközök is nagy fokon támogatják az ilyen alkalmazások létrehozását. Az előadás során konkrét példákon keresztül bemutatásra kerültek az egyes technológiák és az azok használatát segítő alkalmazások.

Hivatkozások

[1] A GCJ Java-fordító. URL `http://gcc.gnu.org/java/gcj2.html`.

[2] A Sun-féle Java 2 Standard Edition Java virtuálisgép-implementáció letöltése. URL `http://java.sun.com/javase/downloads/index.jsp`.

[3] Zsemlye Tamás – Soós István: *Kreáljunk egy kis webet!* Elhangzott a *Magyarországi Web Konferencián*. 2006. URL `http://web.conf.hu/2006/program/i/krealjunkwebet`.

Webes alkalmazásfejlesztés OpenLaszlo keretrendszer segítségével

Kecskeméti László

Szél Miklós

Kivonat

Az OpenLaszlo egy olyan nyílt forrású fejlesztői keretrendszer, amely hatékony webes GUI-fejlesztést tesz lehetővé Flash kimenetet generálva. Fordításkor meghatározhatjuk, hogy melyik Flash verzióra szeretnénk fordítani, így nem jelent problémát a nyílt forráskódú rendszerek alatti Flash verziólemaradás sem. Emellett a LaszloSystems ez év végére ígéri a teljes körű DHTML kimenet támogatását (ez jelenleg béta állapotban van). Előadásunkban demonstráljuk ezen rendszer képességeit egy, a bemutató alatt fejlesztett zenelejátszó alkalmazással, mely a Music Player Daemon nevű nyílt forrású zenelejátszó programra épül. A webes felület backendjéül egy Perl-modul szolgál, mely az MPD oldaláról tölthető le. A felület a népszerű XMMS/BMP zenelejátszók alapján készül el, az előadás időkorlátja miatt csökkentett funkcionalitással (fájllista, lejátszás, léptetés stb.).

Tartalomjegyzék

1. OpenLaszlo

Az OpenLaszlo [4] egy olyan nyílt forrású fejlesztői keretrendszer, amely hatékony webes GUI-fejlesztést tesz lehetővé Flash kimenetet generálva.

A keretrendszer előnye, hogy gyakorlatilag teljesen hordozható alkalmazásokat lehet vele készíteni. A fordítóprogramban meghatározhatjuk, hogy melyik Adobe Flash verzióra szeretnénk az alkalmazást fordítani (legfrissebb stabil verziója Adobe Flash 6-7-8 kimenetet tud generálni), így nem jelent problémát a nyílt forráskódú rendszerek alatti Flash verziólemaradás sem. Azok megnyugtatására, akik nem szimpatizálnak a Flash alapú alkalmazásokkal, az OpenLaszlo-t fejlesztő Laszlo Systems [3] bejelentette, hogy a hamarosan megjelenő új verzió már DHTML-kimenetet is támogat majd.*

2. A szerveroldali alkalmazás

A Music Player Daemon (MPD) [2] a zenelejátszás helyi hálózatról történő vezérlését teszi lehetővé. Támogatja az MP3, Ogg Vorbis, FLAC, AAC, Mod és WAV fájltípusokat, a lejátszási listák betöltését, elmentését. Különösen előnyös lehet a daemon irodai használata, ahol több embernek van beleszólása abba, hogy milyen zene is szóljon a közös légtérben.

Az MPD és a vezérlőfelület közötti adatkapcsolatot többféle szerveroldali programnyelven is (PHP, Perl, Python stb.) megvalósíthatjuk, mi a Perlt választottuk. A CPAN tartalmaz egy Audio::MPD nevű modult [1], amelynek példaprogramját minimálisan módosítva hoztuk létre az adatkapcsolót az OpenLaszlo és az MPD között.

A kis Perl-rutin lényegében státuszinformációkat ad át az OpenLaszlonak XML formátumban, valamint az OpenLaszlo-tól érkező parancsokat dolgozza fel és továbbítja az MPD felé.

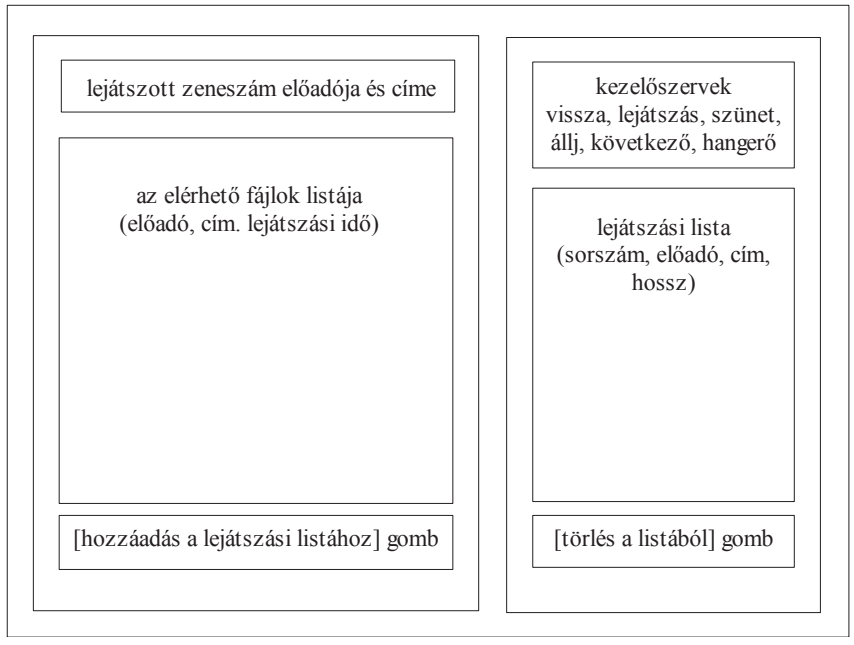
Az előadás időkorlátja miatt csökkentett funkcionalitással bíró felület csak a zenelejátszáshoz szükséges legfontosabb funkciókat ismeri majd (fájllista, lejátszás, szünet, léptetés stb.), ezért az adatkapcsolatért felelős programocska is csak az alábbi funkciókat fogja nyújtani:

1. *listall:* listázza a zenei adatbázist (előadó, számcím, idő, fájlnév);
2. *playlist:* listázza a lejátszási listát;
3. *add:* hozzáadja a paraméterben átadott fájlt a lejátszási listához;
4. *play:* lejátssza az első – vagy adott sorszámú – zeneszámot;
5. *del:* törli a megadott sorszámú zeneszámot a lejátszási listából;
6. *next:* a következő számra ugrik a lejátszási listában;
7. *prev:* az előző számra ugrik a lejátszási listában.

3. A felület

Számos lejátszófelület (frontend) készült már a Music Player Daemonhoz, mind a konzolhívők, mind a grafikus felület szerelmesei számára, és természetesen webes is, például a PHP és HTML alapú phpMp. Az általunk tervezett zenelejátszó felületét az 1. ábra mutatja. Az fejlesztés során sorra vesszük a vizuális megjelenítés, az adatkapcsolat felépítésének és a vezérlés kialakításának egyes lépéseit.

* A linuxos Flash 7 további hátránya, hogy magyar ékezetes ő és ű betűket nem tud fogadni a billentyűzetről – *a szerk.*



1. ábra. A készítendő zenelejátszó-felület vázlata

Hivatkozások

[1] Az MPD-t vezérlő Audio::MPD Perl-modul a cpan-en.
URL `http://search.cpan.org/dist/Audio-MPD/`.

[2] A Music Player Daemon (mpd) honlapja. URL `http://www.musicpd.org/`.

[3] Az OpenLaszlo-t gyáró cég, a Laszlo Systems weboldala.
URL `http://www.laszlosystems.com/`.

[4] Az OpenLaszlo weboldala. URL `http://www.openlaszlo.org/`.

Szoftver- és hardvernyilvántartás az OCS Inventory NG felhasználásával

Kolozs Sándor
<kolozs.sandor@szszi.hu>

Kivonat

Vállalati környezetben igen fontos a számítógépekre telepített szoftverek naprakész nyilvántartása. Ennek elhanyagolása esetén előfordulhat a kereskedelmi szoftverek jogosulatlan, a licencekkel igazolhatóan megvásárolt mennyiség fölött történő használata, vagy ennek ellenkezője is, a meglévő licencek felhasználása helyett fölösleges példányok vásárlása. Ez váratlan anyagi és akár jogi vonzatokkal is járhat. Az se hátrány, ha a rendszergazda átfogó képet kap a gépparkban alkalmazott hardverekről, ezzel előzetesen ellenőrizheti azt, hogy a kiszemelt számítógép kiépítése megfelel-e a rá telepítendő szoftver követelményeinek. Az ilyen feladatok elvégzésének megkönnyítésére használható fel az OCS Inventory NG szoftver. A program GNU GPL licencű, szabadon használható.

A központi adatbázisba egy, a munkaállomásokon futtatott kliensprogrammal vihetőek fel az elérhető információk, majd ezeket az adatokat felhasználva egy pontos leltár készíthető el. A nyilvántartás további vezetését megkönnyíti a kliensprogram állandóan futó szolgáltatásként való feltelepítése, ami garantálja az adatbázis aktualizálását.

A szoftvert egy esettanulmányon keresztül mutatjuk be. Az ismertetésre kerülő esetben a leltározni kívánt munkaállomásokon MS Windows operációs rendszer fut, az OCS Inventory NG központi rendszere pedig egy linuxos szerverre van feltelepítve.

Tartalomjegyzék

1. Informatikai rendrakás

Esetünkben egy vállalat informatikai infrastruktúráját kellett naprakésszé tenni, amely feladat többek között magába foglalta az asztali számítógépek, és az azokra telepített programok feltérképezését, az esetlegesen fölöslegesen feltelepített szoftverek eltávolítását, a szükséges alkalmazások telepítését, frissítését.

Az alkalmazottak által használt munkaállomásokon túlnyomórészt Windows XP Professional és Windows 2000 Workstation rendszerek voltak telepítve, és a felhasználók adminisztrátori joggal rendelkeztek, ennek folyományaként számos gépen akadt néhány oda nem illő szoftver.

1.1. Nagytakarítás

Első lépésként válogatás nélkül minden számítógépről letakarítottuk az alkalmazottak böngészése során begyűjtött adware és egyéb haszontalan programokat, majd frissített vírusirtó és tűzfalszoftver került a gépekre, továbbá a gépeket használó alkalmazottak adminisztrátori jogainak megvonásával előztük meg a számítógépekre a további szoftverek kontroll nélküli telepítését, használatát.

1.2. Személyre szabás

Ezután került sor annak kidolgozására, hogy az adott felhasználók munkájához milyen alkalmazásokra is van szükség. Ezzel előállt egy lista, hogy melyik számítógépre mit kell feltelepíteni, miket lehet fent hagyni, és az összes egyéb programot pedig el kell távolítani. Ezenfelül a gépek adatait is be kellett gyűjteni a meglévő leltár ellenőrzése és aktualizálása céljából.

Mivel az asztali gépekre tervezett Windows rendszerek az egyidejű többfelhasználós működést nem támogatják, a tüzetes átnézést, telepítést személyesen, esetenként VNC segítségével távolról végeztük, ezzel sajnos a gépeiket használni kívánó alkalmazottak munkáját akadályozva, ezért kerestük azt a lehetőséget, ami lehetőleg a felhasználók zaklatása nélkül, automatikusan elvégzi ezen feladatokat vagy legalább azok valamely részeit.

Így találtunk rá az OCS Inventory NG alkalmazásra, mely erre a célra egészen használhatónak tűnt.

2. Az OCS Inventory NG leltárprogram bemutatása

Az Open Computer and Software Inventory Next Generation (OCS Inventory NG) [5] egy, a rendszeradminisztrátorok munkáját segítő alkalmazás. Felhasználásával feltérképezhető a hálózatunkban lévő számítógépek hardverkiépítése, továbbá az azokra feltelepített szoftverek is, illetve nyomon követhető ezek változása. Továbbá lehetőséget ad a rendszergazdáknak szoftverek telepítésére a gép előtt ülő felhasználó beavatkozása nélkül. Az ismerkedést a dokumentáció [3] segítségével kezdtük el.

Az alkalmazás működése már meglévő szabványokon alapul, ezek a HTTP és HTTPS protokollok, illetve az XML adatformátum.

A Menedzsment szerver futtatásához szükséges Linux vagy akár Windows operációs rendszerre telepített Apache/MySQL/PHP/Perl környezet. Az információkat gyűjtő kliens szoftver pedig az alábbi operációs rendszereken működik:

1. Microsoft Windows 95/98/Me/NT4/2000/XP/2003,
2. Linux,
3. Mac OS X,

4. Sun Solaris,
5. IBM AIX.

Ezek egy része nemhivatalos verzió, továbbá a kliens még fejlesztés alatt áll, jelenlegi verziója az *1.0 Release Candidate 3-1 (2006/08/02)*.

2.1. A szoftver felépítése, telepítése

Az OCS Inventory NG összetevői a Menedzsment szerver és a klienseken futó Leltározó ágens. A Menedzsment szerver komponensei:

– Adatbázisszerver, a leltárinformációk tárolására – ez jelenleg 4.1-es vagy újabb verziójú MySQL kiszolgáló lehet.

– Kommunikációs szerver, amely a kliensekkel és az adatbázissal tartja a kapcsolatot, Apache webszerver 1.3.X/2.X szükséges hozzá, és Perl nyelven írt Apache-modulként fut, ezzel jobb teljesítményt ér el, mint ha CGI programként futna, mivel rögtön a webszerver indításakor lefordul, nem pedig minden indításkor újra és újra.

– Telepítőszerver, amely az általunk készített csomagokat és a telepítést vezérlő konfigurációs állományokat tárolja. Ez egy tetszőleges webszerver lehet, egyetlen feltétel, hogy ismerje az SSL-t.

– Adminisztrációs konzol, melyet böngészőnkből kezelve vezérelhetjük az egész alkalmazást, és böngészhetünk a begyűjtött információk között. Ezt PHP-ben valósították meg, ZIP és GD bővítmények szükségesek a működéséhez.

Mivel különösebb hátrányát nem láttuk, ezért a Menedzsment szervert teljes egészében egy közepes teljesítményű, Debian Sarge Linuxot futtató szerverre telepítettük. Bár nagyobb terhelésnél, illetve már meglévő alapkomponenseknél megfontolandó az elosztott rendszer használata is.

Az OCS Inventory NG Menedzsment Szerver függőségei:

– MySQL 4.1 vagy magasabb verzió,
– Perl 5.6 vagy magasabb verzió,
– Apache 1.3.33 vagy magasabb 1.3.X verzió, vagy 2.X vagy magasabb verzió,
– Apache mod_perl 1.29 vagy magasabb verzió,
– PHP 4.3.2 vagy magasabb verzió, ZIP-támogatással,
– Apache mod_php 4.3.2 vagy magasabb verzió,
– A következő Perl-modulok: XML::Simple 2.12, Compress::Zlib 1.33, Apache::DBI 0.93, Net::IP 1.21, DBD::Mysql 2.9004, DBI 1.40 vagy ezek magasabb verziói.

A telepítés nehézséget nem okoz, az `OCSNG_LINUX_SERVER_1.0RC3-1.tar.gz` [4] csomag kibontását követően a `README` fájlban is jelzett függőségek telepítése után a `runme.sh` szkript elindítása után, a feltett kérdésekre többnyire automatikusan megtalált válaszokat kell jóváhagyni, ezután pedig a szoftver pillanatok alatt feltelepül a szerverre.

A windowsos leltározó kliens [6] pedig egy C++ nyelven megírt alkalmazás, futtatásához további összetevőkre nincs szükség. Képes megfelelően paraméterezve az aktuális leltár fájlba generálására, mely a hálózatra nem kapcsolt gépek adminisztrálásánál hasznos lehet, és természetesen képes ezt közvetlenül feltölteni a Kommunikációs szervernek is.

Lényegesebb, hogy lehetőségünk van egy szolgáltatás telepítésére is, amely a háttérben futva időnként elindítja a leltározó klienst, illetve a csomagtelepítőt is szükség esetén. Ez esetünkben különösen hasznos, mivel *SYSTEM PROCESS* jogokkal rendelkezik, és a felhasználókkal ellentétben képes a programok telepítésére is.

3. Az OCS Inventory használata

3.1. Hardver- és szoftverleltár

A szolgáltatás feltelepítését követően nagyjából 10 óra elmúltával automatikusan érkezik a szerverre egy a kliens által összeállított leltár, melyet ezt követően rendszeresen aktualizálni fog a rendszer.

Mit tudunk meg egy windowsos számítógépről? A hardverről szinte mindent. Amit az *Eszközkezelő* böngészésével meg lehet tudni, eltárolja a program, továbbá a telepített alkalmazásokról is kapunk egy listát. Ez sajnos a felmásolt programokat – melyek telepítés híján nem regisztrálják magukat – nem mutatja meg, de így is nagy segítséget nyújt.

Előfordul, hogy a telepített alkalmazások listájának valamely mezője üres, ez a Windowst nem zavarja, de az OCS szerver így nem fogadja el a leltárt. Ezt egy rövid folttal [7] orvosoltuk, remélhetőleg belekerül a hivatalos forrásba.

Továbbá egy összefoglalást is ad a fontosabb adatokról, ezek között a Windows és (ígért fejlesztésként) az Office licencekről, és a hozzá tartozó kulcsokról is, mely adatok a leltár elkészítéséhez igen hasznosak.

Minden számítógéphez tartozik alapértelmezetten egy *TAG* mező, amit a kliens telepítésekor megadhatunk, illetve egyéb adminisztratív adatokat is (például beszerzési dátum, leltári szám) megadhatunk, amit a szoftver a biztonság kedvéért el fog tárolni a kliensen is, így egy esetleges adatbázis összeomlás után a következő adatbeküldéskor automatikusan visszanyerjük azokat.

A kezelőfelületen lehetőségünk van a felvitt gépek adatait böngészni, különféle kritériumok alapján keresni, illetve egyéb adminisztratív feladatokat végrehajtani. Ezek közül hasznos a hálózat feltérképezése, amelyben tájékoztatást kapunk a kliensek által látott hálózatról, így a listában megjelennek olyan eszközök is, melyek nem futtatják az OCS klienst, például a hálózati nyomtatók.

További hasznos lehetőség a duplikált gépek keresése, ahol például a duplikált sorozatszámok listázásával felismerhető a licencek túlhasználása, a Microsoft operációs rendszerek hanyag telepítése. Több ilyen esetet is kiszűrtünk a program segítségével, ebből volt vakriasztás is, ahol a számítógép elnevezését változtatták meg időközben, és így új gépként tűnt fel a listában.

Lehetőségünk van a program opcióinak módosítására, például a leltár beküldések ritkítására, a hálózat felfedezés kikapcsolására.

A hálózatra nem kapcsolt, de leltározni kívánt gépek kézzel összegyűjtött adatait is ezen a felületen lehet legegyszerűbben a rendszerbe feltölteni. Ezt az opciót használtuk a leltár kezdeti feltöltéséhez.

Ezek szemléltetésére az előadásban egy rövid bemutatót tartok.

3.2. Programok automatikus telepítse

Az OCS Inventory erre is lehetőséget nyújt, feltéve, hogy szolgáltatásként futtatjuk azt a windowsos gépen.

Ehhez a gépekre telepített tűzfalon át kellett engedni az `OCSInventory.exe`-t a 80-as és a 443-as porton, a `download.exe`-t pedig a 80-as porton, az OCS szerverekhez.

Továbbá az SSL kapcsolathoz tanúsítvány fájlokat kellett a szerverre generálni, aminek

a kulcsfájlját kell bemásolni a kliens gép OCS Inventory könyvtárba `cacert.pem` néven. Bárhonnan ugyanis nem hajlandó fájlokat letölteni a kliens, csak ha számára megfelelő a kulcsfájl.

A program csupán annyira képes, hogy letöltsön egy ZIP-archívumot, majd kicsomagolás után futtasson belőle egy fájlt, vagy elindítson egy tetszőleges parancsot. Miután ezek futtatásával végzett, letörli az ideiglenesen tárolt ZIP-et és a kicsomagolt fájlokat is. Bonyolultabb interakcióra nem képes a program, a *Next, Next, Yes, Yes, Finish* jellegű telepítéseket nem tudja vezérelni.

Természetesen a ZIP-fájl el is hagyható, ekkor csak sima parancsfuttatásra használjuk a rendszert, ami néha jól jöhet.

A telepítendő programok installáló folyamatának automatizálásában hasznosak

– az *Unattended: A Windows deployment system* oldalon elérhető információk [8]

– az *AppDeploy.com* tudásbázisa [1]

– és az interaktív folyamatokat vezénylő AutoIT-szkriptek [2]

Például a 7-Zip tömörítőprogram *Next, Next, Yes, Yes, Finish*-típusú telepítésére AutoIT-szkriptet írtunk. Általában érdemes a telepítőket automatizált módban futtatni, és ezt szükség esetén AutoIT-tel vezérelni.

Ha elkészültünk egy telepítés automatizálásával, a szükséges összetevőket ZIP-archívumba csomagolva a szoftver *Deployment* | *Build* pontjában feltölthetjük a rendszerbe.

– *Name:* A csomag neve.

– *Operating system:* Csak a Windows választható jelenleg.

– *Protocol:* Csak a HTTP választható.

– *Priority:* ez 0-tól 10-ig választható. A kisebb prioritású csomagokat telepíti először a rendszer. Ha valami oknál fogva valamelyik csomag telepítése nem sikerül, a kliens továbblép a nagyobb prioritásúra. A 0 prioritás különleges, ekkor amíg ezeket nem telepíti sikeresen a rendszer, addig nem léphet tovább.

– *File (deployed on client computers):* A szerverre feltöltendő ZIP-fájl helye.

– *Action:* A feladat, ami lehet:

  - *Store:* A megadott útvonalra másolja a csomag tartalmát a program.
  - *Execute:* A ZIP-archívumot kicsomagolja egy ideiglenes könyvtárba, majd abban a könyvtárban lefuttatja a megadott parancsot.
  - *Launch:* A kicsomagolás után a megadott fájlt elindítja a program.

– *Command:* A futtatandó parancs.

– *Path:* Tárolás esetén az archívum tartalmát ide helyezi a kliens.

– *File name:* A ZIP-archívum neve.

– *User notifications:*

  - *Warn user:* A felhasználóval közölhetünk információkat, illetve akár a hozzájárulását, közreműködését is kérhetjük a telepítéshez.
  - *Text:* Ezt látja a felhasználó.

- *Countdown:* Visszaszámlálás, mialatt:
  - *User can abort:* a felhasználó megszakíthatja,
  - *User can delay:* illetve késleltetheti a program végrehajtását.
  - *Installation completion need user action:* Továbbá kérhetjük is a felhasználó jóváhagyását is.

Ezután feltöltődik a ZIP-csomag, majd beállíthatjuk, hogy azt mekkora darabokban tárolja el a rendszer. Hibás letöltésnél csak az adott darabot fogja a rendszer újra tölteni, a darabokat pedig előre megadott tempóban tölti le a kliens, hogy ne terhelje túl a szervert.

Azt, hogy mekkora méretű ZIP-csomagokat fogadjon el a szerver feltöltésre, a szerveren lévő `php.ini` maximális feltölthető fájlméret beállításával adhatjuk meg. (Figyelem! Esetleg több `php.ini` beállítást is meg kell változtatni.)

Ezekből az információkból összeállítja, és a telepítőszerveren az aktuális időbélyeg után elnevezett könyvtárban elhelyezi a rendszer a csomaghoz tartozó vezérlőfájlt *info* néven, s oda helyezi el a ZIP-fájl darabjait is.

Ahhoz, hogy egy így elkészített csomag megjelenjen az OCS telepíthető csomagok listájában, aktiválni kell a csomagot, a *Deploy* | *Activate* opció segítségével.

Ki kell választani az aktiválható csomagok közül a nekünk tetszőt, majd az *activate* ikonra kattintani.

Ezután két szervercímet kér tőlünk a rendszer, az első az info fájl *HTTPS* protokollal elérhetősége (`https://ocsinventory-ng/download`), a második a részletek *HTTP* protokollal elérhetősége (`http://ocsinventory-ng/download`).

Amennyiben csak parancsot futtatunk, a második természetesen elhagyható, a kiírt figyelmeztetést nyugodtan tudomásul vehetjük, majd továbbléphetünk a programmal.

A program tényleges telepítéséhez ki kell válogatnunk a megcélzott számítógépeket, majd az alul lévő *Deploy* linkre kattintás után kiválaszthatjuk, hogy melyik csomagot szeretnénk elküldeni ezeknek a gépeknek, ezt az *Affect* ikonra kattintással érhetjük el.

A következő adat beküldési ciklusnál a kliensek észre fogják venni a számukra kiadott csomagokat, és elindítják a letöltőrutint, amely megpróbálja elvégezni a munkáját, és az eredményről a szervert is tájékoztatja. Ha minden jól megy, a felhasználó csak néhány felugró értesítést lát, vagy még azt sem, de a háttérben a gépre telepített programok megszaporodtak. Természetesen a programok eltávolításának automatizálását is meg lehet oldani, például egy jól megírt *AutoIT-szkript* segítségével.

Ennek a folyamatnak a működéséről is következik egy rövid bemutató az előadásban.

4. Összefoglalás

Az OCS Inventory NG alkalmazás, bár lehetőségeinek kis részét használtuk ki, így is jelentős segítséget nyújtott a gépek feltérképezésében. Jó szolgálatot tett a leltár ellenőrzésében és az azonos sorozatszámú szoftverek kiszűrésével a licencek összeszámolásánál. A jövőbeli telepítések automatizálásával a szoftver a rendszergazda eszköztárának egy igen hatékony részét fogja képezni, egyéb programokkal – mint például az AutoIT – kombinálva szinte csak a rendszergazda fantáziája szab az alkalmazásoknak és a lehetőségeknek határt.

Hivatkozások

[1] Az AppDeploy.com tudásbázisa. URL `http://www.appdeploy.com/packages/browse.asp?cat=all`.

[2] Az AutoIT weblapja. URL `http://www.hiddensoft.com/autoit3/`.

[3] Az OCS Inventory friss dokumentációja PDF és ODT formátumban. URL http://ocsinventory.sourceforge.net/index.php?page=1_0_rc3-1.

[4] OCS Inventory linuxos szerver ocsng_linux_server_1.0rc3-1.tar.gz. URL http://prdownloads.sourceforge.net/ocsinventory/OCSNG_LINUX_SERVER_1.0RC3-1.tar.gz?download.

[5] Az OCS Inventory weblapja. URL http://ocsinventory.sourceforge.net/.

[6] OCS-Inventory windowsos leltározókliens ocsng_win32_agent_1.0rc3-1.zip. URL http://prdownloads.sourceforge.net/ocsinventory/OCSNG_WIN32_AGENT_1.0RC3-1.zip?download.

[7] Szabó Péter foltja az OCS-Inventory szerverhez. URL http://www.szszi.hu/~pts/ocsinventory/pts-ocsinventory-nullsoftware.patch.

[8] Unattended: a Windows deployment system. URL http://unattended.sourceforge.net/installers.php.

Extrém rendszeradminisztráció: djbware és társai

Korn András
<korn.andras@tmit.bme.hu>

Budapesti Műszaki és Gazdaságtudományi Egyetem
Távközlési és Médiainformatikai Tanszék
1117 Budapest, Magyar Tudósok Körútja 2.

Kivonat

Elvesztettük. Nem tudjuk, hogyan és mikor, de elvesztettük a kapcsolatot a Unix-filozófiával. A programjaink egyre több mindent csinálnak, és ennek egyre nagyobb részét a mi tudtunk és kérésünk nélkül, egyre automatikusabban. Elindultunk a lejtőn: a Linuxaink elindulását vezénylő programok összetettségének legalább háromnegyedét a bitkolbászok és egyéb csicsák megjelenítése adja. Szerencsére még nem késő visszafordulni…

Ritka, hogy néhány szoftvercsomag annyira meg tudja osztani az online közösséget, mint Daniel J. Bernstein programjai. Egyesek szerint tökéletesek, mások szerint öncélúan excentrikusak. A cikk a DJB-féle fejlesztési és üzemeltetési szemléletet próbálja bemutatni a daemontools eszközkészlet runit nevű hasonmása példáján keresztül.

Tartalomjegyzék

1. Bevezetés

Ebben a cikkben arra fogok törekedni, hogy bemutassak egy olyan programcsomagot, amely a manapság szokásosnál szigorúbban veszi a Unix-filozófiát; azt a filozófiát, amit röviden úgy foglalhatnánk össze, hogy egy program egy dolgot csináljon, de azt jól.

Kicsit hosszabban így fogalmazhatnánk: ne olyan programokat írjunk, amelyek mindent megcsinálnak, hanem csak valamilyen jól körülhatárolt feladatkörbe tartozó feladatokat oldjanak meg minél rugalmasabban, egyszerűbben, hatékonyabban, biztonságosabban, robusztusabban és modulárisabban. Valamint úgy, hogy a mi programunk képes legyen más programokkal együttműködve más, bonyolultabb feladatokat is megoldani, és más programok is használhassák a mi programunkat saját képességeik bővítésére.

Törekedjünk arra, hogy – hacsak nem megy a teljesítőképesség rovására – ne valósítsunk meg olyan funkciót, amelyet egy másik program megvalósít, hanem használjuk inkább azt a másik programot, és az így felszabadult időben írjunk meg valami olyat, amit más még nem írt meg, vagy igyunk egy pohárral. Ha összetett funkcióról van szó, akkor egy egész üveggel.

Nem kell magyarázni a spanyolviasz újrafeltalálásának hátrányait: elfecsérelt idő, elpazarolt elektronok, és ráadásul amit mi mellékesen megírunk, mert szükségesnek látszik, lehet, hogy nem lesz ugyanolyan jó, mint az a megvalósítás, amely éppen ennek a funkciónak a kedvéért született.

Jó példa a Unix-filozófiát követő programra a grep, a find vagy a sed.

Bizonyos értelemben ellenpélda az Apache, a MySQL, a cron, az inetd, és még sorolhatnánk – ezek ugyanis mindannyian megvalósítanak legalább egy olyan funkciót, amelyet külső program is tudna számukra nyújtani: a „daemonizációt”. A MySQL ráadásul a folyamatmonitorozást is házon belül oldja meg, holott mindezt tudná pl. a *daemon* nevű program.

Mégis, talán az egyik legnagyobb bűn, amit el lehet követni, az, ha kikapcsolhatatlan többletfunkciót építünk a programunkba. Az *and* (auto-nice-daemon) és a *hddtemp* például egészen a közelmúltig képtelenek voltak *nem* daemonizálni magukat, de biztosan találhatunk olyan programot, amely máig képtelen az előtérben maradni.

„Na de mi értelme kikapcsolni a daemonizálódást egy rendszerszolgáltatásban?” – kérdezhetnénk. Többek között erre a kérdésre kapunk majd választ az alábbiakban.

Ez a cikk jórészt egy, a Műegyetemen a *Unix/Linux kiszolgálók üzemeltetése* című választható tárgy [3] keretében elhangzott előadás átdolgozása; ennek megfelelően nemigen találunk benne irodalmi hivatkozást, és a nyelvezete is lazább, mint amit egy „tudományos” cikktől elvárnánk. Mentségemre szóljon, hogy ezt a cikket nem tekintem „tudományos munkásságom” részének.

2. Mi a djbware?

Szűk értelemben véve *djbware* minden olyan program, amelyet Daniel J. Bernstein [1] írt. Kevesen vitatják (meggyőzően), hogy a programok maguk jó minőségűek; annál többen haragszanak azonban a szerzőre, a legváltozatosabb okokból, de gyakran nem megalapozatlanul. Bernstein úrnak rengeteg kérdésről van nagyon határozott véleménye, és nem szokott habozni ennek a véleménynek keresetlen szavakkal hangot adni, ami nem a népszerűség záloga.

A szűk értelemben vett djbware egyébként jogállását tekintve nem szabad szoftver. DJB álláspontja szerint ugyanis a szerzői jog elegendő útmutatást ad arra nézve, hogy egy, az Interneten közzétett programot milyen módon szabad használni és terjeszteni, így nem mellékel szoftvereihez licencszerződést (némelyiket azonban a public domainbe helyezte). A szabad szoftverekben többek között azt szeretjük, hogy ha akarjuk, módosíthatjuk őket, és a módosított verziót is terjeszthetjük; erre azonban csak a licencszerződés biztosíthat lehetőséget,

vagyis, ha nincs licenc, a módosított programot nem terjeszthetjük.

Tágabb értelemben djbware-nek tekinthetők azok a programok is, amelyeket DJB saját programjai ihlettek, esetleg azoknak a szabad reimplementációi, vagy akár csak hasonló fejlesztési filozófiával születettek, és ugyanaz a szemlélet tükröződik bennük, amely DJB programjaiban is: az egyszerűségre való szélsőséges törekvés; a feladatok szétválasztása; a konfiguráció gépbarát (és nem emberbarát) kezelése stb. A következőkben röviden bemutatok néhány szűk vagy tág értelemben vett djbware-t.

A daemontools és a runit a hagyományos System V initet váltja ki egy olyan rendszerrel, ami párhuzamosan indítja el a szolgáltatásokat, nem szekvenciálisan; automatikusan újraindítja azokat, amelyek megálltak; gondoskodik a naplóüzenetek kezeléséről; és, mintegy mellékhatásként, az egyes szolgáltatásokhoz tartozó PID-k (process ID-k, folyamatazonosítók) nyilvántartását is egyszerűvé és egységessé teszi.

A socklog a syslogd-t váltja ki; sokkal egyszerűbb, mivel egyszerűen a standard kimenetre írja a naplóüzeneteket. Ezek fájlokba válogatása egy másik program (a multilog vagy az svlogd) feladata. A naplófájlokat automatikusan rotálhatjuk, ha elértek egy bizonyos méretet (nemcsak adott időközönként, mint a logrotate esetén). Ha akarjuk, megakadályozhatjuk, hogy naplóüzenetek elvesszenek, akár azon az áron is, hogy az üzenet naplózásáig megáll a szolgáltatás.

A djbdns olyan DNS-szerver-csomag, amely élesen szétválasztja a BIND három funkcióját: az autoritatív szervert, a cache-elő rekurzív rezolvert és az AXFR-szervert, amely a zónatranszferért felelős. Jól átgondolt működésének köszönhetően eleve elkerül számos olyan problémát, amelyek a BIND-dal éveken át együtt jártak vagy még mindig együtt járnak (cache poisoning, memória elfogyása, root compromise lehetősége).

A qmail olyan MTA (Mail Transfer Agent), amely szélsőségesen moduláris felépítésének köszönhetően egyszerűen testreszabható.

Ebben a cikkben elsősorban a runit csomagról [2] lesz szó, amelyet ugyan nem DJB írt, de csaknem izomorf a DJB-féle daemontools csomaggal; a runittal való ismerkedés előtt azonban röviden szót ejtenék a hagyományos System V init néhány problémájáról.

3. Bajok az inittel

Az init speciális folyamat; helyes működése az egész rendszer szempontjából kritikus. Emiatt jó lenne, ha – a Unix-filozófiával összhangban – egyszerű lenne, és jól körülhatárolt feladatai lennének. Minél összetettebb egy program, annál nagyobb valószínűséggel tartalmaz hibákat.

Ehhez képest az init kódja meglehetősen hosszú és bonyolult, és sok feladata van:

- egy aránylag összetett szintaxisú konfigurációs fájlt, a `/etc/inittab`-ot kell értelmeznie, és az ezzel kapcsolatos összetett viselkedést implementálnia;
- egyszerre több gyermekfolyamat állapotát kell nyomon követnie, a véget ért folyamatok helyett esetleg másikat indítania;
- nem túl kifinomult UPS-kezelés is van benne;
- piszkálnia kell az utmp-t, ha egy gyermekfolyamata véget ér.

Mindezt nem lenne muszáj az 1-es PID-del futó folyamatban csinálni; a runit – mint azt majd látni fogjuk – ennél sokkal kevesebbet bíz az init helyett futtatott programra. Ennek többek között az az eredménye, hogy a runit 96 k memóriát foglal (dietlibc-vel csak kb. 20-at), míg a hagyományos init kb. 1800 k-t – másfél megabájt memória persze ma már senkit sem vág földhöz, de azért a programok összetettségéről valamit elárulnak ezek az értékek.

Elegendő lenne, ha az init el tudná indítani azt a programot, amely a rendszer elindításáért felelős, majd egy másikat, amely a rendszer futásáért felelős; végül pedig, amikor emez kilépett, akkor a rendszer leállításáért felelős programot.

Nem feltétlenül fontos ugyan, de az init bootfolyamata is szükségtelenül lassú, bárhogyan is trükközünk az initszkriptek párhuzamos futtatásával.

Ennél nagyobb horderejű probléma, hogy az init képtelen a leállt szolgáltatásokat automatikusan újraindítani, kivéve, ha *respawn*-ra állítjuk őket az inittabban, ebben az esetben viszont akarattal sem tudjuk őket leállítani, csak runlevelváltással – ez viszont meglehetősen rugalmatlan megoldás, mert így minden szolgáltatás-kombinációhoz külön runlevelnek kellene tartoznia, azonban csak kb. négy áll rendelkezésre.

Rejtélyes problémákat okozhat az, hogy az initszkriptek vagy az init, vagy az őket indító shell környezetét öröklik; így előfordulhat, hogy egy initszkript mást csinál, ha parancssorból indítjuk el, és mást rendszerindítás közben.[1] Azt, hogy egy initszkript rendszerindításkor valóban az elvárt módon viselkedik-e, teljes bizonyossággal csak úgy tudjuk tesztelni, ha tényleg újraindítjuk a rendszert.

A System V init legsúlyosabb problémája azonban az, hogy nem tudja megbízhatóan leállítani a szolgáltatásokat.

3.1. Szolgáltatások leállítása a System V init keretrendszerében

A szolgáltatás leállításához általában valamilyen signalt (pl. TERM) kell küldeni a szolgáltatáshoz tartozó folyamatnak. Csakhogy: mi a PID-ja ennek a folyamatnak?

A „megoldás”, gondolhatnánk, az ún. pidfájl – egy olyan fájl, amibe a szolgáltatáshoz tartozó folyamat beleírja, hogy mi a process ID-ja. De:

– a pidfájlt le lehet törölni, felül lehet írni;
– ha a program kilép, a megadott PID-del indulhat másik folyamat – ha ennek küldjük a signalt, az „Nem Jó”.

A Debian *apache2* csomagjának initszkriptje segítségével mutatom be, milyen megoldások születnek. A szkript* így kezdődik:

```
ENV="env -i LANG=C PATH=/usr/local/bin:/usr/bin:/bin"
APACHE2="$ENV /usr/sbin/apache2"
APACHE2CTL="$ENV /usr/sbin/apache2ctl"

apache_stop() {
  PID=""
  PIDFILE=""
  AP_CONF=/etc/apache2/apache2.conf

  # apache2 allows more than PidFile entry in the config but only the
  # last found in the config is used; we attempt to follow includes
  # here, but only first-level includes are supported, not nested ones

  for i in $AP_CONF ‘awk ’$1 ~ /^\s*[Ii]nclude$/ &&
   $2 ~ /^\// {print $2}’ $AP_CONF‘; do
    PIDFILE=‘grep -i ^PidFile $i | tail -n 1 | awk ’{print $2}’‘
    if [ -e "$PIDFILE" ]; then PID=‘cat $PIDFILE‘; fi
  done
```

[1] Ilyen probléma lehet, hogy egyes, kézzel felrakott szolgáltatások initszkriptjei csak akkor működnek, ha van érvényes `$HOME` környezeti változó. Parancssorból indítva tökéletes, rendszerintításkor mégsem indul el a szolgáltatás.

* A közölt forráskódokban a szóközöket kissé összenyomtuk az eredetihez képest – *a szerk.*

Itt a szkript megpróbálta megkeresni az apache2 konfigurációs fájljaiban szereplő utolsó `PidFile` direktívát, mivel az az érvényes. Sajnos az apache2 konfigurációs fájljai tetszőleges mélységben include-olhatják egymást, és ezt a fenti shell-szkript meg sem próbálja követni, úgyhogy lehet, hogy már itt elvérzik, és nem találja meg a megfelelő pidfájlt – hanem mondjuk egy másikat, egy régit, és esetleg az abban található PID-nak küldi majd a signalt.

```
errors=‘$APACHE2 -t 2>&1‘
```

Itt a szkript megpróbálja megállapítani, hogy az apache2 konfigurációja szintaktikailag hibátlan-e; ha nem, akkor – mivel az apache2 állítólag fut, hibás konfigurációval azonban nem indul el – biztos, hogy a konfigurációt az apache2 elindítása óta módosították, vagyis akár az imént megtalált `PidFile` direktíva is kerülhetett oda az indulás után. Vegyük észre, hogy ha a konfigurációt az apache2 indítása után módosítottuk, és az új konfiguráció hibátlan, akkor a módosítás ténye sosem tudatosul ebben a szkriptben, vagyis boldogan megpróbálja felhasználni az új konfigurációban szereplő pidfájlt, amelyről a futó apache2 mit sem tud.

```
if [ $? = 0 ]; then
  # if the config is ok then we just stop normally
  if [ -n "$PID" ]; then
    $APACHE2CTL stop
```

Az `apache2ctl stop` is csak a pidfájlban szereplő folyamatnak küld TERM signalt, nem foglalkozik azzal, hogy az valóban egy apache2 folyamat sorszáma-e.

```
    CNT=0
    while [ 1 ]; do
      CNT=$(expr $CNT + 1)
      [ ! -d /proc/$PID ] && break
```

Versenyhelyzet, mert a stop művelet óta indulhatott új folyamat ezzel a sorszámmal; sőt azóta, hogy a PID változó értéket kapott, már kiléphetett az az apache2, és indulhatott egy másik, amit az `apache2ctl stop` sikeresen le is állíthatott, mi azonban még az előző apache2-példány PID-ját most birtokló folyamat terminálódását várjuk hasztalan.

```
      if [ $CNT -gt 60 ]; then
        if [ "$VERBOSE" != "no" ]; then
          echo " ... failed!"
          echo -n "Apache2 failed to honor the stop command, "
          echo "please investigate the situation by hand."
        fi
        return 1
      fi
      sleep 1
    done
  else
```

Üres maradt a `$PID` változó, de jó a konfiguráció – hát ez meg hogy lehet? Pl. úgy, hogy az egyik jó konfigurációval indítottuk el az apache2-t, majd átírtuk a konfigurációt, de még mindig az előző apache2-példány fut. A szkript azonban csak széttárja a karját:

```
    if [ "$VERBOSE" != "no" ]; then
      echo -n " ... no pidfile found! not running?"
```

Te vagy a számítógép! Mondd meg te, hogy fut-e!

```
    fi
  fi
else
```

Nem volt jó a konfiguráció; kezdjünk hát vad találgatásba a futó apache2 PID-ját illetően.

```
  [ "$VERBOSE" != "no" ] && echo "$errors"
  # if we are here something is broken and we need to try
  # to exit as nice and clean as possible
  # if pidof is null for some reasons the script exits automagically
  # classified as good/unknown feature
  PIDS=‘pidof apache2‘ || true
  REALPID=0
  # if there is a pid we need to verify that belongs to apache2
  # for real
  for i in $PIDS; do
    if [ "$i" = "$PID" ]; then
      # in this case the pid stored in the
      # pidfile matches one of the pidof apache
      # so a simple kill will make it
      REALPID=1
    fi
  done
  if [ $REALPID = 1 ]; then
    # in this case everything is nice and dandy
    # and we kill apache2
    kill $PID
```

Találtunk egy szerencsétlen apache2 nevű folyamatot, amely rossz helyen volt rossz időben – a PID-ja véletlenül megegyezett egy olyan PID-val, amit egy szintaktikailag hibás apache2-konfiguráció tökéletlen elemzése révén nyert ismeretlen eredetű pidfájlból olvastunk ki valamikor rég. Gyorsan ki is lőttük; túl sokat tudott. Különben sem hiszünk a véletlenekben. Vesznie kellett, ez volt a sorsa. Ez volt megírva a csillagokban. Hatalmas karmikus teher nehezedett rá.

```
  else
    # this is the worst situation... just kill all of them
    #for i in $PIDS; do
    #       kill $i
    #done
    # Except, we can’t do that, because it’s very, very bad
```

Ez a megoldás már a szkript elszánt szerzőjének is sok volt; szerencsére nem irt ki ész nélkül minden apache2 nevű folyamatot.

```
    if [ "$PIDS" ] && [ "$VERBOSE" != "no" ]; then
      echo " ... failed!"
      echo "You may still have some apache2 processes running.  There are"
      echo "processes named ’apache2’ which do not match your pid file,"
      echo "and in the name of safety, we’ve left them alone.  Please review"
      echo "the situation by hand."
    fi
    return 1
  fi
fi
}
```

Talán egyetérthetünk abban, hogy ezt rossz nézni.

A problémát éppen az okozza, hogy a rendszerszolgáltatások „daemonizálva” futnak, tehát indítás után indítanak egy gyermekfolyamatot, amely új folyamatcsoportot (process groupot) nyit, leválik a terminálról és a háttérben folytatja futását; a shellünk által indított folyamat pedig lényegében azonnal kilép. Így sosem tudhatjuk teljes bizonyossággal, hogy a szolgáltatáshoz milyen PID tartozik, vagyis hogy kinek kell küldeni a TERM signalt, amikor le akarjuk állítani a szolgáltatást.

Ezt a gondot elkerülhetnénk, ha a szolgáltatás nem a háttérben futna, hanem az a program, amellyel elindítottuk, nem térne vissza egészen addig, amíg a szolgáltatáshoz tartozó folyamat működik. A szülőfolyamat hivatalból tisztában van a gyermekfolyamatai PID-jával, így mindig tudja, kinek kell signalt küldeni; nekünk pedig csak annyi lenne a dolgunk, hogy ezzel a szülőfolyamattal kommunikáljunk.

Többek között éppen ezt teszi lehetővé a *runit.*

4. Mi a runit?

A *runit* eleinte a DJB-féle *daemontools* szabad reimplementációja volt; BSD-szerű licencfeltételek vonatkoznak rá. Mára szinte a teljes SystemV init lecserélhető vele egy robusztusabb, rugalmasabb rendszerre.

A runit alapgondolata az, hogy juttassuk érvényre a Unix-filozófiát: az init (és környéke) feladatait osszuk fel jól körülhatárolható részekre, és egy-egy ilyen feladatrészt egy-egy kicsi és egyszerű program segítségével oldjunk meg. A runit csomag programjai tipikusan kevesebb, mint ezer sor C kódból állnak, és sosem nő a memóriafoglalásuk, mert nem foglalnak dinamikusan memóriát.

Azon kívül, hogy kiváltja az initet és az initszkripteket, segít a naplózás megszervezésében is. Be lehet vezetni kicsit, nagyon és teljesen.

Ha kicsit vezetjük be, akkor megmarad a System V init, de bizonyos szolgáltatásokat kezeltethetünk a runittal. Ennek sok előnye van:

- A szolgáltatások tiszta környezettel indulnak.
- Újraindulnak, ha kilépnek (és ezt szeretnénk).
- Opcionálisan megbízható és rugalmas naplózást kapnak:
  - naplóüzenet nem vész el *naplózás közben* (a syslog esetében ez nem teljesül);
  - méret alapján rotált logok;
  - utófeldolgozás lehetősége rotációkor (ezt a *logrotate* is tudja);
  - akkor is zavartalanul naplózhatunk, ha a szolgáltatás chrootban fut (nem kell trükközni a `/dev/log` sockettel);
  - a naplózás és a szolgáltatás egymástól függetlenül újraindítható/leállítható.
- Kiválasztott felhasználók *sudo* nélkül is menedzselhetnek bizonyos szolgáltatásokat.
- Könnyedén és megbízhatóan lekérdezhető a szolgáltatások státusza.
- Könnyedén és megbízhatóan küldhető nekik signal.
- Könnyedén biztosíthatjuk ezeket a tulajdonságokat a felhasználók saját „szolgáltatásainak” – pl.:
  - *screen,* benne *irssi* vagy valamilyen azonnali üzenetküldő program (a géppel együtt indul, és újraindul, ha elszállna);

- *eggdrop* ircbot (ezt sajnos át kell írni, mert mindenképpen a háttérbe akarja rakni magát, de így nem kell percenként cronból ellenőrizni, hogy fut-e);
- *fetchmail*;
- *signify* e-mail aláírás (`.signature`) generátor;
- VNC-szerver, benne valamilyen X session;
- akár *apache* vagy egyéb.

Ha a runitot „nagyon” bevezetjük (egy fokozattal jobban), akkor emellett még az initet is kiváltjuk, de a rendszerindulást továbbra is a hagyományos initszkriptjeink vezénylik le, csak az inittab válik feleslegessé. Minél több szolgáltatást állítunk át úgy, hogy a runit menedzselje, annál gyorsabbá válik az indítás és a leállítás, mert sokmindent egyszerre fog csinálni a runit (azért nem kell azzal foglalkoznia, hogy minek mi után kell jönnie, mert ezt az intelligenciát nekünk kell megvalósítanunk a `run` szkriptekben – lásd később).

Ha teljesen bevezetjük a runitot, akkor az *rcS* után már csak a runit által menedzselt szolgáltatásaink futnak, vagyis az *rc2* már nem is kell, hogy lefusson. Az *rcS* által elvégzett egyszeri teendők kívül esnek a runittal összefüggő változások körén; az *rcS* szkriptet változtatás nélkül elindíttathatjuk a runittal is.

4.1. A runit architektúrája

A runit által menedzselt szolgáltatásokhoz tartozó fájlok szolgáltatásonként egy-egy könyvtárban kapnak helyet. Minden egyes szolgáltatásnak van egy ilyen szolgáltatáskönyvtára *(service directory).*

Ebben található az a program (általában szkript), amely `run` névre hallgat, az adott szolgáltatást elindítja és futtatja; addig nem léphet ki, amíg a szolgáltatás nem lépett ki.

Létrehozhatunk a könyvtárban egy `down` nevű fájlt, ami azt jelzi, hogy a szolgáltatás normális állapota az, hogy nem fut; a runit nem fogja automatikusan elindítani, csak ha erre külön megkérjük.

Lehet a könyvtárban egy `log` nevű alkönyvtár, ha a szolgáltatáshoz naplózást is szeretnénk – lásd később.

Lehet ott egy `check` nevű program, ami 0-s visszatérési értékkel lép ki, ha a szolgáltatás működik (ami nem feltétlenül azonos azzal, hogy fut), egyébként nullától különbözővel; ezt a programot természetesen nekünk kell megírnunk.

Lehet egy `finish` nevű program, amit abban az esetben kell lefuttatni, ha a `run` kilép; ebben célszerű a szolgáltatás leállása után esetleg szükséges feladatokat elvégezni (pl. a konzolos bejeletkezést vezérlő *getty,* mint szolgáltatás esetén az `utmp`-ben jelezni, hogy az adott konzolon véget ért a login session).

Végül pedig a runit létrehoz egy `supervise` alkönyvtárat a szolgáltatáskönyvtárban – erről szintén lesz később szó.

A szolgáltatáskönyvtárakat általában a `/var/service` alá symlinkeljük be, ha az adott szolgáltatást futtatni akarjuk, és töröljük a rá mutató symlinket, ha már nincs szükségünk rá (ha csak ideiglenesen akarjuk leállítani, arra van más módszer).

A `/var/service` könyvtárat figyeli a `runsvdir`, és minden alkönyvtárra indít egy `runsv` folyamatot, ami a szolgáltatás futtatásáért és a vele való kommunikációért felelős. Erről a következő szakaszban lesz szó.

4.2. A runit részei

A runit csomag számos programot tartalmaz; mindegyik valamilyen viszonylag kicsi részfeladatot lát el, és emiatt egyszerű a kódja. Tekintsük át ezeket felülről lefelé haladva a folyamatfában, amely az init lecserélése esetén pl. így nézhet ki:

```
runit-+-events/0
[...]
      |-runsvdir-+-runsv-+-dnscache
      |   |      |       ‘-svlogd
      |   |      |-runsv-+-qmail-send-+-qmail-clean
      |   |      |       |            |-qmail-lspawn
      |   |      |       |            ‘-qmail-rspawn
      |   |      |       ‘-svlogd
      |   |      |-4*[runsv-+-socklog]
      |   |      |       ‘-svlogd]
      |   |      |-runsv---dd
      |   |      |-runsv---klogd
      |   |      |-runsv---cron
      |   |      |-runsv-+-munin-node
      |   |      |       ‘-svlogd
      |   |      |-8*[runsv---getty]
      |   |      |-runsv-+-ntpd
      |   |      |       ‘-svlogd
      |   |      |-runsv---sshd---sshd---zsh---screen
      |   |      |-runsv-+-smartd
      |   |      |       ‘-svlogd
      |   |      |-runsv---mdadm
      |   |      |-runsv---pound---pound---2*[{pound}]
 [...]
```

runit-init ♦ Ez a rendkívül egyszerű program, amely a `/sbin/init`et váltja ki, mindössze 76 sornyi C kód.

Ha 1-es a process ID-ja (vagyis ő az init), akkor elindítja (*execve()* hívással) a `runit`ot.

Ha más a process ID-ja, akkor úgy viselkedik, mint a `telinit`, tehát mintha runlevelt akarnánk váltani, de csak a 0-s és a 6-os runlevelt ismeri. Ezeknek a „runleveleknek” a hatása a következő:

Rendszerleállítás (0-s runlevel):

1. Létrehozza a `/etc/runit/stopit` fájlt (ez csak egy flag-fájl, a runit megnézi majd, hogy létezik-e és futtatható-e).
2. Futtathatóvá teszi.
3. Elveszi a jogokat a `/etc/runit/reboot` fájlról.
4. CONT signalt küld az 1-es folyamatnak (ami remélhetőleg a runit, és ettől majd elkezd leállni).

Újraindítás (6-os runlevel):

1. Létrehozza a `/etc/runit/stopit` fájlt (ez csak egy flag-fájl, a runit megnézi majd, hogy létezik-e és futtható-e).
2. Futtathatóvá teszi.
3. Létrehozza a `/etc/runit/reboot` fájlt (ez csak egy flag-fájl, a runit megnézi majd, hogy létezik-e és futtható-e).
4. Futtathatóvá teszi.
5. CONT signalt küld az 1-es folyamatnak (ami remélhetőleg a runit, és ettől majd elkezd leállni).

runit ♦ Ez a program való arra, hogy 1-es PID-del fusson és az init feladatait ellássa, vagyis:

– elindítsa a rendszert,

– leállítsa a rendszert és

– a kettő között gondoskodjon a szolgáltatások futtatásáról.

Ha init helyett a runit indul, akkor a `/etc/runit/1` nevű szkriptet indítja el először. Ennek a tartalma lehet pl. ilyesmi:

```
#!/bin/sh

PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
export PATH

/etc/init.d/rcS
/etc/init.d/makedev start
/etc/init.d/systune start
/etc/init.d/ud start
/etc/firewall/firewall start
/etc/init.d/rmnologin start
/etc/init.d/xend start
/etc/init.d/xendomains start

touch /etc/runit/stopit
chmod 0 /etc/runit/stopit
```

Az 1-es szkript tartalmazza a rendszerindításkor kiadandó parancsok listáját, emiatt nem hordozható korlátozás nélkül.

Ne feledjük, hogy a beállított és exportált környezeti változókat a szkript gyermekfolyamatai örökölni fogják.

Vegyük észre, hogy a fenti szkriptben használhattunk volna *rc.d* mechanizmust is a szolgáltatások elindítására, ami kicsit nehezebben átlátható, egy picit lassúbb, cserébe modulárisabb, hordozhatóbb és könnyebb automatizálni a karbantartását – de erre nem biztos, hogy van igény.

Konkrétan írhattuk volna akár azt is, hogy

```
[...]
/etc/init.d/rcS
/etc/init.d/rc 2
[...]
```

Így a „hagyományos” módon indult volna el a 2-es runlevel tartalma.

A `/etc/runit/1` után – micsoda meglepetés – a `/etc/runit/2` jön:

```
#!/bin/sh
PATH=/usr/local/bin:/usr/local/sbin:/bin:/sbin:/usr/bin:\
  /usr/sbin:/usr/X11R6/bin
exec env - PATH=$PATH \
runsvdir -P /var/service 'log: ......... [...] .........'
```

Innentől, amíg a runsvdir ki nem lép, ő monitorozza a szolgáltatásainkat. Ez tekinthető a rendszer normális, futó állapotának.

A `-P` kapcsoló hatására a runsvdir saját folyamatcsoportban indít el minden szolgáltatást. A `log:` és az őt követő sok pont egy olyan „helyet” hoz létre, ahová a szolgáltatások naplóüzeneteket írhatnak akkor is, ha sem a diszk, sem a hálózat nem elérhető – nevezetesen a

runsvdir folyamat *nevébe*. Az üzeneteket a ps outputjában olvashatjuk el. Tipikusan a naplózóprogramok hibaüzenetei kerülnek majd ide, hiszen ha naplózás közben lép fel hiba, az gyakran éppen a hálózati vagy a diszkre történő naplózás lehetetlenségének köszönhető, tehát másképpen nem biztos, hogy tudnánk naplózni ezeket az üzeneteket.

Láthatjuk, hogy a 2-es szkript lényegében nem tartalmaz rendszerspecifikus dolgot, tehát hordozható.

Ha kilép a runsvdir, kilép a `/etc/runit/2` is; ha a visszatérési értéke 0 volt, elindul a `/etc/runit/3`, ami a leállításért felelős.

Ha nem 0 volt a 2-es szkript visszatérési értéke, akkor a runit újra elindítja (feltételezi, hogy a runsvdir valamilyen hibával elszállt).

A 3-as szkript tartalma például a következő lehet:

```
#!/bin/sh
exec 2>&1
PATH=/sbin:/bin:/usr/sbin:/usr/bin
LAST=0
test -x /etc/runit/reboot && LAST=6
echo 'Waiting for services to stop...'
sv -w196 force-stop /var/service/*
sv exit /var/service/*
echo 'Shutdown...'
/etc/init.d/rc $LAST
```

Elvileg a 3-as szkript is lehetne rendszerfüggő, mivel a rendszer leállításával kapcsolatos parancsokat kell benne felsorolnunk, azonban gyakran nem az.

Ha megnyomjuk a Ctrl-Alt-Delt (és a kernelt úgy állítottuk be, hogy erről értesítse az initet), vagy INT signalt küldünk az 1-es PID-jű folyamatnak, akkor a runit megnézi, létezik és futtatható-e a `/etc/runit/ctrlaltdel` szkript; ha igen, lefuttatja, majd küld magának egy CONT signalt.

A CONT signal hatása az, hogy ha létezik és végrehajtható a `/etc/runit/stopit` fájl, akkor a runit leállítja a rendszert; kilövi a 2-es szkriptet, és rátér a 3-asra. Ha a `/etc/runit/reboot` is létezik és végrehajtható, akkor leállítás helyett a szolgáltatások leállítása után újraindítja a gépet.

A `/etc/runit/ctrlaltdel` szkript tartalma lehet pl. ilyesmi:

```
#!/bin/sh
PATH=/bin:/usr/bin
MSG="System is going down in 14 seconds..."
touch /etc/runit/stopit
chmod 100 /etc/runit/stopit && echo "$MSG" | wall
/bin/sleep 14
```

runvsdir ♦ A runsvdir feladata a runsv-folyamatok menedzselése.

Normális körülmények között a `/etc/runit/2` indítja el, de bárki futtathatja (pl. ennek a segítségével csinálhat akár saját magának megbízható szolgáltatásokat egy felhasználó).

A runsvdir odavált a megadott könyvtárba (itt `/var/service`, amire mutató symlinket a daemontools-hoz szokott rendszergazda létrehoz a gyökérben), és minden (vagy legfeljebb 1000) itt található alkönyvtárhoz és alkönyvtárra mutató symlinkhez indít egy runsv-t (ehhez a runsv-nek benne kell lennie a PATH-ban). A ponttal kezdődő nevű könyvtárakat kihagyja. Ha valamelyik runsv kilép, a runsvdir újraindítja.

Öt másodpercenként megnézi, megváltozott-e a paraméterként kapott könyvtár mtime-ja, inode-ja, vagy az, hogy melyik blokkeszközön található. Ha megváltozott, akkor megnézi, van-e benne új alkönyvtár, vagy alkönyvtárra mutató symlink (ha igen, indít hozzá runsv-t), vagy szűnt-e meg régi (ekkor leállítja az adott runsv-t és a szolgáltatást, ami hozzá tartozott).

runsv ♦ A runsv feladata egy szolgáltatás (és esetleg egy hozzá tartozó naplózószolgáltatás) monitorozása, elindítása, leállítása.

Egyetlen paramétert kap indításkor, egy könyvtár nevét. Tevékenysége így foglalható össze:

1. Induláskor odavált, és lefuttatja a `./run` programot.
2. Ha ez kilép, és létezik a `./finish`, lefuttatja azt.
3. Ha a `./finish` nem létezik, vagy kilép, újra elindítja a `./run`-t.

Ha a szkriptek „azonnal” kilépnek, a runsv egy másodpercet vár az újraindítási kísérletek között.

Ha létezik a `./down` nevű fájl, a runsv nem indítja el azonnal a `./run`-t, csak akkor, ha erre külön megkérjük.

Ha létezik a `./log` könyvtár, akkor a runsv létrehoz egy csővezetéket, amellyel a `./run` és a `./finish` standard kimenetét a `log/run` standard bemenetére irányítja (ezt azért tudja megtenni, mert a saját gyermekfolyamatairól van szó, akik tőle öröklik a standard fájlleírókat).

A továbbiakban a log könyvtárat a runsv a főszolgáltatás alszolgáltatásaként kezeli; a `log/run`-t is futtatja; újraindítja, ha megáll; és írhatunk log/finish scriptet is. A log-alszolgáltatást a főszolgáltatástól függetlenül leállíthatjuk, újraindíthatjuk stb.; ráadásul a pipe-nak köszönhetően a főszolgáltatás üzenetei akkor is gond nélkül eljutnak a `log/run` standard bemenetére, ha a főszolgáltatás, vagy a naplózás – vagy mindkettő – chrootban fut.

A runsv létrehoz egy `./supervise` (és esetleg `./log/supervise`, ha van `./log`) könyvtárat. Ezek tartalma:

– `status`: bináris fájl, ami a szolgáltatás állapotával kapcsolatos információkat tartalmazza; kompatibilis a daemontools *supervise* programjának `status` fájljával.

– `stat`: szövegfájl. Tartalma lehet:

  - „run”: a szolgáltatás fut;
  - „down”: a szolgáltatás nem fut;
  - „finish”: a szolgáltatáshoz tartozó finish program fut.

  A fentiek még kiegészülhetnek a következőkkel:

  - „paused”: a szolgáltatást felfüggesztettük;
  - „got TERM”: a szolgáltatásnak küldtünk TERM signalt, de még nem lépett ki;
  - „want down”: a szolgáltatást le szeretnénk állítani, de még nem állt le;
  - „want exit”: a runsv-t is le szeretnénk állítani, de ehhez előbb a szolgáltatást kell, az pedig még nem állt le.

– `pid`: a szolgáltatáshoz tartozó éppen futó program (run vagy finish) process ID-ját tartalmazza; a hagyományos pidfájlokkal ellentétben ennek a tartalmát (a versenyhelyzetektől eltekintve) készpénznek vehetjük, hiszen a runsv tudja, mi a saját gyermekfolyamatának a PID-ja, és mindig azt írja ebbe a fájlba.

– `control`: egy fifo, amibe parancsokat írhatunk (ezt csinálja az *sv* program, lásd később):

  - u: „up”. Ha a szolgáltatás nem fut, elindítja; amikor kilép, újraindítja.

- d: „down”. Ha a szolgáltatás fut, kap egy TERM, majd egy CONT signalt (hogy a TERMre akkor is tudjon reagálni, ha fel volt függesztve). Ha a `./run` kilép, lefut a `./finish`; amikor az is kilép, a runsv nem indítja újra a szolgáltatást.
- o: „once”. Ha a szolgáltatás nem fut, elindítja. Amikor kilép, nem indítja újra (de a `finish` lefut).
- p: „pause”. Ha a szolgáltatás fut, felfüggeszti (STOP signalt küld neki).
- c: „continue”. Ha a szolgáltatás fut, folytatja (CONT signalt küld).
- h: „hangup”. Ha a szolgáltatás fut, HUP signalt küld neki.
- a: „alarm”. ALARM signal.
- i: „interrupt”. INT signal.
- q: „quit”. QUIT signal.
- 1: USR1 signal.
- 2: USR2 signal.
- t: „terminate”. TERM signal. Ha a szolgáltatás „up” állapotban volt, és a TERM hatására kilép, akkor ez lényegében egy újraindítással egyenértékű, mert a runsv újra el fogja indítani.
- k: „kill”. KILL signal.
- x: „exit”:

1. Ha a szolgáltatás fut, kap egy TERM, majd egy CONT signalt.
2. Amikor kilép, lefut a finish.
3. Ha nincs log alszolgáltatás, a runsv kilép.
4. Ha van log, a runsv lezárja a log standard inputját, és megvárja, amíg a `log/run` is kilép.
5. Lefut a `log/finish`.
6. A runsv kilép.
7. A log szolgáltatásnak nem lehet exit parancsot küldeni (illetve lehet, de ignorálja).

Gondoljunk arra, hogy ha nem fut a runsv (és így nincs, aki elvegye a fifoba írt karaktert), akkor a fifoba író művelet alapértelmezés szerint blokkolódni fog.

Ha a runsv TERM signalt kap, úgy tesz, mintha „x” parancsot kapott volna.

A runsv parancsok hatásának testreszabása ♦ Megtehetjük, hogy a fenti parancsok hatását megváltoztatjuk. Ehhez létre kell hoznunk a szolgáltatáskönyvtárban egy control nevű alkönyvtárat, amiben a parancsokról elnevezett programokat helyezhetünk el. Ha olyan parancsot adunk ki, amelyhez tartozó control program létezik és futtatható, akkor a runsv

1. lefuttatja a `control/`*`parancs`* programot, és megvárja, amíg kilép;
2. ha a visszatérési érték sikeres, a runsv nem kézbesíti a kért signalt a szolgáltatásnak;
3. ha sikertelen, akkor mégis kézbesíti.

A naplózó alszolgáltatás vezérlése nem testreszabható.

sv ♦ Az *sv* feladata az, hogy információt szolgáltasson a runsv által menedzselt szolgáltatások állapotáról, illetve hogy parancsokat adjon a runsv-nek. Képes továbbá korlátozottan emulálni egy SystemV initszkript működését.

Indítása:

```
# sv [-v] [-w sec] command services
```

vagy

```
# /etc/init.d/service [-w sec] command
```

A *services* helyén felsorolhatunk egy, vagy több szolgáltatást, amikre hatni szeretnénk; a hozzájuk tartozó runsv-s könyvtár nevét kell megadni.

A *command* lehet a runsv által ismert egy karakteres parancsok valamelyike (de kiírhatjuk a teljes parancsot is, mert csak az első karaktert nézi), valamint a következők, amelyek a SystemV initszkript emulációt segítik:

- start: mint az „up”, de megvárja, hogy a szolgáltatás el is induljon (alapértelmezés szerint 7 másodpercet). A szolgáltatáshoz tartozó `check` szkripttel ellenőrzi, hogy elindult-e, vagy ha ilyen nincs, akkor azt várja meg, hogy epszilonnál hosszabb ideje fusson. Ha 7 (vagy `sec`) másodperc után nincs eredmény, kilép, a szolgáltatás pedig továbbra is próbál elindulni.
- stop: mint a „down”, de megvárja, hogy a szolgáltatás le is álljon (mint fent).
- restart: mint egy „down”, majd egy „up” egymás után.
- shutdown: mint az „exit”, de megvárja (timeout mint fent), hogy a runsv is kilépjen.
- force-stop: mint a „stop”, de időtúllépés esetén KILL signalt küld a szolgáltatásnak.
- force-reload: mint egy „term” és egy „continue”, de ha nem áll le időben, kap egy KILL signalt is.
- force-restart: mint a „restart”, de ha nem áll le időben, kap egy KILL signalt is.
- force-shutdown: mint a „shutdown”, de ha nem áll le időben, kap egy KILL signalt is.

Emellett van még egy `check` parancs, ami azt vizsgálja meg, hogy a szolgáltatás az általunk utoljára kért állapotban van-e, illetve ha nincs, akkor a megadott ideig vár arra, hogy oda eljusson.

Az initszkript-emuláció értelme az, hogy ha pl. van egy apache szolgáltatásunk, akkor megcsinálhassuk ezt (debianos példa):

```
# dpkg-divert --local --rename /etc/init.d/apache
# ln -s /usr/bin/sv /etc/init.d/apache
```

Így a `/etc/init.d/apache` segítségével majdnem a szokásos módon kezelhetjük az apache-t: működik a stop, a restart, a start és társai, mert az sv megnézi, milyen néven hívtuk meg, és az olyan nevű szolgáltatást próbálja menedzselni. Sajnos pl. „reload” akciót nem tudunk csinálni, ezt az sv egyelőre nem támogatja – pedig a check mintájára nem lenne nehéz. Ez nyilván nem jelenti azt, hogy nem tudjuk újraolvastatni az apache-val a konfigurációját, ha az apache amúgy képes erre (mellesleg képes), csak azt, hogy a runit ehhez nem nyújt nekünk segítséget.

Runlevel-emuláció: runsvchdir ♦ Ha nagyon a szívünkhöz nőttek a runlevelek, runit mellett is használhatjuk őket; ráadásul tetszőlegesen sok lehet belőlük, és szabadon adhatunk nekik neveket.

Az elgondolás lényege az, hogy több `/var/service`-szerű könyvtárat csinálunk (runlevelenként egyet), más-más szolgáltatáskönyvtárak symlinkjeivel, és a runsvdirt rábeszéljük, hogy váltson át egy másikba, és értelemszerűen állítsa le azokat a szolgáltatásokat, amelyeknek az új könyvtárban nincs symlinkje, azokat pedig, amelyeknek az előzőben nem volt, de itt van, indítsa el. Ehhez

– létre kell hoznunk valahol (alapértelmezés szerint a `/etc/runit/runsvdir` könyvtárban) egy-egy ilyen runlevel-könyvtárat;

– ebben létre kell hoznunk egy „current” nevű symlinket, ami az aktuális runlevelhez tartozó runlevel-könyvtárra mutat;

– a `/var/service`-t (vagy ami a `/etc/runit/2`-ben a runsvdir paramétere) le kell cserélnünk egy symlinkre, ami erre a current nevű symlinkre mutat.

Ezután a current symlink átállítása runlevelváltást eredményez. Ezt csinálja meg a *runsvchdir* parancs, és ráadásul egy, az éppen elhagyott runlevelre mutató symlinket is létrehoz „previous” néven. Ez azért jó, mert attól, hogy a runsv folyamatok TERM signalt küldtek a lelövendő szolgáltatásoknak, még nem biztos, hogy mindegyik le is állt; a „previous” symlinken keresztül követhetjük nyomon a legegyszerűbben, hogy az előző runlevelben futott szolgáltatások státusza hogyan alakul.

chpst ♦ A *chpst* feladata az, hogy valamilyen környezetet beállítson, és abban elindítson egy gyermekfolyamatot. Ennek megfelelően a unixos folyamatok állapotterének jelentős részét tetszésünk szerint beállíthatjuk:

– UID;
– GID;
– *supplementary groups* („kiegészítő csoporttagságok”);
– környezeti változók (a daemontools envdir-jéhez hasonló módon; lásd lejjebb);
– chroot;
– nice;
– kölcsönös kizárás (a megadott program csak egy példányban futhasson);
– erőforráskorlátok:
  • memória (vagy az összes, vagy csak az adatszegmens);
  • megnyitható fájlok száma;
  • folyamatok száma;
  • létrehozható fájlok maximális mérete;
  • core fájl max. mérete;
– új process group létrehozása a program elindítása előtt;
– standard fájldeszkriptorok bezárása (input, output, error szelektíven) a program elindítása előtt.

Ami hiányzik, pedig nem lenne rossz:

– scheduler beállítása (lehetne pl. realtime, bár ez nem hordozható);
– capability-k beállítása (szintén nem hordozható);
– umask;
– munkakönyvtár;
– max. CPU-idő;
– hard erőforráskorlátok (csak a soft limiteket állítja, de nem tudja őket nagyobbra állítani a hard limiteknél, vagyis ha van valamilyen default hard limit, annál nagyobb soft limitet a chpst nem tud beállítani; előtte ulimittel kell módosítani a hard limitet);
– supplementary groupok megtartása (eldobja őket).

A környezeti változók beállítása úgy történik, hogy létrehozunk egy könyvtárat, amiben minden fájl egy beállítandó környezeti változóról van elnevezve. A változó értéke az adott fájl tartalmának első sora lesz. A nullás kódú karaktereket újsorrá alakítja, a sorvégi szóközöket és tabulátorokat pedig lenyeli. Ha a fájl nulla bájt hosszú, akkor a róla elnevezett változót a chpst törli a környezetből.

A kölcsönös kizárás megvalósításához a chpst megpróbál lockolni egy fájlt; ha sikerül, akkor elindítja a megadott programot úgy, hogy az megkapja a lockolt fájlt is. Ha az elindított program bezárja az öröklött fájlleírót, akkor a kölcsönös kizárás nem biztosított; szerencsére messze nem minden program zárja be az összes fájlleíróját. Ezt a mechanizmust használhatjuk pl. arra is, hogy egy cron-feladat ne tudjon elindulni, ha az előző inkarnációja még fut.

svlogd ♦ Az svlogd feladata az, hogy a standard inputon érkező sorokat automatikusan rotált naplófájlokba írja.

A naplózó alszolgáltatások általában az svlogd-t futtatják; az svlogd egyébként sokkal okosabb, mint a *daemontools* multilogja.

A naplózás a következőképpen működik:

– Az svlogd kilép, ha a standard inputon fájl végét olvas.
– Egy svlogd-példány egy vagy több könyvtárba logol.
– Egy könyvtárba csak egy svlogd-példány logolhat.[2]
– Minden könyvtárnak saját konfigurációja van, ami eldöntheti, hogy az adott könyvtárba a bemeneten olvasott sorok közül melyeket kell naplózni és melyeket nem.
– Egy sor több könyvtárba is naplózódhat.

  A könyvtárakban a következő fájlok találhatóak:

  • `current`: az aktuális logfájl, amibe éppen ír az svlogd.
  • @4000000044b600e22b73217c.s jellegű fájlok korábbi naplófájlok, amelyeknek az utófeldolgozása sikeres volt; a nevük a rotáció időpontja *tai64n* időformátumban.
  • @4000000044b600e22b73217c.u jellegű fájlok: mint fent, de nem történt utófeldolgozás.

[2] Erre pl. az apache2 esetében oda kell figyelni: ha a hagyományos módon, initszkripttel, háttérben futva akarjuk futtatni, és egy `ErrorLog` direktívában `|svlogd /var/log/apache2/error` jellegű naplót állítottunk be, akkor az előtérben futó apache2 indít egy svlogd-t, mielőtt elindítaná saját második példányát, amely szintén indít egy ugyanoda logoló svlogd-t, még azelőtt, hogy az előtérben futó apache2-példány kiléphetne és lelőhetné a saját svlogd-jét. A két svlogd egyszerre próbál ugyanabba a könyvtárba írni, aminek az lesz a következménye, hogy a második kilép – az apache2 viszont nem indítja újra.

- Az utófeldolgozás közben még megjelenhetnek rövid életű `.t` kiterjesztésű fájlok és egy previous nevű fájl.
- `config`: az svlogd adott könyvtárra vonatkozó konfigurációját tartalmazza. A formátum egyszerű: egykarakteres, sor eleji parancsok, majd a hozzájuk tartozó argumentumok.
- Van még néhány egyéb fájl (`state`, `newstate`, `lock`), amiket az svlogd saját céljaira használ.

A config nevű fájlban a következő parancsokat használhatjuk:

– s*méret*: a naplófájlokat rotálni kell, ha elérték a megadott *méret*et. Ha nulla, nincs méret alapján történő rotáció.

– n*darab*: a megadott *darab*számú korábbi naplófájlt tartunk meg. Ha 0, nem töröl régi naplókat.

– N*darab*: legalább a megadott *darab*számú korábbi naplóájlt tartunk meg. Ha elfogyott a hely, és ennél több régi log van, az svlogd törli a legrégebbit.

– t*másodperc*: ha az aktuális napló elérte a megadott kort, és nem üres, akkor rotálunk. Ha 0, nincs időalapú rotáció.

– !*posztprocesszor*: rotációkor a naplót a megadott program (pl. egy tömörítő vagy egy elemző) dolgozza fel. Alapértelmezés szerint nincs utófeldolgozás.

– u*a.b.c.d[:port]*: a megadott IP-cím megadott portjára is elküldi az üzenetet (UDP-vel) (valójában csak az első *hossz* bájtot, a *hossz* megadható a parancssorban).

– U*a.b.c.d[:port]*: mint a kis u, de *csak* UDP-n logol, a helyi logfájlba nem írja bele az üzenetet.

– p*prefix*: a megadott *prefix*et odailleszti az összes naplózott üzenet elé. Ez akkor jó, ha UDP-n logolunk, és a fogadóoldalon könnyen szét akarjuk válogatni az üzeneteket; a prefix alapján lesz a legegyszerűbb.

– -*minta*: a *mintá*ra illeszkedő sorokat nem naplózza (lásd lejjebb).

– +*minta*: a *mintá*ra illeszkedő sorokat mégis naplózza.

– e*minta*: a *mintá*ra illeszkedő sorokat kiírja a standard errorra (lásd lejjebb).

– E*minta*: a *mintá*ra illeszkedő sorokat mégsem írja ki a standard errorra.

A posztprocesszort az svlogd a háttérben futtatja, úgyhogy közben megy tovább a logolás. Ha viszont a következő rotációkor még fut az előző processzor, az svlogd blokkolódik, amíg ki nem lép; ez blokkolhatja az svlogd-be logoló szolgáltatást is! A posztprocesszort érdemes lehet a *tryto* felügyelete alatt futtatni; ez néhányszor megpróbálja, de ha egyszer sem sikerül adott idő alatt befejezni, akkor inkább lemond a feldolgozásról és feldolgozás nélkül menti el a logot. A *tryto* a *socklog* csomagban található.

A mintaillesztés két metakaraktert ismer:

– A csillag jelentése: tetszőleges sztring, amely nem tartalmazza a csillagot követő karaktert;

– A plusz jelentése: a pluszt követő karakterből egy vagy tetszőleges számú.

Vegyük észre, hogy ez sokkal gyengébb, mint a reguláris kifejezések, cserébe viszont sokkal gyorsabb az illesztés, és az esetek döntő részében még a pluszra sincs szükség.

Pl. az a minta, hogy „*PAM_unix[*]: (ssh)*” illeszkedik minden olyan sztringre, amelyben az első nagy P betű után az jön, hogy „AM_unix[”, majd egy tetszőleges sztring a következő „]” jelig, amelyet a „: (ssh)” sztring követ, ezután pedig még lehet bármi a sorban. Az a sor, hogy „Péntek: PAM_unix[1234]: (ssh) session opened for user root” nem illeszkedik, mert a „Péntek” P-betűje után nem az jön, hogy „AM_unix[”.

A minták segítségével könnyen szétdobálhatjuk pl. a sokféle „daemon” üzenetet aszerint, hogy melyik szolgáltatás naplózta; vagy pl. naplózhatjuk egy könyvtárba az összes, az sshval kapcsolatos üzenetet attól függetlenül, hogy „auth” vagy „daemon” syslog facility-vel érkezett-e.

Az svlogd parancssori opcióival megadhatjuk, hogy

- minden sor elejére illesszen egy *tai64n* időbélyeget (`-t`);
- minden sor elejére illesszen egy lexikografikusan rendezhető, ember számára olvasható UTC időbélyeget (`-tt`);
- cserélje a nem nyomtatható karaktereket valami másra (`-r`);
- egyéb karaktereket is cseréljen (pl. szóközöket és tabokat válogatás nélkül aláhúzásra);
- mekkora pufferérettel dolgozzon. Ezt nagyon nagy log-forgalom esetén érdemes az alapértelmezés szerinti 1024 bájtról 4, vagy akár 16–64 kilobájtra is emelni, ha azt tapasztaljuk, hogy a szolgáltatás arra vár, hogy logolhasson; a legkönnyebben úgy jöhetünk rá, hogy emiatt lassú a naplózás (és így a szolgáltatás), ha strace-eljük az svlogd-t, és azt látjuk, hogy mindig teli pufferrel tér vissza a read() rendszerhívása. Normális esetben több karaktert próbál olvasni, mint amennyi van.

Az svlogd a következő signalok hatására tesz valami különlegeset:

- HUP: az összes log-könyvtár konfigurációját újraolvassa, a log-könyvtárakat újra megnyitja; ha valamelyik megszűnt, azt eldobja.
- TERM: feldolgozza a pufferében levő adatokat (eldönti, hova kell logolni az adott sorokat és kiírja őket), megvárja, hogy véget érjenek az esetleg futó posztprocesszorok, majd kilép.
- ALRM: rotálja az összes nem üres logot.

A standard errorra való logolást pl. arra lehet használni, hogy bizonyos üzenetekről e-mailt kapjunk. Ennek a funkciónak a bemutatása túlmutat a cikk keretein; keressünk a „socklog-notify” szókapcsolatra a weben.

utmpset ♦ Kis segédprogram, amelyet a *getty* jellegű szolgáltatások finish szkriptjéből kell meghívni; a bejelentkezési naplóban rögzíti, hogy az adott terminálon véget ért a login session. Ezt amúgy az init csinálná, de a runitban nincs benne az ezzel kapcsolatos kód.

4.3. Runitos rendszer építése

Hogyan is érdemes egy runitos rendszert felépíteni?

Törekedjünk arra, hogy minden szolgáltatásunkat a runit monitorozza, és lehetőleg a standard outputra naplózzanak, így mindegyiknek egyéni svlogdje lehessen, és ne függjenek egy

rendszerszintű naplózószolgáltatástól. Az svlogd-k naplófájljai külön logikai köteten legyenek, hogy semmi ne használhassa el előlük a helyet. Ez azért fontos, mert önmagukban az svlogd naplóinak méretére (ha sehol nem nulla sem az *n*, sem az *s* érték) mindig van érvényes felső becslésünk, tehát ha más nem ír oda, és elég helyet csináltunk, akkor a hely biztosan nem fogy el. Ha elfogy a hely, az azért baj, mert blokkolódhatnak a szolgáltatásaink.

A rendszerszintű naplózást célszerű a socklogra bízni.

Próbáljunk gondoskodni arról, hogy a szolgáltatásaink, ha valami bajuk van, akkor ne megálljanak, hanem kilépjenek, hogy a runit újraindíthassa őket.

Mindig törekedjünk a kód és a konfiguráció szétválasztására, hogy a szkriptjeink hordozhatóak legyenek.

run szkriptek írása kezdőknek ♦ Egy szolgáltatást futtató run szkript alapreceptje a következő:

1. standard error átirányítása a standard outputra (hogy a logban jelenjen meg);
2. a szolgáltatás elindítása az `exec` paranccsal olyan módon, hogy a meghívott parancssor csak akkor térjen vissza, amikor kilép a szolgáltatás.

Egy minimális run szkript valahogy így fest (rinetd):

```
#!/bin/sh
exec 2>&1
exec rinetd -f
```

A szolgáltatások egymás közötti függőségeit többek között úgy tudjuk érvényre juttatni, ha a függő szolgáltatás run szkriptjének elején elhelyezünk egy „`sv start` *`szükséges-szolgáltatás`* `|| exit 1`” jellegű sort. Ez szavatolni fogja, hogy a függő szolgáltatás addig nem is próbál meg elindulni, amíg a futásához szükséges szolgáltatás el nem indult.

Ha új run szkriptet írunk, akkor kiindulhatunk a szolgáltatás initszkriptjéből. Nézzünk egy példát, mondjuk az apache2-t. Az initszkript a következő (csak az indításért felelős részt hagytam benne):

```
#!/bin/bash -e
#
# apache2              This init.d script is used to start apache2.
#                      It basically just calls apache2ctl.

ENV="env -i LANG=C PATH=/usr/local/bin:/usr/bin:/bin"

#edit /etc/default/apache2 to change this.
NO_START=0

set -e
if [ -x /usr/sbin/apache2 ] ; then
  HAVE_APACHE2=1
else
  exit 0
fi

. /lib/lsb/init-functions

test -f /etc/default/rcS && . /etc/default/rcS
test -f /etc/default/apache2 && . /etc/default/apache2
if [ "$NO_START" != "0" -a "$1" != "stop" ]; then
```

```
  [ "$VERBOSE" != "no" ] && log_warning_msg "Not starting apache2 - \
edit /etc/default/apache2 and change NO_START to be 0.";
  exit 0;
fi

APACHE2="$ENV /usr/sbin/apache2"
APACHE2CTL="$ENV /usr/sbin/apache2ctl"

# Stupid hack to keep lintian happy. (Warrk! Stupidhack!).
case $1 in
 start)
  [ -f /etc/apache2/httpd.conf ] || touch /etc/apache2/httpd.conf
  # ssl_scache shouldn't be here if we're just starting up.
  [ -f /var/run/apache2/ssl_scache ] && rm -f /var/run/apache2/*ssl_scache*
  # /var/run and /var/lock could be on a tmpfs
  [ ! -d /var/run/apache2 ] && mkdir /var/run/apache2
  [ ! -d /var/lock/apache2 ] && mkdir /var/lock/apache2
  # Make sure /var/lock/apache2 has the correct permissions
  chown www-data /var/lock/apache2
  log_begin_msg "Starting apache 2.0 web server..."
  if $APACHE2CTL startssl; then
    log_end_msg 0
  else
    log_end_msg 1
  fi
esac
```

Ugye emlékszünk, milyen hosszú volt a leállításért felelős rész – mindazt nem kell megír-nunk, a runit elintézi (kivéve, ha nagyon buta dolgokat csinál a szolgáltatásunk – pl. a szülőfolyamat ki tud lépni anélkül, hogy az összes gyermekét kiléptetné).

Csak érdekességképpen nézzük meg, mit is importálunk a `/lib/lsb/init-functions` fájlból – bár azért nem másolom be mind a 290 sort:

- start_daemon() – egy wrapper a *start-stop-daemon* köré.
- pidofproc() – megpróbálja megtippelni egy szolgáltatás PID-ját – de pl. vakon bízik a pidfájl tartalmában.
- killproc() – mint a pidofproc, de még meg is próbálja kilőni a megtalált folyamtot.
- log_use_fancy_output() – a csicsaképességet detektálja; a többi függvény gyakran hívogatja, ez a függvény viszont külső programot is hív, úgyhogy a párhuzamosítás révén elért gyorsulásnak talán búcsút is mondhatunk.
- log_success_msg(): `echo "$@"` – emiatt aztán érdemes volt külön függvényt írni! Idővel majd bizonyára ez is csicsa lesz.
- log_warning_msg() – csicsa.
- get_lsb_header_val() – az initszkriptben elhelyezett kommentekből információt kibányászó segédfüggvény pl. komplikált függőségkezeléshez.
- log_daemon_msg() – csicsa annak érdekében, hogy ugyanaz az initszkript futhasson különböző disztribúciók alatt, és az outputja igazodjon az adott disztribúció initszkript-look'n'feeljéhez. Hja kérem, fontos dolgok...
- log_progress_msg() – csicsa.
- log_end_msg() – csicsa.

– log_action_msg(): `echo "$@."` – csicsázás számára fenntartott hely.
– log_action_begin_msg(): `echo -n "$@..."` – dettó.
– log_action_cont_msg(): `echo -n "$@..."` – dettó.
– log_action_end_msg() – csicsa.

Mérleg: 10 db csicsázó függvény, 2 db esetleges PID-tippelő, 1 db értelmesnek nevezhető függvény. Az apache2 elindításához szigorúan véve egyiknek sincs köze.

Az a bizonyos `/usr/sbin/apache2ctl`, amit az initszkript meghív, szintén egy shell-szkript. 103 sor hosszú, de „start” paraméter esetén a tevékenysége így foglalható össze:

```
/usr/sbin/apache2 -k $@
```

A szkript „startssl” esetén pedig ezt csinálja:

```
/usr/sbin/apache2 -k start -DSSL
```

Ha ezt összevetjük azzal a több száz sornyi kóddal, amit a disztribúciónk lefuttat az apache2 elindítása során, talán nem minden alap nélkül merülhet fel bennünk az „aránytévesztés” szó.

Írjunk első közelítésben egy egyszerű run szkriptet a fentiek alapján:

```
#!/bin/bash
exec 2>&1
#edit /etc/default/apache2 to change this.
NO_START=0
MAXFILES=1024
test -f /etc/default/apache2 && . /etc/default/apache2
```

Beolvastuk a Debian-féle konfigot a kompatibilitás érdekében.

```
chown :apacheadmin ./supervise ./supervise/control
chmod g+w ./supervise/control
chmod g+x ./supervise
```

Jogokat adtunk az apacheadmin csoportnak a szolgáltatás menedzselésére.

```
[ -x /usr/sbin/apache2 ] || exec sv down .
```

Ha nincs apache bináris, leállítjuk magunkat.

```
[ "$NO_START" != "0" ] && exec sv down .
```

Ha a configfile szerint nem kell futnunk, akkor kilépünk.

```
[ -f /etc/apache2/httpd.conf ] || touch /etc/apache2/httpd.conf
```

A debianos init-szkript létrehozta ezt a fájlt, ha nem létezett, így tettünk hát mi is.

```
[ -f /var/run/apache2/ssl_scache ] && rm -f /var/run/apache2/*ssl_scache*
[ ! -d /var/run/apache2 ] && mkdir /var/run/apache2
[ ! -d /var/lock/apache2 ] && mkdir /var/lock/apache2
chown www-data /var/lock/apache2
```

A fenti kód a debianos szkriptből lett átemelve.

```
ulimit -H -n $MAXFILES
```

A chpst csak a softlimiteket állítja, a hardot nekünk kellett.

```
exec chpst -o $MAXFILES /usr/sbin/apache2 -k start -DSSL -DNO_DETACH
```

A fentit pedig az apache2ctl-ből puskáztuk.

Ezzel véget is ér a szkript, amely így már működik, de nem túl hordozható. Bele van drótozva a *www-data* felhasználó, az *apacheadmin* csoport, és egy csomó könyvtár helye. Nem választottuk szét a kódot és a konfigurációt.

Egy haladó run szkript ♦ Próbáljuk meg még egyszer, kicsit általánosabban:

```
#!/bin/bash
exec 2>&1
SVNAME=$(basename $(pwd))
```

Kitalálja, hogy hívják a szolgáltatást, amiben fut (pl. apache2-svn).

```
SVCONFIG=/etc/default/$SVNAME
```

Ez lesz az erre a példányra vonatkozó configfájl.

```
MAXFILES=1024
RUNASUSER=www-data
SERVERROOT=/etc/$SVNAME
APACHECONFIGFILE=$SERVERROOT/apache2.conf
ADMINGROUP=apacheadmin
VARRUNGROUP=root
VARRUNMODE=700
SSL="-DSSL"
```

Néhány alapértelmezést állít be.

```
[ -r /etc/default/apache2 ] && . /etc/default/apache2
```

Betölti a debian-configot; ez is felülbírálhatja a fentieket, mondjuk gépenként más és más lehet.

```
[ -r $SVCONFIG ] && . $SVCONFIG
```

Betölti a specifikus configot (ami átírhatja a fenti defaultokat).

```
chown :$ADMINGROUP ./supervise ./supervise/control
chmod g+w ./supervise/control
chmod g+x ./supervise

[ -x /usr/sbin/apache2 ] || exec sv down .
[ "$NO_START" != "0" ]   && exec sv down .

[ -f $SERVERROOT/httpd.conf ] || touch $SERVERROOT/httpd.conf
rm -f /var/run/$SVNAME/*ssl_scache* 2>/dev/null
```

Felesleges a törlendő fájlok létezéséről meggyőződni; nem okoz bajt, ha egyszerűen megpróbáljuk letörölni őket.

```
mkdir -p /var/run/$SVNAME /var/lock/$SVNAME
chown $RUNASUSER:$VARRUNGROUP /var/run/$SVNAME /var/lock/$SVNAME
chmod $VARRUNMODE /var/run/$SVNAME /var/lock/$SVNAME
```

A `/var` alatti könyvtárakat konfigurálható jogokkal hozza létre.

```
pkill -u $RUNASUSER apache2
```

Csúf, de az apache2 esetében sajnos szükség lehet rá; mentségünk, hogy csak az adott felhasználóként futó apache2-ket lőjük le, ilyen felhasználóként pedig akkor csak azok fussanak, amik hozzánk tartoznak.

```
ulimit -H -n $MAXFILES
exec chpst -o $MAXFILES \
        sudo -u $RUNASUSER \
        /usr/sbin/apache2 \
                -d "$SERVERROOT" \
                $SSL \
                -DNO_DETACH \
                -f "$APACHECONFIGFILE"
```

Azért sudo, mert az belép a megfelelő supplementary groupokba, a chpst pedig eldobná őket, ha nem soroljuk fel explicit módon.

Ha ezt a run szkriptet bemásoljuk egy apache2-svn nevű könyvtárba, és belinkeljük a `/var/service`-ba, akkor a szkript, amikor elindul, és az apache2-svn-re jellemző konfigurációt fogja beolvasni a `/etc/default/apache2-svn`-ből. A kód és a konfiguráció jól elkülönül; a szkript vándorolhat gépről gépre anélkül, hogy módosítani kellene. Elég a `/etc/default/apache2-*` fájlokat módosítani.

Vegyük észre, hogy itt a sudo miatt eleve nem rootként indul el az apache2, tehát nem fog tudni 1024 alatti porton figyelni, hacsak ezt valahogyan nem engedélyeztük a `RUNASUSER` számára.

Az ilyen sablonszerű run szkripteket úgy ugyan nem használhatjuk, hogy ugyanazt a könyvtárat több példányban, csak mindig más néven belinkeljük a `/var/service` alá, mivel mindegyikhez tartozó `runsv` ugyanazt a supervise alkönyvtárat akarná használni; annak viszont természetesen nincs akadálya, hogy a run szkriptet az egyes példányok saját könyvtáraiban egy a template run szkriptre mutató symlinkkel helyettesítsük. Így a run szkriptet könnyedén tarthatjuk pl. Subversion repositoryban.

Logoló run szkript ♦ Egy minimális, svlogd-t indító run szkript valahogy így fest:

```
#!/bin/sh
exec chpst -u log svlogd /var/log/sv/szolgaltatasneve
```

Ezzel az a baj, hogy ha a szolgáltatást egy új gépre másoljuk, ott még esetleg nincs meg ez a könyvtár, és ebben az esetben az svlogd csődöt mond. Egy kicsit feltupírozott sablon-logoló szkript festhet pl. így:

```
#!/bin/sh
cd ..
SVNAME=$(basename $(pwd))
LOGDIR=/var/log/sv/$SVNAME
LOGDIRMODE=700
LOGDIRGROUP=root
LOGUSER=log
CONFIG=/etc/default/$SVNAME
SVLOGDOPTS="-t"
ROTSIZE=50000
ROTNUM=50
POSTPROC="tryto -pP gzip -9"

[[ -r /etc/default/svlogd ]] && . /etc/default/svlogd
```

Ez utóbbi sor a rendszerszintű svlogd-alapbeállításokat tölti be.

```
[[ -r $CONFIG ]] && . $CONFIG
```

Az erre a konkrét szolgáltatásra vonatkozó beállításokat tölti be.

```
mkdir -p $LOGDIR
chown $LOGUSER:$LOGDIRGROUP $LOGDIR
chmod $LOGDIRMODE $LOGDIR
cd $LOGDIR || exit 1
if [[ ! -f config ]]; then
        cat <<EOF >config
s$ROTSIZE
n$ROTNUM
!$POSTPROC
EOF
fi
```

5. Zárszó

A runit nem az egyetlen olyan programcsomag, amely megpróbál a ma uralkodó „featurista” irányzattal szakítva több kis program együttműködése révén megoldani egy nagy feladatot. Akinek tetszik ez a filozófia, bátran tekintse meg DJB saját szoftvereit, elsősorban a djbdns-t és a qmailt, valamint pl. Gerrit Pape (a runit írója) socklog névre hallgató rendszernapló-megoldását.

Hivatkozások

[1] Daniel J. Bernstein honlapja. URL `http://cr.yp.to/`.

[2] A runit honlapja. URL `http://smarden.org/runit/`.

[3] A Unix/Linux kiszolgálók üzemeltetése nevű választható tantárgy wikis honlapja. URL `https://wifi.tmit.bme.hu/unixlinux/`.

Linux használata egy közintézményben

Kovács László

Kivonat

Az előadás fókuszában a Linux bevezetésével, elterjesztésével, használatával az elmúlt közel nyolc évben egy országos könyvtárban (Országos Pedagógiai Könyvtár és Múzeum) szerzett tapasztalatok állnak.

Az előadás négy fő pont köré csoportosítva foglalja össze a történteket:

1. Először is néhány szó az előzményekről: pénztelenségről, váratlan gépvásárlásról (30 gép, szoftver nélkül) és a megfelelő megoldásról: Linuxot minden gépre! Ezt egy házi tanfolyammal kellet megalapozni, ahol a Linuxhoz szoktatás mellett a hangsúly a problémamegoldó szemlélet kialakításán volt: alapfeladatokat (levelezés, Internet böngészés, szövegszerkesztés) egy adott eszközzel, de lehetőleg attól nem függve kell elsajátítani.

2. Ezután néhány technikai részlet következhet: telepítés, felügyelet, frissítés. Ezek kialakításánál a fő szempont az volt, hogy egyszerűen és általános eszközök (tar, gzip, ssh) segítségével elvégezhetők legyenek. Itt kap helyet a vékonykliensek üzemeltetési tapasztalatainak összefoglalása is.

3. A következő pont az elmúlt évek sikereinek összegzése: sok év zavartalan működés; az újabb beszerzéseknél már természetes a Linux telepítése; a szabad szoftver beköltözött a köztudatba: bizonyos feladatra csak azt tartják alkalmasnak (olvasói gépek), ill. szívesebben használják (Gimp stb.)

4. Végül szó esik a kudarcokról is: miután megfelelő források álltak rendelkezésre beszivárogtak a kereskedelmi szoftverek; ennek külső okai: pl. Akadémiai Select (szoftverpolitika), előre telepített szoftverek; belső okai: (személyi) szoftverfüggőség, kialakulásának okai: rosszul felépített oktatás (ECDL, tanfolyamok, iskolák).

Tartalomjegyzék

1. Bevezetés

A cikk rövid összefoglalást ad az elmúlt években egy könyvtárban[1] a szabad szoftverekkel kapcsolatban végbement történésekről. Az áttekintés sokféle elemből tevődik össze, elsősorban historikus elbeszélése az eseményeknek, de nem nélkülözi a szakmai, informatikai részt sem, végül pedig némi szociológiai, lélektani íz is vegyül a többi közé, amikor is a szoftverhasználat hátterét próbáljuk megérteni.

1.1. Miért fontos?

Mivel a tanulmányban egy közintézményről van szó, érdemes itt néhány mondatban elidőzni: a közintézmények közpénzből élnek, ezért nem mindegy, hogy azt mire használják fel. Mivel a közpénzeket nem közpazarolni kellene [8], hanem értelmes célokra, esetleg magyar szellemi projektekre, talán egy (több) magyar fejlesztésű Linux-disztribúció támogatására kellene fordítani. Ebben a tanulmányban azt igyekszem bemutatni, hogyan lehet néhány adódó lehetőséget, néhány egyszerű ötlettel kihasználni, és ezzel beúsztatni a szabad szoftvereket a köztudatba. Azonban általános receptet nem tudok adni, és azt is látni fogjuk, hogy egy ilyen apró kezdeményezés nem teljesedhet ki a külső ellenhatások miatt.

1.2. Miért szabad szoftvert?

A könyvtári Linux-használat bemutatása előtt érdemes kissé messzebbről indulni, feltenni a kérdést: ki miért használ szabad szoftvert, illetve miért áll át ezek használatára? Az erről szóló hírek ma már nem nevezhetők ritkaságnak. Számos intézmény, testület, szervezet, esetleg cég áll át kisebb-nagyobb mértékben szabad szoftverek használatára. Az egyéni lelkesedésen, elhivatottságon, elkötelezettségen túl ezek mögött a törekvések mögött alapvetően két okot látunk. Az egyik megközelítés szerint szabad szoftverek használatával költségcsökkentést, a rendelkezésre álló anyagi források jobb felhasználását akarják elérni. Tehát az átállás egyik lehetséges megközelítése gazdasági természetű (példa erre: [2, 15, 16]). A másik szemlélet az egy bizonyos vállalattól, országtól való függést, kiszolgáltatottságot igyekszik megszüntetni (erre különösen az ázsiai országok esetében láthatunk példákat [1, 19])[2]. Ebben az esetben a megalapozás politikai természetűnek nevezhető.

Hozzánk és az itt leírt projekthez az első áll közelebb (de nem tartom elvetendőnek a második gondolatot sem), azonban nem beszélhetünk egyértelműen költségcsökkentésről, hiszen nekünk nem volt mit elköltenünk. Tehát a mi vállalkozásunk esetében a költségeket a végletekig kellett csökkenteni: a semmiből, ingyen kellett kihozni valami jót.[3]

2. A könyvtári Linux projekt

2.1. Előzmények

Mint az előzőekből kiderült a projekt nem annyira a racionális megalapozáson, hanem inkább a szükség kiváltotta kényszeren alapszik, ezzel együtt törekedtünk arra, hogy az eredmény minél jobb legyen. A vállalkozás elindítója egy váratlan gépvásárlás volt[4], melynek során a könyvtár saját mértékeihez és addigi eszközparkjához mérve sok és jó minőségű számítógép-

[1] Országos Pedagógiai Könyvtár és Múzeum
[2] Mindkét esetre lehetne még sorolni további példákat is, itt csupán a tendenciák felvázolása, az illusztrálás volt a cél.
[3] Nem merek általánosítani, de attól tartok, nem egyedi esetről van szó.
[4] Csak a pontosság kedvéért írom ide, hogy ez 2000 elején történt. A Linux jelenléte ennél régebbi, egy szerveren és egy munkaállomáson ekkor már kb. két éve használtuk.

hez jutott hozzá.[5] Az 1. táblázat foglalja össze a vásárolt gépek főbb jellemzőit.

1. táblázat. A vásárolt számítógépek

	Processzor	Memória	Videó	HDD	Mennyiség
A	Celeron-333	64–128 MB	S3Trio3Dx2	3,2 GB	15
B	Celeron-333	64 MB	S3Trio3Dx2	–	10

Az alapvető eltérés a HDD hiánya a *B* összeállításnál és az *A* konfiguráció esetében az esetenkénti kétszeres memória méret. Így eltérő módon kellett őket felhasználni. Az *A* esetében nem volt probléma a Linux telepítése, leszámítva a videókártyát, amit akkor az X nem támogatott. Viszont a frame bufferes X-szerver használata megoldotta a problémát.

2.2. Disztribúciók, szoftverek

A kiválasztás szempontja alapvetően az volt, hogy a felhasználás fő területeihez megfelelő, jól működő, lehetőleg magyarul kommunikáló szoftverek álljanak rendelkezésre. A használati területek szokványosnak mondhatók: 1. elektronikus levelezés, 2. webböngészés[6], 3. irodai tevékenységek. Kiegészítő, egyéni jelleggel előfordulnak más feladatok is, például képszerkesztés. A szempontok között egyre fontosabbá vált a magyar nyelv jó támogatása, így egyenes út vezetett egy magyar fejlesztésű disztribúcióhoz. Az eddig használt fontosabb disztribúciókat foglalja össze a 2. táblázat.

2. táblázat. A használt fontosabb disztribúciók

Beszerzés éve	Disztribúció	Felhasználás célja
1999	Red Hat 6.0	szerverek és asztali gépek
2001	Mandrake 8.1	főként vékonykliensek
2003	UHU-Linux 1.0	asztali gépek
2004	UHU-Linux 1.1	asztali gépek
2006	UHU-Linux 2.0	asztali gépek

Látható, hogy nem túl sűrűn, s csak indokolt esetben újítunk. A legfontosabb indokok: 1. adott feladatot nem lehet megoldani (vagy nem megfelelő szinten), 2. működést, használhatóságot befolyásoló hibák merülnek fel (és ezek már megoldódtak az újabb kiadásban), 3. túlságosan elavult. Részben meghatározza a váltást az is, hogy van-e mire váltani; így a disztribúciók megjelenési frekvenciája, gyakorisága is befolyásolja az újításokat.

2.3. Felkészítés, oktatás

A Linux bevezetését nem lehetett volna véghezvinni egy alapvető ismereteket közvetítő házi tanfolyam lebonyolítása nélkül. Ennek a tanfolyamnak először is meg kellett ismertetni a felhasználókkal a grafikus felület használatának sajátosságait (menük, egér használat), és hozzá kellett szoktatni ehhez őket. A korábban megszokotthoz képest (MS-DOS) ez jelentős eltérést jelentett.

Az ismerkedésen túl a tanfolyam két alapvető témát ölelt fel: 1. internet-használat: e-mail és böngészés, 2. irodai programok: *StarOffice* és később *OpenOffice.org* . A grafikus környezet ismeretlenségén túl további nehézséget jelentett, hogy általában nem voltak előismeretek, nem lehetett mihez kapcsolni az elmondottakat. Ez részben előnyt is jelentett, mert akik ko-

[5] Ekkor az átlagos gép 386-os volt MS-DOS-szal a fedélzetén, és még ebből sem jutott mindenkinek.
[6] A munkavégzésnek és az olvasói tájékozódásnak is ez a fő eszköze, mivel a könyvtári rendszer webalapú.

rábban nem használtak Windowst, azoknál a rossz megszokások sem rögzülhettek, és nem kérdezhették, hogy ez vagy az miért így működik.

Természetesen csak korlátozott célokat lehetett kitűzni: alapvető műveletek, funkciók elsajátítását, a számítógép-használattal szembeni idegenkedés csökkentését (egér használata[7]), problémamegoldó szemlélet kialakítását. Ez utóbbi azt jelenti, hogy a tanfolyam, csúnya szóval, megpróbált platformfüggetlen lenni. A Linuxot és az itt elérhető programokat csak eszköznek használva, egy adott feladat elvégzéséhez szükséges gondolkodást, műveleteket igyekezett továbbadni. Az elektronikus levelezést tekintve ez azt jelenti, hogy egy új levél írásához szükség van egy üres területre, ahová be lehet írni a címet, a tárgyat és magát az üzenetet. A levél íráshoz először ezt kell elővarázsolni és csak utána jöhet a levél tényleges megírása. Ezt először végig gondolva a továbbiakban meg kell keresni, hogy az adott eszközzel ezeket a lépéseket miként lehet elvégezni. Mindegy, hogy a programnak mi a neve, grafikus vagy karakteres felületen működik-e, ugyanazt kívánjuk elvégezni vele. Ennek a tevékenységnek kell megragadni a lényegét, és nem azt kell megjegyezni, hogy az *X* menüben az *Y* pontot kell választani. Sajnos ez az üzenet nem igazán ért célhoz. Valószínűleg túl sok dolgot kellett egyszerre elsajátítani, és ez elterelte erről a figyelmet.

A későbbiekben nem volt szükség átfogó, általános tanfolyamra, így a képzés alapelve nem, de módszere változott: igény szerint egyénileg történt, főként olyanoknak, akik kereskedelmi termékekkel már elég jól elboldogultak, de a szabad szoftvereket, Linuxot még meg kellett szokni (ilyen, kissé fájdalmas, nem is mindig sikeres átállás az MS Office-ról OpenOffice.org-ra történő váltás). Szélesebb kört érintő esetben volt néhány bemutató az ismeretbővítéshez, új eszköz, program használatba vételéhez.

3. Néhány technikai részlet

A projekt ismertetéséből természetesen nem maradhat ki a megvalósítás során alkalmazott gyakorlati megoldások összefoglalása sem. Mindegyik esetében a fő szempont az egyszerűség volt: egyrészt azért, hogy ne kelljen sokat dolgozni, másrészt (és ez fontosabb), hogy az adott feladatot könnyen és gyorsan el lehessen végezni.

3.1. Telepítés

A konfigurációkat összegző táblázatból kiderül, hogy viszonylag sok géphez jutott hozzá a könyvtár. Azt a számot mindenképpen túllépte ez a mennyiség, amivel még külön-külön lehetne foglalkozni. Már ennyi gépre sem éri meg egyesével rendszert telepíteni. Ezért a rendelkezésre álló eszközöket és a lehetőségeket figyelembe véve olyan módszert kellett kitalálni, ami meggyorsítja ezt a műveletet. A megoldást az egységes beszerzés hordozta magában: homogén gépparkra kellett telepíteni. Így elegendő volt egyetlen gépen elvégezni a telepítést és a beállítást, a többi gépre ezt a telepített rendszert lehetett átvinni. Az „átvitel” kezdetben a `dd` paranccsal[8], majd később a `partimage` [14] programmal történt.[9] Ezek után már csak egy ellenőrzésre (tényleg elindul-e a rendszer) és a felhasználó(k) beállítására volt szükség. Ezáltal a telepítést viszonylag gyorsan el lehetett végezni és az eredmény megnyugtatóan egységes lett.

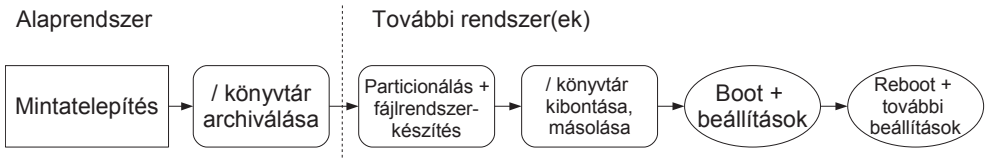
Későbbi beszerzésekkel a kezdetben egységes géppark egyre változatosabbá vált. A telepítéshez új megoldást kellett találni; lehetőleg egyszerűt és könnyen használhatót. A szüksé-

[7] Ma már elképzelhetetlennek tűnik, akkor ez jelentős probléma volt, ami ráadásul éppen a lényegről vonta el a figyelmet.

[8] Mivel más okból egyébként is meg kellett a gépeket bontani, ez a művelet a merevlemezek kiemelésével történt.

[9] Több program is létezik, amely partíció másolást, manipulálást tesz lehetővé. Ezek egyike a fent említett `partimage`. A talán legrégebbi ilyen szabad szoftver a GNU `parted` [7]. Kereskedelmi termék szabad megfelelője a `G4L`, Ghost for Linux [6].

ges eszközök a telepítő adathordozókon túl egy Live CD és azon belül a mindenhol rendelkezésre álló tar és gzip programok. A műveletek sorrendjét mutatja az 1. ábra.



1. ábra. A telepítés gyorsítása mintarendszer készítésével

A műveleteket két csoportra lehet osztani, ennek megfelelően az ábrán szaggatott vonal választja el a mintául szolgáló gépen elvégzendőket a további, a cél gépeken végrehajtandó teendőktől. A keretek alakja arra utal, hogy a művelet végzésre használt rendszert honnan töltöttük be. Három lehetőség van a telepítés fokozatai alapján: 1. a rendszer telepítő CD (DVD)-ről indul (téglalap), 2. Live CD-ről indított rendszer (lekerekített téglalap), 3. a telepített rendszer indul el a végleges helyén (ellipszis).

Látható, hogy a műveletsor itt is csupán néhány egyszerű lépésből áll. Az alapja ennek is az egy gépen elvégzett, jól átgondolt telepítés. Ennek során egyrészt ki kell választani minden szükséges programot, másrészt figyelni kell az erőforrás-korlátokra is: a legkisebb lehetőségekkel rendelkező gépen is el kell férnie és működnie kell a telepítésnek. Ehhez a megoldáshoz a home könyvtáron kívül mindent egy partícióra telepítünk, bár ez ellentmond a vonatkozó ajánlásoknak [10] és nézeteknek, a tapasztalat azt mutatja, hogy nincs gond az ilyen telepítéssel. Sőt, ez a megoldás előnyösnek is bizonyult: ugyanis azokon a gépeken, ahol erre van szükség, a home könyvtár tartalma minden induláskor újraíródik egy archív fájlból, ami a jó állapotot tartalmazza. Így nem okoz gondot ha a felhasználó valamit átállít, megváltoztat, egy újratöltés mindent megold.

A másik fontos eleme ennek a telepítési megoldásnak a mintatelepítés archiválása, majd az archívum kibontása a célszámítógépen. Ezt két egyszerű paranccsal lehet elvégezni. Az összecsomagolásra a következő szolgál:

```
$ ls telepítés_helye > /tmp/file.lista;
$ tar zcvf - -C telepites_helye -T /tmp/file.lista --exclude lost+found |
  ssh -l root szerver "cat > /valahol/telepito/gyoker_konyvtar.tar.gz"
```

Látható, hogy szokványos eszközök összekapcsolásával és néhány paraméterrel megoldható a dolog. A telepítésre ugyanezen programok szolgálnak más paraméterekkel:

```
$ ssh -l root szerver "cat /valahol/telepito/gyoker_konyvtar.tar.gz" |
  tar zxvf - -C /telepites/helye/
```

Az eddigiekből kiderül, hogy semmiféle automatikus frissítési, vagy telepítési eljárást nem alkalmazunk[10]. Az intézmény mérete és a gépek mennyisége lehetővé teszi, hogy minden géphez odamenjünk, elvégezzük a fenti trükkökkel meggyorsított telepítést, és az egyéni igények, valamit a konfiguráció szükségletei szerint méretre igazítsuk a rendszert.

3.2. Üzemeltetés, felügyelet

A telepítésnél említett körülmények (nem túl nagy cég, nem túl sok gép) ellenére a felügyeletet úgy célszerű megoldani, hogy ne kelljen minden hiba, probléma miatt az adott géphez menni. A gondok egy része megoldható távolról is (pl. fölösleges fájlok törlése a felhasználó

[10] Azon érdemes lenne elgondolkodni, hogy a telepített programok új verzióit a helyi hálózaton lévő kiszolgálóról például *apt-get* segítségével automatikusan frissítsék a gépek.

könyvtárából), de többségüknél mégis a helyszínen kell elvégezni a javítást. Léteznek kifejezetten távadminisztrációt, távbeállítást lehetővé tevő szoftverek[11], azonban ezek számunkra túlságosan bonyolultak, nagyméretűek. Minden gondunkat megoldja az ssh használata.

Az ssh-n kívül egyetlen kis méretű, házi fejlesztésű TCL/Tk programocskát használtunk, amit a távfelügyelet kategóriába lehet sorolni. Ez a kis program üzenetküldésre szolgált, a felhasználó egy gombra kattintással nyugtázhatta az üzenet elolvasását.

3.3. Vékonykliensek

Vékonykliensen *(thin client)* minimális kiépítéssel rendelkező számítógépet értünk [17]. Ezek a gépek a működésükhöz szükséges erőforrások többségét egy kiszolgáló gépen veszik igénybe. Tipikus példák erre a grafikus felhasználói felületet biztosító X-terminálok, amelyek esetében az alkalmazások a kiszolgáló gépen futnak.[*]

A konfigurációkat összegző táblázatból kiderül, hogy a Linux bevezetését elindító gépvásárlás során, főként takarékossági okokból a gépeknek egy jelentős részét merevlemez (valamint floppy- és CD-ROM meghajtó) nélkül szereztük be (*B* konfiguráció). Ezek tehát a fenti meghatározásnak is megfelelő, minimális kiépítésű gépek voltak. Az ilyen gépeknek már hagyománya volt a könyvtárban, hiszen egészen addig a számítógépek többsége ilyen volt[12]. Kézenfekvő volt, hogy ezeket a gépeket az eddig megszokott módon a Novell szerver segítségével MS-DOS-t boot-olva üzemeljük be. Azonban sokkal hatékonyabban is ki lehetne őket használni, amire szintén a Linux nyújtott megoldást.

Merevlemez nélküli rendszer betöltéshez léteznek kész megoldások [5], létezik hozzá leírás is [4], de mi ezeket nem tudtuk használni. A fő szervező és meghatározó tényező a fentebb már emlegetett nagyarányú pénztelenség volt. Olyan megoldást kellett kitalálni, amit a meglévő eszközeinkkel elő tudunk állítani, kiegészítve, kibővítve őket a fellelhető szoftveres megoldásokkal.

A rendszer betöltési folyamat vázlatosan a következő volt: Novellről DOS-t bootoltak, majd loadlin-nel Linuxot, a további rendszer betöltés és minden lemez szükséglet biztosítása NFS segítségével [13] történt.[13] A 2. ábrán láthatjuk a folyamat lépéseit[14].

Felmerülhet a kérdés, miért így oldottuk meg: erre a magyarázat – a forráshiányon túl – az, hogy ezzel a megoldással változatlanul lehetett DOS-t betölteni, ha éppen arra volt szükség. Az ábrából látszik, hogy egy kis módosítástól eltekintve a DOS változatlan formában indul el. Ez a kis módosítás a Linux betöltéséhez kellett. Kísérletezések eredményeként alakult ki a fenti megoldás, aminek fontos eleme, hogy a Novell eléréshez szükséges drivereket el kellett távolítani, utána indulhatott csak a Linux a loadlin segítségével zavartalanul. Mindez egy RAM-disk segítségével történt.

Ezeket a gépeket könnyű volt karbantartani (központi menedzselhetőség). Egyszerű és gyors volt a frissítésük. Hátrányuk az volt, hogy működésüket nagyban befolyásolta a hálózat terheltsége. Ez jelentette a végüket: sajnos a hálózatunk elavultsága, megbízhatatlansága miatt meg kellett szüntetni őket.

---

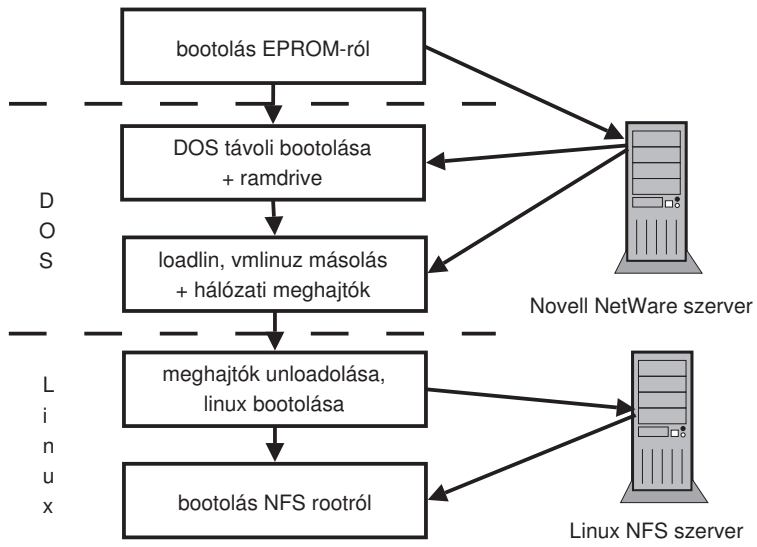
[11] Kettőt mindenképpen érdemes megemlíteni: Linuxconf [11] és a Webmin [18]. Lásd még kiadványunk Cfengine-ről szóló cikkét.

[*] Lásd még kiadványunk vékonykliensekről szóló cikkét – *a szerk.*

[12] Mint az előzőekben utaltam rá, az átlagos számítógép Novellről DOS-t bootoló 386 gép volt.

[13] A rendelkezésre álló EPROM-ok Novell RPL (Remote Program Load) protokoll használatát tették lehetővé, míg a fent említett megoldás (EtherBoot) BOOTP, DHCP illetve TFTP protokollt alkalmaznak. De hasonlóan működik a NetBoot [12] is.

[14] Csak a főbb lépéseket foglalja össze az ábra, a pontos részleteket nem tartalmazza, itt nincs is szükség kitérni ezekre.



2. ábra. Merevlemez nélküli kliensek rendszerindítása hálózatról

4. Tapasztalatok

4.1. Sikerek

A sikerek közül ki kell emelni azt az időszakot, amikor a könyvtár gépeinek nagy részén a Linux biztosított kellemesen használható grafikus felhasználói felületet. Ez magában foglalja az előzőekben említett kezdeti gépvásárlás szinte valamennyi gépét, és a később beszerzett gépeket is[15]. Ez az időszak sokéves zavartalan működést jelentett. Ez alatt a Linuxot megszokták, mint lehetséges megoldás, alternatíva beépült a köztudatba.

Ezeken a gépeken, mint fentebb már említettem, KDE grafikus felület működik, böngészéshez Mozilla (helyenként Firefox), irodai munkákhoz OpenOffice.org (korábban StarOffice), levelezéshez Kmail, képszerkesztéshez GIMP áll rendelkezésre. Sikernek mondható, hogy ezeknek a programoknak egy része már Windows alatt is használatban van, különösen ha a feladat is indokolja ezt (pl. PDF-készítés).

A Linux sikerét, elismertségét mutatja az is, hogy például az olvasói gépek (a könyvtár látogatók által használt gépek) egy részének esetében kifejezett kérés volt, hogy Linux legyen rajtuk, különösen a megbízhatóság és a biztonság miatt. Ezeken a gépeken alkalmazzuk azt a fentebb már említett megoldást, hogy a home könyvtár minden induláskor újraíródik egy tar fájlból, így mindig visszaáll az alapállapot[16].

4.2. Kudarcok

Legfőbb kudarcként azt kell megemlíteni, hogy a szabad szoftvereket nem igazán szerették meg, s azok nem váltak elfogadottá.

A Linux és a szabad szoftverek sorsát, megítélését nagyban meghatározta az a kiváltó ok, ami teret nyitott bevezetésüknek. Mivel nem alapos megfontoláson és elhatározáson alapult

[15] Ebben a tanulmányban csak a munkaállomásokról esik szó. A szerverekkel kapcsolatban egyetértés van, itt 1998 óta használunk kizárólag Linuxot.

[16] Ennek előzménye az volt, hogy az olvasók az általuk használt gépeket rendszeresen elállították. Erre volt válasz a Linux a fenti megoldással. Itt csak a legszükségesebb programokat hagytuk elérhetően (természetesen a saját munkánkat sem akarjuk megnehezíteni: root-ként rendelkezésre állnak a megfelelő szoftverek). Így a Linuxot kevésbé tudják elrontani, de általában nem is értenek hozzá.

használatuk, hanem a pénztelenség vezetett ide, ennek enyhülésével ezeknek a szoftvereknek az elfogadottsága is romlott.

Miután rendelkezésre álltak megfelelő források, szép lassan beszivárogtak a gépekre a kereskedelmi szoftverek. Tovább erősítették ezt a tendenciát az újabban beszerzett gépek előre telepített szoftverei. Mostanára már a szabad szoftverek vannak kisebbségben. A mai arányok: 41 Linux, 20 Windows 98, 18 Windows XP és 6 Windows NT rendszert futtató kliens van.

Ez a folyamat külső és belső okokra vezethető vissza: külső okokon az intézményen kívülálló lehetőségeket, hatásokat, belső okokon a felhasználók hozzáállását, adottságait értjük.

A külső okoknál az előbb említett előre telepített szoftvereken túl meg kell említeni például az Akadémiai Select rendszerét. Ezen belül például Hungarnet Egyesület tagintézményei kedvezményesen és egyszerűen juthatnak kereskedelmi szoftverekhez. Az igazán jó az lenne, ha ezeken az egyébként már bejáratott csatornákon szabad szoftverekhez is hozzá lehetne jutni.

A külső, intézményes okokon túl a nem megfelelő elfogadottságot belső, a felhasználók egyéniségében, oktatásában, hozzáállásában rejlő okok is kiváltották. Legfőbb indoka az elfordulásnak: otthon nem ez van (természetesen az előre telepített szoftverrel vásárolható gépekkel, vagy a baráti szívességként végzett telepítéssel nem Linux kerül az otthonokba). Hasonló kifogás: nem ezt tanultam (ECDL, könyvtár-informatikai képzés). Ez már kicsit problémásabb dolog, intézményesült gondokra mutat rá. Általában azokon a helyeken, ahol informatikai tudást lehet szerezni, a tudással együtt (rosszabb esetben ahelyett) szoftverfüggőséget[17] alakítanak ki. A rosszul felépített oktatás, tematika miatt a felhasználó nem tud elvonatkoztatni az adott, a tanfolyamon alkalmazott szoftvertől, így a feladatot csak azzal tudja megoldani. Sajnos az oktatás bármely szintjén kimutathatók ennek jelei, a tanfolyamoktól kezdve, a közoktatás informatikai részén át, a felsőoktatásig[18]. Bár nem sok intézményt ismerek, de véleményem szerint a felsőfokú informatikai végzettséget adó képzések esetében sem sokkal jobb a helyzet. Ez ellen a függőség ellen nagyon nehéz bármit is tenni: hiába a jobb minőségű esetleg többet tudó szoftver, ha egy kicsit is másként kell használni, olyan fokú ellenérzést vált, ki, hogy a legnagyobb igyekezettel sem lehet elérni a szabad szoftverek elfogadását.

Nem szabad elfeledkezni arról sem, hogy az elfogadottságot gyengíthetik apró bosszúságok is. Valami nem működik, vagy nem úgy működik, ahogy az megszokott, nem ott van valami egy menüben, ahol elvárható lenne. Ezek az apróságok összeadódva azt az érzést keltik, hogy a Linux egy macerás, nehezen használható rendszer. Három dolgot lehet megemlíteni az ilyen jellegű gondok orvoslására:

Először is növeli az elfogadottságot, sőt az elismertséget, ha egy szabad szoftver jól működik, sok funkcióval rendelkezik. Példa erre intézményünknél a GIMP[19], amit nem csak Linux alatt használnak és tökéletesen elegendőnek bizonyul a felmerülő feladatok megoldására.

Másodszor segíthet az is, ha egy alkalmazás nagyon hasonlít, úgy néz ki, úgy használható, mint kereskedelmi megfelelője. Ezt megértve kezdtem el becsülni a Windows kinézetre fabrikált ablakkezelőket (pl. KDE Windows témák). Első pillantásra ezeknek túl sok hasznuk nincs: ha nem ugyanazt csinálja, miért akar ugyanúgy kinézni? Azért, hogy az idegenkedést, bizonytalanságot csökkentse: legalább első ránézésre megegyeznek, és ez egyfajta kapaszkodót ad.

---

[17] Szoftverfüggőségről beszélhetünk szervezeti és egyéni értelemben. Egy intézmény esetében a szoftverfüggőség azt jelenti, hogy az adott szervezet tevékenysége oly mértékben kapcsolódik egy szoftverhez, hogy nélküle működésképtelen [9]. A személyes függőségre ezt a meghatározást úgy fordíthatjuk át, hogy az egyén minden informatikai tudása, tapasztalata egy adott szoftverhez kötődik, csak ezzel képes feladatai elvégzésére.

[18] Vannak persze kivételek és örvendetes kezdeményezések, mint a Linuxos ECDL vizsga, vagy az UHU-Linux érettségi változata. Sok ilyen kellene, hogy legalább egészséges egyensúly legyen.

[19] Ne feledkezzünk meg a Firefox-ról és a Thunderbird-ről sem.

Végül meg kell említeni a szoftverek rendszeres frissítését és a felhasználók jó támogatását is. Mivel a szabad szoftvereket általában elég jó tempóban fejlesztik, így egy zavaró hiba, vagy hiányosság az újabb verziókból hamar eltűnik. Ezzel meg is lehet mutatni ennek a fejlesztési modellnek az erősségét, ami vonzóbbá teheti őket a zárt, kereskedelmi termékekkel szemben. Sajnos úgy tűnik ezen a területen vannak mulasztásaink a könyvtári Linux projektben. Mint fentebb látható, elég ritkán szánjuk el magunkat frissítésre, emiatt egy-egy probléma sokáig megmarad, elbizonytalanítva, bosszantva a felhasználókat. Ez jelentős részben hozzájárul a kudarcokhoz.

Az eddigiektől eltérő tényező, de nem kevés problémát okoz az olyan szoftverek alkalmazásának szükségessége, amelyek csak Windows alatt léteznek (ilyen pl. esetünkben az Ariel nevű könyvtárközi kölcsönzést lebonyolító program, az MNB[20] CD-kezelő szoftvere). Megoldást jelenthet a Wine alkalmazása, de bármennyire is sokat fejlődött, bizonyos funkciók így sem kelthetők életre vele. Marad tehát a Windows-vásárlás és telepítés.[*] Ezek után kézenfekvő, hogy az adott gépen más feladatokra is a Windows-t használják. Természetesen nem példa nélküli, hogy csak az adott szoftver kedvéért indítják el a Windows-t, de ehhez kell némi elszántság. Sajnos összességében ez is az elfogadottság ellen hat és megalapozza a kudarcokat.

4.3. Összegzés, tanulságok

Akkor van igazán haszna ennek a tanulmánynak, ha ebből a projektből összeszedjük a mások számára is használható tanulságokat.

Az első tanulság, hogy a Linux bevezetéséhez egy kis intézménynél nagyon kevés dologra van szükség: némi lelkesedésre, kedvező körülményekre és (nem anyagi természetű) támogatásra. A kivitelezéshez egy átlagos Linux-disztribúcióban minden rendelkezésre áll, a feladatok nagy részét pedig meg lehet oldani néhány mindenhol elérhető paranccsal. Világossá válhatott az is, hogy a kedvező körülmények, lehetőségek kihasználása megfelelő elhatározás nélkül hosszú távon nem vezet egyértelmű sikerekhez. A körülmények változása kihat a vállalkozás eredményeire is.

A következő tanulság az, hogy külső és belső hatások egyaránt a sikerek ellen hatnak. Ezeken úgy lehet úrrá lenni, ha felmutatjuk a szabad szoftverek vitathatatlan előnyeit: megbízhatóság, gyors fejlesztés, frissítés.

További tanulság, hogy szükség van intézményi változásokra is. Vannak olyan országok, nem is túl messze tőlünk (pl. a szomszédban Horvátország), akik ezt egészen komolyan gondolják [3]. Nálunk is foglalkoztak a gondolattal [9], de úgy tűnik ebben a stádiumban megrekedtek a dolgok. Amíg ebben nincs elhatározás és cselekvés, marad a szélmalomharc.

Utolsó tanulságként meg kell állapítani, hogy egy ilyen kis intézmény nem tudja magát függetleníteni a külső hatásoktól, trendektől. Viszont példa lehet azok számára, akik megfelelő szinten határozhatnak költségekről, támogatásokról, hogy „nem csak egy járható út van”[21].

---

[20] Magyar Nemzeti Bibliográfia, kiadja Országos Széchényi Könyvtár, Arcanum Adatbázis Kft.

[*] Ide tartozik a Windows virtualizált, Linuxon belüli futtatása pl. QEMU vagy VMWare segítségével – *a szerk.*

[21] A Linux.hu portál címfelirata és a Perl programozási nyelv mottója.

Hivatkozások

[1] Berta Sándor: Csak Linux-kompatibilis PC-ket vásárol Tajvan, 2006. június 8. URL http://www.sg.hu/cikkek/45135/csak_linux_kompatibilis_pc_ket_vasarol_tajvan.

[2] Berta Sándor: Megkezdi a Linuxra való átváltást München, 2006. szeptember 22. URL http://www.sg.hu/cikkek/47405.

[3] Croatian open source software policy, 2006. URL http://www.szszi.hu/wiki/20060825_Croatian_Open_Source_Software_Policy.

[4] Diskless nodes HOW-TO document for Linux. URL http://www.faqs.org/docs/Linux-HOWTO/Diskless-HOWTO.html.

[5] EtherBoot project. URL http://www.etherboot.org.

[6] G4L, Ghost for Linux. URL http://sourceforge.net/projects/g4l.

[7] GNU parted. URL http://www.gnu.org/software/parted/index.shtml.

[8] Közpazarlás, szabad software/Linux és kormányzat. URL http://www.magyarorszag.hu/parbeszed/tudomany/infotars/topic.html?fid=522.

[9] A Linux operációs rendszer kormányzati felhasználásának lehetőségei, nemzetközi tapasztalatok elemzése, 2002. URL http://www.fsf.hu/oss_gov_hun.html.

[10] Linux Partition HOWTO. URL http://tldp.org/HOWTO/Partition/.

[11] Linuxconf. URL http://www.solucorp.qc.ca/linuxconf/.

[12] NetBoot. URL http://netboot.sourceforge.net/.

[13] NFS-Root mini-HOWTO. URL http://www.faqs.org/docs/Linux-mini/NFS-Root.html.

[14] Partimage. URL http://www.partimage.org/.

[15] Tóth Kristóf: Afrikában is jelentős a kormányzati érdeklődés a Linux iránt, 2003. URL http://www.sg.hu/cikkek/28239/afrikaban_is_jelentos_a_kormanyzati_erdeklodes_a_linux_irant.

[16] Tóth Kristóf: Austinban a közeljövőben Linuxra cserélik a Windowst, 2003. URL http://www.sg.hu/cikkek/30375/austinban_a_kozeljovoben_linuxra_cserelik_a_windowst.

[17] A „vékony kliens” címszó. URL http://hu.wikipedia.org/wiki/Kliens#V.C3.A9kony_kliens.

[18] WebMin. URL http://www.webmin.com/.

[19] Zádori István: Japán, Kína és Dél-Korea Windows alternatívát készül kifejleszteni, 2003. szeptember 1. URL http://www.sg.hu/cikkek/28762.

A sima védelemtől a Sarbanes–Oxley megfelelőségig. Milyen kihívásokkal kell megküzdeni a Linux alapú antivírus-védelemben?

Máriás Zoltán

Tőzsér és Máriás Szoftver Iroda Kft.

Kivonat

A Linux mára már nemcsak a maroknyi lelkes rajongó operációs rendszere, hanem a nagy cégek és szervezetek is komoly alternatívaként tekintenek rá. Ahhoz, hogy ezek az érdeklődők a linuxos szerverekre a levelezés megoldása, a tűzfal- és az adatbázis-kezelési feladatok ellátása mellett más komolyabb feladatokat is rá merjenek bízni, a Linuxot megfelelő védelemmel kell ellátni. A védelem gyakran törvényben szabályozott, például a pénzügyi adatokat kezelő számítógépekre az USA-ban a Sarbanes–Oxley megfelelőséget (SOX) elváró törvény vonatkozik. Ma már hosszútávon tarthatatlan az a hozzáállás, hogy a nem Windows-alapú operációs rendszereket nem, vagy alig kell vírusvédelmi és egyéb IT biztonságtechnikai szolgáltatásokkal felszerelni. Jelenleg ugyan valóban kisebb az érdeklődés a rosszindulatú kódok készítői részéről a Linux iránt, de ez korántsem jelenti azt, hogy a Linux immúnis lenne, vagy a crackerek képtelenek ilyen jellegű kód megírására.

A cikk vázolja a Linuxra leselkedő fenyegetéseket, betekintést nyújt a törvényi szabályozásba, majd ismerteti, mit kell tudnia a Linux rendszereket védő integrált vírusvédelmi megoldásoknak.

Tartalomjegyzék

1. Fenyegetések Linux alatt

Még a nagyfokú biztonsággal – és kevésbé célpontként kezelt – operációs rendszerekre is készülnek és készülni fognak kártevők. A legnépszerűbb Linux-változatokra készült szoftvertermékek újra és újra, és egyre komolyabb javításokra szorulnak. Elég itt az utóbbi időben sorozatos sérülékenységekről rendszeresen hírt adó Secunia riasztásait vizsgálni, amelyek a *Mozilla Thunderbird* [16], a *Firefox* [15] termékekről szólnak. E sérülékenységek nagy részének minősítése „erősen kritikus" [3] *(highly critical).* De nem tökéletes többé az *Opera*, a *ClamAV* [13, 14], az *OpenSSH* vagy az *OpenSSL* sem. A megfeszített ütemben zajló fejlesztések és a valódi, alapos tesztelési mélységek hiánya egyaránt a stabilitást kezdik ki. És ha folyamatosan nem frissítjük a Linux operációs rendszer kernelét, valamint az azon futó termékeket, könnyen válhatunk magunk is célponttá.

Ugyan a Linux alapú operációs rendszerekre eddig megjelent vírusok az összes rosszindulatú programnak mindössze 0,09 %-át teszik ki (a SophosLabs statisztikája [1] szerint), védekezni – a fentiek miatt – mégis kell, és szükséges. Gondoljunk csak a *Troj/Lindoor-B* [19] névre keresztelt hátsó bejáratot nyitó trójai programra, amely a rosszindulatú felhasználónak olyan távoli héjat tud lehetővé tenni, amellyel az átveszi a rendszer feletti irányítást. De a 2005 decemberében megjelent *Linux/Mare-A* féreg [10] is ügyesen terjed a kiaknázható *PHP*-szkripteken keresztül.

Az eddig megismert körülbelül 120 000 vírus közül mindössze 100 célozza meg a Linux vagy a UNIX operációs rendszert, de az arány romlására lehet számítani. Főleg azért, mert – többek között a Gartner kutatása és felmérése [8] alapján – a szervereken futtatott operációs rendszerek arányát tekintve a Linux növekszik a leggyorsabban. Ezt alátámasztja az a tény is, hogy az IBM és a Novell után egy újabb nagyágyú jelent meg [11, 17] a Linux-színtéren: az Oracle, amely ugyanolyan professzionális, vállalati szintű támogatást fog nyújtani a Linuxhoz, mint adatbázis-kezelőjéhez, köztes szoftvereihez és alkalmazásaihoz.

Ha a Linux nem is fertőződhet meg, de fertőzni azért tud. Ennek eklatáns példája a *W32 Nyxem* vírus D variánsa, amely kiválóan tud terjedni a Samba fájlmegosztásokon keresztül is. Vagy elég ugyanezt a vírust egyetlen levélben továbbítani egyetlen Windows-alapú számítógépre, hogy a fertőzés megtörténjen.

2. A törvényhozó testületek elvárásai

A jogalkotók, akik a nemzetközi vállalatok és a kormányzati szervezetek IT biztonságára vonatkozó törvényeket hozzák, nem tehetnek, és nem is tesznek különbséget operációs rendszer platformok között: ugyanazt a védelmi szintet várják el a Linuxtól is, mint más rendszerektől. Ezért a Linux alapú rendszerek és alkalmazások fejlesztőire és üzemeltetőire folyamatosan növekszik a nyomás a jogalkotók részéről.

Az olyan törvényi rendelkezések értelmében, mint az az Egyesült Államokban a SOX *(Sarbanes–Oxley)* [12, 5, 6, 18] és a HIPAA *(Health Insurance Portability and Accountability Act)* vagy az Egyesült Királyságban az Adatvédelmi Törvény *(Data Protection Act),* tudni kell igazolni, hogy például egy Linux gép nem képezhet semmifajta rést a szervezet pajzsán. Ezek – a globalizáció miatt – nemzetközi szinten is érvényes rendelkezések a magánszemélyek jogait hivatottak biztosítani, és további feladatokat rónak a rendszergazdára. A SOX például előírja, hogy minden olyan számítógépet védeni kell, amelyen pénzügyi adatot tárolnak vagy azon pénzügyi adat halad át. A HIPAA ugyanilyeneket ír elő a magánszemélyek egészségügyi adatainak esetében. Általánosan mondható, hogy ezek a törvények a következő kikötéseket tartalmazzák:

– Információbiztonság: minden olyan tényezőt ki kell zárni, amely az adatok megváltoztathatóságának vagy megsemmisülésének kísérletét lehetővé teszi.

– Az ellenőrzés igazolása: a szervezetnek képesnek kell lennie igazolnia azt, hogy a felügyelet és az ellenőrzés folyamatos. A központi naplók, az auditnyomok és a jelentések nélkülözhetetlenné váltak.

3. Víruskeresők szoftverfrissítése

A már említett Secunia IT-biztonsági riasztószolgálat több víruskeresőben (szabad és fizetős szoftverben egyaránt) talált magas fokú, kritikus hibát az elmúlt évben. Ami érdekes, hogy a sérülékenység feltárásától (származzon az magától a fejlesztőtől vagy független forrásból) és nyilvánosságra hozatalától számítva napok, sőt néha hetek telnek el addig, amíg a Linux-disztribúció is lép egyet. Ezek után viszont további napok vagy hetek telnek el addig, amíg a javítókészletek letöltésére és alkalmazására a rendszergazdák vállalkoznak. Ezt az időszakot tudják kihasználni a rosszindulatú kódok a terjedésre, a rendszer feletti irányítás átvételére, a puffer-túlcsordulások előidézésére. Például a Secunia az utóbbi fél évben négyszer (április 6-án, május 1-jén, augusztus 7-én és most napokkal ezelőtt, október 16-án) jelzett magas fokú, kritikus hibát a *ClamAV* 0.x-es változatában. Ezek közül az utolsó hibának a javítására a Secunia szerint Debian-on 7 napot, Gentoo-n pedig 8 napot kellett várni. Az eltelt egy hét más ClamAV-hiábák és más disztribúciók (pl. Gentoo, Mandriva, Trustix és SUSE) esetén is tipikus időköznek mondható.

A modern Linux-disztribúciók biztosítanak internetes szoftverfrissítési lehetőséget, és külön csapat foglalkozik a javítások figyelésével, a javított csomag leghamarabbi elkészítésével és publikálásával. Arról azonban általában a rendszergazdának kell külön gondoskodnia, hogy ezek a frissítések felkerüljenek a gépre (ez Linux alatt könnyen automatizálható). Általában csak a disztribúció részét képező csomagohoz tölthetők le kényelmesen frissítések, így a kézileg telepített kereskedelmi szoftvereknél a rendszergazdának egyenként kell foglalkoznia a rendszeres szoftverfrissítéssel (általában szintén könnyen automatizálható), ha a szoftvernek nincs önfrissítő funkciója.

Kereskedelmi szoftverek esetén további gondot jelenthet, hogy a licenc megújítását időben kezdeményezni kell, ha a vírusvédelemben nem engedhető meg kimaradás. A probléma fokozódik, ha a szervezetnél a licenceket központilag szerzik be: egyes közintézményekben ez heteket is igénybe vehet, így a rendszer hetekig védtelen maradhat, ha vírusvédelmét kizárólag egyetlen kereskedelmi termékre bízza a rendszergazda.

4. A vírusvédelemtől elvárt funkciók

Szinte minden jelentős vírusvédelmi cég rendelkezik linuxos megoldással. E megoldások között nagy eltérések tapasztalhatók.

Az ideális Linux-alapú vírusvédelmi megoldásnak a következőket kell tudnia [2]:

– A kernel újrafordításakor a már letöltött vírusvédelmi motor automatikusan, menet közben újrafordításra kerül (többek között a Talpa kernelmegoldásnak is köszönhetően).

– A lehető legtöbb Linux-disztribúció támogatása „polcról leemelhető formában”.

– A helyi fájlrendszer, a Samba, az NFS és az osztott fájlrendszerek *(distributed file system)* támogatása.

– Az LSM, syscall és a VSM osztott fájlrendszerek vizsgálatának támogatása.

– Átfogó és rugalmas kivételkezelés.

További fontos szolgáltatások lennének:

– Minimalizálni a vírusszkennelés által okozott rendszerterhelést.

– Központi letöltési megoldást biztosítani hálózatos környezetben.

– Bármilyen típusú – legyen az kliens vagy szerver – telepítési képesség.

– Web alapú konfigurálás és állapotfigyelés helyi vagy távoli elérés esetén is.

A *SAV for Linux* – amely képes a fenti feltételek legnagyobb részének megfelelni – az úgynevezett *Decision Caching*[1] technológiát használja a terhelés csökkentése érdekében. Ugyanez a technológia más megoldásokban is szerepel, viszont a Genotype[2] megelőző vírusellenőrző technológiával ötvözve már stabil védelmi megoldást nyújt.

A Linux védelmi frissítések letöltése vegyes hálózatban történhet Windows szerveren keresztül Samba meghajtókra. Tiszta Linux környezetben kijelölhető egy dedikált, ún. proxy szerver, amely a többi gépet is ellátja frissítéssel. Lehetőség van arra is, hogy minden gép külön frissítsen.

Ahhoz, hogy a Linux kivívja méltó helyét a szkeptikusok világában, minden apró, leselkedő csapdát ki kell védeni. Ez pedig hivatalosan támogatott IT biztonságtechnikai megoldások és vírusvédelem nélkül nem valósítható meg. Csak így lehet a Sarbanes–Oxley és HIPAA előírásokkal megnehezített IT világban a Linux számára további elkötelezetteket szerezni.

Hivatkozások

[1] A Sophos fejlesztői: Virus protection isn't just a Windows issue. Jelentés, 2006. február, SophosLabs. URL `http://www.sophos.com/sophos/docs/eng/papers/Sophos-NonWindows-wpus.pdf`.

[2] A Sophos fejlesztői: Why Linux threats mean business. Jelentés, 2006. március, SophosLabs. URL `http://www.sophos.com/sophos/docs/eng/papers/Sophos-LinuxThreats-wpus.pdf`.

[3] A biztonsági kritikusság besorolása a secunia szerint. URL `http://secunia.com/about_secunia_advisories/`.

[4] A Descision Caching technológia. URL `http://www.sophos.com/products/es/endpoint/sav-linux.html`.

[5] Ernst and Young könyvvizsgálócég: A Sarbanes–Oxley törvény, 2006. URL `http://www.ey.com/global/content.nsf/Hungary/Sarbanes_Oxley_Torveny`.

[6] Ferbal Könyvvizsgálói Tanácsadó és Szolgáltató Bt: A 2002. évi Sarbanes–Oxley szabályok, avagy a könyvvizsgáló függetlenségére vonatkozó ajánlások, 2003. június. URL `http://www.ferbal.hu/publications/sarbanes.doc`.

[7] A Genotype technológia. URL `http://www.sophos.com/security/topic/behavioral-protection.html`.

[8] Jeffrey Hewitt: Linux making strong inroads in server market. Jelentés, 2005. április 4., Gartner Group. URL `http://www.gartner.com/DisplayDocument?ref=g_search&id=476459`.

[1] A Decision Caching [4] egy szabadalmaztatott technológia, amely lehetővé teszi a teljesítménynövelt hozzáféréskori [9] szkennelést. Egyszerűsített lényege, hogy csak azok az állományok kerülnek ellenőrzésre, amelyek az utolsó vizsgálat óta megváltoztak.

[2] A *Genotype* [7] egy viselkedésalapú technológia, amely a meglévő antivírus-motor olyan képessége, hogy felismerje a futtatható állományok azok elindítása előtti gyanús viselkedését.

[9] Rainer Link: *On-access virus scanning on Linux/Unix*. Elhangzott a *LinuxTag* című konferencián. Wiesbaden, 2006. URL http://www.openantivirus.org/talks/linuxtag2006-on-access-virus-scanning.pdf.

[10] A Linux/Mare-A féreg, 2005. december. URL http://en.securitylab.ru/virus/243422.php.

[11] Micskó Gábor: Az Oracle beszállt a Linux-bizniszbe saját, módosított red hattel. Hungarian Unix Portal, 2006. október 26. URL http://hup.hu/node/6185.

[12] A Sarbanes–Oxley törvény hivatalos honlapja. URL http://www.sarbanes-oxley.com/.

[13] Secunia advisories: Debian update for ClamAV, 2006. október 23. URL http://secunia.com/advisories/21939/.

[14] Secunia advisories: Debian update for ClamAV, 2006. augusztus 21. URL http://secunia.com/advisories/21562/.

[15] Secunia advisories: Debian update for mozilla-firefox, 2006. augusztus 30. URL http://secunia.com/advisories/21675/.

[16] Secunia advisories: Mozilla thunderbird multiple vulnerabilities, 2006. szeptember 15. URL http://secunia.com/advisories/21939/.

[17] Mark Shuttleworth: Fundamentally, this is free software in a proprietary wrapper, 2006. október 25. URL http://blogs.the451group.com/opensource/2006/10/25/fundamentally-this-is-free-software-in-a-proprietary-wrapper/.

[18] Gregg Stults: An overview of Sarbanes–Oxley for the information security professional. Jelentés, 2004. május 9., SANS Institute. URL http://www.sans.org/reading_room/whitepapers/legal/1426.php.

[19] A Troj/Lindoor-B hátsóbejáratú trójai program. URL http://secunia.com/virus_information/22645/lindoor-b/.

Víruskérdések

Mátrai József
<jmatrai@virusbuster.hu>

programozó,
VirusBuster Kft.

Kivonat

Miért van Linuxon antivírus, Windowson meg antivírus és antispyware? Mit kerget a vírusirtó? Van ami káros, de nem a vírusirtó hatáskörébe tartozik? Minden olyan dolog, melyet a vírusirtó fertőzöttnek jelez, és a felhasználók szoktak vele találkozni, az működőképes dolog? Mi az a töredékminta? Milyen evolúciós lépései voltak a káros programok fejlődésének és melyik lépésnél tart a Linux világ? Hogyan lehetnek megtévesztőek a vírusstatisztikák és mennyire gyakoriak a linuxos vírusok? Mire jó a Linux a más OS alatt futó vírusok elemzésekor?

A fenti kérdések megválaszolásán túl elemzünk néhány Linux alatt életképes kártevőt.

Tartalomjegyzék

1. Kérdések és válaszok

Az alábbiakban azokra a kérdésekre igyekszünk választ adni, melyeket a 2005. évi GNU/Linux szakmai konferencián a felhasználók tettek fel a linuxos vírusokkal, linuxos víruskereséssel kapcsolatban.

1.1. Mi ellen van az antivírus (AV)?

Eredetileg majdnem minden káros program fájlfertőző vagy boot szektort (esetleg partíciós táblát) fertőző vírus volt. Az ezredfordulóra megfordult a trend, és jelenleg a káros programok nagy része trójai program, vagy féreg.

fájlfertőző vírus (file infector) ♦ Egy program ilyen vírussal fertőzött, ha elindítása után más programot úgy módosíthat, hogy annak is az itt leírt tulajdonsága lesz.

féregprogram (worm) ♦ Önmagát a hálózaton át továbbítani tudó program, vagy a helyi gépen önmagáról sok másolatot készítő program.

hátsó ajtót nyitó program (backdoor) ♦ A géphez a felhasználó akarata ellenére távoli elérést nyújtó program. Az SSH-daemon nem az! A *backdoor* vagy önmagát telepíti kérdés nélkül, vagy akadályozza az észrevételét, esetleg akadályozza az eltávolítását.

trójai faló (trojan horse) ♦ Mindenféle önműködően nem terjedő, de káros program.

rootkit ♦ A gép működésre vonatkozó információt elrejtő program: például fájlokat, futó folyamatokat rejt el. Általában egy hátsó ajtó nyomait rejti.

kártevő (malware) ♦ A fentiek bármelyike.

megtréfáló program (joke) ♦ Káros lehet, ha valaki ijedtében adatmentés nélkül kikapcsolja a gépet.

adware ♦ Információt letöltögető és megjelenítő program. Ez magában nem baj, csak az *adware* nem mindig a felhasználó tudtával és beleegyezésével kerül fel a gépre, gyakran nehéz eltávolítani.

riskware ♦ Önmagában nem káros, de használata veszélyt jelent, vagy van olyan káros program amely telepíti, vagy egy ilyen program működéséhez szükséges.

A kár mértéke szempontjából vannak jó, rossz és csúnya szoftverek. A fenti felsorolásban csúnyának (*grayware, potentially unwanted software)* számít az adware, a riskware és a megtréfáló program, a többi kártevő rossznak tekintendő.

1.2. Mit kell tudnia az antivírusnak?

Az antivírus alapfunkciója, hogy eldöntse egy fájlról vagy más objektumról, hogy van-e benne ismert kártevő. A komoly megrendelők elvárják az antivírus-cégektől, hogy független tesztelők minősítsék a termékeiket. Ezek a tesztek rendszerint nem azt mérik, hogy futó programokról el tudja-e dönteni a termék, hogy tevékenységük rossz szándékú-e, hanem csak annyit, hogy az *ismert* kártevőket megtalálja-e.

A tesztelő nem készíthet káros programot és a meglévőeket sem módosíthatja, tehát az alábbi tevékenységek nem megengedettek: „Átírtam a vírusban az `XYZ` stringet `ZYX`-re, úgy látom, így is működik. Megnézem, megfogja-e a kereső.” „Tömörítettem a mintát UPX 2.0 futás közbeni kicsomagolóval.” Néhány tesztelő cég és szervezet a sok közül: VirusBulletin

[4], ICSA, AV-Test GmbH (és Otto-von-Guericke-University Magdeburg) [1], magyar tesztelő a CheckVir [2].

Az antivírust kiegészítő egyéb biztonsági termékek:

– *tűzfal*: A hálózati forgalmat korlátozza.

– *rootkit-detektor:* A működés közben lévő rootkiteket ismeri fel. Megnézi, hogy a rendszerprogram kód- és adatterületeit módosítja-e egy harmadik program úgy, hogy a folyamat, fájl, registry bejegyzés (Windows alatt) elrejtése létrejöhet.

– *antispyware:* még nem alakult ki az egységes fogalma. Van, aki adware-t is antispywarerel detektál és irt. Sok futás közbeni megfigyelésből áll, de van benne szignatúra alapú technika is. Példa antispyware funkció: URL címek blokkolása.

1.3. Minden, amit az antivírus fertőzöttnek jelez, az működő kártevő?

Nem, mert sok a töredékminta. Töredékminták lehetnek:

– Tipikus e-mail töredékminta: az elküldött fájl x bájt hosszú. Az átvett fájl y bájtos, $y < x$, a fájlok első y db bájtja megegyezik.

– Tipikus fájlmegosztásos hiba: a lemásolandó fájl és a lemásolt fájl is x bájt hosszú, és van olyan y egész szám ($0 \leq y < x$), melyre igaz, hogy a lemásolt fájlban az y-adik bájt után csak 0x00 értékű bájt van, de ez a lemásolandó fájlra nem igaz. A fájlok első y db bájtban megegyeznek.

– Egyéb: például a fájlból hiányzik az összes 0x0D bájt (a sorvégejel-konverzió miatt).

Honnan lehet tudni, hogy egy fájl ép minta-e? Futtatható fájlok esetén a fejlécből következtethetünk a minimális méretre. Például a fejlécnek a szekciókra vonatkozó részét a `/bin/grep` ELF programfájl esetén az `objdump -h /bin/grep` paranccsal írathatjuk ki. A parancs kimenete ez lehet:

```
/bin/grep:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .interp       00000013  08048134  08048134  00000134  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .note.ABI-tag 00000020  08048148  08048148  00000148  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .hash         0000031c  08048168  08048168  00000168  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .dynsym       00000640  08048484  08048484  00000484  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .dynstr       000003e0  08048ac4  08048ac4  00000ac4  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .gnu.version  000000c8  08048ea4  08048ea4  00000ea4  2**1
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .gnu.version_r 00000070  08048f6c  08048f6c  00000f6c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  7 .rel.dyn      00000030  08048fdc  08048fdc  00000fdc  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  8 .rel.plt      000002b0  0804900c  0804900c  0000100c  2**2
```

```
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 9 .init          00000017  080492bc  080492bc  000012bc  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
10 .plt           00000570  080492d4  080492d4  000012d4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
11 .text          0000db20  08049850  08049850  00001850  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
12 .fini          0000001b  08057370  08057370  0000f370  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
13 .rodata        00002294  080573a0  080573a0  0000f3a0  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
14 .eh_frame_hdr  00000024  08059634  08059634  00011634  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
15 .data          00000034  0805a000  0805a000  00012000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
16 .eh_frame      0000007c  0805a034  0805a034  00012034  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
17 .dynamic       000000c8  0805a0b0  0805a0b0  000120b0  2**2
                  CONTENTS, ALLOC, LOAD, DATA
18 .ctors         00000008  0805a178  0805a178  00012178  2**2
                  CONTENTS, ALLOC, LOAD, DATA
19 .dtors         00000008  0805a180  0805a180  00012180  2**2
                  CONTENTS, ALLOC, LOAD, DATA
20 .jcr           00000004  0805a188  0805a188  00012188  2**2
                  CONTENTS, ALLOC, LOAD, DATA
21 .got           0000016c  0805a18c  0805a18c  0001218c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
22 .bss           0000097c  0805a300  0805a300  00012300  2**5
                  ALLOC
```

A kilistázott 22 szekció közül a legfontosabbak:

- *.text:* a programkód,
- *.rodata:* a konstans globális változók,
- *.data:* a kezdeti értékkel rendelkező globális változók,
- *.bss:* a kezdeti értékkel nem rendelkező (inicializálatlan) globális változók.

Minden szekcióra meg van adva, hogy mettől meddig tart a memóriában és a programfájlban. A minimális fájlméret meghatározásához a `CONTENTS` flaggel rendelkező szekciókra ki kell számolnunk a file_off + size összegeket, és a legnagyobbat kell venni, jelen esetben ezt a *.got* szekció adja: $0x1218c + 0x16c = 74488$. Esetünkben a vizsgált fájl mérete 75692 bájt, ami legalább akkora, mint a minimális fájlméret, tehát a fájl nem töredékminta.

1.4. Vannak-e más furcsaságok, melyeket sok antivírus detektál?

Kedvenceim a *Trojan.BAT.FormatC.A*-hoz hasonló néven felismert minták. Ezek azon fájlokra illeszkednek, melyek a `format c:` stringet tartalmazzák. Ez, a DOS-os időkből megmaradt, ma már általában kárt okozni nem képes parancs a merevlemez formázását végzi, amely egyben a tárolt adatok törlésével jár. A parancs ma azért nem tud kárt okozni, mert a `c:` rendszerlemez a rendszer működése közben nem formázható. Véleményem szerint ezt a mintát nem kéne senkinek sem felismernie, de sajnos sokan nem értenek velem egyet.

1.5. Milyen evolúciós lépései vannak a malware fejlődésnek tetszőleges platformon? Hol tart a linuxos világ?

A kártevők fejlődésnek egy tetszőleges rendszeren az alábbi állomásai vannak:

– *Proof of concept:* valaki csinál egyet, hogy lássák, hogy meg lehet csinálni, és hogy lássák, hogy a készítő milyen okos. Hogy legyen erre az OS-re is, erre a CPU-ra is, erre az interpretált nyelvre is stb.

– Vadon élő minta megjelenése: ez a változat már kárt okoz a felhasználóknak, a káreseteket jelentik.

– Polimorf (önmagát változtató) kód megjelenése: akár annyira változékony is lehet, hogy nincs benne állandó, csak rá jellemző bájtsorozat.

– Lopakodó minta *(stealth malware)* megjelenése. Ez az állomás időben felcserélhető az előzővel.

– Tömeggyártás, mintapolimorfizálás: az új variáns generálása részben automatizálható. A generálás lépései: forráskód módosítása, fordítás, csomagolás röptömörítővel[1].

Bootvírusok esetén az 1–4. pont közel egy időben valósult meg (1986 körül), az 5. pontig nem jutott el a fejlődés. A bootvírusok az operációs rendszer előtt töltődnek be, ezért attól függetlenül is tudnak működni, ám nem minden bootvírus működik minden operációs rendszerrel.

Az MS-DOS-ra írt fájlfertőző vírusoknál az 1–4. állomások közel egy időben jelentek meg (1986 körül). Az 5. pont nem valósult meg a vadon élő vírusok terén, de léteznek automatizált DOS fájlfertőző vírusvarináns-generáló programok. Néha napjainkban is kérnek segítséget a *OneHalf* (DOS fájlfertőző és bootvírus is) variánsainak eltávolítására, így ő tekinthető a nagy öregnek.

16 bites Windowsra (pl. Windows 3.1) kevés vírus készült. Sokáig a többség DOS-t használt, majd áttértek a 32 bites programokra.

A Windowson futó 32 bites programok története néhány fájlfertőzővel indult. 1998-ban tartoltak a *Win9x.CIH* variánsai taroltak, Magyarországon is felkerültek még újságok CD mellékleteire is. Az ezredforduló után egyre több e-mailben terjedő féreg vált ismertté. Megjelentek a botok, az önmagukat terjesztő, a számítógéphez távoli hozzáférést biztosító programok, számuk 2004 körül megugrott. Ezek nem polimorf, de mintapolimorfizálással készített minták (lásd fent). Érdemes megemlíteni a 2005-ös Sony BMG botránysorozatot [3], melynek első lépése az volt, hogy a Sony szándékosan spyware-t tartalmazó audio CD-t hozott forgalomba. Később visszavonta ezeket a lemezeket, és kiadott egy uninstalláló programot is, ám e programba becsúszott egy kihasználható hiba, ami rootkit bejutását tette lehetővé (voltak, akik ki is használták a hibát). A szükséges technológiát már 2003-ban is alkalmazták.

32 bites, x86-os Linuxra eddig az 1., 2., 3. pont valósult meg. A legtöbb kártevő fájlfertőző.

Itaniumon futó Windows esetén csak az 1. pont megvalósulásáról tudunk.

A SymbianOS-t futtató mobiltelefonokra 2004-ben jelent meg a *Cabir* nevű vírus, ami *proof of concept*nek indult, de állítólag voltak fertőzött felhasználók is. Azóta kb. 100 vírusváltozat készült erre a rendszerre.

A Microsoft Word makróvírusok körében mind az öt pont megvalósult, bár lopakodónak nevezhető minták, ahol a makrók forráskódját nem mutatja a Word, mindig csak egy adott Word-verzión működnek.

[1] A röptömörített program indulásakor a memóriában tömöríti ki a kódot, majd ráadja a vezérlést.

1.6. Mennyire pontosak a vírusstatisztikák?

Sajnos a jelenlegi antivírus-technológia mellet egy már felismert káros program kis mértékű módosítása egy fel nem ismert, új károkozót eredményezhet (lásd multipolimorfizálás). Általában, ha egy kereső felismer például egy *Rbot*-variánst *Worm.Rbot.XY* néven, és azt megváltoztatják, akkor négyféle eset lehetséges:

1. A kereső továbbra is *Worm.Rbot.XY* néven ismeri fel. Ez ritkán történik.
2. *Worm.Rbot.XY.Gen* néven, a *Worm.Rbot.XY* egy változataként ismeri fel. Ez is ritkán történik.
3. *Worm.Rbot.Gen* néven, a *Worm.Rbot* egy ismeretlen fajtájaként ismeri fel.
4. Fel nem ismert minta lesz.

A VirusBuster program jelenleg 5800 különböző Worm.Rbot.* szignatúrát ismer fel, ezek közül 21 általános (*Worm.Rbot.Gen).*

Látható, hogy a gyakran megváltoztatott kártevők egy-egy variánsa nagyon alacsony előfordulású a vírusstatisztikában, még akkor is, ha a kártevő nagyon gyakori. Továbbá sok vírusstatisztika e-mail-csapdás számlálóval működik, ahol ha van egy gép, mely óránként küld egy fertőzött levelet, alaposan megnöveli a mintaszámot, viszont a fájlmegosztáson át terjedő vírusokat nem számolja meg. Egy fájlmegosztás-csapdás számláló meg nem számolja az e-mailen terjedő vírusokat.

1.7. Milyen gyakoriak a linuxos kártevők?

Tapasztalatom szerint 2006 októberében a legtöbb felhasználó az *I-Worm.Opnis, Worm.Rbot* és a *Trojan.DL.Zlob* Windows kártevők újabb és újabb változatainak eltávolításához kér segítséget, ha csak a variánsok víruslaboros felfedezését számítom, és egy variánst csak egyszer veszek figyelembe (a segítségkérések számától függetlenül). Linuxos vírus eltávolításához kért segítség ritka, kb. évente egy eset, míg a windowsos esetek napi munkát jelentenek. Legutoljára 2005 őszén történt egy Linux.RST.B eltávolítás. Ez egy egyszerű fájlfertőző vírus, a /bin könyvtár állományait fertőzi meg. Azóta is bukkantak fel új linuxos kártevők, de ezeket partnerektől kaptuk mintacserével, nem találkoztunk velük fertőzött felhasználóval. Úgy tűnik, hogy a linuxos kártevők gyakorisága kisebb, mint a windowsosak 1/200-ad része. Még nem találkoztunk olyan linuxos kártevővel, melynek tömeggyártásban állítják elő újabb és újabb variánsait. Viszont van olyan féreg, amely viszi magával a forráskódját, és a fertőzött gépen GCC-vel újrafordítja magát.

A teljes VirusBuster adatbázis 2006. október elején 120 000 szignatúra-bejegyzést tartalmaz. Ez nem ennyi darab kártevő, mert vannak generikus felismerések *(*.Gen)* is, és a használt szignatúra bizonyos kártevő-módosításokra érzéketlen. Az 1. táblázat mutatja a szignatúrák eloszlását. Megjegyzés: egyes boothelyeken keresendő bejegyzések szerepelnek a MS-DOS COM és EXE fájlokban keresendők között is.

1.8. Mire jó a Linux a víruselemzőknek?

Internet-emuláció ♦ Elsősorban arra, hogy emulálják vele az internetet. A Debian disztribúcióban van minden, ami kell: DNS-szerver, levelezőszerver, webszerver, FTP-szerver, IRC-szerver, Windows fájlmegosztás-szerver (Samba), azonnali üzenetküldők stb. Továbbá Linux alá számos olyan szoftver létezik, amely naplózza a kliensgépen tanulmányozni kívánt program hálózati tevékenységét. Nem triviális feladat, mert sok windowsos program és rendszerprogram magától is össze-vissza kapcsolódik mindenfelé, ezeket a naplóból ki kell

1. táblázat. A VirusBuster adatbázisának szignatúra-eloszlása 2006. október elején

Windows x86, 32 bites PE EXE	92 000 db
MS-DOS COM és/vagy EXE fájlban keresendő	18 000 db
Microsoft Word, vagy azt is fertőző Office-vírus	5 000 db
MS-DOS BAT	2 000 db
VbScript	1500 db
mIRC script.in	700 db
csak bootolásban részt vevő szektorokban keresendő	100 db
ELF (majdnem mindig Linux x86)	100 db
`format c:` jellegű MS-DOS BAT	80 db
Windows x86, 16 bites NE EXE	50 db
\windows\system32\drivers\etc\hosts	50 db
shell-szkript	20 db
összesen	120 000 db

szűrni, pl. `time.windows.com` kapcsolódásokat. Még Windows hibákat is lehet emulálni a Nephentes segítségével!

Miért nagyon fontos ez? A crackerek gyakran nem írnak le számos stringeket a programkódba, hanem hosszú szövevényes számolással állítják elő azokat. Ha egy program le akarja tölteni a `http://www.crackize.com/xxx.exe` fájlt, akkor csinálhatja így:

1. 1 perc várakozás.
2. A `www.crackize.com`-hoz tartozó IP cím lekérdezése, ha nincs, ezen a gépen soha többé nem fut.
3. Most történik meg a `/xxx.exe` sztring előállítása és a fájl letöltése.

Emulált internettel nyomkövetés *(debug)* és változók módosítása nélkül, kipróbálással meg lehet állapítani, hogy a program mit tölt le.

Fájlrendszer megváltozásának tanulmányozása, fájlrendszer helyreállítása ♦ Könnyen lehet disk image másolatot készíteni. Például floppy image készítés:

```
cp /dev/fd0 floppy_image.bin
```

Helyreállítás:

```
cp floppy_image.bin /dev/fd0
```

Az eljárás működik, hacsak nincs furmányosan formázva a lemez.

Linuxszal könnyen vizsgálható egy fájlrendszer állapotának megváltozása teszt előtt és után, és könnyen visszaállítható az eredeti állapota.

Féregcsapda (wormtrap) ♦ A botok állandóan próbálkoznak újabb és újabb felhasználói nevekkel és jelszavakkal bejelentkezni más gépekre, oda felmásolni, majd lefuttatni magukat. A Samba rávehető, hogy több felhasználónév–jelszó párost fogadjon el, mint egy normál windowsos gép, így gyorsabban összeszedi a mintákat. A Nephentes Windows hibaemulátorral való gyűjtés előnye, hogy a minta nem fut le, így a féregcsapda nem fertőz meg más gépeket.

A malware elemzés a világ legjobb példája olyan esetekre, ahol a Linux teljesen kiszoríthatná a Windowst. Ez azért nem történt meg (már rég), mert a szabad szoftver készítők nem ügyeltek eléggé a jó dokumentálásra, a programok jó kommentezésére. Sokan teljesen elutasítják a mások által diktált szabványokat (például SGML-ben írnak dokumentációt), így a szakemberek félnek a Linuxtól, mert a többségi dolgok elutasítása a rossz közösségi emberekre emlékezteti őket.

2. Példa egy vírusra: PHP.Faces.A

A vírus egy polimorf *proof of concept* PHP fájlfertőző. A fertőző fájlokba saját, 1249 bájt hosszú kódját az eredeti kód elé másolja, ezzel elérve, hogy a fájl lefutásakor ő fusson le előbb. Így kezdődik:

```
<?php
  $file_______ = fopen(__FILE__, "r");
  $head_______ = substr(fread($file______, filesize(__FILE__)), 0, 1249);
  fclose($file______);
```

Először tehát beolvassa a saját kódját a `$head______` változóba.

```
  $polylist__ = array("file______", "head______", ..., "newvarname");
```

Ezután felveszi a `$polylist__` tömbbe az összes változónevet. Erre azért van szükség, mert a polimorf fertőzést a változónevek véletlenszerű megválasztásával fogja elérni. Tehát minden példánya más változóneveket fog használni. (Ettől még nem lesz teljesen polimorf, mert például a PHP nyelvi elemek és beépített függvények, pl. `php`, `fopen`, `__FILE__`, `substr` változatlanok maradnak.)

```
  for($i_________ = 0; $i_________ < count($polylist__); $i_________++) {
    $newvarname = chr(rand(97, 122));
    for($j_________ = 0; $j_________ < 9; $j_________++)
      $newvarname = $newvarname . chr(rand(97, 122));
    $head______ = str_replace("$polylist__[$i_________]",
      $newvarname, $head______);
  }
```

Itt történt meg a randomizáció. Fontos, hogy minden változónév tízbetűs volt és maradt, ezáltal a vírus hossza nem változott (és a fenti fread() hívásban nem kellett a hosszt módosítani).

Most a más fájlok `$head______`-del történő megfertőzését végző kódrész következik, de ezt nem közöljük, mert az antivírus-cégek közti megállapodás szerint mintát egymás közt szabad cserélni, de nem szabad semmilyen harmadik személynek adni. Megjegyzés: ez a kódrész tartalmaz egy tesztet, ami megakadályozza a fertőzést, ha a célfájl a `php.faces` stringet tartalmazza. Vad életre szánt vírus esetén külön figyelni kéne arra, hogy a kód ezt a stringet is polimorf módon tartalmazza.

A vírus a polimorfizmus miatt a gyakorlatban rondább, nehezebben olvasható, például:

```
<?php
  $ypxqrpsqcc = fopen(__FILE__, "r");
  $bbugesqpty = substr(fread($ypxqrpsqcc, filesize(__FILE__)), 0, 1249);
  fclose($ypxqrpsqcc);
  $dhbpgxtamn = array("ypxqrpsqcc", "bbugesqpty", "dhbpgxtamn", "cctsvcopcx",
    "wurwejtvjx", "ccznwozuuo", "uudxleoyja", "ionwdbkwfh", "zohqscoxob",
    "skzmabzbfe");
  for($cctsvcopcx = 0; $cctsvcopcx < count($dhbpgxtamn); $cctsvcopcx++){
    $wurwejtvjx = chr(rand(97, 122));
    for($ccznwozuuo = 0; $ccznwozuuo < 9; $ccznwozuuo++)
      $wurwejtvjx = $wurwejtvjx . chr(rand(97, 122));
    $bbugesqpty = str_replace("$dhbpgxtamn[$cctsvcopcx]", "$wurwejtvjx",
      "$bbugesqpty");
  }
  /* ------- cenzúrázott rész ------- */
  closedir($uudxleoyja);
  // php.faces  (c) by Kefi, 2003
?>
```

3. Röviden néhány más kártevőről

Linux.HLLP.DebiLove.A ♦ A megfertőzendő fájl elejére írja magát. Amikor a megfertőzött fájlt lefuttatják, ideiglenes fájlba másolja az eredeti fájlt, majd lefuttatja. A fájl elé írás a legegyszerűbb fájlfertőzési technika.

A másik népszerű technika a fájl végére másolódás. A legtöbb nem szövegalapú futtatható fájl-formátum érzéketlen a fájl vége után írt bájtokra. Ennek magyarázata a következő. A fejléc részekre osztja a fájlt úgy, mint tartalomjegyzék a könyvet. Mivel a tartalomjegyzékben nincs utalás nagyobb oldalszámra, mint a könyv eredeti oldalszáma, a hozzácsapott oldalak nem változtatják meg a könyv működését, azokban soha senki semmit nem fog keresni. Ennek eredményeképpen a fertőzött fájl futtatható. (Persze a fejlécen kell egy kicsit változtatni, hogy a vírus kódja is lefusson a fájl indításakor: a vírus kódját felvenni a tartalomjegyzékbe, és a kód elejét belépési pontnak jelölni.)

Linux.RST.A és Linux.RST.B ♦ Károsult magyar felhasználók küldték be. A fertőzött program közepére írják magukat, ezért újra kellett építeniük az eredeti program programfejlécét. A `/bin` könyvtárban fertőznek.

Backdoor.Linux.Initen.A ♦ Egy feltört felhasználói gépről származik. Ez a legtechnikásabb linuxos kártevő, amit valaha is elemeztem. Futás közbeni kitömörítővel csomagolt, csak rootként elindítva működik. Belenyúl a kernelmemóriába is. Távoli gépről küldött bájtsorozatokat tud visszafejteni és továbbítani egy újabb gépnek, vagy a `/bin/sh` bemenetére átirányítani. Ráakaszkodik az INT 0x80-ra, ezáltal minden rendszerhívást elkap. Az INT 0x80 a Linux rendszerhívások belépési pontja (olyan, mint DOS alatt az INT 21h). Az INT 0x80 használatának megértéséhe tekintsük az alábbi C programsort (ami a szabványos hibakimenetre ír tíz bájtot):

```
int got = write (2, "TenBytes\r\n", 10);
```

Ez x86, 32 bit assemblyre fordítva így (is) néz(het) ki:

```
PUSH    DWORD 10    ; 10 -berakása a verembe, azaz ESP (veremmutató)
                    ; csökkentése 4-gyel, majd 10 beírása a memóriába
                    ; az ESP címtől kezdődő 4 bájtba
PUSH a "TenBytes\r\n"   ; sztring címe
PUSH    2               ; a standard hibakimenet fájlleírója
CALL NEAR libc6_write   ; meghívjuk a write függvényt.
ADD     ESP, 0x0C       ; ESP := ESP + 12, a verem takarítása
MOV got, EAX            ; visszatérési érték elmentése
```

A libc6_write() szubrutin pedig csak meghívja a write() rendszerhívást:

```
MOV     EBX, [ESP + 0x04]       ; fájlleíró
MOV     ECX, [ESP + 0x08]       ; kiírandó puffer kezdőcíme
MOV     EDX, [ESP + 0x0C]       ; méret
MOV     EAX, 0x04               ; a write() rendszerhívás funkciókódja
INT     0x80                    ; KERNEL meghívása
RET     NEAR                    ; visszatérés szubrutinból, EAX visszadása
```

Magyarázat: a C nyelv a veremben adja át a paramétereket a meghívott szubrutinnak. A kernelt azonban egy egyszerű *CALL NEAR* szubrutinhívással nem lehet elérni. A libc6_write rutin a veremben átadott paramétereket átrakja azokba a regiszterekbe, ahol a kernel várja azokat. A *CALL NEAR* csak annyit tesz, hogy a veremben megjegyzi 32 biten a következő utasítás címét, majd ugrik az utána írt címre, azaz szubrutinhívást hajt végre. Párja, a *RET NEAR* utasítás pedig a veremből kivett címre ugrik, azaz visszatér a szubrutinból. Az INT

0x80 azonban nem egy egyszerű szubrutinhívás. Létezik egy megszakítás-vektortábla, melybe legfeljebb 256 bejegyzés tehető. Ezek mindegyike valamilyen külső hardware esemény, szoftveres kivétel, vagy szoftveres meghívás hatására elinduló rutint ír le. A rutinok meghívásakor privilégiumszint-váltás és taszkváltás is történhet. Esetünkben az előbbi történik: felhasználói módból (Linux x86 alatt ez a Ring3) kernel módba (Ring0) lépünk. A normál felhasználó jogosultságaival futó program INT 0x80-nal hívja meg a korlátlan jogokkal rendelkező kernelt.

A kernel módban futó kód elől a szoftvert és a hardvert nem védi semmi – ha tehát a kártevő el tudja érni, hogy kódja kernel módban fusson, akkor elvileg bármit megtehet a rendszerrel. Rootként ez alapértelmezésben három lépésben megtehető: 1. a futtatandó kód előkészítése kernelmodulként; 2. a kernelmodul betöltése; 3. a modul kódjának meghívása INT 0x80-on keresztül. A betöltött kernelmodul általában ráakaszkodik az INT 0x80-ra, vagyis ő kezd el futni minden INT 0x80 hívásnál. Általában továbbadja a rendszerhívás kiszolgálásának feladatát az eredeti INT 0x80 rutinnak, de ha a saját, speciális paramétereit kapja, akkor a saját funkcióit hajtja végre – mindezt korlátlanul, kernel módban. Ezt nevezik INT 0x80 hooknak.

Egy lehetséges, kitalált példa, INT 0x80 hook használatára: egy kártevő elrejti saját fájljait az antivírus elől:

1. Az antivírus a readdir() rutin segítségével rekurzívan végigmegy a könyvtárstruktúrán, és minden fájlt ellenőriz.
2. Az antivírus INT 0x80 rutinon keresztül hívja a readdir()-t. Valójában az INT 0x080 hook rutin fut le.
3. Amennyiben nem a readdir() rutint hívják meg, a hook továbbítja a vezérlést az eredeti INT 0x80 rutinnak, és annak visszatérési értékeit kapja meg a hívó.
4. readdir() esetén először meghívja az eredeti INT 0x80-as readdir()-t.
5. Megnézi, hogy a megtalált bejegyzés neve megegyezik-e az egyik elrejtendő fájl nevével. Ha nem, visszatér az eredeti INT 0x80 által visszaadott értékekkel.
6. Egyébként az eredeti INT 0x80 readdir()-t hívja újra, ami már a következő fájlt keresi. Tehát az elrejtendő fájlt az antivírus nem fogja látni.

A *Backdoor.Linux.Initen.A* nem erre használja az INT 0x80 hookot, de a példa jól mutatja, mire gondoltak a DOS/Windows kollégák, amikor a hook technikát kitalálták.

4. Zárógondolatok

Látható, hogy a linuxos kártevők ritkák, de számuk könnyen megnőhet ugrásszerűen is, ez más rendszereken már többször előfordult. A támadásokat nem az operációs rendszer és a biztonsági szoftverek minősége tartja féken, hanem az erős házirend. Míg a Windowst használó felhasználók millió nem tudják, hogy a gépbe távolról is be lehet lépni, boldogan használnak üres jelszót, hogy minél több weblapot nézhessenek meg, a szoftvertelepítők meghagyják a böngésző összes szkript futtató engedélyét. Ilyen rendszereken csoda, ha valaki megússza a fertőzéseket. A Linux-közösség ezt el tudja kerülni, ha nem csak mindenféle termékek használatára beszéli rá a felhasználókat, hanem továbbra is felhívja a figyelmüket a biztonsági házirend fontosságára.

Hivatkozások

[1] AV-Test.org: nemzetközi antivírus-tesztelő szervezet. URL `http://av-test.org/`.

[2] CheckVir: magyar antivírus-tesztelő szervezet. URL `http://www.checkvir.hu/`.

[3] Egyes, a Sony által forgalomba hozott egyes audio CD-ken spyware van (hír). URL `http://www.freerepublic.com/focus/f-news/1526152/posts`.

[4] VirusBulletin: nemzetközi antivírus-tesztelő szervezet. URL `http://www.virusbtn.com/`.

Magas rendelkezésre állású, dinamikus tartalmat kiszolgáló webszerver készítése CARP-pal, OpenBSD alatt

Nagy Róbert

Légrádi Gábor

Süveg Gábor

Kivonat

Az előadás során demonstráljuk, hogyan lehet 30 perc alatt egy olyan fail-over klasztert feltelepíteni, amely csakis szabad szoftvereket használ, és hihetetlenül egyszerű. Beállítjuk a master és a slave kiszolgálókat is, majd különböző hibajelenségeket produkálva megvizsgáljuk, hogy a klaszter nyújtotta szolgáltatás (dinamkus webtartalom: a Magyar BSD Egyesület weboldalának egy példánya) működőképes marad-e.

Tartalomjegyzék

1. Motiváció

Olyan webszerver kell készítenünk, amely majd működéskritikus *(mission critical)* környezetben működik, és minden viszontagság ellenére kimaradás nélkül, folyamatosan üzemel. Éves szinten maximum 5 percet állhat a webszerverünk. Ez azt jelenti, hogy 99,999 % rendelkezésre állású (High Availability – HA) webszervert kell készítenünk. Túl sok pénzünk sincs rá, viszont van a sarokban néhány jobb minőségű szerver. Ezekből kell a feladatot megoldani.

Nem kis feladat ez, és több helyen is buktatókkal teletűzdelt, de azért próbáljuk meg!

2. Követelményelemzés

A magas rendelkezésre állású rendszereknél legnagyobb ellenfelünk a *Single Point of Failure* (SPF), amit magyarra talán úgy lehetne átültetni, hogy az „egy pontból eredő megbízhatatlanság" vagy „egy pontból eredő hibaforrás", vagyis olyan meghibásodás, amely ha bekövetkezik, akkor rendszerünk üzemszerű működése megszűnik. Ha HA rendszert akaruk építeni, akkor rendszerünkből el kell tünteni az SPF-eket.

Képzeljünk el egy fail-over klasztert, azaz legalább két számítógépet (node), melyek – összekapcsolva – ugyanazt a szolgáltatást végzik, hogy ha valamelyik el is romlik, még működjön a szolgáltatás. Ha ezek azonos 230 voltos hosszabbítóba vannak bedugva, akkor hiába figyeli az egyik node szorgalmasan a másik node által küldözgetett hibajelzéseket (*heartbeat*), ha jön a takarító néni, és szépen kitakarítja a konnektorból a betápot. Ebben az esetben az SPF a 230 voltos áramellátás. Vagy képzeljük el *shared SCSI* alapú fail-over klaszter esetén, hogy meghibásodik a node-okat a diszk alrendszerrel összekötő kábel. Ez esetben a frontendek hiába működnek, ha a backend megadta magát. Itt az SPF az illető kábel. Sok egyéb példát lehetne még hozni, de egy biztos; a jelszó az SPF-ek eltüntetése.

Térjünk vissza az eredeti feladathoz: egy webszervert kell építeni, amelynek kihagyás nélkül kell üzemelnie. Tételezzük fel, hogy egy nagy cégnél dolgozunk, ahol külön ember felel az áramellátásért és a hálózati infrastruktúráért, ezért azzal nem kell törődnünk. Tegyük fel, hogy olyan szünetmentes áramforrásokat kapunk, amelyek külön körön vannak, mindegyiket generátor hajtja, ha az áramszolgáltató leáll. Tételezzük fel azt is, hogy a UTP fali aljzatban állandóan van „delej", azaz az ethernet hálózatunk 100 %-os rendelkezésre állással bír. Tekintsünk el az egyéb külső tényezőktől, például attól, hogy bekövetkezhet egy földrengés vagy robbanás, amely miatt a webszerverünk elérhetetlenné válik (egyelőre nem interkontinentális clustert építünk).

Egy ilyen környezetben a mi feladatunk a következő:

1. biztosítsuk, hogy a webszervert futtató vas állandóan, hiba nélkül működjön;
2. biztosítsuk, hogy a vason futó webszerver folyamatosan, kimaradás nélkül szolgálja ki a kéréseket;
3. emellett az összes biztonsági *(security)* és az üzemszerű működést javító *(reliability)* javítás felkerüljön a szerverünkre.

„Ezt lehetetlen megcsinálni! Felmondok!" – mondhatnánk egyszerűen. „Hiába van folyamatos áramellátás, hiába 100 %-os a hálózat, elromolhat az ethernetkártya, tönkremehet a CPU, elfüstölhet a RAM stb. De ha szerencsém van, és ezek nem következnek be, akkor sem tudom a legfrissebb kernelpatcheket feltenni anélkül, hogy újra ne kelljen indítanom a szervert. Két újraindítás egy évben, és már nem is tudom hozni a vállalt rendelkezésre állást."

3. Megoldási lehetőség: CARP OpenBSD-n

A feleadat megoldható a CARP (Common Address Redundancy Protocol) technológiával, amely OpenBSD 4.0-ban alaptelepítésben is a rendelkezésünkre áll. A CARP lehetővé teszi, hogy IP alapú fail-over (hibatűrő) rendszert rakjunk össze, amelyet akár szoftveres módon is állíthatunk, így szoftverhiba esetén is végrehajthatunk egy átterhelést. Mivel dinamikus tartalmat szeretnénk kiszolgálni, ezért több dologra is szükségünk van, mint például a MySQL által nyújtott replikációs megoldásra és egy olyan programra, amely a fájlok szinkronizálását teszi lehetővé (ez az rsync lesz). A megoldás lépései a kiadandó parancsokkal az [1] cikkben olvashatók.

Az előadás folyáman a résztvevők nyomon követhetik ennek a rendszernek a telepítését és beállítását, majd működésének bemutatását és tesztelését.

Hivatkozások

[1] Nagy Róbert: Dinamikus tartalmat szolgáltató webszerver CARP-pal. Hungarian Unix Portal, 2004. június 12. URL `http://hup.hu/node/6247`.

Egy Windows NT-ről Sambára történő átállás gyakorlati tapasztalatai és utóélete

Navrasics István
<nova@vivamail.hu>

Kivonat

A cikk egy kb. 200 gépből álló nagyvállalati hálózat példáján mutatja be a fájlszerver és a felhasználói adatbázis Windows NT 4-ről Debianon futó Samba 3-ra történő migrációját. Az átállás gyakorlati lépéseinek ismertetése után a cikk hosszasabban taglalja a felmerült gondokat, azok megoldásait és elkerülésük módjait, majd beszámol arról, hogy az NT-hez szokott kollégáknak hogyan sikerült elfogadni a megváltozott adminisztrációs felületet.

Tartalomjegyzék

1. Bevezetés

Egy Samba fájlkiszolgáló telepítése és üzemeltetése egy linuxos rendszergazda számára általános feladatnak mondható; sok dokumentáció és sok tapasztalat gyűlt már össze ezzel kapcsolatban. Egy aktívan használt, tartománykiszolgálóként (PDC) működő Windows NT szerver teljes funkcionalitásának lemásolása és Sambára cserélése, mégpedig zökkenőmentes módon, már lényegesen nehezebb feladat. Mivel a témakörben a magyar, illetve az idegen nyelvű szakirodalomban is kevés információ lelhető fel, illetve mivel számos, a levelezési listákon feltett kérdésem megválaszolatlan maradt, bízom abban, hogy a téma érdeklődésre tart számot.

2. Kiindulóhelyzet és célok

Szükségessé vált egy kb. 200 gépből álló, heterogén infrastruktúrájú nagyvállalati hálózat tartományi beléptetőjeként és fájlszervereként működő NT szerver cseréje. A hálózathoz IBM-AIX, Novell, Linux, Windows NT és Windows 2002 szerverek és Windows XP, Windows NT és Windows 9x kliensek csatlakoztak. A fájlszerver cseréjének legfontosabb okai: nem elegendő teljesítmény; hardvercsere; licencszám, támogatás hiánya; működésbeli gondok. Egy újabb Windowsra történő frissítés előtt tettünk egy kísérletet arra, hogy egy Linux operációs rendszeren futó Samba fájlszerverre térjünk át. Ez sikeres volt, így a Windows-upgrade szükségtelenné vált.

Feltétel volt, hogy a meglévő felhasználók, jelszavak, csoportok átvételre kerüljenek, a fájlszerveren tárolt fájlok és a hozzájuk kapcsolódó jogok megmaradjanak, lehetőleg ne kelljen a munkaállomásokat átkonfigurálni és a folyamat minél kevésbé zavarja a vállalat életét.

A felhasználói oldalon nem informatikusokat, hanem irodai dolgozókat kell kiszolgálni, akik érzékenyen reagálnak a szokásos működéstől való legapróbb eltérésre is. A háromfős informatikuscsapat nem Linux-specialistákból áll, hanem mindenkinek értenie kell bizonyos szinten a kábelkészítéstől a vállalatirányítási rendszer sötét bugyraiig mindenhez (természetesen szükség esetén külső támogatással). Emiatt lehetőség szerint átlátható, könnyen kezelhető adminisztrációs módszereket is kellett találni.

3. A migráció folyamata

Az igen részletes Samba3-HOWTO-ban [3] szinte minden Linux-oldali ismeret benne van. A probléma csak az, hogy hiába írtak egy teljes fejezetet *(Migration from NT4 PDC to Samba-3 PDC)* az átállásról, ebben csak igen nagy vonalakban írják le a teendőket, és csak a teljes 800 oldalas HOWTO áttanulmányozása után kerül az ember minden szükséges információ birtokába.

Ebben a cikkben szeretnék rávilágítani azokra a dolgokra, amik a HOWTO migrációval kapcsolatos fejezetéből kimaradtak, más fejezetben szerepelnek, a másik oldalt érintik, vagy amiket máshogy is lehet csinálni.

3.1. A linuxos szerver telepítése

Ezzel a témával itt nem érdemes részletesen foglalkozni. Szükség van egy stabil, szívünkhöz közel álló Linux-disztribúcióra, a Samba csomagra, és arra, hogy a fájlrendszerünk támogassa a POSIX ACL-eket.

3.2. POSIX ACL-ek

Meglepetésemre több – PDC-t üzemeltető – kolléga is úgy nyilatkozott, hogy nem használ POSIX ACL-eket (hozzáférési listákat) [1, 2]. Az *Access Control List* (ACL) a fájlokhoz való hozzáférés szabályozásának egy áttekinthető, hatékonyan használható, biztonságos módja. A POSIX ACL-eket a 2.6-os kernelben minden fontosabb fájlrendszer (ReiserFS, ext2, ext3, JFS és XFS) támogatja; 2.4-es kernel esetén az XFS biztosan, a többi esetében esetleg patchelni kell. Mivel a migrációs célok között a régi jogosultsági rendszer megtartása is szerepel, szükségünk van ACL-ekre, mert ezekkel lehet a legjobban utánozni az eredeti kiszolgáló működését. A POSIX ACL rendszer a hasznosságához képest véleményem szerint nem eléggé ismert, ezért itt részletesebben is írok róla.

A megszokott Unix jogosultsági rendszert (felhasználóhoz, csoporthoz illetve a többiekhez rendelhető írási, olvasási és futtatási jog) azzal egészítik ki az ACL-ek, hogy egy állományhoz vagy könyvtárhoz akár több felhasználó és több csoport engedélyeit is megadhatjuk a szokásos írás-olvasás-futtatás formában. Ezzel kikerülhetők a nyaktörő mutatványok a csoportokkal és felhasználókkal, illetve nem kell biztonsági, kényelmi kompromisszumokat kötni, de legfőképp nem kell változtatni a korábbi szerverüzemeltetésekor kialakult hozzáférésekkel kapcsolatos gondolkodásmódon. [2] részletesen leírja azt a feltételvizsgálatot, ami egy ACL-lel bővített jogosultsági rendszerben eldönti, hogy egy folyamat hozzáférhet-e egy fájlhoz. Egyszerűsítve: a fájl tulajdonosa a tulajdonosi bitek szerint fér hozzá a fájlhoz, egyéb felhasználók a fájl ACL-jeiben felsorolt felhasználói bitek, vagy azok hiányában a felsorolt csoportbitek, azok hiányában pedig az egyéb folyamatokra vonatkozó bitek szerint.

ACL-ek használatához a fájlrendszert (ext2 és ext3 esetén) az `acl` opcióval kell csatolni, és telepíteni kell az `acl` csomagot. Az ACL-eket a `getfacl` paranccsal kérdezhetjük le és a `setfacl` paranccsal állíthatjuk be, de természetesen a chmod, chown, stb. parancsok is szinte változatlan módon működnek az összes ACL-t nem kezelő programmal együtt. Az 1. ábra egy Sambával megosztott fájl ACL-eit mutatja a Windows Intéző tulajdonság-nézetében. Ugyanez a szerveren getfacl-lel megjelenítve:

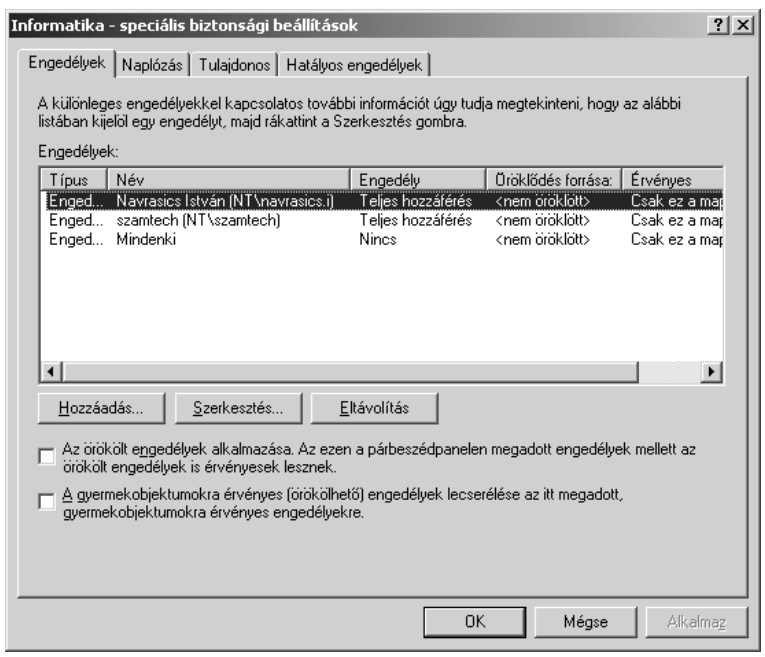
```
# file: Informatika
# owner: NT/navrasics.i
# group: NT/szamtech
user::rwx
group::rwx
group:NT/szamtech:rwx
mask::rwx
other::---
```

Az ACL-ek használatával kapcsolatban számomra az [1] volt a leghasznosabb. Mindenképpen érdemes egy hasonló, részletes ismertetést elolvasni.

Egy egyfelhasználós asztali gép, felhasználók nélküli webszerver vagy adatbázisszerver esetén az ACL-ekre általában nincs szükség, de olyan gépen, amit többen használnak, vagy bármilyen módon fájlszerverként (pl. SMB, FTP) működik, nagy hasznukat vehetjük.

3.3. Felhasználók állományainak átmásolása

A tárolt fájlok átvitelére több út kínálkozik. Csatlakoztathatjuk az NT-n lévő megosztást a linuxos szerverhez, vagy létrehozhatunk egy linuxos megosztást, amit az NT-re csatlakoztatunk. Átvihetjük az adatot valamilyen tárolóeszközön keresztül, illetve hálózaton valamilyen fájlátvitelre alkalmas hálózati protokollal (rsync, scp, FTP, NFS vagy akár SMB). Megoldás lehet akár az átvitelre kerülő adatokat tartalmazó háttértárnak az új kiszolgálóba való beépítése is. Ezen megoldások közös hátránya, hogy nem viszik át a fájlokhoz tartozó hozzáférési jogokat. Néha ez is elegendő. Előnye, hogy használhatjuk a kedvenc fájlkezelőnket annak



1. ábra. Hozzáférési jogok az Intéző tulajdonság-nézetében

minden szeretett tulajdonságával, illetve kiválaszthatjuk, mely fájlokay, könyvtárakat szeretnénk átmásolni.

Ha az ACL-ekre is szükség van, akkor két megoldást tudok javasolni. Az egyik az, hogy egy Windows kliensgépen lementjük a jogokat valamilyen külső eszközzel (pl. a `fileacl.exe`-vel), majd az átállás után visszatöltjük őket az új szerverre; a másik az, hogy az adatok átmásolását a `net rpc share migrate` parancs használatával végezzük (ügyelve a megfelelő paraméterek megadására) – ilyenkor nincs szükség külön lépésre a jogok átviteléhez.

Az alábbiakban az egyes másolási lehetőségeket részletezzük.

Linuxos megosztás csatolása a régi szerverre ♦ A Sambát tartománytag-szerepkörre beállítva létrehozunk egy megosztást. A releváns részlet az `smb.conf`-ban:

```
[global]
  netbios name = migrate
  workgroup = nt
  security = domain
  password server = <régi szerver neve>
  wins server = <régi szerver IP-címe>
  log file = /var/log/samba/log.%m
  max log size = 1000
  syslog = 0
  idmap uid = 10000-20000
  idmap gid = 10000-20000
  winbind separator = /
  winbind enum users = yes
  winbind enum groups = yes
  unixcharset = iso8859-2
  unix password sync = yes

[migrate]
```

```
path = /tmp/migrate
browseable = yes
read only = no
```

Ezt a megosztást hálózati meghajtóként csatoljuk fel az NT-ben, majd az `xcopy` paranccsal (a `/D /E /C /F /H /R` kapcsolók használatával) másoljuk át rá a felhasználói adatokat. A módszer előnye, hogy magára lehet hagyni a gépet, nem fog rákérdezni semmire, el lehet menni kávézni, ebédelni, moziba vagy aludni (az adatmennyiségtől függően). A tevékenység kiválasztásához segítség, hogy a rendelkezésemre álló eszközökkel (100 Mbit/s-es hálózati csatoló, kapcsolt LAN, WinNT 4.0) 6–7 Mbit/s átviteli sebességet sikerült elérni. Ebben a fázisban a két szervert a kapcsoló VLAN funkciójával leválasztottam a hálózatról, bár egyébként nem dolgozott akkor senki az irodákban. A Samba teljesítőképességét növelő ismert finomhangolások alkalmazása (vagy akár a Samba naplózásának átmeneti kikapcsolása) minden bizonnyal jelentős gyorsítást tett volna lehetővé, de ez nekünk nem volt fontos.

Ha már itt tartunk, akkor érdemes a szerveren lévő felhasználók listáját fájlba menteni; így a felhasználói témaszámok *(accountok)* átvitele után meggyőződhetünk arról, hogy nem maradt ki egyik felhasználó sem, ha a létrejött témaszámokat a listában szereplőkkel összehasonlítjuk. Ehhez a Sambát *winbind* használatára kell beállítani, be kell lépni a tartományba a Windowson kiadott

```
net rpc join -w <tartomány> -S <szervernév> -U<admin.felhasználónév>%<jelszó>
```

paranccsal, majd a `/etc/nsswitch.conf` fájlban be kell állítani, hogy a rendszer használja a *winbind* által leképezett neveket:

```
passwd:         files winbind
group:          files winbind
```

El kell még indítani a *winbind* daemont. A tartományban lévő felhasználók listáját a `getent passwd` és a `wbinfo -u` parancsokkal tudjunk lekérdezni. Ha jó az eredmény, tehát látjuk a listában a tartományi felhasználókat, akkor mentsük el a kimenetet, és tegyük el későbbre.

NT megosztás csatolása a Linux fájlrendszerébe ♦ A művelet során leginkább arra kell figyelni, hogy az ékezetek helyesen jelenjenek meg az állományok nevében – már amennyire ez a WinNT esetében lehetséges. A legjobb eredményt az alábbi paranccsal sikerült elérni:

```
smbmount //<szervernév>/<megosztásnév> <csatolási_pont> -o \
 username=<admin_felhnév>,password=<jelszó>,codepage=cp852,iocharset=iso8859-2
```

Ha a Samba szerveren UTF-8-as fájlneveket használunk, akkor a fenti parancsban `iso8859-2` helyett `utf-8`-at kell írni. Ez előtt természetesen az előző fejezetben leírtakhoz hasonlóan kell beállítani a Sambát, és be kell lépni a tartományba.

Megosztások migrálása a net paranccsal ♦ A Sambának saját módszere is van a megosztások átvitelére. Ez a legteljesebb megoldás, mert korrekt módon átvihetők a fájlok, az attribútumok, időbélyegek és a hozzáférési jogok is. A módszer használatához létre kell hozni az új szerveren azokat a megosztásokat, amelyekbe a régieket migrálni szeretnénk. A megosztásnál szükség lesz a *force unknown acl user = yes* beállításra, és futnia kell a Sambának. Én tartományi tag módban futó szerverrel (értsd: *winbind* használatával) próbáltam ki az eljárást (mintakonfiguráció néhány bekezdéssel feljebb).

Megjegyzés: `net` parancs található mind a Windows parancssori eszközei közt, mind a Samba klienscsomagban. E cikkben az utóbbit értjük `net` parancson, tehát a `net` parancsot Linux alatt kell kiadni (általában rootként).

Az alábbi parancs elvégzi az adatok, jogok, attribútumok átmásolását:

```
net rpc share migrate files <megosztásnév> -S <forrásszerver> \
  --acls --attrs --timestamps -U administrator%<jelszó>
```

Ha tartományi tag módban futott a szerver, akkor mentsük le Linuxon a megosztáshoz tartozó jogokat a

```
getfacl -r <könyvtárnév> > <tárolófájl>
```

paranccsal, mert ilyenkor egy-egy felhasználóhoz más UID tartozik, mint a témaszámok átvitele utáni PDC módban. A fájlon le kell majd futtatni egy szkriptet, ami a felhasználók, csoportok nevéből kiszedi a felesleges részeket, ugyanis pl. a tartományi tag szerepben megjelenő *NT/szamtech* csoport átállás után sima *szamtech* csoport lesz, az *NT/navrasics.i* felhasználó átállás után *navrasics.i* lesz, ahol az *NT/* helyébe behelyettesítendő a tartománynév és a *winbind* szeparátorkarakter. A helycsere után ezt visszatölthetjük az alábbi sorral:

```
setfacl --restore <új_tárolófájl> <célkönyvtár>
```

Megjegyzés: a getfacl -r nem menti el a setuid-, a setgid- és a sticky biteket a fájlok jogosultságaiból, de ez itt nekünk nem okoz gondot, mert Windowson ilyenek amúgy sincsenek.

3.4. Csoportok létrehozása és megfeleltetése

Egyes leírások szerint ajánlatos a tartományban lévő csoportoknak megfelelő Unix csoportokat a témaszámok áthozatala *előtt* létrehozni, és a tartományi, illetve unixos csoportokat egymásnak megfeleltetni (`net groupmap`). Én ezt utólag tettem meg, és tudomásom szerint nem történt olyan rendellenesség, amit erre lehetne visszavezetni.

3.5. Hitelesítőadatok átvitele

A linuxos szervert tartalék tartományi kiszolgálónak (BDC-nek) állítsuk be. Az Samba leállítása után lépjünk be a tartományba az alábbi parancs megfelelő paraméterezésével:

```
net rpc join -S <nt_szervernév> -w <tartomány> -U administrator%jelszó
```

Ezután az igen szellemes

```
net rpc vampire -S <nt_szervernév> -U administrator%jelszó
```

parancs elvégzi a felhasználókra és csoportokra vonatkozó adatok replikálását. Ez a lépés a felhasználók kiszolgálásában semmilyen fennakadást nem okoz (értsd: az NT4 szerver továbbra is fut, és ő végzi az autentikációs kérések kiszolgálását – ebben a lépésben még nem cseréljük le).

Ahhoz, hogy ez működjön, az alábbiakra kell nagyon figyelni:

- Az *smbd, nmbd, winbind* kiszolgálók ne fussanak!
- Egyes dokumentációk szerint létre kell hozni egy BDC-témaszámot az NT4 szerveren, de az én tapasztalataim ellentmondanak ennek. Akkor jártam sikerrel, amikor a tartományba való belépéssel egyidőben, automatikusan jött létre a géphez a témaszám, és ezt megelőzően nem létezett ilyen nevű gép a tartományi környezetben.
- Ajánlatos figyelni arra, hogy a sambás felhasználók között ne legyen olyan, akinek a neve megegyezik egy sambás csoport nevével és fordítva; később elmagyarázom, miért.
- Érdemes a standard kimenetet és a standard hibákat fájlba irányítani a későbbi hibakereséshez, ellenőrzéshez.

– Az *add user script, add machine script, add group script* és az *add user to group script* paraméterek az `smb.conf`-ban legyenek korrektül beállítva felkészülve arra az eshetőségre is, hogy az átadott paraméterek majd szóközt is tartalmazni fognak (pl. a *Domain Admins* csoport esetében). Ezt például a paraméterek aposztrófok közé tételével tudjuk biztosítani.

Működő szkriptbeállítások az `smb.conf`-ba (az utolsó 2 sor egy sorba írandó):

```
add user script = /usr/sbin/useradd '%u'
add user to group script = /usr/sbin/adduser '%u' '%g'
add group script = /usr/sbin/groupadd '%g'
add machine script = /usr/sbin/useradd -g machines -c Machine
  -d /var/lib/nobody -s /bin/false '%u'
```

A folyamat a mi esetünkben (kb. 200 felhasználó és 200 gép) néhány másodpercet vett igénybe.

Ellenőrzés: a `pdbedit -L` parancs kilistázza a Samba felhasználókat, a `pdbedit -Lv` *`felhasználónév`* pedig megmutatja egy adott felhasználó fontosabb adatait. Például:

```
Unix username:        navrasics.i
NT username:          navrasics.i
Account Flags:        [UX         ]
User SID:             S-1-5-21-1120865553-1989810170-1905203885-1584
Primary Group SID:    S-1-5-21-1120865553-1989810170-1905203885-513
Full Name:            Navrasics István
Home Directory:
HomeDir Drive:
Logon Script:
Profile Path:
Domain:               NT
Account desc:
Workstations:
Munged dial:
Logon time:           Fri, 01 Sep 2006 07:23:30 GMT
Logoff time:          Tue, 25 Jul 2006 08:41:06 GMT
Kickoff time:         Fri, 13 Dec 1901 21:45:51 GMT
Password last set:    Mon, 02 May 2005 07:33:17 GMT
Password can change:  0
Password must change: Fri, 13 Dec 1901 21:45:51 GMT
Last bad password   : 0
Bad password count  : 0
Logon hours         : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

Érdemes megvizsgálni, hogy minden unixos felhasználónévnek (`getent passwd`) benne van-e a párja a Samba adatbázisában (`pdbedit -L`). Ha az új szerverünk korábban csatlakozott a régire tartományi tagként, és akkor a `getent passwd` vagy a `wbinfo -u` parancs eredményét fájlba mentettünk, akkor egy újabb viszonyítási alap birtokában vagyunk. Javaslom, hogy végezzünk egy rövid ellenőrzést. Bár a tesztek során minden kifogástalanul ment, az éles menetben öt felhasználó nem jött át. Az ellenőrzéshez én a listákat táblázatkezelőbe olvastam, és függőleges keresés *(VLOOKUP)* függvénnyel megnéztem, hogy mindegyik felhasználónév szerepel-e a többi listában; értelemszerűen a `sort` és a `diff` segítségével is célt érhetünk.

Amennyiben nem használunk központi profilokat (tehát a felhasználók a profiljaikat kliensgépenként külön-külön tárolják), akkor felhasználónként ellenőrizzük, hogy a `pdbedit -Lv` szerint mi a profiljuk helye. Ha nem adunk meg felhasználónevet, akkor az összeset lis-

tázza. A kérdéses beállítást a *Profile Path* mező mutatja meg. Ha nem üres, akkor ezt célszerű törölni: `pdbedit -p "" felhasználónév`.

3.6. Csoportok megfeleltetése

A következő lépésként a Windows rendszercélokat szolgáló csoportjait meg kell feleltetni a hasonló szerepű Unix csoportoknak. Erre szolgál a `net groupmap modify` parancs, aminek teljesen önmagyarázó a paraméterezése:

```
net groupmap modify ntgroup="Domain Admins" unixgroup=root
net groupmap modify ntgroup="Domain Users"  unixgroup=users
net groupmap modify ntgroup="Domain Guests" unixgroup=nobody
```

Biztonsági szempontból a *Domain Admins* csoport kritikus (pl. a kliensgépek helyi meghajtóinak `C$`-típusú megosztásokon keresztüli eléréséhez), feltétlenül nézzük meg, hogy kik a tagjai! Érdemes lehet továbbá a vendégek számára külön csoportot nyitni, hogy a hozzájuk tartozó folyamatok ne tudjanak befolyásolni más, a Sambától független, nobody csoportazonosítóval futó folyamatokat (de természetesen általánosságban is célszerű kerülni a nobody felhasználó és csoport használatát).

3.7. Jogosultságok átvitele

Ha a fájlokat a `net rpc share` paranccsal másoltuk át, akkor más feladatunk már nincs. Ha ezt valamilyen okból még nem tettük meg, akkor több lehetőségünk is van.

A Windows XP-t nem sikerült rábírnom, hogy átmásolja különböző meghajtók között az ACL-eket, pedig mind az NT-n, mint a Sambán lévő ACL-eket kezeli, és egy meghajtón belüli áthelyezéskor át is másolja őket. A Windows NT pedig nem dolgozott ACL-ekkel, ha a meghajtó Samba szerveren volt.

Ami megoldás lehet – bár nem teszteltem komolyan –, azok a Windows NT resource kit és hasonló eszközgyűjtemények ACL-manipuláló programjai (*scopy, dumpacl, fileacl, SumInAcl* stb.). Egy szimpatikus eszközzel elvileg le kell menteni a megosztás teljes könyvtárrendszerén belül a jogokat egy harmadik gépen, majd az átállás után visszaállítani azokat fájlból. A legígéretesebb eszköz erre a célra a *fileacl.exe* nevet viseli.

A cégünknél szolgálatot teljesítő fájlszerveren egyetlen nagy megosztást használtak az alkalmazottak, sajnos nehezen átlátható szerkezetben, bonyolult jogosultsági beállításokkal. Emiatt úgy döntöttem, hogy minden egyes könyvtár jogait átnézem, és kézzel állítom be, így egyben ellenőrzöm is, hogy mindenkinek ahhoz és csak ahhoz van-e hozzáférése, amihez kell. Jelenleg folyamatban van egy könnyebben kezelhető könyvtár- és jogosultságrendszer kialakítása.

3.8. Felhasználói profilok

A központi, más néven mozgó profilok (*roaming profile*-ok) használatának sok előnye és sok hátránya van. Mi ezeket eddig sem használtuk, így a migráció során sem okoztak, pontosabban elméletileg nem okozhattak volna problémát. A felmerült gondokról még lesz szó a továbbiakban. Akik használnak profilokat és érdeklődnek a téma után, például a [3] *Profile Migration/Creation* című fejezetében nézhetnek utána a kérdéskörnek.

A mozgó profilok tiltásához a szerveren két dolgot kell tenni. Egyrészt a *logon path* és a *logon script* beállításokat üresen kell hagyni az `smb.conf` fájlban, másrészt pedig minden felhasználó profiljának a helyét külön-külön állítsuk üres sztringre. A profil helyét a

```
pdfbedit <felhasználónév> -p <profil_helye>
```

paranccsal módosíthatjuk.

3.9. Gépcsere

A régi szervert leválasztottam a hálózatról, az új szerver megkapta a régi IP címét, és tartományvezérlő szerepkörben (az `smb.conf`-ban *wins support = yes, domain master = yes, domain logons = yes, security = user*) indítva átvette a tartományi környezet biztosítását. Érdekes, hogy azok a gépek, amelyek be voltak jelentkezve az NT szerverre, *újbóli bejelentkezés nélkül* is belépve maradtak, és hozzáfértek minden állományhoz, amihez joguk volt.

Ennek a megoldásnak nagy előnye volt, hogy minden adat megmaradt a régi gépen, tehát ha bármi balul ütött volna ki, csak egy hálózati csatlakozót kellett volna átdugni a régi rendszer visszaállításához.

4. A megoldott és a még megoldandó problémák

A váltást követő első két munkanap alatt főleg a felmerült apróbb és komolyabb hibák elhárításával foglalkoztam. Ekkor jelentkeztek a későbbiekben bemutatásra kerülő, a profilokkal összefüggő gondok, valamint kiderült, hogy néhány helyen túl szigorúra sikeredett a jogok beállítása. Egy hét után a felhasználók nem jeleztek több problémát.

4.1. Témaszámok elvesztése

A sambás és a unixos felhasználók adatait fájlba írva, és táblázatkezelővel a ket fájlt összehasonlítva, meglepve tapasztaltam néhány hiányosságot. Tisztázatlan okokból 5 felhasználói fiók és néhány (használaton kívüli) gépfiók nem került bele a Samba adatbázisába, noha unixos felhasználóként létrejöttek. Sem a név, sem a beállítások semmi különleges tulajdonságot, karaktert nem tartalmaztak, a sikertelenség pontos okára nem utalt hibaüzenet. Mivel jobb megoldást nem találtam, a hiányzó fiókokat kézzel létrehoztam.

4.2. Profilgond: megváltozott személyesmappa-név

Némely Windows NT és XP kliensen az történt, hogy a *Documents and Settings* mappában eddig a felhasználó nevével egyező nevű könyvtár helyett a felhasználónév.tartománynév mappában kereste a helyi profilt. Ez természetesen nem létezett, emiatt az érintett felhasználók (5 fő) ijedten tapasztalták, hogy nem olyan az asztal, mint amilyen lenni szokott, nincsenek meg a leveleik, dokumentumaik, stb.

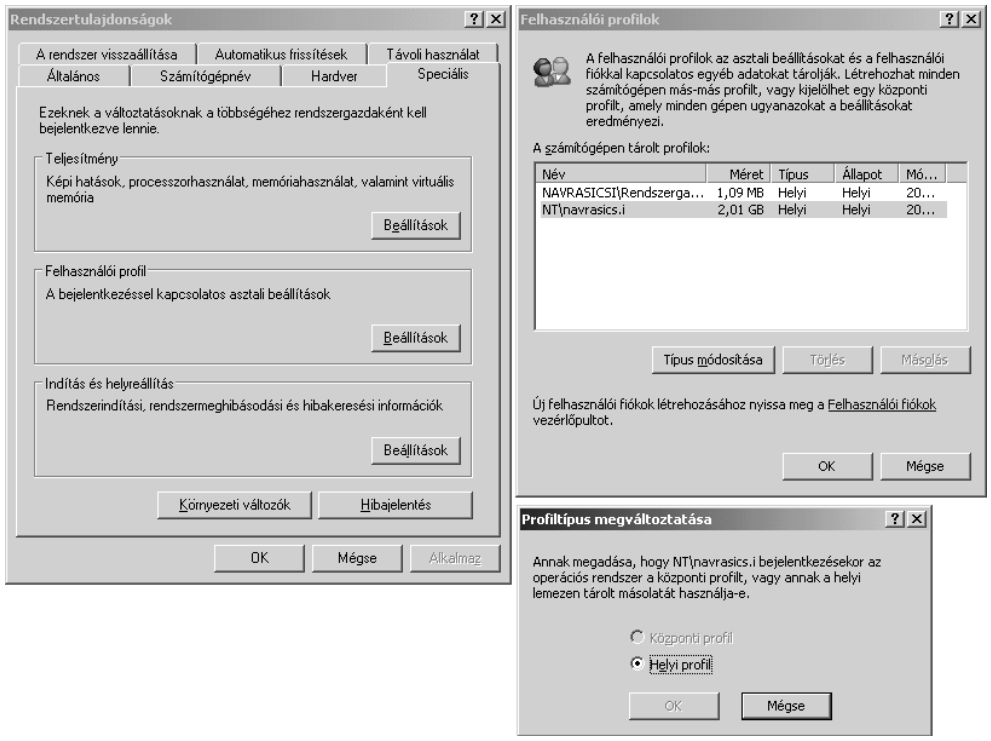
Az eredeti dokumentummappa átmásolásával és néhány beállítás elvégzésével a problémát orvosolni lehetett. A hiba oka az volt, hogy azoknál a felhasználóknál, akiket a `net vampire` hozott létre a linuxos szerveren, ugyanaz maradt a SID, mint korábban volt, de a néhány, „kézzel” utólagosan létrehozott felhasználó esetében ez változott a korábbi állapothoz képest. Emiatt a kliensgépeken futó Windows XP az egyező név ellenére más felhasználónak tekintette őket és más saját könyvtárat és profilt hozott létre a számukra. Windows 98 esetén ez a hiba nem lépett fel, pedig az egyik utólag létrehozott felhasználó gépén ez az operációs rendszer futott.

Hogy ezt a hibát elkerüljük, törekedjünk arra, hogy minden a `net vampire` minden témaszámot átmásoljon, vagy, ha ez semmiképpen sem sikerül, akkor legalább felkészülten várjuk a felhasználó hibajelzését.

4.3. Profilgond: ideiglenes profil használata

Bizonyos felhasználókkal (kb. 3 fő) az történt, hogy a gépük központi profil híján ideiglenes profillal (TEMP néven) léptette be őket. Ez azzal jár együtt, hogy semmilyen személyes beállítást és dokumentumot nem őriz meg a gép két bejelentkezés között.

Ebben az állapotban az ügyfélgépen sajnos nem lehet módosítani a profil típusát (helyi vagy központi) sem felhasználóként, sem rendszergazdaként. A megoldás az volt, hogy létrehoztam a profilkönyvtárat a szerveren (`pdbedit -Lv` *`felhasználónév`* kimenetéből a *Profile Path* mező értékéből megállapítható, hogy hol keresi, és `pdbedit` *`felhasználónév`* `-p` *`profil_helye`* paranccsal be is állítható), beléptem a felhasználó nevével a gépébe, majd a profilját átállítottam központiról helyire (2. ábra). A kilépés után, de még a következő bejelentkezés előtt átneveztem a profilkönyvtárat, és egy darabig még megőriztem.



2. ábra. A profil típusának megváltoztatása (nem a problémás eset)

Hogy ezt a hibát elkerüljük, ellenőrizni kell a felhasználó profiljának útvonalát a `pdbedit -Lv` parancs használatával. Az a jó, ha ez üres – már ha nem használunk mozgó profilokat. Ha nem az, akkor állítsuk üresre már ismert `pdbedit -p ""` *`felhasználónév`* paranccsal.

4.4. Jogosultságok kezelése

Korábban már javasoltam, hogy ne legyen olyan nevű felhasználó a rendszerben, akinek a neve megegyezik egy csoport nevével. Ebből az Intézőben adódhatnak problémák. Kliensgépen a jogok kiosztásánál a nevet be is lehet gépelni, ilyenkor pedig nem egyértelmű, hogy a csoportról vagy felhasználóról van szó. A fájlkezelő veszi az egyiket, és azzal dolgozik. A kétértelműségre semmi sem figyelmeztet. Nehezen felderíthető anomáliákat kerülhetünk el e tanács megfogadásával.

Az Intézőből megtekintve a fájl/könyvtár biztonsági beállításait, zavaró lehet, hogy a listában megjelenik a tulajdonos felhasználó és a tulajdonos csoport egyszer név szerint, másodszor pedig *LÉTREHOZÓ TULAJDONOS* vagy *LÉTREHOZÓ CSOPORT* néven is. Ezt az adminisztrátoroknak meg kell szokniuk.

A megosztáson található fájlok, mappák jogait alaphelyzetben csak a tulajdonosuk mó-

dosíthatja. Ez azért jó, mert a rendszergazda kézben tarthatja a jogok kezelését, ha bizonyos fájlrendszerelemeket saját tulajdonba vesz. Ha mégsem erre van szükség, akkor *DOS file-mode = yes* beállítással az `smb.conf`-ban engedélyezhetjük, hogy bárki manipulálhassa a jogosultságokat, akinek az adott objektumhoz írási joga van.

4.5. DOS-attribútumok (RHSA)

A vállalati tervezőszoftverünk igényli a DOS-os időkből fennmaradt „csak olvasható” attribútum kezelését az ACL-es jogoktól függetlenül. Az alapbeállításokkal a Samba ezt nem kezeli, de bizonyos feltételek teljesülése esetén a feladat megoldható. Két dologra van szükség. Egyrészt az adott megosztás tulajdonságai között meg kell adni az *ea support = yes* beállítást. Ha a megosztást tartalmazó fájlrendszer (Ext2FS, Ext3FS ilyen) ismeri az *user_xattr* opciót, és ezzel is csatlakoztattuk, akkor dolgozhatunk a jó öreg attribútumokkal. (Az *ea* és az *xattr* az *extended attributes* kifejezés rövidítése, ami bővített attribútumot jelent.) Példa a `user_xattr` használatára az `/etc/fstab`-ban:

```
/dev/md3  /home  ext3  defaults,acl,user_xattr  0  2
/dev/md4  /srv   ext3  defaults,acl,user_xattr  0  2
```

4.6. Vírusvédelem

A teljes vírusvédelem még nem megoldott; időnként lefuttatjuk a ClamAV-ot az állományokon, de a védekezést valamilyen kereskedelmi termékkel szeretnénk megvalósítani. A programok kipróbálása és az árajánlatok bekérése, értékelése folyamatban van.

Szubjektív véleményem szerint a kipróbált programok közül az általunk használt Samba- és Debian-verziókhoz egy magyar és egy szlovák cég termékeinek illesztése sikerült a legegyszerűbbre, legáttekinthetőbbre – már amennyire zárt forráskód esetében áttekinthetőségről beszélhetünk. A magyar termék a samba-vfs lehetőséget használja ki a hozzáféréskori védelem megvalósítására. A szlovák szoftver az `LD_PRELOAD` módszerrel fogja el a Samba fájlműveleteit vagy a DaZuKo segítségével vizsgálja a megnyitandó állományokat, és fertőzöttség esetén blokkolja a hozzáférést.

4.7. A linuxos megoldás elfogadtatása

Az idei konferencia egyik vonulata a Linux elfogadtatása a munkahelyen. Ezzel kapcsolatban igen jó tapasztalataim vannak. Két informatikus kollégám közül az egyik a nyugdíjkorhatár környékén már nem igazán motivált új technikák elsajátítására, és ez tulajdonképpen nem is róható fel neki. A másik kollégám ellenállás nélkül szakít időt néha a Sambával való ismerkedésre, jó kérdésekkel, ötletekkel áll elő, otthon is igyekszik linuxos tapasztalatokat gyűjteni.

A döntéshozót sem kellett meggyőzni a Linux választásáról, mivel tűzfalként és levelezőszerverként is megelégedéssel használunk Linuxot. Az átállást oly módon igyekeztem megszervezni, hogyha valami balul ütne ki, bármikor vissza lehessen állítani a régi kiszolgálót a munkába. A meggyőzés egyik módszere lehet ez, hogy a kockázatot a minimálisra csökkentjük, és ehhez mérjük a várható előnyöket. További érv lehet, ha külső szakértő segítségét vesszük igénybe. Lehet demonstrálni a választott eszköz működését egy tesztkörnyezetben vagy valamilyen részfeladat megoldásával.

A felhasználók kis részének volt negatív élménye a váltás miatt, amit nagyobb tapasztalattal talán el lehetett volna kerülni.

Vállalatunknál az informatikai személyzet nem elvakult híve egyik operációs rendszernek, gyártónak vagy platformnak sem. Szeretjük a feladatokat egyszerűen, kényelmesen elvégezni, tehát felmerült az igény, hogy az adminisztrációs feladatokat grafikus felületen is el lehessen végezni. Erre még nem találtunk maradéktalan megoldást, de részfeladatokat meg

lehet oldani menüvezérelt módon.

4.8. Felhasználókezelés

LDAP alapú felhasználónyilvántartáshoz vannak működő eszközök, de mivel mi tdb háttéren dolgozunk (legalábbis egylőre…), ezek sajnos nem jöhetnek szóba. A Webmin *Users and Groups* és *Samba Windows File Sharing* moduljai elég jól használhatók a felhasználókezelésre. A csoporttagság szabályozása, a jelszóváltoztatás, témaszám tiltása, engedélyezése jól működik.

A felhasználók létrehozása az egyetlen, amit nem érdemes itt elvégezni. Létre lehet hozni unixos felhasználót és a unixos felhasználókat át lehet konvertálni sambás felhasználókká, de ez nagyon kényelmetlen, mert az átalakításánál azokat a felhasználókat kell megadnunk, akiket *nem* akarunk átvenni, ez pedig azt jelenti, hogy minden egyes felhasználónévre és gépnévre egyszer rá kell kattintani. Javaslom inkább a parancssort.

Hiányosságként merült fel, hogy Samba szerveren tárolt *felhasználót nem lehet átnevezni*. Sajnos a törlés és új néven létrehozás nem megoldás, mert megváltozik a felhasználót azonosító SID. Ugyanez igaz a gépekre, bár a gyakorlatban a törlés–létrehozás itt kevesebb bonyodalmat okoz, mint a felhasználóknál, mert a hálózatunkban sehol nem találkoztam olyannal, hogy egy kliensgépre SID alapján hivatkozott volna bármi.

4.9. Fájlkezelés, ACL-ek

A fájlkezelés nagyrészt kedvenc freeware kétpaneles fájlkezelő programunkkal történik a kliensgépeken keresztül. A hozzáférést egyszerűbb esetben a Windows Intézővel szabályozzuk, de sajnos van néhány zavaró tényező (lásd *Jogosultságok kezelése*), ami miatt a getfacl/setfacl pároshoz fordulunk. Jó lenne egy Linuxon futó, menüs, önálló ACL-manipuláló alkalmazás, vagy egy a Midnight Commanderbe, Gnome-ba, KDE-be beépülő modul. Ilyet sajnos nem találtam.

Ha egy linuxos helyi fájlrendszeren belül helyezünk át egy fájlt (mindegy, milyen programmal), akkor az ACL-ek megőrződnek. (Ezt teszteltem az mv paranccsal, Midnight Commanderrel, Nautilusszal, Gnome Commanderrel, mindegyik korrektül működött, amíg az ext3-as fájlrendszeren belül maradtam.) Másoláskor az ACL-ek megőrzése a másolást végző programon múlik; neki kell az újonnan létrehozott fájlon az eredetiéivel megegyező ACL-eket beállítani. A `cp` parancsot például a `-p` vagy a `-a` kapcsolóval kell futtatni ehhez (természetesen ez is csak akkor hatásos, ha coreutils csomagunk elegendően új és már tartalmazza az ACL-ek támogatását). Ha csatlakoztatott Samba-megosztáson próbálkoztam, akkor az ACL-ek elvesztek másoláskor és áthelyezéskor is.

A jogok átvitele megoldható az alábbi módszerrel is. A másolandó fájlnak vagy akár könyvtárnak az ACL-jeit el lehet menteni a `getfacl -r` *`könyvtárnév`* eredményének állományba irányításával. Másoljuk át a fájlokat, majd a `setfacl -restore` *`jogokat tároló fájl célkönyvtár`* parancs kiadásával alkalmazzuk a lementett beállításokat a másolatra. Emlékezzünk rá, hogy ezzel a módszerrel a setuid-, a setgid- és a sticky biteket elveszítjük!

Igazán hasznos lenne, ha valaki olyan programokat, konvertereket találna vagy készítene, amivel operációs rendszerek között is lehetővé válna az ACL-ek másolása (bár a feladat nem egyszerű, mivel az NTFS jogosultságkezelése kifinomultabb a POSIX ACL-eknél).

4.10. Mentések, biztonság

Az adatok mentése egy viszonylag egyszerű módszerrel történik. Időzítve mentés készül a felhasználói adatterületről és a fontosabb beállításokat tartalmazó fájlokról. Ezeket NFS segítségével átviszem egy másik gépre is. (A hálózat architektúrája nem tette szükségessé a

titkosított adatátvitelt.)

Az x86-alapú, Sambát futtató szervert márkás alkatrészekből állíttattuk össze; 3 merevlemez tárolja az adatokat RAID tömbökbe szervezve. Végszükség esetén egy kisebb teljesítményű gép néhány perces munkával átveheti a fő szerver teljes funkcionalitását az adatokkal együtt. A vállalat életében ekkora kiesés megengedhető.

4.11. Időzített feladatok

A régi szerveren futott néhány időzített feladat. Szerencsére ott is a cront (nncron) használtuk az időzítéshez, így a migráció egyszerű volt. A következő napok egyik feladata a DOS-os kötegelt állományok shell-szkriptekké alakítása volt.

4.12. A gyakori adminisztrációs műveletek összefoglalása

Az alábbi lista összefoglalja a gyakori műveleteket az őket elvégző parancsokkal. Ha egy művelet Webminben is megbízhatóan elvégezhető, akkor neve után egy felső indexben W szerepel.

sambás felhasználók listázása[W] ♦ `pdbedit -L`

unixos felhasználók listázása[W] ♦ `getent passwd`

felhasználó beállításainak megtekintése[W] ♦ `pdbedit -Lv` *`felhasználónév`*

belépési jelszó módosítása ♦ `smbpasswd` *`felhasználónév`*

új sambás felhasználó felvétele ♦ `smbpasswd -a` *`felhasználónév`*

új unixos felhasználó felvétele[W] ♦
`useradd -c "Teljes név" -s /bin/false` *`felhasználónév`*

sambás felhasználó törlése[W] ♦ `smbpasswd -x` *`felhasználónév`*

unixos felhasználó törlése[W] ♦ `deluser` *`felhasználónév`*

unixos felhasználó hozzáadása egy csoporthoz[W] ♦
`adduser` *`felhasználónév csoportnév`*

unixos felhasználó kivétele egy csoportból[W] ♦ `deluser` *`felhasználónév csoportnév`*

sambás felhasználó átnevezése ♦ Nem lehetséges.

új unixos csoport létrehozása[W] ♦ `groupadd` *`csoportnév`*

új gép hozzáadása (Win 9x) ♦ `useradd` *`új_gépnév`*`$; smbpasswd -a -m` *`új_gépnév`*, Unix-névnél `$` a végén!

új gép hozzáadása (Win XP) ♦ A kliensen átnevezéskor adminisztrátori nevet, jelszót kell megadni, a többi automatikus.

gép törlése[W] ♦ `smbpasswd -x -m` *`gépnév`*`; userdel` *`gépnév`*`$`

gép átnevezése ♦ Sajnos közvetlenül nem lehetséges. A gépet ki kell venni a tartományból, munkacsoport tagjaként át lehet nevezni és utána kell visszatenni a tartományba. Ekkor a gép SID-je megváltozik!

hozzáférési jogosultságok listázása ♦ `getfacl` *`fájlnév`* vagy Windowsból *helyi menü / Tulajdonságok / Biztonság fül / Speciális gomb*.

jogosultság megadása egy fájlhoz (könyvtárhoz) felhasználónak ♦ `setfacl -m u:`*`felhasználónév`*`:`*`jogok fájlnév`* (jogok: szokásos `rwx`-formában) vagy Windowsból mint fent.

jogosultság megadása egy fájlhoz (könyvtárhoz) csoportnak ♦ `setfacl -m g:`*`csoportnév`*`:`*`jogok fájlnév`* (jogok: szokásos `rwx`-formában) vagy Windowsból mint fent.

jogok megvonása felhasználótól ♦ `setfacl -x u:`*`felhasználónév fájlnév`* vagy Windowsból mint fent.

jogok megvonása csoporttól ♦ `setfacl -x u:`*`csoporttól fájlnév`* vagy mint fent.

jogok megvonása mindenkitől ♦ `setfacl -b` *`fájlnév`* (az adott fájl ACL-jeit törli; a hagyományos unixos hozzáférési jogok megmaradnak) vagy Windowsból mint fent.

jogok lementése ♦ `getfacl -r` *`könyvtárnév`* `>` *`jogokat_tároló_fájl`*

lementett jogok visszatöltése ♦ `setfacl --restore <` *`mentett_fájl célkönyvtár`*

5. A végeredmény

Eddig főleg a gondokról, nehézségekről beszéltem, elsősorban azért, hogy a hasonló műveletre vállalkozó kollégák nálam felkészültebben vághassanak neki a feladatnak. Az átállás a problémák ellenére eredményesen zárult. A felhasználók 90 %-ának munkájában semmiféle negatívumot nem okozott az átállás. Az átállás után az alábbi előnyök jelentkeztek (a korrektség kedvéért: a sebességnövekedés valószínűleg részben, a tárhelybővülés pedig teljes egészében a gépcserének köszönhető):

– Nőtt a beléptetés sebessége.

– Nőtt a fájlok elérésének sebessége.

– Nőtt a rendelkezésre álló tárhely.

– Igen tetemes licencdíjat takarítottunk meg.

– Csökkent a vállalatnak egy szoftvergyártótól való függése.

– Ismét van terméktámogatás a fájlszerverre, igaz, egy kicsit más formában.

Hivatkozások

[1] Jörg Arndt és mások: Access control lists in Linux. In *SuSE Linux Administration Guide*. 2. kiad. 2004, SuSE Linux AG. URL `http://www.novell.com/documentation/suse/pdfdoc/SuSE-Linux-Adminguide-9.0.0.0x86.pdf`.

[2] Andreas Grünbacher: POSIX access control lists on Linux. Jelentés, 2003. április 4., SuSE Linux AG. URL `http://www.suse.de/~agruen/acl/linux-acls/online/`.

[3] Jelmer R. Vernooij – John H. Terpstra – Gerald (Jerry) Carter (szerk.): *The Official Samba-3 HOWTO and Reference Guide*. 2005, samba.org. URL `http://samba.org/samba/docs/man/Samba-HOWTO-Collection/`.

Compiere: nyílt forrású vállalatirányítási rendszer kis- és középvállalkozások számára, Oracle Reportsszal és Oracle 10g-vel

Scheer István

Kivonat

Az előadás a Compiere nyílt forrású, kis- és középvállalkozások számára készült vállalatirányítási rendszert mutatja be. A Compiere moduljai: erőforrás-gazdálkodás (ERP), ügyfélkapcsolatok (CRM), kereskedelem, könyvelés. A rendszer tervezése a világpiaci igények figyelembe vételével történt. Globális piaci megoldásként több nyelvet, pénznemet, adónemet, költségelszámolási módot, könyvelési sémát és több szervezetet is képes kezelni. A szerver webes és vastagkliens-felületen is elérhető. Több telephely is összekapcsolható VPN-nel, például ADSL-vonalon. További funkciói: értékesítés, marketing, szolgáltatások, termelés, beszerzés, elosztás, könyvelés, értékesítési erők automatizálása, ügyfélszolgálat, üzleti partnerek, fizetési határidők, anyaggazdálkodás, árképzés engedménynyújtással, készletgazdálkodás, webáruház, számlakiállítás, bankszámla, pénztár, projektgazdálkodás, teljesítményanalízis, pénzügyi jelentések, jóváhagyások, lekérdezések, OLAP-elemző, beszerzési lánc, ajánlat–megrendelés–szállítólevél–számla.

„Ezt egy egysoros programmal meg lehet oldani” Hatékony szkriptnyelvek Unixon 25 éve és ma

Szabó Péter
<szabo.peter@szszi.hu>

Kivonat

A Unix a kezdetektől fogva olyan eszközökkel kényeztette felhasználóit, melyek együtt lehetővé tették az ismétlődő feladatok gyors automatizálását, ezzel a Unixot hatékony munkakörnyezetté téve a programozók, a programozni szerető felhasználók és a rendszeradminisztrátorok számára. Ezen klasszikus eszközök: a csővezetékek és az adatok szöveges, újrabeolvasható tárolási módja és a szkriptnyelvek megtalálhatók a mai Linux rendszereken is, és a a stabilitás és a biztonság mellett ők is hozzájárulnak a Linux népszerűségéhez a programozni szeretők körében.

A cikk egy vázlatos, forráskód-részletekkel illusztrált körképet vázol fel a ma népszerű szkriptnyelvekről. Célja, hogy a Linux iránt érdeklődők ízelítőt kapjanak az automatizálási lehetőségekről, beleértve azt is, hogy milyen feladatnak milyen nyelvben érdemes nekifogni, továbbá hogy szélesítse azok látókörét, akik néhány szkriptnyelvet már ismernek. A bemutatott nyelvek egy része már 25 éve is jelen volt (Bourne shell, AWK, sed, Make, C shell), de jöttek modern trónkövetelők is (Lua, PHP, Perl, Pike, Python, Ruby, TCL).

A cikkben, az egyes nyelvek szintaxisát illusztrálandó, visszatérő motívumként jelennek meg kettő vagy több különböző nyelven is működő programok, melyek egy gyakorlati hasznára (az interpreter elérési útvonalától független `#!` shebang) is fény derül.

Tartalomjegyzék

1. Szkriptnyelvekről általában

A *szkriptelés* [14] egy olyan szoftverfejlesztési fázist jelent, melyben a már létező, különféle komponenseket kapcsoljuk össze egy új feladat megoldása érdekében, a *szkriptnyelv* pedig egy olyan (általános vagy speciális célú) programozási nyelv, melyen a szkriptelés történik, a *szkript* pedig a szkriptelés során előálló program (forráskódja). A *szkript* eredeti angol megfelelője színpadi szövegkönyvet, forgatókönyvet jelent, utalva arra, hogy az is pontosan leírja, hogy kinek mikor mit kell tennie.

Szkriptelhető lehet például egy stratégiai játék, melynek létező komponensei az egyes épületeket, járműveket és az ellenséges katonákat vezérlő szoftverdarabok, és ezeket szkripteléssel összekapcsoljuk úgy, hogy együtt, összehangoltan megvalósítsák a játék egy pályáját. A Quake 3D lövöldözős játék (FPS) egyike volt az első olyan játékoknak, melynek egy része szkriptnyelven íródott. A fejlesztők dokumentálták, hogyan lehet a játékot szkriptelni, így bárki számára lehetővé vált saját ellenségek, pályák stb. létrehozása. A Quake saját, speciális célú szkriptnyelvet használt, a QuakeC-t. A mai, szkriptelhető játékok körében egyre népszerűbb a Lua beágyazott programozási nyelv, melyről a 3.7. szakaszban részletesen szólunk.

Unix rendszereken a szkriptelés főleg ismétlődő feladatok automatizálására használatos. A szkript általában egy összetett feladatot hajt végre, a konfigurációs fájlokban és a parancssori paramétereiben megadott paraméterekkel, a létező unixos szoftverek felhasználásával, összekapcsolásával. A Unix-filozófia [17] része, hogy az egyes programoknak olyan szöveges kimenetet kell produkálniuk, melyek nem csak ember számára olvashatóak, hanem könnyen feldolgozhatók más programmal. Annak idején a Unixokon vált népszerű a csővezeték fogalma. A csővezeték (angolul *pipe*) lehetővé teszi, hogy egy folyamat a kimenetét ne a terminálra vagy fájlba írja, hanem közvetlenül egy másik program bemenetére küldje. Az összekapcsolás folytatható: a másik program is csővezetéken küldi tovább kimenetét egy harmadiknak stb. Csővezeték-hálózatok (angolul *pipeline*) szkriptelés segítségével könnyen kialakíthatók. Megjegyezzük, hogy programokat nem csak csővezetéken lehet összekapcsolni (hanem például kliens–szerver architektúrában), ez azonban Unix alatt körülményesebb, mint az egyszerű csővezeték. Unixon nem csak rendszeradminisztrációs szkriptek vannak, hanem a programozni szerető felhasználók gyakran saját munkájukat automatizálják szkripteléssel.

Egy általánosabb, laza definíció szkriptnyelvnek tart minden olyan programozási nyelvet, melynél fordítás és linkelés nem szükséges (tehát a forráskód elkészítése után közvetlenül futtatható), továbbá a nyelv és a hozzátartozó programkönyvtárak bizonyos feladattípusok programozását jelentősen leegyszerűsítik más nyelvekhez képest. Fő eltérés az előző, komponensintegrációs definícióhoz képest, hogy a laza definíció megengedi egyetlen komponensből álló szoftverek készítését az adott szkriptnyelven.

Cikkünk a Unixon használt legelterjedtebb szkriptnyelveket tárgyalja – ezeken felü is több tucat egyéb, széles körben használt szkriptnyelv létezik Unixra. A tárgyalt nyelvek listája az első verzió kiadásának évével és WikiPedia-beli [19] címszavukkal együtt mutatja az 1. táblázat. Külön meg vannak jelölve azok a nyelvek, melyek csak a laza definícióba férnek bele. A tárgyalt nyelvek mindegyikének létezik szabad szoftver implementációja, és többnyire egyetlen ilyen implementáció terjedt el.

Interpretált nyelvek ♦ Programozási nyelvek között különbséget szokás tenni a forráskód beolvasása szerint. Az ún. *tiszta interpretált* nyelvek esetén (a cikkben tárgyaltak közül a Bourne shell és C shell) egy utasítás beolvasása után az rögtön végrehajtódik, és a következő utasítás beolvasása csak ezután következik. Az egyéb interpretált nyelvek az egész forrásfájlt beolvassák (és esetleg bájtkódot vagy végrehajtási fát építenek), majd a kód memóriabeli, teljesen felépített változatát hajtják végre. A végrehajtás *interpretált,* vagyis a programot a CPU nem közvetlenül hajtja végre, hanem egy olyan segédprogramot, ún. *interpretert* futtat, amely végigmegy a program utasításainak memóriabeli reprezentációján, és minden utasítás-

1. táblázat. Elterjedt szkriptnyelvek Unixon

25 éve is megvoltak:		
Bourne shell	1977–	http://en.wikipedia.org/wiki/Bourne_shell
AWK	1977	http://en.wikipedia.org/wiki/AWK_programming_language
C shell	1979	http://en.wikipedia.org/wiki/C_shell
Make	1977	http://en.wikipedia.org/wiki/Make
sed†	1974	http://en.wikipedia.org/wiki/Sed
Újkeletűek, aktív fejlesztés alatt állnak:		
Lua	1994–	http://en.wikipedia.org/wiki/Lua_programming_language
PHP†	1995–	http://en.wikipedia.org/wiki/PHP
Perl	1987–	http://en.wikipedia.org/wiki/Perl
Pike†	1994–	http://en.wikipedia.org/wiki/Pike_programming_language
Python	1990–	http://en.wikipedia.org/wiki/Python_%28programming_language%29
Ruby	1995–	http://en.wikipedia.org/wiki/Ruby_%28programming_language%29
TCL†	1988–	http://en.wikipedia.org/wiki/Tcl

† komponensintegrációra nem használatos

nál az utasítás típusának megfelelően ágazik el. Ezzel szemben ún. *fordított* (angolul *compiled*) végrehajtás esetén a fordítóprogram (angolul *compiler*) az architektúrának megfelelő, a CPU által közvetlenül végrehajtható kódot állít elő (az ún. *binárist*), a futtatáshoz csak erre van szükség, a forráskódra nem. A fordítás igen időigényes, például a Linux kernel egy mai modern PC-n kb. 15 perc alatt lefordul, apróbb eszközök fordítása is igénybe vehet néhány másodpercet. A cikkben tárgyalt nyelvek egytől egyig interpretáltak. Fordított nyelvre példa a C, a C++ és a Delphi, és fordítás történik C# és a Java esetén is.

Az interpretálás gyorsabb betöltést és lassabb végrehajtást tesz lehetővé, mint a fordítás. Egy interpretált program éles változatának futtatásakor felmerül az igény a nagyobb sebességre. Erre vannak különböző technikák (bájtkód, futás előtti fordítás (JIT), beolvasás után cache-elés), ám a gyorsított végrehajtási sebesség is gyakran elmarad az azonos célú, eleve fordított programok sebességétől.

Sok interpretált nyelvre jellemző (és a legtöbb fordított nyelvre nem) az automatikus memóriafelszabadítás, vagyis hogy a futtatókörnyezet a használaton kívüli memóriát felszabadítja. Ennek tipikus eszközei a hivatkozásszámlálás és a szemétgyűjtés. Az automatikus memóriakezelés egyszerűsíti a programkódot, de a plusz adminisztráció miatt lassítja a futást.

Nagy szoftverek szkriptben ♦ A szkriptnyelvek tehát több szempontból segítik a gyors szoftverfejlesztést (pl. jól használható programkönyvtárak, speciális célú nyelvi elemek, a gyors újraindítás az interpretáltság miatt). Vigyázni kell azonban, mert ha a nagy kódolási sietségben nem figyelünk a forráskód minőségére, egy bizonyos méret fölött a kód érthetetlenné, karbantarthatatlanná válik. A szkriptnyelveket sok kritika éri, hogy könnyű bennük rossz minőségű kódot írni. E sorok írója (a tárgyalt szkriptnyelvek közül) 1000 sor fölött nem ajánlja a Bourne shellt, a C shellt és a sedet, 10000 sor fölött pedig az AWK-t, a Make-et és a TCL-t. A többi nyelv (Lua, PHP, Perl, Pike, Python és Ruby) megfelelő eszközöket kínál nagy szoftverek fejlesztéséhez. A lehetőség persze nem garancia: különösen Perlben, PHP-ban és Lua-ban kell tudatosan lassítani és odafigyelni kódolás közben – Python, Ruby és Pike esetén általában természetesebben, kevesebb odafigyeléssel készül a jó minőségű kód. A legnagyobb Perl-modulgyűjteménybe, a CPAN-be [5] kerülő modulok legtöbbje ékes példája a jó minőségű Perl-kódnak.

Egysoros szkriptek ♦ A cikk címében szereplő felkiáltás („ezt egy egysoros programmal meg lehet oldani!”) lehetne akár a tapasztalt, hatékonyan dolgozó Unix-felhasználók jelmon-

data. Ők ugyanis a napi feladataik során nemcsak gyakran használnak, hanem gyakran írnak is szkripteket, sokszor igen rövid, 80 karakteren elférő egysorosakat, mindig a megfelelő szkriptnyelv(ek)en. Vegyük például azt a feladatot, hogy meg kell találni a legtöbb helyet foglaló felhasználót. Ez az alábbi paranccsal megoldható[1]:

```
$ du -sk /home/* | sort -rn | less
```

A fenti parancs a felhasználókat a *home* könyvtáruk méretének csökkenő sorrendjében jeleníti meg. A használt szkriptnyelv a Bourne Shell, amely elvégzi a `/home/*` kifejtését, és három programból álló csővezeték-hálózatot hoz létre (a parancsbeli `|` karakterek mentén). Elképzelhető azonban, hogy egyes felhasználók *home* könyvtárai nem közvetlenül a `/home`-on belül találhatók (pl. `/home/tanar/hannibal`). A javított megoldás, ami a `/home` alkönyvtáraiban található *home* könyvtárakat is megjeleníti, a következő:

```
$ </etc/passwd awk -F: '$6~/^\/home\//{print$6}' |
  xargs du -sk | sort -rn | less
```

(A parancs épp 80 karakter lett.) Ha a felhasználói adatok NIS-adatbázisban találhatók, akkor `</etc/passwd` helyett „`ypcat passwd |`” írandó. Tárolásfüggetlen megoldás is létezik, itt „`getent passwd |`”-re kell cserélni. Ha a `getent` parancs nem elérhető, akkor a megoldás Perlben:

```
$ perl -le 'while(@L=getpwent){print$L[7]if$L[7]=~/^\/home\//}' |
  xargs du -sk | sort -rn | less
```

A fenti három megoldás jól példázza a unixos gyors problémamegoldás menetét. A gyakorlott felhasználó ismeri a használható eszközöket (`du`, `sort`, `less`, `xargs` stb.) és a szkriptnyelveket (Bourne Shell, AWK, Perl stb.), és töprengés nélkül, folyamatosan írja be a csővezeték-hálózatot létrehozó shell-parancsot. (Esetleg a `man`ban utánanéz a programok kapcsolóinak.) Majd, ha a parancs nem pont azt csinálja, amit kéne, akkor módosít rajta (pl. szükség esetén áttér AWK-ról Perlre).

E cikknek nem célja, hogy minden kódrészletet teljes mértékben megmagyarázzon. Ha valaki a régebbi eszközök (Bourne shell, AWK, Make, sed) tömör, velős tárgyalására kíváncsi, olvassa el a klasszikusnak számító [9]-et, vagy egyszerűen Linux alatt olvassa el a megfelelő kézikönyvoldalt (angolul *man page*): *bash(1)*, *mawk(1)*, *sed(1)*, Make esetén pedig az Info oldalt (`info make` paranccsal). A modern szkriptnyelvek mindegyikéhez van tutorial, megtekinthető az adott nyelv honlapján.

Kívülállók elfogadhatatlanul nagynak találhatják azt az erőfeszítést, amivel el lehet jutni arra a tudás- és készségszintre, hogy a fentiekhez hasonló egysoros parancsok gépelése folyékonyan történjen. Alternatív megoldásnak javasolhatják a grafikus menedzsment-felületek használatát (pl. Vezérlőpult), kattintgatni végül is sokkal kevesebb tudással is lehet. Vannak azonban hátrányai is a grafikus felületen történő munkának:

– Egy grafikus felület felhasználója nem élvez teljes szabadságot: elképzelhető például, hogy az ő felületén nincs olyan funkció, mellyel a felhasznált tárterület csökkenő sorrendjében fel lehet sorolni a felhasználókat. Ekkor vagy egyenként, kézzel végignézi az összes felhasználót (sok, értelmetlenül elvesztegetett idő), vagy lekérdez minden adatot a felhasználókról, majd egy grafikus táblázatkezelővel feldolgozza az eredménytáblát (még ez is jóval lassabb a szkriptelésnél).

– Mennyire növekszik az idő, ha ugyanezt az információt távolról, lassú kapcsolaton kell megszerezni? A távoli grafikus felületen való kattintgatás órákig is eltarthat, míg pl. a szkriptek könnyedén futtathatók SSH-n keresztül.

[1] a kezdő dollárjel a promptot jelöli – ne gépeljük be

– Ha a vezetőség naponta jelentést kér a foglalt tárterületről, akkor a grafikus felületen dolgozó alkalmazottra ez minden nap plusz terhet ró, míg a Unix alatt szkriptelő rutinos felhasználó 1 perc alatt megírja a szkriptet, és még 5 perc, amíg létrehozza a `cron` jobot, ami gondoskodik a szkript esténkénti lefuttatásáról, és az eredmény e-mailes elküldéséről a vezetőség számára.

– Ha az adott grafikus felületnek új verziója jelenik meg, sok minden máshol lesz rajta, az alkalmazói tudás egy része elavul, a szokásos munkamenetet újra és újra ki kell dolgozni. A unixos eszközök és a Unix alatt használatos programnyelvek viszonylag lassan változnak: az 1975-ben írt shell-szkript kevés változtatással ma is futtatható (a `du`, `xargs` és társai már akkor is megvoltak), és az 1995-ben írt Perl-szkript is szinte ugyanúgy működik ma is, mint írásakor. A szkripteket író Unix-felhasználó tehát számíthat arra, hogy az egyszer megszerzett tudásának, készségeinek évtizedek múlva is hasznát veszi, és nem kell évente felülírnia őket új ismeretekkel. Továbbá lehetősége van lassan, fokozatosan tanulni, a Unix használatának egyre hatékonyabb módjait elsajátítani.

2. Régi nyelvek

A régi nyelvek közös tulajdonsága, hogy az 1970-es évek Unix rendszereihez fejlesztették őket, egyszerűek, manapság is elérhetők szinte minden Unixon, és a sok implementációnak van egy széles körben elfogadott metszete, egy részнyelv, mely minden implementációban azonosan működik.

2.1. Bourne shell: Unix folyamatok integrációja

A *shell* (magyar fordításban *burok* vagy *héj*) az a program, amely bejelentkezéskor és terminálablak nyitásakor elindul, feladata a felhasználói parancsok fogadása (a sor szerkesztése és a korábban kiadott parancsok visszaadása), a parancsok végrehajtása (átirányítás, fájlnevek kiegészítése `*`-ra és `?`-re, csővezetékek létrehozása, külső program(ok) futtatása), és az indított programok vezérlése (angolul *job control*; háttérbe rakás, előtérben futó program kiválasztása).

Egy egyszerű parancs kiadásakor a shell a szóközök mentén argumentumokra bontja, az első argumentumot a program nevének veszi, és a programot elindítja (szükség szerint megkeresve a `$PATH`-on), átadva neki a további argumentumokat. Vannak a shellnek belső parancsai is (pl. `echo`, ami kiírja argumentumait), ezeket ő maga hajtja végre külső parancs meghívása nélkül. Az elindított program alapértelmezésben a terminálról ír és onnan olvas, és a hibaüzeneteket is a terminálra írja. A program által kilépéskor visszaadott státuszkód a shelltől lekérdezhető. A shellben lehetőség van ezek átirányítására, például a `Bash` shell a `foo <be >>ki 2>/dev/null` parancs hatására a `foo` programot futtatja úgy, hogy az a `be` fájlból olvasson, a `ki` fájl végére fűzze hozzá kimenetét, és hibákat a `/dev/null`-ra írja (amit a Unix eldob). A shell ezen felül még kiegészíti (angolul *globbing*) a fájlneveket, így például a a `foo ?a*` parancsot futtatás előtt kifejti `foo bar ba tart`-tá, ha az aktuális könyvárban `bar`, `ba` és `tart` azok a fájlok, melyeknek második betűje `a`.

A shell nem csak egyszerű parancsot fogad el, hanem ezekből `|` karakterrel összefűzött csővezeték-hálózatot is (és az egyes programok elindításáról, ki- és bemenetük összekapcsolásáról ő gondoskodik), továbbá vannak vezérlőszerkezetek (pl. `if`, `while`), és egyes shellekben függvényeket írhatunk (a függvény egyetlen státuszkódot adhat vissza, ami nemnegatív egész szám). A shell kezel string típusú változókat, melyek parancsok építésekor (általában `$` után) behelyettesítődnek. Az újabb shellek (pl. `Bash`), ellentétben az eredeti Bourne shellel, egydimenziós tömböket is kezelnek, és egész számokkal is tudnak számolni.

Az első Bourne shell a kezdeti unixos shellekből leszűrt tapasztatok után keletkezett 1977-

ben. A mai Linux rendszerek alapértelmezett és egyben legelterjedtebb shellje, a Bash felülről kompatibilis az eredeti Bourne shellel. Az újabb shelltípusok (pl. Korn shell és Zsh) is ilyenek, de egymástól is vettek át nyelvi elemeket. Az ash egy apró, az eredeti Bourne shell szolgáltatásait megvalósító shell, interaktív parancsbevitelt kényelmessé tevő rutinok nélkül – ideális telepítőkészletek, RAM-ból futó beágyazott rendszerek shell-szkripteléséhez (pl. BusyBoxban [4]).

Mint említettük, a Bourne shell tiszta interpretált. Emiatt egy hosszan futó, fejlesztés alatt álló szkript futása közben előfordulhat, hogy módosul a forrásfájl, és a shell a következő utasítást már az új fájlból akarja venni. Ez általában nem kívánatos. Megelőzhetjük, ha a szkript érdemi részét körbevesszük `if true; then ... exit; fi`-vel.

Stringkezelés ♦ Az összefűzésen kívül az összes többi stringkezelő művelet shellben igen keserves, például egy olyan eljárást, ami a V változóban az aaa stringet felváltva bbb-re és ccc-re cseréli, túl körülményes csak belső parancsokkal megírni. Az alábbi megoldás nem működik minden Bourne shellben, de megy Bash-ben, Zsh-ban és pdksh-ban:

```
repaaa() {
  local X="$V" Y=bbb Z="${V#*aaa}";  V=""
  while [ "$Z" != "$X" ]; do
    V="$V${X%aaa$Z}$Y"
    if [ "$Y" = bbb ]; then Y=ccc; else Y=bbb; fi
    X="$Z"; Z="${X#*aaa}"
  done
  V="$V$X"
}
```

Ugyanez Perlben sokkal rövidebb, gyorsabb, és jóval kevesebb gondolkodással előáll:

```
sub repaaa() { my $C=1; $V=~s/aaa/($C^=1)?"bbb":"ccc"/ge }
```

A Bourne shell tehát remek eszköz átirányításra, valamint külső programok integrálásra csővezeték-hálózatokkal és egyszerű vezérlessel, ám nehézkessé válik, amikor stringkezelésre vagy számolásra kerül sor. További probléma, hogy a shell-szkriptek igen lassúak. A fenti problémás esetekben érdemes áttérni pl. Perlre; Perlben ugyanis – a megfelelő kiegészítő modulokkal – minden feladatot meg lehet oldani, és csak súlyos CPU- vagy memóriaínség esetén kell Perlről valamely fordított nyelvre váltani. (Ízlés szerint Perl helyett egyéb modern szkriptnyelvvel is dolgozhatunk.)

Kalandos kedvűek kipróbálhatják a Bash helyett a Zsh nevű shellt, amely számos előnyét ötvözi a Bourne, a C és a Korn shellnek. A legfontosabb előnyei a Bash-sel szemben: interaktív lehetőségei (prompt, fájlnév-kiegészítés) jobban testreszabhatók, egyszerre több kimeneti fájlba is át tud irányítani, továbbá a ∗∗ mintával az alkönyvtárakat (rekurzívan) is ki tudja egészíteni. Mindezek olyan szolgáltatások, melyek kényelmesebbé, hatékonyabbá teszik a mindennapi munkát. A Zsh hátránya például, hogy csak a legújabb verziók működnek jól UTF-8-as terminálon.

Shell-szkriptek írása ♦ A shell használata esetén szinte észrevétlenül válik a felhasználó programozóvá. Például a 148. oldalon található listázó parancsot (amely a home könyvtárbeli helyfoglalást szerint rendez), könnyen szkriptté alakíthatjuk:

```
#! /bin/bash --
# Használat: listaz.sh [<hossz>]
# a foglalt terület szerinti csökkenő sorrendben listázza ki a
# felhasználók home könyvtárait. Csak az első <hossz> db könyvtárat
# mutatja (vagy 20-at, ha nincs <hossz>).
```

```
</etc/passwd awk -F: '$6~/^\/home\//{print$6}' | xargs du -sk |
  sort -rn | head -"${1:-20}"
```

A szkriptet futtathatóvá tétele (`chmod +x listaz.sh`) után a `./listaz.sh 3` paranccsal próbálhatjuk ki.

A Bourne shell a `#` jeltől figyelmen kívül hagyja a sorok tartalmát, így megjegyzés helyezhető el. A `"${1:-20}"` kifejezés pedig a szkript első argumentumát (`$1`) adja vissza, vagy annak hiánya (vagy üressége) esetén `20`-at.

A fájl első, `#!`-lel (az ún. *shebang*-gel) kezdődő sorát a Bourne shell a `#` jel miatt figyelmen kívül hagyja, ám ez a sor nem fölösleges: a kernel számára azt írja elő, hogy az adott fájl futtatásához egy külső programot kell indítani – tehát a `./listaz.sh 3` parancs helyett `/bin/bash -- ./listaz.sh 3` fog lefutni, vagyis a `Bash` interpreter fogja futtatni a `./listaz.sh` shell-szkriptet az egyelemű `3` argumentumlistával. Ez pont az, amit szeretnénk.

Rendszerműködtető shell-szkriptek Linuxon ♦

- `/etc/init.d/*`: a szolgáltatásokat indító és leállító szkriptek,
- `*.ebuild`: Gentoo disztribúcióban a csomagkészítést és -fordítást vezérlő szkriptek (`Bash` alapú, saját keretrendszerrel).
- `preinst`, `postrm` stb.: Debian-alapú disztribúciók csomagjainak felrakásakor és leszedésekor lefutó szkriptek.
- `/etc/cron.*/*`: a `cron` által periodikusan (naponta, hetente stb.) futtatott rendszerkarbantartó szkriptek.
- A 2.4-es Linux kernelben a `make menuconfig`-ra lefutó kernelfordítás előtti konfiguráló program is shell-szkript volt.
- A Debian telepítője is sok egyedi shell-szkriptet tartalmaz.
- A Knoppix Linux LiveCD indulásakor shell-szkriptek vezérlik a hardverfelismerést stb.

GNU-s szabad szoftverek lefordításában is sok shell-szkript vesz részt (`AutoConf`, `AutoMake`, `LibTool`, lásd még [7]), a fordítás előtt lefutó `configure` szkript is Bourne shell-szkript: őt az `AutoConf` generálja a `configure.in` és egyéb fájlokból M4 makrónyelv felhasználásával.

2.2. C shell: a Bourne shell alternatívája

A C shell a Bourne shell egyik elődjéből fejlődött ki, mert akkoriban az elődből számos kényelmi elem hiányzott: indított programok vezérlése, parancstörténet (angolul *command history*), korábbi parancsok felhasználása (angolul *history substitution*), tömbkezelés, home könyvtárok kifejtése `~`-re, álnevek (angolul *alias*), számolás, fájlnév-kiegészítés *Tab*-bal. Ezek manapság a `Bash`-ben mind megvannak, ezért a C shell választása nem jár számottevő előnnyel.

Unixra a legelterjedtebb, továbbfejlesztett implementáció a `tcsh`.

A C shell a C nyelvről kapta a nevét, mert szintaxisa a C nyelvéhez hasonló (ami kicsit könnyebben olvasható, mint a Bourne shell-szkriptek) Ezért az apró esztétikai előnyért azonban nagy árat fizetett: nem tudja futtatni se a Bourne shellre, se a Bourne shell elődjére írt szkripteket. A C shell tehát megosztja a shell-szkriptek fejlesztőit: vagy Bourne shellre fejlesztenek, ami minden Unixon futtatható, vagy C shellre, ami sokkal kevésbé ismert és

használatos (manapság a legtöbb Linux-disztribúció nem rak fel alapból C shell implementációt). Emiatt többnyire Bourne shell-szkriptek születnek.

A C shellt szintaxisában levő precedencia miatt kritika érte. Például az

```
if ( ! -e foo ) echo bar >foo
```

utasításban a `>foo` átirányítás akkor is végrehajtódik (üres fájlt hozva létre), ha a feltétel nem teljesül.

C shell és Bourne shell szétválasztása ♦ Manapság a `/bin/sh` általában egy Bourne shellre mutat (Linux alatt a `Bash`-re, beágyazott Unixokon általában a `BusyBox`-ba [4] épített `ash`-ra). Csak évtizedekkel ezelőtt volt szokás egyes rendszereken C shellt berakni `/bin/sh`-nak. Ma már nem szükséges tehát, hogy a `#! /bin/sh` shebanggel kezdődő shell szkriptek felkészüljenek, hogy esetleg Bourne shell helyett C shellben futnak.

Ám – a lényesen eltérő szintaxis ellenére – kis trükközéssel elérhető, hogy ugyanaz a szkript fusson mindkét shelltípusban. Persze csak a legeleje fut mindkettőben: a kód egy elágazással kezdődik, amely vagy a csak Bourne shellben, vagy a csak C shellben működő részre ugrik. A helyzetet bonyolítja, hogy már az elágazó utasítás (`if`) szintaxisa is lényegesen különbözik a két shelltípusban. Azért van megoldás:

```
eval '(exit $?0)' && eval '#\
echo Ez csak Bourne-kompatibilis shellben, pl. ash, bash, ksh, zsh; exit'
echo Ez csak C shellben
```

A szétválasztás azon alapul, hogy a `$?0` C shell esetén `1`-et ad vissza, míg Bourne shell esetén `00`-t (feltéve, hogy a legutoljára befejeződött parancs státuszkódja 0). A többi utasítás (`eval` és `exit`) és szintaktikai elem mindkét shellben ugyanazt csinálja.

2.3. Make: inkrementális szoftverfordítás

A Make nyelv több, egymástól függő műveletre bontható munka leírását teszi lehetővé. Ilyen munka például egy szoftver lefordítása (fordítás: forrásfájlokból objektumfájlok készítése, majd linkelés: az objektumfájlokból bináris linkelése, majd a dokumentáció generálása) vagy egy LaTeX-dokumentum lefordítása (pl. forrás → DVI → PostScript). A leírás alapján a Make program (Linux rendszereken általában a `GNU Make`) az egyes műveleteket automatikusan sorrendezi: ha egy *B* művelet függ az *A* művelettől, akkor a sorrendben *A* meg fogja előzni *B*-t. Ezután a Make a kialakított sorrendben végrehajtja a műveleteket. A Make inkrementálisan dolgozik: ha a munka elvégzése után néhány fájl megváltozik (melyektől függ a munka eredménye), és újrafuttatjuk a Make-et, akkor csak azokat a műveleteket hajtja végre, melyekre a megváltozott fájlok hatással vannak. Ily módon egy szoftverfejlesztés (módosítás, lefordítás, kipróbálás) során lerövidíthető a fordítással töltött idő, ha azt Make-kel végezzük: ugyanis csak a megváltozott fájlok (és a tőlük függő esetleges egyéb fájlok) kerülnek újrafordításra. Az időnyereség manapság is számottevő: nem mindegy, hogy 500 db C++ fájlból 1-et vagy 500-at kell újrafordítani, ha egy fájl fordítása 2 másodpercig tart.

A Make által végrehajtandó műveleteket szabályok írják le. Példa szabályra:

```
# A harmadik sort tabulátorral kell kezdeni.
foo.o: foo.c foo.h bar.h
        gcc -c -o foo.o foo.c
```

A fenti szabály jelentése: ha a `foo.o` fájl nem létezik, vagy régebbi, mint a `foo.c`, `foo.h` és `bar.h` fájlok bármelyike, akkor a `gcc -c -o foo.o foo.c` shell-paranccsal készítendő el e fájlokból a `foo.o` új változata.

A Make-ért szerzője 2003-ban *ACM Software System Award* díjban részesült [1].

A GNU Make implementáció számos kiegészítő szolgáltatást nyújt: string típusú változók kezelése, stringműveletek, preprocesszor (a Makefile egyes sorainak feltételes kihagyása), függvény külső parancs hívására (a parancs kimenetét stringként adja vissza), szabály az összes adott kiterjesztésű fájlra, változók felülbírálása parancssorból stb.

A Make fontos szerepet kap a GNU szoftverek fordításában [7]. Egy GNU szoftver fordítása így történik:

1. Az első lépés a forráskód letöltéséve és kicsomagolása.

2. A `./configure` parancsot kell futtatni (szükség esetén a fordítást befolyásoló argumentumokkal, például a telepítési célkönyvtárral és a használandó programkönyvtárakkal kiegészítve), amely nagy vonalakban ezt teszi: detektálja a rendszer néhány jellemzőjét (melyik függvény melyik `.h` fájlban és melyik programkönyvtárban érhető el), felderíti a függőséget a fordításhoz szükséges forrásfájlok között, majd legenerálja a `Makefile.in`-ből a `Makefile`-t (a Make számára) és a `config.h.in`-ből a `config.h`-t (a C fordító számára).

3. A `make` parancs a `Makefile`-ban található szabályokat követve levezényli a szoftver lefordítását.

4. A rootként kiadott `make install` parancs telepíti (felmásolja) a lefordított szoftvert.

A Make-et használják még a Linux kernel fordítására, a NIS felhasználó-adatbázis frissítésére, Debian-csomagoknál a program fordítására és a csomag különböző fájljainak elkészítésére is. LaTeX-dokumentumok Make-kel automatizált fordítására nincs általánosan elfogadott séma, a szerzők szükség esetén egyedi `Makefile`-okat írnak.

Nem véletlen, hogy a függőségek felderítését nem a Make, hanem a `configure` szkript végzi. Erre ugyanis a Make nem képes. További hátránya a Make-nek, hogy körkörös függőségek esetén hibát jelez (pl. LaTeX-dokumentumok többmenetes fordításakor tipikus a körkörös függőség), és nem támogatja az elosztott fordítást sem (vagyis nem lehetséges egy nagy szoftvert több gépen, párhuzamosítva fordítani).

A fenti problémák orvoslására számos szoftver született [15], például dmake (elosztott fordítást támogat, az OpenOffice.org fordításához használják), Ant (Javaban, főleg Java-programok fordítására), SCons (Pythonban), PBS (Perlben), Rake (Rubyban).

Hordozható Make shebang ♦ A Make alapesetben az aktuális könyvtár `Makefile`-jából olvassa a szabályokat. Ezt `-f` kapcsolóval bírálhatjuk felül. A `#! /usr/bin/make -f` shebang segítségével elérhető, hogy a Make-szabályokat tartalmazó fájl közvetlenül futtatható legyen. De mit tegyünk, ha a `make` parancs elérési útját nem ismerjük (lehet például `/usr/local/bin/make` is)? Futtassuk a szkriptet `#! /bin/sh` shebang-gel, és bízzuk a shellre az elérési útvonal megtalálást. Igen ám, de ekkor úgy kell a fájl első néhány sorát megírnunk, hogy shell-szkriptként és Make-szabályfájlként is működjön. Íme a megoldás, kezdjük így a szabályfájlt:

```
#! /bin/sh
#\
eval '(exit $?0)'&&eval '\
M=make;>/dev/null type -p gmake&&M=gmake;exec "$M" -f "$0" ${1+"$@"}'; \
>/dev/null which gmake&&exec gmake -f "$0" $argv:q;exec make -f "$0" $argv:q
# Don't touch lines 1--6: http://www.inf.bme.hu/~pts/Magic.Make.Header
```

A fenti megoldás először a `gmake` parancsot keresi (hasznos például FreeBSD-n, ahol a `make` parancs a BSD Make-jét futtatja, a `gmake` pedig a GNU Make-et), majd a `make`-et.

Működik Bourne és C shellben is (a már ismert `$?0`-s trükkel választva szét a két shellt). A shell és a Make szétválasztásának alapötlete az, hogy a `#\`-t tartalmazó sor shellben egysoros megjegyzés, Make-ben viszont a következő sorban folytatódó megjegyzés. Tehát a Make a fenti egész fejlécet megjegyzésnek tekinti, figyelmen kívül hagyja.

Megjegyezzük, hogy azért kell Bourne és C shell esetén más kódot futtatnia a fejlécnek, mert a kapott argumentumok továbbadása máshogy történik (Bourne shell esetén a hordozható megoldás a `${1+"$@"}`, C shell esetén pedig a `$argv:q`).

Van egy rövidebb, a `env` programot használó megoldás is, amivel egy futtatható szkriptfájl elkezdhető. Bash-szkript esetén például a `#! /usr/bin/env bash` sorral kezdve a szkriptet biztosak lehetünk benne, hogy a `bash` program fogja végrehajtani, bárhol is legyen a `$PATH`-on – feltéve, hogy van `env` program a `/usr/bin`-ben. (Ez például modern Linux, FreeBSD és Solaris rendszerekre igaz is.) Ugyanez a trükk Make esetén `#! /usr/bin/env make -f` shebang-sort eredményezne, ami nem működne, mert a kernel az argumentumokat nem vágja fel szóközök mentén, tehát a `make␣-f` parancsot próbálná elindítani, ahelyett, hogy a `make` parancsot futtatná az `-f` kapcsolóval. Hátrány még, hogy `gmake`-kel nem is próbálkozik.

2.4. sed: stringműveletek

A sedet az 1970-es években az igény hívta életre, hogy a shell nem kínált elég hatékony eszközöket szövegfeldolgozásra. Később ez a helyzet javult, ám a mai shellekben is kifejezetten kényelmetlen egy string belsejéből adatot kinyerni. A Unix-filozófiát [17] követve nem a shellt bővítették, hanem egy céleszközt fejlesztettek ki.

Az sed általános szövegfeldolgozási modellje a következő. A sed-szkript egy szabálylista. Minden szabály egy feltételből és a feltétel teljesülésekor végrehajtandó (esetleg összetett) utasításból áll, melyek módosíthatják az adott sort (és használhatnak változókat, melyek értéke két sor közt megmarad). A sed a bemenetét soronként olvassa, minden sorra végigmegy a szabálylistán, és az eredményül kapott sort kiírja a kimenetére. Ugyanezt a modellt követi az AWK, továbbá a Perl, a Ruby és PHP is meghívható olyan kapcsolókkal, hogy a kapott szkriptet soronként dolgozza fel.

A sedet shell-szkriptekből szokás hívni, a sed-szkriptet parancsargumentumban megadva. Például az alábbi Bourne shell-részlet a `MACI` változóban cseréli az összes barnamedvét és jegesmedvét mosómedvére:

```
MACI="‘echo "$MACI" | sed ’s/\(barna\|jeges\)\(medve\)/mosó\2/g’‘"
```

Hasonló bonyolultságú parancsokkal lehet a shell-szkriptben fájlneveket átalakítani, pl. a kiterjesztésüket megváltoztatni.

A sed leghatékonyabb eszköze, a reguláris kifejezés mind szűrésre (a feltételrészben), mind cserére (az utasításrészben), mind csere utáni feltételes elágaztatásra használható. További eszközök: karaktercsere, string másolása az aktuális sorból az egyetlen változóba és vissza, elágaztatás, sorszám-intervallumok megadása a feltételben.

A sed korlátai: nem lehet benne számolni, az aktuális soron kívül csak egy változója van, csak soronként tudja a bemenetet feldolgozni, nem támogatja a strukturált programozást (ciklus, függvény), a szintaxisa nehezen olvasható. Shell-szkriptből hívva további hátrány, hogy lassú, mert minden stringkezelő művelethez új sed folyamatot kell indítani.

Az ismert korlátok ellenére a sed azért van még ma is használatban, mert biztosan lehet számítani rá, hogy minden Unixon fent van, és a szabványos utasításai ugyanúgy működnek benne mindenhol. Így tehát a GNU AutoTools `configure` shell-szkriptje is sok helyen használ sedet. Bonyolultabb szövegfeldolgozási műveletekre a minden Unixon jelen levő eszközök közül inkább az AWK ajánlott (lásd a 2.5. szakaszban).

Bár a sed nem tud számolni, reguláris kifejezéseit vezérlési szerkezeteivel kombinálva írható benne számológép (ami például a kisiskolás módon, számjegyenként ad össze és szo-

roz). Az elvi lehetőséget illusztrálandó egy ilyen számológép készült 1996-ban, ami a `dc` fordított lengyel sorrendű, tetszőleges pontosságú számológépet emulálja (tehát a $2 + 3 \cdot 4$ kiértékeléséhez a `2 3 4*+p` parancsot kell küldeni a bemenetére). A számológép forráskódja megtalálható `dc.sed` néven a Debian `sed` csomagjában és itt [6] is.

Hordozható sed shebang ♦ A Make-nél felvetett problémát oldjuk meg sed esetén is: hogyan lehet olyan szkriptet írni, amelynek első shebang sorába nincs bedrótozva az interpreter elérési útja. A megoldás útja is a szokásos: a szkriptet `#! /bin/sh` shebang segítségével shellben futtatjuk, és a szkript eleje arra utasítja a shellt, hogy keresse meg, és indítsa el az interterpretert az adott szkripttel. A szkript eleje nem csak shellben fut, hanem az adott interpreterben is, és ott nem csinál semmit. A megoldás sed esetén (Bourne és C shellekkel is működik) az alábbi fejléc alkalmazása:

```
#! /bin/sh
\false;true;eval '(exit $?0)'&&eval '\
: f{bin;case c in esac;exec sed -n -f "$0" ${1+"$@"}'; \
exec sed -n -f "$0" $argv:q
:in} # Don't touch lines 1--6: http://www.inf.bme.hu/~pts/Magic.sed.Header
```

A fejléc shell-szkriptkénti értelmezése nem okoz nehézséget. A sedes értelmezés kicsit bonyolultabb: `\f`-től `f`-ig egy reguláris kifejezést láthatunk, utána a kapcsos zárójelek közt a `bin` utasítás elugrik a `:in`-re (kihagyva a `c...`, `e...` és `e...` utasításokat, tehát összességében a fenti fejléc sedben semmit nem csinál – és épp ez a feladata.

Az `-n` kapcsoló azt eredményezi, hogy a (módosított) sort tartalmát a sor feldolgozása után a sed ne írja ki. Ezt a hatást elérhetjük `-n` nélkül is (vagyis akkor is, ha a szkript `sed -f prog.sed`-ként van meghívva), ha egy `d` utasításból álló sort helyezünk el a szkript végére.

2.5. AWK: stringműveletek strukturált programban

Az AWK a sed sor–feltétel–utasítás modelljét követő, klasszikus unixos szövegfeldolgozó nyelv. Szintaxisa a C nyelvéhez hasonló, és ennek megfelelően képes egész és lebegőpontos számokkal számolni, tartalmaz strukturált vezérlési szerkezeteket (`if`, `while`), kezel tömböket, és lehet benne saját függvényt definiálni. Eltérés a C nyelvhez képest, hogy a string beépített típus (a kapcsolódó memóriakezelést az AWK-értelmező végzi), az értékeket egymás mellé írással (vagyis az üres operátorral) lehet konkatenálni, a tömbök asszociatívek (vagyis lehet például stringgel indexelni őket), a reguláris kifejezés nyelvi elem (tehát nem kell duplázni a backslasheket), a sor végén pontosvessző nélkül is be lehet fejezni az utasítást, a függvénydefinícónak más a szintaxisa, és hogy a program nem csak függvény- és egyéb definíciókból és deklarációkból, hanem függvénydefiníciókból és feltétel–utasítás párokból áll. Újdonság a sedhez képest, hogy a sorokat szóközök mentén mezőkre bontja (és ezután ízlés szerint hivatkozhatunk az egész sorra vagy annak mezőire), hogy mind a sorhatárt, mind a mezőhatárt jelző karakter átállítható, továbbá hogy külső fájlokól és folyamatokból érkező adatokat is tud fogadni (és írni is).

Linux rendszereken a `mawk` és `gawk` (GNU AWK) szabad implementációk elterjedtek, az utóbbi kicsit több funkciót tartalmaz, például TCP-kapcsolat nyitását.

Az AWK volt az első széles körben elterjedt unixos szkriptnyelv, melyben hosszabb, többezer soros szövegfeldolgozó szkripteket is kényelmes volt írni, és az eredményül kapott forráskód általában mások számára is érthető, átlátható lett. Eközben az egysoros szkriptek írása is kényelmes maradt. Az AWK hatással volt több modern szkriptnyelvre is.

Megfigyelhető, hogy egyes Unix-parancsok funkciói integrálódtak a szkriptnyelvekbe. Például a sed képes kiváltani a `grep`et, az AWK a `cut`ot, és némi programozással a `wc`-t is. A számolás megjelenése az AWK-ban fölöslegessé tette az `expr`et. Ezek a funkciók modern

szkriptnyelveinkben, kissé eltérő szintaxissal tovább élnek.

Az AWK fő hátrányai: bináris fájlokat nem tud kezelni (a nullás kódú karakterrel és a fájl végén a soremelés hiányával lehetnek gondok), a beépített függvények köre nem bővíthető (például külső programkönyvtárak beemelésével), valamint nagy szoftverek írásához nem ad külön támogatást.

Az AWK, a sed, a `bc`, a `dc` is azon egyszerű unixos szkriptnyelvek egyike, melyek teljes leírása egyetlen kézikönyvoldalon elfér.

Hordozható AWK shebang ♦ A szokásos feltételeket teljesítő fejléc, melyet a futtathatóvá tett szkript elejére kell írni, az alábbi:

```
#! /bin/sh
false && 0>/dev\/null; true; \
eval '(exit $?0)'&&eval '\
exec awk -f "$0" -- ${1+"$@"}'; #\
exec awk -f "$0" -- $argv:q #/
# Don't touch lines 1--6: http://www.inf.bme.hu/~pts/Magic.AWK.Header
```

A shell-szkriptként történő értelmezés annyi újdonságot tartogat, hogy a `true`-ra azért volt szükség, hogy a `$?0` Bourne shellben `00`-t adjon vissza (és ne `10`-t). A fejlécet AWK-szkriptként ilyen: `változónév && 0>/regexp/`, ami mindenképpen hamis, mert egy `/regexp/` mintaillesztés eredménye sosem negatív – tehát a fejléc AWK-ban az elvárásoknak megfelelően működik: nem csinál semmit.

3. Mai nyelvek

A tárgyalt mai nyelvek közös jellemzői:

- tanultak elődjeik és kortársaik hibáiból, és igyekszenek jobb megoldást adni;
- hatékony stringkezelést biztosítanak reguláris kifejezésekkel;
- lehetőség van bennük objektumorientáltan programozni;
- aktív fejlesztés alatt állnak, az új verziókban nyelvi újítások is várhatók;
- Unixra egyetlen implementációjuk terjedt, és sok esetben azt tekintik az adott nyelven helyes programnak, amit a referencia-implementáció elfogad (egyes nyelveknek van még Java JVM-es vagy .NET-es impelentációja is);
- Windowsra és más rendszerekre is van implementációjuk, ezek általában a Uniximplementáció portjai;
- C nyelven bővíthetőek (és ezáltal illeszthetők hozzájuk C API-jű programkönyvtárak);
- lehetővé teszik nagy szoftverek fejlesztését, és – a Lua kivételével – különböző segédeszközökkel (pl. teszt-keretrendszer, dokumentációgeneráló, debugger) támogatják is;
- a Lua kivételével, a megfelelő kiegészítőkkel támogatják mind a szöveges, mind a grafikus (Gtk, Qt vagy saját), mind a webes alkalmazások írását;
- lehetőséget biztosítanak webalkalmazás írására, és az alkalmazás perzisztens futtatására (például a webszerverrel CGI helyett FastCGI protokollon kommunikálva);
- a Lua, a Ruby és a PHP kivételével fejlett Unicode-támogatást nyújtanak;

– dinamikus kötést használnak, és a Pike kivételével nem ellenőrzik a változók típusát;

– éves vagy nagyobb gyakorisággal az adott nyelvvel foglalkozó nemzetközi konferenciákon mutatják be a legújabb fejlesztéseket.

A mai szkriptnyelvek között konvergencia figyelhető meg: bár a nyelvek szintaxisa és filozófiája közt sok a különbség, az adat- és vezérlési szerkezetek, a beépített és kiegészítő programkönyvtárak közt nagy az átfedés. A második és a harmadik szkriptnyelv elsajátítása már jóval könnyebb az elsőnél.

3.1. PHP: könnyen elsajátítható webprogramozás

A PHP eredeti célja az volt, hogy a weboldalak egy része a szerveren, dinamikusan generálódhasson, és ezzel a felhasználók élőbbé tehessék a honlapjaikat. A nyelv és az implementáció is sokat fejlődött az első elterjedt verzió, a PHP/FI óta. A PHP5 egy objektumorientált programozást is lehető tévő, több webszerverrel és több protokollon keresztül is működni képes, széles körben népszerű, könnyen elsajátítható, sok kiegészítő modullal (pl. adatbázis-illesztők) rendelkező, AJAX-os fejlesztésre is alkalmas szkriptnyelv. Népszerűségét jelzi, hogy több webszolgáltató (köztük ingyenesek is) lehetővé teszi PHP-szkriptek futtatását a szerveren.

A PHP rendkívül könnyen tanulható, azonnal használatba vehető, a bonyolultabb szolgáltatásait elég csak akkor megismerni, amikor szükség van rá. A PHP-ről szóló könyvekből Dunát lehet rekeszteni (magyar fordításban is több tucat jelent meg), csakúgy, mint a webes tutorialokból. Rendkívül hasznosak a PHP honlapján levő, függvényekre lebontott referencialeíráshoz fűzött felhasználói megjegyzések, ahol jó eséllyel megtalálható a megoldás, ha egy adott függvény nem úgy működik, ahogy szeretnénk (vagy éppen nem a megfelelő függvénnyel próbálkozunk).

A PHP-s webfejlesztést számos eszköz támogatja, köztük például az Eclipse szabad IDE és a Zend Studio fizetős IDE, de vannak egyéb eszközök, pl. dokumentáció-generálók is.

A PHP szintaxisa a C nyelvéhez hasonló, azzal a különbséggel, hogy a változónevek előtt dollárjelet kell használni, a változókat nem lehet deklarálni (és a függvényargumentumok típusát sem kell megadni). A kódot `<?php` és `?>` közé kell tenni (`include`-olt fájlok esetén is), az ezeken kívüli szövegrészeket a PHP változatlanul a kimenetre másolja. A kimenet webalkalmazás esetén a böngészőnek küldött oldal, parancssori futtatásnál pedig a szabványos kimenet.

A PHP alapból minden HTTP-kéréskor újra betölti és értelmezi a kérést kiszolgáló kódot és a kód által használt, PHP nyelvű programkönyvtárakat is. Az emiatt bekövetkező lassulás ún. *source cache* segítségével megelőzhető. Szabad *source cache* például az APC, és több kereskedelmi implementáció is van. Olyan szoftverek is letölthetők, melyek a szkript végrehajtását gyorsítják (a bájtkód optimalizálásával vagy eltérő futtatási stratégia alkalmazásával).

A PHP-t számos hátránya akadályozza abban, hogy webes felületen kiszorítsa az egyéb szkriptnyelveket (gondolunk itt főleg a Perlre; Rubyra és a Pythonra). Egyes hátrányok a nyelvi természetűek (pl. nincs *closure* a nyelvben; a lokális változók helyett a globálisakat kell külön deklarálni; függvény által visszaadott tömböt nem lehet közvetlenül indexelni; a string- és tömbkezelő műveletek nem fogalmazhatók meg olyan tömören; mint más nyelvekben; külön odafigyelést igényel, hogy egy argumentumot referencia vagy érték szerint adunk át egy függvénynek; nincs megfelelő Unicode-támogatás stb.), mások pedig a beépített függvények és a kommunikációs interfész hiányosságai (pl. túl sok a beépített függvény; kevés a beépített osztály; a függvények elnevezése és paramétersorrendje nem konzisztens, ezért nehezen megjegyezhető; a hiba esetén visszaadott érték nem konzisztens; nincs jól átgondolt, hivatalos út kivételkezelésre és adatbázis-kezelésre; nem lehet minden kivételt elkapni (pl. ha nem létező függvényt hívunk, akkor a hibaüzenetet mindenképpen megkapja weboldal

látogatója); nagy fájlok feltöltéséhez túl sok memóriát (!) használ; a feltöltés folyamata nem szkriptelhető; a komponensek nem kombinálhatók szabadon, például FTP-ről részlegesen letöltött fájlt nem lehet közvetlenül a kimenetre írni). A fentiek miatt egyes, más szkriptnyelvekhez szokott fejlesztőknek kifejezetten kényelmetlen PHP-ben dolgozniuk, mert folyamatosan azt érzik, hogy az adott feladatot a saját, kedvenc nyelvükön sokkal gyorsabban, javítgatások és idegeskedés nélkül megoldanák.

A hátrányok nagy része abból adódik, hogy a mai PHP egy hosszú, szerves fejlődés eredménye, és a fejlesztők az új szolgáltatásokat legtöbbször csak hozzárakták a meglévőekhez, és ahol csak lehet, nem változtattak, hogy a régi PHP-szkriptek továbbra is működjenek. Emiatt a mai PHP nem pont a mai igényeket szolgálja ki.

3.2. Perl: svájcibicska minden rendszeren

A Perl egy általános célú, hatékony szkriptnyelv. Eredetileg szövegfeldolgozásra készült, pontosabban szövegfájlokból információ-kinyerésre és jelentések generálására. Azóta kiegészült a rendszerhívások és a Unixokon elérhető programkönyvtárak használatának lehetőségével, és számos modul született, amely ezeket kényelmessé és hatékonnyá teszi. Megalapozottá vált a mondás, hogy Perlben mindent meg lehet csinálni, sőt – a Perl jelmondata szerint – „többféleképpen is meg lehet csinálni”. Ez nem is meglepő, hisz a Perl kitalálója nyelvész, és az emberi beszédhez hasonló programnyelvet kívánt alkotni, és a beszéd jellemzője, hogy ugyanazt a gondolatot sokféleképpen ki lehet fejezni, mindenki az izlése és a képességei szerint formálja beszédét.

A Perl szintaxisa csak távolról hasonlít a C nyelvére. A C-hez képest újdonság, hogy a változóneveket – a változó típusától függően – `$`, `@` vagy `%` jellel kell kezdeni, a változók típusát nem lehet deklarálni, a változókat nem kell deklarálni (de ajánlott), a függvénydefiníció szintaxisa más, vannak reguláris kifejezések, és stringen belül is van változó-behelyettesítés. A Perlről egy rövid, magyar nyelvű bevezető leírás olvasható [18]-ban. A Perlt sokan kritizálják amiatt, hogy nehezen olvasható a kód. A nehéz olvashatóságnak részben az az oka, hogy túl sok nyelvi elem kínálkozik, melyek nagy része írásjelekből áll, így egész hosszú kódrészletek is keletkezhetnek, melyben betűt csak elvétve találni. Jobban olvasható, a Perlhez hasonló tudású szkriptnyelv a Ruby, és nagyon jól olvasható szkriptnyelv a Python.

Egyes Perl-programozók érdekességképpen közzétesznek szándékosan elbonyolított, érthetetlenre írt, rövid programkódokat [16]. Az alábbi kód [11] például

```
not exp log srand xor s qq qx xor
s x x length uc ord and print chr
ord for qw q join use sub tied qx
xor eval xor print qq q q xor int
eval lc q m cos and print chr ord
for qw y abs ne open tied hex exp
ref y m xor scalar srand print qq
q q xor int eval lc qq y sqrt cos
and print chr ord for qw x printf
each return local x y or print qq
s s and eval q s undef or oct xor
time xor ref print chr int ord lc
foreach qw y hex alarm chdir kill
exec return y s gt sin sort split
```

csak Perl kulcsszavakból áll – és ezek a kulcsszavak együtt (írásjelek nélkül) egy működő Perl-programot alkotnak, amely kiír egy rövid üzenetet. Még szerencse, hogy „többféleképpen is meg lehet csinálni”, és a nagy perles projektek nem az elbonyolított utat választják.

Talán a Perl a legtöbb operációs rendszeren használható szkriptnyelv. Fő implementáci-

óját, a Perl 5-öt több tucat Unix és számos nem Unix rendszerre portolták, a legtöbb Linux-disztribúció alapértelmezésben feltelepíti. A Perl 6 megjelenése lassan egy évtizede húzódik, de az előzetes leírások és félkész implementációk azt mutatják, hogy a Perl 5-től lényegesen különböző, teljesen átdolgozott nyelvre és implementációra számíthatunk, Perl 5-ös szkripteket futtatni tudó, kompatibilitási móddal.

A Perlt sokféle célra használják: egysoros szkriptekkel megoldható feladatokra (a sed és az AWK nyomdokain), szövegfeldolgozásra, bináris fájlok feldolgozására, rendszeradminisztrációs műveletekre, egyszerűbb grafikus felületekhez (pl. grafikus Linux-telepítő), webalkalmazásokban, IRC-kliens szkriptelésére (pl. Irssi), szövegszerkesztő szkriptelésére (pl. Vim).

A Perlben írt, újgenerációs, modell–nézet–vezérlő alapú webalkalmazás-keretrendszerek közül a Catalystet és a Maypole-t kell megemlítenünk. Ezen keretrendszerek olyan modellt, programkönyvtárakat és futtatókörnyezetet biztosítanak, melyek jobb kódot eredményező és hatékonyabb webalkalmazás-fejlesztést tesznek lehetővé, mint a korábban megszokott módszer („elkezdem írni a CGI-szkriptet, és folyamatosan bővítem minden funkcióval, ami csak kell”). Az említett keretrendszerek célkitűzésben és tudásban a Ruby on Railshez hasonlók.

A Perl annak idején reguláris kifejezéseivel *de facto* szabványt teremtett. Később megszületett a PCRE, ami a Perl-kompatibilis reguláris kifejezések Perltől független megvalósítása egy programkönyvtárban. Használja a PHP, az Apache, a Potsfix és az Nmap is. A reguláris kifejezéssekkel történő szövegfeldolgozás hatékonysága azonban nemcsak a reguláris kifejezések szintaxisától, hanem azoknak a programozási nyelvbe való integrálásától is függ. A PCRE csak az előbbit veszi át a Perltől, az integrálás szoftverenként más és más. (A további megjegyzések nem a PCRE-re vonatkoznak.) AWK, Perl és Ruby esetén például a reguláris kifejezések a nyelv részei, és van beépített illesztés, csere és feldarabolás művelet. PHP, Python, Lua, Pike és Java esetén viszont az integráció nem teljes, ezért minden regexp-művelet kényelmetlen, nehézkes, és fölöslegesen bonyolítja a kódot ezekben a nyelvekben pl. a Perlhez képest.

Minden fejlesztő számára nyitott a lehetőség, hogy az általa készített Perl-modulokat az egész közösség számára szabadon elérhetővé tegye. Ehhez biztosít infrastruktúrát a CPAN [5], ahol a legtöbben publikálják a saját fejlesztésű moduljaikat. Az olyan nagy projektek legfrissebb kiadott verziói, mint maga a Perl 5, a Perl 6, a Catalyst és a Maypole is a CPAN-ről érhetők el. A CPAN a Perl-modulok kifogyhatatlan tárháza. Egyik fő szolgáltatása, hogy webes felületén kereshetünk a Perl és a Perl-modulok dokumentációjában, a Perl-modulok metaadataiban (pl. modulnév, szerző, disztribúció). A megtalált modul forráskódját a weben le is lehet tölteni, ám erre inkább a CPAN shell ajánlott, amely a Perl telepítésekor felkerül a rendszerre, és a CPAN-ről tudja magát és más modulokat is frissíteni. Részletes magyar nyelvű leírás a CPAN-ről [13]-on.

Szkriptek perzisztens futtatására használható a Perl-specifikus SpeedyCGI és az általános FastCGI protokoll is.

A Perlt – a rosszul megírt forráskód nehezen olvashatósága mellett – kritika szokta érni azért, mert nem kínál egy egységes utat az objektumorientált programozásra, továbbá, hogy a memóriafelszabadítás nem szemétgyűjtésen, hanem referenciaszámláláson alapul. Ezeket a hátrányokat a Perl 6 javítani fogja.

Hordozható Perl shebang ♦

```
#! /bin/sh
eval '(exit $?0)' && eval 'PERL_BADLANG=x;PATH="$PATH:.";export PERL_BADLANG\
 PATH;exec perl -x -S -- "$0" ${1+"$@"};#'if 0;eval 'setenv PERL_BADLANG x\
;setenv PATH "$PATH":.;exec perl -x -S -- "$0" $argv:q;#'.q
#!perl -w
+push@INC,'.';$0=~/(.*)/s;do(index($1,"/")<0?"./$1":$1);die$@if$@__END__+if 0
```

```
;#Don't touch/remove lines 1--7: http://www.inf.bme.hu/~pts/Magic.Perl.Header
```

Ez a megoldás kicsit hosszabb a megszokottnál. Részletes magyarázata angol nyelven [12]-ben olvasható. A `PERL_BADLANG=x` értékadásra azért volt szükség, hogy a Perl ne mutasson figyelmeztető üzenetet, ha a locale beállítás hibás a rendszeren. A Perl, ha az első, shebang sorban nem látja a `perl` stringet (mint esetünkben), akkor a programot a shebang sorban látott parancsnak adja át. Ez esetünkben nem jó, mert végtelen ciklust kapnánk (a shell meghívja a Perlt, ami visszahívja a shellt stb.). A végtelen ciklust úgy kerüli el a megoldást, hogy a Perlt a `-x` kapcsolóval hívja, melynek hatására a Perl a `#!perl`-es sortól kezdve kezdi értelmezni a szkriptet. Ekkor viszont a hibaüzenetben elromlanak az oldalszámok, ennek orvoslására a következő sorban a szkript `do`-val betölti és futtatja önmagát (majd `__END__`-del kilép, hogy ne fusson kétszer). Amikor viszont elölről kezdi futtatni a szkriptet, akkor a `do`-t tartalmazó sort végre se hajtja, mert azt el van rejtve egy `q+...+ if 0;` utasításban.

A fentieket figyelembe véve a fenti fejléc négy különböző módon képes futni (és mindegyik módnak van haszna): Bourne shell-szkriptként, C shell-szkriptként, Perl-szkriptként elölről és Perl-szkriptként a `#!perl` sortól.

Hordozható SpeedyCGI shebang ♦

```
#! /bin/sh
eval '(exit $?0)'||eval 'setenv PERL_BADLANG x;set S="-w";>/dev/null\\
which speedy&&exec speedy $S "$0" $argv:q;exec perl $S -eshift=~/\(.\*\)/"\\
;do(("\$0=\$1")=~m,^.?.?/,?"\$0:qq,./\$0,\)\;die\$@if\$@ "$0" $argv:q#'if 0;
eval 'export PERL_BADLANG=x;type -p speedy>/dev/null&&exec speedy -w \
"$0" ${1+"$@"};exec perl -w -e"shift=~/(.*)/;do((\$0=\$1)=~m,^.?.?/,?\$0
:qq(./\$0));die\$@if\$@" "$0" ${1+"$@"};:'if 0;BEGIN{delete$INC{+__FILE__};q
#! perl -w
+$0=~/(.*)/;do(($0=$1)=~m,^.?.?/,?$0:qq(./$0));die$@if$@;__END__+}
#"# Don't touch lines 1--10: http://www.inf.bme.hu/~pts/Magic.Speedy.Header
```

Lényegesen új ötletet nem tartalmaz a Perl shebanghez képest. A fő különbség az, hogy a szkriptet először a `speedy` programmal próbálja futtatni, és ha ez nem sikerül (például azért, mert a SpeedyCGI nincs telepítve), akkor a sima `perl`-t használja. Sajnos a Bourne és C shellek annyira különböznek, hogy ugyanazt a funkciót külön-külön kellett bennük megvalósítani.

3.3. Python: jól olvasható, objektumorientált szkriptek

A Python egy általános célú, objektumorientált szkriptnyelv, amely főleg nyelvi tisztaságával, egyszerűségével tűnik ki a modern szkriptnyelvek közül. Emiatt első nyelvként oktatásra is kiválóan alkalmas. A fő eltérés a többi szkriptnyelv szintaxisához képest, hogy a program struktúráját nem kapcsos zárójelekkel, hanem beljebb kezdéssel jelzi. Például 1-től 10-ig a számokat az alábbi programmal lehet kiírni:

```
i=1
while i<=10:
  print i
  i=i+1
print "Vége"
```

További fontos különbség, hogy értékadás nem szerepelhet utasítás közepén. Ez is hozzájárul a jó olvashatósághoz, mert emiatt nem lehet a fenti ciklust egyetlen sorba összevonni, mint például Perl esetén: `$I=0; while ($++I<=10) { print "$I\n" }`.

A legismertebb Python-alkalmazások a `Zope` webes tartalomkezelő, a `Zorp` magyar fejlesztésű, az alkalmazási rétegben működő tűzfal, a Gentoo Linux `Portage` nevű csomagkeze-

lője és a Trac webes projektmenedzsment eszköz (SVN-vizualizációval, hibajegykezelővel és wikivel). Egyes szoftverek, például a XEN, konfigurációs fájljait Pythonban kell elkészíteni.

Hordozható Python shebang ♦

```
#! /bin/sh
"""true"; eval '(exit $?0)'&&eval '\
exec python -- "$0" ${1+"$@"}'
exec python -- "$0" $argv:q #"""
# Don't touch lines 1--5: http://www.inf.bme.hu/~pts/Magic.Python.Header
```

Az alapötlet az, hogy Pythonban a többsoros stringkonstansokat három idézőjel határolja, míg a shell-szkriptben a két nyitó idézőjelnek semmi hatása, a harmadik zárópárja pedig rögtön a `true` szó után található. A Python figyelmen kívül hagyja az önmagában álló stringkonstansokat, tehát a fejléc a Pythonban nem csinál semmit.

Megjegyezzük, hogy a C shell egy parancsot külső programként futtat, ha a parancs neve idézőjelet vagy backslasht tartalmaz. Tehát a fenti `"""true"` C shellben a `/bin/true`-t fogja futtatni, Bourne shellben pedig a shell belső `true` parancsát. Hatásuk ugyanaz.

3.4. Ruby: a programozók kedvence

A Ruby is egy új, objektumorientált, általános célú szkriptnyelv, amely igyekszik ötvözni más szkriptnyelvek jó tulajdonságait. Szintaxisára hatással volt a Python tisztasága (de nem érzékeny a beljebb kezdésre: a blokk végének jelzésére az `end` kulcsszó használatos), és ugyaninnen vette a stringek és a listák intervallummal történő indexelését is. A Smalltalktól vette át azt az elvet, hogy minden érték objektum, még a számok is (ettől még nem pazarolja a memóriát számok tárolásakor), a Perltől és az AWK-tól örökölte a reguláris kifejezések nyelvbe jól integrált kezelését.

A jó olvashatóság érdekében a konstansokat (beleértve az osztályok neveit is) nagybetűvel kell kezdeni, a lokális változókét kisbetűvel (és külön jele van a globális változóknak és a példányváltozóknak). Ez kódoláskor nem jelent nagy terhet, viszont mások kódját sokkal gyorsabban olvashatóvá és érthetővé teszi.

Nyelvi újdonság Rubyban az *iterátor* fogalma. A Ruby-beli iterátor egy kényelmes szintaxisú *closure*, vagyis olyan utasításblokk, amely a környezetének lokális változóit használja, de nem a környezetéből kerül meghívásra. Például az alábbi Ruby-szkript az argumentumait adja össze egész számként:

```
sum = 0
ARGV.each { |arg|  sum += arg.to_i }
p sum
```

A program kapcsos zárójelbe tett része az iterátor: ez egy olyan kódrészlet, ami nem közvetlenül kerül meghívásra, hanem az `each` metódus hívja az `ARGV` tömb minden elemére egyszer. Az iterátor nem forradalmian új ötlet; a Ruby újítása abban áll, hogy kényelmes szintaxist biztosít rá, és előszeretettel használja standard programkönyvtárában.

Egyedi megoldás a Rubyban, hogy feltételben csak a `false` és a `nil` érték számít hamisnak, és például a `0` és a `0.0` és az üres string is igaz. A Ruby alkotója nem véletlenül döntött így: ezáltal tömör és kifejező kód írható az alábbihoz hasonló feladatokra: „ha a kép szélessége nem ismert, akkor próbáld meg GIF fájlként betölteni, és kinyerni a szélességet, és ha ez nem sikerül, akkor próbáld meg JPEG fájlként betölteni, és kinyerni a szélességet”. A kód a következő:

```
width ||= get_gif_width(image) || get_jpeg_width(image)
```

A megoldás akkor működik, ha a hívott függvények `nil`-t adnak vissza, ha nem ismerik a formátumot, és egy szélességet jelölő számot (akár 0-t is!), ha ismerik. Hasonlóan tömör megoldást nem lehet adni olyan nyelvekben, melyek a nullát hamisnak tekintik.

Rubyban bizonyos függvényeknek több nevük van. Például nem kell emlékeznünk, hogy egy string hosszát `length` vagy a `size` metódus adja-e vissza: bármelyiket használhatjuk.

A Ruby ötletesen nevezi el a *getter/setter* metódusokat. Például az `img.width=42` értékadás az `img` objektum `width=` metódusát, a `img.width` lekérdezés pedig a `width` metódusát hívja. Ezáltal a Ruby megelőzi, hogy a kód tele legyen `get`-tel és `set`-tel.

Az osztályokhoz bármikor felvehető új metódus. Ez az alaposztályokra is igaz, így például az egész számokra és a stringekre alkalmazott metódusok körét is bármikor bővíthetjük.

A fentihez hasonló apró ötletek együttese eredményezi, hogy Rubyban keveset kell gépelni, nem kell fölösleges dolgokon gondolkozni, és jól olvasható a kód.

A Ruby nyelv a Ruby on Rails nevű, modell–nézet–vezérlő alapú webalkalmazás-keretrendszer megjelenésével és elterjedésével került be a köztudatba. A Railsszel egyszerű és kényelmes az adatbázis alapú webes alkalmazások fejlesztése. A Rails egy olyan modellt definiál, melyben nagy webalkalmazások párhuzamos fejlesztése is lehetővé válik a kód túlzott elbonyolódása nélkül. A [2] cikkben rövid magyar leírás olvasható a Railsről. A Rails hamar a programozók kedvencévé vált, és hatással volt más programnyelvek webalkalmazás-keretrendszerének fejlődésére is (például hasonló perles keretrendszer a Catalyst). A Rails aktív fejlesztés alatt áll.

A Ruby saját szálkezeléssel rendelkezik, amit sajnos nem elég stabil, gyakran beragadnak a többszálú szkriptek.

A Ruby nem csak webfejlesztéskor remekel, hanem szövegfeldolgozásra, hálózati programozásra (pl. másolgatás több TCP/IP kapcsolat mögött), XML-feldolgozásra is kiváló, sőt: Tk- és GTK-kötése segítségével grafikus alkalmazások is fejleszthetők. Ideális továbbá gyors prototípuskészítésre. Tiszta objektumorientált szemlélete miatt programozási nyelvek tanulásakor első objektumorientált nyelvnek is jó.

Hordozható Ruby shebang ♦

```
#! /bin/sh
eval '(exit $?0)#'+/&&eval '#/ if 1<1&&'\
PATH="$PATH:.";       exec ruby -x -S -- "$0" ${1+"$@"}'
setenv PATH "$PATH":.;exec ruby -x -S -- "$0" $argv:q
#!ruby -w
#'#Don't touch/remove lines 1--6: http://www.inf.bme.hu/~pts/Magic.Ruby.Header
```

A shell és a Ruby szétválasztása az `eval`-lal kezdődő sorban történik. Ezt a sort a shell így látja: `eval '(exit $?0)#' && eval '#\`, a Ruby pedig így: `eval 'string' + /regexp/ if 1<1&&'parancs'`. A parancsot záró ' az utolsó sorban fejeződik be. A Ruby a vérehajtást az `1<1` feltételrész kiértékelésével kezdi, ami hamis, tehát nem értékeli ki se a `'parancs'`-ot (ami külső program futtatását eredményezné), se a `'string'`-et, se a `/regexp/`-et. Elértük tehát a kívánt hatást: a fejléc Rubyban semmit nem csinál.

3.5. Pike: webprogramozás C-szerű szintaxissal

Az LPC a MUD-ok[2] egy nagy részének (az LPMud-oknak) objektumorientált szkriptnyelve volt, vagyis LPC nyelven alakították ki a fejlesztők a MUD világát: a helyszíneket, az ellenségeket (és a barátságos lényeket, például kereskedőket), a felszerelési tárgyakat és a harcrendszert is. A játék logikája is LPC nyelven volt leprogramozva: a rendszer milyen pa-

[2] az 1990-es évek elejének telneten játszható szöveges szerepjátékai, ahol a fő cél az egyre erősebb ellenségeket legyőzése egyre jobb képességekkel és fegyverekkel, esetleg a többi játékos segítségével

rancsokra hogy reagál, egy páncél milyen hatással van viselője harcértékére, mikor és milyen mértékben fejlődnek a játékos képességei stb. Egy világot általában sokan fejlesztettek: nem volt ritka, hogy a játékot indító 3–5 fős csapat mellé később akár tucatnyian csatlakoztak, akik általában kevesebb jogosultsággal rendelkeztek: csak egy-egy küldetés vagy világrész kidolgozása volt a feladatuk, a játék logikáját nem módosíthatták. LPC-ben az egyes objektumokat (pl. játékosok) ki lehetett menteni, hogy a játék újraindításakor tulajdonságaik megőrződjenek.

Manapság a MUD-okat már kiszorították a 3D-s szerepjátékok, és ezáltal az LPC nyelv is vesztett jelentőségéből. A változás nem érintette azonban a Pike-ot, amely az LPC által ihletett, de azóta továbbfejlesztett, önállóan (nem MUD-on belül) használatos, általános célú, objektumorientált szkriptnyelv. Első és legfontosabb ipari alkalmazása Roxen Challenger webszerver volt, amely C-ben és Pike-ban vegyesen volt írva, és mind a szkriptelést, mind a webalkalmazások fejlesztését támogatta Pike-ban. A programról Caudium néven szabad szoftver projekt ágazott le, és a mai napig ez az egyik legnagyobb Pike-ra épülő projekt.

A Pike szintaxisa – az LPC-n keresztül – a C nyelvből származik, és megőrizte azt, hogy a változókat típussal deklarálni kell, továbbá gyűjtemények (pl. tömb, asszociatív tömb) az elemek típusát is meg kell adni. Lehetséges viszont típusmegkötés nélküli változót (`mixed`) is deklarálni. Pike-ban, hasonlóan a szkriptnyelvek többségéhez, van `string` típus, automatikus a memóriakezelés, és az összetett típusok referencia szerint kerülnek átadásra.

Hordozható Pike shebang ♦

```
#! /bin/sh
#if 0
eval '(exit $?0)'&&eval '\
exec pike -- "$0" ${1+"$@"}'
exec pike -- "$0" $argv:q
#endif // Don't touch lines 1--6: http://www.inf.bme.hu/~pts/Magic.Pike.Header
```

A fejléc azt használja ki, hogy a Pike rendelkezik preprocesszorral, így az `#if 0` és az `#endif` közti sorokat figyelmen kívül hagyja. Ugyanezzel a trükk például C nyelven is bevethető, tehát lehet olyan C forrásfájlt írni, ami shell-szkriptként futtatva lefordítja önmagát a megfelelő beállításokkal.

3.6. TCL: grafikus felületeket gyorsan

A TCL nyelv szorosan kötődik a vele együtt fejlesztett Tk grafikus eszközkészlethez. A Tk volt az 1990-es évek közepén az egyik első könnyen szkriptelhető eszközkészlet (angolul *toolkit*) X11 grafikus felületre. Később kiadták a TCL/Tk-t Windowsra és Mac OS X-re is, ezzel lehetővé vált a támogatott rendszerek között hordozható (ám nem teljesen azonos kinézetű) grafikus ablakozó alkalmazások írása. A TCL/Tk az első szkriptnyelvek egyike volt, melyek a Unicode-támogatást nyújtottak nem csak stringekben, hanem reguláris kifejezésekben és a grafikus vezérlőelemeken is. Az eseményvezérelten programozható Tk grafikus eszközkészletet kiegészítették szintén eseményvezérelt socketkezeléssel, ezáltal lehetővé vált TCP/IP kliensek és szerverek írása TCL-ben. A fenti előnyök miatt a TCL/Tk népszerűvé vált gyors prototípus készítésre, főleg grafikus felületek esetén.

A TCL nyelv egyedi szintaxissal rendelkezik, ami (a LISP-hez hasonlóan) zárójelezésen alapul, és az operátorok és a vezérlési szerkezetek teljesen hiányoznak belőle, pontosabban: ezen szolgáltatásoknak nincsen szintaktikus megfelelője. A TCL-program utasításokból áll, melyeket soremelés vagy pontosvessző választ el egymástól. Minden utasítás egy szóközökkel elválasztott lista, melynek első eleme a parancs neve, a további elemek pedig az argumentumok. Ha kapcsos zárójelbe teszünk egy listaelemet, akkor egyben marad, nem bontódik fel

szóközök mentén. Ha szögletes zárójelbe tesszük, akkor TCL-utasításként lefut, és az általa visszaadott string kerül behelyettesítésre. A szintaxisban szerepel még dollárjel után változóbehelyettesítés és stringképzés idézőjellel. A TCL fő adatstruktúrája a string – ha egy a lista stringként van megadva, akkor ugyanúgy bontandó elemeire, mint egy utasítás. Például az alábbi utasítás

```
puts [lindex {a {b {c d} e} f} 1]
```

hatására először a kétargumentumú `lindex` parancs fut le, ami az első argumentumában kapott `a {b {c d} e} f` listának választja ki az első elemét, és mivel a TCL a listaelemeket nullától számozza, az első elem a `b {c d} e` string lesz (ami értelmezhető listaként is, ha listakezelő parancsnak adjuk), de jelen esetben a `puts` parancs kapja meg, ami soremeléssel lezárva kiírja a képernyőre. (A fenti utasítás beírható `tclsh` program indítása után.) Vegyük észre, hogy mind az utasításvégrehajtás, mind az `lindex` eltüntetett egy kapcsoszárójel-párt.

Példa ciklusszervezésre:

```
set i 1;  while {$i<=10} {puts $i; incr i}
```

A fenti programrészlet 1-től 10-ig írja ki a számokat. Láthatjuk, hogy a TCL a stringeket több célra is felhasználja: számok és listák is megadhatók és felhasználhatók stringként, továbbá aritmetikai kifejezést (pl. `$i<=10`) és a kódrészletet is (pl. `puts $i; incr i`) is stringként kell átadni.

A TCL szintaxisa volt az egyik fő akadálya annak, hogy a nyelv általános célú szkriptnyelvként elterjedjen. Gyakori, hogy a TCL-programozó nem tud a feladat megoldására koncentrálni, mert a kapcsos és szögletes zárójelek pontos elhelyezésére kell figyelnie, melyek gyakran 5 vagy még nagyobb mélységben, nehezen követhetően ágyazódnak egymásba – más szkriptnyelvekben ugyanazt a feladatot zárójel nélkül, sokkal gyorsabban meg lehetett volna oldani, és az eredmény is tömörebb és érthetőbb lenne.

A TCL alapértelmezésben nem objektumorientált, de *[incr Tcl]* néven van hozzá ilyen kiegészítő.

A Tk grafikus eszközkészlet szorosan kötődik a TCL-hez. A Perl/Tk és Ruby/Tk párosítások is használnak TCL-t a belsejükben, de ezt, amennyire csak lehet, elrejtik, és az eseménykezelők megírását Perl illetve Ruby nyelven szorgalmazzák. A GTK és Qt eszközkészletek megjelenésével a Tk visszaszorult: a két új eszközkészlet nem csak szebb és intuitívabb a mai felhasználó számára, hanem jóval több beépített vezérlőelemet tartalmaz, és egyedi vezérlőelemek írása sem túl bonyolult. A modern szkriptnyelvekhez van GTK- vagy Qt-illesztés.

A nem grafikus felhasználására példa az egyik legnépszerűbb IRC-bot, az Eggdrop, amely TCL-ben szkriptelhető.

Hordozható TCL shebang ♦

```
#! /bin/sh
# Don't touch lines 1--5: http://www.inf.bme.hu/~pts/Magic.TCL.Header \
eval '(exit $?0)'&&eval '\
exec tclsh "$0" ${1+"$@"}'; #\
exec tclsh "$0" $argv:q
```

Az alkalmazott alaptrükk ismerős: a szétválasztás azt használja ki, hogy TCL-ben a megjegyzés a sor végére tett backslashsel a következő sorban folytatódik, míg shellben nem. Ha TCL/Tk grafikus alkalmazásainkhoz szeretnénk hasonló fejlécet, akkor a fentiben a `tclsh`-t cseréljük `wish -f`-re.

3.7. Lua: letisztult, beágyazott szkriptnyelv

A Lua kilóg a tárgyalt modern szkriptnyelvek sorából, mivel beágyazott környezetre tervezték. Ez egyszerűbb nyelvet, rövidebb kódot, és ezzel együtt kisebb funkcionalitást von maga után. (Összehasonlításul: a Lua 5.0 programkönyvtárának mérete Linux alatt 118 kB, a Perl 5.8.4-é 1150 kB, a Ruby 1.8.2-é pedig 748 kB.) A Luáról bevezető jellegű, magyar nyelvű leírás [3].

Fontosabb eltérések a többi tárgyalt modern szkriptnyelvtől:

– Nincs teljes, objektumorientált kivételkezelés (de azért lehet dobni és elkapni string típusú kivételeket). Hiba esetén a hívási verem elérhető.

– Az objektumorientált programozás támogatása szokatlan. Módszere [8] JavaScript prototípusos megoldásához áll közel. Az öröklődést ún. *metatáblák* segítségével valósítja meg. Az objektum metódusait tartalmazó táblához csatolt metatábla lehetővé teszi, hogy hiányzó metódus hívása esetén a Lua a metódust a szülő osztály táblájában keresse. (Metatáblákkal egyébként tetszőleges objektum tetszőleges operátora felüldefiniálható.)

– A reguláris kifejezések egyszerűbbek, kevésbé kifejezőek a Perlben és a POSIX EREben megszokottnál.

– A szintaxisa kicsit fura – például nem fogad el kifejezést utasítás helyén.

– Szokatlansága ellenére kerek, letisztult, átgondolt egészet alkot. Minden, a szkriptprogramozáshoz szükséges nyelvi elemnek megvan benne a helye.

A Lua első nagy felhasználói a játékprogramok voltak (RPG-k és lövöldözős játékok egyaránt). Azóta több szövegszerkesztő, IDE, webszerver (pl. lighttpd) és peer-to-peer kliens (pl. BCDC) is szkriptelhető Luában. Érdekesnek ígérkezik a LuaTEX, amely a TEX és a Lua tudását egyesíti. Részletes lista a Luában szkriptelhető programokról [10]-ben.

Megjegyezzük, hogy a webszerver szkriptelése nem ugyanaz, mint a webalkalmazások írása egy az szkriptnyelven a webszerver felhasználásával. A webalkalmazások általában a tartalmat generálják, ezzel szemben a webszerverhez írt szkriptek pl. a kérés kiszolgálása előtt végeznek jogosultságellenőrzést, az URL-átírást valósítják meg, és arról döntenek, hogy ez oldal változott-e a böngészőben levő cache-elt változathoz képest. Az Apache mod_perl, mod_python és mod_ruby moduljai és a Caudiumban futó Pike mind szkriptelésre, mind webalkalmazások írására alkalmasak, míg az Apache PHP-modulja inkább csak webalkalmazásokra, a lighttpd Lua-modulja pedig csak szkriptelésre jó.

A Lua megjelenése előtt a komolyabb szövegszerkesztők vagy saját szkriptnyelvet tartalmaztak (pl. Vim, Emacs, Epsilon), vagy – kiegészítőkkel – Perlben vagy Pythonban voltak szkriptelhetők (pl. Vim). Mi lehet az oka, hogy újabban egyre több szoftver választja a Luát az általános célú, „nagy” szkriptnyelvek helyett? A kis mérete mellett a telepítés egyszerűsége (főleg Windows alatt) is fontos ok. Emellett a Lua C-interfésze és forrása jobban átlátható és érthető, mint egy nála sokkal nagyobb szkriptnyelvé – ezzel csökken a nehezen reprodukálható, verziófüggő és rendszerfüggő hibák valószínűsége. Az alkalmazásfejlesztők számára tehát kisebb teher és kisebb kockázat a Luát beemelni, mint a „nagy” szkriptnyelveket.

Hivatkozások

[1] ACM Software System Award díjak.
URL http://awards.acm.org/software%5Fsystem/.

[2] Bártházi András: Ruby on Rails, 2005. augusztus.
URL http://weblabor.hu/cikkek/rubyonrails.

[3] Bevezető leírás a Lua-ról, elsősorban BCDC-seknek.
URL http://ro.4242.hu/cgi-bin/yabb2/YaBB.pl?board=bcdc_help.

[4] BusyBox: a beágyazott Linux svájcibicskája. URL http://www.busybox.net/.

[5] CPAN: a Perl-modulok kifogyhatatlan tárháza. URL http://www.cpan.org/.

[6] dc implementáció sed-ben.
URL http://sed.sourceforge.net/grabbag/scripts/dc.sed.

[7] GNU szoftverfordítási rendszer.
URL http://en.wikipedia.org/wiki/GNU_build_system.

[8] Roberto Ierusalimschy: Objektum-orientált programozás Lua-ban, 2003.
URL http://www.lua.org/pil/16.html.

[9] Brian W. Kernighan – Rob Pike: *A UNIX operációs rendszer*. 1987, Műszaki Könyvkiadó, Budapest.

[10] Luában szkriptelhető programok listája. URL http://www.lua.org/uses.html.

[11] Mike Rosulek: Csak kulcsszavakból álló Perl-szkript, 2003.
URL http://perlmonks.org/?node=Obfuscated%20Code.

[12] Szabó Péter: A magic header for starting Perl scripts. 2003. április., *The Perl Journal*, 13–15. p. URL http://i.cmpnet.com/ddj/tpj/images/tpj0304/0304tpj.pdf.

[13] Szabó Péter: Perl-modulok telepítése a CPAN-ről, 2005.
URL http://www.szszi.hu/wikidev/PerlModulokCPANr%C5%91l.

[14] Szkriptnyelvek. URL http://en.wikipedia.org/wiki/Scripting_language.

[15] Szoftverfordítás automatizálása.
URL http://en.wikipedia.org/wiki/Build_Automation.

[16] Szándékosan nehezen érthetőre írt rövid Perl-szkriptek.
URL http://perlmonks.org/?node=Obfuscated%20Code.

[17] UNIX-filozófia. URL http://en.wikipedia.org/wiki/Unix_philosophy.

[18] Verhás Péter: Perl röviden.
URL http://www.szabilinux.hu/verhas/perl/index.html.

[19] Wikipedia: szabad enciklopédia. URL http://en.wikipedia.org/.

Webalkalmazások és más linuxos trükkök iskolai környezetben

Szabó Zoltán
<szabozoltan69@gmail.com>

Kivonat

A cikk tíz, Linuxszal megvalósított ötletet vázol arra, milyen területeken tud profitálni egy oktatási intézménycsoport a szabad szoftverekből, a belső hálózatból. Tanulságos lehet, hogy nem kell hatalmas fejlesztőgárda ahhoz, hogy egy intézményben a szükséges alkalmazások idővel elkészüljenek nagyobb anyagi ráfordítás nélkül is. A cikk tartalmaz még néhány ötletet az iskolai számítógépterem (vagy akár egy internet-kávézó) linuxosításához. A mindig időszűkében levő rendszergazda lehetőségeit figyelembe véve keres megoldást, hogyan lehet szabad szoftverekkel klónozni a gépeket; egyszerű módon, „röptében” feltenni hiányzó programokat, és beszámol a Windowshoz szokott diákoknak a Linuxra átállás után felmerült igényeiről, a hiányérzetükről, és ezek átformálásáról.

Tartalomjegyzék

1. Helyzetkép

Intézményünkben, a Pannonhalmi Főapátságban két informatikatanár és egy informatikai munkatárs próbál technikai hátteret biztosítani mindahhoz, ami az épületben a számítógépekkel kapcsolatban felmerül: informatikaoktatás, más tantárgyak és az általános művelődés számítástechnikai igényei, a diákok és felnőttek irodatechnikai igényei, könyvelés, egyéb adatfeldolgozás, hálózati kommunikáció. A rendszergazda nem én vagyok, emiatt volt időm az elmúlt évek során az alábbiakra.

2. Az internetes és intranetes fejlesztések áttekintése

Tíz, az intézményünkben megvalósított ötletet vázolnék fel arra, milyen területeken tud profitálni egy intézménycsoport a szabad szoftverekből, a belső hálózatból, végső soron a Linux erejéből.

1. Tanulónyilvántartó rendszer, frissen tartható arcképcsarnokkal, naplóval, szülői értesítővel, osztálykép-rendelő modullal.
2. Fogadónapi beosztást készítő alkalmazás.
3. A számítógépterem-rend felügyeletét elősegítő „melyik gépnél ki?” feltérképező alkalmazás.
4. A Moodle e-learning rendszer és annak sok irányú hasznosítása iskolai környezetben, keresési lehetőség a helyi könyvtárak katalógusaiban.
5. Bővebb és szűkebb telefonszám-listák karbantartása, intranetes és papíralapú publikálása.
6. A belső (mobil és vonalas) telefonszámlák szétküldése és erre a programra épülően egy általános célú körlevélküldő weboldal, melynek célja, hogy mindenki olyan e-mailt kapjon, ami csak neki szól, és többek között pl. a megszólítást is testre lehessen szabni.
7. Turistacsoportok előjegyzési naptára (különböző helyszínekre, átfedő időintervallumokban).
8. Többnapos rendezvényre szóló jelentkezések regisztrációja és naprakészen tartása. Ennek révén az elfogadott jelentkezők még a határidő utolsó pillanataiban is – fájdalommentesen és egyszerűen – tudnak módosítani a kéréseiken (mikor, milyen étkezést kérnek stb.).
9. Intézményünk honlapjának [10] eZ Publish tartalomkezelő rendszerre (CMS) való átállítása, valamint MySQL adatbázis-sémából egy Perl-szkript segítségével PostgreSQL adatbázis-séma létrehozása
10. Belső, intranetes naptár (Moodle alapokon), melyben a különböző munkacsoportok a saját információikat láthatják, és természetesen az adatbevitel és -módosítás is szabályozva van.

Az ötleteket a cikkben megnevezett szabad szoftverekkel valósítottam meg. A hozzáadott egyedi fejlesztéseket kérésre bárkinek el tudom juttatni, ám ez utóbbiaknak csak a felhasználói felületüket volt időm „szépre” elkészíteni – így például nincs hozzájuk telepítő: az adatbázissémákat kézzel kell létrehozni, és a szoftvereket is kézzel kell konfigurálni.

3. A fejlesztések részletezése

3.1. Tanulónyilvántartó rendszer

Néhány éve még nem volt államilag akkreditált tanulónyilvántartó rendszer az országban (ma már van, több is, melyek közül a Taninform a legszimpatikusabb számomra, leszámítva azt az vérlázító tényt, hogy – ellentétben a kezdeti hirdetésekkel – nem megoldható az adatbázis-szerver „háznál” tartása). Akkoriban készítettem el egy webalkalmazást, mely az iskolai élet fontosabb területein segíti a csapatmunkát. A program intranetes webfelületen érhető el, és *Apache*, *PHP* és *PostgreSQL* háttérrel dolgozik (mint az összes többi efféle alkalmazás intézményünkben). Teljes névsor és arcképcsarnok áll rendelkezésre a diákokról és tanárokról, ami sokszor nagy segítség, ha név alapján kell felidézni valakit (pl. az új emberek közül). A diák-arcképek – ha valakinek nem tetszik a sajátja – webfelületen lecserélhetőek (természetesen rendszergazdai jóváhagyással, mert egyébként vicces képek jelennének meg a gyűjteményben). Ugyanez a rendszer egyszerűsíti le az osztályképek rendelését és szortírozását, majd a pénz összeszedését. Órarend, osztályozónapló, haladási napló, zárójegynapló teszi lehetővé – többféle értékelési módot megengedve – az elektronikus adminisztrációt. Negyedévkor és háromnegyed évkor értesítőt küldünk a szülőknek, ennek kiállítása nagyban leegyszerűsödött azon tanárok részére, akik használják az elektronikus naplót – akik pedig nem használják, továbbra is kitölthetik tollal a kinyomtatott lapokat. Egy ideig működött egy felhasználónévvel és jelszóval megtekinthető „webes ellenőrzőkönyv”, de egy („megjegyezhető” jelszavak választása révén kialakult) incidens kapcsán ezt egyelőre megszüntettük.

3.2. Fogadónapi beosztást készítő alkalmazás

Évente két hétvégén tart bentlakásos iskolánk tanári fogadónapot, amikor minden tanár jelen van, és várja a szülőket. A „csak-tegyük-a-papírt-az-ajtóra” módszernél igazságosabb, kevésbé ököljog-alapú és ésszerűbb beosztás elkészítése a cél. A diákok és szüleik jelentkezése webfelületen történik (a tanárokét kézzel viszem be az adatbázisba). A diákok kérései és a tanárok jelenléte alapján optimális beosztást alakít ki egy PHP program (mely az elektronikus napló információit is fel tudja használni – a diákok lakhelyét, érdemjegyeit stb. tekintetbe véve állítja fel a prioritási sorrendet). Hadd idézzek egy levélrészletet, melyet *Csanády Miklós*sal váltottam egy ezzel rokon alkalmazás kapcsán, melyet a Piarista Gimnáziumban használnak: *„A fogadónapon alapértelmezésben 17–19-ig fogadnak a tanárok. Ez kevés szokott lenni. Ha a tanár akar, vállalhat hosszabb időt, jöhet korábban és/vagy mehet később. Ezt ő maga állíthatja be a fogadási időpont előtti hetet megelőzően. A fogadást megelőző hét a szülőké, akik online bejelentkezve bejelölhetik, hogy melyik tanárhoz mikor akarnak jönni. Így egy hét alatt kialakul a beosztás. Ha a tanár időpontjai betelnek, akkor figyelmeztető e-mailt kap, hogy nem vállalna-e hosszabb fogadási időt. Az időpontját ugyanis a tanár hosszabbíthatja ilyenkor is (de már nem szűkítheti). Mind a szülő, mind a tanár beállításait tudja állítani az iskolatitkár, akitől így telefonon is lehet időpontot kérni. A fogadás napján a tanárok ajtaján nyomtatva kifüggesztjük a fogadási beosztást, az érkező szülők pedig névre szólóan megkapják egy kis cetlin. A fogadásról végül egy kis statisztika is készül igazgatónk nagy örömére.”*

3.3. A számítógépterem foglaltságát nyomon követő alkalmazás

A számítógépterem igazságosabb, ésszerűbb használatát segíti az a program, ami egy térkép alakzatban megmutatja, melyik gépre ki van bejelentkezve (arcképpel, névvel, osztállyal), és mióta. A régóta jelenlevőket eltérő háttérszín mutatja, hogy a felügyelő tanár feladata egyszerűbbé és egyértelműbbé váljon. Mivel a tanár épp szemben ül a diákok soraival, az ő számára is készül egy „térkép”, hogy ne kelljen fejben megforgatnia a látványt.

3.4. E-learning rendszer

A diákok (és tanárok) számára a nyílt forrású Moodle e-learning rendszer segíti a fájlok és egyéb információk belső használatú közzétételét, továbbá bizonyos informatikai szakkörök kifejezetten erre az alkalmazásra épülnek, itt található meg az egyes alkalmak anyaga. Nagy előnye, hogy ki-ki a saját sebességének és érdeklődésének megfelelően tud elmélyültebben vagy „fontolva” haladni. Ezen túl a *Szikla-21* webalkalmazás segítségével is kereshetnek dokumentumokat a helyi könyvtárban.

3.5. Belső telefonkönyv vezetése, elektronikus és papír alapú publikálása

Intézményünkben kétféle telefonszámlista van használatban: egy részletesebb, amely a mobiltelefonszámokat is tartalmazza, és egy rövidebb, mely egyetlen A4-es lapra elfér, és amely a fontosabb közhasznú számokat mutatja. A telefonszámok adminisztrátorának bosszantó feladata volt, hogy bizonyos változásokat/törléseket kétszer kellett adminisztrálnia. Azzal a kéréssel keresett meg, hogy készítsek egy olyan felületet, ahol egyszerre tartható nyilván a két lista, és elég legyen egyszer módosítani vagy beszúrni/törölni az információt. Azt is kérte, hogy a kinyomtatandó dokumentumok ugyanolyan igényes minőségűek legyenek, mint eddig. A feladatot szintén webfelületen keresztül, *Apache*, *PHP* és *PostgreSQL* háttérrel oldottam meg, felhasználva az *OpenOffice.org .odt* fájlformátumának jól kezelhető XML struktúráját. Ily módon évente néhányszor kinyomtatásra és sokszorosításra kerülnek a fent említett dokumentumok (szép minőségben), és maga a friss telefonszámlista folyamatosan megtekinthető az intranetes weboldalon – azaz, ha valaki frissebbet szeretne nyomtatni a meglévőnél, bármikor megteheti. Az igazsághoz hozzátartozik, hogy a program nem végez teljes és maradéktalan XML-értelmezést, és időnként előfordul, hogy szintaktikai hiba kerül az adatokba, megakadályozva ezzel az *.odt* fájl megnyitását. Ezen hibák elég ritkák, és kijavításukra megvannak a manuális eszközök (*xmlwf,* azaz xml-well-formed, és a *checkXML*), melynek eredményei aztán visszajuttathatók az adatbázisba – célravezetőbbnek találtam ezeket használni, mint sok-sok munkaórát áldozni a tökéletes program kialakítására.

3.6. Telefonszámlák szétküldése e-mailben

Egy másik intranetes oldal segítségével a belső (mobil és vonalas) telefonszámlák e-mailen történő szétküldését automatizáltam, személyre szabható megszólítással, a megfelelő csatolandó fájlok meglétének figyelésével. Itt fontos szempont volt, hogy ki-ki csak a neki szánt információt láthassa – nyilván senki nem vágyik arra, hogy más böngéssze a részletes telefonszámláját. Másrészt iskolánk szervezi (a Rákóczi Szövetséggel együtt) a *Cultura Nostra* országos történelmi versenyt. Az erre történő jelentkezési lap is webfelületen jelenik meg [2], de ami a még izgalmasabb; az értesítők szétküldését is PHP program végzi, mely az előbb említett telefonszámla-szétküldőhöz hasonló funkciójú (egész pontosan: azt használtam fel az elkészítéséhez). A cél az volt, hogy mindenki olyan e-mailt kapjon, ami csak neki szól, és a megszólítást is testre lehessen szabni, valamint az is, hogy a papíralapú levelek borítékjainak címzését se kelljen kézzel végezni, ha már egyszer rendelkezésünkre áll minden szükséges információ.

3.7. Turistacsoportok előjegyzési naptára

Az elmúlt években megnőtt az igény a turisták csoportos látogatására. A befutó kérések frappáns dokumentálására és nyilvántartására készítettem egy webfelületen használható látogatási előjegyzési naplót, amely *több helyszínen* lehetővé teszi a csoportok regisztrálását, sőt az intézménycsoport egyéb irányaiból érkező kéréseket is figyelembe tudja venni (pl. a bazilika

foglaltságára vonatkozóan), mindezt természetesen a megfelelő időintervallumok lefoglalásával. Tehát pl. ha valamikor foglalt a bazilika, az nem jelenti azt, hogy ugyanakkor ne tudna látogatócsoport indulni a kaputól – csupán azt kell elkerülni, hogy ne ugyanakkor akarjanak bazilikát nézni, amikor ott rendezvény (pl. esküvő) van.

3.8. Többnapos rendezvényekre regisztráltak nyilvántartása

Húsvétkor nagy létszámban jelentkeznek látogatók a Főapátság háromnapos rendezvényére. Az erre való jelentkezés történhet webfelületen. Minden jelentkezéshez készül egy felhasználónév–jelszó pár, ami a jelentkezéskor még nem látszik, ám abban az esetben, ha a szervezők elfogadják a jelentkezést, elég ezeket az információkat visszaküldeni. Így az elfogadott jelentkezők még a határidő utolsó pillanataiban is tudnak módosítani a konkrét kéréseiken (mikor, milyen étkezést kérnek stb.).

3.9. Tartalomkezelő rendszer a honlapon

Intézményünk honlapját [10] statikus fájlokból átvittem *eZ Publish* [4] GPL licencű tartalomkezelő rendszerre (CMS), mely az alábbi szabad szoftverekre épül: *Apache*, *PHP* és *PostgreSQL*. A kezdetben egyedüliként használható *MySQL* adatbázis-sémából egy *Perl* szkript segítségével [9] elkészítettem azt a *PostgreSQL* adatbázis-sémát, amit – némi változtatással – azóta is használ az *eZ Publish* közösség, pontosabban a közösségnek az a része, amely előnyben részesíti a *PostgreSQL*-t a *MySQL*-lel szemben.

3.10. Belső naptár munkacsoportoknak

Ezen a nyáron egy intranetes naptárat is készítettem (*Moodle* alapokon), melyben a különböző munkacsoportok a saját információikat láthatják, és természetesen az adatbevitel és -módosítás is szabályozva van. Örömteli mellékterméke lett ennek a naptárnak, hogy a közösen használt, évről évre előkerülő (és csak apró módosításoknak alávetett) fájlokat webes eszközökkel lehet ide fel- és letölteni, ily módon egy rendkívül inhomogén hálózaton is „mindenki ugyanazt láthatja”. Meg kell még említenem, hogy ez a naptármodul a *Moodle* programba a *Greek School Network* néven található meg, ilyen veretes görög nevekkel ékesen, mint *Avgoustos Tsinakos* és *Jon Papaioannou*. Igen jól érthető a kódban minden, nem okozott nehézséget belenyúlni, ahol jónak láttam valamiféle módosítást. Az „*ingyen kaptátok, ingyen is adjátok*” elvnek megfelelően aztán az általam végrehajtott módosításokat javasoltam a *Moodle* közösségnek is 2006 júliusában – nem tudom, beledolgozzák-e valamelyik következő verzióba [7].

4. A számítógépterem linuxosítása

4.1. Partíciómásolás, a rendszer sokszorosítása

Iskolákban, internet-kávézókban, teleházakban a rendszergazda örömteli feladata, hogy a gépekbe „lelket öntsön”. Nálunk nem én vagyok a rendszergazda, mint már említettem, de a linuxos partíció(k) klónozása az eddigiekben mégiscsak valahogy nekem jutott. Fontos szempont volt, hogy – ha már a Linux felé szeretném terelni a gyerekeket – maga a klónozás is szabad szoftverekkel történjen, nem pedig a *Ghost* vagy más kereskedelmi program segítségével.

A Linuxvilág részletesen ír [1] a *G4U (Ghost for Unix)* programról, és olvashatunk benne arról is, hogyan érdemes kétszáz számítógépet felkészíteni a hallgatói használatra [8]. Mindkét szerző a *G4U* és egy *ftp*-kiszolgáló segítségével oldja meg a képmásfájlok mozga-

tását; egyikük partíciónként (*uploadpart/slurppart*), a másikuk merevlemezenként (*uploaddisk/slurpdisk*). Nagy szerep jut az *sfdisk* parancsnak, mellyel parancssorból tudunk particionálni. Például a partíciós tábla elmentése:

```
sfdisk -d /dev/hda > /mnt/pendrive/p3particio
```

Az elmentett partíciós tábla visszatöltése:

```
sfdisk /dev/hda < /mnt/pendrive/p3particio
```

(Vigyázat, az a kis *-d* különbség nem elhanyagolható! Az emlékezet rostáján könnyen áthullik...)

Az ilyesféle partíció-variálások kínos következménye lehet, hogy betöltéskor csak annyit látunk, hogy GRUB (ennek oka, hogy a GRUB `stage2` fájlja máshová került a merevlemezen). Nem ritka tapasztalat, hogy a diákok azzal keresnek meg, hogy egy-két gép bizony „GRUB-os" lett... Ekkor segíthet az, ha pl. egy *Knoppix* live CD-ről bebootolva újratelepítjük a *GRUB*-ot:

```
install (hd0,0)/grub/stage1 (hd0) (hd0,0)/grub/stage2 p (hd0,0)/grub/menu.lst
```

Természetesen más merevlemez-paraméterekkel is rendelkezhet az adott számítógép, ez csak példa. Kisegíthet minket még egy *GAG* betöltésszervező is (ami pl. rajta van az *Ultimate Boot CD*-n vagy a *SystemRescueCd*-n [11]).

A magam számára azt az utat szoktam követni az operációsrendszer-sokszorosításkor, hogy egy-egy *SysRescCd*-t elindítva, a *partimage* (és a *partimaged*) programot használva hozom létre vagy töltöm vissza a kívánt partíciókat, például így:

```
partimage restore -z1 -b -o /dev/hdaX FELCSATOLT_SMB_KONYVTAR_A_KEPFAJLLAL
```

Fontos, hogy a *partimaged* nem tud akárhány kapcsolatot kezelni (talán csak 16-ot), így esetleg több képmásfájl-szolgáltató szervergépet is használnunk kell a klónozáskor.

Ha nincs lehetőség *ftp* vagy *partimaged* szervert felállítani, szükség lehet *ssh*-n keresztül átpréselni egy partíciót. Ez (tömörítéssel) így történhet a *hdaX* partíció esetén a */mnt/cel5* felcsatolása után:

```
cd /mnt/hdaX && tar -czf - ./ | ssh root@masikgep 'tar -xzpf - -C /mnt/cel5'
```

SSH-n történő másolásra hasznos lehet még az *rsync* is. Ha a fájlrendszeren POSIX-attribútumokat *(POSIX extended attributes)* vagy ACL-eket használunk, akkor győződjünk meg róla, hogy az alkalmazott módszer azokat is átviszi-e. Ha nem, akkor a másolás után ezeket külön ki kell dumpolni, a dumpot átvinni a célgépre, majd ott érvényesíteni.

4.2. A telepített rendszerek helyi javítása

Szép lehetőség a *GRUB* azon opciója, hogy megadható az, hogy legközelebb melyik menüpontnak megfelelő operációs rendszer induljon. Ez főleg akkor lehet gyümölcsöző, ha készítünk egy kifejezetten karbantartási célú partíciót (pl. a klónozáshoz szükséges *SystemRescueCD* merevlemezről futtatható változatával). Így nézhet ki a *menu.lst* fájl:

```
default saved        # fontos
timeout 10

title the old kernel # vagy: az általában futtatott rendszer
root (hd0,0)
kernel /old_kernel
savedefault
```

```
title the new kernel # vagy: a ritkán futtatott rendszer
root (hd0,0)
kernel /new_kernel
savedefault 0        # fontos
```

Amikor olyan operációs rendszert (vagy új kernelt) szeretnénk indítani, amit csak egyszer kell futtatni, akkor előzőleg adjuk ki a `grub-set-default 1` parancsot. Ekkor a másik opció töltődik be, de betöltéskor már át is állítja a legközelebb betöltendő menüpontot a másikra. Lehetőség van a *fallback* opció használatára is, ha több bizonytalan betöltendő menüpontunk is van. Lehet távoli fájlrendszeren levő kernelt is betölteni (sőt, a `menu.lst`-t is el lehet itt helyezni). Részleteket (konkrét példával a hálózati betöltésre) az [5] oldalon a GRUB-felhasználók közössége által szerkesztett wikiben [6].

Nagyon szeretem a *SysRescCd* azon tulajdonságát, hogy lehetőség van „*autorun*” módban (automatikus futtatással) indítani (pl. menet közben felcsatolt hálózati meghajtón levő) programokat az *ar_nowait* és *ar_source* indítóparaméterek megadásával, melyek aztán a helyi gépen futnak. Ráadásul a CD tartalmát be lehet tárazni a memóriába is, így egy idő után ki is lehet venni a CD-t.

Némi ügyességgel elő lehet állítani olyan *SysRescCd*-t, ami magától betöltődik és elindul, mégpedig az általunk kívánt automatikus indítási paraméterekkel [12]. Azt hiszem, ez közel áll ahhoz, amit „minden rendszergazda álmának” titulálhatunk. Vagyis: berakom a CD-t, és elmegyek ebédelni, közben a gép felhúzza magára a kívánt képmásfájl(oka)t.

4.3. A telepített rendszerek távoli javítása

A gyakorlat persze nem mindig ilyen rózsás; egyrészt nem lenne valami környezetbarát megoldás annyi CD-t gyártani, ahány gép van, másrészt célszerű egy kicsit szinkronizálni a fájlok másolását, mert egyébként a gyorstár egy idő után már nem tudja tárolni a másolandó részleteket, és a sok winchesterfej-mozgatás igen lerontja a sebességet. Magyarul: az a jó, ha egyszerre indul az összes gép klónozása. Ezt úgy lehet elérni, hogy be kell tennünk egy olyan ciklust az automatikusan induló *bash*-szkript-be, amely másodpercenként ránéz egy fájlra, hogy megvan-e. E „reteszfájl” törlésével lehet aztán egyszerre indítani a klónozást (csak aztán nem kell elfelejteni újra „megérinteni” egy *touch* paranccsal, különben legközelebb nem lesz meg ez az összeváró hatás). Szemmel láthatóan gyorsabban történik így a másolás, mintha egymástól függetlenül indítgatnám a másolást egyik gépen a másik után. Az adatcsomagok hálózati ütközése sokat lassít, de azért egy 5GB-os partíció 30 gépre való másolása 15 perc alatt lezajlik így nálunk.

Az „időnként nézzünk rá egy fájlra” módszer egy megvalósítása:

```
while ! test -f "megvagy_e"; do sleep 1; done; továbbmehetsz
```

Ez a módszer egy másik ötletet is adott. Gyakran kerülhet a rendszergazda olyan helyzetbe, hogy valamilyen program hiányzik a gépekről, vagy hogy valami nincs jól beállítva, nem megfelelő egy-egy fájl vagy könyvtár jogosultsága. Ilyenkor egy lehetőség az újraklónozás, de ez – még a fent vázolt remek lehetőségekkel együtt is – meglehetősen körülményes és időigényes, és egyáltalán nincs arányban azzal a felhasználói igénnyel, hogy „ugyan, jó lenne még nekem ez-az”. Ehelyett egy olyan megoldással szoktam élni, hogy bootoláskor elindítok (*root*-ként) egy folyamatot a háttérben, amely egy hálózati fájlra nézeget rá 20–30 másodpercenként. Ha megvan a fájl, akkor egy másik (ugyanabban a könyvtárban levő) fájlt lefuttat a rendszer (a munkaállomásokon). Azért nem egyetlen fájllal oldom ezt meg (hogy „amikor kész a parancsfájl, hadd fusson”), mert a parancsfájl szerkesztése nem mindig sikeres elsőre, és jó, ha előbb ki lehet próbálni, és az összes gépen csak javítás után indul a parancsfájl, amikor meghúzzuk a „reteszt”.

Például a parancsfájlban az

```
apt-get install -y csomagnév >/dev/null 2>&1
```

paranccsal telepíthetünk egy csomagot. Az `-y` opció arra való, hogy ne kelljen a felmerülő telepítési kérdésekre manuálisan válaszolni. Ezen felül még a semmibe irányítja a kimenetet és a hibákat. Ha a parancs nem idempotens (vagyis másodszori kiadásának is van hatása), akkor érdemes körülvenni egy teszttel: ha a `/tmp/futottam-már` fájl létezik, akkor a parancsot nem hajtjuk végre, egyébként végrehajtjuk, és létrehozzuk a `/tmp/futottam-már`-t. Ha az adott disztribúció rendszerindításkor törli a `/tmp`-t (pl. a Debian ilyen), akkor máshova hozzuk létre a fenti fájlt. A fentiek eredményeképpen amikor a szkript legközelebb lefut, és újra megvizsgálja a „futottam-már” fájlt, már ott találja, és nem kísérli meg újra a csomagtelepítést. Az igazsághoz hozzátartozik, hogy ha túl nagy a leszedendő csomag, akkor ez a módszer csődöt mondhat – egyszerűen megtagadja a repository-szerver ennyi gép egyszerre történő kapcsolódását. Ilyenkor inkább `dpkg -i`-vel kell feltenni a helyi hálózatról az előre leszedett csomago(ka)t, vagy saját tükröt létrehozni.

Mondanom sem kell, hogy ez csak Debian-alapú disztribúcióknál működik – számomra nagy visszavető tényező a SUSE Linuxszal szemben, hogy YaST-ban nem ilyen egyszerű így parancssorból ezt-azt feldobálni. (Láttam ugyan már parancssori eszközt is erre, de nem volt még időm/energiám kitapasztalni.) *(Ed)Ubuntu*val is próbálkoztam, de eddig egy adott ponton mindig csődbe jutottam a géptermi telepítéskor (pl. amikor a *Microsoft Windows* alatti betűkészleteket megpróbáltam áttenni az Ubuntu betűihez, és lefuttattam az ilyenkor szükséges parancsokat – egyszer csak nem látszott semmilyen betű a grafikus felületen).

Kedvelt pillanat a tanórák végén, hogy egy jól megfogalmazott retesszel a „*halt*” parancsot keltjük életre az egyes gépeken, mégpedig úgy, hogy az erre kiszemelt, megjutalmazandó diák maga hozhatja létre azt a fájlt, ami megindítja a lavinát – s az összes gép az ő gombnyomására kapcsol ki.

Tisztában vagyok azzal, hogy ez a „házi *bash*-démon” nem a legkifinomultabb megoldás, és rejt buktatókat. Ennél szebb utókonfigurációt tesz lehetővé pl. a konferencián előkerülő *Cfengine*, vagy Koblinger Egmont egykori *autoinstall* [3] csomagja, amely MD5-alapú összehasonlítással megoldja, hogy csak annyi információt kelljen átküldeni a hálózaton a munkaállomásokra, ami a „szükséges” és a „meglévő” merevlemez-tartalom közti különbség.

4.4. Az átállás nehézségei és örömei

Az volt a tapasztalatom a Microsoft Windowshoz szokott diákok átállásával kapcsolatban, hogy nagyon sokat jelent nekik a csillogás-villogás. Emiatt a SUSE Linux és a KDE ablakkezelő rendszer sokkal jobban tetszett a tömegeknek elsőre, mint pl. az UHU-Linux 1.2 a maga Gnome2 ablakkezelőjével. Nálam ugyanis csak Linuxot használhatnak tanórákon a diákok; bár nemcsak ez van a gépeken, és a szabad gépidejükben bizony csak a legbátrabbak töltenek be Linuxot, akik még azzal is szembe mernek nézni, hogy társaik szemében ezzel durva módon tanárkövető magatartást tanúsítanak. Az a bevallott célom ezzel a ráhatással, hogy ha már a tananyagon kívüli érdekességekkel is foglalkoznak az órán (mert, valljuk be, ez gyakran előfordul), akkor legalább Linux alatt tegyék. Ami gond és visszatartó erő szokott lenni: sokan hiányolják a natív MSN-ezés lehetőségeit; nem mindig elégedtek meg az MSN Web Messengerrel, sem a gaim-mel, mert valahogy nem szokott sikerülni a bejelentkezés e programokkal. Természetesen az *MSIE*-re kihegyezett weboldalak is ellene hatnak a Linuxnak. (Ezzel szemben az Ubuntu Linuxban pillanatok alatt feltelepíthető *Internet Explorer* némi mosolyt csalt a felhasználók arcára.) Azonban sokat nyom a latban a Linux gyors indulási sebessége. Többeket lenyűgözött pl. egy VectorLinux hihetetlen gyorsasága, ahogy egy böngészőprogram egyetlen másodperc alatt lábra áll. A jelenlegi UHU 2.0 is igen kedélyes – nem

is gondolná az avatatlan szemlélő, hogy milyen tudatformáló hatása van annak az apróságnak, hogy a betöltéskor a rajzolt kislány szomorú képet vág, ha Microsoft Windows operációs rendszer menüpontra állítjuk a GRUB-ot, ellenben mosolyog a szabad operációs rendszerek esetén. Igenis, lássa a diák, hogy van, amit csak eltűrünk, mert néha megkerülhetetlen, és van, amit szeretünk, mert felcsillan benne valami az ember igazi értékeiből.

Hivatkozások

[1] Auth Gábor: G4U – Ghost for UNIX. 2006. augusztus. 67. sz., *Linuxvilág*. URL `http://www.linuxvilag.hu/node/212`.

[2] A Cultura Nostra országos történelmi verseny honlapja. URL `http://www.bences.hu/cn/`.

[3] Kobilinger Egmont: autoinstall: Linux és Windows 9x és rendszerek automatikus partícióklónozása hálózatról induló linux segítségével, 2003. július 8. URL `ftp://ftp.uhulinux.hu/sources/autoinstall/`.

[4] eZ Publish: GPL licencű tartalomkezelő rendszer. URL `http://ez.no/`.

[5] Free Software Foundation. *GNU GRUB manual*. URL `http://www.gnu.org/software/grub/manual/html_node`.

[6] A GRUB wikije. URL `http://grub.enbug.org/`.

[7] A javított Moodle-naptármodul. URL `http://moodle.org/mod/forum/discuss.php?d=49682`.

[8] Medve Zoltán: Nagyban kicsit más – Linux az oktatási szférában. 2005. szeptember. 56. sz., *Linuxvilág*. URL `http://www.linuxvilag.hu/node/3002519`.

[9] My2Pg: Mysql-ből postgresql-be konvertáló perl-szkript. URL `http://www.bences.hu/z/My2Pg`.

[10] A Pannonhalmi Főapátság honlapja. URL `http://www.bences.hu/`.

[11] System Rescue CD: linuxos bootolható CD rendszermentéshez. URL `http://www.sysresccd.org/`.

[12] Szabó Zoltán: Linux-sokszorosítás iskolai gépekre. 2005. április. 51. sz., *Linuxvilág*. URL `http://www.linuxvilag.hu/node/3002401`.

Esettanulmány egy tagnyilvántartó programról PHP, Komodo, SchemaSpy, Proform és Moodle felhasználásával

Szabó Zoltán
<szabozoltan69@gmail.com>

Kivonat

A cikk néhány szabad szoftvert mutat be és hasonlít össze, melyek az LME által kiírt Tagnyilvántartó pályázatára készült webalkalmazás kapcsán felmerültek. A tárgyalt szoftverek: PHP, SchemaSpy, Proform, Moodle, ADODB, PostgreSQL, Xdebug, PhpPgAdmin és Komodo (ez utóbbi nem szabad szoftver). Ismertetésre kerül még néhány ötlet és trükk PHP webalkalmazás-fejlesztéshez.

Tartalomjegyzék

1. Helyzetkép

Az LME által kiírt *Tagnyilvántartó pályázat* kapcsán szeretnék beszélni arról, milyen lehetőségeket találtam – a Linux alatt ingyen futtatható, illetve a szabad szoftverek világában – egy *ADODB(PostgreSQL) + Apache/PHP* alapú alkalmazás fejlesztéséhez.

Azért fogalmaztam ilyen körülményesen a gondolatjelek között, mert sajnos az *ActiveState* cég *Komodo* programja, ami ezen írás főszereplője, nem szabad szoftver, de ingyen futtatható egy hónapig. Iskolai célra ingyen kaptam végleges licencet is – csak el kellett küldenem egy aláírt, beszkennelt űrlapot az *ActiveState*-nek. (A `$HOME/.komodo` törlésével és magának a programnak az újratelepítésével – sok más időkorlátos programmal ellentétben – nem lehet meghosszabbítani az időkorlátot.)

2. Példaképeim

A tagnyilvántartó pályázat előtt jó kedvvel, bőséggel írtam már házi használatra webalkalmazásokat, melyek közül a kezdetieken már magam is nagyokat mosolygok – letapogatható egyfajta tanulási folyamat, míg az ember megismer egy ilyen eszköztárat. Mire elérkezett a pillanat, hogy el kell kezdeni a tagnyilvántartó programot, lett néhány példaképem programozási módszerek és elvek terén. Egyik a *Moodle* e-learning rendszer [6], másik *Gyuris Gellért* webfelület-készítési módszertana [2] és az általa elkészített *ProForm* űrlapkezelési technika [1, 3].

A program kódolási stílusát tekintve Moodle kódolási irányelvét [7] próbáltam követni, mely nagyon egyszerű, nagyon plasztikus és gusztusos utat ajánl a tartalom és a forma (unalomig ismételt, de igazán szépen nem sok alkalmazásban megvalósított) szétválasztására. A Moodle alapján valósítottam meg az *ADODB* eszközöket is (hogy lehessen pl. *MySQL*-lel is használni a programot, bár magam a *PostgreSQL*-t használtam), a munkamenet-kezelést, a többnyelvűség lehetővé tételét és a súgók rendszerét. Szintén innen van a bolondbiztos (és látványos, valamint *Microsoft Windows*hoz szokott felhasználók számára kellemes varázslóólmeleg közérzetet biztosító) telepítőrendszer. Ha telepítéskor szereti valaki a webes felületen való lépegetést és világos mondatokat a háttérben zajló eseményekről, akkor csak el kell indítani a böngészőprogramot, az URL mezőbe beírva azt a helyet, ahová kibontottuk az archívumot („tarlabdát”). A telepítő mindent ellenőriz, amire szükség lesz, és a végén létrehozza (a megfelelő helyen) a tagnyilvántartó program konfigurációs fájlját. Ezek után végigvezet bennünket a program a GPL licenc elfogadásán, az adatbázistáblák kialakításán, és lehetőség lesz az online verzióinformációkba való betekintésre. Majd a nyitóoldal nyílik meg elöttünk, és szabad a pálya a program használatához. (Az igazsághoz hozzátartozik, hogy magát az adatbázist nem lehet a böngészőből létrehozni – az e célra használható szkripthármast lásd lentebb.)

A design terén pedig teljesen a *ProForm* eszközcsoportra támaszkodtam, ami igen egységes és felhasználóbarát webfelületet tesz elérhetővé. Saját definíciója szerint: „A ProForm egy több – webalkalmazásokban használatos – felületet magában foglaló eszközcsoport. Ezek a felületek rögzítettek, *szabványos* megoldások. Két méretre készültek: 800×600 (770 px) ill. 1024×768-as (990 px) méretekre.”. Alig hisz az ember a szemének, hogy mindez szabad szoftverekkel és normál böngészőkkel megvalósítható. Néhány példa: bizonyos típust elfogadó szövegmezők beíráskor történő ellenőrzése (akár *XMLHttpRequest* elemekkel és adatbázis-használattal, és esetleg egy felvillanó és eltűnő „pipa” általi visszaigazolással), hiba esetén azonnali piros bekeretezés, egér-rámozdításkor szöveges iránymutatás, egymástól függő mezők tiltási lehetősége (vagy kötelezővé tétele), intelligens dátumválasztó, automatikusan növekvő magasságú szövegbeviteli mező.

3. A Unicode kihívásai

Korábban inkább az ISO-8859-2 karakterkódolást részesítettem előnyben, mind az adatbázis, mind a `.php` és `.html` fájlok kódolása terén, viszont mivel az imént említett két példaképem az UTF-8 mellett tette le voksát (csakúgy, mint sok Linux-disztribúció friss kiadása) gondoltam, itt a pillanat, hogy én is erre induljak. Ekkor még nem tudtam, hogy a fő nehézség az lesz ebből a döntésből következően, hogy a szöveges változókra vonatkozó PHP függvények (pl. kivágás: *substr*, első karakter nagybetűsítése: *ucfirst* stb.) nem könnyen bánnak el efféle hol egy, hol két bájt hosszúságú betűkkel.

Régi cimborám, az *nedit* remekül kezeli a *ctags* fájlokat (aminek segítségével könnyen megtalálható egy-egy függvénydeklaráció, s így hihetetlenül gyorsan lehet mozogni egy újrafelhasznált forráskódban), azonban nem kezeli az UTF-8-as karakterkódolást. Bár az *nedit* honlapján azt írják, hogy hosszú távon igencsak szükség lesz erre, egyelőre ez az út nem támogatott.

Eleinte (az UTF-8 szerkesztési lehetőség miatt) a *Kate* irányába indultam. Végül mást választottam, de annak, aki a teljesen szabad szoftverek közül szeretne editort választani, a *Kate*-et ajánlot.

A *Kate*-nek hiányossága a lassú indulás, a *ctags* fájlokkal történő navigálás hiánya, a gyors alkalmazásfejlesztéshez nélkülözhetetlen szintaxisellenőrzés és az előregyártott sötét színösszeállítás hiánya. A sötét hátteret és a megfelelő színeket némi munkával sikerült beállítani. Ha van a gépünkön feltelepített PHP-értelmező, akkor egy `php -l` paranccsal látszólag pótolható a szintaxisellenőrzés, a hatékony munkavégzéshez mégis jobb egy azonnal látható hullámos aláhúzás és azonnali szövegszerű hibajelzés.

Kísérleteztem még a *Zend Studio*-val, ami szintén egy remek program, de a szememben inkább csak ezüstérmes. Lehet vele a szokásos funkciókon túl programkódot és teljesítményt elemezni, adatbáziskapcsolatot kiépíteni, és a szerveroldalon is csodákra képes. Egy kissé terjengősnek és lassúnak találtam (talán *J*ava motor miatt), valamint voltak vele gondok azzal kapcsolatban is, hogy a távoli webszerveren levő (de a helyi gépen nem jelenlevő) `.php` fájlokat nem kezelte túl ügyesen, amikor nyomkövetésre került a sor. Részletes leírás a programról [8]-ban.

4. A nyertes: Komodo

Végül is számomra az ActiveState cég *Komodo* programja vált be. Hátránya, hogy egyelőre nincs magyar nyelvű menüje (ugyanez persze igaz a *Zend Studio*-ra is), és hogy nem szabad szoftver. Előnye viszont számtalan sok van: natív ELF bináris program, szép a megjelenése (a menürendszert, súgót és magát az editort nézve is), ügyesen kezel mindenfajta fájlt, nem lehet becsapni aposztrófokkal vagy HTML/PHP részletek váltakozásával a szintaxiskiemelést. Fantasztikusan jó a dokumentációja – elegáns módon, JavaScripttel teszi lehetővé a keresést a HTML-alapú fájlokban (webszerver használata nélkül!). Igazán jól felszerelt fejlesztőkörnyezetet kapunk a kezünkbe, van benne több programnyelvhez is tanfolyam (persze PHP-hez is). Projektalapon is történhet a munka, mint ahogy azt kulturált fejlesztőkörnyezetekben megszokhattuk, de puritán editorként is működik.

Beépített szintaktikai ellenőrzés igyekszik megelőzni azt, hogy hibás kódot próbáljunk meg – időt pazarolva – futtatni. (Ez csak akkor működik, ha található a gépünkön futtatható PHP.) Ha a hiba nem pusztán szintaktikai jellegű, akkor jó arányérzék kell ahhoz, hogy megbecsüljük, melyik a gyorsabb: belenézni a forráskódba, és kideríteni, mi nem stimmel, vagy néhány „házi praktikát” bevetni (pl. `echo $változó`, `var_dump`, `print_r`), vagy éppen „hivatalosan” nyomkövetni. Komodoban igen egyszerűen lehet ez utóbbit megtenni, és komolyabb hibáknál gyakran ez a leggyorsabb. Kifejezetten ez volt a tapasztalatom olyan esetben,

amikor más verziójú PHP-környezetben próbáltam futtatni a programomat, mint ahol fejlesztettem. PHP 4-en sok minden egész másként viselkedett, mint a PHP 5-ön. Bizonyos objektumokat például PHP 4 alatt referencia szerint kell átadni (`&$obj`), míg PHP 5 alatt külön jelzés nélkül is (`$obj`) referencia szerinti átadás történik. Az ehhez hasonló kompatibilitási problémák példányainak megtalálására a nyomkövető kiválóan használható.

5. Hibakeresés PHP programokban

A nyomkövetés pontos előkészítésével kapcsolatban javaslom [10]-et. Ez részletesen leírja, hogy miként lehet előkészíteni szerver oldalon az *Apache*-ot a frissen lefordított *Xdebug* kiterjesztés (és protokoll) fogadására, mi mindent érdemes beírni a *php.ini* konfigurációs fájlba (pl. *localhost*-ot adjunk meg *remote_host*-ként, azaz távoli gépként), és a beírni a *php.ini* konfigurációs fájlba (pl. *localhost*-ot adjunk meg *remote_host*-ként, azaz távoli gépként, és a legfontosabbat: a(z esetleg távoli) webszerver-géphez egy SSH-alagút nyitásával, az `ssh -R 9000:localhost:9000` *`loginnév@gépnév`* paranccsal kapcsolódjunk. Az *Edit* | *Preferences* | *Debugger* | *Proxy* menüpont alatt a *Listen for debug connections on port* értékét ugyanarra érdemes állítani, mint amit a *php.ini*-ben adunk meg az *Xdebug* számára (alapértelmezetten ez 9000). Beállításainkat ellenőrízhetjük a *Debugmenuitemsep Listener Status* menüponttal. Ha még nem lenne bekapcsolva, kattintsunk a *Debug* menü *Listen for Remote Debugger* pontjára (azaz: *Távoli nyomkövető figyelése*).

A PHP programok nyomkövetéséhez valahogy tudatnunk kell a webszerverrel, hogy az adott pillanatban éppen mi a szándékunk (ugyanis nyilván nem akarjuk azt, hogy ezentúl csak nyomkövetési állapotot produkáljon). A leggyakoribb, és a *Komodo*-ban megvalósított*Xdebug* protokoll által is használt módszer az, hogy böngészőnkben a normál URL végére írunk egy „*?XDEBUG_SESSION_START=<idekey>*” kiegészítést, mint GET argumentumot. (Ennek a *<idekey>*-nek a *php.ini*-ben is szerepelnie kell. Ennek biztonsági szerepe is van, nehogy boldog-boldogtalan nyomkövetni tudja a programunkat. Persze a biztonságot ennél is jobban megalapozhatjuk a fent említett SSH-alagúttal). Amikor először kap arra megbízást a böngésző, hogy keresse fel a `http://www.debugme.hu/ezaz.php?XDEBUG_SESSION_START=nagggyontitkos` oldalt, akkor sütivel (*cookie*) rögzíti az *idekey* információt, és a későbbiekben már arra támaszkodik (vagyis elfelejthető a GET argumentum megadása). A sütiket is használó, munkamenet alapú nyomkövetésről bőven olvashatunk a [9] weboldalon.

A *Komodo* dokumentációja szerint az *<idekey>*-nek meg kell egyeznie a *Debug* | *Listener Status*-beli *Proxy Key* értékével, azonban azt tapasztaltam, hogy ha helyi gépen futtatjuk a webszervert, akkor ez nem kötelező, sőt, még a *php.ini*-beli *idekey*-értékkel sem kellett, hogy megegyezzen a fenti GET argumentum.

A GET argumentum célirányos használatát próbáltam egyszerűbbé tenni egy általam módosított, *Xdebug* nevű Firefox-bővítménnyel [11], amely a Gubed PHP-nyomkövető [4] módosítása.

Amikor megkezdődik a nyomkövetés a *K*omodoban, megjelenik a sárga nyilacska az első futtatott `.php` fájl első valódi kódot tartalmazó sorában. Lehet lépegetni vagy engedni a futást az első (akár feltételes) töréspontig stb. Sőt, a PHP-kódon belül is elhelyezhetünk töréspontot az `xdebug_break();` használatával (bár ez nem kezdeményez új munkamenetet, és egyébként sem szép belenyúlni a kódba).

6. Az adatbázis – Komodo, PhpPgAdmin, SchemaSpy

Az adatbázis elkészítésében is segítségemre volt a *Komodo*, hiszen a `.sql` kiterjesztésű fájlokat alapértelmezetten SQL szintaxiskiemeléssel mutatja (ez a tulajdonsága persze beállítható

másként is).

A teszteléshez nélkülözhetetlen az adatbázis feltöltése (pl. *PhpPgAdmin*-nal) és megjelenítése (*SchemaSpy*). Igazán hasznosnak bizonyult a *PhpPgAdmin* azon tulajdonsága, hogy lehet belőle exportálni az adatbázist, vagy akár annak csak adatokat tartalmazó részét (!) többféle módon: *COPY* használatával, vagy egzakt SQL parancsokat tartalmazó fájl gyártásával.

A *SchemaSpy* a táblák kapcsolódási grafikonjain túl számos más segítséget ad, kijelzi az esetleges anomáliákat is. Egy indítószkriptben érdemes tárolni a megfelelő paraméterezést, pl.:

```
java -jar schemaSpy_3.0.0.jar -t pgsql -host localhost \
  -s public -db tnydb -u postgres -o outputkonyvtar \
  -cp ./postgresql-8.0-312.jdbc3.ja
```

A háttérben egy Java-program áll, melynek futtatásakor az alábbiakat láthatjuk:

```
Using database properties:
    [schemaSpy_3.0.0.jar]/net/sourceforge/schemaspy/dbTypes/pgsql.properties
Connected to PostgreSQL - 7.4.6

Gathering schema details.....................................(6sec)
Writing/graphing summary.............(9sec)
Writing/graphing results.....................................(38sec)
Wrote relationship details of 35 tables/views to directory 'tagnyilvantarto2'
  in 54 seconds.
Start with tagnyilvantarto2/index.html
```

Valóban, az utolsó sorban látható fájl böngészőben való meghívásakor láthatóvá válnak különféle füleken a megfelelő grafikonok, ábrák, szöveges összefoglalások és részletezett kimutatások. A vizualizációs eszköz kipróbálható a [12] webcímen.

7. Héjprogramos trükkök

A program fejlesztése közben *bash*- és *psql*-szkriptekkel oldottam a törlési/adatbázis-létrehozási feladatokat. Néhány példa:

Háromlépéses adatbázis-újralétrehozás (nyilván minden adatbázis-módosításnál ezt kellett tenni):

dbujra1: `su -c 'sh ./dbujra2'; echo "Ne felejtsd schemaspy-t!"; read a`

dbujra2: `su postgres -c 'psql -d template1 -c "\i dbujra3"'`

dbujra3:

```
DROP DATABASE tnydb;
CREATE DATABASE tnydb WITH ENCODING='UNICODE' owner=tnyuser;
```

Az első alkalommal ez volt a *dbujra3*:

```
CREATE GROUP moo;
CREATE USER tnyuser PASSWORD 'tnyjelszo' CREATEDB NOCREATEUSER IN GROUP moo;
CREATE DATABASE tnydb WITH ENCODING='UNICODE' owner=tnyuser;
ALTER USER tnyuser NOCREATEDB;
```

Ezeknek az akkurátus „*csinálhasson adatbázist*” és „*ne csinálhasson adatbázist*” változtatásoknak az a jelentősége, hogy ha netán rossz kezekbe kerülne az adatbázis elérését lehetővé tevő *tnyuser* jelszó, akkor a lehető legkisebb kárt tudja tenni a támadó (már ha nem tudja a *postgres* vagy a *root* jelszót).

Ha netán volna saját jelszava a postgres felhasználónak, akkor az első két lépés egybe is vonható:

```
su postgres -c 'psql -d template1 -c "\i dbujra3"'
```

Talán meglepő a *dbujra1*-ben az a figyelmeztetés a *schemaspy*-ra vonatkozóan; az volt a tapasztalatom, hogy ha nem drótozom bele az adatbázis-újralétrehozásba ezt a figyelmeztetést, akkor elmarad a *schemaspy* által készített grafikonrendszer a valóságtól, aminek számos kínos következménye van. Ez általában is hasznos elvnek bizonyult: feledékeny ember írjon magának ellenőrzőlistát, vagy huzalozza bele a saját szkriptjeibe a megfelelő felkiáltójeleket, ahogy az egyszeri tanár is mondta: „*Fiam, ha ilyen lüke vagy, hogy mindent elfelejtesz, írj fel mindent. Én is mindent felírok.*”

Még egy egyszerű *bash*-szkript arra a célra, hogy miként készítsük el a publikum számára a `tnytest` könyvtárban állva *texttt* nevű alkönyvtárunk alapján a `tny_1.tgz` tarlabdát úgy, hogy a saját konkrét adatbázis-paramétereink (`config.php`) mégse legyenek közhírré téve, és a szimbolikus linkek helyett maguk a fájlok kerüljenek a tömörített állományba:

```
cd -P .
rm tny_1.tgz
mv tny/config.php .
cd ..
tar -cvhzf tnytest/tny_1.tgz tny
mv tnytest/config.php tnytest/tny/
```

A `cd -P` azért fontos, mert egyébként – szimbolikus linkek megléte esetén – a `cd ..` becsaphat minket. (Szeretem ezt egy *alias*-szal el is nevezni pl. `cv` névre, mert gyakran jól használható.)

Ezek után már csak egy jól célzott *scp* paranccsal fel kell tölteni a tarlabdát a megfelelő szerverre. (Amit nyilván érdemes szkriptbe tenni, nem mindig kézzel irkálni.)

Szeretnék felvillantani egy olyan *bash*-kódrészletet (az [5] cikkből), amelyet *cron*-ból naponta meghívva biztonsági mentést készíthetünk egy teljes webalkalmazásról (a programkönyvtár, az adatkönyvtár és az adatbázis kerül mentésre). A fájlok neve elé a szkript az aktuális dátumot beszúrja, ezért bármikor könnyedén visszaállíthatjuk az oldalt arra a napra, amelyikre szeretnénk. (E programrészletben a *tny* a *tagnyilvántartó* program rövidítése.)

```
#!/bin/bash
BACKUP_PATH=/mnt/backup_disc; # Vagy ahová a mentéseket szánjuk
TNY=/var/www/html/tny;        # Vagy ahol a programkönyvtár van
TNY_DATA=/var/www/tnydata;    # Vagy ahol az adatkönyvtár van
DBNAME=tnydb;                 # Adatbázis-adatok
DBUSER=tnyuser;
DBPASS=password;
BACKUPNAME_DB=backup.gz;
BACKUPNAME_TNY=tny_backup.gz;
BACKUPNAME_TNY_DATA=tnydata_backup.gz;
TODAY=‘date +%Y_%m_%d‘;
#TODAY=$(date +%Y_%m_%d)
```

Az alsó `TODAY=` variáns előnye az egymásba ágyazhatóság. A kód így folytatódik:

```
tar -cvzf $BACKUP_PATH/$TODAY_$BACKUPNAME_TNY $TNY
tar -cvzf $BACKUP_PATH/$TODAY_$BACKUPNAME_TNY_DATA $TNY_DATA

pg_dump -U $DBUSER $DBNAME | gzip - >$BACKUP_PATH/$TODAY_$BACKUPNAME_DB
#mysqldump -u$DBUSER -p$DBPASS $DBNAME |...
```

Az utolsó sor a MySQL-es megoldást mutatja.

A szkript csak akkor működik, ha nem kér jelszót a *postgresql*. Kellően szigorú beállítások esetén csak a *postgres* felhasználó tud ilyet, annak *crontab*-jába kell beszúrni az időzítést.

Még egy utolsó (csak távolról idetartozó, de szerintem igen hasznos) *bash*-trükköt hadd osszak meg, amit szintén gyakran lehet használni házi programgyártáskor. „Elrontási tartalékot” így lehet kevés gépeléssel készíteni: `cp eredetiprogramom.php{,0}`. Később így lehet felmérni a keletkezett különbséget: `diff -u eredetiprogramom.php{0,}`.

8. Képkezelők

A *Gimp* és a *GQView* is nélkülözhetetlen volt egy-egy grafika, nyomógomb, szimbólum elkészítéséhez, megnézéséhez, ezek használatára azonban most nem térek ki, csak az eszközkészlet teljessége kedvéért említem meg őket. Alapszinten minden webprogramozónak ismernie kell ezeket az eszközöket is.

Hivatkozások

[1] Gyuris Gellért: *A ProForm űrlapkezelési technika*.
URL `http://opensource.nexum.hu/proform`.

[2] Gyuris Gellért: *Webfelület-készítési módszertan*. 2004. március 22. URL `http://arcok.ujevangelizacio.hu/bubu/web.webfeluletmodszertan.html`.

[3] Gyuris Gellért: *Felhasználóbarát űrlapok, avagy mit tegyünk a Web Forms 2-ig és az XForms-ig ?* Elhangzott a *http://web.conf.hu/2006/program/i/proform* című konferencián. 2006. URL `http://web.conf.hu/2006/`.

[4] Gubed PHP-nyomkövető. URL `http://gubed.sf.net/`.

[5] Horváth Ernő: MOODLE: Egy ingyenes e-learning keretrendszer. 2004. november. 46. sz., *Linuxvilág*. URL `http://www.linuxvilag.hu/node/3002283`.

[6] A Moodle e-learning rendszer. URL `http://www.moodle.org/`.

[7] A Moodle kódolási irányelvei. URL `http://docs.moodle.org/en/Coding`.

[8] Novák Áron: Zend Studio – PHP fejlesztés egységbe zárva. 2006. október. 69. sz., *Linuxvilág*. URL `http://www.linuxvilag.hu/node/255`.

[9] PHP-kód munkamenenet-alapú nyomketése Xdebug-gal.
URL `http://www.xdebug.org/docs-debugger.php#browser_session`.

[10] Szabó Zoltán: PHP-nyomkövetés Xdebuggal. 2006. november. 70. sz., *Linuxvilág*.
URL `http://www.linuxvilag.hu/`. Megjelenés alatt.

[11] A Szabó Zoltán által módosított Xdebug Mozilla-bővítmény PHP kód nyomkövetésére.
URL `http://www.osb.hu/z/xdebug.xpi`.

[12] A tagnyilvántartó-program adatbázissémája SchemaSpy-jal vizualizálva – webes demó. URL `http://www.bences.hu/z/tagnyilvantarto2`.

Xen klaszterek

Szalai Ferenc
<szferi@niif.hu>

NIIF/AVAXIO

Kivonat

A cikk röviden bemutatja a Linux alatt egyre népszerűbb Xen virtualizációs technikát, mind a nyílt forrású, mind pedig a kereskedelmi változatát. Ez követően tárgyalásra kerül, hogy a hagyományos klaszterezési technikák (HA, Loadbalance, HPC) hogyan használhatóak virtuális szerver klaszterek kialakítására. A leírás kitér az ilyen rendszerek építésénél óhatatlanul megjelenő közös, elosztott tároló alrendszerek kérdéskörére is.

Tartalomjegyzék

1. Bevezető

A virtualizációs megoldások növekvő népszerűségének okai:

- költségcsökkentés (hardver és üzemeltetés);
- tesztkörnyezetek kialakítása;
- skálázhatóság, megbízhatóság, rendelkezésre állás és biztonság növelése;
- terület- és áramfelhasználás csökkentése.

Ezek az előnyök számos nyílt forráskódú szoftver rendszerrel kiaknázhatóak, mint például: QEmu, Linux-VServer, User Mode Linux, BSD Jail, Open Solaris Container. A hatékonyság és funkciógazdagság tekintetében azonban az utóbbi idők legnépszerűbb megoldása a Xen virtuális gép monitor (VMM).

2. Xen

A Xen maga egy hipervizor, ami a processzor szuperprivilegizált módjában futó alkalmazás. A hipervizor a virtuális gépek és a hardver közötti kapcsolatot biztosítja. A virtuális gépek (Xen terminológia szerint *domain*-ek) általában nem férnek közvetlenül a hardverhez (PCI, hálózat stb.), hanem egy ún. frontend meghajtó segítségével virtuális eszközöket használnak. A virtuális gépek kernelét a hipervizor által biztosított API használatára fel kell készíteni (ezt nevezik paravirtualizációnak), ami számos esetben egy portolási folyamatra emlékeztet. A Linux kernel esetén a Xen maga például az x86-os architektúra aleseteként jelenik meg. A virtualizálni kívánt operációs rendszer módosítására abban az esetben, ha a fizikai processzor megfelelő virtualizációt támogató technológiával (Intel: VMX, AMD: SVM) rendelkezik, nincs szükség. A Xen ilyen processzorok esetén Windows rendszerek virtualizálására is alkalmas.

A Xen az 1. ábrán látható architektúrát követi. Jól látható, hogy a frontend driver párja, az ún. backend egy Domain 0 nevű adminisztratív virtuális gépben található. A fizikai eszközöhöz csak a VMM-en keresztül lehet hozzáférni. Egy Xent futtató gépen, mivel a Xen hipervizor maga nem teljes értékű operációs rendszer, minden esetben található legalább egy ilyen adminisztratív virtuális gép, ami hagyományos operációs rendszer érzetét nyújtja az rendszergazdáknak. Továbbá ezen az adminisztratív virtuális gépen keresztül tudjuk kezelni (indítani, leállítani, konfigurálni stb.) a többi virtuális gépet is.

Xen Enterprise ♦ A Xen nyílt forrású technológia lassan az összes nagyobb Linux-terjesztés részévé válik (Debian, Red Hat, SUSE stb.). A szoftver ipari szintű támogatását a XenSource [10] nevű cég látja el világszerte, mely partneri kapcsolatban áll a legnagyobb szoftver- és hardvergyártókkal (IBM, Intel, AMD, Microsoft, Novell stb.). A XenSource cég azon túlmenően, hogy a nyílt forrású fejlesztést teljes mértékben támogatja, nemrégiben létrehozta a Xen kibővített, elsősorban az ipari igényekhez optimalizált változatát Xen Enterprise néven. A Xen Enterprise elsősorban az alábbiakban tartalmaz többet, mint a nyílt forrású változat:

- teljesítményoptimalizáció (I/O és grafika területén);
- speciális hardverek támogatása (elsősorban a virtualizációs támogató processzorok és néhány, a Linux kernelében nem található FC, iSCSI stb. kártya támogatása);
- licencelt technológiák (pl. Microsoft Virtual Disk Image formátum);

– grafikus menedzsmentfelület;

– létező fizikai gépek virtualizációját támogató eszközök;

– teljes körű támogatás üzemeltetéshez és rendszerintegrációhoz a partnercégeken keresztül.

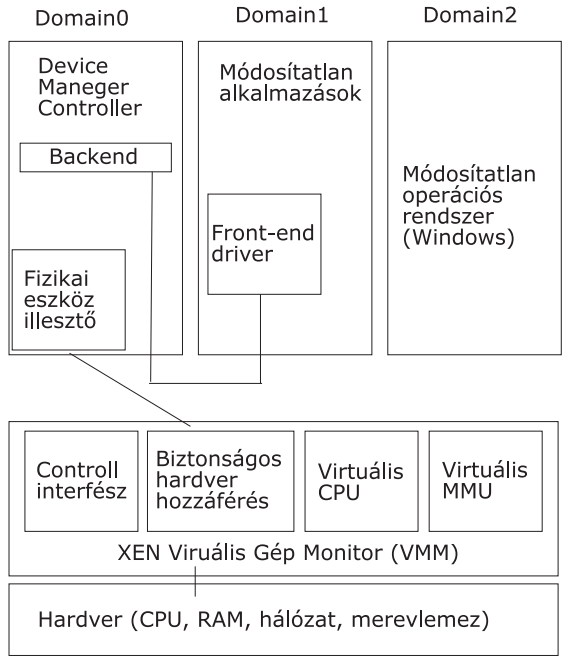
3. Xen klaszterek

A virtualizációs megoldások, így a Xen egyik legfontosabb felhasználási területe a szolgáltatások (web, FTP, levelezés, fájlszerver, adatbázis stb.) hosztolása. Ha azonban minden szolgáltatás egy fizikai gépen van, virtuális gépek formájában, joggal aggódhatunk azon, hogy a fizikai gép meghibásodása totális katasztrófát okoz az üzemeltetésben. Ebből kifolyólag a virtualizált infrastruktúra kialakításkor kézenfekvő valamilyen nagy rendelkezésre állású klaszterezési technika (HA klaszter) alkalmazása. A virtualizációs és a HA klaszterek együttesével egycsapásra biztosíthatjuk az összes szolgáltatásunk egyenszilárdságát nagy hardverberuházás nélkül.

Nagy rendelkezésere állású (HA) klaszterek ♦ A nagy rendelkezésre állású klaszterek alapötlete, hogy legalább két, közel azonos hardvert használunk. Egyiken (master) futnak a szolgáltatások, esetünkben a virtuális gépek, míg a másik melegtartalék (slave). A két gép folyamatosan figyeli a másik működését. Ha a mastergéppel vagy azon futó alkalmazásokkal, virtuális gépekkel hiba történik, a slave átveszi a master szerepét. Számos nyílt forrású projekt (pl. Hearthbeat) [6] létezik a HA funkcionalitás biztosítására Linux alatt.

A 2. ábrán az egyik legegyszerűbb Xen HA klaszter összeállítást láthatjuk. Ebben az esetben az összes virtuális gépet egyszerre mozgatjuk a Xen hosztok között és elsősorban azok fizikai meghibásodása ellen védekezünk.

Már itt is látható azonban, hogy ehhez a feladathoz elengedhetetlenül szükséges egy közös



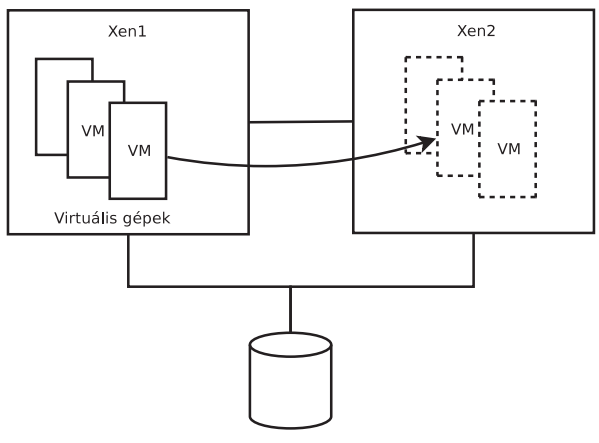
1. ábra. A Xen architektúrája

tároló rendszer. Ez lehet egy FC, iSCSI alapú külső lemeztömb, de használhatjuk az egyre népszerűbb és jóval olcsóbb ATA-over-Ethernet (AOE) [3] alapú technológiát is. A külső adattároló tömb ebben az esetben még meg is spórolható DRDB [4] használatával. Fontos megjegyezni, hogy ebben a legegyszerűbb esetben nincs szükség klaszter fájlrendszerre vagy klaszter kötetkezelésre *(volume management),* hiszen egyszerre csak egy frontendről használjuk a lemezeket. Ugyanakkor ez komoly korlátozást is jelent.

Terhelésmegosztó klaszterek ♦ Ekkor a terhelésmegosztás elsősorban a hosztgépek hatékony kihasználását jelenti. Hiszen miért hagynánk parlagon heverni a melegtartalékunkat, vagy miért ne vonhatnánk be több fizikai host gépet is a munkába, ezzel növelve a skálázhatóságot és a rendelkezésre állát? A jó hír az, hogy ez is megoldható, a rossz, az, hogy ebben az esetben komoly tervezést igényel a fizikai gépek közös tárolórendszerhez való hozzáférése. Amennyiben a virtuális gépek virtuális lemezeit LVM logikai köteteken tároljuk, úgy a CLVM [1] kiegészítéssel biztosíthatjuk, hogy a volume group-hoz több gépről is hozzáférhessenek. Ha a virtuális lemezek tárolására képfájlokat *(image file)* használunk, úgy közös fájlrendszert (Red Hat GFS [5], OCFS2 [7] stb.) kell alkalmazunk. Figyeljünk arra, hogy a Linux szoftver RAID megoldása nem támogatja a klaszterezést, így azt ebben az esetben mindenképpen a tároló tömbön kell elvégezünk.

A Xen rendszer képes a virtuális gépek futását felfüggeszteni (az `xm save` paranccsal), állapotukat elmenteni, majd egy későbbi időpontban, esetleg más helyen ebből az állapotból a virtuális gép futását folytatni (`xm restore`). Ezt a technikát alkalmazhatjuk a virtuális gépek hosztgépek közötti biztonságos mozgatására, de a virtuális gép image-ek mentésénél is nagy hasznát vehetjük. Ha azonban rendelkezünk közös fájlrendszerrel vagy klaszterizált LVM-mel, a Xen futási idejű virtuálisgép-migrációt is lehetővé tesz. Ilyenkor a virtuális gép működésében csak néhány tizedmásodperc kiesés fog bekövetkezni.

Nagyteljesítményű klaszterek ♦ Habár a Xen használatától nem várunk teljesítménynövekedést, sőt, minél több virtuális gép osztozik a fizikai gép erőforrásain, annál kevesebb teljesítmény jut az egyes virtuális gépekre, nagyteljesítményű rendszerek tervezésekor is használnak Xen-t, elsősorban a grid technológia területén. Ebben az esetben a virtuális gépekből hagyományos Beowulf típusú klasztereket építünk: egy frontend gép, sok számoló kliens csomópont, a kliensek közös (pl. NFS) fájlrendszert látnak, a feladatok futtatására klaszterütemező rendszert használnak (pl. Condor[2], SGE [9], OpenPBS[8]). Az ilyen rendszerek építésekor azt az előnyt használjuk ki, hogy a virtuális gépeken futhatnak teljesen más ope-



2. ábra. A legegyszerűbb Xen HA klaszter

rációs rendszerek, rendszerváltozatok, és számos tudományos műszaki alkalmazás igényel speciális futtatási környezetet (pl. speciálisan konfigurált libc, specifikus operációsrendszer-változat). A virtuális HPC klaszterek segítségével módunk van dinamikusan, a futtatási kérelmek érkezésekor a megfelelő futtatási környezettel rendelkező számítási klienseket elindítani a fizikai csomópontokon, és azokat virtuális klaszterekké szervezni. Mivel a Xen alkalmazásából adódó teljesítménycsökkenés kevesebb, mint 5 % általános esetben, így ez elfogadható cserébe a flexibilitását.

Hivatkozások

[1] CLVM: LVM megvalósítása klaszterben.
URL `http://sources.redhat.com/cluster/clvm/`.

[2] Condor: klaszterütemező rendszer. URL `http://www.cs.wisc.edu/condor/`.

[3] Coraid: ATA-over-Ethernet-alapú, linuxos hálózati tároló.
URL `http://www.coraid.com/`.

[4] DRDB: RAID-1 megvalósítása helyi hálózatom. URL `http://www.drbd.org/`.

[5] GFS: hálózati közös fájlrendszer.
URL `http://www.redhat.com/software/rha/gfs/`.

[6] Linux HA: nagy rendelkezésreállású Linux építésével kapcsolatos információk és szoftverek gyűjteménye. URL `http://www.linux-ha.org/`.

[7] OCFS2: hálózati közös fájlrendszer.
URL `http://oss.oracle.com/projects/ocfs2/`.

[8] OpenPBS: klaszterütemező rendszer. URL `http://www.openpbs.org/`.

[9] SGE: klaszterütemező rendszer. URL `http://www.sun.com/software/grid/`.

[10] XenSource: a Xen gyártója. URL `http://www.xensource.com/`.

LyX: egy alternatív szövegszerkesztő

Szőke Sándor
<alex@lyx.hu>

Kivonat

Mind az oktatásban, mind a mindennapi munka során szükség van hosszabb terjedelmű dokumentumok, dolgozatok, műszaki, vagy tudományos szövegek elkészítésére. Ahhoz, hogy ez minél gördülékenyebb legyen, bizonyos szövegszerkesztési feladatok megfelelő automatizálására van szükség. A szöveg írása, készítése során a szerző elsősorban arra törekszik, hogy a készülő szöveg minél világosabban tükrözze a közlendő információt. Nem jó, ha a szerző figyelmét írás közben elvonja a dokumentum külalakjának állítgatása.

A cikk a LyX szövegszerkesztőt mutatja be, amely lehetővé teszi a tartalom és a kinézet szétválasztását. Grafikus felületén látszik a több, mint 10 éves fejlesztés eredménye. A program sokféle formátumot kezel, sokfélébe tud exportálni is, több platformon elérhető, és magyar felhasználói felülettel is rendelkezik. A LyX-et mind a vele most ismerkedők, mind pedig a LaTeX-et már ismerők eredményesen tudják használni. A LyX LaTeX-alapokon nyugvó, GPL-es szabad szoftver.

Tartalomjegyzék

1. Kinek ajánlott a LyX?

Bizonyára már sokan készítettek hosszabb, esetleg komplexebb dokumentumot, például szakdolgozatot vagy műszaki dokumentációt. Biztos vagyok benne, hogy ezen dokumentum elkészítése során sok kihívást kellett megoldani; beleértve a használt program (valószínűleg az MS Word valamely verziója) által javasolt „intelligens” beállítások folyamatos javítgatásán át, a megjelenés korrigálásáig, végül a teljes dokumentum egységes formátumának ellenőrzését is. Valószínűleg volt, aki a nyomtatás során lepődött meg „A megadott stílus nem létezik!”, esetleg „A könyvjelző nem létezik!”, stb. üzenetek 10–20 oldalon (netán minden oldalon) való megjelenésétől. Volt, aki órákat töltött azzal, hogy ezeket kijavítsa, volt, aki egy másik számítógépen módosítás nélkül – *sikeresen* – nyomtatta ki munkáját!

Akik mindezeket megélték, illetve akik mindezeket el szeretnék kerülni, fellélegezhetnek, van egy olyan program, amely ezeket a problémákat orvosolni tudja; méghozzá úgy, hogy a nyomtatás minősége professzionális legyen. Tehát, akik nem szeretnének ilyen apróságokkal foglalkozni, kicsit húzódjanak közelebb, mivel számukra sok érdekes információ fog elhangzani.

Egy olyan program ismertetése következik, amely a megszokottól eltérően a tartalom elkészítésére helyezi a hangsúlyt, nem pedig a megjelenésre. A megjelenésről egy már régóta jól bevált, a nyomdai szedés technikáján alapuló program – a LaTeX – gondoskodik.

Megismerheti a program használatának előnyeit és persze a hátrányait is, az „újszerű” gondolkodás alapelveit és a munkát könnyűvé tevő trükköket, a program képességeit, valamint bővíthetőség és a testreszabhatóság eszközeit.

Végül a programot *melegen* ajánlom mindazoknak, akik szakdolgozat előtt állnak, mivel rendszerint kevés az a bizonyos pár nap az elkészítéshez. Ha íráskor elég a szövegre figyelni, ez a program nagyon jó szolgálatot tehet.

2. Javasolt felhasználási területek

A legfontosabb különbség más, grafikus felületű szövegszerkesztőkhöz (és hasonlóság a TeX-hez) képest, hogy LyX-ben a dokumentum elkészítése során a szerző a mű papíron való megjelenésével kiemelten nem foglalkozik. Azt egy utolsó fázisban, a nyomtatás előtt csiszolhatja tökéletesre. (Természetesen ez csak egy ajánlás, lehet foglalkozni menet közben is a megjelenéssel, ahogy azt egyéb programok használata közben megszokhattuk. Azonban érdemesebb azt egy professzionális programra bízni, és csak akkor avatkozni közbe, amikor feltétlenül szükséges.)

Ahhoz, hogy megfelelően ki tudjuk használni a program adottságait, nézzük meg, milyen dokumentumok elkészítéséhez ajánljuk:

- diplomamunka, disszertáció,
- dokumentáció (pl. berendezésekhez),
- feljegyzés,
- forgatókönyv (film, színdarab),
- jelentés,
- (szak)könyv (pl. szakácskönyv, műszaki kiadvány stb.),
- levél (hagyományos formátum alapján),
- matematikai kiadványok (pl. dolgozati feladatlap),
- önéletrajz,
- prezentáció,

– regény, novella, költemény,
– újság-, folyóiratcikk (nemzetközi folyóiratokhoz, sokféle formátumban).

A listából kitűnik, hogy pl. plakátok, címkék vagy hasonló rövid (1–2 oldalas) nyomtatványok elkészítéséhez nem ajánljuk. Ezeknél minden esetben pontosan akarjuk megszabni a szöveg elhelyezkedését. Az ehhez kapcsolódó technológia a WYSIWYG [11] (What You See Is What You Get), a betűszó szokásos fordítása „azt kapod, amit látsz", de lefordítható betűszóként is: Azt Látod, Amit Kapsz, HűEn – ALAKHŰ. Az ALAKHŰ szövegszerkesztőkben szerkesztés közben is a dokumentum nyomtatási képe látszik, a változtatások hatása azonnal megjelenik.

Ez LyX esetében nem így van, tehát plakátok, címkék stb. csak kicsit körülményesen szerkeszthetők vele. Az ilyen feladatokat továbbra is az OpenOffice.org vagy Word programokkal érdemes elvégezni. A fenti felsorolásba nem véletlenül került bele a prezentáció: ezek készítését ugyanis a LyX jól támogatja. Ezen előadáshoz tartozó fóliák is LyX-ben készültek.

A felsorolás további tanulmányozása után azt fogjuk tapasztalni, hogy olyan elemeket tartalmaz, amelyek szerkezete valamilyen precíz szabályszerűséget mutat (pl. egy regény fejezetekre, azokon belül bekezdésekre osztható fel, ezek előtt lehet köszönetnyilvánítás, a végén pedig egy epilógus). Ezeket a szabályszerűségeket a szerkesztőprogramok eddig még nem tudták jól kihasználni. Azt azért meg kell említenem, hogy az ilyen jellegű próbálkozások sokak életét nehezítik meg már régóta (pl. a Word automatikus stílusformázásának helytelen beállítása és használata). Egy ilyen logikai felépítés alapján definiálhatunk nagy számban különféle formátumokat, sokféle megjelenéssel (amelyek természetesen testreszabhatóak). A felsorolásban az egyes elemek jobbára csak a logikai felépítésükben különböznek, bár közülük néhány közel azonos struktúrát használ. A művek terjedelme egyes esetektől eltekintve (pl. levél) rendszerint nagy, ami a papíron való megjelenítés testreszabásakor adhat többlet munkát. Szerencsére nekünk csak akkor kell beavatkozni, amikor a háttérben működő szedőprogram rosszul dönt (szerencsére ez elég ritkán fordul elő).

3. Mi gondoskodik a szuper megjelenésről

A program működésének megértéséhez először egy átfogó képet kell kapnunk arról, hogy mi történik a dokumentum begépelésétől a papíron való megjelenéséig.

A forrásfájl(oka)t a program a LaTeX által feldolgozható formátumra (.tex) alakítja, ezáltal a már létező LaTeX segédprogramok közvetlenül használhatóak lesznek. A következő lépés a dokumentumban használt képek EPS formába történő átalakítása. Ezután szinte bármilyen LaTeX programcsomag használható a további feldolgozáshoz. Alapesetben egy eszközfüggetlen fájl – DVI [1] – fog keletkezni. A kapott DVI-fájlban az a legjobb, hogy bármilyen operációs rendszer alatt is nézzük azt meg, minden esetben azonos eredményt fogunk kapni, innen a neve is! Ezen fájlt olyan kimeneti formátumúra alakíthatjuk, amilyenre csak szeretnénk, legyen az PS, PDF vagy akár HTML (halkan jegyezem meg, hogy létezik konverter RTF [2] és ODT[3] formátumokra is). Természetesen az átalakítást a programon belülről indíthatjuk el. A köztes átalakítási fázisok automatikusan végrehajtódnak. Vannak olyan kiegészítő programcsomagok, amelyek használata során az átalakítási folyamat nagyon kötött, használatukhoz szükség van beavatkozásra. Vannak olyan LaTeX-programok is, melyekkel a `.tex` fájlból közvetlenül jön létre a célformátum, pl. pdflatex esetén PDF.

A program által nyújtott szolgáltatások nagy része láthatóan a LaTeX környezetre, valamint annak a sajátosságaira épül. Ezen függőség eredményeként a programhoz felhasználható a

[1] DVI – Device Independent File
[2] MS Worddel való megjelenítéshez
[3] nyílt formátum, például az OpenOffice.org programhoz

CTAN [7] archívban fellelhető LaTeX-kiegészítők többsége. Ezzel egy már kidolgozott és gazdag eszközkészlethez jut a felhasználó (lásd még [10]).

A munka érdemi részét a TeX-re épülő makró csomag, a LaTeX végezni el. Donald E. Knuth 1983-ban készítette el a TeX írásszedő nyelvet[4], ami a nyomtatás „trükkjeinek”, algoritmusainak a lemodellezése. Ez a nyelv alapvetően egy olyan parancskészletet tartalmaz, amellyel az egyes betűk szedésének tulajdonságai (méret, típus, hely stb.) befolyásolhatóak, valamint a nyelv kiváló makrókészítési képességekkel rendelkezik. Ennek jelentőségét felismerve Leslie Lamport 1985-ben elkészített egy olyan makrócsomagot, amellyel nagyon egyszerűvé vált egy dokumentum elkészítése. Ezzel a LaTeX térhódítása kezdetét vette. Az 1980-as évek végére, hogy még könnyebben lehessen dokumentumokat készíteni, a LaTeX-guruk elkészítették a LaTeX 2ε-t, ami (még) az aktuális változat. Ez utóbbi makrócsomag használatával nagyon kényelmessé vált a dokumentumok elkészítése, testreszabása. Ezáltal egy normál szövegszerkesztővel (pl. vi, emacs) is könnyedén készíthetjük el professzionális kinézetű dokumentumunkat. Bár ez nagyon jól hangzik, egy kicsit azért komplikált, mivel elég sok LaTeX-parancsot kell a dokumentum elkészítéséhez megjegyezni. Akit ez bővebben érdekel, olvassa el ezt remek kézikönyvet [9], valamint tömérdek információt talál a LaTeX magyarításának honlapján [3].

A LaTeX nagyon jó kimenetet szolgáltat, azonban használata körültekintést igényel. Emiatt készült hozzá jó néhány szerkesztőprogram, amelyek segítségével nem kell a rengeteg LaTeX parancsot észben tartani. A teljesség igénye nélkül néhány: Kile, TeXnicCenter, WinEdt, TeXmacs, WinShell, Winefish, VIM-LaTeX, TeXmaker, TclTexed, Euphoria, DoLaTeX, WinTeX, GNU TeXmacs, TeXShell stb. Ezen programokkal már jóval könnyebben lehet a LaTeX dokumentumunkat elkészíteni, mivel menük, eszköztárak állnak a rendelkezésünkre; esetleg a parancsok begépelése során felbukkanó menüből választhatjuk ki a megfelelőt, esetenként a szükséges struktúra beszúrásával együtt.

Az eddig elhangzottakból világosan látszik, hogy a TeX egy sima szövegfájlt vár, ami tartalmazza a szedéshez szükséges parancsokat, valamint magát a szöveget. Ezért készülhetett hozzá olyan sok szerkesztőprogram, mind máshogy próbálja segíteni a munkát. Azonban egyik sem tudta a dokumentumot úgy megjeleníteni, hogy az a logikai struktúrát tükrözze, a középpontban pedig a szöveg tartalma álljon. Ezen szempont volt a LyX elkészítésének fő mozgatórugója. A képernyőn való megjelenítés egyre jobban kezd hasonlítani a végleges dokumentumhoz, ahogy azt az *alakhű* megjelenítésnél megszokhattuk.

4. A szövegszerkesztés „újszerű” megközelítése

Ahhoz, hogy egy program szövegszerkesztésekor a szerző munkáját minél jobban megkönnyítse, bizonyos feladatokat át kell vállalnia a szerzőtől, többek között az alábbiakat:

- szöveg szedésének automatizálása,
- automatikus elválasztás,
- a szöveg egységes formátumának kezelése,
- a (kereszt)hivatkozások kezelése, automatikus frissítése,
- képek helyének meghatározása és szedése,
- bizonyos elemek számozása (képek, képletek, táblázatok stb.),
- irodalomhivatkozások nyilvántartása, kezelése,
- tárgymutató készítése,
- többnyelvű szöveg szerkesztése.

[4] 1-es verzió 1983-ban, 2.0-s verzió 1986-ban, 3.0-ás verzió 1990-ben, utolsó javítás 2002-ben

Ezen feladatok átvállalása bizonyos dolgok másképp működését eredményezi. A szerzőnek a szöveg bevitelekor nem szabad törődnie a papíron való megjelenéssel, hanem csak arra kell koncentrálnia, mi az a szöveg amit éppen bevitt, pl. fejezetcím vagy „normál szöveg”, esetleg egy felsorolás egyik eleme. Ez többek között azt is eredményezi, hogy a képernyőn nem találjuk meg a vonalzót, ahová a tabulátorokat szokás elhelyezni, mivel a szedést nem a szerző, hanem a háttérben egy program fogja elvégezni. Szerkesztés közben a képernyőn a dokumentumot a logikai struktúrájának megfelelően láthatjuk, jól áttekinthető formában.

Egy további érdekes tulajdonság az is, hogy egymás mellé nem üthetünk le két *szóközt*, vagy egymás után hiába ütjük le többször az *Enter* billentyűt, csak az első leütéskor történik valami (a *Tab* billentyű hatására egyáltalán nem történik semmi!). Ezek a furcsa jellegzetességek tükrözik azt, hogy a dokumentumban két szó között egy szóköznek van értelme, valamint két bekezdés között szintén csak egy soremelésnek van értelme. Amennyiben egy cím következik, annak formátumát át kell változtatni pl. *Szakaszcím*-re, ezt a program felismeri és ennek megfelelően fogja szedni azt. Tudni fogja, hogy nagyobb helyet kell kihagyni, mint két bekezdés között, valamint a cím betűtípusát meg kell változtatni, azt ki kell emelni valamilyen előre megadott szabályszerűség[5] alapján.

4.1. A szöveg bevitele

Egy dokumentum lényege alapvetően a tartalma. Ahhoz, hogy ez a tartalom könnyen elolvasható és feldolgozható legyen, a dokumentumot rövidebb, egymástól jól elhatárolható részekre kell bontani. Ezek az elkülönülő részek rendszerint valamilyen címet, esetleg számot kapnak, olyan betűkészlettel, amely jól elüt a szöveg törzsében alkalmazottól. A hagyományos szövegszerkesztők esetében minden szövegrész (és minden egyes betű) megjelenésének részleteit külön-külön nekünk kell megadni. Ezáltal a dokumentum nyomtatása után előfordulhat, hogy az egyes címeket vagy bekezdéseket nem a korábban használt betűtípussal vagy betűmérettel szedtük, esetleg az egyes bekezdések között kihagyott hely nem azonos stb.[*] Ezek a problémák a LyX használatával teljesen megszűnnek, mivel a szöveg bevitele alapvetően két fázisból áll:

1. szöveg beírása,
2. szöveg funkciójának meghatározása.

A kiválasztás során meghatározzuk, hogy a beírt szöveg milyen szerepet töltsön be, valamilyen cím legyen, esetleg normál szöveg. Utóbbi esetben nincs további teendőnk, míg az előbbi esetben azt is meg kell adnunk, milyen szintű a cím. A választott funkció határozza meg a szöveg szedésének a módját is, ami (többek között) lehet:

– normál szöveg – a beírt szöveg nagy része,
– felsorolás: számozással vagy jelöléssel,
– szakaszcím többféle szinten, mind tartalomjegyzékben megjelenő, mind rejtett,
– dokumentum címe, szerzőjének neve, készítési ideje,
– irodalomjegyzék, függelék, illetve egyéb strukturális elem.

[5] Ezeket a szabályokat ún. osztályokban (classes) és stílusokban (styles) lehet megadni.

[*] A WYSIWYG szövegszerkesztők karakter-, bekezdés- és egyéb stílusok segítségével próbálják szétválasztani a tartalmat a kinézettől. Ez egyszerűbb formázási feladatoknál megoldást is jelent, de jól kinéző dokumentumot (pl. függőleges kihagyás automatikus csökkentése két, egymás fölötti szakaszcím között) pusztán stílusok segítségével nem lehet létrehozni – *a szerk.*

Mivel szerkesztés közben nem kell folyamatosan beállítgatni a használt szöveg betűméretét és tulajdonságait, nem fogjuk összekeverni a szerkesztés során – ami tarthat hetekig is – a használt betűtípusokat, sőt megkönnyíti egy olyan dokumentum összeillesztését is, amelynek egyes fejezeteit különböző szerzők készítik el.

4.2. Alapvető szolgáltatások

A program a szerző munkáját elsősorban a szöveg automatikus szedésével segíti, egy komplex dokumentum elkészítésénél ez mégis kevés. A dokumentumba bizonyos esetekben plusz információkat kell bevinni, ezeket *betétekkel* oldjuk meg. Ezek a betétek a képernyőn gombokként jelennek meg, amelyekkel úgy dolgozhatunk, hogy rájuk vagy tartalmukra kattintunk. Ilyen betétekkel lehet megoldani pl. képek, címkék, láb- és széljegyzetek, irodalmi hivatkozások és TEX-parancsok beszúrását. Ezek a betétek többnyire kétféle állapottal rendelkeznek:

Zárt ♦ Ilyenkor kevés helyet foglalnak el a képernyőn, és csak a funkciójukat tudjuk megállapítani.

Nyitott ♦ Ekkor látjuk a teljes tartalmukat. A kurzorral, mint folyamatos szöveg, át tudunk menni rajtuk.

Az egyik leghasznosabb funkció, ami a LATEX környezet használatával automatikusan működik, az automatikus elválasztás. Persze ahhoz, hogy ez megfelelően működjön, meg kell adnunk a dokumentum nyelvét, valamint érdemes telepítenünk a legfrissebb magyar elválasztási modult, amit innen [3] tölthetünk le.

Sokszor kell használunk pl. utalásokat az egyik oldalon szereplő ábrára vagy egy bizonyos szakaszra. Az ilyen utalások bizonyos programoknál a nyomtatás során néha hibásak lehetnek. A LATEX rendszer használatával ezek az utalások nagyon könnyen elkészíthetőek, valamint mindig helyesek lesznek. A használatukhoz a szövegbe címkéket kell elhelyeznünk, ezek a címkék nem számokat, hanem neveket kapnak. Címkét gyakorlatilag bárhova elhelyezhetünk, de érdemes arra törekednünk, hogy az minden esetben a szövegtörzsbe kerüljön. Ezáltal az esetleges LATEX problémákat könnyen elkerülhetjük. Amikor utalni szeretnénk valahova, a „Beszúrás” menü „Kereszthivatkozás” parancsát választva egy olyan ablak nyílik meg, ahol láthatjuk a dokumentumban elhelyezett címkéinket. Mivel neveket látunk és nem számokat, könnyen ki tudjuk választani a megfelelőt. Amennyiben nem a hivatkozott szakasz számát, hanem pl. az oldalszámát szeretnénk beszúrni a dokumentumunkba, azt a dialógusablakban ki tudjuk választani.

Az utalásokhoz hasonlatos az irodalomjegyzék és ennek a hivatkozásainak használata. A LYX sokféle formátumú és megjelenésű irodalomjegyzék használatát támogatja. Használhatunk mások által szerkesztett irodalomjegyzék-adatbázisokat, amelyekből több dokumentumban is szemezgethetünk, de mi is létrehozhatunk ilyen fájlokat külső szerkesztőprogramok segítségével, végül csak egyszerűen a dokumentumban létrehozhatunk egy „Irodalomjegyzék” részt, és újsorjellel elválasztva begépelhetjük a kívánt bejegyzéseket. Ez utóbbit csak végszükség esetén alkalmazzuk, mivel az egységes megjelenési formátum sérülhet.

Vannak esetek, amikor az egyes szövegrészekhez oda nem illő magyarázatokat szeretnénk csatolni, ekkor használhatunk láb- vagy széljegyzeteket. A lábjegyzetek automatikus számozását[6] a rendszer számon tartja, az mindenkor a valóságnak fog megfelelni. Itt meg kell említenem, hogy táblázatokba kicsit nehézkes lábjegyzeteket bevinni (lásd [8]).

Mivel emberek vagyunk, előfordul, hogy hibázunk, sőt ha valaki sokat gépel, sokat hibázhat. Ezen hibák kijavításának megkönnyítése érdekében használhatunk helyesírás-ellenőrző

[6] vagy a csillagok számát

programot. Használhatjuk többek között az *ispell,* az *Aspell* és *Hunspell* programokat (magyar nyelvhez ez utóbbi ajánlott). Itt fel kell hívnom a figyelmet arra, hogy a program egyelőre nem kezeli az UTF-8-at, ezért a helyesírás-ellenőrző program beállítása Latin-2-es kódolást kell megadni.

Lehetőségünk van olyan programokat is felhasználni a dokumentáció készítéséhez, aminek a formátumát a LyX jelenleg nem támogatja. Ahhoz, hogy mindezt sikeresen meg tudjuk valósítani, készítenünk kell egy parancsfájlt, ami tartalmazza azokat a parancsokat, amelyekkel a külső program által használt formátumról a LaTeX által támogatott formátumra tudjuk alakítani a fájlunkat, ezáltal akármilyen program által szerkesztett fájlt be tudunk ágyazni a dokumentumunkba.

4.3. A képletek királya

Azt hiszem, nyugodtan jelenthetem ki – ebben sok matematikát oktató főiskolai és egyetemi tanár is meg fog erősíteni –, hogy a programban nagyon könnyen készíthetünk kiváló megjelenésű matematikai formulákat. A program egy kitűnő képletszerkesztővel rendelkezik, ami természetesen a LaTeX által nyújtott megjelenéssel párosítva verhetetlen! Elég sok dolgozati feladatlap készült már ezzel a programmal. Akinek olyan dokumentumot kell készítenie, amelyben sok képlet szerepel, próbálja ki bátran, nem fog csalódni.

Egy szöveges LaTeX szerkesztőprogramot használó a képleteket szövegként viszi be. A LyX támogatja a LaTeX-parancsok közvetlen begépelését, emiatt akik ezt a módot szeretik, azok is élvezni fogják a program használatát. A képlet módban bevitt parancsokat a program azonnal értelmezi, és a szükséges formázási műveleteket azonnal végre is hajtja, és megjeleníti a képletet (nem teljesen WYSIWYG módon). Ezáltal nagymértékben meg lehet gyorsítani a képletek bevitelét. Egy olyan képlet bevitele, mint pl. a

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i,$$

sem tart sokkal tovább, mint a kiolvasása. Persze, mindehhez egy kis gyakorlatra azért szükség van.

Más szövegszerkesztők esetén számos probléma adódhat képletek bevitelekor (ezek LyX-szel és LaTeX-hel nem jelentkeznek):

- a képletek (különösen a szóközök, a nagy zárójelek és a szummák) csúnyák;
- ha a szöveg méretét megváltoztatjuk, a képletek mérete nem változik;
- nincs meg az összes szükséges szimbólum;
- hosszú képlet közepén nem lehet eltörni a sort;
- többszáz képlet esetén a program instabillá válik;
- többszáz képlet esetén az elmentett dokumentumfájl jóval nagyobb lesz, mint ha ugyanolyan hosszú sima szöveget vinnénk be.

5. Előnyök és hátrányok

Nem maradhat ki az előnyök és hátrányok megemlítése sem. Aki ezt a programot használja, mindenekelőtt professzionális kinézetű dokumentumokat fog készíteni. A szerkesztésre fordított idő nagymértékben le fog csökkenni, mivel a formázással többé nem kell foglalkozni. Az utolsó fázisban, amikor a dokumentum elkészült, szükség lehet a megjelenítés módosítására. Ezen módosítások végrehajtása esetenként nehézkes lehet, mivel előfordulhat, hogy

a módosítás komolyabb LaTeX-ismeretet[7] igényel. Egy wikioldal is született [5], ahol sok hasznos információt találhatunk a program használatához (angol nyelven).

Időnként a formátumok közötti konverzió is okozhat problémákat. Igaz ez az RTF↔TeX és TeX→LyX irányokra is. Léteznek programok mindkét[8] átalakításhoz, de nem minden esetben képesek a fájlban található összes művelet visszaadására. Ha nem tud mindent lefordítani a program, bizonyos szövegrészek TeX ERT[9] betéttel kerülhetnek beszúrásra. Időnként azonban sikertelen az átalakítás, és nem jön létre fájl. Ilyen esetekben járható út lehet először pl. HTML formába alakítani át a fájlt, majd onnan a kívánt másik formátumba.

5.1. Testreszabhatóság

A felhasználó kényelme egy alapvető szempont volt a program kifejlesztése során, ezért szinte mindent meg lehet változtatni a felhasználói felületen, kezdve a menük elhelyezkedésével a gyorsbillentyűkön át az eszköztárakig. Felvehetünk új elemeket vagy törölhetünk a régiek közül úgy, ahogy a kedvünk tartja. Ezáltal olyan felületet alakíthatunk ki, amellyel a leghatékonyabban tudunk dolgozni. A program dokumentációjában ezzel egy teljes kézikönyv foglalkozik.

6. A fejlesztés irányvonalai

A program már régóta elérhető a következő operációs rendszerek alatt: Unix, Linux, MacOS X, Windows, CygWin. A program aktuális változata a cikk írásának pillanatában az 1.4.3-as. Kinézete az előző, 1.3.x sorozathoz képest nagyon sokat változott, azonban elég nagy változások történtek a program belső szerkezetében is. A felhasználói felületért felelős programrészeket kiemelték a kódból, és a teljes programot átstrukturálták. Hivatalos Windows-támogatás csak az 1.4.1 változat óta van; bár már korábban is használhattuk Windows alatt a Ruurd Reitsma által készített változatot, és CygWin segítségével az eredetit már elég régóta. Unicode-támogatás az 1.5-ös sorozatban, XML-alapú fájlformátum az 1.6-os sorozatban várható.

A program honlapa a [4] címen található meg, a magyar honosításé pedig a [2] címen.

Hivatkozások

[1] A LaTeX-hez kapcsolódó levelezőlisták a TUG-on, amely az usa-beli TeX-társaság. URL `http://tug.org/mailman/listinfo`.

[2] A LyX magyar nyelvű weboldala. URL `http://www.lyx.hu/`.

[3] A LaTeX magyarításának honlapja. URL `http://www.math.bme.hu/latex/`.

[4] A LyX weboldala. URL `http://www.lyx.org/`.

[5] A LyX wikije. URL `http://wiki.lyx.org/`.

[6] Angol nyelvű, TeX-hez kapcsolódó levelezőlista kezdőknek. URL `http://tug.org/mailman/listinfo/texhax`.

[7] CTAN: A TeX-nek és kiegészítőinek lelőhelye. URL `http://www.ctan.org/`.

[7] Szerencsére léteznek levelezőlisták, ahol kérdéseinkre válaszokat kaphatunk [1, 6].
[8] Van egy beépített átalakító program: *tex2lyx*, amellyel a TeX→LyX irány jól járható.
[9] Evil Red Text – Beszúrt TeX kód

[8] Hogyan kell LyX-ben táblázatba lábjegyzetet bevinni.
URL http://wiki.lyx.org/\LyX/Tables\#footintab.

[9] Tobies Oetiker és mások: *LaTeX 78 percben*. 1998, (kiadó nélkül). URL http://www.math.bme.hu/latex/dl/latex78.pdf. Fordította Németh László.

[10] TeXnik: nemhivatalos angol nyelvű TeX faq. URL http://www.texnik.de/.

[11] WYSIWYG–ALAKHŰ. Magyarul a wikipédiában.
URL http://hu.wikipedia.org/wiki/WYSIWYG.

Bevezetés a puffertúlcsordulásos hibák elleni védekezésbe

Tóth Csaba

Kivonat

A különféle programok és rendszerek sebezhetőségeinek jelentős része puffertúlcsordulásos típusú. Az exploitok többsége a memóriában különböző helyeken elhelyezkedő adatokat írja felül a megfelelően előkészített információval, és a végrehajtási útvonal módosításával később az injektált programkód hajtódik végre. A crackerek eszköztárában megtalálható exploitok mellett a vírusok és a férgek is javarészt puffertúlcsordulást *(buffer overflow)* kihasználó kódot vesznek segítségül a terjedéshez. Ha valamilyen módon ki tudnánk védeni a puffertúlcsordulási hibákat, akkor a rosszindulatú programok nagy többsége nem tudná kifejteni tevékenységét. Legjobb, ha a védelem lényegében hardveres. Minél szélesebb körben kell ezen megoldások használatát lehetővé tenni, a biztonságot szem előtt tartó gondolkodást még jobban el kell terjeszteni a felhasználók és a programozók körében egyaránt. Nagyon fontos az „alapból biztonságos” *(secure by default)* alapelv, melynek alkalmazásában az OpenBSD példaértékű.

A dolgozat bemutatja azokat a haladó technikákat, melyekkel puffertúlcsordulásos támadást lehet kivitelezni, elemzi a kódhibák keletkezésének elkerülhetetlen okait, és ismerteti a lehetséges szoftveres és hardveres védelmi megoldásokat, és betekintés nyújt a támadók ellenlépéseibe is.

Tartalomjegyzék

1. A piros vagy a kék pirula?

Sokak szerint az ő saját gépükön nincsen értékes adat, felőlük be is törhetnek a gépükre, őket nem zavarná, csak ne terhelődjön le nagyon a rendszer és ne szálljon el semelyik program. Homokba dughatjuk a fejünket, és dönthetünk úgy, hogy nem veszünk tudomást a minket fenyegető veszélyekről, elbagatellizálhatjuk őket. Ha nem szeretnénk, hogy ismeretlenek gépünkön érzékeny adatok után kutassanak, zombiként használják fel más támadásokhoz vagy spamküldésre, akkor szembe kell néznünk a fenyegetésekkel és meg kell próbálnunk védekezni ellenük.

Ha valamilyen módon ki tudnánk védeni a puffertúlcsordulási hibákat, akkor a rosszindulatú programok nagy többsége egyáltalán nem tudná kifejteni a tevékenységét, és a maradék is csak sokkal nehezebben. Hangsúlyozni kell, hogy a puffertúlcsordulásos hibák a biztonsági rések csak egy részét alkotják. Rengeteg másféle típusú rés létezik, mint például a helytelen konfigurációból származó rések, konkurenciát vagy versenyhelyzetet *(race condition)* kihasználó lyukak, SQL-beszúrásos *(SQL injection)* típusú támadások, kanonikalizáción alapuló rések. Általában egy sikeres támadáshoz több rés egyidejű jelenléte is kell, de a puffertúlcsordulási hibák általában ott szerepelnek a listán, alapvetőek a támadás sikeres véghezviteléhez.

A harc a biztonsági szakemberek és a crackerek között egy örökkévalóságig tartó verseny lesz. Sosem lesz tökéletesen biztonságos, de ugyanakkor jól használható rendszer. A használhatóság és a kényelem miatt mindig engedünk majd valamennyit a biztonságból, a crackerek viszont annyira kreatívak és ügyesek, hogy képesek lesznek az adódó lehetőségeket kihasználni. A biztonsági szakemberek pedig csak reagálnak az új betörési technikákra.

2. A puffertúlcsordulás támadás természete

Mielőtt a védelmi megoldásokra rátérnék, bemutatjuk, hogy a puffertúlcsordulásos támadásokon belül milyen lehetőségek kínálkoznak a támadásra, és erre milyen ismert haladó technikák léteznek. Ezután arra próbálunk választ találni, hogy milyen okai vannak a hibák keletkezésének.

2.1. A puffertúlcsordulásos támadás működése

A *puffer* adattárolásra szolgáló terület, amely meghatározott, véges mennyiségű adatot képes befogadni. Veremtúlcsordulásról akkor beszélünk, hogyha a program olyan adatot próbál eltárolni a veremben levő puffer területére, aminek a mérete nagyobb a puffer kapacitásánál.

Olyan utasítások okozhatnak elsősorban problémát, amelyek valamilyen forrásadatot egy célterületre másolnak, de nem ellenőrzik a méretüket (pl. strcpy(), memcpy()). A másolás folyamán mindaddig nincs baj, ameddig nem haladtuk meg a puffer méretét. Szépen feltöltjük a kívánt adattal. Azonban a művelet folytatódhat, és további adatokkal elkezdhetjük felülírni a puffer után található szomszédos memória területeket. Ezzel felülírunk adatokat, és valószínűleg a programvégrehajtás útvonalát és a végrehajtott utasításokat is megváltoztatjuk. Egy puffertúlcsordulás kihasználása [13] lehetővé teszi egy támadó számára, hogy általa szolgáltatott kódot injektáljon a végrehajtási útvonalba (ez a kód részben a puffer mögé kerül). Ez a kód lehetővé teheti, hogy rendszerszintű jogokat szerezzen, ezáltal jogosulatlan hozzáférést biztosítson a rosszindulatú crackerek számára, és lehetővé tegye a kártékony kód további másolását és szaporítását.

A puffertúlcsordulásokat általában több kategóriába sorolhatjuk mind a kihasználhatóság nehézsége, mind a technológia fejlődése alapján. Noha nincs formális definíció, megegyezés alapján jelenleg három kategóriába osztják azokat [9, 16]. Az első generációs puffertúlcsordulások a veremterület klasszikus felülírását foglalják magukba. A második generációs

technikák a heapet vagy a függvénymutatókat is érintik, ezen kívül az úgynevezett „eggyel elszámolt” *(off-by-one)* sebezhetőségek (lásd később) is ide tartoznak. Harmadik generációs túlcsordulások a printf() formátumstring támadások és a heap struktúra nyilvántartásának sebezhetőségei.

2.2. Első generációs támadások

A klasszikus veremtúlcsordulásos sebezhetőségnél egy adott függvényen belül található egy fix méretű lokális tömb. A végrehajtás során ez a tömb a veremben helyezkedik el. Később egy óvatlan utasítás határellenőrzés nélkül másol adatot a tömbbe. A túlcsordulás során felülíródnak a puffer után elhelyezkedő változók értékei, ezek után a függvényhívás előtt elmentett verem keretmutató *(stack frame pointer),* majd az elmentett visszatérési cím is felülíródhat. A támadó cselesen úgy állítja össze a felülíró adatokat, hogy a visszatérési cím helyére olyan memóriacím kerüljön, hogy a függvényből való visszatéréskor a program vezérlés ennek hatására szintén a rosszindulatú adatok között elhelyezett kódra kerüljön. A pontos memóriacímeket a program statikus elemzésével (melyik függvény hol található) és visszafejtéssel (dissasembly) gyerekjáték meghatározni.

Néha a támadás azt igényelheti, hogy a cracker pontosan igazítsa *(align)* az adatokat (hogy a vezérlés egyből pontosan a bejuttatni kívánt shellkódra kerüljön), azonban ez a probléma a *NOP slide*-nak nevezett technikával minimalizálható. Ekkor a puffer az elmentett EBP és EIP címek mellett rengeteg NOP (No OPeration, x86 architektúrákon 0x90 a gépi kódja) utasítást tartalmaz, és a csúszda legvégén a rosszindulatú kód várja a végrehajtást. Ekkor nem kell pontosan tudnunk azt, hogy hol kezdődik az injektált kód, elég, hogyha a *NOP slide* valamelyik bájtjára kerül a vezérlés a függvény visszatérés után, ez már garantálja azt is, hogy végül a végrehajtás a lényegi kártékony kódnál köt ki.

Szintén könnyíti a támadó helyzetét, ha az általa kívánt kódot el tudja helyezni környezeti változóba. Ennek a változónak a helye meghatározható, így az „egg” kódot nem is a felülírás folyamata során kell injektálni. Számos „cseles” technika található még a kreatív crackerek tárházában.

2.3. Második generációs támadások

Off-by-one túlcsordulások ♦ Programozói figyelmetlenségből sokszor előfordul, hogy megpróbálunk odafigyelni a problémára, de egy bájttal a puffer előtt kezdünk el írni, vagy egy bájttal túlírjuk a puffert. Ezeket „off-by-one” típusú túlcsordulásoknak nevezi az irodalom, és a crackerek egy ilyen pici tévedést kihasználva is el tudják érni, hogy az általuk kívánt kód hajtódjon végre, azonban ez már nagyobb hozzáértést és ügyességet igényel.

Heaptúlcsordulások ♦ Gyakori tévedés, hogy ha dinamikusan foglalunk memóriaterületet (ezáltal a verem helyett a heapen helyezkednek el az adatok), akkor csökkentjük a támadhatóság esélyét. Ez koránt sincs így. Noha a cracker számára minden klasszikus sebezhetőség egy feldobott labda, amit nehéz kihagyni, a heaptúlcsordulás ugyanilyen könnyen kihasználható.

A heap memória a dinamikusan foglalt adatok tárolására szolgáló memória. Ez logikailag elszigetelt a kódtól és a veremtől. Akkor használunk dinamikusan foglalt memóriát, ha nem tudjuk előre, hogy a program futása során mekkora területre lesz szükségünk, vagy az igényelt hely már nem férne el a vermen. A heap memóriaterülete általánosságban nem tartalmaz olyan visszatérési címet, mint a verem. Az elmentett visszatérési cím felülírásának híján a végrehajtási útvonal eltérítése nehezebb feladat. De egyáltalán nem lehetetlen, és biztosak lehetünk benne, hogy a crackerek képesek erre.

Egy példa lehet, amikor a heapen egy fájl nevét tárolja a program, amibe később írni fog.

Ha a program nevét felülírás segítségével megváltoztatjuk, akkor már egész érdekes dolgokat lehet csinálni, csak fantázia kérdése.

Függvénypointerek ♦ Szintén trükkös azoknak az eseteknek a kihasználása, ahol egy függvénypointert van lehetősége átírni a támadónak. Nem olyan triviális, mint a klasszikus túlcsordulás, de kódvisszafejtési tapasztalatokkal nem nehéz dolog.

2.4. Harmadik generációs támadások

Formatstring támadások ♦ Számos standard C függvény teszi lehetővé, hogy karaktereket írjunk fájlokba, pufferekbe és a képernyőre (pl. printf(), sprintf(), fprintf(), illetve ezek több több-bájtos (wide) és mutatott paraméteres változatai). Ezek a függvények az értékeket sokszor nemcsak elhelyezik a képernyőre, hanem formázzák is azokat. A formázási képességek lehetővé teszik, hogy a programozók szabályozzák az értékek megjelenésének módját, például egy számérték kiírható decimális és hexadecimális formában is.

A kiírás minden téren teljes mértékig szabályozható; szám esetén például milyen pontossággal írja ki a számot, használjon-e előtét-nullákat, és ha igen, maximum hányat, legyen-e tizedespont, maximálisan kiírt karakterek, stb. Igazán érdekesek olyan extra formázó karakterek, mint pl. a „%n”, ami kiírja, hogy eddig hány bájtnyi adatot írt ki a függvény. Szintén érdekes a „%6$n”, ami azt mondja, hogy a formázóstring paraméter utáni hatodik paraméter által mutatott memóriaterületre helyezze el a függvény azt, hogy eddig hány karaktert írt ki. Ha ilyen speciális formázás is van a kódban, akkor az nagyban megkönnyítheti a támadó dolgát.

A formázóstring és az utána következő paraméterek a veremben egymás után következnek. A támadó ennek ismeretében a paraméterek manipulációjával eléri, hogy felülírás történjen. Sajnos még az sem segít, hogyha a biztonságos változatot használunk (snprintf(), lásd [40]).

Még könnyebb a támadó dolga, ha már maga a formázóstring sem egy előre lekódolt érték, hanem paraméter a függvényben. Ekkor a formázóstring paraméter felülírásával elhelyezheti a neki tetsző rosszindulatú formázó karaktereket, amelyek segítenek neki a kívánt helyre injektálni kódokat és a kívánt helyre terelni a vezérlést.

A heapnyilvántartás támadása ♦ Ezeknek a támadásoknak az esetlegesen a heapen található adatok felülírásán túl az lehet a célja, hogy manipulálják a heapen található memóriablokkok nyilvántartásához használt ún. metaadatokat. A heapen elhelyezkedő lefoglalt memóriablokkok mindegyikéhez tartozik egy kis méretű metaadat rész, ami a heapkezelő működéséhez szükséges adatokat tárolja. A metaadatok tartalma heapkezelőtől függően, implementációról implementációra változik. Általában minden rendszerben különbözik egy picit, ráadásul lehetősége van a programoknak arra, hogy a rendszer által biztosított heapkezelő helyett sajátot használjanak. Erre akkor lehet szükség, ha a program ismeri, hogy milyen sajátosságokkal fog rendelkezni az általa lefoglalt memóriadarabok halmaza, és ehhez optimalizál egy nyilvántartási rendszert az éles programban. Ezáltal gyorsabbá teheti a memóriakezelést. Fejlesztési és tesztelési fázisban megkönnyítheti maga számára a teljesítmény- és terhelési mérések elvégzését és a memóriakezelési hibák kiszűrését.

A heapkezelők alap működési algoritmusa is különbözhet, azonban ez leszűkíthető pár elterjedt módszerre. A támadó célja általában csak annyi, hogy egy adatterületről átírjon a szomszédos területre, és közben okosan írja felül a szomszédos területhez tartozó kicsi lokális metaadat rekeszt. Itt általában olyan adatok találhatók, mint a memóriarekesz hossza, a következő memóriarekeszre utaló adat. A metaadatok kb. 8–32 bájtot foglalnak. Még zárt forrású rendszereknél is kódvisszafejtéssel könnyen kitalálható, hogy mivel célszerű felülírni ezeket a részeket ahhoz, hogy ezzel befolyásoljuk magának a heapkezelőnek a működését, és

a kezelő segítségével közvetett módon az „egg” kódra kerüljön a vezérlés [16].

A kivételkezelő rendszer támadása ♦ Szintén egy lehetőség a támadásra, hogy a veremtúlcsordulás segítségével átírják a veremben tárolt kivételkezelők címeit. Ezek után, ha a támadó eléri, hogy kivétel generálódjon, akkor már az általa manipulált helyre kerül a vezérlés. Azért haladó kategória ez a technika, mert sokszoros indirekciót, finomhangolást igényel a kihasználása [9].

2.5. A puffertúlcsordulásosos hibák eredete

A puffertúlcsordulásosos sebezhetőségek mind-mind programozói hibából fakadnak. A program írója figyelmetlen volt, és nem is volt tudatában, hogy hibát idézhet elő az általa írt programrészlettel, vagy csak egyszerűen nem foglalkozott valamilyen okból a lehetséges támadással. Például nem ítélte jelentős fenyegetésnek, netán későbbre halasztotta a kód letisztítását, de végül megfeledkezett róla vagy nem maradt rá ideje.

Programozói körökben elterjedt szállóige a „90/10-es” szabály. Ez annyit tesz, hogy a projektek nagy részénél a kód 10%-a a projekt idejének első 90%-ában születik meg, míg a programsorok fennmaradó 90%-a a maradék 10%-nyi időben keletkezik. Tévedni emberi dolog, minden ember hibázik. A programozó is ember, aki ráadásul szellemileg megterhelő vagy rendkívül monoton feladatot végez. Viszont általában pont a legkritikusabb részeket és hibajavításokat a programozó iszonyatos hajtás közepette végzi, óriási nyomás nehezedik rá. A korábban említett faktorok eleve megnövelik a hibázás valószínűségét, azonban az utóbb említett fáradtsággal megnehezített feladat elvégzése során nem csoda, hogy a hibák száma még magasabbra emelkedik.

Amíg világ a világ, ez így lesz. A legrosszabb helyzet a statisztikák alapján a programozás hőskorában volt. Ahogy a programfejlesztés menedzselésének tudománya és gyakorlata egyre jobban fejlődik, a helyzet javulni látszik. Bármi legyen is a hibák oka, a lényeg az, hogy sajnos a biztonsági rések ott lapulnak a programokban, akár nyilvánosságra hozza őket egy felfedező, akár nem.

3. Védekezési lehetőségek

Hogyan védekezhetünk a hibák ellen? Egyik fő irányvonal, hogy megpróbálhatjuk elkerülni, hogy hibákat vétsünk. Másik fő irányvonalbeli módszerekkel pedig védekezhetünk a programokban ott lapuló, nem ismert hibák ellen.

3.1. A hibák elkerülése

A hibák kivédésére az egyik kézenfekvő lehetőség az elkerülésük, vagyis hogy a biztonsági rések ne is legyenek ott, ne is legyen hibás a kód. Ez a korábbi okfejtésem szerint egy utópisztikus cél, de szerencsére több módszerrel is közeledhetünk az ideális állapot felé.

Biztonságos programnyelv használata ♦ Először is sok újabb programnyelv a természetéből fakadóan ellenállóbb a puffertúlcsordulásos kód írásával szemben, mert szigorúan típusosak, és beépített konténer osztálygyűjteményeket tartalmaznak. Az osztálygyűjteménybe tartoznak a tömbök, hash táblák, halmazok, stringek, és ezek metódusai ellenőrzéseket végeznek. Például a Java-ban a módosítható karakterláncokat alapesetben is *StringBuffer* osztályokban tároljuk, és a metódusok érzékelik, ha mondjuk egy karakterlánc másolási művelet során túl akarnánk írni egy szükségesnél kisebb méretű memória területen. C# nyelven is hasonló a helyzet. A felülírás megtörténte előtt kivételt dobnak, amit a program vagy robusztus módon lekezel, vagy elszáll. Maga a felülírás viszont biztosan nem történik meg, és ezál-

tal nincs is lehetőség arra, hogy ezt kihasználva a támadó fél elindítson egy rosszindulatú kódot, legfeljebb DoS támadást kivitelezhet. Természetesen a beépített ellenőrző kódoknak teljesítménybeli ára van.

A Java újgenerációs programnyelv, azonban a világon jelenleg rengeteg program fut, amely régebbi nyelveken íródott. Ezeknek az átírása nemcsak elképzelhetetlenül sok munka lenne, hanem teljesítmény és egyéb más szempontok miatt sem lenne megtehető jelen pillanatban. A mai napig is C nyelven íródnak a különféle UNIX operációsrendszer-leszármazottak magjai, a Linux kernel (rendszermag), vagy például a Microsoft Windows magjának legnagyobb része is (itt a C és C++ nyelv keveredik). Nemcsak az operációs rendszerek forráskódja C nyelvű, hanem a programcsomagok jelentős része is. Ennek egyrészt történelmi, másrészt praktikus okai vannak. A történelmi ok az, hogy a UNIX operációs rendszer írói pont a kernel és annak körítésének megírásához találták ki ezt a nyelvet egy másik nyelv továbbfejlesztésével. Olyan nyelvet kellett megalkotni, amiből optimális gépi kód generálódhat, hiszen egy operációs rendszer magjának gyorsnak kell lennie. Ebből több dolog is következik.

Vannak próbálkozások Java nyelvű operációs rendszerre, de egyelőre a processzorok órajelének megtorpanásával úgy néz ki, hogy kernel írására egyelőre nem létezik hatékonyabb és megfelelőbb eszköz a C nyelvnél.

C nyelven nemcsak magasabb szintű, nagy kifejezőerejű kódok írhatók (rövid kóddal írunk le bonyolult funkciót, például generikus típusok), hanem rendszerközeli szintű kifejezéseket is megfogalmazhatunk. A struktúrák és egyéb objektumok memóriatérképének teljes kontrollja, a bitmezők használata, a pointerműveletek és pointeraritmetika, a gépi kódú programrészletek beágyazásának *(inline assembly)* lehetősége mind-mind nagyon jól jönnek egy hatékony és gyors kód írásakor. Ha különböző programnyelven írt interfészek vagy programok között kell kapcsolatot teremtenünk, akkor az elérhető eszközök tárháza miatt szintén a C a kézenfekvő megoldás, legrosszabb esetben az *inline assembly* is bevethető. Az operációs rendszer írásakor a platform hardverével való együttműködéshez ezek egész egyszerűen elkerülhetetlenek.

A C nyelv fordítóprogramjai végletekig képesek optimalizálni az általunk írt kódot. Ez egyrészt annak köszönhető, hogy a nyelv közelebb áll a rendszerszinthez más, assemblynél magasabb szintű nyelvekhez képest. A Java nyelven íródott programokkal sok esetben nem tudunk megfogalmazni rendszerközeli kifejezéseket. A nagyobb kifejezőerejű nyelvek (melyekben egy kifejezéssel bonyolultabb műveleteket is leírhatunk) esetén nehezebb lehet optimalizált kódot előállítani. A Java nyelvű programok a természetükből fakadóan köztes kódra fordulnak, a köztes kód platformfüggetlensége, illetve a két lépésben történő fordítás (a második lépés a köztes kódról az adott platform gépi kódjára fordítás) nem tesz lehetővé olyan szintű optimalizációt, mint egy C nyelvű program esetén. A C nyelv jó optimalizációját segíti az érett kora is: már nagyon régóta fejlődnek a hozzá készült fordítók, a technológia kiforrott.

A „bitfaragás” vagy a pointeraritmetika egy hozzáértő kezében aranyat ér. A C nyelvnek fontos tulajdonsága, hogy bármit szolgai módon megcsinál, amit kérünk tőle. Ez adja a nyelv szabadságát – a C sok más nyelvnél szabadabb memóriahasználatot tesz lehetővé. Nincs például a karakterláncok másolásánál ellenőrzés, aminek köszönhetően viszont gyorsabb kódot kapunk. Ha mi azt írjuk le, hogy most írjuk felül a verem memóriaterületét, akkor azt ő szépen végre is hajtja. Nagyon ritka esetekben éppen ez lehet, amit akarunk, azonban ez ad lehetőséget a hibák figyelmeztetés nélküli elkövetésére: sem fordítás közben, sem futási időben nem kapunk hibaüzenetet, csak amikor már felülírtuk a vermet, és hibás visszatérési cím került az EIP-be, akkor száll el a program. Szerencsésebb esetben. Nem szeretném a DoS támadásokat lebecsülni, de rosszabb esetben elszállás sincs, hanem végrehajtódik egy alattomos injektált kód. A kezdő programozók nehezen elsajátíthatónak tartják a nyelvet, ami igaz is, pont a korábban említett dolgok miatt. A C-vel bármilyen alacsony- vagy magasszintű

kifejezés leírható. Viszont pont a nehezebben megtanulhatósága és a szabadsága segíti elő a puffertúlcsordulási hibák elkövetését.

Speciális könyvtárak, függvénykészletek, elemző eszközök használata ♦
A szakirodalom behatóan foglalkozik olyan kódellenőrzők tervezésével, melyek a statikus forráskódot elemzik, és megkeresik benne a puffertúlcsordulási hibákat (is). Programfutás közbeni elemző eszközök is léteznek: egyes memóriaszivárgást és memóriafogyást (a C-ben nincsen szemétgyűjtés) figyelő segédprogramok egyben a puffertúlcsordulási hibákra is figyelmeztetnek. Tehát ellenőrző eszközök léteznek mind statikus, mind dinamikus (azaz futásidejű) ellenőrzésre. Ilyenkor nem szükséges a programkód módosítása, dinamikus eszközöknél néha újrafordítás vagy hozzálinkelés szükséges.

Egy megoldás lehet az is, ha a szokásos programkönyvtárak helyett speciálisakat használunk. Ezekben a könyvtárakban megtalálható a standard Kernighan–Ritchie könyvtárak biztonsági szempontból naiv hozzáállású függvényeinek (pl. malloc(), memcpy() és strcpy()) biztonságosabb megfelelői. Előfordulhat, hogy a forráskódunkban le kell cserélni a standard hívásokat a könyvtárbeli megfelelőkre, de vannak olyan megoldások is, amik transzparensek. Ennek hatására a memória-menedzsment során a szabványos metaadatok helyett a speciális metaadatokban található információk segítségével a könyvtár futási időben megakadályozza a puffertúlírást. Azonban egyes könyvtárak használata egyrészt a forráskód valamilyen szintű átírásával jár (de pl. a libsafe könyvtárnál nem kell forrást módosítani), másrészt az ellenőrzések lefutása nagyban lelassítja a program futását (hasonlóan a Java vagy C# tömb osztályainak határellenőrzéséhez). Ez nem meglepő, a mérnöki szakmában szinte mindig van trade-off: ez a biztonság ára.

3.2. Létező hibák elkapása szoftveres védelemmel

Jó, ha megpróbáljuk elkerülni a hibák kódba kerülését, de biztos, hogy az emberi hibázást sohasem lehet teljesen kiküszöbölni, ezért feltételezhetjük, hogy a programokban mindig lesznek kiaknázásra váró puffertúlcsordulásos biztonsági rések, ezért fel kell készülni ismeretlen rések elleni védelemre. A védelem lehet teljesen szoftveres (pl. kanári szó típusú veremvédelmek), de általában szükség van valamilyen szintű támogatásra hardver oldalról is. A legbiztosabb az, ha a támogatás lényegében hardveres, és a platform hardvere képes megfogni ezeket az eseteket, az operációs rendszernek csak támogatnia kell a processzor adott funkcióit.

Kanári szavas védelem ♦ Először nézzük azt az esetet, ha csak szoftveres úton próbáljuk megakadályozni rosszindulatú műveletek végrehajtását. Ezek előnye, hogy jó eséllyel teljesen platformfüggetlenek lesznek. Számos ilyen megoldás született, a *StackGuard* és a *ProPolice* hasonló elven működnek. Az ilyen fajta védelem nem akadályozza meg magát a túlírást, hanem csak detektálja, és nem engedi a nem kívánt kódot futtatni.

Lényege, hogy egy kanári szót (*canary word,* más helyeken *booby trap*nek is nevezik) helyeznek a vermen található visszatérési cím és a vermen lévő változók közé. A kanári szó onnan kapta az elnevezését, hogy régen a vészhelyzetek elkerülésére a bányászok magukkal vittek a mélységbe egy kanári madarat. Folyamatosan figyelték az állapotát: ha a kanári nem érezte jól magát vagy esetleg elpusztult, az arra utalt, hogy mérgező gáz (sújtólég, szénmonoxid, egyéb gáz) van a levegőben, jobb mielőbb elhagyni a terepet.

Az informatikában veremvédelem esetén a kanári szó egy függvényhívásonként egyedi számértéket jelent. A függvényprológus a függvény kódjának végrehajtása előtti adminisztrációs műveletek szakasza. A kanári értéket egy véletlen információt biztosító forrás segítségével határozzák meg, és a függvényprológus során elhelyezik a visszatérési cím mellé, ugyanis a támadó célja a visszatérési cím felülírása, hogy a saját injektált kódjára oda tudja

adni a vezérlést. Az értéket ezen kívül elmentik egy olyan biztos helyre is, ahol nincs kitéve támadásnak. A függvényepilógus a prológus párja, azokat az adminisztrációs műveleteket foglalja magába, amik a függvény tényleges kódjának lefutása után következnek, közvetlen mielőtt a függvény kódjában sor kerül a visszatérésre. A függvényepilógus során automatikusan ellenőrzésre kerül, hogy a kanári szó memóriahelyén található érték ugyanaz maradt-e, mint a biztos helyre elmentett másolat.

Az elmélet az, hogy ha valamelyik vermen lévő változón keresztül veremfelülírás történik, akkor az esetek többségében a sikeres támadáshoz a kanári szót is felül kell írnia a támadónak. A veremfelülírást kihasználó exploitok nagy része ugyanis úgy működik, hogy a veremre injektálja a rosszindulatú kódot, miközben a felülírás során egyúttal a visszatérési címet is pont úgy módosítja, hogy az a veremre injektált kódra mutasson. Ezért a függvényből való visszatérés művelete helyett (kiolvasódik a visszatérési cím, ami hagyományos esetben a hívó utasítás *(call, jmp)* utáni utasítás memóriacíme, majd itt folytatódik a programvégrehajtás) a rosszindulatú kód kapja meg a vezérlést. Azonban, mivel a kanári szó a visszatérési cím és a változók között van, ezért a felülírás során a kanári szó is felülírásra kerül egy értékkel. A rendszer ezt az anomáliát észleli, hibaüzenetet küld, és megtagadja a további program végrehajtást.

Ha a kanári szó nem változott, akkor nagy valószínűséggel nem történt támadás. A kanári szó értéke olyan módon generálódik, hogy egy feltételezett támadó számára nehéz legyen kitalálni. A crackerek azonban mindig fejlődnek, és nem zárhatjuk ki teljesen annak a lehetőségét, hogy egyszer képesek lesznek olyan okosan kódot injektálni, hogy még egy cseles kanári szót is pont a saját értékével írjanak felül, és ezáltal a védelem ne jelezzen. Ennél sokkal frappánsabb utat is választhatnak: el lehet érni, hogy a kanári szót átugorják, ahhoz viszont a kihasznált függvénynek bizonyos feltételeket teljesítenie kell.

A kanári szavas védelem „bekapcsolása” egyáltalán nem igényel semmiféle kódmódosítást, viszont a programot újra kell fordítani, a támogatás egy része a fordítóprogramban található. A védelem függvényhívásonkénti és függvény visszatérésenkénti automatikus elvégzésének biztosítása gyakorlatilag a fordító feladata, a fordítót módosítják erre a célra. Ha egy olyan rendszert szeretnénk, ahol minden program tartalmazza a védelmet, megeshet, hogy mindent újra kell fordítanunk (ez természetesen csak nyílt forráskód esetén lehetséges, zárt forráskódú szoftvernél a kiadó vállalatnak kell kibocsátania a támogatással újrafordított binárisokat). Egyes rendszerek eleve tartalmazzák a védelmet, pl. OpenBSD (ProPolice, mellette W^X), Adamantix és Hardened Gentoo (SSP, mellette exec-shield vagy PaX).

A megoldásnak természetesen ára is van. A számítási költségek főleg akkor jelentkezhetnek jobban, ha kis méretűek a függvénytörzsek vagy gyakoriak a függvényhívások. A fejlettebb eszközök (ProPolice, SSP) megpróbálják detektálni, hogy az adott függvényen belül szóba jöhet-e egyáltalán a túlírás (pl. van-e karaktertömb változó, milyen műveleteket végeznek ezen), és ha nem szükséges, akkor nem alkalmazzák fölöslegesen a technikát, ezzel erőforrást takaríthatunk meg. Ezt figyelembe véve a rendszer egyes becslések szerint kb. 2–3%-nál semmiképpen sem lassul jobban.

A fejlett eszközök (ProPolice [15], SSP [38], StackGuard [12, 11], StackShield [42]) ezen felül megváltoztatják a lokális változók memóriabeli sorrendjét oly módon, hogy a felülírásnak jobban veszélyeztetettek közelebb kerülnek a kanári szóhoz, ezáltal nagyobb a valószínűsége, hogy a felülírásuk kiderül. A ProPolice és a PaX alkalmazása esetén egyaránt még maga a kernel is lefordítható a védelemmel, és a teljesítmény sem szenved jelentős csorbát. Azonban sokszor egy támadónak nemcsak a vermen található visszatérési cím felülírása lehet a célja, hanem szerencsés esetben elégséges, ha pár változót felülír, ezáltal a program végrehajtása a feltételes elágazások során úgy módosul, hogy a támadó eléri célját.

Null kanári és lezáró kanári ♦ További védelmi trükköket érdemes bevezetni azzal kapcsolatosan, hogy tudjuk, hogy a legtöbb esetben a veremfelülírás kiindulópontjai karakter-

lánc-változók, és a felülírást ezekbe a karakterlánc-változókba valamilyen vártnál hosszabb karakterlánc másolása váltja ki. Vegyük figyelembe, hogy a C stringek sajátossága, hogy a string végét egy lezáró nulla értékű bájt jelzi. Ha az előbbiekben ismertetett „random kanári" szón kívül elhelyezünk a veremben egy „null kanári" szót is, ami egy nulla értékű szó, akkor a string műveleteket befejezi, lezárja ez a karakter a következő módon. Ahhoz, hogy a támadó észrevétlenül felülírja ezt a kanárit, nullás kódú karaktert kéne az input stringbe csempészni, de ez a C szabályai szerint le is zárja a stringet, nem folytatódik a további túlcsordulás a visszatérési cím felé. Hasonló elveken alapul a „lezáró kanári" is. A gets() vagy más műveleteknél előfordul, hogy újsorjelek (CR vagy LF = 0x0D vagy 0x0A), vagy fájlvége-jel (0xFF) zárja le a bemenetet. A „lezáró kanári" ezek alapján egy nullás bájt és a korábbi értékek kombinációja.

XOR random kanári ♦ Az az érzésünk támadhat, hogy ez a védekezés már olyan cseles, hogy átverhetetlen. A Phrack magazin 56. számának 5. cikkében [7] és sok más írásban publikáltak már olyan trükköket, amivel kanári típusú védelmek korai verziói kicselezhetők. Tekintsük a következő, kihasználható függvénytörzset:

```
int f (char ** argv) {
  int pipa;     /* használaton kívüli változó */
  char *p;      /* a veremben közvetlenül a[] után van */
  char a[30];
  p=valami();
  strcpy(p, argv[1]); /* felülírja p-t is (de a kanárit nem) */
  strncpy(p, argv[2], 16); /* a kanári mögötti visszatérési címre ír */
}
```

A függvény hívásakor a verem (cím szerint növekvő sorrendben) ezeket tartalmazza: a, p, pipa, keretmutató *(frame pointer),* kanári, visszatérési cím, paraméterek. A visszatérési cím felülírása két lépésben történik meg, miközben a kanári nem sérül. A támadás részletes leírása [7]-ben olvasható. Kivédhetjük ezt a támadást azzal, ha a random kanári szót függővé tesszük a visszatérési címtől. Például XOR-oljuk össze a random kanárit vele, ezt nevezzük *XOR random kanári* védekezésnek. Az információk tehát a korábbi forrásokon kívül a visszatérési címre is támaszkodnak, ezért hiába nem sérül meg maga a kanári szó, a visszatérési cím megváltozása az epilógusban lévő ellenőrzésnél kiderül. Ma már a védekező megoldások XOR random kanárit használnak.

A csatának természetesen ezzel nincs vége. A crackerek nagyon ügyesek, felkutatják az elmentett kanári szavakat tároló memória helyet és megpróbálják manipulálni, vagy megpróbálják kitalálni a kanári szót, és ennek ismeretében intelligensen eltüntetni a nyomokat [37, 25, 7, 10, 40, 3]. De általában ennél sokkal frappánsabb és egyszerű kerülő utat találnak.

Memóriakiosztás megváltoztatása ♦ Borsot törhetünk a támadók orra alá, ha cselesen változtatjuk meg a futtatható kódok, adatok, metainformációk memóriabeli elhelyezkedését.

ASCII-pajzs (ASCII shield) ♦ Az ASCII-pajzs nevű védelem abból indul ki, hogy a kihasználás során stringműveletek által valósul meg a felülírás. A támadó célja mindig az, hogy a program valamely más részére adja át a vezérlést, legtöbbször az általa injektált kódra. Linux alatt hagyományos esetben a program kódja a 0x08048000 címre képeződik le (mappelődik). Ha a végrehajtható kódok címtartománya 0x01000000 alatti lenne, akkor az garantálná, hogy bármely utasítás memóriarekeszének címében legyen legalább egy nulla bájt (nevezetesen a legfelső bájt). Ez azért jó, mert ha injektálással (függvénypointer vagy memóriacím értékének felülírása) akarja a programvégrehajtást megváltoztatni a támadó, akkor a stringműveletek esetén a nulla lezáró érték félbehagyja a műveletet (akárcsak a null kanárinál) [22].

Az ASCII-pajzs technikának nincsen teljesítményvonzata, viszont a futtatható kód memóriabeli pozíciója linkeléskor dől el, ezért a védett helyre kerüléshez módosított (patchelt) linkerrel való újraszerkesztésre (linkelésre) van szükség.

A gyakorlatban az ASCII-pajzs technika úgy jelenik meg a Linuxban, hogy azok a kódrészletek, amiket a PROT_EXEC flag jelöl, a 16 MB alatti területre képeződnek le. Az ASCII pajzs területe még precízebben nézve a 0x00000000 és 0x0100ffff címek közötti terület (az első 16 MB + 64 kB), mert még ennél is garantálódik, hogy van legalább egy nullás bájt a címben. Az exec-shield [22] például erre a területre képezi le a futtatható állományokat. A PaX rendszer ezt még randomizációval is megfűszerezi, ezekről a későbbiekben lesz szó [32, 26, 35, 27, 29, 31].

Érdekes módon jelentőséget kap itt a processzor architektúrák bájtsorrendje (endianness). Az Intel x86 architektúra *little endian,* tehát először a kisebb helyi értékű bájt következik a felülírás során. Ez pech, mert a támadónak szerencsés esetben lehet, hogy nem számít, hogy az utolsó nullás bájtot nem tudja felülírni, (a nulla ugyanis terminálja a felülírási műveletet). Ez még bőven elég lehet egy támadónak, gondoljuk csak meg, hogy az off-by-one támadásoknál egy bájt felülírása is elég [9, 16].

Randomizációs technikák ♦ Megnehezíthetjük a crackerek dolgát, ha megpróbáljuk a célpontjaikat (például az adatok és futtatható kódok helyét a memóriában) „elrejteni” előlük. A memóriában tárolt információk esetén leghatásosabb, ha ezeket mindig véletlenszerű helyekre tesszük, ezek a cím randomizációs technikák. A randomizációs technikákban az a jó, hogy elhanyagolható költséget jelentenek, azonban a crackerek számára egy 32 vagy 48 bites véletlen szám kitalálása nagyon nehéz feladat. Egy crackernek fölösleges is ezzel próbálkoznia, hacsak nincs a véletlen forrásnak valamilyen gyengesége, akkor valamilyen más kerülő úton kell célt érnie.

Egy támadónak az exploit sikeres futtatásához ismernie kell, vagy ki kell találnia a memóriakiosztást *(memory layout).* Egy adott klasszikus veremtúlcsordulásos sebezhetőség esetén is bizonyos paraméterektől (általában operációs rendszer/disztribúció verziója, kernel verzió, service pack) függ az injektált kódban az a cím, amivel a vermen lévő visszatérési címet felülírjuk. Mivel más a memóriakiosztás, ezért más helyen található a verem és ezzel együtt máshova kerül az injektált kód is. A valódi exploitokban és sokszor a proof-of-concept kódokban is sokszor kis listát (vagy metodológiát) találhatunk, ami segítségével különböző Linux-disztribúciók és más operációs rendszerek (SunOS, HP-UX, SGI IRIX, egyéb UNIX-ok, Windows-ok) verziószámától, és/vagy a kernel verziószámától függően meghatározhatjuk a bemeneti paramétereket. Különböző verziók esetén más és más az érték, azonban ha nem használunk semmilyen randomizációs védelmet, akkor két különböző, de ugyanolyan verziójú rendszernél az értékek megegyeznek. Randomizációs védelemnél viszont futtatásról futtatásra változik a memória kiosztás, ezáltal egy adott támadáshoz a megfelelő értékek kitalálása rendkívüli mértékben megnehezedik. Persze a crackerek nem esnek kétségbe, hanem más módon próbálkoznak.

Hasonlóan nehezítjük a támadók eredményességét, hogyha véletlenszerűvé teszünk olyan értékeket, amelyek korábban determinisztikusan változtak (pl. egyesével növekedtek), és ezzel párhuzamosan szerepet kaphattak egy támadásnál. Erről a későbbiekben lesz szó.

Teljes ASLR (Full Address Space Layout Randomization) ♦ A teljes címtartomány-kiosztás randomizációja alatt azt értjük, hogy minden fontos címtartomány-paramétert véletlenszerűvé teszünk. Ezek alatt értjük:

– A program verem-báziscímének véletlenszerűsítését.

– A program memória leképezések (mmap) véletlenszerűsítését, beleértve a függvénykönyvtárakat is.

– A programkód báziscímének véletlenszerűsítését.

– A kernel veremcímének véletlenszerűsítését.

Mindez elhanyagolható költségvonzattal jár, azonban a programok újraszerkesztése (linking) szükségessé válik. A PaX definiálta először ezt a fogalmat, és biztosította is egyben a lehetőséget. A GRSecurity tartalmazza a PaX-ot. A címtartományok randomizációja és a korábban ismertetett ASCII pajzs technika együttesen is alkalmazható.

Más paraméterek randomizációja ♦ Nemcsak a memóriacímek véletlenszerű sorsolása erősítheti védelmünket. Az első szélesebb körben elterjedt védelemből alkalmazott randomizációs technika a BSD-ben jelent meg, és a TCP/IP protokollban egy bizonyos IP ID (Internet Protokoll IDentifier) mezőnek a kezdeti értékét tette véletlen számmá. Ezáltal bizonyos hálózattal kapcsolatos támadások válhattak nehezebbé. Szintén véletlenszerűsíthető az RPC (Remote Procedure Call) szolgáltatások privilegizált portja, a RPC XID azonosítója, ami az RPC elleni támadásokat nehezíti. Szintén elterjedt ma már a TCP kapcsolatok forrásportjainak és a folyamtok azonosítóinak *(process ID, PID)* véletlen generálása. Ezáltal egy támadónak nehéz megjósolnia egy támadás során újonnan megnyíló portnak a számát vagy újonnan elinduló folyamatnak az azonosítóját. Régebben ezek az értékek egyesével vagy determinisztikusan nőttek a rendszerben, ezáltal könnyen megjósolhatók voltak. Ma már többek között mind az OpenBSD (alapértelmezésben), mind a GRSecurity-s Linux (benne a PaX) ahol csak tudja, használja a randomizációs védelmet.

Alacsony entrópiájú véletlen számok ♦ Sok megoldás kitüntetett figyelmet fordít arra, hogy erős véletlen szám generálásához kibővítse az alacsony entrópiájú információforrás medencéjét megtöltő bemeneti adatok körét, így a támadó számára a véletlenszám kitalálása jelentősen megnehezedjen.

Összefoglalva azt mondhatjuk, hogy a véletlenszerűsítés költségvonzat nélkül teszi a behatoló számára olyan nehézzé a támadást, mint egy 16/32/48 bites véletlen szám eltalálása (64 bites architektúráknál 48/64/80 bites a véletlen szám).

Nem végrehajtható memóriaterületek ♦ Az első generációs technikák működéséhez alapvetően szükséges, hogy a veremmemória területe írható jogosultságokkal rendelkezzen, és végrehajtható is legyen. Kihúznánk a támadók méregfogát, hogyha a verem terület ugyan írható-olvasható lenne, azonban az itt elhelyezett adatok utasításként való végrehajtását hardveres közreműködéssel megtagadnánk. Ez csak a jéghegy csúcsa. Sok újabb generációs exploittechnika épül a Linux/UNIX variánsokon a program memória szegmensszerkezeten belül GOT és a PLT, a destruktorok, általánosságban a heap-területek, vagy a deregister_frame tartalmának felülírhatóságára [10, 40]. Ha ezek a memóriaterületek nem lennének végrehajthatóak, akkor hiába lenne sikeres a rosszindulatú kód injektálása (maga az injektálás nem akadályozódik meg), már nem kerülhetne végrehajtásra.

Igaz ugyan, hogy a hardveres segítség miatt kivétel (exception) dobódik, és gyakorlatilag elszáll a program (Segmentation Fault, más néven SIGSEGV), azonban valószínűleg ez még mindig sokkal jobb eset annál, minthogy egy rosszindulatú kód árnyékolási technikák alkalmazásával (például egy kész rootkit) észrevétlen módon fenyegetően ott lapuljon egy ártatlan masinán.

A legjobb az lenne, hogyha az architektúránk minden egyes memórialaphoz nyilván tudná tartani külön jellemzőként, hogy az végrehajtható-e. A régebbi i386 architektúrák sajnos lapok esetén együttesen kezelik az olvasás elleni védelmet és a végrehajtás elleni védelmet jelző flageket (PROT_READ | PROT_EXEC). Ez azt jelenti, hogyha egy lap olvasható, akkor végrehajtható is egyben. Mivel például a verem területnek írhatónak és olvashatónak kell lenni, ezért az előbbiek azt jelentik, hogy az itt lévő adatok egyúttal végrehajthatók is. A tá-

madók pont az olyan helyeket keresik, amik írhatók és végrehajthatók is egyben, mert oda tudják elhelyezni a kis „ajándékcsomagjukat” – gyakran nevezik tojásnak (*egg*), amit a cracker odatojik (*drop*) a célterületre.

A támadó valamilyen módon valahova beviszi a shellkódot, ez írás művelettel kell járjon, viszont ahhoz, hogy később végre is hajthassa (sokszor igen szövevényes úton, de végül rákerül a vezérlés) végrehajthatónak is kell lennie a területnek. Mi van, ha nincs is ilyen terület? Csak annyi, hogy a támadó dolga bizonyos mértékben megnehezedik. A célunk, hogy ne legyen a memóriakiosztásban olyan terület, ami egyszerre írható és végrehajtható is. Ezen kívül lehetőleg szeparáljuk a csak olvasható és az írható-olvasható adatokat is. Ez jelentős átszervezéssel és munkával jár, de megéri.

A szegmentáláson alapuló védelmi módszerek ♦ Tehát az Intel x86 architektúráknál az olvashatóság és a végrehajthatóság együttesen kezelődik, viszont szegmentált memóriakezelés használatakor lehetőség van a szegmens méretének határértékét meghatározni. Adódik, hogy írjunk elő egy limitet a kódszegmensre, ami alatt végrehajtható kódok vannak (0 memóriacímtől a limitig). A limit felett lévő dolgok pedig csak írható, és írható-olvasható részre bomlanak. Ezek után az a teendőnk, hogy a korábban említett memóriakiosztást aszerint (az előbbi stratégiának megfelelően) rendezzük el, hogy a végrehajtható kódok a határvonal alá kerüljenek.

Futás közbeni költségvonzat elhanyagolható, mert annyit jelent, hogy mivel ez a határvonal minden folyamatra más és más lehet, folyamat környezetváltásnál (process context switch) mindig át kell állítanunk a végrehajthatósági limitet a kódszegmens-deszkriptorban. Ez 6 bájt írását jelenti a GDT-be, mely csak 2-3 órajelciklusnyi időt vesz el/időbe kerül, így elhanyagolható.

Az IBM PowerPC (PPC) architektúráknál ugyan nincs támogatva laponként a végrehajthatóság szabályozása, azonban részletesebben felbontható a memória, mint az i386 szegmentálási rendszerénél látott kettévágás. Tizenöt darab határoló vonal segítségével 16 zónára oszthatjuk a memóriát, a zónák 256 MB méretűek lehetnek. Sajnos ezzel sem vagyunk sokkal előbbre, mert kompatibilitási okok miatt célszerű az i386-os rendszerben alkalmazott trükköt használni itt is.

A módszer hatására a platformfüggetlen gép szintjén már úgy fog látszani, minthogyha laponkénti végrehajthatósági bittel is rendelkeznénk. Ilyen elven működik a Solar Designer-féle *on executable stack patch,* a PaX egyik fajta védelme, az OpenBSD WˆX nevű védelmi vonala [14], a Linux exec-shield technikája. Amíg Solar Designer foltja kifejezetten csak a verem védelemre fókuszált, addig a többi megoldás az mmap()-olt adatok és a heap lehetőleg minél nagyobb részét is védi. A PaX és az exec-shield még kombinálja a fenti technikát az ASCII pajzzsal és randomizációval. Az OpenBSD is komplex védelmet nyújt a többi védelmi vonallal együtt [4]. A későbbiekben ezek működését részletesen is bemutatjuk. A Motorola 68K (680x0), VAX, MIPS, és ezeken kívül még sok beágyazott architektúra esetén sajnos nem tudjuk megvalósítani a fent vázolt védelmet, mert kulcsfontosságú architekturális részek hiányoznak.

A szegmentálási technikának elhanyagolható a költségvonzata, a hátránya viszont az, hogy sok olyan szoftver elszáll, ami olyan piszkos trükköket alkalmaz, amik nem mennek át a végrehajtási védelmen. Ilyen trükkök lehetnek a PROT_READ memóriaterületeket automatikusan végrehajthatónak feltételező szoftverek, GCC trampoline-ok, vagy olyan szoftverek, amik futás közben futtatható kódot generálnak. Problémás szoftver például az Emacs, az X-szerver, GCC trampoline-okat magukba foglaló szoftverek.

Az elszállás kellemetlen, azonban még mindig jobb, mintha hibás szoftverekkel dolgoznánk. Nemrég az OpenBSD olyan lépéseket tett, melyek következtében még több hiba fog felszínre kerülni, több program fog elszállni, több programról fog kiderülni, hogy rossz minőségű kódrészletek vannak bennük [5]. Theo de Raadt következetes és szigorú biztonságra

törekvő magatartásával az egész nyílt forrású közösségnek jót tesz, hiszen a megtalált hibák (pl. egy X-szerverben több mint 10 éve benne levő hiba) más operációs rendszerekben és disztribúciókban is okozhattak elszállásokat. A Linux példát vehetne az OpenBSD „secure by default” elvéről: ahelyett, hogy Alan Cox visszautasítaná az PID randomizáló kernel folt alkalmazását, vagy Linus visszautasítaná a PaX default kernelbe befűzését, ezeket a technológiákat igenis elérhetővé lehetne tenni az alap kernelekben is. Szerencsére jó irányba mutat, hogy Molnár Ingo exec-shield foltja bekerült a kernelbe, azonban ez nevetségesen kevés az üdvösséghez. Biztonsági szempontból ajánlott lenne az egész GRSecurity-t és a PaX-ot az alap kernelbe is befoglalni.

További megfontolások a memóriakiosztás megváltoztatásával kapcsolatban ♦ Ahhoz tehát, hogy a kívánt memóriaterületek végrehajtás elleni védelemmel legyenek felruházva, az eddigi memóriatérkép átrendezése szükséges, a programokat egytől egyig újra kell linkelni. Ezek után még le kell küzdeni a korábban említett „kompatibilitási problémákat” is, azaz ki kell gyomlálni a programokból azokat a hibákat, amik az új technológia miatt elszállást eredményeznek.

Hogy ne legyen félmunka, amit elvégzünk, érdemes még egyéb óvintézkedéseket is tenni. Említettem, hogy az i386 architektúrák sajnos együttesen kezelik az olvasás elleni védelmet és a végrehajtás elleni védelmet jelző flag-eket (PROT_READ | PROT_EXEC), melynek hardveres oka van. Azonban problémák vannak magával az ELF végrehajtható állományokkal is.

Biztonsági szempontból súlyos, hogy az ELF ABI a vermet végrehajthatóként jelöli meg, ezért a probléma azoknál a rendszereknél is jelen van, amelyek amúgy képesek volnának laponként kezelni a végrehajthatóságot (Alpha (61x84), SUN Sparc, SUN Sparc64, HP PA RISC).

Egyes végrehajtható fájlokban az úgynevezett *text* szegmensrészben együttesen találhatók a végrehajtható kódok és hozzájuk tartozó konstans adatok. A régebbi *a.out* bináris formátumban csak *.text, .data és .bss* szegmensek voltak. Úgy tűnik, hogy ahogy a világ átváltott az ELF formátumra, sajnos nem használták ki a lehetőségeit. Mivel a végrehajthatósági védelem, az ASCII pajzs és a címkiosztás-randomizáció is a program újralinkelését igényli, ezért érdemes úgy szervezni a fájlokat, hogy a konstans adatok, valamint a GOT és a PLT is a csak olvasható *.rodata* szegmensbe kerüljenek (ez utóbbi kettő is gyakori célpontja a túlcsordulásos támadásoknak). Mivel más architektúrákra is kihat a dolog (az i386 memória architektúrájának hiányosságát szokták *funky* vagy *mickey mouse* jelzővel is illetni), ezért mindenképpen rendbe kell ezeket a dolgokat tenni.

A GCC Trampoline-ok és a futás közben gerenrálódó kódok ügye ♦ A *trampoline* apró végrehajtható kódrészlet, ami futás közben generálódik, amikor a beágyazott függvények címét előveszik. Általában a függvényen belüli függvényt *(nested function)* tartalmazó függvény verem területén jönnek létre. A GCC makrókat biztosít a definiálásukhoz és használatukhoz. Alacsony szinten nézve a fordító CISC és RISC architektúráknál máshogy oldja meg a feladatot. Külön nehezítő körülmény, hogyha a processzorban külön van választva az adat- és az utasításcache (az i386 architektúra ilyen): ilyenkor speciális makrókkal ki kell üríteni az utasítástcache-t, különben gondokat okozna. A GCC trampoline-ok működőképességének megőrzéséhez nem a veremre, hanem más helyre kell tenni a trampoline-kódot, vagy pedig az adott futtatható állományra ki kell kapcsolni a végrehajtás védelmet.

Hasonlóan gondosan át kell alakítani a kernel által generált jelzés visszatérítő kódtörzseket *(signal return stubs),* amik szintén futás közben generálódnak.

Az OpenBSD WˆX rendszere ♦ A WˆX a PROT_WRITE XOR PROT_EXEC rövid alakja, a módszer célját sugallja: válasszuk külön az i386 architektúrában lapozási szinten egy-

bemosódó jogosultságokat. A szegmentálás aktiválásával és a korábban említett határvonal megfelelő állításával a módszer megakadályozza a támadásnak kitett verem, heap és egyéb memóriarészekben elhelyezkedő esetleges futtatható kódok végrehajtását [4, 14]. Működését Theo de Raadt levele alapján mutatnám be.

A régi a.out statikus végrehajtható állományoknál az alábbi a memóriaelrendezés (az `rw` az írható-olvasható, az `rx` pedig az olvasható-futtatható területet jelöli, a memóriacím alulról felfele növekszik):

```
          verem
          főprogram bss   rw
          főprogram adat  rw
------------------------------ ide húzható a határvonal
          főprogram kód   rx
00000000  használaton kívül
```

Ez nem működik dinamikus végrehajtható állományok esetén:

```
          verem
          libc adat       rw
          libc kód        rx
          luk
          ld.so adat      rw
          ld.so kód       rx
          luk
          főprogram adat  rw
          főprogram kód   rx
00000000  használaton kívül
```

Nincs hova meghúzzuk a vonalat, mert bárhova tennénk, lenne alatta és felette is írható memória. Azt lehet csinálni, hogy minden összetartozó egységben az írható-olvasható és az olvasható-végrehajtható rész közé beiktatunk egy 1 GB méretű rést (virtuális címtartományban):

```
          verem
          libc adat       rw
          luk
          ld.so adat      rw
          luk
          főprogram bss   rw
          főprogram adat  rw
40000000  használaton kívül
------------------------------ ide húzható a határvonal
          ...
          libc kód        rx
          luk
          ld.so kód       rx
          luk
          főprogram kód   rx
00000000  használaton kívül
```

Láthatóan megoldódott a probléma. A W^X a 3.4-es OpenBSD-ben mutatkozott be egy négy fő bástyából álló, komplex védelmi megoldás részeként. A megoldások magukba foglalják a korábban OpenBSD-vel kapcsolatban említett összes átszervezési lépést, ezen kívül pedig a *ProPolice* veremfelülírási védelmet is.

Az Molnár Ingo-féle exec-shield technológia ♦ Az exec-shield egy veremvégrehajtást megakadályozó kiegészítés a Linux kernelhez (a kernel már alapban tartalmazza), ami védi a heap és más támadásnak kitett memória területek egy részét is [22, 23]. A védelem egyben az ASCII shield technológiát is használja, és kombinálható címtartomány véletlenszerűsítéssel is.

Molnár Ingo levele alapján nézzük meg, hogyan változik a memóriakiosztás a technika alkalmazásával. A módszer alkalmazása előtti memóriakiosztás:

```
bffff000-c0000000 rwxp 00000000 00:00 0
4012f000-40133000 rw-p 00000000 00:00 0
40129000-4012f000 rw-p 00111000 16:01 4058    /lib/libc-2.2.5.so
40018000-40129000 r-xp 00000000 16:01 4058    /lib/libc-2.2.5.so
40013000-40014000 rw-p 00000000 00:00 0
40012000-40013000 rw-p 00011000 16:01 3759    /lib/ld-2.2.5.so
40000000-40012000 r-xp 00000000 16:01 3759    /lib/ld-2.2.5.so
0804c000-0804e000 rwxp 00000000 00:00 0
0804b000-0804c000 rw-p 00003000 16:01 3367    /bin/cat
08048000-0804b000 r-xp 00000000 16:01 3367    /bin/cat
```

Normál esetben a linuxos futtatható kód báziscíme 0x08048000. A módszer alkalmazása után a memóriakiosztás:

```
40234000-40235000 r--p 00955000 03:01 464809  locale-archive
40207000-40234000 r--p 0091f000 03:01 464809  locale-archive
40201000-40207000 r--p 00915000 03:01 464809  locale-archive
40001000-40201000 r--p 00000000 03:01 464809  locale-archive
40000000-40001000 rw-p 00000000 00:00 0
-------------------------------------------------------------- határvonal
01005000-01006000 rw-p 00000000 00:00 0
01004000-01005000 rw-p 00003000 16:01 2036120 /home/mingo/cat-lowaddr
01000000-01004000 r-xp 00000000 16:01 2036120 /home/mingo/cat-lowaddr
0024e000-00250000 rw-p 00000000 00:00 0
0024a000-0024e000 rw-p 00132000 03:01 319439  /lib/libc-2.3.2.so
00117000-0024a000 r-xp 00000000 03:01 319439  /lib/libc-2.3.2.so
00116000-00117000 rw-p 00014000 03:01 319365  /lib/ld-2.3.2.so
00101000-00116000 r-xp 00000000 03:01 319365  /lib/ld-2.3.2.so
```

Látható, hogy az elrendezés után a futtatható kódok mindegyike az ASCII pajzs védelme alá került (az `0x00000000`–`0x0100ffff` tartományba), a legnagyobb végrehajtható cím a `0x01003fff`. A végrehajtható határvonal is ide húzható. Persze az exec-shield védelem nagyon ígéretes, de önmagában kevés, nem véd sok ismert támadás ellen, amit a Molnár Ingo is több helyen hangsúlyoz.

PaX ♦ A PaX egy komplex védelmi rendszer, amely több különféle védelmi mechanizmus együttes alkalmazásával az összes szoftverhiba kihasználásának megakadályozását tűzi ki célul [8]. Sok vitafórumon hangzottak el már téves kijelentések a PaX-szal kapcsolatban. Ezekkel kapcsolatban fontos leszögezni, hogy a PaX nem csak egy verem végrehajtás elleni védelmi módszer. A PaX módosításai nem csak a kernel területét érintik, hanem a felhasználói programokat is magukba foglalják.

A PaX filozófiája a támadások következő három szintjét különítik el:

– Rosszindulatú kód bevitele a rendszerbe és végrehajtása.

– Létező kód nem megfelelő sorrendben való végrehajtása.

– Létező kód végrehajtása a tervezett sorrendben, de manipulált adatokkal.

A klasszikus shellkód-injektálási technika az első kategóriába tartozik, az ún. *return-to-libc* stílusú sebezhetőségek pedig a második kategóriába esnek.

A nem végrehajtható lapok (NOEXEC [30]) és az mmap/mprotect korlátozások (MPROTECT [28]) egy eset kivételével megakadályozzák az megfelelő kategóriába tartozó támadásokat. A teljes címtartományt véletlenszerűsítő technika (ASLR [26]) mind a három kategória kihasználását megnehezíti.

A PaX védelmi módszerei ♦

– VMA Mirroring ([36]): ez nem egy védelmi rendszer, hanem egy olyan alaptechnológia, amely szükséges mind a SEGMEXEC ([34]), mind a RANDEXEC ([31]) technológia implementálásához. A PaX fejlesztői a Linux kernel VM (virtuális memória menedzser) alrendszerét úgy módosítják, hogy egy taszk memóriaterülete két címtartományon keresztül nézve is látható legyen, de különböző jogosultságokkal.

  • ASLR: A címtartomány-véletlenszerűsítés kiterjed több területre [26]:

    ◦ RANDUSTACK [35]: a felhasználói veremcímek randomizálása

    ◦ RANDKSTACK [27]: egy taszk kernel szintű veremcímének randomizálása

    ◦ RANDMMAP [29]: az mmap() kernel interfészen keresztül kezelt memóriaterületek címének randomizálása

    ◦ RANDEXEC [31]: véletlenszerűség bevezetése a fő végrehajtható állomány leképezett címeibe

Az egyes technikák, amennyire csak lehet, platformfüggetlenek. Az ASLR technika ugyan nagyrészt platformfüggetlen, de abban eltérések mutatkoznak, hogy a címek mely bitjeit randomizálják.

– MPROTECT [28]: megakadályozza, hogy új végrehajtható kódot vezessenek be egy folyamat címtartományába. Ezt az mmap() és az mprotect() interfészek korlátozásával érik el. A technikának csak akkor van értelme, hogyha a NOEXEC módszer is aktív.

– NOEXEC [30]: a végrehajthatóság laponkénti szabályozása. Erre két alapvető megoldást kínál a PaX. Vagy a szegmentált memóriakezelést és a VMA Mirroring-ot használva valósítja meg (SEGMEXEC [34]), vagy pedig lapozásos memóriakezelésben végrehezvitt bravúros trükkel éri el a laponkénti végrehajthatóság szabályozását.

– SEGMEXEC ([34]): A Linux alapvetően nem használja a szegmentálást, bekonfigurálja úgy a hardvert, hogy az egész virtuális címtartomány egyetlen szegmens legyen. Azonban a szegmensek beállításával elérhető, hogy implementáljunk nem végrehajtható lapokat is.

  Az alapvető ötlet az, hogy a 3 GB-os felhasználói címtartományt két egyenlő 1,5 GB-os részre vágjuk. Az adatszegmens-leírót úgy állítjuk be, hogy a memória első felét fedje le, a kódszegmens pedig a második felét foglalja magába. Mivel a végrehajtható rész adatokat is tartalmazhat, ezért biztosítani kell, hogy az ilyen helyek mind a két szegmensben látsszanak és tükrözve legyenek. Ez a felállás kettéválasztja az adateléréseket az utasítás-beolvasásoktól olyan módon, hogy különböző lineáris címtartomány lesz használva a két célra. Ezért az eléréskor lehetőség van szabályozásra/beavatkozásra. Konkrétan: ha egy csak-adat leképezés van jelen, akkor ez a 0–1,5 GB tartományban van, hogyha utasításként próbálják kiolvasni, akkor a logikai cím az 1,5–3 GB tartományon belülre akarna hivatkozni, ami viszont kivételt okoz, és az eset lekezelhető.

  A címterület kettéosztásához és a tükrözött leképezés menedzseléséhez a VMA Mirroring technológiát használják.

– PAGEEXEC ([33]): Teljes értékű (!) laponként végrehajtási szemantikát implementál. Az implementáció során a PaX felüldefiniálja az x86 lapleíró bejegyzésekben található User/Supervisor bit jelentését, úgy, hogy azok a végrehajthatóságot/nem-végrehajthatóságot jelentsék, miközben biztosítják, hogy a nem végrehajtható lapokban az adatelérés továbbra is lehetséges maradjon.

  A memóriakezelő kivételkezelését emiatt át kell szervezni egy kicsit, de a PaX mérnökei nagyon rafináltak. A megvalósításnál fontos szerepet kap, hogy az i386-os architektúra külön adat- és utasítás TLB-vel rendelkezik (DTLB, ITLB), így megkülönböztethető az, hogy egy laphoz utasítás- vagy adatelérés miatt fordulnak. Ha nem lenne ez a sajátosság, akkor nem lehetne implementálni ezt a fajta PAGEEXEC-et az i386-os architektúrákon. Érdekesség, hogy az AMD K5-ös és K6 processzorok esetén noha külön ITLB és DTLB van, mégis – egyéb kizáró okok miatt – csak AMD K7-től fölfelé működik a PAGEEXEC.

Összefoglalva azt mondhatjuk, hogy a rendszer hihetetlenül mély szintű ismeretével a PaX mérnökei bravúrosan olyan technikákat vittek véghez, amit normális esetben megvalósíthatatlannak hinnénk. A PaX-nak ahol csak lehet, platformfüggetlenek a kódjai. A technikáknak persze vannak költségbeli vonzatai [41, 43].

A PaX dokumentációjában hosszú lista található nagyra törő jövőbeli terveikről, és ha elolvassuk a tervek részletes megvalósítási tervezeteit, akkor azt látjuk, hogy valószínűleg párat hamarosan véghez is visznek a fejlesztők.

3.3. Hardveres védelem: laponkénti végrehajthatóságot szabályozó bit

A technológia megjelenése ♦ 2004 elején hírek kezdtek szárnyra kapni, hogy az AMD alkalmazni fog a következő generációs x86 kompatibilis 64 bites processzoraiban egy laponkénti végrehajthatóságot lehetővé tevő technológiát. Eddigre már túl voltunk jópár óriási károkat okozó internetes féreg fertőzési hullámon. Habár a súlyos fertőzések következtében a biztonságra törekvés egyre nagyobb teret hódít, de véleményem szerint még mindig nem fordítanak elég figyelmet rá.

A misztifikált technológiát általánosságban *Execution Protection*ként emlegetik, az AMD „Enhanced Virus Protection” (EVP) néven marketingelte az újítást, máshol a szóban forgó bitről elnevezett „NX” (No Execution) névvel illették. Hasonló technológia az Intel platformon az Itanium termékvonalon már kezdettől fogva létezett (az Itanium nem x86 architektúrájú), ezért az Intelnél a bitet XD-nek (eXecution Disable), vagy az Itanium alapján DX-nek (Deny eXecution) hívják. Teljesség kedvéért a Microsoft által használt elnevezés a „Data Execution Prevention” (DEP), ami a technológia köré épülő szoftveres támogatási rendszert jelöli elsősorban.

A technológia jelen lesz minden 64 bites x86-64-es AMD processzorban, az Intel termékmarketingtől függően teszi elérhetővé az új chipjeiben. Szintén bejelentette a támogatását a VIA, a Transmetának pedig csak a kód-morfoló (code-morphing) szoftverét kellett módosítania, a hardver változatlan maradhatott [17]. A technológia bevezetését végig a hardvergyártók, az operációs rendszerek írói, a meghajtóírók és más szoftverírók aktív párbeszéde és kölcsönös segítségnyújtása jellemezte.

Az elnevezések áttekintése mellett technikailag a pontos meghatározása a szóban forgó NX (No Execute) bitnek a következő: egy eddig nem használt bit, a 63. bit a laptábla és lapkönyvtár bejegyzésekben [1]. Ahhoz, hogy a technológia működjön, az operációs rendszernek a CPUID processzor beazonosítással meg kell győződnie, hogy támogatott-e az Execution Protection, támogatott esetben az EFER (Extended FEature Rgister) regiszterben be kell billenteni az NXE (No-Execute Page Enable) bitet [2], és ha a processzor védett módban

dolgozik, akkor a PAE módot is be kell kapcsolnia [6]. Ezek után lehet használatba venni a laptábla bejegyzésekben található bitet és élvezni a hardveres támogatás előnyeit.

Ami ellen nem véd ♦ A médiában néha túlzó, elég hangzatos kijelentéseket láthattunk. Tény, hogy mivel a vírusok, férgek és egyéb rosszindulatú programok terjedéséhez sok más faktoron kívül a puffertúlcsordulás típusú támadások is szükségesek, ezért hogyha az NX bitet megfelelően támogatják az operációs rendszer részéről, ezeknek a legnagyobb része működésképtelen lesz. Azonban a néhol elhangzott információkkal ellentétben ez sem képes meggátolni minden túlcsordulásos támadást (mint ahogy a médiában messiásként feltüntetett egyik rákgyógyszer sem söpörte el eddig a kegyetlen betegséget a Föld színéről). Szintén a média pongyolasága, hogy egyes helyeken a technológiát úgy aposztrofálták, mint ami már magát a puffertúlcsordulást is észreveszi (tehát a felülírási folyamatot), a valóság azonban az, hogy a hardveres kivétel csak jóval azután generálódik, miután a felülírás megtörtént: akkor, amikor éppen el kezdene végrehajtódni az injektált kód.

Az Execution Protection a puffertúlcsordulást nem akadályozza meg, továbbra is törekedni kell a biztonságos programozásra. Olyan függvényeket kell használni, amelyek határellenőrzést hajtanak végre, figyelik az argumentumok hosszát, vagy bemeneti paraméterként beadhatjuk egyes paraméterek hosszát. Tartsuk be ezen kívül az összes biztonságos programozással kapcsolatos ajánlást. Ha a programunk futás közben végrehajtható kódot generál, akkor használjuk az operációs rendszer által biztosított függvényeket, amikkel végrehajtható memóriablokkot foglalhatunk, vegyük olyan kicsire ezt a memória helyet, amennyire csak lehet (ezzel csökkentjük a támadási felületet), és amint lehetséges, vegyük le a végrehajthatósági engedélyt.

Támogatás a Linuxon és UNIX-variánsokon ♦ A nyílt forrású operációs rendszerek a zárt társaiknál gyorsabban reagáltak az újdonságra. Molnár Ingo már 2004. július 2-án elküldte azt a kernel patch-et Linus Torvalds-nak, ami lehetővé tette a támogatást [18, 21], és ami a 2.4.22/23 és a 2.6 verziójú 64 bites kernelekben jelent meg a stabil ágakon. A 64 bites kernelek korábbi verzióiban és a 32 bites kernelekben nem támogatott a technológia. A támogatás azonban kezdetben alapesetben nem volt bekapcsolva. Később Linus Torvalds az apró kompatibilitási problémákat figyelembe véve és a technológia súlyos biztonsági problémákat megakadályozó voltára való tekintettel javasolta, hogy alapesetben is aktiválva legyen. A nagyobb Linux szállítók, mint a Red Hat és a SUSE átvették a támogatásokat. Nem voltak meghajtóspecifikus problémák, mert az x86-64 támogatás kapcsán a 64 bitre áttérésnél már kigyomlálták az ezzel kapcsolatos eseteket.

3.4. Betekintés az x86-os architektúrák hardveres támogatásaiba

Mialatt az NX bit megjelenésének voltam tanúja, folyamatosan ott motoszkált a fejemben, hogy az x86 architektúra kiterjedt hardveres támogatással rendelkezik, ezért nem értettem, hogy a szegmensvédelem lehetősége miatt miért van szükség laponkénti bitre is. Sőt, ha adott a szegmens védelem, akkor miért merül fel egyáltalán a probléma. Hogy a kérdésre választ kapjunk, részletesebben meg kell ismernünk az architekturális hátteret.

A 80386-os processzor messze megelőzte a korát. Megjelenésekor már benne volt minden potenciál, ami egy igazán fejlett operációs rendszer támogatásához szükséges jellemez (többfeladatos, többfelhasználós, többszálú operációs rendszer). Azonban, ha jobban visszaemlékezünk, akkoriban még a DOS-os érában voltunk, a Windows 3.1 pedig finoman szólva sem használta még ki a 386-osban rejlő lehetőségeket. Meglepő, hogy mennyire megelőzte a szoftvereket a hardveres támogatás nyújtásával.

A memóriavédelem támogatásának története az x86-os architektúrában ♦ A szegmensvédelem miatt elméletileg a No Execute (NX) bitre sohasem lett volna szükség [39, 24]. Már a 80286-os processzortól kezdődően – szegmentálásos memóriakezelés formájában – biztosított volt a memóriavédelem. Az egyes szegmensek írhatóság, olvashatóság és végrehajthatóság szempontjából bármilyen kombinációban megjelölhetőek, és ezek a tulajdonságok teljesen külön kezeltek. A szegmensek mérete pontosan beállítható a szegmenslimit segítségével, és erre hardveresen támogatott határellenőrzést is kapunk, a lapozott memóriakezelésnél viszont a lapméret a legkisebb felbontás (általában 4 kB), amivel szabályozhatunk egy memóriaterületet. Ezen felül még négyszintű privilegizált védelem is rendelkezésre áll: a 0. szinten lévő kódok mindent elérhetnek, a 3. szintű kódok a legkisebb privilégium szinttel rendelkeznek. Ha egy alacsony privilégiumú kód egy magasabb privilégium szintet igénylő műveletet akar végrehajtani, hardveres kivétel generálódik.

Sajnos a szegmentáláson alapuló memóriavédelem sohasem vált népszerűvé az operációs rendszerek körében, az egyetlen rendszer, ami valamennyire kihasználta, az OS/2 1.x sorozata volt. A 80386-ossal kezdődően lapozáson alapuló virtuális memóriakezelést is támogat a processzor, aminek megvan a saját laponként szabályozható memóriavédelmi rendszere. Valószínűsíthetően a programozás leegyszerűsítése és esetleg a hardveres implementáció megkönnyítése miatt a laponkénti memóriavédelmet a szegmenseknél megtalálható három különálló bitről mindössze egy bitre redukálták, ami csak annyit mond meg, hogy egy adott lap csak olvasható és futtatható, vagy pedig írható is. Ez azt a szomorú tényt jelenti, hogy hardverileg bármelyik lap, amit el tudunk érni, egyben végrehajtható is.

A szegmentált és a lapozott rendszer kombinálható is. Fontos azonban megjegyezni, hogy amíg a processzor védett üzemmódjában (protected mode, minden mai jelentős operációs rendszer ebben a módban tevékenykedik) a lapozásos memóriakezelés opcionális, addig a szegmentálás viszont nem az: mindenképpen „be van kapcsolva”. Az operációs rendszernek tehát mindenképpen be kell állítania a szegmensleírókat (-deszkriptorokat), és azokban lévő korábban már említett írás-, olvasás- és végrehajtás-biteket, hogy egyáltalán el tudják érni a memóriát.

A jelenlegi memóriakezelési gyakorlat ♦ Miért szükséges akkor a *No Execute* bit? Mert a szegmens memóriavédelem csak szegmensenként működik, az operációs rendszerek sajnálatos módon úgy állítják be a kódszegmenst, hogy lefedje az egész virtuális címtartományt, és minden végrehajtható legyen. Más szóval explicit jogot adnak arra, hogy bármi bárhol végrehajtható legyen, kikapcsolják a szegmensek nyújtotta védelmet. Persze a szegmentálás figyelmen kívül hagyásának megvannak a technológiai okai. Például a lapozásos memória kezelésnél belső fragmentáció alakul ki, a szegmentálásnál viszont külső fragmentáció lép fel. A memóriakezelést más elven kell vezérelni a két esetben.

A szegmentálás védelmi vonalának kikapcsolásával még ott van a lapozás memóriavédelme, azonban mint korábban említettem, ott a végrehajtást nem tudjuk külön kezelni. A korábban említett privilégiumvédelmet úgy implementálják a mai operációs rendszerek, hogy a kernel és a hardver közvetlen elérését igénylő meghajtóprogramok a 0. szinten futnak, minden más (felhasználói szintű) szoftver pedig a 3. szinten. Emiatt a felhasználói szintről nehezebb veszélyeztetni a rendszer magját, de a lapozás védelmi jellemzői miatt a mai gépek többsége a puffertúlcsordulásos vírusok és férgek táptalaja.

3.5. Szoftveres vagy hardveres megoldás legyen?

A probléma egyik megoldása lenne persze a szegmentált memóriakezelés védelmének használata. Az Exec-shield [22, 23] és a WˆX technológiák [4, 14] pont a szegmentálás védelmének reaktiválásával operálnak, azonban a módosításokkal jelentős változásokat indukáltak.

A szoftverek előnye a hardverekkel szemben, hogy relatíve könnyebb őket módosítani.

Azt gondolhatnánk, hogy könnyebb és olcsóbb telepíteni egy újabb verziójú Linuxot, mint processzort cserélni. Másik oldalról viszont az utóbbi évtizedek megmutatták, hogy az operációs rendszerek a hardvereknél hosszabb életciklusúak. Gyakran jóval olcsóbb egy új processzor, mint egy új operációs rendszer, nem is beszélve a szoftver- és hardverkompatibilitási problémákról. Úgy tűnik, hogy egy operációs rendszer valamilyen szintű újraírása (a szegmentálás bevezetése a memória menedzser újraírását jelentené, ez pedig maga után vonná a rendszer nagy részének módosulását is) sokkal gazdaságtalanabb lenne, nem beszélve a piacra kerülésig eltelő időről.

Az NX bitet nagyon könnyű volt implementálni, mert a processzornak azelőtt is volt módszere arra, hogy leellenőrizze, hogy egy adott lapot elérhet-e. Az NX bit esetében a processzor gyártók feladata csak annyi volt, hogy a módszert kicsit más körülmények között aktiválják, elég könnyű volt implementálniuk. Az Intel vezető fejlesztője azt nyilatkozta, hogy a processzor terveikben mindössze egy új kaput (gate) kellett felvenniük, és három utasítássort hozzá kellett adniuk a VSDL-kódhoz. Persze ne feledjük, hogy így is implementálni kell a szoftveres támogatást.

Az utóbbi évtizedben egyre nagyobb vírus és féreg fertőzési hullámok söpörtek végig a világon, a biztonsági szakemberek pedig különféle bravúros megoldásokkal és workaroundokkal próbálták csökkenteni a támadási felületeket. A hardver gyártók mindezt szemlélve megelégelhették a dolgot (talán egy kicsit későn), és a hardvermódosítás egyszerűsége miatt „kifejlesztődhetett” a hiányolt bit.

4. Összefoglalás

Betekintést kaphattunk a számítógép-védelmi szakemberek és a crackerek közötti harcba. Sokféle bravúros és érdekes technikát láthattunk. Egyesek hasonló elven alapulnak, mások pedig teljesen eltérő módon működnek, és más a védelmi céljuk. Személy szerint ezeknek a technológiáknak a híve vagyok. Hasonló alapelveket vallok, mint Jason Miller [19, 20]: minél szélesebb körben kell ezeket a megoldásokat elterjeszteni és használatukat lehetővé tenni. Fontos az is, hogy a rendszerek alapbeállítása biztonságos legyen: a hálózati szerverek legyenek inaktívak, a biztonsági technológiák bekapcsoltak. Élen jár és példa értékű ezen a téren az OpenBSD [5].

Úgy érzem, hogy eddig nagy lemaradásban volt a védelmi szakma a crackerekhez képest, manapság viszont a komplex védelmeket együttesen és megfelelően használva már nemcsak figyelhetjük és bekaphatjuk az ellenség lövedékeit, hanem háríthatjuk őket, és megerősödve felegyenesedhetünk.

A biztonságot szem előtt tartó gondolkodást még jobban el kell terjeszteni, a felhasználók és a programozók körében egyaránt. A szoftverek minőségbiztosítási metodikáját fejleszteni kellene, ezzel javulna a szoftverek minősége, kevesebb hiba lenne a programban, amit egy támadó ki tud használni. Ugyanis minden hiba egyben támadási felület is. Ahogy Ray Lillard fogalmazott a KernelTrap.org-on, Theo de Raadt 3.8-as biztonsági fejlesztései körül kialakult vitában: *Buggy code is also insecure code. Fix your damn code and quit whining.* (A hibás kód biztonsági szempontból is hibás. Javítsd ki a nyamvadt kódodat és ne sírjon a szád.)

5. Fogalmak és rövidítések

ASCII-pajzs (ASCII shield) ♦ Egy olyan egybefüggő memóriaterület, ahol elhelyezkedő bármely memóriarekesznek címében van legalább egy 0 értékű bájt. Ezért, ha a rekesz memóriacíme szerepelne egy karakterláncban, akkor ezen a karakterláncon végzett műveletek (strcpy(), strlen(), stb.) félbeszakadnának, lezárulnának.

Ez azért fontos, mert a támadásokhoz használt adatcsomagok *(egg)* sokszor stringműveletek segítségével jutnak be a rendszerbe. Az ASCII-pajzs ezért két dolog ellen is védhet: egyrészt ha közvetlenül itt szereplő adatokat (vagy kódot) akarnak felülírni, akkor az nem sikerülhet teljesen. Másrészt, ha ezen a területen található függvény memóriacímének szerepelnie kell a húsvéti tojásban, az szintén nehézség a támadónak. Megjegyzendő, hogy a védelem csak azokra az esetekre nyújthat gyógyírt, amikor stringműveletek történnek, memóriamásolásra (memcpy) például nem. Ezen kívül van olyan eset, amikor a támadónak elegendő csak azt a pár bájtot felülírni, amit tud (például a memória cím 4 bájtjából csak a legkisebb helyiértékű a nulla, de ez nem fontos a NOP slide-ok miatt).

ASLR – Address Space Layout Randomization ♦ A program konkrét futó példánya az operációs rendszer és segédkönyvtárak által is meghatározott memóriaképet mutat. A memóriában többek között megtalálható a futtatott kód és a hozzá tartozó adatok. Hagyományos esetben ezek a program minden indításakor ezek a virtuális memória ugyanazon helyére kerülnek. Az ASLR [26] technikák segítségével azonban ezek a program indulásának pillanatában véletlenszerű helyekre kerülnek. Ezáltal a támadó feladata megnehezedhet, egy sérülékenység kihasználásához általában az ASLR-technikák miatt a támadást címfüggetlenre kell megírnia.

BSD – Berkeley Software Distribution ♦ Az ősi UNIX operációs rendszerből kivált egyik fő irányvonal. A BSD később tovább ágazott, mint pl. OpenBSD, FreeBSD, NetBSD.

Canary word – kanári szó ♦ Speciálisan összeállított bájtok halmaza, melyeknek a sérülése jelzi, hogy puffertúlcsordulás történt. A bányákban régen alkalmazott módszerről kapta nevét, ahol a bányászok egy kanárit vittek magukkal, és a kanári viselkedéséből következtettek arra, ha sújtólég vagy más mérgező gáz van a levegőben.

CISC – Complex Instruction Set Computer ♦ Olyan processzorarchitektúra, melybean az elemi utasítások nem csak egyszerű műveletekre korlátozódnak, hanem találunk olyan mnemonikokat is, melyek igen komplex műveleteket valósítanak meg (ilyen pl. a számítási eredmény memóriába írása különböző indirekt címzésekkel). CISC processzor például az Intel 80x86,vagy a Motorola 680x0 architektúra. Általában az összetettebb végrehajtó egységek és a nagyobb bonyolultságból következő összetettebb strukturális függőségek miatt kevesebb gépi regisztert tartalmaznak az ilyen processzorok. Ezzel a filozófiával ellentétes a RISC elv.

CR – Carriage Return ♦ Egy speciális értékű (0x0D) bájt, mely fájl sorvéget jelent. A szöveges fájlok sorokból állnak, melyeknek a végét az LF és a CR karakterek jelzik. Az elnevezés történelmi okokból fakad, ugyanis karakteres nyomtatókon ez a bájt jelentette a „kocsi vissza” parancsot, amikor a nyomtató fej visszament a sor elejére. Operációs rendszertől függ az előfordulása a CR-nek és az LF-nek: a Microsoft operációs rendszerek (DOS, Windows) CR és LF karaktereket helyez a sorok végére ebben a sorrendben. UNIX és Linux operációs rendszereken csak LF karaktert találunk a sor végén, míg Macintosh rendszereken pedig csak CR-t. A különbségeknek történelmi és tradicionális okai vannak, melyeken már csak nagyon nehezen lehetne változtatni.

DEP – Data Execution Prevention ♦ Az NXE bit által biztosított technológiának a Microsoft általi elnevezése.

DX – Deny Execution ♦ Az Intel által már régebben biztosított NXE-hez hasonló megoldás, azonban ezt nem az x86 architektúrákon, hanem az Itanium architektúrákon nyújtják.

EBP – Extended Base Pointer ♦ Az i386-os architektúrájú processzorok egyik regisztere, amelynek speciális szerepe is van. Minden függvény hívásnál a veremre elmentődik az értéke, így veremtúlcsordulásoknál felülíródhat, szerepet kaphat.

EFER – Extended Feature Register ♦ Az x86 architektúrában egy speciális regiszter, mely segítségével különböző kiterjesztett módokat aktiválhatunk vagy kapcsolhatunk ki a processzorban. Az egyik ilyen mód az, amelynél használatba lehet venni a laponkénti végrehajtást szabályozó bitet. Ez az EFER regiszter NXE bitjének bebillentésével érhető el.

Egg ♦ A puffertúlcsordulást kihasználó exploit programoknak része (legtöbbször). Így nevezik a rendszerbe beinjektált kódot, melyre a sérülékenység kihasználása közben rákerül a vezérlés. Lásd még NOP-slide. Az injektált kód általában valamilyen rendszer megtévesztésére szolgáló preparált adatokat tartalmaz (például kódvisszafejtéses *(reverse engineering)* módszerekkel felderített táblázat bizonyos értékekkel kitöltve), valamint ezen kívül az esetek többségében egy shellkódot. Lásd még a shellkód címszót.

EIP – Extended Instruction Pointer ♦ Az x86-os architektúrákon így nevezik azt a speciális 32 bites gépi regisztert, amely a végrehajtandó utasítás memóriarekeszének címét tárolja. A regiszter értéke utasítás végrehajtásonként automatikusan növelődik a megfelelő értékkel.

ELF – Executable and Linking Format ♦ Egy szabványos többplatformos bináris végrehajtható fájlformátum. UNIX és Linux világban széles körben használt. Az ELF-fájl általában architektúra- és operációs rendszer-specifikus, tehát például egy azonos architektúrájú FreeBSD közvetlenül nem, csak közvetve tudja futtatni a Linuxra készült ELF binárisokat.

EVP – Extended Virus Protection ♦ Az NXE bit által biztosított technológiának az AMD általi marketing elnevezése.

GCC – GNU Compiler Collection ♦ Nyílt forrású fordítóprogram család, melynek egyik leghíresebb tagja a C-fordító. Bizonyos szakemberek véleménye szerint a 2.95-ös verzió a legjobb, de már rég túl vagyunk a 3.4-es verzión is, és készül a 4.1.

GDT – Global Descriptor Table ♦ Az Intel i386-kompatibilis számítógépekben a virtuális memóriakezeléshez kapcsolódó nyilvántartás rendszer egyik fontos része. A táblázat bejegyzéseket tartalmaz további leíró táblázatokra, ezáltal a rendszer többszörös indirekción keresztül tartja nyilván (a hardver és az operációs rendszer közösen) a lefoglalt virtuális címtartományokat, és az ezek logikai és fizikai leképezéséhez szükséges adatokat.

Gentoo ♦ Olyan Linux-disztribúció, melynél a teljes rendszer az alapoktól kezdve a legutolsó csomagig bezárólag telepítéskor forráskódból lefordul.

GOT – Global Offset Table ♦ ELF bináris állományok egy szegmense, a futás közben pedig memóriatérképének egy része. Függvények címeit tároló memóriaterület, a haladó szintű puffertúlcsordulásos támadásoknál fontos szerepet kap, egy potenciális célpont. Ebben a táblázatban található például a free() és a deregister_frame_info hívások belépési pontja.

GRSecurity ♦ Különféle védelmi technikák gyűjteménye, amivel Linux kerneleket és rendszereket tehetünk biztonságosabbá. Tartalmaz kanári típusú védelmi technikát, randomizációs megoldásokat, továbbá egyéb technikákat is, melyek e cikk keretein kívül esnek, például szerep alapú elérést szabályozó rendszert (Role Based Access Control).

LF – Line Feed ♦ Egy speciális értékű (0x0A) bájt, mely fájl sorvéget jelent. A szöveges fájlok sorokból állnak, melyeknek a végét az LF és a CR karakterek jelzik. Az elnevezés történelmi okokból fakad, ugyanis karakteres nyomtatókon ez a bájt jelentette a „sordobás” parancsot, amikor a laptovábbító henger pontosan egy sort tekert előre a papíron. Operációs rendszertől függ az előfordulás a CR-nek és az LF-nek: a Microsoft operációs rendszerek (DOS, Windows) CR és LF karaktereket helyez a sorok végére ebben a sorrendben. UNIX és Linux operációs rendszereken csak LF karaktert találunk a sor végén, míg Macintosh rendszereken pedig csak CR-t. A különbségeknek történelmi és tradicionális okai vannak, melyeken már csak nagyon nehezen lehetne változtatni.

NOP – No OPeration ♦ Az összes processzor architektúrában jelen levő assembly mnemonic, amely egy olyan gépi utasításnak felel meg, ami (elméletileg) nem csinál semmit, a processzor flag-jei és regiszterei nem változnak pár triviális regiszter kivételével, mint például az utasításszámláló. Egyes assembly nyelvekben néha máshogy rövidítik ezt a mnemonicot.

NOP-Slide ♦ Az exploitok által injektált kódok része az esetek többségében. Így nevezik a shellkódot megelőző NOP gépi utasítások sorozatát, mely néha igen hosszú, több ezer bájt is lehet. A lényege, hogy a támadónak könnyebb dolga van, mert nem kell a memóriában olyan pontosan pozicionálnia az injektált shellkódot. Ezáltal kevesebb ideig tart a támadó kód megírása, megbízhatóbban működik, változó körülmények között is működőképes maradhat. Azért nem kell pozicionálnia, mert elég elérni, hogy a vezérlés rákerüljön a NOP utasítások sorozatának bármely részére. Innentől a processzor szépen egymás után végrehajtja a shellkódig található NOP-okat, szemléletesen: úgy ráhuppan a shellkódra, mintha egy csúszdán csúszna le.

NULL – nulla ♦ Nulla értékű bájt vagy nyelvi kifejezés.

NX – No Execute ♦ Másik elterjedt elnevezés az AMD elnevezésekből származó NXE bitre, de a komplett technológiát is emlegetik ilyen néven.

NXE – No-Execute Page Enable ♦ Az EFER regiszter azon bitje, amely bebillentésével használhatóvá válnak a laponkénti végrehajtást szabályozó bitek.

PAE – Page Address Extension ♦ A Pentium Pro processzorral kompatibilis processzorok (i686) egy működési módja, amelybe kapcsolva a processzort a korábbi memóriánál jóval több virtuális memóriát van lehetőség megcímezni. A PAE nélkül 4 GB (32 cím bit) a folyamatok számára elméletileg összesen megcímezhető virtuális memória mennyisége. A folyamatok elméletileg egyenként is megcímezhetnek 4 GB memóriát. A gyakorlatban az operációs rendszerek saját céljukra elkülönítenek 1 GB vagy 2 GB virtuális címtartományt, így a felhasználói folyamatok számára 3 GB (illetve 2 GB) virtuális cím marad. PAE nélkül ezen osztoznia kell a folyamatoknak. A PAE azt eredményezi, hogy az összes virtuális címtartomány 64 GB-ra (36 bit) növekszik. Sajnos azonban továbbra is egyetlen folyamat egyszerre csak 4 GB-ot tud megcímezni. (Hasonló nagyságrendű virtuális memória-korlátozás x86-64 esetében nem jelentkezik.)

PaX ♦ A Linux rendszerek biztonságát növelő rendszer, mely sok komponensből áll, összetett módon képes védeni az operációs rendszert. Több lehetőséget kínál (PAGEEXEC és EGMEXEC) túlcsordulás-védelemre, ezen kívül teljes körű randomizációs szolgáltatást is biztosít, illetve fontos megemlíteni, hogy szerep és szabály alapú hozzáférés védelmi megoldást is tartalmaz.

PID – Process IDentifier ♦ UNIX/Linux rendszereken ez egy egyedi szám, ami egyértelműen azonosít egy adott folyamatot.

PLT – Procedure Linkage Table ♦ ELF bináris állományok egy szegmense, a futás közben pedig memóriatérképének egy része. Függvények címeit tároló memóriaterület, a haladó szintű puffertúlcsordulásos támadásoknál fontos szerepet kap, egy potenciális célpont.

ProPolice ♦ Haladó szintű kanári típusú védelmet alkalmazó rendszer.

PROT_EXEC ♦ Memóriaterületekre vonatkozó jelzőbit, ami akkor van bekapcsolva, ha a terület adatai végrehajthatóak. A Linux/UNIX operációs rendszerek kerneleiben C nyelven a PROT_EXEC makróval/konstanssal hivatkozhatunk a kérdéses bitre.

PROT_READ ♦ Memóriaterületekre vonatkozó jelzőbit, ami akkor van bekapcsolva, ha a terület adatai olvashatóak. A Linux/UNIX operációs rendszerek kerneleiben C nyelven a PROT_READ makróval/konstanssal karaktersorozattal hivatkozhatunk a kérdéses bitre.

RISC – Reduced Instruction Set Computer ♦ Olyan processzorarchitektúra, melyben az egyes elemi utasítások leginkább egyszerűbb műveletekre korlátozódnak, nem találunk olyan mnemonicot, ami a CISC processzoroknál látott komplexebb műveleteket valósítanak meg. RISC processzor például az IBM PowerPC, az SGI MIPS, a HP PA RISC architektúra. Általában az egyszerűbb végrehajtó egységek és az egyszerűbb struktúrális függőségek miatt jóval több gépi regisztert tartalmaznak az ilyen processzorok, mint a CISC társaik.

RPC – Remote Procedure Call ♦ Egy régi szabványosított technika, melynek segítségével egyik számítógépről végrehajthatunk másik számítógépen elhelyezkedő kódokat. Manapság már nemigen használt (kivéve: NFS és NIS), ennek ellenére sokszor feleslegesen jelen vannak olyan rendszerekben, melyek nem a „secure by default” elv alapján működnek. Mint minden szoftver, az RPC kiszolgálásában részt vevő komponensek is tartalmaznak biztonsági réseket, melyeket időről időre felfedeznek és kihasználnak a támadók.

shellkód ♦ Az exploit injektált kódjának (lásd még: egg) egy lényeges része. A shellkód már az adott processzorarchitektúra gépi utasításait tartalmazza, és eredményeképpen elindul a megtámadott rendszerben egy parancssori terminál (vagy ablak). A shellkódok pici, tömör remekművek, mert sokszor speciális feltételeknek kell eleget tenniük: ha input stringben kerül a rendszerbe, akkor nem szabad, hogy a gépi utasítások sorozata nulla bájtot tartalmazzon, mert ez lezárná a string műveletet. Ezen kívül általában minél rövidebbnek kell lennie. Ezeknek a feltételeknek léteznek csupán két tucat bájtból álló példái.

A sebezhetőségnek köszönhetően a kapott parancssor általában rendszeradminisztrátori (avagy root) jogokkal rendelkezik, így a támadó innentől kezdve azt csinál, amit akar.

SSP – Stack Smashing Protection ♦ Haladó szintű kanári típusú védelmet alkalmazó rendszer.

Trampoline ♦ Vermen elhelyezkedő végrehajtható kód, mely más kódokra adja tovább a vezérlést. Ez egy olyan legális technológia, ami sajnos megsérti azt a biztonságtechnikai alapelvet, hogy vermen nem helyezkedhet el végrehajtható kód. Ha alkalmazunk olyan technológiát, amely gátolja a vermen való kód végrehajthatóságot, akkor az a trampoline-okat alkalmazó programokat is működésképtelenné teszi, noha nem rosszindulatú támadásról van szó, hanem normál működésük ilyen. A megoldás csak az lehet, hogy az érintett szoftverekben ki kell küszöbölni a trampoline-ok alkalmazását. Érintett szoftver például az X-szerver, de máig is hasonló gondok vannak a Java futtatókörnyezetekkel, melyek futtatható kódot generálnak a működésük során.

W^X – Write Xor Execute ♦ Azt az elvet jelenti, miszerint ha egy program végrehajtható kódjait és adatait tekintjük, akkor az ezeket tartalmazó memóriaterületek között nincs olyan, ami egyszerre végrehajtható és írható. Célszerűen a program adatai írhatóak, a végrehajtható kódok pedig kizárólag csak végrehajthatóak. Matematikai szemlélettel azt mondhatjuk, hogy a W és az X által alkotott memóriaterületek diszjunkt halmazok, metszetük üres. Ha egy feltételezett támadónak sikerül az adatterületekre injektálnia a kártékony kódot, akkor nem tudja átadni rá a vezérlést, mert az adatokat tartalmazó memóriaterület csak írható.

XD – Execution Disable ♦ Az AMD NXE bitjét az Intel XD bitnek nevezi.

XOR – eXclusive OR ♦ Egy matematikai logikai művelet, melynél ha egyezik a két operandus logikai értéke, akkor hamisat ad eredményül, ellenkező esetben pedig igazat. Azzal a speciális tulajdonsággal rendelkezik, hogy ha egy bitsorozatot kétszer egymás után XOR-olunk egy másik bitsorozattal, akkor visszakapjuk az eredeti bitsorozatot. Ha egy bitsorozatot önmagával XOR-oljuk, akkor nullát kapunk eredményül. A művelet egyes tulajdonságait a számítástechnika minden területén használják (számítógépes grafika, kriptográfia stb.). A XOR kanárinál arra szolgál, hogy a kanári értékébe belekombinálják a visszatérési címet is. Gyakorlatilag információt kevernek hozzá az eddig létező információhoz, más oldalról nézve pedig rejtjelezzük a visszatérési címet a kanári véletlen értékével.

Hivatkozások

[1] Advanced Micro Devices. *AMD64 Architecture Programmer's Manual, Volume 2*. 2003. szeptember.

[2] Advanced Micro Devices. *BIOS and Kernel Developer's Guide for AMD Athlon™ 64 and AMD Opteron™ Processors*. 2004. április.

[3] CORE Security Technologies Advisory: Multiple vulnerabilities in stack smashing protection technologies. 2002. április 23., *SecurityFocus*.

[4] Jeremy Andrews: OpenBSD: Buffer overflow „solutions". 2003. január 31., *KernelTrap news*. URL `http://kerneltrap.org/node/573`. to the 2003-01-30 mail of Theo de Raadt: recent security changes in OpenBSD.

[5] Jeremy Andrews: OpenBSD: Improved memory allocation, beta testing 3.8. 2005. augusztus 23., *KernelTrap news*.
URL `http://kerneltrap.org/node/5584`. to the 2005-09-22 mail of Theo de Raadt: 3.8 beta requests.

[6] Rich Brunner: *Enhanced virus protection in AMD Opteron™ and AMD Athlon™ 64 processors*. Elhangzott a *WinHEC Conference-en*. 2004. május 1.

[7] Bulba – Kil3r: Bypassing Stackguard and Stackshield. 11. évf. (2000. május 1.) 56. sz., *Phrack Magazine*.

[8] Peter Busser: Memory protection with PaX and the stack smashing protector. 2004. március., *Linux Magazine*.

[9] Eric Chien – Péter Ször: *Blended attacks exploits, vulnerabilities and buffer-overflow techniques in computer viruses*. Elhangzott a *Virus Bulletin Conference-en*. 2002. szeptember.

[10] Core Security Team. *Vulnerabilities in your code – Advanced Buffer Overflows*. 2002. október 31.
URL http://www.vodun.org/papers/exploits/core_vulnerabilities.pdf.

[11] Crispin Cowan – Steve Beattie – Ryan Finnin Day – Calton Pu – Perry Wagle – Erik Walthinsen: *Protecting systems from stack smashing attacks with StackGuard*. Elhangzott a *Linux Expo* című konferencián. Raleigh, 1999. május.

[12] Crispin Cowan – Calton Pu – David Maier – Heather Hinton – Peat Bakke – Steve Beattie – Aaron Grier – Perry Wagle – Qian Zhang: *StackGuard: Automatic detection and prevention of buffer-overflow attacks*. Elhangzott az *USENIX Security Symposium* című konferencián, 7 konferenciasorozat. San Antonio, 1998. január, 63–77. p.

[13] Crispin Cowan – Perry Wagle – Calton Pu – Steve Beattie – Jonathan Walpole: *Buffer overflows: Attacks and defenses for the vulnerability of the decade*. Elhangzott a *DISCEX* című konferencián. 2000.

[14] Theo de Raadt: i386 WX. OpenBSD mailing list at MARC, 2003. április 17.

[15] Hiroaki Etoh – Kunikazu Yoda: Protecting from stack-smashing attacks. Jelentés, 2000. június 19., IBM Research Division.

[16] Halvar Flake: *Third generation exploitation, smashing the heap under Win2k*. Elhangzott a *Blackhat Briefings Windows* című konferencián. 2002.

[17] Eric Grevstad: Cpu-based security: The NX bit. 2004. május 24., *earthweb.com*. URL http://hardware.earthweb.com/chips/article.php/3358421.

[18] Micskó Gábor: x86 No Execute támogatás. Hungarian Unix Portal, 2004. június 3. URL http://hup.hu/node/6185.

[19] Jason Miller: Secure by default. 2004. május 13., *SecurityFocus*.

[20] Jason Miller: Security-related innovation in Unix. 2005. augusztus 28., *SecurityFocus*.

[21] Michael S. Mimoso: NX slams door on Linux buffer exploits. 2004. június 8., *Open Source News*.

[22] Molnár Ingo: Announcement document of „Exec Shield”, new Linux security feature, 2003. május.

[23] Ingo Molnár: Exec Shield announcement. Linux Kernel Mailing List, 2003. május 2.

[24] MS – Jerry Coffin: AMD Athlon64 4000+ review; a word (or several!) about the no-execution bit. 2004. október 19., *lostcircuits.com*, 11–13. p.
URL http://www.lostcircuits.com/cpu/amd_a64-4000/.

[25] Aleph One: Smashing the stack for fun and profit. 7. évf. (1996. november 8.) 49. sz., *Phrack Magazine*. URL http://www.phrack.org/archives/49/P49-14.

[26] Pax Team: *Address space layout randomization*. 2003. március 15.
URL http://pax.grsecurity.net/docs/aslr.txt.

[27] Pax Team: *Kernel stack randomization*. 2003. január 24.
URL http://pax.grsecurity.net/docs/randkstack.txt.

[28] Pax Team: *mmap() and mprotect() restrictions*. 2003.
URL http://pax.grsecurity.net/docs/mprotect.txt.

[29] Pax Team: *mmap() randomization*. 2003. január 24.
URL http://pax.grsecurity.net/docs/randmmap.txt.

[30] Pax Team: *Non-executable pages design & implementation*. 2003. május 1.
URL http://pax.grsecurity.net/docs/noexec.txt.

[31] Pax Team: *Non-relocatable executable file randomization*. 2003. február 19.
URL http://pax.grsecurity.net/docs/randexec.txt.

[32] Pax Team: *Overall description of Pax: design and implementation*. 2003. május 1. URL http://pax.grsecurity.net/docs/pax.txt.

[33] Pax Team: *Paging based non-executable pages*. 2003. március 15.
URL http://pax.grsecurity.net/docs/pageexec.txt.

[34] Pax Team: *Segmentation based non-executable pages*. 2003. május 1.
URL http://pax.grsecurity.net/docs/segmexec.txt.

[35] Pax Team: *Userland stack randomization*. 2003. február 12.
URL http://pax.grsecurity.net/docs/randustack.txt.

[36] Pax Team: *Vma mirroring, the core of SEGMEXEC and RANDEXEC*. 2003. október 6.
URL http://pax.grsecurity.net/docs/vmmirror.txt.

[37] Chris Ren – Michael Weber – Gary McGraw: Microsoft compiler flaw technical note. Jelentés, 2002. február 14., Cigital.
URL http://www.cigital.com/news/mscompiler-tech.pdf.

[38] Matt Rickard: ProPolice protected Gentoo Linux: GCC extension for protecting from stack-smashing applications. Jelentés, 2003. március 24., Gentoo Technologies.

[39] Anand Lal Simpi: A bit about the NX bit; virus protection woes. 2004. október 11., *Anandtech*.
URL http://www.anandtech.com/cpuchipsets/showdoc.aspx?i=2239.

[40] Core Security Team: *Vulnerabilities in your code – Format Strings*. 2002. december 20.
URL http://packetstormsecurity.org/papers/unix/core_format_strings.pdf.

[41] Pedro Venda: PaX comprehensive performance impact tests, 2005. október 20. URL http://www.pjvenda.org/linux/doc/pax-performance/.

[42] Vendicator: *Stack Shield: A „stack smashing" technique protection tool for Linux*. 2000. január 8. URL http://www.angelfire.com/sk/stackshield/.

[43] Nigel Watling: Security check. 2003. április., *Code Zone Magazine*, 6–12. p.

Összeurópai hallgatói műholdépítés csoportmunkájának támogatása szabad szoftverrel

Végh Károly

Kivonat

A cikk bemutatja azt a szabad szoftverekből összeállított csoportmunkát támogató rendszert, amely az SSETI nemzetközi hallgatói műholdépítő projekt résztvevőit szolgálja ki. Részletesen szólunk a rendszerrel szemben támasztott követelményekről, a szoftverválasztás szempontjairól, a szoftverek integrálásáról, és kitérünk a fontosabb beállításokra is. Szólunk felhasználóknak a rendszerre való felkészítéséről és a biztosított folyamatos segítségnyújtásról is.

Tartalomjegyzék

1. Motiváció

Vegyünk két tucat európai hallgatót. A lelkesebb fajtából. Azt a fajta „kockát”, aki mindig is teljesen komolyan űrhajós akart lenni. Ajánljuk fel nekik, hogy építhetnek egy műholdat. A kitörő zűrzavar csillapodtával közöljük velük, hogy egy fityinget sem kapnak, minden anyagot, erőforrást maguknak kell beszerezniük, önállóan kell kialakítaniuk valami kooperáló szervezeti formát, néha konzultálhatnak szakértőkkel, néha összegyűlhetnek egy műhelymunka erejéig, de hogy hogyan dolgoznak össze nemzetközileg, az az ő dolguk.

Lehet-e egyáltalán sikeres egy ennyire decentralizált projekt? Hogyan fognak együtt dolgozni elszórva egy tucat országban? Az interneten, persze, de konkrétan hogyan? Milyen platformra lesz szükségük? Milyen szoftvereket használnak majd? Mennyi új/régi technológiát tudnak felhasználni/megtanulni? Mennyire flexibilis egy rakétatudós, ha számítógéphasználatról van szó?

2. Történelem

A SSETI (Students Space Exploration and Technology Initiative) ESA (European Space Agency) projekt azért jött létre, hogy hallgatóknak tapasztalatgyűjtési lehetőséget biztosítson valódi űrprojektben való részvétellel. Lelkesen, lendületesen, önszerveződően vetették magukat bele az alapító diákcsapatok Európa majd’ 15 országából, olasz elektrotechnikusokból, német rakétameghajtó-tervezőkből, dán mikrokamera-fejlesztőkkel, angol rádióamatőrök támogatásával, spanyol rendszeranalitikus hallgatókkal, bécsi szerverinfrastruktúrával. Három műholdprojekt fut párhuzamosan, az Express, amit már fellőttek, az ESEO, a Föld körüli elliptikus pályára szánt műhold, és az ESMO, a Hold körüli pályára szánt műhold. Mindhárom már-már sci-fibe illő technológiákat használ, a Föld mágneses terére támaszkodva stabilizálva, esetleg kémiai rakétameghajtás helyett ionmeghajtással haladva – de az infrastruktúrát nem ellenőrzik az ESA szakértői.

A hallgatók, a mérnökök és a professzorok Európában szétszóródva élnek, tehát a csapatok közötti kommunikáció kulcsfontosságú.

Ez a maroknyi elszánt „műholdőrült” még hajlandó volt csak news-on, IRC-n, FTP-n át együttműködni, amit anno 1998 környékén egyetemi gépeken (PA-RISC-eken, Alphákon, PC-ken) Linuxszal – mai szemmel rettenetes módon telepítve – néhány lelkes Linuxfelhasználó állított össze, ezeknek legtöbbjét migrálni kell egy új, tervezett platformra.

De aztán gyorsan kezd növekedni a projekt. Sokan szeretnének csatlakozni, de mérnökpalánta (ill. végzett mérnök) létükre kommunikálni csak e-mailben, reply-all, quote-all módon, toppostolva (azaz alulra téve az idézett leveleket, felülre írva a választ) és MSN-en át tudnak. Valamint hallgatókról van szó, akik különféle egyetemi hálózatokat használnak. Sokan közülük egyetemeik hálózati házirendje okán nem érik el közvetlenül a projekt news-, levelező-, FTP- és IRC-szerverét. A netikettről nem tudják, mi fán terem, valamint teljesen normális, illetve számukra megszokott, hogy az egyetemük sokfajta hálózati szolgáltatást blokkol.

Hát hogyan támogassa a szabad szoftver világ egyszeri rendszergazda fia az ilyen megnyirbált szárnyú, de repülni vágyó *interstellar-engineer wannabe*-ket („csillagközi mérnök akarok lenni”)?

3. Platformtervezés

A SSETI Infrastructures (INFRA) csapata több forrásból kapott kifutóban levő UltraSparc és x86 architektúrájú gépeket. Ezekkel lehetőség nyílt egy teljesen új szerverplatform kiépítésére. Az új infrastruktúra kialakításának fő szempontjai klasszikusnak mondhatóak: hibatűrés, magas rendelkezésre állás, felhasználóbarát felület.

A hibatűrést és a magas rendelkezésre állást ajándékba kapott régebbi szervereken elérni nem kevés kihívással jár, de a kétdiszkes szervereken szoftver RAID1-es Logical Volume Managerrel (Linux alatt LVM, Solarison az LVM-szerű SVM) készített kötetekre ma már nem magas követelmény. Az INFRA csapat egyéni támogatóktól kapott még egy dualhostolható külső SCSI-tömböt is, amin módunkban állt a multiowner diszkcsoportokon definiált RAID1+0 alapú köteteket két szerver között megosztani, és ezek szolgáltatásait IP-címestül igény szerint egymásra terhelni.

A szűkös anyagi források miatt az új platform építésénél tehát a hardver után az első kérdés, hogy melyik szabad operációs rendszert használjuk. Hogy melyik Unix lesz a kiválasztott, az a nyújtott szolgáltatások szempontjából majdnem mindegy, de a rendszeradminisztrációt és a hardverrel való együttműködést illetően nem elhanyagolható, és fontos szempont a platform konzisztencia is. A SSETI INFRA csapat által épített platformon a SPARC szervereken Solaris 10 fut, az x86-alapúakon pedig Debian GNU/Linux.

3.1. Az INFRA szolgáltatásai

Ami az alapszolgáltatásokat illeti, többfélére van szükség:

1. IRC: online találkozókhoz, megbeszélésekhez,
2. news: hosszabb, összetettebb, szálakba rendezett és nem valósidejű vitákra,
3. FTP: adatcserére,
4. levelezőlisták,
5. webszerver: a dinamikus weboldalaknak, a közös használatú portálnak és a webes klienseknek.

Látható, hogy az infrastruktúra klasszikus ISP (Internet Service Provider) elemekből áll.

3.2. Rendszer- és szoftvermenedzsment

Több szerver esetén megfontolandó egy telepítőszerver készítése, nemcsak időtakarékossági okokból, hanem főként a szerverek konzisztens felépítését biztosítani. A konzisztens szerverkezelési koncepció kidolgozása nem fér bele előadásunk kereteibe.

Debian GNU/Linux esetén nem kérdés a szoftvercsomagok kezelésének módja, adott egy nagy és jól felépített csomag-repository, melyben a szükséges szoftverek túlnyomó részét megtaláljuk előre csomagolva, ami roppant kényelmes megoldás.

Solaris 10 esetén már több alternatíva merülhet fel a szabad szoftverek csomagból történő telepítésére. Választhatóak a SunFreeWare (SFW) csomagok, és azoknak már a Solarisba integrált verziói, a Companion CD csomagjai, és persze külső tárolók is használhatóak, illetve mindezek együtt is. Az INFRA csapat a Blastwave mellett tett le a voksát, mert csomagkezelője, a *pkg-get* hasonlít az *apt-get*-hez, továbbá a csomagok gyakran frissülnek, és a frissítések telepítése egyszerű. A Blastwave karbantartói évente többször közzéteszik a repository stabil ágát képező másfélezres csomaglistát, valamint létezik instabil és teszt ág is, a Debian fejlesztési modell mintájára. Éles környezetben természetesen a stabil ág csomagjait javasolt használni.

Szabad szoftveres megoldásokhoz szükségünk lesz egy szabad szoftveres környezetre. Az elemei legyenek szabad szoftverek, elterjedtek, tartsuk szem előtt a „ne bonyolítsd el” (Keep It Simple, Stupid – KISS) szabályt és az alkalmazások bővíthetőségét. Webszerverből is, webes alkalmazásokból is akad bőven. A webszerver legyen Apache, az adatbázis MySQL, kellhet még Perl és PHP. Mindezek könnyen és gyorsan telepíthetők csomagból.

Ha a választott szoftver nem lenne fellelhető a csomag-repositoryban, akkor persze fordíthatunk forrásból, ilyen esetben azonban fontos elkülöníteni a kézzel fordított szoftvereket egy külön könyvtárstruktúrába, vagy, hosszútávra is gondolva, csomagot készíteni belőlük.

3.3. Honlap

A projekt elért eredményeiről és a projektről magáról tájékoztatni kell az érdeklődőket, a támogatóknak visszacsatolást kell nyújtani – ehhez elengedhetetlen egy honlap.

Fontoljuk meg a projekthonlap lehetséges megoldásait, gondoljuk végig, hogy mi mindenre kell vigyáznunk egy közösségi publikus weboldal építésénél. Mire lesz szükségünk pontosan? A grafikai tervet a projekt PR csapata, a tartalmat a csapatkoordinátorok és a koordinátorcsapatok fogják szolgáltatni. Mindezen felhasználók nem feltétlenül webmesterek, ennek megfelelően leginkább WYSIWYG módon tudnak tartalmat szerkeszteni. Ezen felül modulárisan felépített honlapot érdemes készíteni, a különböző elvárt lehetőségek szem előtt tartásával. Egy projekthonlapon mindig felmerülhet az igény képgalériára, visszacsatolási és elérhetőségi űrlapokra, könnyen kezelhető menürendszerre, newsflash modulra, cikkkezelésre, közzétételi mechanizmusra stb., egyszóval szükségünk lesz egy tartalomkezelőre. Az [1, 2] weboldalak voltak segítségünkre a választásban, mi a Mamboserver-eredetű Joomla! mellett döntöttünk modularitása, komponenseinek, kiegészítőinek nagy választéka, PHP-háttere és felhasználóbarát editorfelülete okán.

Majd' minden elvárásunkra találtunk már kész plugint/komponenst: visszaszámláló futtatásidőzítőt, newsflash modult, pár kattintással cserélhető WYSIWYG editort és percek alatt telepíthető dokumentumkezelő komponenst. Objektum alapú a link- és képkezelésünk, van honlaptérképünk, és mindezt (némi segítséggel és oktatással persze) majdnem teljesen rábízhattuk a kevesebb adminisztrátori tapasztalattal rendelkező, de érdeklődő felhasználókra is. A menedzsment-csapat felügyeletére bíztuk a tartalomellenőrzést – ez a CMS-be épített felhasználói szereprendszer segítségével könnyen megoldható volt. Ami az adminisztrátorokra maradt, az a komponenstelepítés, a szerkesztők támogatása és a frissítések.

3.4. Központosított, közös használatú portál

A már említett különböző egyetemi hálózati korlátozások miatt mindenképpen szükség van webes kliensekre a különböző hálózati szolgáltatásokhoz: webftp-re, php-irc-re, webes newskliensre. Ezek valóban könnyebbséget jelentenek azoknak a felhasználóknak, akiknek nem áll módjukban saját klienst telepíteni, vagy a szolgáltatást a saját portján elérni. De pár száz felhasználó felett már mindenféle extra elvárásoknak kell megfelelni, valamint több, egymástól elkülönülő alkalmazás helyett hatékonyabb lenne egy központi webplatform, amelynek egyes moduljai a webkliensek.

Groupware-választás ♦ Hogyan egyesítsük a vállalati professzionalitást és a hallgatói rugalmas együttműködést egy nonprofit nemzetközi projektben? Új platformra, új koncepcióra és főleg új központi felhasználó- és szolgáltatáskezelésre van szükség. Webes csoportmunkához a kulcsszó persze: groupware, lehetőleg központi autentikációval, csoportkezeléssel, szintén moduláris felépítéssel. Szükségünk lesz feladatkezelőre, tudásbázisra, webmailre, webes FTP-kliensre, naptárra, címjegyzékre, valamint saját modulok készítésének és integrálásának lehetőségére. Fontos, hogy az autentikácót igénylő szolgáltatások legalább a backenddel (a közös használatú portál adatbázisával) együtt tudjanak működni. A projektünk hosszú távra szól, ezért fontos, hogy aktívan használt és fejlesztett szoftvert válasszunk. Az INFRA csapat a megvizsgált szabad groupware-megoldások közül (Opengroupware, dotproject, eGroupWare, PHProjekt) az eGroupWare-t választotta.

3.5. eGroupWare

Az eGroupWare (eGW) egy PHP-ban fejlesztett, adatbázis alapú groupware. telepítés után a felhasználók regisztrálhatják magukat, regisztrációkor megadják a projekt számára fontos információkat, majd e-mailcím-ellenőrzés után jelentkeznek az adminisztrátorcsapatnál témaszámuk aktiválására. Ez utóbbi fontos lépés, hiszen egy földrajzilag decentralizált projektnél – mint a SSETI esetében is – nem könnyű ellenőrizni a regisztrált felhasználók projekttagságának valódiságát. A SSETI meglehetősen jól átlátható adatbázis-koncepcióval érkezik – kitűzött célja, hogy a felhasználókezelés központosítva az eGW-n, pontosabban annak adatbázisán keresztül történjen.

Csoportkezelés ♦ A groupware-ben definiálandó felhasználói csoportokat érdemes a szervezet felépítésének megfelelően kialakítani, csapatonként egy-egy csoportot létrehozni. Sajnos ma még nem alakíthatóak ki metacsoportok, például a ESEO és az ESMO csapattagjait nem tudjuk a „SSETI” nevű csoportba is automatikusan belefoglalni, azaz a csoportkezelés egyszintű. Regisztrációkor a frissen készült témaszám automatikusan a *Default* csoport tagjává válik. Ezen csoportnak csak két groupware modulhoz van hozzáférése, a *Home* modulhoz, ami csak egy üdvözlő szöveget jelenít meg, valamint a *Logout* modulhoz, ami az aktuális munkamenetet lezárja. Témaszám-aktiváláskor kézzel kell hozzáadni a regisztrált felhasználót a saját csoportjaihoz, amivel a csoporthoz rendelt modulok hozzáférhetővé válnak számára.

Globális és lokális kategóriák ♦ Kategóriák segítségével több modul rendszerezi az általa kezelt adatokat. Ezen kategóriák két csoportba oszthatók: globális és lokális kategóriákba. A globális kategóriák minden – kategóriákat használó – groupware modulban megjelennek, a lokálisak értelemszerűen csak az adott modulban, ahol definiálva lettek. A kategóriák faszerkezetbe rendezhetőek, továbbá a portál adminisztrátorai által készített globális kategória-fákhoz a felhasználók lokális alkategória-ágakat is rendelhetnek, ezzel előre felosztva a projekt jellemző tevékenységi/információs területeit alegységekre, de elég szabadságot biztosítva a csapatoknak, hogy a saját területükön leginkább megfelelő adatstruktúrákkal dolgozzanak.

Feladatkezelés a ProjectManager modullal ♦ Egy nemzetközi csoportokból felépülő projektnél fontos a feladatok átláthatósága, egymástól való függőségeik kezelése, nyomonkövethetőségük, a határidők felügyelete stb. Az eGroupWare alapcsomagjában található ProjectManager modul meglepően kifinomult. A projekteket több szinten alprojektre bonthatjuk, egy-egy feladathoz rendelhetünk illetékes személyt, további munkatársakat, leírást, tervezett és valódi kezdő- és céldátumot, százalékos projektstátuszt, alprojektekre lebontott grafikus státuszábrázolást és még sorolhatnánk.

A szolgáltatások elérése natív klienssel ♦ Fontos, hogy ne csak webes kliensek létezzenek, hanem a backend-szolgáltatások natívan is elérhetők legyenek, a felhasználók lehetőség szerint használhassák a saját, megszokott klienseiket. Persze csak a webes témaszámukkal.

Tudásbázis ♦ A SSETI-nél az első üzembe helyezett publikus modul a Knowledge Base (tudásbázis) volt, ahol az első, már fellőtt műhold projektjének tapasztalatait tároltuk, első lépesben egy „Lessons Learned” (mit tanultunk belőle) nevű globális kategória alá rendezett lokális kategóriákban. Ezen kategóriafa cikkeinek szerepe a projekt tapasztalatainak megőrzése a még futó, és a jövőbeli projektek számára.

Levelezés ♦ Néhány csapatnak (Treasury, PR, INFRA, Management, LEGAL) szüksége volt hivatalos e-mailcímekre (`@sseti.net`), ehhez a groupware szelleméhez híven ezeknek a csoportoknak biztosítottuk a portál webmail *(Felamimail)* moduljához való hozzáférést. Mivel ez volt az első modul, ami tisztán csak kliensként működött, ennél merült fel először a dupla autentikáció problémája, azaz a portálba való bejelentkezés után a tagoknak még egyszer, a felhasználó/jelszó párral azonosítaniuk kellett magukat az IMAP-szerveren is (ugyanúgy, mint amikor saját IMAP-klienst használva közvetlenül kapcsolódtak a levelezőszerverhez). A *Felamimail* modul kiváló tulajdonsága, hogy a portálfelhasználó felhasználónevével és jelszavával képes (további felhasználói lépések nélkül) az IMAP-szerverre belépni. Ehhez persze szükséges, hogy a felhasználó IMAP-felhasználóneve és jelszava megegyezzen a groupware portálon használt adataival. Központi felhasználókezelés még nem lévén, szerettük volna a groupware-en át lebonyolítani az autentikációt. Az IMAP-szerverünk, a courier-imap szerencsére tud SQL-ből autentikálni. Az IMAP-szerver számára egy csak SELECT kiadárására alkalmas SQL-hozzáférést biztosítottunk a felhasználókat tartalmazó táblához. Szerencsére a Courier és az eGroupWare ugyanúgy használják az MD5 hashfunkciót, így a jelszó hashét elég volt egy példányban tárolni. Fontos szempont, hogy a groupware saját adatbázis struktúráját ne változtassuk meg külső autentikációs projekteknél, ezért például a *Maildir* elérési útvonalát, a mail-autentikáció minden szükséges információját és a szükséges extra adatokat egy új SQL-nézetben *(view)* gyűjtöttük össze, így végeredményben nem volt szükségünk adatszinkronizációra különböző autentikációs adatbázisok között.

FTP ♦ Az eGroupWare következő csak-kliens modulja az FTP webkliens volt, az eGroupWare *contrib* modullistájából.

Az új elvárásoknak megfelelően új FTP-szervert építettünk, az adatokat a külső diskarray egy RAID1+0-ba összefogott megosztott kötetére helyezve. Újra szembekerültünk az autentikációs problémával – és ismét egy SQL nézet lett a megoldás. A megfelelő FTP szerverszoftver kiválasztása már komolyabb feladat volt, konkrét elvárásaink a következők voltak: chroot funkció, TLS, SQL autentikáció, MD5-tel vagy crypttel hashelt jelszavak. Tapasztalataink szerint két megoldás jöhet igazán szóba: a Pure-FTPd és a ProFTPD. Egyik megoldás sem nyújt teljesen kerek megoldást, a ProFTPD csak OpenSSL-en át hajlandó SQL-ből base64 kódolású, hexadecimális formában generált MD5 jelszavakkal autentikálni. A Pure-FTPd adatbázisból való autentikáció esetén nem képes virtuális felhasználókat csoporthoz rendelni, valamint egyik FTP-szoftver sem hajlandó a virtuális felhasználók UID-jét feloldani felhasználónevekre SQL autentikációval. Végső – bár nem teljes – megoldásként az eGroupWare jelszó-hashelési funkcióját átállítottuk cryptre, valamint Pure-FTPd-t kiegészítettük funkcionális rendszercsoportokkal.

IRC ♦ Az eGroupWare ugyan rendelkezik egy chatmodullal, de az szigorúan követi a groupware irányvonalat és nem IRC-kompatibilis, valamint nem kliens–szerver felépítésű. Egy webes IRC modul viszont elengedhetetlen, szintén a groupware platformba integrálva. Ez utóbbi kitétel nélkül a CGI::IRC implementáció tökéletesen megfelelne igényeinknek, már ha hajlandóak vagyunk CGI-megoldásokat használni. A webportál és a szerver biztonságának érdekében a CGI::IRC alkalmazást egy másik szerveren, egy külön Solaris zónában futtatjuk, Apache helyett thttpd-vel, és egy eGroupWare frame modul gondoskodik a portálban való megjelenéséről. Önálló IRC-szerverként Hybrid-ircd-t futtatunk.

3.6. Felmerülő hiányosságok

A kiemelkedő szoftverek sem tökéletesek, néhány említésre méltó szoftverhiányosság az eGroupWare-ben: a már említett egyszintű csoportkezelés komoly időt vesz igénybe az adminisztrátorok részéről, a regisztráció során a konfigurálható extra kérdések információi nin-

csenek közvetlenül integrálva a későbbi aktív témaszámmal, valamint a teljesítmény a használt PHP template-motor miatt meglehetősen gyenge – egy-egy kattintás a portálban százas nagyságrendű SQL SELECT lekérdezést vált ki. Az online dokumentáció nem mondható teljesnek. Ez elmondható a Joomla!-ról is, bár ez utóbbiról kiváló könyv kapható.

Az FTP megoldásnál említett virtuális és rendszerfelhasználók kevert megoldása sem feltétlenül szépségdíjas megoldás, bár természetesen a funkcionalitás mindig előbbrevaló.

4. A felhasználók támogatása

Az ezres nagyságrendet közelítő projekttagság esetén a felhasználók támogatása nem elhanyagolható. Érdemes a tudásbázisba egy FAQ-t összeállítani, a felhasználókat az eleinte nekik gyakran idegen hírcsoportok használatra bátorítani, az általános kommunikációt és hatékony együttműködést tovább javító modulokat a közös használatú portálba tervezni (projektnaptár, címlista, verziókövetési rendszermodul az adatmegosztáshoz és -kezeléshez, integrált webes news-kliens, dokumentációs rendszer stb.).

A különböző szolgáltatásokhoz természetesen javaslunk szabad klienseket, Mozilla Thunderbirdöt levelezésre és hírolvasásra, Chatzillát IRC-kliensként, Filezillát FTP használatra. A felhasználóknak javasolt szabad kliensekhez természetesen tudunk szoftvertámogatást nyújtani. Lehetőleg próbáljuk rendszeresen használni ezeket a szoftvereket magunk is, hogy szükség esetén hatékony segítséget tudjunk nyújtani.

A felhasználók dolgának megkönnyítésére egységes hostnév-képzési szabályt alkalmazunk: *szolgatatas.projektdomain.tld*, példa hosztnevek: `www.sseti.net`, `ftp.sseti.net`.

4.1. Online viselkedési formák, netikett, felhasználótájékoztatás

Az önálló news- (NNTP), IRC- stb. szerver installálása nem jelent túl nagy kihívást, a legtöbb implementáció bőven lefedi alapvető igényeinket. Az igazi feladatot viszont a felhasználók jelentik. Legtöbbjük már hallott fórumokról, és chatelt már MSN-en, de newsgroupokról és online chatről vajmi keveset tud. Az érdekes jelenség, miszerint a szabad szoftverek online világában, a newsgroupokban, levelezőlistákon jobban ügyelnek a netikett betartására, mint az átlagfelhasználó, ez esetben is megfigyelhető – és természetesen a platform üzemeltetőire, a rendszergazdákra hárul az online viselkedési formák bemutatásának, bevezetésének a feladata. E szociális nézőpont korántsem alábecsülendő, messzemenően hatékonyabban kommunikálhatunk udvarias, megfontolt, letisztult, valamint a netikettet betartó módon.

Az első lépés egy online szabálygyűjtemény készítése, illemtan és felhasználási dokumentáció összeállítása, és a felhasználókhoz való eljuttatása.

A főbb tisztázandó pontok (melyeknél a használt szoftver releváns volt) az alábbiak voltak: a hírcsoportokat preferáljuk kommunikációra a levelezéssel és az MSN-nel szemben, betartjuk a beidézési szabályokat („mindig csak az aktuális szövegrészt idézd a válaszod fölött, ne teljes levelt alulra való beidézésével”), IRC-n a betartjuk a SSETI-specifikus „pontszabályt” (*dot rule*: online megbeszélésen nem szakítunk addig félbe senkit, amíg nem ír egyetlenegy pontot egy sorba, jelezvén, hogy mondandójának végére ért), valamint a különböző IRC csatornák szerepeit megértjük, és a megfelelő csatornára írunk.

5. A szabadszoftver-világ és a SSETI kölcsönhatásai, pillantás a jövőbe

A szabad szoftver filozófiának nem az online kommunikációmód minőségi megváltozása az egyetlen hatása. „Az információ szabad akar lenni” *(information wants to be free)* elv több

más ponton is megfigyelhető a projektben. Például a már lezártnak tekinthető SSETI Express műhold teljes technikai dokumentációja elérhető oktatási projektek számára. Valamint az Express esetén van egy sokkal Linux-közelibb pont is – ugyanis az űrjármű *(spacecraft)* fedélzeti adatkezelő egységén is egy real time-os patcheket tartalmazó kernelű mini Linux futott, szintén dán fejlesztők és diákok büszkesége.

A szabad szoftvereket a nyílt dokumentumformátumok használata is kíséri, ugyanis néhány egyetemen politikai hátterű megfontolásból szívesebben veszik, ha nem konkrét kereskedelmi szoftverhez kötött formában továbbít az ember adatot. Ekkor kerül előtérbe például az OpenDocument formátum.

A projekt nagyon gyorsan fejlődik, az Express fellövésekor egy felhívást követően világszerte rádióamatőrök önkéntesen lesték-várták az Express adását, amit dekódolva továbbítottak a Mission Control Center szerverének – ennek sikeréből indul ez idő tájt egy SSETI-vel párhuzamos nemzetközi projekt, a GENSO, a Global Educational Network of Satellite Objects (Genso japánul: elem, alkotórész – Műholdobjektumok Globális Oktatási Hálózata), amiben nemzetközi összefogásban rengeteg földi állomás kiépítése a cél világszerte.

6. Kinek van kedve űrhajót építeni?

A sok új megvalósítandó ötlet rengeteg munkával és feladattal jár. A projektek jellemzően igen nyitott felépítésűek, a SSETI-ben diákok, PhD hallgatók és professzorok is részt vehetnek. A GENSO még ennél is nyitottabb. Lelkes munkaerőre mindig szükség van, az INFRA csapatban is. Ki tudja, talán írhatunk még a Holdról az addig még kiépítendő WGAN-on (Wireless Giant Area Network) át előadást a 2020-as Linux konferenciára – célunk a világuralom...

Hivatkozások

[1] cms matrix: tartalomkezelők összehasonlítása. URL `http://cmsmatrix.org/`.

[2] OpenSourceCMS: nyílt forrású tartalomkezelők összehasonlítása. URL `http://opensourcecms.com/`.

Kettős játszma:
SELinux a Red Hat Enterprise Linux 5-ben

Béres László
<beres.laszlo@ulx.hu>

senior IT engineer, trainer
ULX Kft.

Kivonat

Éles, magas biztonsági igényű környezeteinkben a hálózati határvédelem mellett rendkívül fontos egy adott rendszeren belül található szolgáltatások, folyamatok határvédelme, a felhasználók feladatköreinek elkülönítése is. Az SELinux a hagyományos DAC (Discretional Access Control) mellett a MAC (Mandatory Access Control) alapelvének megfelelően működik. Szabályai előírják a rendszer szolgáltatásainak, folyamatainak egyes erőforrásokhoz (fájlrendszerekhez, hálózati interface-ekhez, eszközökhöz stb.) történő hozzáférését, korlátozhatják a rendszerfelhasználók által indítható folyamatokat. Ezekben a rendszerekben a kettős védelemnek köszönhetően megszűnik a root felhasználó mindenhatósága, így biztonságosabb rendszert kapunk.

A előadás bemutatja az SELinuxot az RHEL 5-ben megjelenő, egyszerűbben használható menedzsment eszközökkel és szabályokkal együtt.

Az SELinuxról ♦ Az amerikai Nemzetbiztonsági Hivatal (NSA) által fejlesztett SELinux (Security Enhanced Linux) népszerűsége az utóbbi időben jelentősen megnőtt, többek között a Red Hat Enterprise Linux 4-es változatának köszönhetően. Az RHEL 4 gyárilag tartalmazza a legfontosabb szolgáltatások extra védelméhez szükséges SELinux keretrendszert, valamint számos szolgáltatáshoz a biztonsági házirendet *(policy).*

A hagyományos Linux kernel által használt jogosultságkezelés a felhasználók jogaitól teszi függővé a rendszerfolyamatok erőforrásokhoz, valamint fájlokhoz történő hozzáférését. Ha egy folyamat a *root* nevében fut, akkor hozzáférhet az operációs rendszer tetszőleges részéhez. Ha ez a folyamat egy támadó által módosításra kerül, akkor bizony a behatoló a rendszer bármely komponense felett teljes jogkörrel fog rendelkezni.

Az SELinux implementálása bevezette a Mandatory Access Control (MAC) modell használatát, valamint megjelenítette a biztonsági adminisztrátor fogalmát. Utóbbi meghatározhat egy, az említett tradicionális jogosultsági modellt (DAC – Discretional Access Control) kiegészítő szabályrendszert, amelyben azt definiáljuk, hogy pontosan ki mihez férhet hozzá. Az egyes felhasználókat tartományokba (domain) delegáljuk, és pl. ha egy adott felhasználó – szándékosan vagy véletlenül – úgy határoz, hogy más userek számára is elérhetővé teszi állományait, az SELinux szabályai még ezt is megakadályozhatják.

A fentiekből logikusan következik, hogy az SELinux használatával teljesen zárt, megbízható rendszert is készíthetünk, így tökéletesen különválaszthatjuk a rendszer üzemeltetéséért felelős hagyományos root felhasználót, valamint a biztonsági szabályokért felelős security administratort.

Mi várható az előadásban? ♦ Az előadásból megismerhetjükaz SELinux működésének alapelveit, egyes üzemmódjait (enforcing/permissive, illetve targeted/strict), a Red Hat Enterprise Linuxban található megvalósítását. Működés közben mutatjuk be a nemsokára megjelenő Red Hat Enterprise Linux 5 új SELinux segédeszközeit, amelyek a szabályok létrehozásához, implementálásához, a hibásan működő alkalmazások felderítéséhez nyújtanak segítséget.

Az openSUSE projektről és a Novell SUSE Linux termékcsaládjáról

Az openSUSE projekt [1] a Novell által támogatott közösségi program. A Linux általános felhasználását ösztönözve az openSUSE.org [2] ingyenes, megkönnyítve a világ egyik legelterjedtebb Linux-disztribúciójának, a SUSE Linuxnak elérését. Az openSUSE projekt eredményeit felhasználva építi fel a Novell vállalati felhasználásra ajánlott platformját, a SUSE Linux Enterprise 10-et. A Novell vállalati Linux termékcsaládjának két legfontosabb tagja a szerver és a desktop megoldás.

SUSE Linux Enterprise Server 10 – megoldás létfontosságú rendszerekhez ♦ Leállásokkal, biztonsági problémákkal, vagy éppen magas költségektől szenved? Az adatközpontnak segítenie kell a céget, nem pedig akadályoznia a gördülékeny működést. Itt az ideje valami újat választani a vállalati feldolgozáshoz – pontosan ezért fordul egyre több szervezet a világon a Novell által kínált SUSE Linux Enterprise Server [4] felé. Ez az innovatív adatközponti „erőmű" ugyanis minden szempontból készen áll az üzleti felhasználásra. Maximális megbízhatóságot és biztonságot kínál – és a lehetőséget az infrastruktúra költségeinek akár 70 százalékos csökkentésére.

A SUSE Linux Enterprise Server egy vállalati szintű kiszolgáló, amely úgy készült, hogy megbirkózzon a nagyobb adatközpontok terheléseivel is. Csak a SUSE Linux Enterprise Server kínál nyílt, méretezhető, nagy teljesítményű adatközponti megoldást, amely beépített virtualizációt, alkalmazásvédelmet és rendszerfelügyeletet tartalmaz a hardver architektúrák teljes skáláján. A SUSE Linux Enterprise Server használható általános célú kiszolgálóként, de hangolható különféle terheléstípusokhoz is, és problémamentes együttműködést kínál a meglévő informatikai infrastruktúrával. SUSE Linux Enterprise Servert használva a cég látványosan csökkentheti költségeit, úgy, hogy közben a piac legbiztonságosabb és legmegbízhatóbb adatközponti kiszolgálóját használja.

SUSE Linux Enterprise Desktop 10 – költséghatékony, rugalmas alternatív asztali környezet ♦ Költséghatékony, egyszerűen használható, biztonságos asztali környezetet keres? Ezen a területen is szükség van választási lehetőségre! A SUSE Linux Enterprise Desktop [3] az első olyan vállalati Linux asztali környezet, amelyik megfelel mindezen igényeknek, és együttműködik a meglévő rendszerekkel.

Miért érdemes a SUSE Linux Enterprise Desktopot választani vállalati felhasználásra? A válasz egyszerű. Nyílt forráskódú megoldás, vagyis használata nem jelent magas licencköltségeket. A felhasználókat szem előtt tartva készült, így egyszerűen használható és kompatibilis a már használatban lévő rendszerekkel. Rugalmas, biztonságosan üzembe helyezhető, képes általános asztali rendszerként működni, de vékonykliens-alapú környezet is kialakítható belőle.

A SUSE Linux Enterprise Desktopot választva olyan megoldáshoz jut, ami:

1. egyszerűen használható,
2. együttműködik a meglévő rendszerekkel,
3. biztonságos,
4. költséghatékony,
5. rugalmasan telepíthető és üzemeltethető.

Ha mindehhez hozzávesszük a Novell világszínvonalú műszaki támogatását és az új Novell Customer Centert, látható, hogy a SUSE Linux Enterprise Desktop minden szempontból megfelel az üzleti elvárásoknak.

Találja meg a megfelelő helyet a intézményében, ahol elkezdheti használni a SUSE Linux Enterprise Desktopot, hogy kihasználja a termék nyújtotta költségmegtakarítást.

Hivatkozások

[1] Magyar openSUSE weboldal. URL `http://hu.opensuse.org/`.

[2] Az openSUSE projekt weblapja. URL `http://opensuse.org/`.

[3] SUSE Linux Enterprise Desktop termékismertető.
URL `http://www.novell.com/hungary/termekek/linux/sled10/`.

[4] SUSE Linux Enterprise Server termékismertető.
URL `http://www.novell.com/hungary/termekek/linux/sles10/`.

(x)