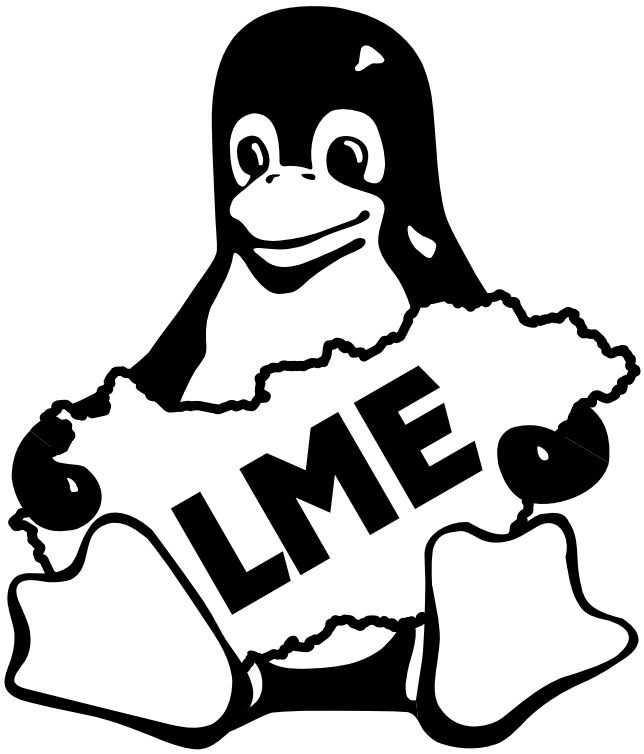
IV. GNU/Linux Szakmai Konferencia



Budapest, 2002. november 9.

A kiadvány tördelése a TEX 3.14159 verziójával készült,
GNU/Linux operációs rendszeren.
A TEX az American Mathematical Society bejegyzett védjegye.

Szerkesztette: Zelena Endre *ezelena@lme.linux.hu*

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

Tartalomjegyzék

A konferencia támogatói

Fő támogató: **UHU-Linux Kft.**

Arany fokozatú támogatók:

- Oracle Hungary Kft.
- Sun Microsystem Kft.
- Computerworld Számítástechnika

Támogatók:

- BalaBit IT Kft.
- LSC Kft.
- Mission Critical Linux Kft.
- Multiráció Kft.
- Pilatus-Comp Kft.
- Software Station

Előszó

Üdvözöljük a már hagyományosnak nevezhető rendezvényen, melyet a Linux-felhasználók Magyarországi Egyesülete ez évben már negyedik alkalommal rendezett meg, s ennek eredményeként elkészülhetett ez a mintegy 150 oldalas, a Konferencia tervezett előadásait tartalmazó kiadvány.

Mivel az előadásokkal kapcsolatos felhívásunkra jelentkezett előadók száma örvendetesen oly magas volt, hogy (sajnos) nem volt lehetőségünk minden előadást megtartani, így e kiadvány több előadásanyagot tartalmaz, mint a Konferencián ténylegesen elhangzottak.

Az idei Konferencia reményeink szerint merőben más lesz mint az eddigiek. Nem csak szakmai tartalmában, hanem helyszínében is.

Ez évben ugyanis különleges helyszínt választottunk, miáltal az üzleti világ, és az Informatikai vezetők fokozottabb érdeklődésére is számítunk. Az érdeklődők több szekcióban hallgathatják az előadásokat.

Egyesületünk 1998 őszén azzal a céllal jött létre, hogy összefogja a Linux-szal foglalkozó szakembereket és vállalatokat, szakmai fórumokat teremtsen, széleskörűen terjessze a Linux-szal kapcsolatos ismereteket, jogi személyiségként képviselje a Linux-hívők hazai társadalmát.

Négy éves működésünk alatt sok eredményt könyvelhettünk el, és megelégedésükre szolgálhat az a tény, hogy folyamatosan képesek vagyunk a fejlődésre.

A hullámzó, de éves szinten növekvő taglétszámmal együtt az Egyesület szerepe folyamatosan nő.

Reméljük, hogy az előadások, illetve a kiállítások találkoznak érdeklődésével, és az Egyesület jövőbeni rendezvényein is viszontlátjuk.

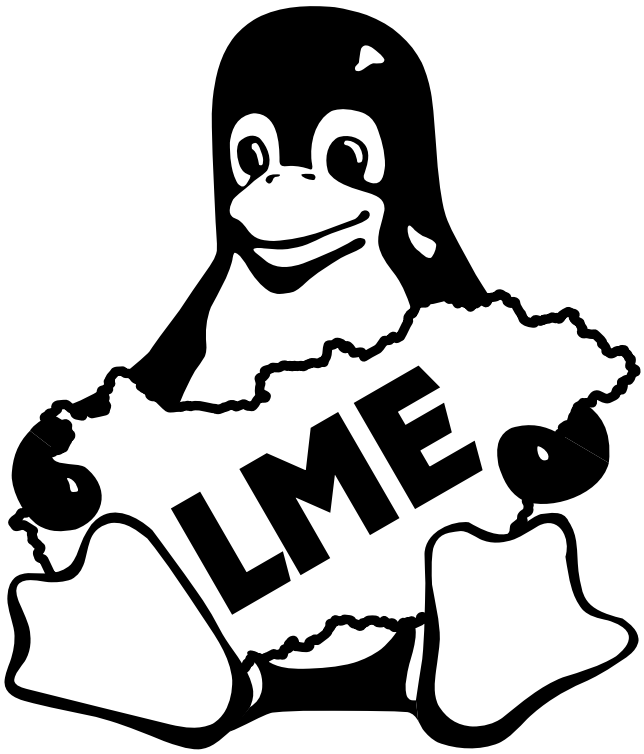
Észrevételeit, javaslatait, kérjük ossza meg velünk a Konferencián személyesen az LME standján, vagy később elektronikus levélben az *info@lme.linux.hu* címen.

E rövid bevezető zárásaként azt kívánjuk, hogy forgassák haszonnal kiadványunkat, és találkozzunk 2003-ban is, a következő még nagyobb szabású Linux Szakmai Konferencián!

Tisztelettel,

A konferencia szervezői

Előadások



A Caesar-kódtól az SSH-ig – a rejtjelezés rövid története

Bodnár Csaba
<bocs@missioncriticallinux.hu>

2002.10.01.

Kivonat

A történelem során mindig voltak olyan társadalmi csoportok, akik szerették volna üzeneteik ill. jegyzeteik tartalmát titokban tartani: az uralkodók, politikusok és a katonai vezetők részéről már az ókorban jelentkezett ennek igénye.
Adataink biztonsága mára meghatározó jelentőségűvé vált – különösen, ha napi rendszerességgel az interneten dolgozunk: távoli kiszolgálóra ssh-val lépünk be, fontosabb adatainkat scp-vel másoljuk, cégünk telephelyei között titkosított VPN-csatornákat építünk ki és félve adjuk meg bankkártyánk adatait akkor is, ha tudjuk, hogy https protokollal kommunikál böngészőnk a kiszolgálóval.
Korunkat az információ korának nevezik, de nevezhetnénk ugyanígy az információbiztonság korának is, hiszen ha az információ a legfőbb érték, akkor annak védelme elsődleges fontosságú. Úgy gondolom, érdemes áttekinteni, hogyan jutottunk el eddig, hogyan fejlődött a titkosítás – amit főleg katonai körökben rejtjelezésnek is neveznek – az elmúlt évezredek során, illetve mi várható a jövőben.

Tartalomjegyzék

1. Szteganográfia – az üzenet elrejtése

Már az ókorban is tudták, hogy annak legegyszerűbb módja, hogy az ellenség ne tudja meg az üzenetet, az üzenet elrejtése. Míg a szteganográfia az üzenet létezését palástolja, addig a kriptográfia az üzenet tartalmát. Gyakran a kettő kombinációját használták: ha a futárnál meg is találták az elrejtett üzenetet, a tartalmát akkor sem tudták meg. Példák szteganográfiára:

- Hérodotosz: Demaratosz üres viasz írótábla deszkájára írja, majd viasszal fedi el üzenetét.
- Hisztaiaszeusz a küldönce fejét leborotváltatta, ráírta az üzenetet, majd megvárta, míg a haja kinő. Akkoriban még erre is volt idő…
- Kínaiak: selyemre, azt labdaccsá gyúrták, viasszal borították, majd lenyelte az üzenetvivő.
- kemény tojásban üzenet: timsó és ecet elegyével írtak rá, a fehérjén kirajzolódik az írás.
- láthatatlan tinta: növény (pl. pitypang) nedvével.
- Második világháború, Dél-Amerika: német ügynökök, mikropont módszer: fényképészeti eljárással egy oldalnyi szöveget pöttyé zsugorítottak.

2. Kriptográfia – az üzenet titkosítása

A kriptográfia két ága az átrendezés és a behelyettesítés.

2.1. Átrendezés

Gyakorlatilag anagrammát készítünk a szövegből. A kódoláshoz/dekódoláshoz egyértelmű módszerre van szükség. Rövid szöveg esetén könnyen megfejthető, hosszú esetén nehéz megfelelő, könnyen használható módszert találni. Példák átrendezésre:

- az iskolások fésűs módszere, rácsos módszere.
- a világ első katonai rejtjelező eszköze a spártai szkütalé: a szkütalé egy sokszögletű bot, amelyre szíjat tekertek és úgy írták rá a szöveget. i.e. 404-ben Lüszandrosz szkütalé segítségével kapott hírt, hogy a perzsák támadást akarnak indítani ellene.
- A II. világháború alatt a németek által használt Enigma rejtjelezőgép az átrendezést és a behelyettesítést egyaránt alkalmazta.

2.2. Behelyettesítés

2.3. Monoalfabetikus behelyettesítés

A monoalfabetikus legrégebbi legrégibb leírása a Káma-Szútrában (!) található, ami 64 művészet tanulmányozását írja elő a nők számára, ezek egyike a titkosírás művészete. Itt az egyik ajánlott módszer az ábécé betűinek találomra történő párosítása.

Julius Caesar gyakran használt behelyettesítéses kódot: minden betű helyett az ábécében utána következő harmadikat írta le. Ezt Caesar eltolásos ábécéjének vagy egyszerűen Caesar-kódnak nevezzük.
A monoalfabetikus behelyettesítéses kódábécé egyszerűséget és megbízhatóságot jelentett évszázadokon keresztül.

2.4. A monoalfabetikus kód megfejtése: Gyakoriságelemzés

I.sz. 750-től az iszlám kultúra aranykorát éli, felvirágzott a tudomány és a művészet, a titkosírásokat napi rendszerességgel használták a közhivatalokban. Így nem csoda, hogy elsőként az arabok ismerték fel a monoalfabetikus behelyettesítéses kód megfejthetőségét gyakoriságelemzéssel.
Minden nyelvre jellemző, hogy az adott nyelven leírt szövegben a betűk milyen gyakorisággal fordulnak elő. Így kellően nagy mennyiségű szöveget gyakoriság szempontjából elemezve ki tudjuk következtetni, hogy a kódszöveg valamely jele az ábécé mely betűjének felel meg. Ezt a reneszánsz-kori Európában is ismerték, és az olasz városállamok, a Vatikán, valamint az egyes uralkodók diplomáciai levelezésében is használták. (És azok megfejtésében is…) Mária skót királynő titkos levelezését is ezzel a módszerrel fejtették meg, így fény derült Erzsébet királynő ellen szőtt összeesküvésükre, Máriát pedig kivégezték.

Összegzésül elmondhatjuk, hogy a gyakoriságelemzés meggyengítette a monoalfabetikus kódolás nyújtotta biztonságot.

2.5. Homofonikus behelyettesítéses kód

A homofonikus kód olyan monoalfabetikus kód, melyben az egyes betűknek (általában a gyakran használtaknak) több sifréje (helyettesítése, kódja) is lehet, amelyek száma az adott betű gyakoriságához igazodik. Ha az „a” gyakorisága 8%, akkor akár 8 jelet is rendelhetünk hozzá.

Sokáig megfejthetetlennek tűnt XIV. Lajos, a Napkirály grand chiffre-je, „nagy kódja”. Ezzel titkosították legfontosabb üzeneteit, haditerveit, politikai „húzásait”. Csak évszázadokkal később, 1890-ben fejtette meg Étien Bazeries őrnagy három év alatt. Itt is homofonikus kódról volt szó, de nem a betűk, hanem a szótagok voltak kódolva.

2.6. Polialfabetikus behelyettesítés (Vigenére-sifre)

Az ötlet: ne egy, hanem több kódábécét használjunk. Alberti, Trithemius és Porta munkáit Blaise de Vigenére kovácsolta új, egységes és erős kódrendszerré az 1560-as években. Nem egy, hanem 26-féle kódábécé, amelyek mindegyikét egy hellyel eltolják az előzőhöz képest. A titkosítás módja:

- Vigenére-tábla elkészítése,
- a nyílt szöveg minden betűjét társítjuk a kulcsszó megfelelő betűjéhez (pl. fölé írjuk),
- a táblából kikeressük a megfelelő kódot.

Előnye, hogy gyakoriságelemzéssel megfejthetetlen. Rengeteg kulccsal alkalmazható. Hátránya, hogy sokkal bonyolultabb használni, ezért kétszáz évre feledésbe merült, helyette inkább homofonikus kódolást használtak.

2.7. A polialfabetikus kód megfejtése: Babbage

Charles Babbage angol matematikus sokféle dologgal foglalkozott: sebességmérő, „bölényhárító”, statisztika, mortalitási táblázatok, stb. A számológépével a mai számítógépek alapjait is ő rakta le. Babbage egy bristoli fogorvos hatására kezdett el foglalkozni a Vigenére-sifre megfejtésével: rájött, hogy ha a kódolt szövegben ismétlődések vannak, ezek segítségével – megfelelően hosszú szöveg esetén – meg tudta állapítani a kulcs hosszát. Ha a kulcs pl. 5 betűből áll, akkor a feladat visszavezethető 5 monoalfabetikus szöveg elemzésére, ami gyakoriságelemzéssel megoldható.

3. Az I. és a II. világháború kriptográfiája

3.1. A rádió megjelenése

A rádió előnye a kommunikáció felgyorsítása, hátránya a lehallgathatóság. Megjelenésével megnőtt a titkosításra való igény. A rejtjelfejtők legnagyobb problémájává a kódszövegek hatalmas mennyisége vált. Korábban az elfogott üzenet ritka volt, a rádió megjelenésével a rejtjelfejtőkre a kódszövegek valóságos áradata zúdult.

3.2. Az ADFGVX

Az első világháborús német ADFGVX-kód egy monoalfabetikus behelyettesítéses és egy átrendezéses módszer együttes alkalmazását jelentette. A francia Georges Painvinnek sikerült feltörnie.

A világháború idején Zimmermann német külügyminiszter a mexikói elnöknek küldött rejtjelezett táviratot, melyben arra próbálja rávenni, hogy támadja meg az Egyesült Államokat és erre beszélje rá Japánt is. Az angolok a táviratot elfogták és megfejtették, majd publikálták. Ennek hatására lépett be az Egyesült Államok a háborúba.

3.3. Egyszeri kulcsos módszer

A Vigenére-sifre alapvető gyöngesége az, hogy ciklikus jellegű, segítségével azonban egy sokkalta erősebb módszer készíthető:

- Legyen a kulcs egyenlő hosszú a kódolandó szöveggel. Ez kivédi a ciklikusságból adódó gyengeséget. A rejtjelfejtő továbbra is megpróbálhat gyakran használt szavakat a nyílt szövegbe beilleszteni és így értelmes részeket találni a kulcsban (vagy fordítva), ami akár a teljes szöveg megfejtéséhez is vezethet!
- Ennek megoldására legyen a kulcs teljesen véletlenszerű (random).

Matematikailag igazolható, hogy az egyszeri kulcsos módszer valóban abszolút megbízható. A gyakorlati alkalmazásával azonban vannak problémák, ugyanis rengeteg kulcsra van szükség (nem „reciklálható”): Nehézkes volt a random kulcsok előállítása, a kulcsok szétosztása (kódkönyvek) pedig sokszor áthidalhatatlan nehézséget okozott. Emiatt alig-alig használták az egyszeri kulcsos módszert.

3.4. A titkosítás gépesítése

Kódtárcsa: Leon Alberti találta fel a XV. században. Két korong, egyik valamivel nagyobb, mint a másik. A kisebbet a nagyobbra helyezzük, közös tengellyel összefogjuk

és ráírjuk mindkettő szélére körben az ábécét. Segítségével egyszerű Caesar-kódú üzeneteket sifírozhatunk. Ha mindegyik betű kódolása után elfordítjuk a tárcsát egy megfelelő kulcs szerint, akkor polialfabetikus kódot is generálhatunk. A kódtárcsa megkönnyítette a kódolást.

3.5. Az Enigma

A II. világháború alatt mind a központi hatalmak (a német Enigma ill. Lorenz SZ40 vagy a japán Purple), mind a szövetségesek (a brit Typex ill. az amerikai SIGABA) használtak retjelezőgépeket. A németek által a második világháború előtt és alatt használt Enigma-rejtjelezőgép (Arthur Scherbius találmánya) a fenti kódtárcsa elektromos változatán alapul:

- Az eredeti modellbe három, fix huzalozású tárcsát építettek be, a tárcsák kivehetők és egymással is felcserélhetők voltak (polialfabetikus kód).
- Kapcsolótábla, mellyel betűket lehet felcserélni (hat betűpárt) (átrendezéses módszer).
- Gyűrű, amely szintén befolyásolja a sifírozást.

A kulcs a tárcsák „alapbeállítása” és a kapcsolótábla beállítása volt, ezeket kódkönyvekben négy hetente kapták meg és naponta változtatták.

3.6. Az Enigma-kód feltörése

Lengyelország már a harmincas években tartott egy esetleges német lerohanástól, így megalakították a Biuro Szyfrów-t, a kódirodát. Ciezki százados vezetésével sikerült feltörniük az Enigma kódját: rendelkezésükre állt egy kereskedelmi verziójú Enigma (a katonaitól különböző huzalozással) és megszereztek olyan dokumentumokat, amelyből előállíthatták a katonai verziót. A németek a kódkönyv kódjait „fő kódkulcsként” használták, amellyel az egyes üzenetek egyedi kulcsait kódolták. Az egyedi kulcsokat kétszer egymás után írták be, – elkerülendő a tévedéseket és a rádióinterferencia okozta hibákat – és ez vált a rendszer gyenge pontjává. Az egyik kódfejtő, Marian Rejewski, ebből olyan összefüggéseket állapított meg, amelyből kiindulva egy évnyi munka eredményeképpen meg tudták fejteni a kódot és évekig dekódolni tudták az üzeneteket. Kódfejtő-gépet is konstruáltak hozzá (Rejewski-bomba). A németek egy idő után változtattak az Enigmán, ezt még tudták követni a lengyelek, de amikor elkezdtek három helyett öt tárcsát használni, majd a kapcsolótáblákon hat betűpár helyett tízzel kellett számolni, akkor feladták a dolgot. Néhány héttel Lengyelország lerohanása előtt átadták a megszerzett információkat, tervrajzokat és egy Enigma-másolatot a franciáknak és az angoloknak.

Az angolok a Bletchley Parkban a lengyel módszerekkel és jóval több pénzzel, így technikával meg tudták fejteni a nehezebb német kódot is, így rendszeresen értékes hadi értesülésekhez jutottak. A csoport tagja, Alan Turing matematikus (ld. Turing-gép) közben olyan összefüggéseket tárt fel a kódszövegben, amelynek segítségével – ha van az embernek némi támpontja – több Enigmát hurokba kötve záros határidőn belül megfejthető volt a keverőtárcsák aktuális beállítása, vagyis maga a kulcs. Ezen az elven alapuló komplett dekódoló gépeket (Turing-bombákat) építettek, így amikor a németek újra „csavartak egyet” az Enigma kezelésén (nem duplikálták többé a kulcsokat), követni tudták a változást. Sokak szerint a Bletchley Park tevékenysége nagyban

hozzásegítette a szövetségeseket a háború megnyeréséhez, de legalábbis lerövidítéséhez.

3.7. A navaho-titkosítás

A II. világháború legtrükkösebb kódolási módszerét az amerikaiak találták ki és használták a csendes-óceáni térségben: navaho indiánokat alkalmaztak rádiósként, akik anyanyelvükön tartották a kapcsolatot egymással. A navaho nyelv egyetlen ismert nyelvcsaládba sem tartozik bele, korábban nagyon kevesen tanulmányozták, így ideálisnak tűnt a feladatra. A „navaho-titkosítást” a japánok soha nem tudták megfejteni.

4. Számítógépes kriptográfia

4.1. Az első szabvány (DES)

Horst Feistelt – amerikai német emigráns lévén – a háború alatt, majd később sem hagyta az NSA érvényesülni, pedig ő mindig is kriptográfiai kutatásokkal akart foglalkozni. Végül a hetvenes évek elején megalkothatta az IBM-nél a Lucifer rendszert. A titkosítás egy igen képletes leírás szerint a következőképpen néz ki:

„A titkosítási eljárás kicsit a dagasztáshoz hasonlít. Képzeljünk el egy hosszúra nyújtott tésztamasszát, amelyre valamilyen üzenetet írtak. A tésztát először 64 centiméteres darabokra vágják, ezt fölszabdalják, összegyúrják, hozzácsapják a következő darabhoz, és megint összegyúrják. Ezt a műveletet ismétlik újra meg újra, míg az üzenet teljesen össze nem gabajodik. 16 dagasztás után elküldik a kódszöveget, a címzett pedig a művelet fordított irányú végrehajtásával kibogarássza a tartalmát.”

A titkosításhoz tartozik egy kulcs is, az üzenet a választott kulcstól függően milliárdnyi módon sifrírozható. 1976-ban a Lucifer vált szabvánnyá *DES* (Data Encription Standard) néven.

4.2. Az aszimmetrikus kulcs

Továbbra is komoly probléma volt azonban a titkos kulcsok eljuttatása a túloldalra. A rendszeresen titkosított adatokkal dolgozó szervezetek költségvetésében komoly pénzügyi tételekként jelentek meg az ezzel kapcsolatos kiadások. A hetvenes években Diffie, Hellman és Merkle a Stanford egyetemen talált egy módszert arra, hogyan lehet nyilvánosan kulcsot cserélni. Nem sokkal később Diffie kitalálta az ún. aszimmetrikus kulcsot. Itt tulajdonképpen két kulcsról van szó: az egyikkel sifrírozni lehet, a másikkal desifrírozni. A desifrírozó kulcsot hívjuk ma privát kulcsnak, a sifrírozó pedig a nyilvános kulcs. A nyilvános kulccsal kódolva bárkinek küldhetünk titkos üzenetet, azt csak ő maga, a privát kulcsával tudja majd elolvasni.

4.3. RSA

Diffie csak magát az elvet találta ki, a gyakorlati megoldásra az RSA-„hármas” (Rivest, Shamir, Adleman) jött rá 1977-ben. Módszerük szerint a nyilvános kulcs két elég nagy prímszám szorzata, a privát kulcs pedig maga a két prímszám. Mivel a prímtényezős felbontás – kellően nagy számok esetén – hosszadalmas, időigényes feladat, ez garanciát jelent az adatok biztonságára nézve.

Nemrég kiderült, hogy ugyanezt a dolgot a brit biztonsági szolgálatnál már néhány évvel korábban kitalálták: az elvet James Ellis, a gyakorlati megvalósítást pedig Clifford Cocks és Malcolm Williamson dolgozta ki.

4.4. PGP és GPG

A nyolcvanas években Phil Zimmermann – aki azon a véleményen volt, hogy mindenkinek joga van a saját személyes adatai védelmére – kifejlesztett egy könnyen kezelhető, gyors titkosító szoftvercsomagot, melyet PGP-nek (Pretty Good Privacy) nevezett el. A PGP az RSA-t és egy IDEA nevű szimmetrikus kódolást egyaránt tartalmazott: a sebesség érdekében az RSA-t csak az IDEA kulcsának kódolására használta. A szoftver emellett digitális aláírás készítésére is használható volt.

A programot freeware-ként felrakta egy publikus helyre, ahonnan rengetegen letöltötték és kezdték el használni. Emiatt először a szabadalom jogtalan felhasználásával, majd illegális fegyverexporttal vádolták. Végül nem ítélték el és – miután az RSA Security-val megállapodott – a PGP azóta is terjed, nem kereskedelmi célokra szabadon letölthető. Maga az alapkérdés azonban még mindig nyitott, mi a fontosabb, a kriptográfia szabadság és az egyéni jogok védelme, vagy a bűnüldöző szervek lehetősége a betekintésre. A döntő tényező valószínűleg mindig az lesz, mitől fél jobban a közvélemény: a bűnözéstől vagy az államtól.

Bár a PGP magán használatra ingyenes, üzleti célúra azonban nem. Ezt az „apró problémát” küszöböli ki a GPG vagy GnuPG (Gnu Privacy Guard), ami az OpenPGP szabvány (RFC 2440) nyílt forráskódú megvalósítása. Mivel az eredeti PGP-ben lévő IDEA algoritmus licensszel védett, helyette válogathatunk az ingyenes algoritmusok bő választékából (ElGamal, DSA, AES, 3DES, Blowfish, Twofish, CASTS, MD5, SHA-1, RIPE-MD-160 és TIGER).

4.5. TripleDES

A kilencvenes évek elejére az 56-bites DES algoritmus már nem volt elég biztonságos. Számítani lehetett arra, hogy egyszerre akár több ezer számítógépből álló klaszter próbálja feltörni titkos kódunkat. Ekkor jött a TripleDES, amely három, egymástól független kulcsot használ, így megháromszorozza a lehetséges kulcsok számát $(5, 2 \cdot 10^{33})$, így a feltöréshez szükséges erőfeszítéseket is. Az évtized közepére ez sem volt elég jó, bár a biztonsága megfelelő volt, más problémák voltak vele: túl lassú volt, a sávszélességek megnövekedtek, így egyszerre a korábbinál jóval nagyobb mennyiségű adatot kellett megfelelő sebességgel titkosítani. Az NSA/NIST így pályázatot írt ki az új szabvány (Advanced Encryption Standard, AES) megtervezésére.

4.6. Az új titkosítási szabvány (AES)

A pályázat első körébe 15 fejlesztői csapat jutott be, köztük egészen nagy (RSA Labs) és egészen kis (a puerto ricoi illetőségű Georgoudis FROG nevű algoritmusával, mely első ilyen jellegű próbálkozása volt) nevekkel. A második körbe öten jutottak be, közülük az ismertebbek:

- Az RSA Laboratories egy mindössze néhány soros, nagyon erőteljes algoritmussal, amely azonban túlságosan processzorigényes volt.
- TwoFish algoritmus az ismert BlowFish algoritmus fejlesztőitől.

A győztes a Rijndael algoritmus lett, két flamand kriptográfus, Daeman és Rijman tervezésében. Hogy lehet, hogy egy belga algoritmus lett az új amerikai titkosítási szabvány? Előnyös tulajdonságai miatt, úgymint: könnyű szoftveres implementálhatóság, a komponensek általános áttekintehetősége, alacsony erőforrásigényű hardverimplementációs tapasztalatok.

5. Alkalmazások

Az internet használatával megnőtt az igény a biztonságos kommunikációt lehetővé tevő programokra. Ejtsünk most néhány szót a legelterjedtebb alkalmazási módokról. A PGP-ről korábban – Phil Zimmermann tevékenysége kapcsán – volt már szó.

5.1. SSH és OpenSSH

Tatu Ylönen finn diák 1995-ben kifejlesztette az SSH első verzióját, megnyugtató biztonságú alternatívaként az addig előszeretettel használt telnet helyett, a telnet protokoll ugyanis nemhogy a kommunikációs csatornát, hanem magát a felhasználói nevet és jelszót sem titkosította. Mindenki SSH-t kezdett használni, mindaddig, amíg Tatu meg nem alapította az ssh.com nevű biztonságtechnikai céget, az ssh forrásának licencelését pedig úgy módosította, hogy azt már nem igazán lehetett „nyílt forrásúnak” nevezni. Mint sok más fejlesztőt, az OpenBSD fejlesztői csapatot is nagyon zavarta ez a dolog, hiszen az OpenBSD-ben szerverként és kliensként egyaránt használták már az ssh-t. Fogták hát a legutolsó szabad forrású verziót és azt fejlesztették tovább OpenSSH néven. 2000 májusára az OpenSSH már az új, SSH2 szabványt is támogatta. Csak „licencmentes” titkosítási algoritmust használ, úgymint 3DES, Blowfish, CAST128, Arcfour and AES. A kezdeti kulcscsere RSA-val vagy DSA-val történhet.

5.2. SSL

Az SSL (Secure Sockets Layer) a web-alapú kommunikáció titkosítására szolgáló szabványos módszer, melyet a Netscape fejlesztett ki. Az SSL a következő szolgáltatásokat nyújtja egy TCP/IP kapcsolat esetén: adattitkosítás, szerver authentikáció, üzenet integritásellenőrzés, opcionális kliens authentikáció. Az SSL-t az összes ismert web kiszolgáló és böngésző szoftver ismeri és támogatja, de SSL-be mi magunk is becsomagolhatunk bármit, pl. IMAP-et vagy POP3-at. Az SSL a következő algoritmusokat támogatja: DES, DSA, KEA, MD5, RC2, RC4, RSA, SHA-1, SKIPJACK, Triple-DES.

Érdemes külön szót ejteni a szerver authentikációról. Ennek segítségével a böngészőnk megbizonyosodhat arról, hogy az elérni kívánt szerver valóban az-e, akinek mondja magát, egy erre feljogosított független állami vagy magán hatóság (certification authority, CA) szerverének segítségével. Ugyanez lehetséges a másik irányba is, vagyis hogy a kliens igazolja magát (opcionális kliens authentikáció), de ezt ritkán használják.

5.3. IPSEC

Míg az SSL az átviteli szinten (transport layer), addig az IPSEC a hálózati szinten (network layer) dolgozik. Az SSL két alkalmazás, az IPSEC pedig két számítógéphálózat között biztosít biztonságos kommunikációs csatornát. Az IPSEC-et protokollt az IETF

(Internet Engineering Task Force) fejlesztette ki, része lesz az a TCP/IP következő generációjának, az IPV6-nak, de a jelenlegi IPV4 alatt is széles körben használják. Az IPSEC-re alapozva virtuális magánhálózatok építhetők ki az interneten keresztül pl. cégünk telephelyei között. Egyik legelterjedtebb nyílt forráskódú implementációja a freeswan.

6. A jövő a kvantumkriptográfia?

Az RSA nagyon erős kód, nagyméretű kulcsok esetén feltörése csak a prím-faktorizáció valamilyen új matematikai módszerrel történő felgyorsításával lenne lehetséges, erre azonban nincs sok esély. A kódtörők újabban a kvantumfizikában reménykednek, jóllehet, a kvantumszámítógép elve „dacol a józan ésszel”. Ennek elméletét David Deutsch brit fizikus dolgozta ki. A kvantumszámítógép egyszerre hihetetlen mennyiségű számítás elvégzésére lesz képes, megépítéséig azonban még jónéhány elméleti és gyakorlati akadályt le kell győzni. A kutatók között sincs egyetértés abban, megépíthető-e belátható időn belül vagy sem. Peter Shor és Lov Grover a Bell Laboratóriumban már programot is írtak – kvantumkomputeren történő futtatásra. Ha a kvantumszámítógép egyszer elkészül, újra a kódfeltörők lesznek előnyben, hiszen azzal az RSA pillanatok alatt feltörhető lesz.

Ők azonban már dolgoznak a kvantumelméleten alapuló titkosítási módszeren: Stephen Wiesner ötlete alapján Charles Bennett és Gilles Brassard a fény polarizációján alapuló módszert dolgozott ki és túl van több sikeres teszten, vagyis fénykábelen ill. a levegőben sikerül átjuttatni polarizált fénnyel kódolt adatokat biztonságosan.

7. Hivatkozások

Simon Singh: Kódkönyv, A rejtjelezés és a rejtjelfejtés története
Bruce Scneier: Applied Cryptography
David Kahn: The codebreakers
Daniel J. Barrett, Ph. D. and Richard E. Silverman: SSH: The Secure Shell

`http://www.rsa.com/`
`http://www.pgp.com/`
`http://www.gnupg.org/`
`http://www.openssh.org/`
`http://www.ssh.com/`
`http://www.freeswan.org/`
`http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf`
`http://www.nist.gov/aes/`

E-mail SPAM védelem

Czakó Krisztián
<slapic@linux.co.hu>

2002. október 21.

Kivonat

Az Internet fejlődésével megjelentek a hagyományos postaládáinkban már ismerté vált kéretlen reklám-anyagok. Míg a való életben ezek mennyiségét az előállítás érzékelhető költsége kordában tartja, az Interneten az e-mail olcsó és gyors célba juttathatósága szinte korlátlan lehetőségeket biztosít az erre szakosodott marketing vállalkozásoknak. Ugyan törvényi szabályozás már létezik sok országban, az elkövetők általában lenyomozhatatlanok, így jogi úton szinte lehetetlen védekezni ellenük. Nincs sajnos más megoldás, mint felvenni velük a harcot, és megpróbálni kiszűrni leveleinkből a szemetet.

A legegyszerűbb szűrési megoldás, hogy ránézünk levélre, és szinte azonnal tudjuk, hogy ez egy kéretlen reklám, azaz SPAM. Ez mindaddig működik, míg napi pár ilyennel találkozunk. Akik már feliratkoztak nyilvános levelező listákra, írtak webes fórumokra tudják, hogy e-mail címüket ezen helyekről azonnal begyűjtik, és ettől kezdve nincs ami visszatartsa a gátlástalan „spammereket”. Ilyenkor már a kézi megoldás nem megfelelő. Szerencsére találunk több, nyílt forráskódú programot, melyet akár felhasználóként, akár rendszergazdaként munkára foghatunk, és amelyek igen jó hatékonysággal szűrik ki a SPAM-et.

Tartalomjegyzék

1. Védelmi megoldások

Ahhoz, hogy egy szoftver megállapíthassa egy-egy levélről, hogy az SPAM-e, kétféle eljárást alkalmazhatunk. Az egyik elterjedt módszer, hogy nem a levelet, hanem a küldő számítógépet (szervert) vizsgáljuk. Rendelkezésünkre áll a gép IP-címe, így ezt felhasználhatjuk. Erre az elvre épül az RBL (Realtime Blocking List) rendszere. A másik megoldás, ha a vírus-ellenőrzőkhöz hasonlóan a levelet magát vizsgáljuk. Ez utóbbira, azaz a tényleges tartalomszűrésre is számos jó megoldás született. Lássuk részletesebben az említett két módszert.

1.1. RBL

Az RBL lényege, hogy nyilvántartjuk azokat, akik valami olyat tettek, amit nem szabad. Azaz ez egy feketelista. Az RBL indulásakor három ilyen lista létezett: spamküldők listája, nyílt továbbító rendszerek (open relays) és a közismerten változó IP számot használó betárcsázók listája. Utóbbiak nem a tettük miatt kerültek lajstromba, hanem mert a változó IP miatt nem lehet visszakövetni a tényleges elkövetőt. Számukra ez csak akkor probléma, ha a szolgáltatójuk nem üzemeltet olyan e-mail szervert, melyen át elküldhetik leveleiket. Ez utóbbi viszont igen ritka.

Az idők során az RBL sokat változott. Az eredeti RBL üzleti célúvá válásakor megjelent számos szabadon használható alternatíva, saját adatbázisokkal. Így egyre nehezebbé vált kiválasztani, melyik a jó és melyik a rossz adatbázis. Márpedig ez az egyik „kulcskérdése” a rendszernek, mondhatni az egyik gyengéje. Az adatbázisba történő bekerülés és kikerülés lépései kulcskérdést jelentenek. Ha túl könnyen felvesznek valakit az adatbázisba, vagy túl nehezen lehet onnan kikerülni az probléma lehet, de lehet gond az ellenkezője is. Ahhoz, hogy az adatbázisokat világszerte használni lehessen, azoknak megbízható, pontos adatokat kell tartalmazniuk, így üzemeltetésük nehéz.*

Az induláskor meglévő három adatbázis típus mellett megjelentek újabb, új szempontok alapján működő rendszerek. Ilyenre példa a WHOIS adatbázisok adatait elemző megoldás. Ebbe akkor kerül be egy cím, ha a WHOIS adatbázis, melynek léteznie kell minden címhez nem létezik vagy hibás (pl. rossz a kapcsolattartó e-mail címe).

Az RBL sikerének titka az egyszerű működési elv: az adatbázis egy szabványos DNS szerver, melyben az IP számokat lehet keresni, azaz RR típusú rekordokat. Amennyiben az IP rendelkezik RR-el, az IP szerepel az adatbázisban. Az adatbázis tartalmazhat egy kiegészítő TEXT mezőt, mely leírhatja, miért. Az egyszerűségéből kifolyólag hamar implementálták számos levelezőszerver szoftverben, így használható az összes elterjedt nyílt forráskódú SMTP szerveren is (pl. Postfix, Qmail, Sendmail, Exim, Courier). Programtól függően lehetőségünk van az adatbázis alapján elutasítani a levél átvételét vagy a levél fejlécében megjegyezni, hogy a küldő valamely RBL adatbázisban megtalálható.

*Néhány jellemző RBL adatbázis:
http://balckholes.mail-abuse.org/
http://relays.mail-abuse.org/
http://dialups.mail-abuse.org/
http://relays.ordb.org/
http://ipwhois.rfc-ignorant.org/
http://orbs.dorkslayers.com/
http://relays.osirusoft.com/

1.2. Tartalomszűrés

Míg az RBL a levél átvétele előtt elutasíthatja azt, tartalomszűrésnél meg kell várnunk a levél megérkezését, így itt a felesleges hálózati forgalom elkerülése nem megoldott. A tartalomszűrés egy nehéz feladat, mivel a SPAM levelek változnak, fejlődnek. Itt is felmerül az adatbázis alkalmazása, de lehetőség van szövegelemző módszerekre is. A megoldások mutatnak példát mindkettőre, ráadásul az adatbázis alkalmazására is több szép megoldást találunk.

Általános jellemzője a tartalomszűrőknek, hogy megpróbálják a levél törzsét és fejlécét oly módon elemezni, hogy a jellemző személyre szabottság és hasonló apróbb eltérések kiküszöbölhetőek legyenek.

Distributed Checksum Clearinghouse (DCC)[1]

Az általam talált egyik legjobb megoldás a SPAM kiszűrésére. Adatbázist használ, szövegelemzést nem végez. Pozitív tulajdonsága, hogy lehetővé teszi számos adatbázis együttműködését, így csökkentve az ellenőrzések okozta hálózati forgalmat.

A DCC elvi háttere, hogy lehetővé teszi a felhasználók számára leveleik összehasonlítását. Az Interneten üzemelő több mint 70 szerver számlálja a különböző ellenőrző összegeket, mely adatokat a kliens számára elérhetővé tesz. A kliens a szervertől megtudja, hogy az adott levél hány DCC-t használó helyre (felhasználóhoz vagy szerverre) érkezett meg, és ez alapján hozhat döntéseket.

Működésének lényege, hogy a DCC-t használó felhasználók illetve szerverek összes leveléről többféle eljárással ellenőrző-összeget készítünk, majd ezeket elküldjük a legközelebbi DCC szervernek. Az ellenőrző-összeget olyan algoritmusokkal készítjük, mely lehetőleg azonos eredményt ad a minimálisan eltérő (pl. személyre szabott) levelek esetén. A DCC szerver gyűjti az ellenőrző összegeket, és csak azokat, azaz a levélről semmi mást nem küldünk be. A DCC szerverek kicserélik egymást közt azon ellenőrző összegeket, melyekből az adott szerveren sok gyűlt össze. Ez szintén csökkenti a hálózati forgalmat, mert a kevésszer előforduló valószínűleg helyi (így nem, vagy csak helyi SPAM) levelek adatai nem kerülnek továbbításra. A DCC kliens az ellenőrző-összeg beküldésekor a szerver válaszában megkapja, hogy eddig hányszor jelentették ugyanazt. A SPAM egyik jellemzője, hogy sokezer vagy akár sok millió címre is elküldik. Ha egy ellenőrző-összegből sok van, a kérdéses levél nagy valószínűséggel SPAM. A DCC-t használó SMTP szerver vagy felhasználói program eldöntheti, hogy levelet megjelöli mint SPAM, megsemmisíti vagy egyszerűen továbbítja. A megjelölt levelet a felhasználó levelezőprogramja vagy levélszűrője segítségével szintén egyedileg kezelheti.

A rendszer egyik gyengéje, hogy a nagy létszámú levelező listák esetén is SPAM-nek minősíthet leveleket, ezért ilyen esetekre „fehér-listát” kell készíteni.

A rendszer annál hatékonyabb, minél többen használják. Nem igényel plusz munkát a felhasználótól, a levelek jelentése teljesen automatizálható. Kényelmes és hatékony módszer.

A DCC teljes egészében C-ben készült, így működése kis erőforrásigényű, de a DCC-szerver üzemeltetése napi 100-500 ezer levél esetén már komoly szervert igényel.

[1] http://www.rhyolite.com/anti-spam/

Vipuls's Razor[2]

A fentiekhez hasonlóan ellenőrző-összegekkel dolgozik, de a DCC-vel szemben nem minden e-mail kerül az adatbázisba. Csak a felhasználó által felismert SPAM-et kell beküldeni. Ennek előnye, hogy az első beküldés után már mindenki kiszűrheti az adott SPAM-et (a DCC-nél ehhez beállítástól függően akár többszáz jelentés is kellhet). Ugyanez hátrány is lehet, hiszen egy téves jelentés jóhiszemű e-mail levelek eldobását is okozhatja. Éppen ezért a most megjelenő v2 változatban már a felhasználóknak regisztrálniuk kell magukat, és lehetőségük lesz a jelentés visszavonására is.

A SPAM ellenőrzése itt is egyszerű és automatizálható, de ha aktív részesei szeretnénk lenni a rendszernek, regisztrálnunk kell magunkat a jelentések beküldéséhez, melyet csak kézzel illik megtenni. Itt is annál hatékonyabb a rendszer, minél többen jelentenek, de a lekérdezés itt egyben nem okoz jelentést is, így a rendszernek kevesebb aktív tagja van.

Perl script, így használata nagyobb erőforrást igényel.

Spamassassin[3]

Talán a fellelhető legösszetettebb megoldás. Elsősorban szövegelemzésen alapul, de külső programként képes használni a dcc és a razor szolgáltatásait valamint az RBL-t is. A program Perl modulként készült, és egy központi Perl script hívja a modulokat. Az egyes elemző funkciókat egy-egy önálló modulban valósították meg.

A minősítés pontozásos módszerrel történik. Az egyes elemző funkciók (így a külsők is, mint a DCC vagy Razor) plusz vagy mínusz pontokat jelentenek. A végső döntést pedig a rendszergazda által beállítható ponthatár alapján hozza meg a rendszer. Az alapértelmezés 5 pont, mely az esetek többségében helyes döntést eredményez. A döntés eredményét a program a levél fejlécében, tárgyában és törzsében képes jelenteni. Számomra a kényelmes megoldást a csak fejlécben történő jelentés adta, mivel így a levél törzse nem változik meg, és levélszűrővel kiválogatható a SPAM.

A program folyamatos fejlesztésével érik el, hogy az elemző rutinok pontosak legyenek. Saját tapasztalatom szerint ez a program minősített több olyan levelet is SPAM-nak, mely nem volt az. Ezen levelek vizsgálatánál kiderült, hogy a program döntése ennek ellenére jogos volt, mivel ezen levelek felépítése igen hasonlított a SPAM-ekére. Általában céges hírlevelekről volt ugyanis szó, amik hasonló céllal, de nem kéretlenül kerülnek kiküldésre, mint a SPAM-ek. Ilyen esetekben itt is a „fehérlista" alkalmazása jöhet szóba. A DCC azért nem minősítette SPAM-nek, mert sokkal kevesebb címzetthez jutott el a levél.

Hasznos tulajdonság az automatikus „fehérlista" (AWL), mely minden levél SPAM szintje alapján értékeli a feladót, folyamatosan követve, hogy az általa küldött levelek többsége SPAM vagy sem. Ez hosszú távon oda vezet, hogy csökkennek a hibás döntések, hiszen aki általában nem küld SPAM-et, az valószínűleg nem is fog, még ha egy-egy levele annak is tűnik, míg egy jellemző SPAM küldő (bár a spammerek címe még rövidtávon sem szokott állandó maradni) nem valószínű, hogy tisztességes levelet fog küldeni.

Lehetőségünk van figyelni a szöveg kódolását. Például ha általában magyarul levelezünk, plusz SPAM pontokat jelenthet ha a levél spanyolul vagy japánul íródott.

[2] http://razor.sourceforge.net/
[3] http://www.spamassassin.org/

További módszerek

A fentiek természetesen nem mutatják be a kialakult módszerek teljes skáláját. Léteznek a kevésbé bevált vagy elterjedt megoldások mellett új és ígéretes módszerek is. Talán az egyik ilyen új és igen jónak látszó tartalomszűrő megoldás az, amelyik a Bayes-féle feltételes valószínűségi tételen alapul [4].

2. Gyakorlati megoldások

A módszerek után lássuk hogyan használhatjuk ezeket az életben. Az RBL használatához szükség van a küldő címére, mely csak az SMTP kommunikáció során áll rendelkezésünkre, így azt csak ott lehet alkalmazni. A tartalomszűrő megoldások esetén már a felhasználó is alkalmazhatja a programokat, igaz azon funkciók, melyek csak az SMTP szervernél működnek (pl. a DCC IP szerinti szűrése) nem, vagy csak részben érhetőek el.

2.1. DCC

A DCC-t szerzője elsősorban sendmailhez ajánlja, így mindössze két kliens oldali változata létezik: a dccm, mely a Sendmail Milter rendszerére épül valamint a dccproc, mely a szerző szerint a procmail-hez íródott, de más (pl. maildrop) szűrő esetén is kiválóan működik (a spamassassin is ezt használja). A dccm teljes mértékben együtt tud a sendmaillel működni, azaz lehetőségünk van a levél SMTP során történő visszautasítására. Erre sajnos más levelezőszerver esetén nincs kész megoldás, bár a Courier szűrő rendszeréhez viszonylag kis munkával illeszthető kell legyen. Postfix esetén egy egyszerű scriptet készíthetünk, mely a dccproc-ot használja és a sendmail paranccsal küldi vissza a levelet a Postfixnek, ahonnan a script a content_filter opcióval kaphatja meg. Qmail esetén sincs kész megoldás, de az interneten találunk leírásokat, hogyan illesszünk a qmail-queue helyére dccproc-os szűrőt (azonos módszer, mint a Qmail-hez történő vírusellenőrző illesztése).

Amennyiben felhasználói szinten szeretnénk használni, a dccproc alkalmazható. Procmail esetén a legegyszerűbb megoldás, ha a felhasználó .forward vagy .qmail fájlból a procmail előtt a dccproc-nak adja a levelet (| dccproc | procmail). Ha maildropot használunk még egyszerűbb az élet, mivel a .mailfilter fájl elejére írt `xfilter "dccproc"` megfelel.

A DCC minden levél fejlécéhez egy X-DCC-*-Metrics: sort ír. A * helyén minden esetben a használt DCC szerver neve szerepel. Ez független attól, hogy az MTA vagy a felhasználó hívta-e meg a DCC-t. Ha felhasználóként futtatjuk a dccproc-ot, és az MTA-nk is megtette már ezt, az utóbbi DCC felülírja az előbbi bejegyzését. Ilyen esetekben arra is legyünk figyelemmel, hogy az MTA-nk már jelentette a levelet egy DCC szervernek, így azt mi ne tegyük meg ismét (-Q opció a dccproc-nál). A dccproc telepíthető felhasználóként is, ha a home könyvtárunkból tudunk programot futtatni. Használatában nincs jelentősége, hogy központilag vagy egyedileg történt meg a telepítés. Íme egy maildrop .mailfilter példa:

```
xfilter "$HOME/bin/dccproc.sh"
if (/^X-DCC-.*-Metrics: .* bulk/)
    to Maildir/.spam/
```

[4] http://www.paulgraham.com/spam.html
Köszönet Verők Istvánnak (vi@inf.bme.hu) a tippért.

Az xfilter csak akkor kell, ha az MTA nem végezte el nekünk a DCC ellenőrzést. A dccproc.sh tartalma:

```
#!/bin/sh
dccproc -c CMN,NEVER,50 -w ~/.dcc/whiteclnt
exit 0
```

A dccproc -c opciója adja meg, hogy hány találat esetén tekintjük a levelez SPAM-nek. Három adat tartozik hozzá: mely ellenőrzőösszegeket használjuk (CMN: általános, common), mikor naplózzunk, mikor tekintsük SPAM-nek. A „NEVER” megadása esetén az adott funkciót (a mi esetünkben a levél naplózását) kikapcsoljuk. Alapértelmezés szerint mindkét funkció kikapcsolt minden esetben. Az 50-es határ esetünkben annyit jelent, hogy ha az általános (body, fuz1, fuz2) ellenőrzőösszegek valamelyike legalább 50-et talált, az X-DCC fejlécsorba bekerül a „bulk” szó, valamint a dccproc 67-es vissztérési értékkel lép ki. Utóbbi a maildrop számára nem jó, ezért van a script végén exit 0. A -w segítségével adhatunk meg „fehérlistát” (whitelist). Itt tehetjük meg, hogy nagy forgalmú levelezőlistákat nem veszünk bele az ellenőrzésbe.

A fenti módszerrel minden SPAM-nek minősített levelet a Maildir/.spam/ fiókba teszünk, amit a kedvenc levelezőprogramunkkal (Mutt, Pine, stb.) vagy IMAP-on keresztül (pl. Courier-IMAP) tudunk olvasni. Ha már hosszú ideje alkalmazzuk a DCC-t, és a „fehérlistáink” jók, nem kerül véletlenül sem hibás pozitív a SPAM-ek közé, akár meg is szüntethetjük a tárolását a bulk levelek /dev/null-ba irányításával. A SPAM-ek esetében a feladónak történő visszaküldés sajnos értelmetlen szokott lenni. Itt jöhet jól a Sendmail és a dccm, mely az SMTP DATA végén azonnal visszautasítja a levelet. Legyen az a küldő szerver gondja.

Ezt én a sendmail 8.12.3 (Debian woody) változatával próbáltam ki. Telepítése pofonegyszerű. A dccm lefordítása és telepítése után a dccm misc/ könyvtárában található két m4 scriptet (dcc.m4, dccdnsbl.m4) kell a sendmail cf/features/ könyvtárába tenni, a sendmail.mc-hez hozzáadni a FEATURE(dcc) opciót, a dccm-et elindítani (/var/dcc/libexec/start-dccm), végül indulhat a sendmail. Ha a /var/dcc/dcc_conf-ban a DCCM_REJECT_AT változót beállítjuk, a sendmail nem csak megjelöli a spam-eket, hanem a beállított értéknél nagyobb dcc találat esetén visszautasítja:

```
<slapic@pilatuscomp.hu>: host mail.pilatuscomp.hu[212.52.161.233] said:
550 5.7.1 mail g8UChbUZ015403 from ::ffff:212.52.160.4 rejected by
wanadoo-be DCC.
```

2.2. Razor

A razor használata igen egyszerű. A razor-check program bemenetére küldött levelet ellenőrzi. A levelet nem adja tovább, azaz nem hívható köztesen procmail vagy maildrop előtt. A levelet nem módosítja. Ha a levél megtalálható az adatbázisban a visszatérési értéke 0, ha nem akkor 1. Ha az adatbázisba SPAM-et szeretnénk jelenteni, arra a razor-report parancs való, mely szintén a bemenetére várja a levelet. mindkét program elfogad fájlnevet is paraméterként.

Mint az elején írtam, az aktív (jelentést is küldő) felhasználók alacsonyabb száma miatt sokkal kevesebb SPAM elfogására találtam alkalmasnak a tesztek során. Ezért önálló használatát nem javallom, de mint kiegészítő nagy segítséget nyújthat. Számomra leginkább a Spamassassin-al együtt alkalmazva vált be.

2.3. Spamassassin

Kezelése szintén egyszerű, az igazi kihívást a pontozási rendszer kézi hangolása okozza, ha valaki ez utóbbira rászánja magát. Mivel a „gyári” beállítás ezügyben egészen jó, erre ritkán jut idő és energia. A működés a már ismert: a bemenetére várja az ellenőrizni kívánt levelet, és a kimenetén adja vissza a módosított levelet. Így könnyedén alkalmazhatjuk procmail vagy maildrop esetén, valamint a Postfix programmal is könnyen párosítható. Az újabb verziók tartalmazzák a spamd szervert, mely SMTP alapon tud működni, még egyszerűbbé téve a Postfix integrációt.

Ha nem SMTP-vel akarjuk hívni, Postfix esetén az alábbi módszert javasolom. Első lépésként hozzunk létre egy egyszerű scriptet (/usr/local/sbin/filter.sh):

```
#!/bin/sh
INSP_DIR=/var/lib/filter
SENDMAIL=/usr/sbin/sendmail
SPAMASSA=/usr/bin/spamassassin
REJ_MSG="Message content rejected"
# Exit codes from <sysexits.h>
EX_TEMPF=75
EX_UNAVA=69
cd $INSP_DIR || { echo $INSP_DIR does not exist; exit $EX_TEMPF; }
# Clean up when done or when aborting.
trap "rm -f in.$$; rm -f out.$$" 0 1 2 3 15
cat | $SPAMASSA -P -x -a > out.$$ || { echo $REJ_MSG; exit $EX_UNAVA; }
$SENDMAIL "$@" < out.$$
exit $?
```

Ezután adjuk a /etc/postfix/master.cf-hez az alábbi sort:

```
filter unix - n n - - pipe user=filter argv=/usr/local/sbin/filter.sh
                                        -f ${sender} -- ${recipient}
```

Állítsuk be a Postfix content_filter opcióját (content_filter=filter:). A legjobb hely erre a master.cf-ben az SMTP sor.
Az smtp szolgáltatás sorának végére beírt `"-o content_filter=filter:"` kiegészítés pont jó.

Ha ezzel megvagyunk, a Spamassassin beállítását is végezzük el, mielőtt a Postfix újraindításával bekapcsoljuk a szűrést. A beállítások általában a /etc/spamassassin/ könyvtárban vannak. Itt számos állomány van, mind egy-egy kétjegyű számmal kezdve. A számok a feldolgozási sorrendet határozzák meg. A később jövő opció felülbírálja az előbb jövőt. Ezen fájlokhoz a legjobb nem hozzányúlni, a local.cf kivételével. Ide írjuk a saját beállításainkat. Mivel az „l” betű később van az ASCII-ban mint bármely szám, ez lesz legutoljára végrehajtva, azaz itt bármit felülbírálhatunk. Íme egy példa a local.cf-re:

```
ok_locales en hu de
required_hits 5
auto_report_threshold 30
rewrite_subject 0
spam_level_stars 0
subject_tag *****SPAM*****
report_header 1
use_terse_report 0
defang_mime 0
skip_rbl_checks 1
auto_whitelist_factor 0.5
```

```
auto_whitelist_path /var/lib/spamassassin/auto-whitelist
auto_whitelist_file_mode 0700
dcc_add_header 1
dcc_body_max 50
dcc_fuz1_max 50
dcc_fuz2_max 50
dcc_timeout 10
```

A példa az általam jónak talált beállításokat mutatja. Nézzük meg, mit is jelentenek ezek.

Az ok_locales opció beállítása szerint angol, magyar, és német nyelvű szöveget szoktam kapni, így más nyelveken íródott szöveg nagy valószínűség szerint SPAM. A required_hits alapján azt a levelet jelöli a Spamassassin spam-nek, mely legalább 5 pontot kapott. A jelölésre az "X-Spam-Flag: YES" fejléc bejegyzés szolgál. Ezen felül az "X-Spam-Status:" sor minden levélben jelzi hány pontot és miért kapta a levél, de csak röviden. A SPAM-nek minősített levélbe bekerül a részletes leírás, ha a use_terse_report 0. Ha a "report_header" 1, akkor a fejlécben "X-Spam-Report:" jelöléssel, ha 0 akkor a levél törzsében kap ez helyet. Ha a rewrite_subject 1, a levél tárgya elé beszúrásra kerül a subject_tag szöveg. A levél olvashatóságát lehet „elrontani" a MIME mód megváltoztatásával. Ehhez a defang_mime opció szükséges. A skip_rbl_checks 1 esetén az RBL vizsgálatok nem kerülnek végrehajtásra. Ez Postfix esetén felesleges, jobb ha az SMTP szerver ezt elvégzi. Az auto_whitelist* opciók szabályozzák az automatikus „fehérlistát". Ez egy bonyolult eljárás, mely folyamatosan változtatja egy-egy feladó AWL értékét, és az értéket a levél ellenőrzésekor alkalmazza. Ez ronthatja és javíthatja is a pontszámot. A dcc opciók a már ismert DCC beállításokat takarják.

A fenti filter.sh script kis módosításokkal alkalmas procmail vagy maildrop esetében is. A fő különbség, hogy nem a sendmail-t, hanem a procmail-t vagy a maildrop-ot kell hívni, maildrop xfilter esetén simán a standard outputra engedhető a spamassassin kimenete. A szűrés igen egyszerű:
Procmail esetén:

```
:0:
* ^X-Spam-Flag: YES
Maildir/.spam/
```

Maildrop esetén:

```
if (/^X-Spam-Flag: YES/)
to Maildir/.spam/
```

3. Összefoglaló

Mindenek előtt egy kis statisztika. Az évek során összegyűlt pár tucat e-mail címemre (melyből kb. 2-3-at használok aktívan, a többire szinte csak spam jön) RBL használatával havi kb. 500 spamet fogott meg a spamassassin. Az RBL kikapcsolásával mindössze 10 nap alatt több mint 300 spam érkezett, melyek kb. 90%-át mind a Spamassassin (Razor-al) és a DCC megfogta, kb. 9% volt amit csak az egyik tekintett Spam-nek. A maradék 1%, azaz kb 2-3 spam megérkezett a saját postaládámba. Eközben a Spamassassinnak volt pár hibás pozitív találata, de mind indokolt volt, mint a bevezetőben is írtam. A legtöbb hibás pozitív a Spamassassin levelezőlistáról jött, ahova rendszeresen idéznek be spam-et. Ezt egy „fehérlista" beállítással ki lehetett küszöbölni.

Véleményem szerint a legjobb megoldás a SPAM védelemben az lehetne, ha a legtöbb e-mail szerver már át sem venné a SPAM leveleket. Így a töménytelen spam a küldők és a segítőik (lásd open relay szerverek) nyakán maradna. Utóbbiaknak legalább eszükbe jutna beállítani szervereiket. Ehhez viszont olyan szűrőmegoldásokra lenne szükség, mely jelenleg csak a sendmailben található meg.

4. Hivatkozások

`http://balckholes.mail-abuse.org/`
`http://relays.mail-abuse.org/`
`http://dialups.mail-abuse.org/`
`http://relays.ordb.org/`
`http://ipwhois.rfc-ignorant.org/`
`http://orbs.dorkslayers.com/`
`http://relays.osirusoft.com/`
`http://www.rhyolite.com/anti-spam/`
`http://razor.sourceforge.net/`
`http://www.spamassassin.org/`
`http://www.paulgraham.com/spam.html`

Az embertől az államig: a nyílt forráskód kormányzati haszna

Deim Ágoston

2002. október 21.

Kivonat

Az előadás célja, hogy az embertől a cégeken át egészen az államig, – melyet az emberek alkotnak – bemutassa, milyen gazdasági és más előnyei lehetnek a szabad szoftvereknek a kormányzati munkában.

Tartalomjegyzék

1. Az ember

Az emberek nagy csoportját két részre oszthatjuk: felhasználó és fejlesztő. Érdemes külön foglalkozni velük, mert céljaik és a számukra hasznosítható előnyök eltérőek. Nem véletlenül csak előnyökről beszélünk. Hátrányok ugyanis nem jelentkeznek az egyéni felhasználóknál.

Gyakran elhangzik a vád, hogy ha senki sem fog fizetni a szoftverekért, akkor senki sem fogja karbantartani a szoftvereket. És akkor a felhasználóknak ismét fizetniük kell a szoftverekért. Ez nem feltétlenül igaz, ugyanis pont a nyílt és szabad szoftvereknél van meg a lehetőség a későbbi módosításra, folytatásra, amit bárki megtehet, nem csak a szoftver írója. Másfelől a szabad szoftver hívőinek is le kell számolniuk azzal a tévhittel, hogy a szabad feltétlenül ingyenest is jelent. A nyílt forráskódú fejlesztés is pénzbe kerül, ezért van szükség támogatásra illetve kiegészítő tevékenységre.

1.1. A fejlesztő/informatikus

A nyílt forráskód haszna a fejlesztő számára nagyon sok lehetőséget biztosít. Hiszen rengeteg példát láthat, rengeteg eszközt kap a kezébe, amivel alkothat. Számára kibővül a lehetőségek köre. A nyílt forrású alkalmazások tanulmányozásával rengeteget tanulhat, és amennyiben nem a forráskódot használja, saját, akár kereskedelmi alkalmazásba is beleilleszthet az ötleteket.

Itt elsősorban olyan esetekre gondolok, mint egy szoftver moduláris felépítéshez gyűjtött ötletek, kinézet, viselkedés, esetleg fájlformátum. Vagy csak egyszerűen a szabványokhoz való ragaszkodás. Semmi esetre sem jelenti azonban a tanulás azt, ha a forráskódot átnézve, a program belső változóneveit másképpen elnevezve saját, kereskedelmi alkalmazásába illeszti azt a fejlesztő.

Ha a fejlesztő nyílt forrású alkalmazást fejleszt akár komoly hírnevet is elérhet. Ekkor megnő az értéke a munkaerő-piacon, hiszen sikeres és elismert fejlesztőként jobb állásajánlatok közül válogathat. Ha be is fejezi ekkor szabad szoftveres pályafutását – ez nem jellemző, sőt! – akkor is van, aki tovább vigye a fejlesztést. Egy sikeres fejlesztés tovább vitelére nyilvánvalóan sok a jelentkező. Az alkalmazó cégek azonban sokszor megkövetelik, hogy folytassa a fejlesztést, mert ez a cégnek is reklám lehet, valamint ehhez kapcsolódóan tud szolgáltatásokat értékesíteni.

1.2. A felhasználó

A felhasználó szempontjából a legjobb talán a helyzet. Egyrészt nem kötelező kifizetnie a szoftvert, amihez hozzájut, más részről pedig biztosítva lehet arról, hogy a szoftvernek valóban lesz folytatása. Amíg ugyanis egy cég csődbe mehet, addig a fejlesztői közösséggel ez nem történhet meg.

Felmerülhet persze, hogy abbahagyják egy szoftver fejlesztését, de ez a legritkább esetben fordul elő, főleg, ha népszerű az alkalmazás. Talán még ennél is fontosabb az, hogy jogkövető magatartást tanúsít, nem vét a szellemi tulajdont védő törvények ellen, ha szabad szoftvert használ. Ha pedig segítségre van szüksége, azt rendkívül sok forrásból megszerezheti. A szomszéd szakembertől a támogatást nyújtó cégeken át a levelezőlistákig. Számára ez is költséghatékony, nagy választék esetén a támogatás is olcsóbb. Hogy miért éri meg mégis olcsóbb támogatást nyújtani a felhasználóknak, azt a cégeknél megvizsgáljuk.

2. A cégek

A cégeket, csakúgy, mint az embereket két csoportra oszthatjuk: felhasználó és fejlesztő cégekre.

2.1. Felhasználó cégek

A felhasználó cégeknél az előny egyértelmű. Elsősorban nem a szoftver ára számít, hanem a teljes tulajdonlási költség (TCO = Total Cost of Ownership). Ez a szoftver bekerülési árán túl tartalmazza a szakemberek fizetését, a szükséges hardver fejlesztéseket, az üzemeltetés egyéb költségeit vagy például azokat a károkat, melyeket egy leálló informatikai struktúra okozhat.

Egy nemrég megjelent tanulmány szerint a Linux, mint legnépszerűbb szabad operációs rendszer teljes tulajdonlási költsége fele akkora, mint egy Windows alapú megoldásé, és harmada annak, amibe egy kereskedelmi Unix alapú rendszer kerül.

A felhasználó cégek előnyei közé sorolható, hogy nem lesznek egy gyártóhoz kötve, az általuk használt szoftvert pedig testre is szabhatják, illetve megbízhatnak külső céget, hogy tegye ezt meg. A nyílt forráskód egyik legnagyobb előnye ez: a cégek a leghatékonyabb struktúrát alakíthatják ki a nyílt forráskód segítségével. Mint a magán felhasználóknál, itt is jelentkezik a jogkövető magatartás megjelenése, azaz a cégek nem követnek el törvénysértést, ugyanis nem lopják el a szoftvert.

2.2. Fejlesztő/informatikai cég

A fejlesztő cégek gyakran panaszkodnak arra, hogy mi lesz később, hiszen senki sem fog hozzájuk fordulni. Ez nem igaz, hiszen minél több felhasználó kapcsolódik be a rendszerbe a nyílt forráskód által, annál több szakemberre lesz szükség.

A legtöbb szoftver, mint például a képnézegetők, zenei fájlokat lejátszó alkalmazások, egyszerű szövegszerkesztők így is tömegével voltak megtalálhatók a piacon. A többi alkalmazás, mint például a webkiszolgálók, levelező kiszolgálók, nagy tudású szövegszerkesztők és táblázatkezelők pedig mind egy cég kezében voltak. Ők tehát nem vesztenek semmit, inkább nyernek.

Minden a helyes gazdasági modell megalkotásán múlik. Nagyon sok nyílt forrású fejlesztőeszköz található a világhálón, ezek segítségével professzionális alkalmazások készíthetők. A cég tehát itt is spórolni tud. Rengeteg a dokumentáció, a levelezőlisták mind a tudás aranybányái. Mivel a problémák a legritkább esetben egyediek, ezért valahol, valaki már megtalálta a megoldást.

A szabad szoftverek nélkül nem lenne ekkora fejlesztői bázis és az informatikai piac tetszhalott lenne, vagy egy-két nagy szoftvergyártó szorításában vergődne. Látszik tehát, hogy bár ellentmondásnak tűnik, de az informatikával foglalkozó cégek számára is költségcsökkentést hoz a szabad szoftverek világa.

2.3. A jó üzleti modell megtalálása

A legfontosabb, hogy az értékesítésből származó haszon helyett a kapcsolódó szolgáltatások értékesítésére kell koncentrálni. Fontos bevételi forrás lehet természetesen az egyszeri bevétel is, mint például egy szoftver testre szabása, de rögtön ehhez kapcsolódóan értékesíteni lehet a hosszabb távú támogatást. Forgalmazhat könyveket, ajándéktárgyakat, tarthat tanfolyamokat. Minél több a felhasználó, annál több munka lesz.

3. Az állam

3.1. Gazdasági és egyéb előnyök

Ellenőrizhető kormányzati rendszerek

A legfontosabb tényező a kormányzati rendszereknél nem (csak) a rendszer ára, hanem annak biztonsága. Szabad szoftverek esetén lehetőség van a szoftverek ellenőrzésére, biztonsági auditálásra. Ez egyre fontosabb, hiszen napvilágra került már olyan tény is, hogy a legelterjedtebb operációs rendszerben olyan hátsó kapu volt (csak volt?), melyhez egy kulccsal a Microsoft, egy kulccsal az amerikai nemzetbiztonsági hivatal, egy kulccsal pedig ismeretlenek rendelkeztek.

Miért fontos a kód auditálhatósága? Egy képzett számítógépes betörő (cracker)[1] nem kell, hogy rendelkezzen a rendszer forráskódjával, anélkül is be tud törni a rendszerekbe. Elég csak azokra az időkre gondolni, mikor még nem terjedtek el a szabad szoftverek és rengeteg betörés volt. A szakemberek azonban kijavíthatják a hibákat, ha rendelkeznek a forrással. A világhálón pedig rengeteg fejlesztő és cég tanulmányozhatja a forráskódot, míg a zárt forrású termékeknél erre esély sincsen.

A tapasztalatok alapján látszik, hogy a szabad szoftverek jelentik az egyetlen lehetőséget a kormányzati munkában. Nem véletlenül választotta az Amerikai Nemzetbiztonsági Szolgálat (NSA) fejlesztési platformjának az olyan szabad rendszereket, mit a Linux vagy a FreeBSD.

Gyártó függetlenség

A legnagyobb előny, hogy megszűnik a kormányzat függősége a gyártóktól vagy kisebb mértékben lesz jelen. Ha kis mértékben jelen is van, akkor is kiváltható a gyártó/beszállító. Ennek egyik legjobb példája volt, amikor a Microsoft bejelentette, új irodai termékei nem futnak majd régebbi rendszerein. Ezzel rákényszerítette a felhasználókat, hogy újabb rendszereket vásároljanak tőle.

Amennyiben szabad szoftvereket használ a kormányzat, ez nem fordulhat elő, hiszen a forráskód birtokában megbízható egy cég, hogy integrálja az új funkciókat a régi termékbe.

Természetesen a szoftverek itt is rendelkeznek életciklussal és újabb funkciók az újabb változatban jelennek meg, de egy régebbi alkalmazás futtatható az új rendszeren is. Ennek lehetősége más platformokon is elérhető lenne, hiszen csak az alkalmazások által használt úgynevezett függvénykönyvtárak régebbi változatainak kell megtalálhatónak lenni a rendszeren.

3.2. Munkahely-teremtés, munkaerő képzés

A nyílt forráskód rengeteg munkahelyet tud teremteni. Téves elképzelés az, hogy ezáltal munkanélküliek lesznek a programozók. Az átlag felhasználó nem fog programokat írni, és minél több felhasználó fér hozzá a rendszerhez – annak ingyenes volta miatt – annál több szervizesre, programozóra, rendszergazdára lesz szükség.

Ekkor új cégek alakulhatnak, akik aztán más munkaerőnek is segítenek munkához jutni. Egy cégnek szüksége van könyvelőre, takarítóra, titkárnőre és még sok más feladatra alkalmazhatnak munkavállalókat. Ezen kívül a költségvetésben is megjelennek az általuk fizetett adók és járulékok.

[1] Nem keverendő össze a *hacker*-rel. A különbségről bővebben a Hacker-HOWTO-ban, a `http://www.lme.hu/forditas/hacker-howto.html` oldalon lehet olvasni.

3.3. Oktatás

Az oktatásban is fontos szerepe van a szabad szoftvereknek. Rengeteg szabad fejlesztő eszköz létezik, rengeteg alkalmazás, melyek forráskódját, fejlesztési módszereit tanulmányozhatják az informatikát tanuló diákok.

Mivel működő, éles rendszereket tanulmányozhatnak, láthatják a helyes fejlesztési módszereket. Szabad szoftverek fejlesztésében való részvétellel gyakorlatot is szerezhetnek, melyet később hasznosíthatnak. Az ország informatikusainak értéke tehát megnő. Lokális tudásbázist lehet a szabad szoftveren nevelkedett fejlesztőkből alkotni, akik javíthatják az ország versenyképességét.

3.4. Állampolgárok jogkövető magatartása

Nincsen többé szükség arra, hogy a „kör bezárulásától” kelljen félni, amennyiben szabad szoftvert választhatnának az állampolgárok. Jelenleg a kormányzat nem támogatja eléggé a szabad rendszereket, a legtöbb felhasználó (állampolgár) nincs is tisztában azzal, hogy ő tulajdonképpen folyamatosan törvényt sért.

Ezt könnyen ki lehetne küszöbölni, ha a felhasználókat tájékoztatnák a lehetőségekről. Itt lenne nagy lehetőség, ha az iskolai oktatásban megjelenne a szabad szoftverek bemutatása, itt elsősorban a Linux lehetne az, melyet be lehetne mutatni. Meg kell ismertetni a diákokat a lehetőségekkel, a jogi vonzatokkal is, és hagyni, hogy ő válaszon. A legfontosabb, hogy egyáltalán tisztában legyen a lehetőségekkel.

4. Kitekintés

4.1. Sikeres szabad szoftver felhasználás

Európa

Németország: A német kormányzat folyamatosan tér át szabad szoftver használatára rendszerein, és a törvényhozásban (Bundestag) is át akarnak térni szabad szoftverekre.

Rendkívül sokrétű fejlesztést folytatnak, többek között kormányzati információs rendszerüket is szabad szoftverekre alapulva fejlesztik.

Dánia és Norvégia: E két ország kormánya döntött nemrég úgy, hogy több tízezer kormányzati munkaállomáson többé nem kívánják használni a Windowst, helyette Linuxot telepítenek a gépekre.

Franciaország: A francia kormányhivatalokban és iskolákban kötelezően szerepeltetnek egy Linux-szal telepített számítógépet.

Ázsia

Kína: A kínai kormányzat tervei között egy saját Linux disztribúció kiadása szerepel *RedFlag Linux* néven. Az ország programozóit „mozgósították”, hogy elérjék a célt. Több tíz illetve százezer programozó fogja fejleszteni a rendszert, hogy teljes mértékben ki tudják váltani a Microsoft operációs rendszerét.

Korea: A koreai kormány nemrég kötött megállapodást új beszállítókkal. Ezek között van a Hancom nevű cég, mely irodai programcsomagokat fejleszt, és képes kezelni az ismert dokumentum típusokat. A koreai kormányzat szintén hajlandó a Linux további fejlesztéséért fejlesztőknek fizetni.

Észak-Amerika

Amerikai Egyesült Államok: Az Amerikai Egyesült Államok nemzetbiztonsági szolgálata is támogatja a szabad szoftvereket, itt készítették el a *SELinux*-ot kormányzati felhasználásra.

Ehhez kapcsolódóan a Washingtoni Egyetemen folyamatosan auditálják a Linux kernel forrását és az alapvető alkalmazásokat, hogy az megkapja a Common Criteria minősítést. Ez ahhoz szükséges, hogy akármelyik kormányhivatal használhassa a Linuxon alapuló rendszereket.

Ezen is túlmenően Kalifornia állam törvényhozása tervezi bevezetni a *Digital Software Security Act* elnevezésű törvényt. Ez megtiltaná, hogy kormányhivatalok (az államon belül) olyan alkalmazásokat vegyenek illetve használjanak, melyhez nem kapnak forráskódot. Ez gyakorlatilag csak a szabad szoftverek használatát tenné lehetővé. Érdekes tény, hogy Redmond város (itt van a Microsoft székhelye) önkormányzata is áttért a Linux használatára a hivatalokban.

Közép- és Dél-Amerika

Mexikó: A Red Escolar program keretében az iskolákat a jövőben linuxos munkaállomásokkal és kiszolgálókkal kívánják felszerelni, ettől több, mint 100 millió dollár(!) megtakarítást várnak.

Mexikó város önkormányzata is több millió dollárnyi megtakarítást remél attól, hogy Linuxra és más szabad szoftverre szándékozik átállítani rendszereit.

Peru: A perui kormányzat teljes mértékben át kíván térni szabad szoftverek használatára az eddigi megoldások helyett.

UHU Szerver és UHU Tűzfal verzió: fejlesztési koncepciók

Deim Ágoston, Illés Márton

2002. október 21.

Kivonat

Az előadás bemutatja az UHU kiszolgáló és tűzfal változatának tervezési, megvalósítási menetét. Ismerteti a kihívásokat, melyeket már meglévő alkalmazások átalakítása vagy éppen új alkalmazások írása állított.

Tartalomjegyzék

1. A kezdetek

Cégünk kezdetektől fogva a Linuxot helyezte megoldásai középpontjába, így teljesen természetes volt, hogy minden kezdeményezés örömmel tölt el minket, ami a Linux hazai elterjedését segíti. Annál is inkább, mivel cégünk döntő többsége LME tag is. Gyakran felmerülő problémának láttuk azonban, hogy egy adott alkalmazás nem tudott magyarul, egy adott hardver eszközhöz nem tartozott könnyen használható beállító program. A Linuxhoz még nem értő, de erre a platformra áttérni akaró felhasználókat gyakran elriasztották ezek a tények. Szép sorban megjelentek az egyre könnyebben települő, egyre szebb külalakkal rendelkező disztribúciók, melyek alatt a legtöbb alkalmazás már képes volt magyarul megszólalni. Ezek a disztribúciók elnyerték a felhasználók tetszését, mi is ezeket ajánlottuk azoknak a felhasználóknak, akik asztali számítógépen akarták használni a rendszert. Azonban ezeknek a disztribúcióknak pont az egyik előnyük lett a hátrányuk. A hivatalos csomagok száma miatt a disztribúciók DVD méretűre híztak. Ekkor viszont a programok túlzott választéka volt, ami elriaszthatta a felhasználót. Mit használjak, mire és miért pont ezt? Ez volt az általános kérdés. A disztribúciók mindent oda akartak adni a felhasználóknak. Léteztek és léteznek ma is olyan disztribúciók, melyek elférnek egy CD-n, azonban nem rendelkeznek magyar felülettel. Adott volt tehát két megoldás, de egyik sem volt teljes mértékben kielégítő. Ezért szolgált nagy örömünkre, mikor az első béta letöltése után először feltelepítve az UHU-Linux kliens verzióját, megtaláltuk benne azt, ami egybe vágott a mi elképzeléseinkkel, kliens oldalon. Követtük a fejlődés pillanatait, majd több körülmény hatására egymásra találtunk. Régi tervünk volt, hogy saját disztribúciót készítsünk, mely a mi általunk felállított kívánalmaknak megfelel. Az UHU-Linux csapata pedig egy kiszolgáló verziót is létre akart hozni, logikus volt tehát, hogy a két, közös szemléletű társaság megvalósítsa azt.

1.1. A motivációk

Mindenkinek van kedvenc disztribúciója, mi sem voltunk kivételek. A mi kedvencünk a Debian volt, minden erősségével és hibájával együtt. Be kellett azonban látnunk, hogy a legtöbb cég nem a bő programválasztékra vagy a korlátlan szabadságra vágyik, sőt... A legtöbb cégnek nem is kell több, mint egy olyan kiszolgáló, mely a mindennapos irodai vagy egyéb számítástechnikát igénybe vevő munkát összefogja és irányítja. Ennek felismerése, a mindennapos tapasztalatok és visszajelzések vezetett minket elsősorban abba az irányba, hogy egy egységes, könnyen karbantartható cél-disztribúciót hozzunk létre. A Debian egy gyönyörű projekt a maga nemében, de néha lassabban halad a fejlesztése a cégeknél elvárt tempóhoz képest. Ez legnagyobb erősségükből adódik, hogy csak akkor adnak ki új disztribúciót, mikor késznek tekinthető. Ezt a minőségbiztosítási szempontot mi is teljes mértékben támogatjuk, e mellé a verziókövetések gyorsaságát tűztük ki. Olyan disztribúciót akartunk készíteni, mely minden kis- és középvállalkozás igényeit maradéktalanul kielégíti, de megállja a helyét nagy hálózatokban és infrastruktúrális környezetekben is.

1.2. Kutatás és összegzés

A velünk kapcsolatban álló cégeknél tapasztalt problémákat csoportosítva jutottunk el az alábbi megállapításokig.
A cégek jogos igénye, hogy központilag lehessen adminisztrálni a felhasználókat. Egy vállalati rendszergazdának ugyanis, főleg, ha sok kiszolgálót és kliensgépet kell fel-

ügyelnie, nem sok ideje marad a különálló felhasználói adatbázisokat karbantartani kiszolgálóprogramok szerint. A hibázás esélye is megnő a nagyobb terhelés miatt, így belátható, hogy csak egy könnyen és jól adminisztrálható rendszert lehet bevezetni, ha igazán eredményesek akarunk lenni. A disztribúciókon végignézve pedig nem találtunk olyat, mely könnyen telepíthető, telepítés után rögtön rendelkezik a központi felhasználó-adatbázis elérhetőségével és ehhez kapcsolódó menedzsment felülettel. A menedzsment felület természetesen nem csak a felhasználókezelést tartalmazza, hanem a kiszolgálón futó alkalmazásokat is beállíthatjuk rajta keresztül. Lényeges pont, hogy a felhasználók adminisztrálásával ne csak a felhasználók hozzáadását, csoportokba szervezését és jelszavainak módosítását lehessen megoldani, hanem hozzá lehessen rendelni a felhasználókat a szolgáltatásokhoz. Ez utóbbit akár úgy, hogy minden szolgáltatáshoz külön jelszót kapjon a felhasználó. Nincs szükség nagy programválasztékra sem. Ami kell, az a megbízható működés és a támogató cégtől a gyors és pontos terméktámogatás. Ezt ismét csak akkor lehet megtenni, ha megfelelően kicsi marad a kiszolgálóprogramok száma.

1.3. A célok kitűzése

A felmerült tapasztalatok, felmérések és az azokból levont következtetések alapján állítottuk tehát össze azon követelményeket, melyek teljesítésével elérhetjük a kívánt eredményt. Nem csak a saját magunk által tapasztalt problémákat elemeztünk, hanem figyelembe vettük a fórumokon, levelezőlistákon közzétett levelek azon részét, melyet olyan rendszergazdák írtak, akik szeretnének (szeretettek volna) áttérni Linux operációs rendszerre a Microsoft alapú platformokról, de nehézségekbe ütköztek, melyeket csak hosszas tanulási folyamattal tudtak volna megoldani. A tanulási folyamat nem azért hiúsul meg legtöbbször, mert lusták ezek az emberek, hanem mert ez időbe kerül. Nekik pedig pont ebből nincs sok, hiszen gyakran egyedül látják el feladataikat. Ha pedig idejük és kedvük is van a tanuláshoz, akkor a kísérletezéshez, tanuláshoz nincs szabad számítógép vagy számítógépek. Fontos az első benyomás is. Bár tökéletesen megfelelne a célnak egy karakteres telepítő alkalmazás, azonban egy kép többet mond száz szónál is, ezért szükségesnek ítéltük a grafikus telepítést. A rendszergazdák számára fontos az, hogy mindig megfelelő segítséget kapjanak, tudják, hogy egy általuk addig nem használt funkció mit eredményezhet. Belátható, hogy a fenti követelményeknek megfelelni nem könnyű. Ezért ahol már volt létező megoldás, ha használható volt, akkor azt beépítettük a rendszerbe, szükség esetén módosítottuk azt.
A célok tehát:

- Központosított felhasználói adatbázis, szabványos alapokon,
- A központi adatbázist elérni képes alkalmazások használata,
- Kiszolgáló alkalmazások könnyű beállíthatósága ,
- Biztonság ,
- A feladatokat maradéktalanul ellátó, minimális mennyiségű csomag a könnyű karbantarthatóság és felhasználói támogatás miatt ,
- Különböző hardvert feltételező konfigurációs fájlok különböző géptípusokhoz ,
- Jól működő csomagkezelés ,
- Grafikus telepítő alkalmazás ,

- Kezelhető és könnyen bővíthető menedzsment eszköz, valamint
- Sok és pontos segítség a szoftver mellé, lehetőleg a menedzsment szoftverben.

2. A tervezés

Kihez vagy mihez kellett alkalmazkodni?

Alapvető célkitűzés volt, hogy a szabványok útján kell haladni. Ennek alapján teljességgel figyelembe vettük a *Linux Standards Base* valamint a *Filesystem Hierarchy Standard* ajánlásait. Mint már említettem, a Debian rendszer nagy hatást tett ránk, ennek megfelelően a fejlesztési szemléleteit ahol lehet, magunkévá tettük.

Alapoktól új disztribúció vagy már meglévő átalakítása?

Felmerült bennünk, hogy alapoktól új disztribúciót készítünk, azonban az UHU kliens készítőivel egyetértésben sokkal hasznosabbnak láttuk, ha közös alapokra helyezzük a minden változatban előforduló alkalmazásokat. Ezért mindenhol ugyanazokkal az alapcsomagokkal találkozhatunk mindhárom UHU változatnál. Ez a későbbiekben jelentősen megkönnyíti a verziókövetéseket és az egységes terméktámogatást. Az UHU kliensből felhasználtuk a telepítőt is, mely megfelelt a grafikus telepítő iránti igényeinknek: szép és bővíthető.

Szoftver kiválasztási szempontok

A szoftver kiválasztásánál elsőrangúnak ítéltük meg, hogy teljesítse a saját típusán belüli követelményeket, de mindemellé biztonságos és az RFC-ket valamint a de facto szabványokat kövesse. Ennek alapján a következő kiszolgáló alkalmazások kerültek be a rendszerbe:

- Felhasználó adatbázis és címtárszolgáltatások: OpenLDAP
- Levél küldés és fogadás: postfix
- Levelek letöltése: Courier-IMAP
- Web-kiszolgáló: Apache
- Adatbázis kezelő: Postgresql és MySQL
- Fájlszerver: Samba
- Nyomtató kiszolgáló: CUPS
- Cache proxy: Squid
- FTP kiszolgáló: Proftpd
- Levelező listák kezelése: Sympa
- Távoli menedzsment: OpenSSH és egy átalakított Webmin
- Névkiszolgáló: BIND 9-es verzió
- DHCP kiszolgáló: ISC DHCPD
- Fax szerver: Hylafax

Menedzsment eszköz kérdései

Mint látható, a menedzsment eszköznél is egy már jól bevált alkalmazásra támaszkodtunk. A Webmin előnyei közé tartozik, hogy nem szükséges számára külön webszerver, képes SSL kapcsolatok kezelésére és ami a legfontosabb, modul rendszerűen épül fel, új modulok készítése tehát nem okoz problémát. A programozási felület jól dokumentált és átlátható, az alkalmazás maga Perlben íródott, csak szabványos modulokat használva. Már fentebb írtam, hogy elsődleges szempont: a felhasználónál a felhasználóhoz kapcsolódó összes beállítás elérhető legyen. Fontos azonban az is, hogy a felhasználó saját maga be tudja állítani jelszavát, személyi adataitm, vagy azt, hogy hova kerüljenek továbbításra a levelei (természetesen csak akkor, ha erre az adminisztrátor engedélyt adott).

A szabványok tanulmányozása

A következő nagy lépés a szabványok tanulmányozása volt, mely rengeteg olvasással és elemzéssel járt. Miután a szoftvereknél kikötés volt, hogy RFC-hez és szabványhoz igazodjanak, ezért a már előrébb említett *LSB*-hez és *FHS*-hez igazítottuk a fájlrendszer, a függvénykönyvtárak és az indítószkriptek elhelyezkedését, funkcióit.

2.1. Az „igazi” tervezés

Az összesítések után következett egy rendkívül nehéz időszak, mely a kigyűjtött információk rendszerezésével és keretbe foglalásával telt el. Eközben kényszerűségből néhány dolgot már implementálni kellett. Például, hogy megbizonyosodjunk valóban hatékonyan használható lesz-e egy adott alkalmazás módosítás után a feladatra, módosítottuk, majd teszteltük.

3. Fejlesztés

3.1. A rendszerterv gyakorlati végrehajtása

A rendszer tervezése után természetszerűleg következett a fejlesztés. A tervezés közben kényszerűségből már kifejlesztett részeket beillesztettük a rendszer alapjaiba. A következő lépés a funkciók – levelezés, webszolgáltatás stb. – szerinti szolgáltatások buildelése egy körben. Ezzel párhuzamosan történt a Webmin alapú menedzsment felület felkészítése a kiszolgáló alkalmazások beállítására. A kritikus pont a már összeillesztett *teljes* rendszer tesztje. Megeshet, – és meg is esett már – hogy részenként jól, egészben azonban nem működnek megfelelően az alkalmazások. Vagy egymással vagy a menedzsment felülettel. A hibák itt nem olyan mértékűek voltak, melyek a rendszer újragondolását tették volna szükségessé, csak a részek összeillesztésénél csúszott be néhány kisebb hiba.

3.2. Javítások elkészítése, átalakítása, alkalmazása

Az alkalmazások némelyikénél szükséges volt, hogy további kiegészítéseket tegyünk a hibátlan vagy a mi általunk elvárt működés biztosításához. Ezeket a javításokat (patchek) mind visszajuttattuk vagy éppen visszajuttatjuk a fejlesztőknek és a közösségnek.

3.3. Tesztelés menete avagy melyik szoftverhez mit használunk?

A fejlesztés egyik legfontosabb eleme a tesztelés. Itt derülnek ki azok a hibák, melyek megakadályozhatják, hogy a végtermék beváltsa a hozzá fűzött reményeket. A kiszolgálóknál rendkívül fontos volt, hogy tudjuk, mekkora teljesítményre is képes, attól függően például, hogy mekkora teljesítményű számítógépen fut. Ez ugyanis a konfigurációs fájlok beállításaink újragondolására kényszeríthet minden fejlesztő céget, így minket is. Néhány esetben ez meg is történt, ahol alacsonyabb teljesítményű géptípusokat használtunk.

- HTTP/HTTPS: siege – `http://www.joedog.org/siege/index.shtml`
- FTP: dkftpbench – `http://www.kegel.com/dkftpbench/`
- SMTP: postfix saját eszközei
- POP3/IMAP: saját eszköz, `http://www.lsc.hu/download.html`

4. A jövő

4.1. Verziókövetés

A kiszolgálót alkotó alkalmazások oldalait és levelező listáit folyamatos figyelemmel kísérjük, hogyha valami váratlan hiba derülne ki, reagálni tudjunk. Természetesen az általános programfrissítéseket is figyelembe vesszük és az UHU Szerver újabb kiadásaiban már az újabb változatok kapnak majd helyet.

4.2. A jövő tervei

A kiszolgáló jövője, reméljük sikeres lesz. Úgy érezzük mindent megtettünk, de ami még fontosabb, mindent meg is fogunk tenni. A fejlesztés nem áll meg, folyamatosan halad a felhasználók visszajelzései alapján. A módosított és új Webmin modulok LGPL licenc alatt is kikerülnek a világhálóra, hogy minél többen tanulmányozhassák. A kiszolgáló verzió, a módosítások szintén visszakerültek és visszakerülnek ott is, ahol a licenc megengedné a forrás bezárását. Ennek első példája a Webmin, mely BSD licencelésű, így megtehettük volna, hogy az összes módosítást visszatartjuk, de mi nem kívántunk ezzel a módszerrel élni. A legjobb útnak azt látjuk, ha nyílt alapokra helyezzük a rendszert, ez ugyanis már bevált és jó minőségbiztosítási eszköz. Vannak természetesen olyan cégek, melyek jobban örülnek, ha megvehetik a terméket, ettől ugyanis szerződéses alapú kötelezettségeket várhatnak el. Ők megtehetik, hogy megveszik a kiszolgáló verziót és a módosított felületet. A fejlesztés elején ugyanis már számítottunk erre és a menedzsment felület első változatai kereskedelmi verziók voltak, majd kihasználva a duál licencelés lehetőségeit, kibocsátottuk a modulokat LGPL licenc alatt is. Úgy érezzük, ezzel megtaláltuk a lehető legjobb megoldást mind az üzleti felhasználók, mind a közösség számára.
A felhasználók visszajelzései alapján tervezzük, hogy újabb szoftvereket építünk be a termékbe, bővítve lehetőségeit. Természetesen továbbra is törekedni fogunk arra, hogy ne csússzunk el abba az irányba, amikor már nem tudjuk teljes mértékben irányítani a folyamatokat. Éppen ezért, egy maximális csomaglétszámot mindig fenntartunk, és azt az álláspontunkat is, hogy minden feladatra egyetlen egy, általunk kiválasztott szoftver legyen megtalálható a kiszolgálóban.

5. Hivatkozások

http://www.joedog.org/siege/index.shtml
http://www.kegel.com/dkftpbench/
http://www.lsc.hu/download.html

ISDK – Information System Development Kit

Hajba Szilárd
<szilu@symbion.hu>

2002. október 15.

Kivonat

Az *ISDK* egy GPL/LGPL licencelésű fejlesztőrendszer, információs rendszerek készítéséhez UNIX-platformra. Története egy, 1997-98 körül megfogalmazódott gondolattal kezdődött: fejlesszünk ki egy integrált vállalatirányítási rendszert Linuxra!

Az igények speciálisak voltak, s nem találva ezeknek megfelelő fejlesztőrendszert, saját fejlesztőeszköz elkészítése mellett döntöttünk. Mivel semmi sem szólt a GPL/LGPL licencelés ellen, így az *ISDK* már a kezdetektől fogva ilyen licenceléssel készül, természetesen GNU eszközökre építve.

Az előadásban a fejlesztés történetét, a kialakított eszközöket ismertetem, továbbá megkísérlem a rendszer tulajdonságait néhány példán keresztül bemutatni.

Tartalomjegyzék

1. Történeti áttekintés

1998-ban fogalmazódott meg a gondolat, hogy belefogjunk egy Linux alatti integrált vállalatirányítási rendszer fejlesztésébe. Nagyon speciális igényeket fogalmaztunk meg magunknak, melyekhez nem találtunk számunkra megfelelő fejlesztőrendszert, ezért döntöttünk egy saját kidolgozása mellett. Mivel nem volt különösebb okunk az ellenkezőjére, ezért a fejlesztőrendszer már a kezdetektől szabadon hozzáférhető volt GPL/LGPL licencelés alatt. Az említett szoftver azóta öt ügyfélnél működik éles üzemben, az egyiknél négy telephelyen. Az *ISDK* a kezdet után fél évvel átment egy komolyabb struktúraváltozáson, de azóta alapfelépítésében változatlan. A fejlesztés persze nem állt meg, több komolyabb bővítés került bele az utóbbi években. Verziószáma jelenleg 2.2.3-nál tart.

2. Fejlesztési eszközök

Az *ISDK* lényegében egy rutinkönyvtár, kiegészítve a fejlesztést, karbantartást segítő segédeszközökkel. Az alábbiakban röviden összefoglalom az *ISDK* egyes moduljainak feladatát.

2.1. ECPP – EC előfordító

A rendszer fejlesztésének kezdetén komoly fejtörést okozott a fejlesztéshez használandó programnyelv kiválasztása. Felmerült többek között a C/C++, valamint néhány szkriptnyelv (például Perl, Python, Tcl). A szkriptnyelvek mellett szólt a sokkal egyszerűbb használat és az, hogy több olyan hasznos funkcióval rendelkeznek, melyek megkönnyíthetik a programozói munkát. A C++ is jó választásnak bizonyult volna, de akkoriban még egyikünk sem volt profi C++ programozásban, és sokat hallottunk a C++ szabványosítási problémáiról.

Választásunk végül a C nyelv mellett állapodott meg, mely már régóta szabványosított volt, valamint nyitva hagyta a kiskaput, hogy később más nyelveket (elsősorban szkriptnyelveket) illesszünk hozzá, lehetővé téve egy *ISDK* segítségével fejlesztett szoftver utólagos bővítését akár a felhasználó által is.

A C nyelv mindazonáltal kissé „fapadosnak” bizonyult ilyen feladatok megoldásához, ezért dolgoztunk ki egy nem túl bonyolult előfordítót, így kialakult az EC nyelv, mely jelenleg az egyetlen támogatott nyelv. Természetesen az előfordító C++ kompatibilis, tehát az *ISDK*-ban fejlesztett alkalmazások íródhatnak ECC nyelven is, jelenleg azonban az *ISDK* nem rendelkezik C++ osztálykönyvtárral, így az *ISDK* hívások C-s függvényhívásokként érhetők el.

Az EC nyelv elsősorban az *ISDK* Object Registry-nek a használatát hivatott megkönnyíteni, melyen keresztül a formok, osztályok, paraméterek, callback-függvények, form és egyéb objektumok címezhetők meg, valamint tartalmaz néhány olyan funkciót, ami a C nyelvből „kimaradt”, például a kivételkezelést.

2.2. Egyéb segédeszközök

Az *ISDK* tartalmaz egy struktúra módosító eszközt, mely egy *ISDK* kompatibilis adatbázis szerkezetét naprakész állapotba képes hozni a megfelelő leírás szerint, hogy a folyamatos szoftverfrissítések egyszerűbbek legyenek.

Tartalmaz továbbá egy egyszerű beágyazott dokumentációs rendszert, melyből man, HTML, LaTeX formátumban generálhatók dokumentumok. A konkrét kódolás megkönnyítésére készítettünk szintaxis fájlokat vim szövegszerkesztőhöz (az általunk preferált szövegszerkesztő :)), valamint tag fájl generátorokat az általunk fejlesztett nyelvekhez (EC, FML, ESQL). Az ESQL nyelv szintén egy „előfordítós” nyelv, mely a struktúra leírására használatos.

3. Az *ISDK* rutinkönyvtár felépítése

3.1. Object registry

Az object registry-ről már korábban ejtettem néhány szót. Ez gyakorlatilag egy szótár az *ISDK* objektumok eléréséhez, melyet szinte bárhonnan egységesen el lehet érni. Ezen keresztül kezeljük például egy ablak objektumait a programból, vagy ezen keresztül éri el egy FML nyelven leírt form gomb objektuma az EC-ben leírt callback függvényt.

3.2. Adatbázis-kapcsolat

Az adatbázis-kapcsolat modul jelenleg sajnos csak PostgreSQL motort támogat, de nagyon könnyen illeszthető hozzá bármilyen más SQL adatbáziskezelő. Az alapvető SQL funkciókon kívül (kapcsolódás, lekérdezés futtatás, tranzakció kezelés, stb..) tartalmaz magasabb szintű speciális funkciókat, melyek segítségével például közvetlen kapcsolatot hozhatunk létre a formok és a Browser objektumok, valamint adatbázis között.

3.3. Browse-alrendszer

Ez a modul egy nagyon fontos eleme az *ISDK* rendszernek. A fentebb említett Vállalati Információs Rendszer leendő felhasználói akkoriban Clipper programokat használtak, ami köztudottan rendkívül felhasználóbarát, leginkább a közvetlen adatbázis megjelenítési képességei miatt. Ez a funkció sajnos tistán SQL-ben nem valósítható meg, alapvető architektúrai eltérések miatt. A felhasználók részéről viszont ez az igény nagyon erősen jelentkezett. Ez a tény volt a leginkább mérvadó abban a döntésünkben is, hogy egy saját fejlesztőrendszert kell készítenünk. Hosszas fejtörés eredményeként született meg az *ISDK*-ban jelenleg elérhető közvetlen adatbázis elérési architektúra, mely az adatbázis bizonyos nézettábláinak speciális tükrözési technikáján alapul.

A Browse-alrendszer kétféle browse-tábla objektumot különböztet meg, melyek egymástól teljesen eltérően kezelendőek.

Az első típus a távoli (Remote Browse), melynek lényege, hogy egy háttérfolyamat a különböző, gyakran használt nézettáblákat (pl.: különböző kódszótárak, cikk-, ügyfél-listák) folyamatosan frissen tartja egy olyan adatszerkezetben, mely hozzáférhető az összes futó program számára és rendelkezik közvetlen adatbázis megjelenítéshez szükséges műveletekkel.

A második típus a helyi (Local Browse), melyet a program egy konkrét futó példánya hoz létre, egy általában kisebb mennyiségű adatot tartalmazó nézettáblára ideiglenesen (például egy konkrét számla tételei). A két különbözően kezelendő Browse-objektummal a megjelenítő objektum ugyanazon a felületen egységesen kommunikál.

3.4. Cserélhető megjelenítő motor (grafikus, szöveges)

Az *ISDK* a formok megjelenítését egy absztrakt megjelenítő felület segítségével végzi. Ezzel a megoldással sokféle különböző megjelenítő illeszthető a rendszerhez, ezáltal az *ISDK*-ra épülő programok – elvben – többféle megjelenítő felületen futtathatók.

Jelenleg az egyetlen teljesen működőképes felület a GTK widget-könyvtárat használja, valamint készült egy szöveges (ncurses alapú) widget könyvtár, melynek a fejlesztése sajnos félbemaradt.

3.5. Saját formleíró nyelv (FML)

Az *ISDK* nem egy vizuális fejlesztőrendszer, így a formok tervezéséhez sem „húzd – dobd" jellegű eszközt fejlesztettünk. Az FML egy kicsit XML-utánérzésű nyelv, mely nagyon egyszerű és szövegszerkesztővel kényelmesen kezelhető.

Két eszköz segíti az *ISDK*-s szoftverfejlesztőket a formok tervezésében, az egyik a vim szövegszerkesztővel használható szintaxis színező, a másik pedig egy form megjelenítő segédprogram.

3.6. Speciális kifejezés-kiértékelő

Az *ISDK*-ba épített kifejezés-kiértékelő nagyon sokoldalúan használható. Tartalmaz speciális operátorokat különböző feladatokra (például szövegformázó műveleteket a riportok generálásához), függvényeket szöveg- és dátumműveletekhez. Szabályozott módon hozzáférhetővé tehetők akár több különböző form objektumai, melyeknek értékére a kifejezésekben hivatkozni lehet.

3.7. Riport generátor

A riport generátor ismét egy jó példa arra, hogy néha érdemes élni vizuális tervezőrendszerek nélkül. Az *ISDK* riport generátora nagyon jól vizsgázott több, mint 200 bonyolult és kevésbé bonyolult áruforgalmi, pénzügyi, főkönyvi listával. A lista objektumait (sorok, fejlécek, láblécek, csoportosítások, ...) kifejezések segítségével tetszőlegesen leírhatjuk, hozzárendelhetünk SQL lekérdezéseket, valamint egyszerű programokat, melyek különböző összegzéseket, vagy egyéb összesítő műveleteket végeznek.

Fejlesztésekor fő motivációnk az volt, hogy minden lehetséges listát, szigorúan előírt formátumú bizonylatot létre tudjunk benne hozni, így az *ISDK* alkalmazások felhasználói kaphassanak egy olyan felületet, ahol az összes beépített lista, bizonylat formátumát meg tudják változtatni igényeik szerint az általuk létrehozott riportok mellett. Egy SQL ismeretekkel rendelkező személy által a használata egy nap alatt elsajátítható.

3.8. *ISDK* nyomtató filter

UNIX rendszereken nagyon jól bevált stratégia, hogy a nyomtatott állományok a munkaállomástól a nyomtató szerverig egységes formátumban (PostScript) mozognak és csak közvetlenül a nyomtatás előtt kerülnek lefordításra a nyomtató saját nyelvére. A generált riportok gyakran hosszú oldalakon keresztül szöveges információkat tartalmaznak, ezért mi ugyanezt az alapelvet próbáltuk átültetni a szöveges állományok nyomtatására. A riport generátor a nyomtatott állományba speciális formázó szimbólumokat ír kiemelt, dőlt, aláhúzott karakterek nyomtatásához, melyeket egy egyszerű nyomtató szűrőprogram fordít le a konkrét nyomtató formázó parancsaira. Ezáltal

könnyen megvalósítható, hogy bármely munkaállomásról bármely nyomtatóra nyomtassunk anélkül, hogy minden munkaállomáson be kellene állítani a nyomtatókat.

4. Jövőbeni fejlesztési irányok

4.1. Browse NG

A jelenlegi browse implementációnk használatakor rengeteg ötletet gyűjtöttünk össze, melyekből összeállt egy teljesen új koncepció. Az új generációs browse-alrendszer, amely már szinte teljes egészében elkészült, még több lehetőséget rejt magában, mint a jelenleg használt változat.

4.2. Kifinomultabb plugin támogatás (megjelenítés, adatbázis motor)

Jelenleg az *ISDK* modulokat (megjelenítő, adatbázis motor) csak a könyvtár újrafordításával lehet cserélni. Terveink között szerepel egy plugin támogatás, hogy ezek a modulok dinamikusan cserélhetőek legyenek, valamint az, hogy adatbázis-kapcsolatoknál kapcsolatonként lehessen definiálni az adatbázis-szerver típusát.

4.3. Más nyelvek támogatása (ruby, python, . . .)

Jelenleg csak az EC/ECC nyelveket használhatjuk fejlesztésre az *ISDK*-hoz. A több éves tapasztalatok szerint lehet EC nyelven komoly szoftvert fejleszteni, de néhány szempontból kényelmesebb lenne, ha szkriptnyelveken megírt függvényeket is tudnánk illeszteni a programhoz. Ezeket a nyelveket bővítésként szeretnénk a későbbiekben az *ISDK*-hoz kapcsolni, így egymástól függetlenül – akár egy alkalmazáson belül is – több programnyelv is használható lenne.

A GNU/Linux rendszermag optimalizálása tűzfalakon

Harka Győző
<carlos@ttk.pte.hu>

2002. október 21.

Tartalomjegyzék

Bevezetés

Rengeteg irodalom szól az ipchains, iptables, stb. tűzfalak használatáról, s így ezeket némi utánajárással, utánaolvasással bárki könnyűszerrel optimalizálhatja, alakíthatja kedve szerint.

Ugyanakkor a kernel fordítási paramétereiről, illetve a sysctl-en keresztül dinamikusan beállítható paraméterekről szóló szakirodalom jóval szegényesebb, angol és magyar nyelven egyaránt.

Különösen fontos ezen változók megfelelő ismerete, mivel igen sokan nem fordítanak saját kernelt, – hacsak rá nem kényszerülnek driverek vagy egyéb egyedi funkciók miatt – így azokat a beállításokat használják, amelyekkel a rendszer üzemel ugyan, de nem optimális teljesítménnyel, s nem optimális biztonsági szinten. Ez utóbbi tipikus példája az ipchains segítségével elutasítani a teljes ICMP forgalmat a gép felé, amely praktikusabban oldható meg a `/proc/sys/net/ipv4/icmp_echo_ignore_all` kernelparaméter 1-re történő beállításával

Fontos tisztázni hogy mindezen paraméterek működése nem függ attól, hogy milyen Ethernet-csatolót használunk, csak – mint azt látni fogjuk – annak sebességétől. Előadásom keretében a következő GNU/Linux kernel változók optimális beállítását, szerepét ismertetem részletesen:

- `/proc/sys/net/core/netdev_max_backlog`
- `/proc/sys/net/core/rmem_max`
- `/proc/sys/net/core/wmem_max`

Említést teszek továbbá a `/proc/sys/net/unix/max_dram_qlen` paraméterről is.

1. Néhány szó a kernel-paraméterekről

1.1. A /proc/sys/net/core/netdev_max_backlog

A forrásban a receiver rutinok között található integer, a `net/core/dev.c`-ben :

```
int netdev_max_backlog = 300;
```

Alapértelmezett értéke mind a 2.2-es mind pedig a 2.4-es kernel szériában 300. Több géptípuson is végeztem méréseket a fenti paraméter különböző értékeivel, megállapítandó, hogy milyen hatással vannak a hálózati forgalomra, s hogy milyen architektúrán, milyen tűzfalszoftver használatakor mely beállítás az ideális.

Elmondható, hogy gépünk hálózatkezelésének sebességére alapvető befolyással van a fenti érték, főleg abban az esetben amikor az illető gép mint tűzfal vagy mint router működik.

1.2. A /proc/sys/net/rmem_max, wmem_max

Ez a két változó főleg abban az esetben fontos, ha a csomagok egy helyi program által kerülnek feldolgozásra.

Beállításuk elsősorban a samba, NFS, http, ftp szolgáltatások sebességére van hatással.

2. A tesztelés menete

2.1. A tűzfalszkriptekről

Beható vizsgálatok után egyértelművé vált az, hogy akár ipchainst, iptablest, ipfwadmot, akár egyéb tűzfalszoftvert használunk, a kernel-paraméterek sebességre gyakorolt hatását ez nem befolyásolja, így az egyszerűség kedvéért az ipchains-nél maradtam. Minden teszt három alapvető tűzfal beállítás mellett került végrehajtásra:

- egy sorból álló minimális filter sor.
- tipikusnak mondható tűzfal, masquerading,valamint 5 tűzfal szabály az adott portra.
- ötszáz sorból álló szabályrendszer, amelyben a teszt kedvéért valamennyi beérkező csomag végighalad mindegyik szabályon.

Az ftp es a http protokollok tesztelése során a /dev/ram partíción tárolt adatok kerültek átvitelre, kiküszöbölendő az állományrendszer, illetve a háttértár változó elérési sebességének „statisztika-romboló” hatását.

Hasonló megfontolásokból kapcsoltam ki a swap partíciókat is.

A gépeken minden, a tesztelés szempontjából nem elengedhetetlenül szükséges processz leállításra került, azonban hogy a lehető leghívebben tükrözzem egy átlagos működés közbeni állapot meglétét műterhelés gyanánt a `"cat /dev/zero > /dev/null"` processz került elindításra, amellyel egy egyenletes terhelést biztosítottam.

Így lehetővé vált a tesztelés különböző processzor, illetve memória foglaltságok mellett.

A gépek a tesztelés alatt fizikailag külön Ethernet-hálózaton kerültek elhelyezésre. A tesztelések folyamán a következő szempontokat vettem figyelembe:

- különböző hardware konfigurációk,
- három különböző összetettségű tűzfal szkript,
- 2.2-es illetve 2.4-es kernelszéria,
- az általam vizsgált paraméterek értékei,
- a rendszer terheltsége.

3. Eredmények, konklúzió

3.1. netdev_max_backlog

Első megfigyelésem az volt, hogy ezen érték növelésekor az arp kérések feloldása – főként 900 felett lassulni kezdett.

A három különböző teszt tűzfal szkripttel végzett kísérletek a várt eredményt produkálták: A sebességbeli különbség legmarkánsabban az egyszerű tűzfalnál jelentkezett, az átlagosnak mondhatónál közepes, az összetettebnél pedig gyengébb sebességbeli hatásokat figyeltem meg.

Mindazonáltal a különbségek olyannyira minimálisak hogy azok a mérési hibahatárt súrolják, s még ha meg is bízom a kísérleteimben, nem képviselnek releváns eltérést.

A tesztelések alapján megállapítható hogy az optimális teljesítmény PC-ken, valamint alpha-n a 900-as beállításra, míg a SUN architektúrákon az 1200-asra esik.

Meg kell azonban jegyezni hogy a teljesítmény-görbének ez a két csúcspontja, mind a PC-knél mind pedig a SUN rendszereknél, így mivel mint fentebb említettem az 1200-as értéknél az arp kérések már láthatóan megsínylik a beállítást, praktikus mindkét rendszernél a 900-as értéknél maradni.

Megállapítható továbbá hogy a kernelparaméter leginkább két esetben hat a hálózati forgalom sebességére :

- Abban az esetben ha az adott hardware-t terheljük le teljesítőképessége határáig. (egy esetleges magas terheltség, vagy agresszív swap használat során)

- Extrém hálózati terhelés esetén.

Mint ebből is kitűnik e kernelparaméter értéke nem elsősorban mint általános sebességnövelési tényező játszik szerepet, s így nem a kernel tuningolók kedvenc játékszere, hanem gépünk stabilitását javítja, azáltal hogy pozitív hatását a nagyobb terhelések alatt fejti ki legmarkánsabban.

Meglepetést okozhat például egy flood esetén amikor az (ana)cron démon által automatikusan indított slocate() adatbázis futása alatt kísérlik meg gépünket leterhelni a hálózaton.

3.2. rmem_max, wmem_max

Az rmem_max és wmem_max értékeiről elmondhatók hogy a lokális processzek annál gyorsabb működésre képesek, minél nagyobb értéket adunk ezeknek, persze csak ha van elég feláldozható memóriánk.

Néhány régebbi alkalmazás azonban – például a tftp – instabillá vált ezen értékek átállításának hatására.

Néhány programnál pedig – például samba – az igazi teljesítménynövekedés akkor jelentkezik, ha a következő szűk keresztmetszetet is megszüntetjük, és megemeljük a `/proc/sys/net/unix/max_dram_qlen` értékét is.

Az eddig végrehajtott tesztek során keletkezett eredmények, tekintve hogy szélsőséges, s ideális körülmények között mért tesztadatokról van szó, nem használhatók fel egy abszolút skálán történő kalibrációhoz.

Folyamatban van a mérések kiterjesztése működésben lévő gépekre, hogy valós, jobban felhasználható adatokhoz juthassak.

Természetesen ez utóbbi jóval nagyobb mintavételezést követel meg mint az eddig végrehajtott tesztek.

A tesztek számadatai megtalálhatók a `http://firewall.ttk.pte.hu/tuzfalteszt/` weboldalon, napról napra több adatból generálva az eredményeket, és ugyanitt folyik a napi működés közbeni tesztelések kiértékelése is.

A hosszú távú cél a vizsgálat kiterjesztése néhány további, applikáció specifikusabb paraméter vizsgálatára, s egy olyan script összeállítása, amely ellenőrizve a gépen található programokat, a memória méretét, és a processzor teljesítményét automatikusan optimális értékekre állítja a rendszermag paramétereit.

4. Függelék

4.1. a tesztgépek leírása

A dokumentum elkészítéséig használt tesztgépek:

- Intel 5x86 100 MHz, 8 Mebibájt RAM. LAN: 3c509c (10Mb UTP),
- Pentium III. 800 MHz, 128 Mebibájt RAM. LAN: SMC PCI 10/100 UTP,
- DS-20-Alpha 500 MHz, 512 Mebibájt RAM. LAN: Eepro100 10/100 UTP,
- SUN Sparc-station 20, 64 Mebibájt RAM. LAN: integrált vezérlő.

FTP teszt szkript

```
#!/bin/bash
for i in $(seq 3 24); do
  sync
  ifconfig eth0 down
  ifconfig eth0 10.10.2.2 up
  echo ${i}00 > /proc/sys/net/core/netdev_max_backlog;
  echo "size:1byte netdev_max_backlog: ${i}00" >> data.ftp
  echo "size: $s netdev_max_backlog: ${i}00"
  cat alma|ftp -v >> data.ftp
  sleep 1
done
```

PING teszt szkript

```
#!/bin/bash
c=100000
for s in 64 2048 4096 8192 16384 32708; do
  for i in $(seq 3 24); do
    sync
    ifconfig eth0 down
    ifconfig eth0 10.10.2.2 up
    echo ${i}00 > /proc/sys/net/core/netdev_max_backlog;
    echo "size: $s netdev_max_backlog: ${i}00" >> data.ping.$s
    echo "size: $s netdev_max_backlog: ${i}00"
    ping -f -s $s -c $c 10.10.2.2 >> data.ping.$s
    sleep 1
  done
done
```

WWW teszt szkript

```
#!/bin/bash
for s in 512 1024 4096 8192; do
  for i in $(seq 3 24); do
    sync
    ifconfig eth0 down
    ifconfig eth0 10.10.2.2 up
    echo ${i}00 > /proc/sys/net/core/netdev_max_backlog;
    echo "size:$s netdev_max_backlog: ${i}00" >> data.www.$s
```

```
    echo "size: $s netdev_max_backlog: ${i}00"
    wget -a data.www.$s 10.10.2.2/index${s}.html
    rm -f index${s}.html
    sleep 1
    wget -a data.www.$s 10.10.2.2/index${s}.html
    sleep 1
    rm -f index${s}.html
  done
done
```

Csomagszűrő tűzfalak – netfilter

Kadlecsik József
KFKI Részecske és Magfizikai Kutatóintézet
<kadlec@blackhole.kfki.hu>

2002. október 21.

Kivonat

Az előadásban összefoglaljuk a Linux 2.4-es kernel-sorozatában található csomagszűrő tűzfal-funkció tulajdonságait, képességeit. Kitérünk az új lehetőségekre és a jövőbeli fejlesztés irányaira.

Tartalomjegyzék

1. Bevezetés

A Linux kernelben, annak meglehetősen korai verzióitól fogva létezett csomagszűrő funkcionalitás, amely több lépésben fejlődött, változott. A lépcsőket általában a usertérbeli konfigurációs program nevéhez szokták kötni: *ipfw*, *ipfwadm*, *ipchains*, és a legutolsó a sorban, az *iptables*. Az *iptables* kernel-beli „párja” a netfilter.

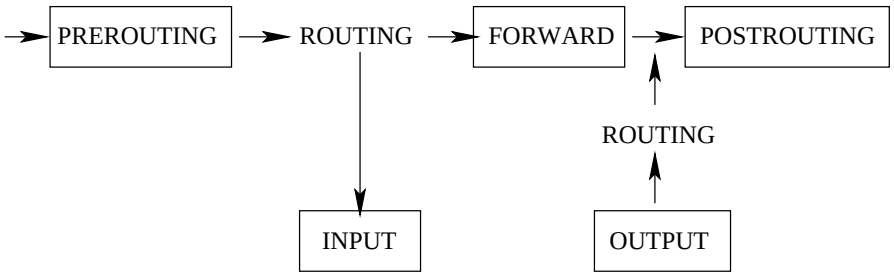
A netfilter a 2.3-as fejlesztői kernellel jelent meg, a 2.4-es stabil kernel-sorozat már ezt tartalmazza. Léteznek „backward-compatibility” modulok, amelyekkel az *ipchains*, sőt az *ipfwadm* formálisan tovább használható a netfilter konfigurálására, de ez inkább nem ajánlott: áttetetnénk a jó öreg Trabi kormánykerekét a vadonatúj Mercedesünkre, csak azért, mert azt szoktuk meg?

2. A netfilter szerkezete

A netfilter néhány jól definiált komponens együttese és azok modulokban való kibontása:

- A Linux hálózati stack-jeiben (IPv4, IPv6, stb) jól definiált „kampók”, belépési pontok lettek meghatározva a netfilter keretrendszer számára. Ahogy egy csomag a stack-ben feldolgozódik, a stack ezeknél a pontoknál hívja meg a netfiltert.
- A netfilter alrendszerei a kampókhoz regisztrálják magukat, különböző prioritásokkal. Amikor a netfilter a stack-tól megkapja a vezérlést, a prioritások alapján sorra meghívja a regisztrált alrendszereket, amelyek megvizsgálhatják (megváltoztathatják) a csomagot és eldönthetik annak sorsát:
  - ACCEPT: a csomag folytassa az útját a netfilterben és a stack-ben
  - DROP: dobja el a csomagot
  - STOLEN: az alrendszer „ellopta” a csomagot, a stack felejtkezzen el róla
  - QUEUE: a csomag kerüljön át egy queue-ba a usertér felé
- A netfilter része egy aszinkron queue mechanizmus, amellyel csomagok a usertérnek – egy abban futó processznek – átadhatók.
- Végül a rendszer nagyon fontos része a dokumentáció, a howto-k, FAQ, a patch-o-matic és a forráskód :-).

IPv4 esetén a stack öt jól definiált kampónál hívja meg a netfiltert:

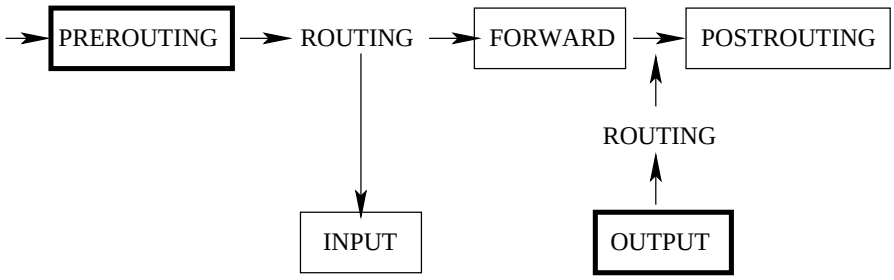


- PREROUTING: ahogy egy interfészről a stack-be jut a csomag, még routing előtt

- FORWARD: routing után, ha a csomag nem lokális processznek szól
- POSTROUTING: routing után, mielőtt a csomag egy interfészen elhagyná a stack-et
- INPUT: routing után, ha a csomag lokális processznek szól
- OUTPUT: routing előtt, ahogy egy lokális processztől a stack megkapja a csomagot

Táblázatot tartalmazó alrendszer esetén a kampókból következik, hogy milyen beépített láncok léteznek a táblázatban.
A következő fejezetekben az IPv4-es alrendszereket tekintjük át.

3. A raw alrendszer



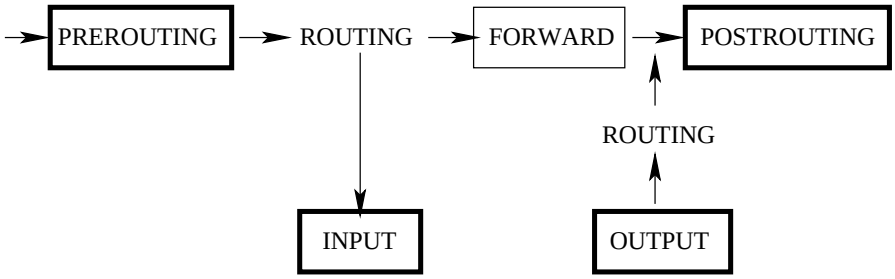
A raw alrendszerbe a PREROUTING és OUTPUT kampókon keresztül jutunk. Ez a legelső alrendszer a netfilterben, minden mást megelőz. Ebből eredően az elsődleges szerepe az, hogy a következő alrendszerek viselkedését módosítsa:

- Egy-egy csomag megjelölése azért, hogy a netfilter naplózza, ahogy a csomag az egyes alrendszereken és azok szabályain áthalad (TRACE). A funkció kiválóan alkalmas arra, hogy saját szabályrendszerünket szelektív módon ellenőrizhessük, hogy valóban úgy és azokra a csomagokra érvényes-e, amelyekre azt terveztük.
- Egyes csomagok megjelölése azért, hogy a netfilter kapcsolat-nyomkövető (connection tracking) alrendszere – és ebből következően a NAT alrendszer – figyelmen kívül hagyja azokat (NOTRACK). Noha a kapcsolat-nyomkövetés elhagyásával a netfilter egyik legfontosabb új eleméről mondunk le, néhány esetben erre kifejezetten szükség van: például a netfilter failover esetében majd erre támaszkodni kell.
- A raw alrendszer módot adhat arra is, hogy a connection tracking-et „rávegyük” olyan csomagok kezelésére, amelyeket egyébként nem fogadna el (pl. ICMP hibacsomagok PMTU felderítés közben, privát IP című részhálózat közbeékelődése esetén).

A raw alrendszer jelenleg még a patch-o-matic-ban található, nem integráns része a netfilter-nek.

4. Kapcsolat-nyomkövetés

A connection tracking alrendszerrel válik a netfilter „stateless” csomagszűrő tűzfalból "stateful" csomagszűrő tűzfallá. Ez nem csak az egyes IP kapcsolatok egyszerű követését jelenti, hanem beépített ICMP hibaüzenet-kezelést, valamint a segédkapcsolatokat használó támogatott protokollok esetén a segédkapcsolatok automatikus kezelését is.



A connection tracking alrendszerbe a PREROUTING, OUTPUT valamint POSTROUTING és INPUT kampókon keresztül jutunk. Elvileg a PREROUTING és OUTPUT kampók elégségesek lennének, de két (rejtett) tulajdonság miatt a másik kettőre is szükség van:

- Az új kapcsolatot jelentő csomagok új elemeket adnak hozzá a kapcsolat-nyomkövetés hash táblázatához. Ez meglehetősen rossz hatásfokú lenne, ha olyan csomagokból származó elemek is hozzáadódnának, amelyeket később a szűrő alrendszerben eldobunk. Ezért az új elemek csak a POSTROUTING és INPUT kampóknál kerülnek be a hash-be, amikor már „túlélték” a szűrési szabályokat (a kapcsolat-nyomkövetés a POSTROUTING és INPUT kampóknál a legalacsonyabb prioritású, azaz az utolsó).
- A kapcsolat-nyomkövetés frgamentált csomagokra nem működik, ezért a PREROUTING/OUTPUT kampóknál automatikus defragmentáció történik. Ennek megfelelően az alrendszernek POSTROUTING-nál a csomagokat szükség esetén újra kell fragmentálnia.

A connection tracking-ben az egyes kapcsolatok állapottere jól definiált – és kicsi. Egy kapcsolat és az ahhoz tartozó csomag állapota lehet:

- NEW: ha a csomag egyetlen létező kapcsolathoz sem tartozik
- ESTABLISHED: ha a csomag egy már létező kapcsolathoz tartozik (és nem újraküldött NEW csomag)
- RELATED: ha a csomag létező kapcsolathoz tarozó ICMP hibaüzenetet hordoz, vagy egy segédkapcsolat csomagja
- INVALID: a csomagot a kapcsolat-nyomkövető rendszer nem tudta kezelni.

A kapcsolatok nyomonkövetése TCP esetén a legegyszerűbb – azonban rögtön beleütközünk egy fogalmi eltérésbe, amely félreértésekre szokott vezetni: conntrack-ban egy NEW állapotú TCP csomag nem feltétlenül kell, hogy egy TCP SYN csomag legyen. A NEW állapot egyetlen feltétele, hogy a csomag ne tartozzon már létező kapcsolathoz. Egy közönséges ACK csomag állapota is lehet NEW!

A kapcsolat-nyomkövetés azért viselkedik így, mert nevéből következően egyetlen feladata a kapcsolatok lehető legjobb nyomon követése. Ugyanakkor ezzel lehetőség nyílik arra is, hogy egy újraindított tűzfal a már fölépült, engedélyezett TCP kapcsolatokat azok blokkolása helyett „on-the-fly” újra kezelni tudja.

Az UDP nem kapcsolat-orientált protokoll, ennek ellenére a legtöbb esetben kérdés- és válasz csomagokból egy virtuális kapcsolat épül föl, amely timeout mechanizmussal kielégítően kezelhető.

Az ICMP kezelése – ahogy arra részben utaltunk már – két részre van választva. Az ICMP hibaüzeneteket a rendszer a már létező kapcsolatokkal veti össze, és ha talál egyezőt, akkor az ICMP csomagot a RELATED kategóriába sorolja. Nem ICMP hibaüzenetek (pl ICMP echo request/reply) kapcsolatként kezelődnek úgy, hogy a válaszcsomag detektálása után a conntrack hash-ből az egység azonnal törlődik.

Segédcsatornákat használó protokollok közül a következők támogatottak:

- FTP (aktív és passzív mód egyaránt) valamint IRC
- patch-o-matic-ból: PPTP, talk (minden variáns), H.323, tftp, stb.

Modularizált kernel esetén ezen protokollok támogatását ún. helper modulok végzik, amelyek *nem* töltődnek be automatikusan. Erről a tűzfal adminisztrátorának kell gondoskodni, különben az adott protokoll támogatása hiányozni fog a kernelből. A modulok kilistázásakor látható számlálóban nem a kezelt kapcsolatok száma tükröződik, hanem hogy hány más modul függ az adott helper modultól (newnat előtt 0, newnat után és NAT mellett 1).

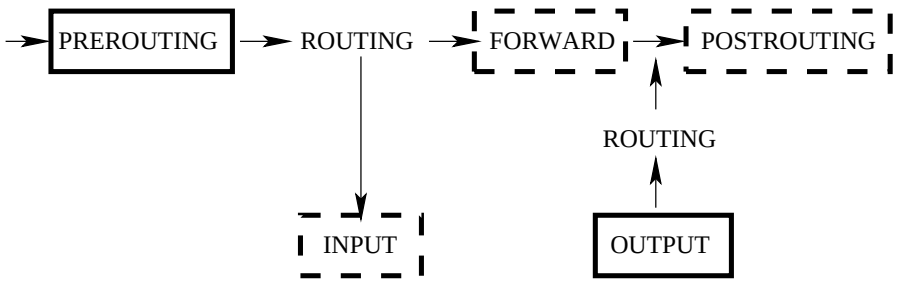
A kapcsolat-nyomkövetés hash táblázatába rögzített mennyiségű kapcsolat „fér bele”, ahol a kapcsolatok maximális száma alapértelmezésben a gép fizikai memóriájának méretétől függ – ezt elérve a rendszer kénytelen csomagokat eldobni. A kapcsolatok maximális száma két paraméterrel is állítható: az ip_conntrack modul hashsize paraméterével, valamint a `/proc/sys/net/ipv4/ip_conntrack_max` paraméterrel. A kettő között az összefüggés a következő:

$$/proc/sys/net/ipv4/ip\_conntrack\_max = 8 * hashsize$$

Az az optimális, ha a kapcsolatok maximális számát a hashsize paraméterrel növeljük meg, és nem az ip_conntrack_max-on keresztül.

A kapcsolat-nyomkövetés alrendszeréhez nem tartozik táblázat, így a működését csupán a patch-o-matic-beli raw táblázaton keresztül lehet befolyásolni.

5. A mangle alrendszer

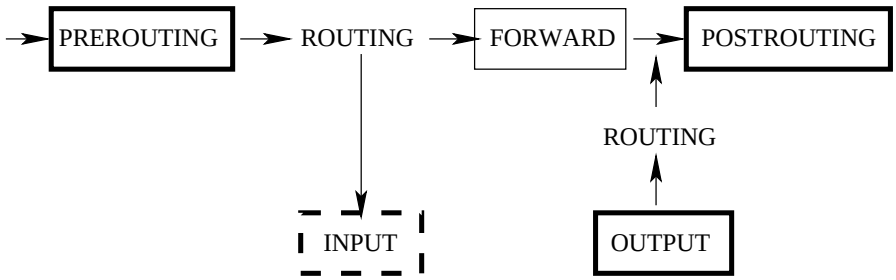


A mangle alrendszerbe a 2.4.19-es kernel előtt a PREROUTING és OUTPUT, míg a 2.4.19-től kezdődően az összes kampón keresztül juthatunk el.

A mangle alrendszer a csomagok általános IP és routing paramétereinek a módosítására szolgál:

- a csomagok „megjelölése” (MARK) speciális routing érdekében
- DSCP/TOS mező átállítása a csomagok továbbítási prioritásának a módosítására
- TCP MSS paraméter „kézzel” való beállítása, amikor a kommunikáló partnernél az ICMP fragmentation Needed csomagokat szükségtelenül blokkolják
- Time To Live paraméter átállítása
- az IP és TCP ECN paraméterek átállítása ECN-t nem támogató partnerek felé

6. A NAT alrendszer



A NAT (Network Address Translation) két típusa lehetséges:

- SNAT: Source NAT, amikor a csomag forráscímét (és portját) módosítjuk
- DNAT: Destination NAT, amikor a csomag célcímét (és portját) módosítjuk

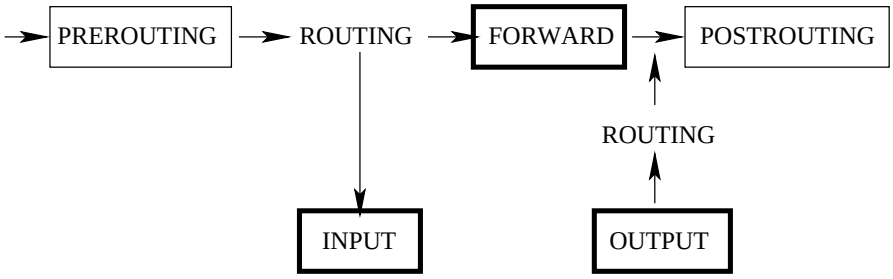
DNAT-ra a PREROUTING és OUTPUT kampóknál van lehetőségünk – SNAT-ra a POSTROUTING, és – 2.4.19 óta – az INPUT kampóknál. Az SNAT speciális esete a MASQUERADE, amikor a csomag egyszerűen a küldő interface IP címét kapja forráscím gyanánt. A DNAT speciális esete a REDIRECT, amikor a csomagot a lokális interfészre (127.0.0.1) irányítjuk át.

NAT-olás történhet IP címtartományra, – akár többre is – ekkor az új kapcsolatok mindig a legkevésbé használt IP címre íródnak át. Átfedő NAT szabályok megengedettek, akár valódi, használt IP címekre is (feltéve, hogy azon gépek forgalma áthalad a tűzfalon). Ha egy kapcsolatra nincs szabály, akkor a nat táblázat default ACCEPT policy-ján keresztül NULL mapping történik (azaz NAT az eredeti címekre). Erre azért van szükség, hogy átfedő NAT szabályok esetén se fordulhasson elő két kapcsolat ugyanazon IP cím/port paraméterekre való címfordítása.

A NAT alrendszerben, ameddig csak lehetséges, az eredeti port őrződik meg. Ha ütközést kell elkerülni, akkor implicit source port átírás történik a következő három port-tartományon *belül*:

- 0-511
- 512-1023
- 1024-65535

7. A szűrő (filter) alrendszer



A filter alrendszert az INPUT, OUTPUT, valamint a FORWARD kampókon keresztül érjük el.

Az ábrából világosan látható, hogy a lokális gépnek/gépről küldött csomagokra vonatkozó szűrési szabályokat az INPUT és OUPUT, míg az átmenő csomagokra vonatkozókat a FORWARD láncban állíthatjuk be. (Az *ipchains* „logikájához” hozzászokott felhasználók számára ez nagy változás és sok félreértés forrása.)

A szűrőszabályokban a NAT-ról meg lehet feledkezni: a filter tábla mindig a a valódi kommunikációt végző IP címeket „látja”.

8. Csomag-egyeztetési lehetőségek

A szabályokban a csomagokkal való egyeztetés (packet matching) lehetőségei szinte végtelenek :-). Az alapértelmezett netfilterben a következők állnak a rendelkezésünkre:

- protokoll szerinti egyezés
- forrás- és célcím[/maszk]
- input (INPUT, FORWARD, PREROUTING) és output (FORWARD, OUTPUT, POSTROUTING) interfészek (az adott láncokban) szerinti egyezés
- TCP esetén forrás- és célport (vagy tartomány), TCP flag-ek (SYN, FIN, stb.) és kombinációik, TCP opciók (MSS érték vagy tartomány)
- UDP esetén a forrás- és célport (vagy tartomány)
- ICMP esetén típus[/kód]
- connection tracking-beli állapot szerint (state): NEW, ESTABLISHED, RELATED, INVALID
- lokális szegmensen MAC cím szerinti egyezés
- lokális gépen a csomagot keltő processz paraméterei alapján (owner): uid, gid, pid, sid, cmd
- érvénytelen csomagok (unclean): rossz checksum, érvénytelen TCP flag-kombináció, stb.
- tos, ttl mezők értéke, csomagméret (length), mark, stb.

A feltételeket lehet negálni, és egyetlen szabályban tetszőleges számú különböző fajta egyezési feltétel szerepelhet.

9. Patch-o-matic

A netfilter keretrendszerhez jellegénél fogva nagyon könnyű bővítéseket írni. Azonban ahhoz, hogy a független bővítések lehetőleg ne ütközzenek egymással, egy patch kezelő rendszerre van szükség – ez a patch-o-matic. A patch-o-matic-ban a patch-ekhez tartozhat egy vagy több speciális formátumú kisegítő patch-file, ezért a kézzel való patch-alkalmazás szinte mindig hibás kernelforrást eredményez. **Minden** patch-hoz tartozik egy help file, amely leírja, hogy az adott patch milyen új funkciót vagy javítást tartalmaz.

A patch-eket csoportokba szervezzük, amelyek egységenként kezelhetők. Egy-egy csoport részhalmazként hivatkozhat egy vagy több másik csoportra:

- submitted: patch-ek, amelyeket a legfrissebb kernel már tartalmaz
- pending: patch-ek, amelyeket a következő kernel már tartalmazni fog
- base: alap patch-ek, amelyek (általában) nem ütköznek egymással és amelyek egyszerűek vagy jól teszteltek
- extra: patch-ek, amelyek vagy ütköznek egyik-másik patch-el, vagy még alaposabb tesztelés vár rájuk, vagy kérdéses funkcionalitást valósítanak meg.

Általában a patch-ek függenek a submitted és pending patch-ektől, így azok alkalmazása gyakorlatilag kötelező :-).

Jelenleg a patch-ek száma körülbelül hetven – csupán a fontosabbakat, érdekesebbeket említjük meg.

- *newnat*: kibővített conntrack/NAT API több segédcsatornát használó protokollok támogatására
- *NETMAP target*: statikus 1:1 S/DNAT
- *SAME target*: SNAT mindig ugyanarra az IP címre egy (kisebb) tartományból.
- *iplimit match*: maximális parallel TCP kapcsolatok száma/kliens IP cím (vagy tartomány)
- *nth match*, *random match*
- *psd match*: portscan detektor
- *recent match*: egyezés a „mostanában” látott kapcsolatokkal (port scanning, stb. kivédésére)
- *quota match*: (byte számláló)
- *time match*
- *CONNMARK target* és *connmark match* kapcsolatok megjelölésére (mark megfelelője kapcsolatokra)

A patch-o-matic a netfilter oly fontos része, hogy az *iptables* csomagtól függetlenné vált és önálló csomagban tölthető le.

10. A netfilter jövője

A netfilter a 2.4-es kernelsorozat alatt is igen sokat fejlődött (lásd például newnat API bevezetése) és a jövőben is több fontos előrelépés várható:

- Jelenleg a konfigurációs program protokoll-család függő: az *iptables*-t kell használni az IPv4-es, az *ip6tables*-t az IPv6-os szabályok beállítására. Folyik a munka annak érdekében, hogy egy, az összes protokoll-család számára közös program készüljön, elkerülve ezáltal a fölösleges kód-duplikálást.
- Készül egy új, koherens API, amellyel a user-interfész tetszőleges programból (GUI, Perl, Python, stb.) kényelmesen elérhető lesz.
- A kapcsolat-nyomkövető réteg meglehetősen elkülönül, működése jelenleg semmilyen módon nem befolyásolható. Az ehhez szükséges API-n szintén folyik a munka.
- Az *iptables* a szabályokból egyetlen bináris adattömböt, táblázatot készít, amelyet aztán átad a kernelnek. Ez nagyszámú szabály esetén az újak hozzáadását egyre lassabbá teszi: az *iptables* lekéri a kerneltől az összes szabályt, hozzáadja az újat, majd a teljes táblázatot visszaadja a kernelnek. (Lényegében emiatt van szükség az *iptables-save* és *iptables-restore* parancsokra.)

  A jelenlegi mechanizmus helyett a szabályok láncolt listában való tárolása fogja jelenteni a valódi megoldást, amelynek az implementálása szintén a közeljövőben várható.
- A connection tracking jelenleg teljesen IPv4 függő. Készül egy protokoll-családtól független kapcsolat-nyomkövető alrendszer, amellyel az IPv6-os kapcsolat-nyomkövetés, NA(P)T-PT, DSTM, stb. támogatása válik lehetségessé.
- A netfilter-failover kapcsolat-nyomkövetéssel együtt való megvalósítása szintén tervezés alatt áll, amellyel a nagyfokú rendelkezésre állást lehet majd biztosítani.

A CVS-ben tárolt forráskód, patch-o-matic, FAQ, HOWTO-k, levelelezési listák és a netfilterhez kötődő egyéb információk a `http://www.netfilter.org/` címen érhetők el.

A PHP-GTK

Körmendy Domonkos
<doma@kormendy.hu>

2002. szeptember

Kivonat

A PHP egy ingyenes, könnyen tanulható scriptnyelv, a GTK pedig egy, több nyelvhez illeszthető, grafikus felület készítésére használható eszközkészlet.
Ennek a két nagyszerű eszköznek az összekapcsolására született a PHP-GTK multiplatformos kiterjesztés, amit előadásomban megpróbálok bemutatni.
Köszönettel tartozom *Mátyási István*nak, akivel együtt fedeztük fel a PHP-GTK szépségeit és együtt dolgoztunk a többször emlegetett fejlesztőesztköz elkészítésén a BME-n, 2002 tavaszán.

Tartalomjegyzék

1. Bevezető

1.1. A PHP

A PHP-t valószínűleg senkinek sem kell bemutatnom, azonban szeretném felvillantani néhány, az előadásom szempontjából fontos tulajdonságát. Ezek a következők:

- Az egyik leginkább elterjedt webes scriptnyelv, terjedése évek óta töretlen
- Multiplatform
- Ingyenes
- Könnyen tanulható
- Rendkívül sok kiegészítés létezik hozzá, gyakorlatilag az összes elterjedtebb protokollt tudja használni
- Kérés-vezérelt: egyszeri lefutással bizonyos bejövő paraméterekből bizonyos kimenetet produkál

1.2. A GTK+

A Gtk+ egy nyílt forráskódú, multiplatform eszköztár grafikus interfészek (GUI-k) létrehozásához. Alapvetően a C++ nyelvhez készült, de sok más nyelvvel is használható már. A következő főbb komponensekből áll: Gtk (ún. widgetek csoportja), GDK (wrapper alacsony szintű ablakozáshoz), GLib (adattípusok, ezekhez alap függvények; a PHP-GTK nem támogatja).

A Gtk+ gyakorlatilag egy osztály-hierarchiát nyújt, melynek elemeiből GUI-t építhetünk. Az egyes osztályok a képernyőn megjelenő elemek. Ezeknek egyrészt szabályozhatjuk a megjelenését, másrészt a hozzájuk kötődő eseményekhez eseményvezérlőket rendelhetünk. (Pl. meghatározhatjuk, mely függvény hívódjon meg, amikor egy gombra kattintunk.)

1.3. A PHP-GTK

Néhány fejlesztőnek eszébe jutott, hogy ötvözni kellene a fenti két, látszólag teljesen összeegyeztethetetlen szoftvert. 2000 őszén nekiláttak, és 2001 márciusában elkészült az első verzió. Az első teljes stabil kiadásra pedig 2002 januárjáig kellett várni.

A PHP-GTK egy PHP extension, mely elérhetővé teszi a GTK widgeteket, mint objektumokat. 95 osztályt és több mint 970 függvényt ad a PHP sajátjaihoz, így az egyik legnagyobb kiterjesztése annak.

2. A PHP-GTK

2.1. Előnyök

A PHP és a GTK összekapcsolásából a következő előnyök származnak:

- Esemény-vezérelt programokat írhatunk PHP-ban,
- GUI-t készíthetünk és használhatunk,

- A programok lokálisan futnak, és hozzáférnek a gép erőforrásaihoz,
- Multi-platform, ingyenes szoftverkörnyezetet kapunk,
- A PHP könnyű programozhatósága megmarad,
- A PHP sok extension-je mind használható.

2.2. A PHP-GTK bemutatása egy példaprogramon keresztül

A példaprogram

A következő program egy „advanced hello world”. Egy ablakot hoz létre, abban egy többsoros szövegdobozt és egy gombot. A gombra kattintva a szövegdoboz végére hozzáfűzi a „hello world!” szöveget. Az ablakot bezárva pedig befejezi futását a program.

```
<?php

  // extension betoltese
  if (!extension_loaded('php_gtk')) {
    dl('php_gtk' . (substr(PHP_OS, 0, 3) == 'WIN' ? '.dll' : '.so'));
  }

  // signal handler: kilep az alkalmazasbol
  function quit() {
    Gtk::main_quit();
  }

  // signal handler: a textbox vegehez fuzi a 'Hello world!' szoveget
  function hello() {
    global $textbox;
    $textbox->insert_text("Hello world! \n", $textbox->get_length());
  }


  // a foablak letrehozasa
  $window =& new GtkWindow();
  $window->set_border_width(1);

  // bezarashoz a kilepo fuggveny hozzarendelese
  $window->connect('destroy', 'quit');


  // vertikalis doboz (kontener)
  $vbox =& new GtkVBox();
  $window->add($vbox);


  // textbox letrehozasa es elhelyezese
  $textbox =& new GtkText();
  $textbox->set_editable(true);
  $vbox->pack_start($textbox);


  // gomb letrehozasa es elhelyezese
  $button =& new GtkButton('Click!');
  $button->connect('clicked', 'hello');
  $vbox->pack_start($button);

  // mindent megjelenitunk
```

```
  $window->show_all();


  // indul a program (a fo ciklus)
  Gtk::main();

?>
```

Magyarázat

Lássuk sorban az egyes részeket, magyarázattal:

```
// extension betoltese
if (!extension_loaded('php_gtk')) {
   dl('php_gtk' . (substr(PHP_OS, 0, 3) == 'WIN' ? '.dll' : '.so'));
}
```

A programunk elején ellenőrizzük, be van-e töltve a php_gtk extension, ha nem, akkor betöltjük. Látható, hogy a fájl kiterjesztése miatt elágazás van operációs rendszerek szerint. A multiplatform működéshez ennyi elegendő is!

```
// a foablak letrehozasa
$window =& new GtkWindow();
$window->set_border_width(1);
```

Létrehozzuk a főablakot. Fontos észrevenni, hogy `=& new`-t használunk. Ha nem ezt tennénk, akkor a `$window` változó nem a létrehozott ablak referenciáját, hanem annak egy másolatát tartalmazná. Ekkor hiába próbálnánk beállítani pl. az ablak keretének vastagságát (mint a példaprogramban), nem történne változás a képernyőn. Új widgetek létrehozásakor használjunk tehát mindig az `=& new`-n-t. (Egyesek csak `&new`-t mondanak - szerintem ez azonban az értékadáshoz kötődik, nem a létrehozáshoz...) Egyedüli kivétel, amikor egy függvényen belül globálisként definiált változónak adjuk az értéket, mert ebben az esetben a PHP automatikusan referenciát tárol benne. (De ekkor sem baj, ha kirakjuk az &-et.) (Ez a kellemetlenség sajnos a PHP működéséből adódik.)

```
// signal handler: kilep az alkalmazasbol
function quit() {
  Gtk::main_quit();
}

// bezarashoz a kilepo fuggveny hozzarendelese
$window->connect('destroy', 'quit');
```

Az egyes widgetek minden rendszer-esemény bekövetkeztekor és felhasználói beavatkozáskor üzeneteket küldenek, melyeket signaloknak nevezünk. Ezek a signalok a widget-hierarchián addig adódnak tovább, míg valamelyik le nem kezeli őket.

Minden widget minden signal-típusához tetszőleges számú kezelőfüggvényt, ún. callbacket rendelhetünk. Ezek a callbackek megkapják a hívó widgetet (tehát egy függvény több widget azonos (vagy akár különböző) signalját is lekezelheti), illetve egyéb paramétereket is átadhatunk nekik.

A fenti példában a főablak destroy signaljához egy kilépést végző függvényt rendelünk. (Enélkül ugyanis hiába kattintgatnánk az ablakot bezáró „X”-re.)

```
// mindent megjelenitunk
$window->show_all();
// indul a program (a fo ciklus)
Gtk::main();
```

Az ablak objektumunk show_all() metódusát meghívva megjelenik a képernyőn az ablak és rekurzívan az összes benne tárolt widget. A Gtk::main() pedig elindítja az eseményvezérlést, és a program ettől kezdve a felhasználó beavatkozására vár.

```
// vertikalis doboz (kontener)
$vbox =& new GtkVBox();
$window->add($vbox);

// textbox letrehozasa es elhelyezese
$textbox =& new GtkText();
$textbox->set_editable(true);
$vbox->pack_start($textbox);
```

Hogy ne legyen üres az ablakunk, létrehozunk egy vertikális dobozt, mely további widgetek pozicionálására szolgál. (A vertikális doboz egymás alatt jeleníti meg a hozzáadott widgeteket. Párja a horizontális doboz, mely pedig egymás mellett.) Ezt belehelyezzük az ablakunkba, az add() metódus segítségével. Majd létrehozunk egy textboxot, szerkeszthetővé tesszük, és hozzáadjuk a vertikális dobozhoz. (A pack_start() annyiban különbözik az add()-tól, hogy precízebb elhelyezést tesz lehetővé. Szabályozhatjuk vele pl. a widgetek közti távolságot. Bővebb információ a dokumentációban található.)

```
// signal handler: a textbox vegehez fuzi a 'Hello world!' szoveget
function hello() {
  global $textbox;
  $textbox->insert_text("Hello world! \n", $textbox->get_length());
}

// gomb letrehozasa es elhelyezese
$button =& new GtkButton('Click!');
$button->connect('clicked', 'hello');
$vbox->pack_start($button);
```

Végül létrehozunk egy gombot, és hozzárendelünk egy kezelőfüggvényt, mely a textboxba ír bele.

Ha nem szeretnénk globális változókat használni (nem is illik! :-))), akkor a következő megoldást is választhatjuk: a callback paraméterként kapja meg a textboxot, melybe írnia kell. Ezt pedig így lehet megvalósítani:

```
// signal handler: a textbox vegehez fuzi a 'Hello world!' szoveget
function hello($sender, $param) {
  $param->insert_text("Hello world! \n", $param->get_length());
}
// gomb letrehozasa es elhelyezese
$button =& new GtkButton('Click!');
$button->connect('clicked', 'hello', &$textbox);
$vbox->pack_start($button);
```

2.3. Korlátok

A PHP-GTK sajnos korlátokkal is rendelkezik. Ezek többsége a Zend Engine-ből (a PHP „motorjából") következik, és a későbbiekben megszűnhet:

- Az `=& new` szükségessége,
- A változók csak az alkalmazás befejeződésekor szabadulnak fel,

- Objektumok tagváltozóinak metódusai nem hívhatóak, közvetlenül, (pl. `$FileSelect->ok_button->set_border_width(1);`)

- Összetettebb ablakokat nehéz létrehozni kódból. (megoldás: Glide)

2.4. Felhasználás

A PHP-GTK egyrészt kis, egyszerű, de mindenképp GUI-t igénylő alkalmazások elkészítéséhez lehet a megfelelő nyelv. Ilyen programok már szép számmal léteznek: van pl. e-mail kliens, news-olvasó, adatbázis-manager, portál távoli szerkesztőrendszere. Másrészt nagyobb, többrétegű alkalmazásokban is használható, vékony- ill. vastag-kliens elkészítéséhez.

Ha megjelenik hozzá egy megfelelő fejlesztőrendszer, akkor igazi RAD-eszköz válhat belőle. Ezzel komoly versenytársa lehetne a Visual Basic, Delphi, Visual C++, Kylix eszközöknek is, a hordozhatósága, egyszerűsége és ingyenessége miatt. Egyelőre csak egy Glade kiegészítésről tudok, mely segít ezen, de annak körülményes a használata.

2002 tavaszán egy ilyen eszköz fejlesztésébe fogtunk a BME-n, ez a munka azonban más irányú elfoglaltságok miatt szünetel. Ha akadna jelentkező, ki beszállna a fejlesztésbe (mert lát benne fantáziát), akkor lehetne folytatni[1]. Kedvcsinálónak egy screenshot látható a programról az utolsó fejezetben.

3. Összegzés

3.1. A jövő

A PHP-GTK utolsó kiadása óta elkészült a Gtk+ 2-es verziója. Ez többek között a multiplatform képességeken javít, és módosít illetve hozzáad új widgeteket. Az erre való átállás a Zend Engine 2 támogatásával együtt várható.

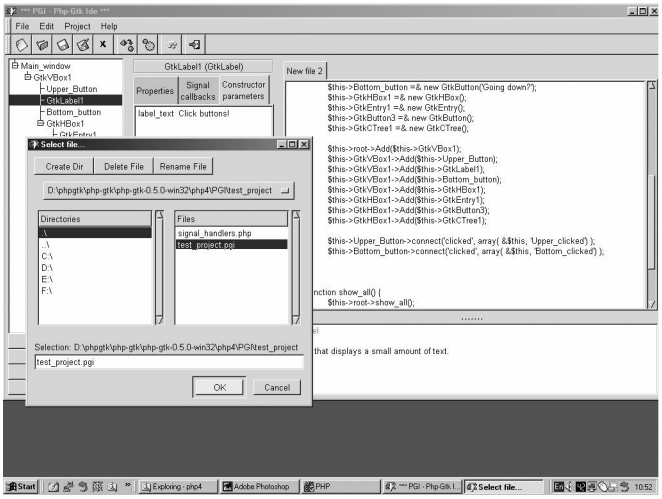
A Zend Engine 2 valamikor 2003 során fog megjelenni, és várhatóan nagy lökést ad majd a PHP-nek. Objektumorientált szemlélettel alkotják újra a belső objektum modellt és pl. kivétel-kezelést is használhatunk majd.

Ezek támogatásával sok, korábban említett korlát megszűnik, és még szélesebb körben használható rendszer lesz a PHP-GTK-ból.

Screenshot

Végül következzen egy screenshot, mely a fent említett, félig kész fejlesztőrendszerről készült, Windows alatt (hogy látsszon a hordozhatóság).

[1] Jelentkezz a <doma@kormendy.hu> e-mail címen!



3.2. Linkek

- `http://gtk.php.net/` – A kiindulópont: dokumentáció, alkalmazások, forráskód, binárisok
- `http://www.gtk.org/` – Gtk+
- `http://www.php.net/` – PHP kezdőoldal
- `http://glade.gnome.org/` – Gtk GUI builder

Központosított felhasználó-menedzsment GNU/Linux környezetben

Kósa Barna
<barna_kosa@freemail.hu>

2002. október 16.

Kivonat

A GNU/Linux rendszerek meghatározó tényezővé váltak a vállalati környezetben. A stabilitás és teljesítmény mellett a GNU/Linux alapú megoldásokat a hatékony és költségkímélő menedzsment és a magas szintű biztonság jellemzi. A menedzsment megoldások közül kiemelt szerepe van a felhasználó-menedzsmentnek. Nyílt forráskódú és kereskedelmi szoftverekre építve olyan központosított felhasználó-menedzsment megoldásokat lehet kialakítani, amelyek kielégítik a legmagasabb igényeket is.

Az alap Linux disztribúciók általában tartalmazzák a központosított felhasználó-menedzsment kialakításához szükséges elemeket: PAM (Pluggable Authentication Module), NIS kliens és szerver, NIS+ kliens, Kerberos kliens és szerver, LDAP szerver.

Ezekből az elemekből lehet összeállítani a konkrét követelményeknek megfelelő megoldásokat.

A dolgozat bemutatja a központosított felhasználó-menedzsment megoldások kialakításának elméleti és gyakorlati oldalát. A különböző komponensek (PAM, NIS, LDAP, Kerberos) részletes ismertetése után egy olyan LDAP és Kerberos alapú megoldás kerül bemutatásra, amelyik gyakorlatilag korlátlanul skálázható, heterogén környezeteket integrál (Linux, különböző Unix változatok, Windows 2000) és megfelel a legszigorúbb biztonsági követelményeknek. A konfiguráció lépéseinek részletes leírása és a konfigurációs fájlok „szakácskönyvként” használhatók a gyakorlati megvalósításban.

Tartalomjegyzék

1. Bevezető

A felhasználó-menedzsment a rendszeradminisztrációs tevékenység egyik legfontosabb területe. Egy modern számítástechnikai környezetben a felhasználók egyidejűleg több alkalmazást használnak (fájlszerverek, levelező rendszer, adatbázis kezelő, Web alapú alkalmazások) és gyakori az alkalmazások közötti adatcsere és kölcsönhatás. A felhasználók és a rendszergazdák életét nagyon megkönnyíti a központosított felhasználó-menedzsment, aminek a lényegét röviden így lehet meghatározni:

- A felhasználókra vonatkozó adatokat (user név, user id, csoportok, szerepkörök, stb.) és az azonosításhoz szükséges információkat (jelszavak, kriptográfiai kulcsok) egy központi adattár tartalmazza. A központi adattárat használó gépek és alkalmazások egy úgynevezett domain-t alkotnak.
- A domain-en belül az azonosítás (authentication) a központi adattárban tárolt információk alapján a helyi alkalmazás szintjén vagy központosítottan történik.
- A központi adattár a felhasználókra vonatkozó adatok mellett számos más, a domain-ben egységesen használt információt tartalmazhat.
- A megfelelő rendelkezésre állás céljából a központi adattárnak több példánya érhető el különböző gépeken (csak olvasható replikák, master és slave szerverek).

A központosított felhasználó-menedzsment nagyon fontos a biztonság szempontjából, mert lehetővé teszi az egész domain-re kiterjedő szabályok kialakítását.

2. A központosított felhasználó-menedzsmentet elősegítő komponensek

GNU/Linux környezetben több olyan rendszerkomponens is található, amely elősegíti a központosított felhasználó-menedzsmentet. Ezek közül a Pluggable Authentication Module (PAM) és a Name Service Switch része minden Linux disztribúciónak.

2.1. Pluggable Authentication Module

Hosszú éveken keresztül a Unix szintű azonosítás a felhasználói név és jelszó segítségével történt a /etc/passwd fájlban tárolt adatok alapján. A hagyományos Unix szolgáltatások (pl. login, telnet, ftp) mellett tetszőleges alkalmazások is használhatják a /etc/passwd fájlban tárolt adatokat megfelelő függvényhívásokon keresztül (pl. getpwent, getpwnam, getpwuid, getpw). A Sun Microsystems által kifejlesztett Pluggable Authentication Module (PAM) de facto szabvánnyá vált a többszintű, rugalmasan változtatható azonosítás megvalósítására. A Linux PAM-implementáció teljesen független a SUN által kidolgozott megoldástól, de ugyanazokra az elvekre épül és működése nagyon hasonló. A Linux PAM nem más, mint egy könyvtárfájlok segítségével kialakított keretrendszer (azonosítási stack). A PAM nagy előnye, hogy úgy tudjuk kialakítani az azonosítási stack-et, hogy nem kell megváltoztatni az alkalmazásokat.

A PAM konfigurálása a /etc/pam.d könyvtárban található fájlok segítségével történik. A Linux PAM az alap UNIX azonosítás mellett (pam_unix) számos más azonosítási mechanizmust támogat (LDAP – pam_ldap, Kerberos 5 – pam_krb5, NT Domain – pam-smb). Minden egyes azonosítást igénylő szolgáltatáshoz a szolgáltatás névének

megfelelő fájl tartozik (pl. login, su, sudo, rlogin). A Linux PAM négy független csoportra bontja az azonosítási feladatot:

- accont – felhasználói azonosítóhoz kapcsolódó tevékenységek,
- auth – a felhasználó hitelességének megállapítása,
- password – jelszavak kezelése,
- session – a felhasználónak nyújtott szolgáltatások.

Az alábbi példa a login szolgáltatást leíró /etc/pam.d/login fájl hitelesítésre (auth) vonatkozó részét mutatja be.

```
auth required /lib/security/pam_securetty.so
auth required /lib/security/pam_nologin.so
auth sufficient /lib/security/pam_stack.so service=system-auth
auth sufficient /lib/security/pam_krb5.so use_first_pass
auth required /lib/security/pam_ldap.so try_first_pass
```

A login funkció esetében az azonosításhoz az alábbi feltételeknek kell teljesülniük:

- A root felhasználóra vonatkozó terminál korlátozás (/etc/securetty): *kötelező*
- A nem root felhasználókra vonatkozó nologin korlátozás (/etc/nologin): *kötelező*
- Unix azonosítás (a system-auth -ban felsorolt pam_unix segítségével): *elégséges*
- Kerberos V azonosítás: *elégséges*
- LDAP azonosítás: *kötelező*

A példából látható, hogy az azonosítás a stack felső elemei felől halad lefelé és attól függően, hogy milyen az adott szinthez tartozó követelmény, halad az alsó szintek felé. Hasonló konfiguráció használható az account vagy password feladatokra.

2.2. Name Service Switch

A UNIX elsődlegesen szövegfájlokban tárolja az adminisztráció szempontjából szükséges információkat (pl. /etc/hosts, /etc/passwd, /etc/group). A különböző adminisztrációs információforrások megjelenése (pl. DNS, NIS, NIS+, LDAP) szükségessé tette a keresési útvonal explicit meghatározását. Erre szolgál a Name Service Switch, aminek a működését a /etc/nsswitch.conf fájl határozza meg. Az alábbiakban egy részlet látható a konfigurációs fájlból:

```
passwd: files nis ldap
shadow: files nis ldap
group: files ldap
hosts: files dns
```

A konfigurációs fájl minden sora tartalmazza az információ nevét és sorrendben a használt forrásokat. Amint a példából látható, ajánlott a keresési útvonalat az egyszerűbb, biztos forrásokkal kezdeni (helyi gépen tárolt fájlok).

2.3. Automount

Amennyiben egy felhasználó különböző gépeken ugyanazt a home könyvtárat szeretné használni, kézenfekvő megoldás a home könyvtárak NFS-en keresztüli használata. Ez a legegyszerűbben az automount démon segítségével valósítható meg. Az automount konfigurálása a /etc/auto.master fájl segítségével történik. Az alábbi példában bemutatott konfiguráció az nfssrv NFS szerverről csatolja a home könyvtárakat a /var/autofs/home könyvtárba:

```
etc/auto.master
/var/autofs/home /etc/auto.home
/etc/auto.home
* -fstype=nfs,soft,intr,rsize=8192,wsize=8192,nosuid nfssrv:/home/&
```

2.4. LDAP

Az LDAP (Lightweight Directory Access Protocol) az X.500 címtár szolgáltatás egyszerűsített változata. A címtár szolgáltatás nem más, mint egy speciális szempontok szerint kialakított adatbázis:

- Az adat objektumok viszonylag kis méretűek
- Az adatbázis elosztott (replikált) és cache-ben tárolt
- Az információ attribútum alapú
- Az olvasási műveletek sokkal gyakoribbak, mint az írás
- A keresés a leggyakoribb művelet.

Más adatbázisokhoz hasonlóan az LDAP is sémákat használ. A sémák olyan szabálycsoportok, amelyek meghatározzák, hogy mi tárolódik a címtár szolgáltatásban. Az LDAP-ot használó alkalmazások külön sémákat használnak. Léteznek szabványos sémák, amelyek gyakorlatilag minden LDAP termékben megtalálhatók. Ilyen például a Unix szintű információkat leíró, az RFC 2307-ben definiált Network Information Service schema. Nagy előnye az LDAP címtáraknak, hogy tetszés szerint lehet a sémákat módosítani vagy új sémákat létrehozni.

Mint minden elosztott és/vagy replikált szolgáltatás, az LDAP címtár szolgáltatás is több szerverből áll. Kliens oldalon LDAP parancsok, LDAP API-t megvalósító könyvtárak vagy Perl modulok biztosítják a kapcsolatot az LDAP szerverekkel.

GNU/Linux környezetben több LDAP-kompatibilis megoldás is létezik: OpenLDAP `http://www.openldap.org/`), Netscape Directory Server, Novell eDirectory. Ezek közül érdemes kiemelni a nyílt forráskódú OpenLDAP-ot, amely része a legtöbb Linux disztribúcióknak. Az OpenLDAP konfigurálásához szükséges információk a függelékben találhatók.

2.5. Kerberos

A Kerberos egy bővíthető hálózati autentikációs protokoll, amit a Massachusets Institute of Technology-n (MIT) fejlesztettek ki a 90-es években. A Kerberos specifikációját az RFC 1510 definiálja. A jelenlegi verzió a Kerberos 5, de az előző Kerberos 4 verziót is használják még. A Kerberos az úgynevezett „trusted third party” megosztott titkos kulcs módszert használja. A felhasználók és alkalmazások (Kerberos néven

principal) azonosítása a Kerberos szerver (Key Distribution Center - KDC) által kibocsátott jegyek (ticket) alapján történik. A jelszavak soha nem kerülnek továbbításra nyílt formában, és a teljes forgalom titkosítható. Titkosításra a Kerberos 5 a 3DES algoritmust használja. A Kerberos szerverek és az általuk kiszolgált objektumok egy Kerberos realm-et alkotnak.
A Kerberos szerver (KDC) két komponensből áll:

- Authentication Server – AS, ez végzi a principal-ok azonosítását
- Ticket Granting Service – TGS, ez végzi a Ticket Granting Ticket-ek (TGT) kibocsátását

Két típusú Kerberos jegy létezik:

- Ticket Granting Ticket (TGT) – a Kerberos principal-ok azonosítására szolgáló token. Az AS és a kliens megosztott titkos kulcsával van titkosítva. A jegyeknek lejárási idejük van (expiration time).
- Session Ticket – kliens/szerver páronként egyedi, a TGS és a szerver megosztott titkos kulcsával van titkosítva. Időpecséttel van ellátva.

A gyakorlatban a Kerberos principal-ok „kerberizált” kliens és szerver programok. A legtöbb Kerberos implementáció tartalmaz „kerberizált” telnet és ftp klienst illetve szervert. Ilyen, Kerberos-hoz illesztett programok írását egy programozói interfész (GSS-API) biztosítja. A központosított felhasználó-menedzsment szempontjából nagyon fontos szerepe van a pam_krb5 PAM modulnak.

A legismertebb Unix változatok mindegyike tartalmazza a Kerberos 5 implementációját. GNU/Linux környezetben két Kerberos implementáció létezik: a MIT által kidolgozott Kerberos 5-nek megfelelő (RedHat: krb5-server, krb5-workstation, krb5-libs és krb5-devel csomagok – Debian: krb5-kdc, krb5-user, libkrb53, krb5-telnetd, krb5-ftpd,krb5-clients) vagy a teljesen független Heimdal `http://www.pdc.kth.se/heimdal/`. A Windows 2000 is a Kerberos 5-öt használja, ami lehetővé teszi a Windows és Unix/Linux vegyes környezetek azonosításának integrációját. A Kerberos realm konfigurálásának részletes leírása a függelékben található.

3. Központosított felhasználó-menedzsment a gyakorlatban

A központosított felhasználó menedzsment kialakításakor a következő szempontokat kell figyelembe venni:

- Lokális és központilag menedzselt felhasználók és csoportok szétválasztása. A lokális felhasználók és csoportok a root és egyéb operációs rendszer által használt usereket illetve csoportokat jelentik (bin, daemon, adm, wheel). Ez a szétválasztás azért szükséges, mert a különböző Linux disztribúciókban vagy Unix verziókban ezekhez a felhasználókhoz és csoportokhoz különböző uid és gid tartozhat.
- A központosított megoldás csak minimális hatással legyen az alkalmazásokra és a felhasználókra (pl. bejelentkezés, jelszó megváltoztatása).
- A felhasználói home könyvtárakat egységesíteni kell.

- Meg kell oldani a gépenkénti bejelentkezések szelektív engedélyezését.
- Biztosítani kell a központosított adattár megfelelő rendelkezésre állását (replikált adatok, master és slave szerverek).
- A központosítást csak megfelelő biztonsági megoldások alkalmazása mellett szabad megvalósítani (pl. A jelszavak csak titkosítva kerülhetnek ki a hálózatra).

A fenti szempontokat figyelembe véve olyan gyakorlati megoldások kerülnek bemutatásra, amelyek a GNU/Linux disztribúciók standard komponenseire épülnek és lehetővé teszik a heterogén Unix és Linux környezetek hatékony és biztonságos integrációját.

3.1. NIS

A Network Information Service-t (NIS) a Sun dolgozta ki a 80-as években. Ez volt az első központosított adminisztrációs adattár Unix környezetben. Eredetileg Sun Yellow Pages-nek hívták, de jogi okokból át kellett keresztelni. Ezért kezdődnek a NIS parancsok az yp betűkkel. A NIS része a legtöbb Linux disztribúciónak.

A NIS egy egyszerű hálózati keresési szolgáltatást valósít meg egy adatbázis és szerver (ypserv, rpc.yppasswd) illetve kliens oldali processzek (ypbind) segítségével. A NIS-ben tárolt információk egy domain-re vonatkoznak. A NIS domain egy lapos (flat) modellnek felel meg. A domain tartalmaz egy master szervert, opcionálisan egy vagy több slave szervert és NIS klienseket. A NIS domain nevét (egyszerű ASCII string) a domainame parancs segítségével lehet beállítani illetve lekérdezni. A domain név megadása Linux disztribúció függő:

- Debian – /etc/sysconfig/network fájl, NISDOMAIN változó,
- RedHat – /etc/defaultdomain fájl.

A NIS adatbázis gdbm (GNU database manager) adatfájlokra épül. A gdbm fájlok kulcs – érték formában tárolják az adatokat és hatékony keresést biztosítanak. A NIS adatbázis feltöltése szöveges fájlokból történik. Alapkonfigurációban ezek a fájlok a /etc könyvtárban találhatók. Minden szöveges fájlnak egy úgynevezett NIS map felel meg (az alapvető map-ek listája a függelékben található). Minden egyes NIS map-hez két gdbm fájl tartozik (pl. passwd.byname és passwd.byuid) A NIS adatok a /var/yp könyvtárban tárolódnak. A NIS map-ek módosítása a /usr/lib/yp/makedbm vagy a make (/var/yp könyvtárban kiadva) parancsokkal történhet. A /var/yp/Makefile módosításával változtathatunk a map-eken vagy kialakíthatunk új map-eket.
A NIS domain kialakítása az alábbi lépésekből áll:

- Master szerver létrehozása – `/usr/lib/yp/ypinit -m` (létrehozza a NIS map-eket),
- Slave szerver létrehozása – `/usr/lib/yp/ypinit -s masterserver`,
- A NIS szerver processzek illetve a NIS kliens oldali processz automatikus indításának konfigurálása.

A NIS map-ek módosítása csak a master szerveren történhet, a slave szerverek csak egy olvasható másolatot tárolnak. A NIS map-ek módosításakor a teljes NIS map-et szét

kell küldeni a slave szerverekre. Erre szolgál az yppush parancs. A slave szerverek szinkronban tartását a cron által ütemezett programokkal (ypxfr_1perhour, ypxfr_1perday, ypxfr_2perday) lehet elérni.

Az ypbind NIS kliens processz broadcast üzenetekkel térképezi fel a NIS szerver processzeket és hozzákapcsolódik (binding) egy szerverhez. Amennyiben a kiválasztott szerver processz nem válaszol, az ypbind egy újabb szervert keres és ahhoz kapcsolódik. A broadcast módszer miatt ez a megoldás csak egy alhálózaton belül működik. Lehetőség van egy adott NIS szerverre kapcsolódni (ypset parancs). Ebben az esetben a kliensek és a szerver különböző alhálózatokon is lehetnek.

Mivel a NIS egy hálózati névfeloldási szolgáltatás, szükség van a helyileg tárolt adatok és a NIS együttes használatára. Ez kezdetben a /etc/passwd és /etc/group fájlok végére tett + (teljes NIS map), +user (egy adott felhasználó) vagy +@netgroup (felhasználó csoportok) segítségével történt. Modernebb megoldás a Name Service Switch használata.

Mivel a NIS adatok a domain összes gépére érvényesek, szükség lehet a gépenkénti bejelentkezés szabályozására. Erre a netgroup NIS map ad lehetőséget. Ilyenkor a /etc/nsswitch.conf fájlban a kompatibilitás módot kell használni a group és passwd névfeloldásra:

```
passwd: compat
group: compat
passwd_compat nis
group_compat nis
```

és a /etc/passwd fájl végén meg kell adni azokat a netgroup map-ben tárolt csoportokat, amelyek számára engedélyezzük (+@group) vagy tiltjuk (-@group) a bejelentkezést.

A NIS előnye, hogy egyszerűen adminisztrálható, jól működik vegyes Unix/Linux környezetben, és kis terhelést jelent a gépek számára. Hátrányai között meg kell említeni a biztonság szinte teljes hiányát: gyakorlatilag bárki rákapcsolódhat a NIS szerverekre, és nagyon könnyű egy hamis NIS szervert elhelyezni.

3.2. NIS+

A NIS+ a NIS továbbfejlesztett változata. Kiküszöböli elődje hibáit és kompatibilitást biztosít a NIS kliensek felé. Lehetővé teszi a fa struktúrájú, hierarchikus domain-ek kialakítását (hasonlóan, mint a DNS), ami jóval nagyobb skálázhatóságot eredményez, mint a NIS. A NIS+ namespace objektumokból áll (csoport és tábla). A szerverekhez való kapcsolódás dinamikus, minden egyes névfeloldási kérés esetén kiválasztásra kerül egy szerver. A NIS-hez hasonlóan a NIS+ domain egy master és egy vagy több replica szerverből, illetve kliensekből áll. Az adatbázis struktúrája is megváltozott. A NIS+ táblák több oszlopot tartalmaznak, és mindegyik oszlop szerint lehet keresni. A NIS+ táblák gyakorlatilag megfelelnek a NIS map-eknek. A NIS+ replica adatbázisok frissítése esetén csak a változások továbbítódnak. A legnagyobb változások a biztonság területén történtek (DES titkosítás, nyilvános és titkos kulcsok, jogosultságok által szabályozott hozzáférés a NIS+ objektumokhoz).

Számos előnye ellenére a NIS+ nem terjedt el, ezért Linux alatt csak NIS+ kliens létezik `http://www.linux-nis.org/nisplus/`. A kliens együttműködik Solaris vagy HP-UX alatti NIS+ szerverekkel.

3.3. LDAP alapú felhasználó-menedzsment és azonosítás

GNU/Linux környezetben több LDAP szerverrel is megvalósítható a központosított felhasználó-menedzsment. Nyilvánvaló okokból a nyílt forráskódú OpenLDAP szerverre épülő megoldás kerül bemutatásra.

A standard OpenLDAP konfigurációnak részét képező Network Information Service (RFC 2307) séma tartalmazza a Unix és Linux által használt összes felhasználó-menedzsmenttel kapcsolatos információt: uidNumber, gidNumber, gecos, homeDirectory, loginShell attribútumok. A sémát az alábbi fájl tartalmazza:

- RedHat: /etc/openldap/schema/nis.schema,
- Debian: /etc/ldap/schema/nis.schema.

Az LDAP címtár lekérdezéséhez először kapcsolódni (bind) kell az LDAP szerverhez. A kapcsolódás felhasználó névvel és jelszóval történik, de beállítható az anonymous binding, amikor nem kell felhasználó nevet megadni. A bind művelet során végrehajtott azonosítás típusa is többfajta lehet. Az úgynevezett egyszerű azonosítás (simple authentication) esetén a jelszavak nyílt formában kerülnek ki a hálózatra. Ezért ezt az azonosítást csak titkosítással együtt (SSL) szabad használni. Sikeres kapcsolódás esetén a felhasználók szabadon kereshetnek az LDAP címtárban. Ezért megfelelő Access Control kialakításával kell biztosítani, hogy tárolt jelszavakhoz csak az adott felhasználó férhessen hozzá.

A fent bemutatott, NIS sémában tárolt adatok és az LDAP szerverhez való kapcsolódást biztosító azonosítás képezi az LDAP alapú felhasználó-menedzsment alapját. Ahhoz, hogy ez a megoldás transzparens legyen az operációs rendszer számára, biztosítani kell az LDAP alapú névfeloldást és az LDAP alapú operációs rendszer szintű azonosítást. Ezeket a funkciókat a PADL Software `http://www.padl.com/Contents/OpenSourceSoftware.html` által kifejlesztett nyílt forráskódú nss_ldap (LDP alapú Name Service Switch) és pam_ldap (LDAP alapú PAM) modulokkal lehet megvalósítani. Ezek a modulok az alábbi csomagokban találhatók: RedHat: nss_ldap és auth_ldap, Debian: libnss-ldap és libpam-ldap. A PADL Software web helyéről letölthető a teljes NIS sémának megfelelő adatok migrációját biztosító Perl-szkript gyűjtemény. A Name Service Switch konfigurálását meghatározó /etc/nsswitch.conf fájlnak az alábbi bejegyzéseket kell tartalmaznia:

```
passwd: files ldap
group: files ldap
```

Az operációs rendszer helyes működése szempontjából kötelező a keresési sorrendben a helyi fájlokat az első helyre tenni. Így a lokális felhasználók azonosítása akkor is megtörténik, ha az LDAP szerver nem elérhető.

Az LDAP alapú azonosítást végző PAM beállítása a 2.1. fejezetben bemutatott konfigurációs fájlban látható.

Az LDAP-ot használó alkalmazások különböző sémákban tárolják az információkat. A jelszavakat szinkronizálni lehet egy LDAP menedzsment szoftver vagy erre a célra megírt programok segítségével.

Az OpenLDAP alapú központosított felhasználó-menedzsment kialakításához szükséges információkat a függelék tartalmazza.

3.4. LDAP alapú felhasználó-menedzsment, Kerberos alapú azonosítás

Az LDAP alapú azonosítás helyett használható a Kerberos alapú azonosítás. Ez az alábbi előnyöket jelenti a központosított felhasználó-menedzsment számára:

- A Kerberos egy szabványos, jóval nagyobb biztonságot nyújtó megoldás,
- Lehetőség van az egyszeri bejelentkezési (Single SignOn) megoldás kialakítására.

A Kerberos alapú azonosítás a Kerberos PAM modul segítségével történik (RedHat: pam_krb5, Debian: libpam-krb5). A 2.1. fejezetben bemutatott /etc/pam.d/login fájlban látható a Kerberos alapú azonosítást megvalósító konfiguráció.

Kerberos alapú azonosítás esetén az LDAP-ban tárolt felhasználókat létre kell hozni a Kerberos-ban is. Ez viszonylag könnyen megvalósítható különböző nyelvekben (C, Perl, Python, PHP) megírt programok segítségével. A pam_krb5 és nss_ldap lehetővé teszi a Linux és Windows Active Directory egyszerű összekapcsolását. Az Active Directory LDAP szerverként működik és Kerberos-t használ az azonosításra. Az SFU (Services for Unix) telepítésével az Active Directory kiegészül a NIS sémának megfelelő attribútumokkal. Active Directory használata esetén az nss_ldap konfigurációs fájlban definiálni kell a NIS sémának történő megfeleltetéseket (lásd a függelékben).

4. Összefoglaló

A legtöbb GNU/Linux disztribúció számos olyan standard komponenst tartalmaz, amelyek segítségével különböző komplexitású központosított felhasználó-menedzsment megoldásokat lehet kialakítani. A konkrét igényeknek megfelelően lehet választani az egyszerűen adminisztrálható, de kevés biztonságot nyújtó NIS, és a legmagasabb igényeket is kielégítő LDAP + Kerberos skáláján.

A GNU/Linux operációs rendszerek fejlettségét mi sem bizonyítja jobban, mint az a tény, hogy ezek a rendszerek egyaránt lehetnek kliensek egy központosított megoldásban, vagy biztosíthatják a megoldás szerver oldali részét.

5. Függelék

5.1. NIS mapek

Részlet a /var/yp/Makefile-ból:

```
YPSRCDIR = /etc
YPPWDDIR = /etc
.........................................
GROUP = $(YPPWDDIR)/group
PASSWD = $(YPPWDDIR)/passwd
SHADOW = $(YPPWDDIR)/shadow
GSHADOW = $(YPPWDDIR)/gshadow
ADJUNCT = $(YPPWDDIR)/passwd.adjunct
ALIASES = /etc/aliases
ETHERS = $(YPSRCDIR)/
BOOTPARAMS = $(YPSRCDIR)/
HOSTS = $(YPSRCDIR)/hosts
NETWORKS = $(YPSRCDIR)/networks
```

```
PRINTCAP = $(YPSRCDIR)/printcap
PROTOCOLS = $(YPSRCDIR)/protocols
PUBLICKEYS = $(YPSRCDIR)/publickey
RPC = $(YPSRCDIR)/rpc
SERVICES = $(YPSRCDIR)/services
NETGROUP = $(YPSRCDIR)/netgroup
NETID = $(YPSRCDIR)/netid
AMD_HOME = $(YPSRCDIR)/amd.home
AUTO_MASTER = $(YPSRCDIR)/auto.master
AUTO_HOME = $(YPSRCDIR)/auto.home
AUTO_LOCAL = $(YPSRCDIR)/auto.local
TIMEZONE = $(YPSRCDIR)/timezone
LOCALE = $(YPSRCDIR)/locale
NETMASKS = $(YPSRCDIR)/netmasks
```

5.2. OpenLDAP konfigurálása (RedHat 7.3)

A /etc/openldap/sldapd.conf LDAP szerver konfigurációs fájl:

```
include /etc/openldap/schema/core.schema
include /etc/openldap/schema/cosine.schema
include /etc/openldap/schema/inetorgperson.schema
include /etc/openldap/schema/nis.schema
include /etc/openldap/schema/redhat/rfc822-MailMember.schema
include /etc/openldap/schema/redhat/autofs.schema
include /etc/openldap/schema/redhat/kerberosobject.schema
TLSCertificateFile /etc/openldap/ssl/server.crt
TLSCertificateKeyFile /etc/openldap/ssl/server.key
TLSCACertificateFile /etc/openldap/ssl/ca.crt
access to attr=userPassword
by self write
by anonymous auth
by * none
access to *
by self write
by * read
database ldbm
suffix "dc=mydom,dc=com"
rootdn "cn=Manager,dc=mydom,dc=com"
rootpw secret
directory /var/lib/ldap
index objectClass,uid,uidNumber,gidNumber,memberUid eq
index cn,mail,surname,givenname eq,subinitial
loglevel 2048
```

A /etc/openldap/ldap.conf LDAP kliens konfigurációs fájl:

```
HOST beta.mydom.com
BASE dc=mydom,dc=com
URI ldap://beta.mydom.com
BINDDN uid=proxyuser,ou=Special Users,dc=mydom,dc=com
BINDPW abcd1234
```

A /etc/ldap.conf nss-ldap konfigurációs fájl:

```
host beta.mydom.com
base dc=mydom,dc=com
ssl start_tls
ssl on
tls_cacertfile /etc/openldap/ssl/server.crt
tls_ciphers TLSv1
```

Active Directory és SFU használata esetén:

```
nss_map_objectclass posixAccount User
nss_map_attribute uid msSFUName
nss_map_attribute uniqueMember posixMember
nss_map_attribute userPassword msSFUPassword
nss_map_attribute homeDirectory msSFUHomeDirectory
nss_map_objectclass posixGroup Group
pam_login_attribute msSFUName
pam_filter objectclass=User
pam_password ad
```

5.3. Kerberos 5 konfigurálása (RedHat 7.3)

A Kerberos működésének és konfigurálásának leírása megtalálható a The Official Red Hat Linux Reference Guide-ban.

Konfigurációs fájlok

/etc/krb5.conf:

```
[logging]
  default = FILE:/var/log/krb5libs.log
  kdc = FILE:/var/log/krb5kdc.log
  admin_server = FILE:/var/log/kadmind.log
[libdefaults]
  ticket_lifetime = 24000
  default_realm = MYDOM.COM
  dns_lookup_realm = false
  dns_lookup_kdc = false
[realms]
  MYDOM.COM= {
    kdc = beta.mydom.com:88
    admin_server = beta.mydom.com:749
    default_domain = mydom.com
  }
[domain_realm]
  .MYDOM.COM= MYDOM.COM
  MYDOM.COM= MYDOM.COM
[kdc]
  profile = /var/kerberos/krb5kdc/kdc.conf
[appdefaults]
  pam = {
    debug = false
    ticket_lifetime = 36000
    renew_lifetime = 36000
    forwardable = true
    krb4_convert = false
  }
```

/var/kerberos/krb5kdc/kdc.conf:

```
[kdcdefaults]
  acl_file = /var/kerberos/krb5kdc/kadm5.acl
  dict_file = /usr/share/dict/words
  admin_keytab = /var/kerberos/krb5kdc/kadm5.keytab
  v4_mode = nopreauth
[realms]
  MYDOM.COM= {
    master_key_type = des-cbc-crc
    supported_enctypes = des-cbc-raw:normal des-cbc-sha1:onlyrealm
    des-cbc-crc:afs3 des-cbc-md4:v4 des-cbc-crc:normal des-cbc-md5:v4
    des-cbc-md4:afs3 des-cbc-crc:v4 des-cbc-raw:afs3 des-cbc-sha1:afs3
```

```
    des3-cbc-raw:onlyrealm des-cbc-crc:norealm des-cbc-md4:norealm
    des-cbc-raw:v4 des-cbc-md5:afs3 des-cbc-md5:normal des-cbc-sha1:v4
    des-cbc-md4:normal des3-cbc-sha1:onlyrealm des3-cbc-raw:normal
    des3-cbc-sha1:norealm des-cbc-raw:onlyrealm des-cbc-crc:onlyrealm
    des-cbc-md5:onlyrealm des-cbc-md5:onlyrealm des-cbc-md4:onlyrealm
    des3-cbc-sha1:normal des3-cbc-raw:norealm des-cbc-sha1:norealm
    des-cbc-sha1:normal des-cbc-raw:norealm des-cbc-md5:norealm
}
```

A konfigurálás lépései

MYDOM.COM Kerberos realm inicializálása:

```
# kdb5_util create -r MYDOM.COM -s
Initializing database ' /var/kerberos/krb5kdc/principal '
for realm 'MYDOM.COM',
master key name 'K/M@MYDOM.COM'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key: xxxxx
Re-enter KDC database master key to verify: xxxxx

# ls -ls /var/kerberos/krb5kdc
total 16
4 -rw-r--r-- 1 root root 25 Jun 17 12:32 kadm5.acl
4 -rw-r--r-- 1 root root 858 Jun 17 12:33 kdc.conf
8 -rw------- 1 root root 8192 Jun 17 12:35 principal
0 -rw------- 1 root root 0 Jun 17 12:35 principal.kadm5
0 -rw------- 1 root root 0 Jun 17 12:35 principal.kadm5.lock
0 -rw------- 1 root root 0 Jun 17 12:35 principal.ok

# kadmin.local
 Authenticating as principal root/admin@MYDOM.COM with password.
kadmin.local: listprincs
 K/M@MYDOM.COM
 kadmin/admin@MYDOM.COM
 kadmin/changepw@MYDOM.COM
 kadmin/history@MYDOM.COM
 krbtgt/MYDOM.COM@MYDOM.COM
kadmin.local: ank root/admin
 WARNING: no policy specified for root/admin@MYDOM.COM;
 defaulting to no policy
 Enter password for principal "root/admin@MYDOM.COM": xxxxx
 Re-enter password for principal "root/admin@MYDOM.COM": xxxx
 Principal "root/admin@MYDOM.COM" created.
kadmin.local: ktadd kadmin/admin
 Entry for principal kadmin/admin with kvno 3, encryption type Triple
 DES cbc mode with HMAC/sha1 added to  keytab WRFILE:/etc/krb5.keytab.
 Entry for principal kadmin/admin with kvno 3, encryption type DES cbc
 mode with CRC-32 added to keytab WRFILE:/etc/krb5.keytab.
kadmin.local: ktadd kadmin/changepw
 Entry for principal kadmin/changepw with kvno 3, encryption type Triple
 DES cbc mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5.keytab.
 Entry for principal kadmin/changepw with kvno 3, encryption type DES
 cbc mode with CRC-32 added to keytab WRFILE:/etc/krb5.keytab.
kadmin.local: listprincs
 K/M@MYDOM.COM
 admin/admin@MYDOM.COM
 kadmin/admin@MYDOM.COM
 kadmin/changepw@MYDOM.COM
 kadmin/history@MYDOM.COM
 krbtgt/MYDOM.COM@MYDOM.COM
```

```
 root/admin@MYDOM.COM
kadmin.local: q
```

Access Control List fájl létrehozása:

```
# cat /var/kerberos/krb5kdc/kadm5.acl
*/admin@MYDOM.COM*
```

Kerberos szolgáltatások elindítása:

```
# /etc/init.d/krb5kdc start
 Starting Kerberos 5 KDC:           [ OK ]
# /etc/init.d/krb524 start
 Starting Kerberos 5-to-4 Server:   [ OK ]
# /etc/init.d/kadmin start
 Starting Kerberos 5 Admin Server:  [ OK ]
```

Kerberos principal-ok létrehozása:

```
# kadmin
Authenticating as principal root/admin@MYDOM.COM with password.
Enter password:
kadmin: ank myself
 WARNING: no policy specified for myself@MYDOM.COM; defaulting to no policy
 Enter password for principal "myself@MYDOM.COM": yyyy
 Re-enter password for principal "myself@MYDOM.COM": yyyy
 Principal "myself@MYDOM.COM" created.
kadmin: ank -randkey host/beta
 WARNING: no policy specified for host/beta@MYDOM.COM; defaulting
 to no policy
 Principal "host/beta@MYDOM.COM" created.
kadmin: ktadd host/beta
 Entry for principal host/beta with kvno 3, encryption type Triple DES cbc
 mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5.keytab.
 Entry for principal host/beta with kvno 3, encryption type DES cbc mode
 with CRC-32 added to keytab WRFILE:/etc/krb5.keytab.
kadmin: ank -randkey ftp/beta
 WARNING: no policy specified for ftp/beta@MYDOM.COM; defaulting
 to no policy
 Principal "ftp/beta@MYDOMAIN.COM" created.
kadmin: ktadd ftp/beta
 Entry for principal ftp/beta with kvno 3, encryption type Triple DES cbc
 mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5.keytab.
 Entry for principal ftp/beta with kvno 3, encryption type DES cbc mode
 with CRC-32 added to keytab WRFILE:/etc/krb5.keytab.
kadmin: q
```

Kerberos működésének ellenőrzése:

```
$ kinit
 Password for myself@MYDOM.COM: xxxx
$ klist
 Ticket cache: FILE:/tmp/krb5cc_500
 Default principal: myself@MYDOM.COM
 Valid starting Expires Service principal
 09/25/02 22:08:05 09/26/02 08:08:00 krbtgt/MYSDOM.COM@MYDOM.COM
 Kerberos 4 ticket cache: /tmp/tkt500
 klist: You have no tickets cached
```

Nagy rendszerek felépítése

Milus János
<janos.milus@t-systems.com>

2002. október 17.

Kivonat

A Linux komoly térhódítást könyvelhetett el az elmúlt tíz évben. Vajon az IT infrastruktúra központi elemeit, a számítóközpontokat, és egyéb „nagy” rendszereket is rá lehet bízni, vagy megmarad népszerű web- és fájlszervernek? Van-e jelentősége az OpenSource-nak a nagygépek világában? Hova pozicionáljuk a Linuxot? Ezekre a kérdésekre keresi a választ ez az előadás.

Tartalomjegyzék

1. A „nagy” rendszerek

Három esetben beszélhetünk nagy rendszerekről:

1. Nagy mennyiségű adattal dolgozik a rendszer
2. Nagy a számításigénye a rendszernek
3. Mission Critical rendszerről van szó

Nagy mennyiségű adattal dolgoznak az adattárházak, a telekom számlázórendszerek, vagy a CRM rendszerek.

Nagy számításigénye van az olyan jellegű feladatoknak, amikor valamilyen „kaotikus” rendszer viselkedését szeretnénk előre megjósolni. Tipikus példája ennek a meteorológia, vagy a különböző fizikai modellezések: az atomrobbanás-szimuláció vagy a virtuális szélcsatorna. Nagy számításigénye van továbbá a kémiai- és gyógyszeripari kutatásoknak, valamint az adatbányászatnak is.

Mission Critical-nak kell tekinteni egy rendszer akkor, ha a leállása, akár rövid időre is, vagy emberéletet veszélyeztet, vagy több mint egymillió dolláros veszteséget okoz. A definícióból következik, hogy a Mission Critical rendszerek esetében a rendelkezésre állás a legfontosabb tényező.

2. A környezet jelentősége

Mint az előző felsorolásból kitűnik, nagy rendszereknél alapvetően fontos a rendelkezésre állás. Ezzel kapcsolatban bevezetünk néhány fogalmat:

Megbízhatóság

Egy hardver elem két meghibásodása között eltelő idő

MTBF

Mean Time Between Failures. Ugyan az, mint a megbízhatóság.

MTTR

Mean Time To Repair. A javításhoz szükséges átlagos idő.

Rendelkezésre állás

Az az időszak, amikor a rendszeren futó szolgáltatás elérhető. Általában egy évre vetítve, viszonyszámként kerül megadásra. Például a 99,9%-os rendelkezésre állás azt jelenti, hogy egy évben a szolgáltatás 8 órát és 45 percet állhat. A hardver gyártók általában az $A = \frac{MTBF}{MTBF+MTTR}$ képlet alapján adják meg a rendszerük rendelkezésre állását. A valódi rendelkezésre állás ennél mindig rosszabb, mert a hardver elem meghibásodásán kívül figyelembe kell venni az operációs rendszer, az adatbázis kezelő, az alkalmazás meghibásodását, valamint az emberi tényezőt.

Az 1. táblázat a TechWise Research Inc. 2000-ben végzett felmérése alapján készült. A felmérésben HP-UX, Sun Solaris, IBM AIX és OpenVMS operációs rendszerek vettek részt, a leállás okát és a leállás hosszát vizsgálták.

Miért ilyen fontos a rendelkezésre állás? A 2. táblázat azt mutatja, hogy a Dataquest 1996-os felmérése szerint mennyibe kerül egy óra leállás különböző területeken.

Leállás oka	OpenVMS	AIX	HP-UX	Sun Solaris	Összesen
Hardver meghibásodás	4,1	9,5	6,0	7,4	33,38%
Operációs rendszer hiba	1,9	2,1	3,2	2,6	12,11%
Adatbázis hiba	1,2	2,2	10,9	6,9	26,21%
Alkalmazás hiba	2,0	3,1	0,7	5,3	13,72%
Emberi hiba	1,8	1,5	2,0	6,5	14,58%
Összesen	11,0	18,4	22,8	28,7	

1. táblázat. Átlagos éves állásidő órában mérve

Iparág	**Működési kör**	**Költség óránként**
Pénzügy	Bróker megbízások	$6,45M
Pénzügy	Credit kártya engedélyezés	$2,6M
Média	Pay-per view TV	$150K
Szolgáltatás	Home shopping (TV)	$113K
Szolgáltatás	Katalógus áruház	$90K
Fuvarozás	Repülőjegy foglalás	$89,5K
Pénzügy	ATM	$14,5K

2. táblázat. Egy kiesett óra költsége különböző területeken

A felmérést természetesen Amerikában végezték, tehát a számok Magyarországra vonatkoztatva túlzóak, de jól mutatják, hogy bizonyos iparágakban egy óra kiesés komoly gazdasági problémát okozhat.

Még egy utolsó adat: Egy Gartner felmérés szerint egy IT beruházás 3 évre vetített teljes bekerülési költségének mindössze 10%-a a a vételi ár.

Milyen következtetéseket vonhatunk le a Linux és a nagy rendszerek kapcsolatából az eddig leírtak alapján?

Mi csak az operációs rendszerrel foglalkozunk…

pedig ez a lehetséges meghibásodások mindössze 12,11%-át adja. A leállások számát és idejét így elsősorban nem még stabilabb operációs rendszerrel, hanem megfelelő fizikai környezettel (gépterem, klíma, szünetmentes tápegység stb.) megbízható elemekből összerakott számítógéppel, HA megoldásokkal, a szakemberek képzésével, tesztelt alkalmazásokkal és adatbázis-kezelőkkel lehet csökkenteni.

Az ingyenesség nem szempont…

mivel a három évre számolt TCO-ban az operációs rendszer ára tulajdonképpen kerekítési hiba. A tradicionális UNIX gyártók pedig jelenleg is beleépítik a UNIX árát a hardver árába, így a felhasználó nem fizet külön licencdíjat például a Solaris vagy a HP-UX használatáért. Sokkal fontosabb érv lehet a Linux mellett például az *üzemeltetés alacsony költsége*.

A Linuxnak meg kell felelni a nagy rendszereknél elvárt követelményeknek…

különben megmarad kedvelt web- és fájlszervernek, és a valódi vállalati piacra nem sikerül betörni. Hogy mik is ezek a követelmények pontosan, és a Linux jelenlegi állapotában mennyire felel meg nekik, arról szól a következő fejezet.

3. Elvárások az operációs rendszertől

Alapvetően a következő témakörökben kell egy operációs rendszernek támogatást nyújtania, ha meg akar felelni a nagy rendszerek által támasztott követelményeknek:

- Nagy háttértár kezelése
- Skálázhatóság
- Folyamatos üzemmenet támogatása
- Erőforrás particionálás
- Biztonság

Ezeknek a szempontoknak a részletes kibontása a következő fejezetekben történik.

3.1. Nagy háttértárak kezelése

Logical Volume Management

A fizikai tárolóeszközök operációs rendszer szintű virtualizációját szolgálja. Segítségével több diszket összefoghatunk egyetlen virtuális diszkké, majd ezt a virtuális diszket tetszőleges módon particionálhatjuk. A legtöbb implementációban ezeket a műveleteket teljesen on-line végezhetjük el. A Linux kernel jelenleg a Sistina által kifejlesztett Linux LVM-et tartalmazza. Az ugyanettől a cégtől elérhető már a Linux LVM2, jelenleg béta tesztes állapotban van, és szintén GPL licenc alá esik. A szintén GPL licencelésű, IBM fejlesztésű Enterprise Volume Management System az LVM-mel ellentétben teljesen moduláris felépítésű, és egy kicsit eltérő filozófiájú. Előnye, hogy LVM szinten kezeli a Linux RAID szinteket, a Linux LVM-en kívül még több más LVM implementációt képes használni (például AIX LVM, OS/2 LVM, S390, GPT), és képes például a hibás szektorok relokációjára. Az EVMS hátránya, hogy jelenleg nem része a kernelnek, külön patchként kell letölteni. Aki pedig kereskedelmi megoldást akar, az használhatja a Veritas Volume Managerét. Linkek:

- `http://www.sistina.com/products_lvm.htm`
- `http://evms.sourceforge.net/`
- `http://www.veritas.com/`

Striping

A diszkek összefűzését teszi lehetővé olyan módon, hogy pl. két diszk esetén a létrejövő virtuális diszk minden páros szektora az egyik, minden páratlan szektora a másik fizikai diszkre esik. Elsősorban teljesítmény hangolásnál van szerepe. A Linux kernelben található szoftver RAID képes stripingra.

PITC / BCV / Snapshot

Point In Time Copy / Business Continuity Volume / Snapshot: három neve ugyan annak a technikának. A lényege, hogy egy parancs kiadásával azonnali és konzisztens másolatot lehet kapni egy diszkterületről. Utána a másolatot fel lehet csatolni, le lehet menteni, de akár egy új adatbázis-kezelő is elindítható rajta, statisztikai lekérdezések

számára. A PITC-et három szinten lehet megvalósítani: a storage szintjén (erre képesek például az EMC Symmetrix vagy a Hitachi HDS storage-ok), ekkor az operációs rendszernek tulajdonképpen nincs szerepe. Kisebb rendszereknél meg lehet valósítani LVM szinten (a Linux LVM illetve a EVMS rendelkezik ezzel a funkcióval), vagy fájlrendszer szinten.

Naplózó fájlrendszer

A naplózó fájlrendszer segítségével bármilyen méretű sérült fájlrendszer helyreállítása konstans idő alatt (kevesebb, mint 5 másodperc) megtörténik. A 2.5-ös sorozatú kernelben a következő naplózó fájlrendszerek találhatóak meg: ReiserFS, XFS, JFS, Ext3. 2002 december 31-én várható a Reiser4, aminek a fejlesztését a DARPA szponzorálta. Néhány beígért újdonság: ACL-ek kezelése, pluginek, audit információk, egyedileg titkosított fájlok támogatása. Linkek:

- `http://www.reiserfs.org/`
- `http://oss.sgi.com/projects/xfs/index.html`
- `http://www-124.ibm.com/developerworks/oss/jfs/`
- `http://www.zip.com.au/ akpm/linux/ext3/`

Klaszter fájlrendszer

A klaszter fájlrendszerek a különböző fürtözési megoldásokban játszanak jelentős szerepet. Lehetővé teszik, hogy ugyan azt a fizikai eszközt egyszerre több számítógép csatolja fel, és arra írási és olvasási műveleteket küldjön. (Képzeljük el, mi történne, ha ezt pl. egy ext2-vel tennénk. . .) Linuxon jelenleg egyetlen klaszter fájlrendszer támogatott, a Global File System (GFS). (Sistina fejlesztés, nem OpenSource.) *GFS fájlrendszer felett támogatott az Oracle 9i RAC (fájlrendszerre, és nem RAW device-ra installálva)!*

- `http://www.sistina.com/products_gfs.htm`

Nagy fájlok kezelése

A 2.4-es sorozatú kernellel és a 2.2.x glibc-vel (gyakorlatilag minden Linux terjesztés már ezt használja) bármelyik támogatott UNIX fájlrendszeren (Ext2, Ext3, XFS, JFS, ReiserFS) lehetséges 2GB-nál nagyobb fájl létrehozása.

Raw I/O kezelés

Raw I/O főleg az adatbázis és a cluster alkalmazásokhoz szükséges. Segítségével az I/O buffert megkerülve lehetséges adatokat írni a diszkre. A 2.5-ös sorozatú kernel fel van készítve a raw I/O-ra. A 2.4-es sorozatú kernelekhez a kiobuf patch szükséges. További információk, és fejlesztések a 2.5-ös kernelen:
`http://sourceforge.net/projects/lse/io`

On-line méretezhetőség

A már említett Linux LVM és EVMS on-line méretezhető. A támogatott UNIX fájlrendszerek (Ext2, Ext3, XFS, JFS, ReiserFS) szintén támogatják az on-line méretezhetőséget. Így Linux alatt nincs akadálya annak, hogy egy szűknek bizonyult fájlrendszert menet közben megnöveljünk.

3.2. Skálázhatóság

Memória skálázhatósága

Intel x86 architektúrán 64GB-ig skálázható a memória. A 64 bites platformokon ez a korlát nem áll fenn.

Processzorok skálázhatósága

A Linux jelenleg 32 processzorig skálázható Intel x86 platformon, és 64 processzorig 64 bites platformokon. Összehasonlításképp: HP-UX 11i: 64 processzor, SUN Solaris 9: 128 processzor, Windows 2000 DataCenter 32 processzor.

Lineáris skálázódás

Ez a fogalom azt takarja, hogy 2 processzornál kétszeres, három processzornál háromszoros stb. teljesítményt adjon le a rendszer. Természetesen ez elméletben sem lehetséges, hiszen a több processzor nagyobb adminisztratív overhead-et jelent, bonyolítja az ütemezést, illetve vannak olyan feladatok, amiken ha egy CPU dolgozik, akkor a többi CPU nem dolgozhat. A Linux teljesítménye ebből a szempontból rendkívül jó: teljes kernel lock (azaz amikor csak 1 CPU dolgozik) mindössze a futásidő 3-4%-ban fordul elő (2.5-ös kernelnél), a Molnár Ingo féle O(1)-es scheduler pedig lehetővé teszi, hogy a processzek és a processzorok számától függetlenül, konstans idő alatt kerüljön a következő futó processz kiválasztásra.

Out-of-the-box skálázhatóság

In-the-box (vagy vertikális) skálázhatóságnak hívjuk, ha egy rendszer dobozon belül bővíthető: több CPU-t, több RAM-ot lehet belerakni. Out-of-the-box (vagy horizontális) skálázhatóságról akkor beszélünk, ha egy rendszer teljesítményét úgy tudjuk növelni, hogy még egy különálló számítógépet bekötünk a rendszerbe. Horizontálisan többféleképpen lehet skálázni egy rendszert:

1. Single image-ként
2. Applikációs szinten
3. Terhelés elosztással

Single image-ről beszélünk, amikor az operációs rendszer elfedi az applikációk elől azt a szintet, hogy több gépen fut. Ilyenkor az egy gépre írt applikációk a bináris módosítása nélkül futtathatók egy több gépes rendszerben úgy, mintha az egyetlen SMP-s gép lenne.
Applikációs szintű horizontális skálázásról beszélünk akkor, amikor valamilyen speciális library és központi framework segítségével az applikációt egyszerre több gépen indítjuk el. Ebben az esetben az applikációk a szokásos, egy gépen belül biztosított kommunikációt nem használhatják, csak a library biztosította kommunikációs csatornák állnak rendelkezésre.
A *terhelés elosztásos* skálázásnál az applikációk nem kommunikálnak egymással, nem is tudnak a többi elindított példányról. Tipikus példája ennek, amikor több webszervert használunk, és egy, a webszerverek elé helyezett load balancer dönti el, hogy a kérést melyik szerver fogja kiszolgálni. Linux alatt támogatott horizontális skálázási megoldások:

- Single image: Mosix `http://www.mosix.org/`
- Applikációs szint: Beowulf `http://www.beowulf.org/`
  vagy Oracle 9i RAC `http://oracle.com/`
- Terhelés elosztás: Linux Virtual Server Project
  `http://www.linuxvirtualserver.org/`

3.3. Folyamatos üzemmenet

Redundáns útvonalak a háttértár felé

Intelligens diszkdobozok esetén a különböző RAID szinteket SPOF nélkül maga a diszkdoboz képes biztosítani. Ekkor az operációs rendszer mint egyszerű diszket látja a különben RAID-be szervezett diszkeket. Ahhoz, hogy a nagy rendelkezésre állást maximálisan biztosítani tudjuk egy számítógépen belül, szükséges, hogy ezt a diszket két független útvonalon keresztül érjük el. SCSI eszközök esetén a 2.4-es sorozatú kernel biztosítja ezt a funkciót.

RAID 0, RAID 1, RAID 5...

és ezek kombinációja. A szoftveres RAID, bár sokkal erőforrás-igényesebb a számítógép szempontjából, mint a hardveres, időnként – főleg költségtakarékossági okokból – elkerülhetetlen. Linux alatt az összes RAID level, és ezek tetszőleges kombinációja támogatott.

Redundáns hálózati útvonalak

Hiába működik az általunk üzemeltetett gép, ha a hálózat felől – akár a hálózati kártya hibájából, akár a hálózati útvonalon valamilyen meghibásodás miatt – nem érhető el a szolgáltatás. Az ethernet kártyák olyan jellegű összefogását, ami egyrészt a sávszélesség növekedését, másrészt a redundanciát biztosítja, a Cisco „Etherchannel”-nek, a Sun „Trunking”-nek hívja, Linux alatt – a 2.4-es sorozatú kernelben – pedig „Bonding driver” néven találjuk meg. Fontos megjegyezni, hogy csak abban az esetben tudjuk használni, ha az az eszköz, akivel közvetlen kapcsolatban áll a számítógép (pl. a switch) szintén képes ezt a protokollt használni.

I/O kártyák menet közbeni ki- és bezónázása

Amennyiben egy I/O kártya elromlik, és a szolgáltatás nem áll le (mert jól tervezett rendszernél minden I/O kártya duplikált) az I/O kártya csere már tervezett leállás keretében történhet. A 7×24-es rendszereknél viszont a tervezett leállás is problémás, és adott esetben veszélyezteti az SLA betartását. Ezért, amennyiben a hardver támogatja az on-line kártyacserét (ezt a gyártók általában hot plug-nak hívják), jó, ha az operációs rendszer képes menet közben I/O kártyát kizónázni a rendszerből illetve I/O kártyát hozzáadni a rendszerhez. Linux alatt ezt a funkciót nem tartalmazza a fő kernel stream egyelőre, de az Atlas projekt keretében (támogatók: Bull, HP, IBM, Intel, NEC, SGI, TurboLinux) elindult ennek a funkciónak a fejlesztése. Jelenleg beta állapotban hozzáférhető, produktív rendszerekhez ezért egyelőre nem ajánlott. Bővebb információ:
`http://sourceforge.net/projects/linux-hotplug`

Memória menet közbeni ki- és bezónázása

Az ECC védett memórák 1 bit hibát javítani tudnak. Az előző pontban leírt gondolatmenet alapján eljuthatunk arra a következtetésre, hogy a memória modulokat is jó lenne menet közben cserélni. Az előbb említett Atlas projekt ezt a fejlesztést is magára vállalta, a megoldásuk egyelőre alpha állapotban van, és kizárólag IA64 architektúrán működik. Bővebb információ:

`http://sourceforge.net/projects/lhms`

Processzor menet közbeni ki- és bezónázása

A processzorok általában nem egyik pillanatról a másikra mennek tönkre, hanem először úgynevezett javítható hibákat generálnak. Ha ezek száma túl magas, az jelzi, hogy az adott processzort a közeli jövőben ki kell cserélni. Amennyiben a hardver képes detektálni ezt, úgy egy több processzoros rendszeren elvárt, hogy a CPU cserét menet közben, a szolgáltatások leállítása nélkül el lehessen végezni. Az előző két ponthoz hasonlóan ezzel is az Atlas projekt foglalkozik, a driver alpha állapotú. Bővebb információ:

`http://sourceforge.net/projects/lhcs`

Kommunikáció a szervíz processzorral

A nagyszámítógépek esetén bevett szokás, hogy a produktív CPU-k mellett található egy különleges processzor, ami a hardver folyamatos tesztelését, illetve a hibák diagnosztizálását végzi. Bizonyos operációs rendszerek – például HP-UX – közvetlenül képes a szervíz processzorral kommunikálni, így a szerviz processzor által biztosított információk elérhetők a logokban vagy a userspace programok számára. Linux alatt ez a funkció az ACPI 2.0 driverként lesz elérhető. Ugyan az Atlas projekt elindított egy ehhez kapcsolódó fejlesztést, kézzelfogható eredmény egyelőre nincs. Bővebb információ:

`http://sourceforge.net/projects/acpi-largesys`

HA Cluster

Amennyiben olyan eszköz hibásodik meg, amit nem tudtunk gépen belül duplikálni (például backplane), akkor a szolgáltatás kiesés minimalizálása érdekében az egyetlen dolog, amit tehetünk, hogy a szolgáltatást egy másik gépről folytatjuk. Ezt automatizálják a különböző HA Cluster megoldások. Ezeket a megoldásokat alapvetően két csoportba sorolhatjuk:

1. Operációs rendszer szintű megoldások
2. Alkalmazás szintű megoldások

Az *operációs rendszer szintű megoldások* előnye, hogy segítségükkel bármilyen alkalmazás fürtözhető. Hátrányuk, hogy maga a cluster szoftver ugyan néhány perc alatt detektálja a hibát, és elkezdi a szolgáltatás másik szerverre történő átállítását, de a hibásan leállt applikáció újbóli konzisztens állapotra való hozása hosszabb ideig, adott esetben több óráig is eltarthat. Tipikus példa erre a komoly terhelés alatt álló adatbázisok újbóli elindítása: a logok görgetése, amíg az adatbázis újból konzisztens állapotba nem kerül, sokáig tart.

Az *alkalmazás szintű* fürtözési megoldások előnye, hogy a szolgáltatások gyakorlatilag kiesett idő nélkül vészelik át az egyik node kiesését. Az alkalmazás szintű fürtözés másik nagy előnye, hogy általában terhelés megosztást biztosít a különböző node-ok között. Az alkalmazás szintű cluster hátránya, hogy speciálisan megírt alkalmazást igényel, és általában jóval drágább mint az operációs rendszer szintű megoldás. A Linuxon használt HA fürtözési megoldások:

- Operációs rendszer szinten: A Mission Critical Linux Kimberlite terméke (GPL licencelésű), A Linux HA project heartbeat alkalmazása (GPL licencelésű), az SGI FailSafe terméke (OpenSource licenc), a HP MC/Serviceguard terméke (kereskedelmi licenc). A legfontosabb linkek:

  ```
  http://oss.missioncriticallinux.com/projects/kimberlite/
  http://linux-ha.org/
  http://oss.sgi.com/projects/failsafe/
  http://www.hp.com/
            products1/unix/highavailability/ar/mcserviceguard/
  http://opencf.org/
  ```

- Alkalmazás szintű megoldás: a már sokat emlegetett Oracle 9i RAC. URL:

  ```
  http://oracle.com/
  ```

3.4. Erőforrás particionálás

PRM / WLM

Process Resource Manager / Workload Manager. Az egy gépen (egy image-en) futó alkalmazás-csoportokhoz CPU és I/O sávszélességet, illetve memória mennyiséget rendelhetünk. Ezzel elkerülhetjük, hogy az egyik szervezet alkalmazásai elhasználják az erőforrásokat a másik szervezet, ugyan azon a gépen futó alkalmazásai elől – ez konszolidációs projekteknél rendkívül fontos. Linuxra OpenSource PRM sajnos nincs, a HP viszont árul PRM szoftver linuxhoz. URL:

```
http://resourcemanagement.unixsolutions.hp.com/
                                        WaRM/prm_linux/index.html
```

Particionálás

Particionálásnak hívjuk, amikor a fizikailag egy gépben található erőforrásainkat több, párhuzamosan futó image (operációs rendszer) között osztjuk meg. A particionálásnak három formáját különböztetjük meg:

1. Hardver domain particionálás
2. Szoftver domain particionálás
3. Virtuális particionálás

Hardver particionálás esetén nem szükséges operációs rendszer szintű támogatás, ilyenkor a gép az operációs rendszer számára úgy látszik, mintha több számítógép lenne.
Szoftver particionálásnak hívjuk azt a képességet, hogy több operációs rendszert futtatunk párhuzamosan egy multiprocesszoros, osztott memóriás gép különböző részein hardveres támogatás nélkül.

Virtuális particionálásról olyankor beszélünk, amikor egy core VM egy virtuális hardvert biztosít, amin egy újabb operációs rendszert – akár saját magát – el lehet indítani. A virtuális hardver különböző paramétereinek (CPU szám, memória nagysága, I/O kártyák) nem szükséges megegyeznie a valódi hardver paramétereivel, sőt, az igazi virtuális particionálásnál előfordulhat, hogy a core VM egy uniprocesszoros rendszeren fut, de a hostolt operációs rendszer már azt látja, hogy duál processzoros SMP rendszeren fut. Mindez Linuxon:

- Szoftver domain particionálás: A már említett Atlas projekt indított egy fejlesztést, de egyelőre nincs kézzelfogható eredménye. Bővebb információ:

  `http://sourceforge.net/projects/sdpart`

- Virtuális particionálás: A közismert *vmware* virtuális gépet készít (kereskedelmi licencelésű). A *User-mode Linux* lehetővé teszi, hogy a Linux kernelt – így egy teljes Linux operációs rendszert – a vmware-hez hasonlóan, önmaga felett indítsunk el. A virtuális gép által használt erőforrásokat szabadon paraméterezhetjük. A User-mode Linux GPL licenc alá esik, és része a 2.5-ös sorozatú kerneleknek. További, specifikus megoldás az IBM zSeries gépek z/VM-je, amely képes a Linux kernel futtatására. Segítségével akár 3000 Linuxot is elindíthatunk egy zSeries-en. Kapcsolódó linkek:

  ```
  http://www.vmware.com/
  http://usermodelinux.org/
  http://www-.ibm.com/servers/eserver/zseries/os/linux
  ```

3.5. Biztonság

ITSEC C2

Az ITSEC C2 vagy az ezzel ekvivalens TCSEC C2 szint, – bár kissé idejétmúlt – elvárt egy nagy számítógépes rendszerben. A Linux bár nincs minősítve, speciális beállításokkal funkcionálisan teljesíti ezek követelményeit.

Naprakész biztonsági frissítések

A programok hibákat tartalmaznak, ez nem jó, de együtt kell vele élni. Fontos szempont azonban, hogy egy felfedezett biztonsági hiba napvilágra kerül-e, és hogy milyen gyorsan jelenik meg rá a javítás. A Linux ezen a területen élen jár.

Kötelező hozzáférés védelmi modellek

A hagyományos UNIX-os diszkrét hozzáférésvédelmi modellel szemben a kötelező hozzáférés védelmi modellek esetében nem egy egész sor suid-es programban, hanem csupán a kernelben implementált biztonsági modulban / frameworkben kell megbízni. Amennyiben helyesen lett telepítve egy kötelező hozzáférés védelem, úgy programhiba esetén az adott szolgáltatást korruptálni lehet ugyan, de a rendszeren futó többi szolgáltatás nem sérül, a bizalmas adatok biztonságban maradnak. Linuxon használható, kötelező hozzáférés védelmi modelleket megvalósító projektek:

- Medusa DS9: a felhasználói programok számára transzparens módon biztosítja a biztonságot. Működési elve: a user-space programok által kezdeményezett műveletekről egy daemon eldönti, hogy engedélyezi-e vagy sem. A daemon példa

implementációját egy C szerű konfigurációs fájl segítségével lehet programozni. URL:

`http://medusa.fornax.sk/`

- RSBAC: Rule Set Based Access Control. Egy framework, amibe tetszőleges security modul illeszthető. A Medusa-hoz képest több előnye is van:

  1. Egyetemi projekt, nem ad-hoc módon, hanem átgondoltan, esetenként matematikai alapokra helyezve folyik a fejlesztése.
  2. Több különböző modellt is implementáltak benne.
  3. Az egyik implementált modell a Bell-LaPadula, amit ugyan Amon Ott (az RSBAC fő fejlesztője) nem tart unixok esetén használhatónak, de jelenleg a legfontosabb biztonsági szabályzás, mivel a Common Criteria „Labeled Security Protection Profile" profájlja erre épül – ez funkcionálisan az ITSEC/TCSEC B1 szintnek felel meg.

  `http://www.rsbac.de/`

- Linux Security Module (LSM). Egy másik security framework, az Immunix gondozásában. Legfontosabb előnye az RSBAC-kal szemben, hogy a 2.5-ös kernel sorozat tartalmazni fogja (a merge jelenleg folyamatban van).

  `http://lsm.immunix.org`

- Security-Enhanced Linux (SELinux): az NSA fejlesztése. Egy régebben, kutatási célból definiált (matematikai) modell alapján készült el. A SELinux modelljében megvalósítható a Bell-LaPadula modell. Bár önálló patch-ként indult, jelenleg az LSM része, így a 2.5-ös kernelbe is be fog kerülni.

  `http://www.nsa.gov/selinux`

4. Hogyan pozicionáljuk tehát a nagy rendszereknél a Linuxot?

Ha valaki végigolvasta a megfogalmazott elvárásokat és a rájuk adott Linuxos megoldást, akkor rendkívül bizakodó lehet: gyakorlatilag az összes követelményt vagy lefedi, vagy a közeljövőben le fogja fedni a Linux. Személy szerint azért néhány dologra felhívnám azok figyelmét, akik ezen a tanulmányon felbuzdulva Linux alapon tervezik a következő rendszerük bevezetését:

- A funkciók egy része a 2.5-ös sorozatban található meg, vagy külön patch-ként kell letölteni. A 2.5-ös sorozat nem stabil. Ismétlem: a 2.5-ös sorozat nem stabil.
- Ha különböző patcheket rakunk a kernelre, akkor nem biztos, hogy az eredmény stabil kernel lesz.
- Ha több különböző patchet rakunk a kernelre, azok interferálhatnak (annak ellenére, hogy külön-külön működtek).

Személyes véleményem szerint a Linux jelenleg korlátozottan használható nagy rendszerek építésekor. Ezeket a korlátokat ismerni kell, mielőtt egy nagy rendszerre Linuxot

tervezünk operációs rendszerként. Amennyiben együtt tudunk élni velük, úgy fenntartás nélkül ajánlhatjuk bármelyik UNIX (vagy Microsoft termék) kiváltására. Az viszont tisztán látszik, hogy elindult egy folyamat a nagy rendszerek piaci szegmensének meghódítására, méghozzá a legnagyobb tradicionális UNIX gyártók támogatásával. A célokat megfogalmazták, a kódok elérhetőek (ugyan még csak Beta állapotban), a fejlesztés folyik. Egy, legkésőbb másfél éven belül ezek a munkák részét fogják képezni a hivatalos, stabil kernelnek, amely így mindazon tulajdonságokkal rendelkezni fog, amivel jelenleg csak a high-end-en bevezetett kereskedelmi UNIX-ok rendelkeznek. A Linux hátránya napról napra érezhetően csökken ezeken a területeken, és mára már egyértelműen kinőtte a PC-s származásából adódó minden gyermekbetegségét. Amire valóban büszkék lehetünk, hogy vannak olyan területek, ahol nem a Linux megy a kereskedelmi UNIX-ok után, hanem azok követik a Linuxot: példaként említhetjük a kötelező hozzáférés védelmi modelleket, vagy a User-mode Linuxot, aminek megfelelő funkcionalitást a Sun csak a Solaris 10-ben kíván majd bevezetni.

Magyar Ispell

Válasz a Helyes-e?-re

Németh László <nemethl@gyorsposta.hu>
Szofi Oktatóközpont, Szeged

2002. október 17.

Kivonat

A Magyar Ispell nyílt forráskódú projekt eredményével a Linux hazai – asztali felületeken történő – elterjedésének egyik legnagyobb gátját, a magyar helyesírás-ellenőrző hiányát sikerült orvosolni. A „műtét” kimenete olyan szerencsés volt, hogy ma a Magyar Ispell jobbnak tekinthető, mint a Microsoft Office-szal piacvezetővé vált Helyes-e? (MorphoLogic).

Hogyan készül egy professzionális nyelvi szoftver? Mi a siker titka? Erről szól a következő írás.

Tartalomjegyzék

1. A Magyar Ispell fejlesztésének története

A Magyar Ispell története elsősorban a (leendő) fejlesztők számára lehet tanulságos, így ha nem érzünk magunkban ilyen jellegű vonásokat, javasolt a következő szakaszra (Válasz a Helyes-e?-re) ugrani.

A leírás nem mélyed el a Magyar Ispell felépítésének részleteiben, ezért további információkért a Magyar Ispell dokumentációjához forduljunk.

Az első szakasz

Több, mint egy éve, a III. GNU/Linux konferencián a Unix (még csak nem is a GNU) filozófia szerepéről beszéltem, a Unix parancsnyelv és segédprogramok képességeit szemléltetve. A konferencia után levélben megkeresett Szántó Tamás (a legaktívabb KDE honosító), és segítségét felajánlva arra kért, hogy folytassak egy korábbi projektet.

Annak a bizonyos, 1998-ban indult projektnek a célja egy magyar modul elkészítése volt a GNU rendszer helyesírás-ellenőrzőjéhez, az Ispellhez.

A motiváció egyszerű volt: az általam az idő tájt sűrűn használt TEX szedőrendszerhez már létezett kiváló magyar elválasztás Miklós Dezső és Mayer Gyula (vagy alternatív megoldásként Verhás Péter) jóvoltából, de az Ispell helyesírás-ellenőrző honosítása még váratott magára.

A valódi motiváció persze más volt: a feladat egyszerre volt izgalmas szellemi kihívás, meg nagy haszonnal járó nemes cselekedet. A GNU kiáltvány elolvasása volt az a meghatározó élmény, ami megszüntetett minden korábbi, programfejlesztéssel (és -eladással) kapcsolatos görcsöt bennem. Miután megismertem a GNU/Linuxot, és a lehetőségeit, szerettem volna a fejlesztésből valahogy kivenni a részem. (Bosszantó volt az is, hogy akkor már évek óta létezett több magyar helyesírás-ellenőrző is, de ezek zárt szoftverek voltak. Elhangzott kósza ígéret is (egy érdeklődő kérdésére válaszolva) a MorphoLogic részéről, hogy Linuxra is megjelentetik programjukat otthoni használatra ingyenesen, de ez ügyben nem történt előrelépés.[1]

1998–1999 táján úgy véltem, hogy pár hónap alatt egyedül is végzek az Ispell magyar modul első használható változatának elkészítésével, ezért nem vittem szélesebb nyilvánosság elé a munkát (a honlapomra került fel egy kisebb leírás). 1999 tavaszán szakkönyvtámogatás céljából 30.000 Ft-ot kértem a Nemzeti Kulturális Örökség Minisztériuma és a Matáv által közösen meghirdetett Magyarországi digitális kultúráért pályázaton, igazából talán a szakmai zsűri véleményére kíváncsian, de a pályázatomat elutasították, és „felülbírálatára” (így, hosszú ű-vel) sem volt lehetőség. (El kell mondani a teljesség kedvéért persze, hogy a pályázat alapján – ami elsősorban a TEX-et és Linuxot jelölte meg célplatformként – én sem támogattam volna a helyükben magamat.) Az elutasítás ösztönzőleg hatott, így azt egy intenzív fejlesztési periódus követte 1999 nyarán, de miután megszületett Évi lányom, a fejlesztést bizonytalan ideig szüneteltettem. Úgy tűnt, hogy végleg, de akkor (két év múlva) jött az a bizonyos levél.

Egészében véve – sok kisebb-nagyobb gyakorlati problémát tekintve – egyáltalán nem volt valószínű, hogy Ispell magyar modulból több lesz egy érdekes próbálkozásnál a későbbiekben.

[1] Ez csak mostanában történt meg, de teljes titokban.

Ragok, jelek, képzők

Kezdetben jól haladt a magyar Ispell modul készítése. Az Ispell kézikönyvoldalak alapján hamarosan írtam egy egy-két szabályt tartalmazó ragozásitáblázat-állományt (affix) és szótárállományt (dict), amiből előállítottam az Ispell által ténylegesen használt szótármodul (hash) első változatát.

Később megvásároltam egy leértékelt, középiskolás szintű nyelvtankönyvet, és még némi könyvtárazással a szükséges mértékig elmélyedtem a magyar nyelvtanban.

Hamar túltettem magam azon a problémán, hogy az Ispell csak ragokat kezel, képzőkről, jelekről, tehát többszörös toldalékolási lehetőségekről szó sincs az esetében. Mivel ez így több ezer toldalék előállítását igényelte a későbbiekben, valamilyen segédeszköz után kellett néznem. Ez az eszköz az *m4* makrófeldolgozó lett. Először az alanyi igeragozást valósítottam meg az m4 segítségével. Itt az m4 szerepe csak annyi volt, hogy az Ispell ragozási tábla szabályaiban szereplő feltételt makrókkal adtam meg (egy feltétel nem más. mint a toldalékolási szabály alkalmazását meghatározó szó eleji, vagy szó végi karaktersorozat, vagy egy karaktersorozat-minta). A makrózás után már nem kellett félnem attól, hogy az azonos feltételek egy idő után szép lassan divergálódnak a szélrózsa minden irányába: az állomány elején elég volt átírnom egy makródefiníciót ahhoz, hogy több tucat azonos feltételt tartalmazó szabályt módosítsak.

Ezután következett a névszóragozás. Itt az m4 a fejlesztés fejlődésének köszönhetően már a három hangrend (egy mély, és két magas) egységes kezelésének az eszközévé is vált, továbbá a többszörös toldalékolást is – legalábbis a makródefiníciók szintjén – sikerült vele leírni. Ez jótékony hatással volt a névszóragok kezelhetőségére, de helyenként igencsak idegesítővé vált az m4 paraméterek levédése, különösen az m4 feltételes vizsgálatokban.

A fejlesztés végére maradt a tárgyas igeragozás. Itt az m4 szerepe kiteljesedett, a forrásállomány nemcsak a saját dokumentációját tartalmazta kvázi literális makróprogramozásként, hanem a saját tesztrendszerét is. A forrás a lehető legnagyobb pontosságra törekedett a kivételek kezelésére vonatkozóan, ehhez részletes irodalmat is sikerült beszerezni a Magyar Elektronikus Könyvtárból. Nem meglepő, hogy ez a fajta tárgyasigeragozás-leírás sosem készült el.

A képzők esetében sajnos csak részleges megoldásra futotta az Ispell lehetőségeiből. Hiányzott például a -ság, -ség névszóképző, az igékhez kapcsolódó képzők nagy része (ható-, műveltető-, gyakorítóképző, stb.) Ennek oka az volt, hogy a képzők használata esetén sokszorosára növekedett a ragozási szabályok száma, és vele együtt a szótármodul mérete.

A fejlesztés első szakaszának két igen lényeges megoldásáról is szót kell ejteni még, ami nagyban hozzájárult a megvalósuláshoz és a fejlesztéshez.

Főnevek tőváltozatai

Az első, és egyben legsúlyosabb probléma a főnevek tőváltozatainak szabályozása volt. Nyitótő (kéz/kezet), hangkivetős (bokor/bokrot), vagy a zárt toldalékváltakozásos csoportok (ház/házak), stb. tartoznak ide. Az ilyen típusú tövek egy részére az jellemző, hogy nem szótári tő, vagyis önállóan nem létezik (például *töv* (tövek), *terh* (terhek).

A probléma komolyságát jelzi, hogy a Helyes-e?-ben a mai napig nem sikerült tökéletesen lekezelni a nyitótövek problémáját (l. később kézéé, lovnyi).

A Magyar Ispell esetében sikerült meglepően jó megoldást találni a problémára, a nem szótári tő helyett egy létező (a többes számú) alakból történő levezetéssel (a részleteket a Magyar Ispell dokumentációban találjuk).

Szókincs

A legtöbb munkát a fejlesztés első szakaszában a szókincs feldolgozása jelentette. A ragozott szöveg feldolgozásához többnyire a parancssort, illetve héjprogramokat használtam segédeszközként.[2]

A szókincs kezelésének legfontosabb vívmánya, hogy a unixos hagyományokhoz híven egyszerű szöveges állományban kerültek tárolásra a tőszavak a Magyar Ispell forrásában. Az állományok (neve) egyben a a szófaji kategorizálás is (pl. *fonev*, *ige_alanyi*, *ige_targy*, *melleknev*, *tulajdonnev*, *ragozatlan*).

A szókincsállományok egymezős szöveges adattáblák, magyarul soronként egy szót tartalmaznak. A szavak további (logikai típusú) tulajdonságait külön táblák tárolják. Például a fonev állományhoz tartozik még egy fonev_mely allomany, ami azokat a szavakat sorolja fel ismét a fonev állományból, amelyek magas hangrendű szótaggal végződnek, mégis mély hangrendű a ragozásuk (pl. kávénak). A fonev_osszetett azokat a szavakat ismétli a fonev-ből, amelyek összetett szavak, és így tovább.

A redundancia nyilvánvaló, a fonev állományhoz tartozó többi fonev_ állomány tartalmát egyszerűen összevonhatnánk a fonev tartalmával egy egyszerű szöveges adattáblába (flat database, rekordok a sorok, a mezők a soron belül szóközzel, tabulátorral, vagy mással vannak elválasztva, mint például a /etc/passwd esetében). Vegyük számításba viszont azt is, hogy ezek a kiegészítő állományok általában kis méretűek, és a feltételezett összevonás után szükséges lenne mindig HAMIS, vagy NULL értékre is. Ezt tehát ezzel a szerkezettel megspóroltuk.

A legfőbb előnye azonban ennek a felépítésnek, hogy kézzel és Unix segédprogramokkal könnyen kezelhető. Mind a szótárbővítés, mind az ellenőrzés és javítás szempontjából igen szerencsés választásnak bizonyult ez a talán kevéssé becsült adatszerkezet.

A magyar Ispell szótármodul előállításához a Magyar Ispell forrását „fordítani” kell. Mindez unixos segédprogramok segítségével történik. Többnyire awk, sed, cut programok csőbe kötve, amelyek előállítják a szótárállományt (dict) fordítási időben. Ebből, és az m4-gyel létrehozott ragozásitáblázat-állományból az Ispell segítségével állítható elő a szótármodul-állomány.

A fejlesztés második szakasza

Tamás segítségével gyakorlatilag a Magyar Ispell a közösségi fejlesztés fázisába lépett. Részletes hibalistákat kaptam tőle, amelyekben rámutatott a tipikus nyelvtani problémákra, tévesztésekre is, így minden szabadidőmet a javításra és fejlesztésre fordíthattam (ez éjféltől általában hajnali 3-ig, esetleg tovább tartott).

2001 őszének legfontosabb felismerése az volt, hogy a Magyar Ispell esetében a képzőkkel kapcsolatos problémát csak a képzett szóalakok redundáns felvételével lehet

[2] Az *ms* fantázianevű héjprogram például magyar ábécérendbe rendezte egy szóállomány sorait, az ismétlődő szavak felesleges példányainak törlésével egyetemben. A magyar ábécérendbe történő rendezést ebben az időben az angol ábécérendbe soroló sort végezte az *msort* csomagolóprogrammal: az msort a sort bemenő karaktereit, illetve többjegyű mássalhangzóit a felső ASCII tartományba helyezte, minden így kapott karaktert három q betűvel bevezetve. A q betűk nagysága (Q/q) hordozta az arra vonatkozó információkat, hogy a kódolt betű nagybetű-e (többjegyűeknél melyik nagybetű: Gy/GY), ékezetes-e, illetve hogy ismétlődő többjegyű mássalhangzók esetén egybe voltak-e írva azok, vagy sem (pl. ggy/gygy), Ennek a kódolásnak az érdekessége, hogy így a magyar ábécérendbe sorolásra jellemző ékezetes betűk között fennálló ekvivalenciát a sort által ismert kis- és nagybetűpárok között fennálló ekvivalenciával lehetett ideiglenesen helyettesíteni, és a sort kimenetét az msorttal visszaalakítva helyes magyar ábécérendbe-sorolást kaptunk. (Két speciális esettől eltekintve: szokatlan betűméretek esetén többjegyű mássalhangzóknál (gY), valamint ha három szó csak az ékezet meglétében és a betűk nagyságában különbözik ugyanazon karakterpozícióban.)

tárhatékonyan lekezelni. Ez a redundancia ugyanis lényegesen kisebb volt, mint az redundancia, amit a több ezer új ragozási szabály előállítása jelentett volna.

Ezek a képzett alakok a szótármodul fordítási idejében állnak elő. Ilyenek a melléknevek -ság, -ség főnévképzős, és az igék ható- és műveltetőképzős alakjai, valamint a melléknévi igenevek.

Egy kétsoros héjprogram segítségével modularizált lett a szókincs is, így a szakszavak (pl. matematikai szókincs) végre elkülönülhetett az alapszókincstől.

2001 karácsonyára a Magyar Ispell megérett a nyilvánosságra hozatalra, de erre csak január első felében került sor. Ennek, és később az OpenOffice.org integrációnak köszönhetően sokan segítettek hibalisták készítésével és egyéb javaslatokkal is.

Tamás visszatért a KDE-hez, helyét Bíró Árpád vette át (már korábban csatlakozva hozzánk). Árpád sok olyan rejtett hibát vett észre, amelyek megoldása később a Magyar Ispell minőségi fejlődését eredményezték (például az összetett szavak kezelése terén), valamint több szókincsmodul (mint a matematikai) készítéséért felelős.

2001 végén talált ránk Godó Ferenc, aki magánszorgalomból gyűjtött, hatalmas, rendezett szólistájának átadásával jelentős mértékben növelte a Magyar Ispell szókincsét, és ezzel együtt használhatóságát is.

Nyelvészünk, Trón Viktor 2002 márciusában csatlakozott hozzánk, rögtön kijavítva egy írott és beszélt nyelvi hibámat (nem ikes ragozásról volt szó, azon inkább az ő javaslatára enyhítettem, így a szótár azóta fogadja el a „dolgozok”-ot is a dolgozom mellett, hasonlóan az élőnyelvben megszokotthoz). Viktor később összegyűjtötte, és feldolgozta a mai magyar és történelmi helységneveket is, és a Magyar Ispell „fordítóprogramon” is javított.

A fejlesztés közben folyamatosan kerültek ki az újabb és újabb Magyar Ispell változatok a http://www.szofi.hu/gnu/magyarispell oldalra. Ezzel párhuzamosan SuSE RPM és Debian csomagokat Koblinger Egmont – aki egyben Magyar Ispell fejlesztő is – és Pásztor György készítettek.

MySpell

Godó Ferenc hívta fel a figyelmemet 2001 végén az OpenOffice.org-ban MySpell fedőnév alatt megjelent egyszerű helyesírás-ellenőrző kódra. Kevin Hendricks programja az Ispell affixtömörítő algoritmusán alapszik, az eredeti C-s program lényegi részének C++-os megvalósítása. Egyszerűsége miatt kiváló alapot jelentett egy jobb magyar helyesírás-ellenőrző elkészítéséhez. David Einstein Mozilla fejlesztő az algoritmus hatékonyságát növelte az Ispell szintjére. Én elsősorban a hiányzó összetettszó-ellenőrzést valósítottam meg, aminek első egyszerű formája be is került a MySpell kódba. A hibás szavakra történő javaslattevés minőségét sikerült nagy mértékben előmozdítanom egy egyszerű, de nagyszerű ötlet segítségével: egy cseretáblázattal a többkarakteres eltéréseket is felismeri a MySpell, és javaslatot tesz. Gyakorlatilag a j→ly javaslattevés általánosítása volt ez. Az ötlet része, hogy a cseretáblázat a ragozási táblázatot tartalmazó állományban foglal helyet, vagyis adott nyelvre jellemző mintákat adhatunk meg vele. (Amikor elkészítettem a foltot, kétségeim voltak afelől, hogy vajon a magyar nyelven kívül van-e ennek különösebb jelentősége. Egy hetet vártam, mire feltettem az sw-discussion OpenOffice.org levelezőlistán a kérdést: Jó-e ez? Egy barátságos francia rögtön gratulált az ötlethez, és a folt hamarosan bekerült a MySpellbe.) Jellemző, hogy az angol nyelvhez Kevin Hendricks mintegy 80 cserét adott meg, a magyar cseretáblázat pedig már 50 csere leírását tartalmazza (például ijj→íj: dijját→díját, vagy öss→ős: erössen→erősen, elösször→először). Megemlíthető még a német helyesírási szótár, amiben már megtalálható a ß→ss, illetve a ss→ß csere.

Ahhoz, hogy az MySpell minél jobban lefedje a magyar helyesírási szabályzatot, jelentős bővítéseket kellett végrehajtani, különösen az összetettszó-képzésben. Ez indokolta a külön MySpell-HU ág létrehozását, illetve az, hogy sok speciális tulajdonságot nem sikerült kellően általánosítani.

Noll Jánosnak és Bencsáth Boldizsárnak köszönhető, hogy a magyar OpenOffice.org Linux és Windows platformra készített változatai a magyar nyelvre szabott MySpell-HU-t tartalmazzák az eredeti MySpell helyett.

Miben más a MySpell-HU, mint az eredeti MySpell? A legfontosabbak a következőek: 6–3 szabály alkalmazása az összetett szavaknál; az összetett szavak keresztbe ellenőrzése; a kötőjelezett szavak, toldalékolt számjegyek kezelése; sok kisebb egyéb szabály kezelése (ez utóbbiaknak (lesz) köszönhető például a *rovarirtószer*, *kéthónapos* szavak kiszűrése). Van egy-két egyéb figyelemreméltó tulajdonsága is, mint például az ékezetesítés, a ragozható tőszavak felvétele és a szótagismétlések javítása.

Az egyszerűség kedvéért az eddigiekben, és a továbbiakban is a Magyar Ispell elnevezés alatt mind a magyar Ispell modult, mind a magyar MySpellt érintő fejlesztéseket értem.

Anyagi támogatás

A fejlesztés sikerének könyvelhető el, hogy kérés nélkül olyan támogatók is jelentkeztek, aki anyagi segítséget ajánlottak fel a projekt folytatásához.

Mátó Péter 12 000 Ft-os felajánlásának köszönhetően két példányt vásároltam a Magyar Helyesírási Szótárból, amelyből egyet Bíró Árpádhoz juttattam el. A szótárak azóta számos vitás esetben szolgáltak döntőbírául[3].

Mayer Gyula segítségével a TypoTEX Kft. az SZT-ISZ-10 pályázaton elnyert kormányzati támogatás 10%-át, 300 000 Ft+áfa összeget ajánlotta fel a Magyar Ispell dokumentációja, és egy tesztrendszer elkészítésének fejében.

A pénz egy nagy teljesítményű számítógép vásárlására lett fordítva, ami ironikus módon lehetővé tette a Magyar Ispell mellett a Helyes-e? hibáinak feltárását is.

2. Válasz a Helyes-e?-re

Eltérő technológiák

A Magyar Ispell „hagyományos” GNU technológián alapszik. Az Ispell eredete a 80-as évekbe, vagy még korábbra nyúlik vissza. Geoff Koenning, az Ispell szerzője dolgozta ki az affixtömörítési algoritmust, ami elengedhetetlen volt a Magyar Ispell megvalósításához (l. Magyar Ispell dokumentáció).

Az Ispell hasítótáblában (asszociatív tár, vagy hash) tárolja a tőszavakat. Egy szó ellenőrzése során a rendszer több lépésben ellenőrzi, hogy a szó tőszóként benne van-e a hasítótáblában, vagy pedig kell keresni hozzá egy illeszkedő ragozási szabályt (ebben segít az affixtömörítési algoritmus). Ha egy ragozási szabály illeszkedik, és az illesztés után előálló szó benne van a hasítótáblában, még hátra van egy ellenőrzés: vajon arra a tőszóra tényleg alkalmazható-e az a ragozási szabály (erre vonatkozóan minden egyes tőszóhoz még kiegészítő információk is el vannak tárolva, az ún. ragozási osztályok egykarakteres kapcsolói).

[3] Bár mint kiderült, nem kell mindent készpénznek venni, amit a szótár ír. Talán a beiratkozás szó, ami tévesen hosszú í-vel szerepel a szótárban.

A MorphoLogic cég Helyes-e? ellenőrzője véges állapotú automatán alapul. Jellemzője, hogy nagyon tömör helyesírási szótárral rendelkezik.

Összevetve a két eltérő technológiát, a következőt lehet kiemelni: a szabályos ragozások, képzők kezelését könnyebb egy véges állapotú automatán alapuló helyesírás-ellenőrzővel elvégezni, de a kivételek, és az összetett szavak ellenőrzése már speciális megoldásokat igényel, ráadásul nehezen általánosítható módon. A nehézség oka az, hogy gyakorlatilag itt szófaji információkat is be kell építeni a rendszerbe, és eljutunk megint a hasítótáblához.

Naszódi Mátyás (MorphoLogic) személyes levelében is ezt erősíti meg: a Helyes-e? technológiai megoldásai végesnek bizonyultak, és nincs továbblépési lehetőség.

Gyakorlatilag évek óta változatlan formában kerül a Helyes-e? a Microsoft alkalmazásokba, így az Office XP-be is.

Nem helyes!

A Helyes-e? számos hibát tartalmaz. A teljesség igénye nélkül, csak az eddig meglelt hibák közül a legfontosabbakat sorolom fel:

Nyelvtani hibák

- Határozószavak elfogadása igekötőként: belémegy, föléugrik, alulugrik.
- -nyi képző a nem szótári töveken: keznyi, szüznyi, lovnyi.
- Dupla birtokjel a nyitótöveken: kézéé, szűzéé, vízéé.
- -szerű: észszerű, firkászszerű elfogadása ésszerű, firkásszerű helyett.
- Feddd elfogadása.
- Túlzott nyelvi pluralizmus, ami megkönnyíti a tévesztések elfogadását. Pl. követezők elfogadása a következők mellett az igeképzők általánosítása miatt. Befejezett melléknévi igenevek szerepeltetése összetett szavakban: elitélt, űrült, serdűlt.

Összetett szavakat érintő hibák

- Szóbokor-letiltások. Pl. eb (ebtenyésztés, ebtartás, ebír), hő (hőszabályzó, motorhő, hőháztartás), sí (síkesztyű, síbaleset, sískemüveg, sítábor), só (sólepárlás, sómennyiség), íz (ízérzékelés, ízfokozó, eperíz), hó (hókotrás, hóeltakarítás, műhó), lé (ananászlé, tojáslé, léalma), ló (lónyak, lópatkolás, lópofa), ól (birkaól, kecskeól, pulykaól), öl (kocsiöl), sas (sasmadár, sasfióka, sasbefogás), sut (kemencesut), az ominózus kakas (kakaspörkölt, amire csúnya szót javasol a Helyes-e?), stb.
- Melléknevek és egyéb szófajok: kishiba, nagyház, felsőkutya, ilydolog, olymacska, magyarvonatkozású, csipásszemű.
- Olyan összetett szavak, amelyeket nem szabadna elfogadni: karvaj, szívéjes, ünnepéj, színtű, szűrrealista, elitélt, tökéj, stb.
- Négy szóból álló összetett szavak el nem fogadása: barnakőszénkoksz, diófalevélszár.

Hibás alakok

Veé, Szűntet, tűntet, bíztat (ragozható igék, nem pedig ragozott melléknévi igenevek), fűzek, kűzdőtér, izület, tikfa, hivő, hivat, kompatíbilis, neutrinó, stb.

Hibás besorolások

Tettról, fiája, valakiseknek, kéthónapos, stb.

Csevej

A Helyes-e? ismert súlyos hibája. A korábbiakkal ellentétben a helyes alakot itt nem fogadja el az ellenőrző, és helyette a rosszat (csevely) javasolja.

Magyar Ispell

A Magyar Ispellben a fent részletezett hibák kiszűrésére ellenőrzésre kerültek a legtipikusabb hibákért felelős i/í, j/ly, o/ó, u/ú, ü/ű tévesztések.

Bíró Árpáddal, és Trón Viktorral a Magyar Ispell levelezési listán folytatott párbeszédet követően a korábbi szóbokor-letiltó mechanizmus helyett az összetett szavak keresztbe ellenőrzésével valósul meg a helyes összetételként elfogadott, de tipikus tévesztések kiszűrése (pl. ilyenek a szervíz, elitélt, színtű, tökéj, stb. alakok).

Ennek eredményeként az összetett szóként külön fel nem vett, tipikus tévesztéssel előálló összetett szavakat a Magyar Ispell felismeri, és javítja.

Helyesírás-javító program

A Helyes-e? talán az eddigieknél is súlyosabb hibája, hogy tevékenysége szinte kizárólag csak az ellenőrzésre szorítkozik: javítási képességei alkalmatlanná teszik arra, hogy akár a legegyszerűbb elütéseket is kijavítsa.[4]

A Magyar Ispell a következő egyedülálló tulajdonságokkal rendelkezik:

- Az összes egy betű távolságra lévő hibás szóalak javítása (betűcsere, betűkihagyás, felesleges betű, betűelütés).
- Az összes adott nyelvre jellemző több betűt érintő hiba javítása (a javítási cseretáblázat segítségével).
- Szótag-duplázások javítása. (Pl. oktatatás, segítségégével. Érdekes, az -at/-tat képzők általános használata miatt a Helyes-e? elfogadja helyesnek az oktatatás, iktatatás, stb. szavakat.)
- A javaslatok valószínűségi sorrendben történő felkínálása (először a tipikus hibák, aztán a betűgyakoriság alapján).

Ezek a részben mennyiségi jellemzők minőségi változást jelentenek: a felhasználó nem kényszerül a kézi módosításra, elég, ha a felkínált javaslatok közül választ.

[4] Természetesen javít a Helyes-e? is, de a betűtévesztések esetén félig-meddig, hiányzó betűk esetén pedig egyáltalán nem javasol semmit.

3. És a jövő?

Tennivaló akad bőven. Tapasztalt vagy bátor C++ programozók segíthetnek az OpenOffice.org magyar elválasztási osztályának elkészítésében, ami a kettőzött többjegyű mássalhangzókat is képes lenne (helyesen) elválasztani. A szinonimaszótár bővítése és javítása, a kötőjelek mellett a nagykötőjel lekezelése a helyesírás-ellenőrzőben. Segítségre van szükség az OpenOffice.org, és egyéb alkalmazások fordításában, ellenőrzésében, használatában, és ha jónak ítéled, népszerűsítésében.

Elképzelhető, hogy a Magyar Ispell nem lesz alkalmas minden nyelvi probléma megoldására. Lehet, hogy egy véges állapotú automata hozzáadására is szükség lesz az igények növekedése esetén. Szerencsére ez nem probléma. A véges állapotú automata készen van. Pasi Ryhanen írta, és jelenleg a legjobb finn helyesírás-ellenőrzőként működik a finn OpenOffice.org-ban, és semmi akadálya sincs, hogy a segítségével akár egy jobb magyar helyesírás-ellenőrzőt készítsünk.

Linux alapú klasztertechnológiák

Papp Dániel

2002.09.26.

Kivonat

A Linuxban is megjelentek olyan fürtözési megoldások, amelyek korábban csak nagygépes rendszerekben voltak elérhetők. Ez a fajta technológia az elmúlt években dinamikusan fejlődött. Megjelentek a szuperszámítógép teljesítményt nyújtó, terhelésmegosztó (load-balancing), nagy elérhetőséget biztosító (high availability, HA) és az egy rendszernek látszó (single system image) fürtök.

Tartalomjegyzék

1. Tudományos-technikai klaszter

1.1. Tulajdonságok

A szuperszámítógép teljesítményét nyújtó fürtözésre jó példa a Beowulf, amely nagy számításigényű tudományos-technikai feladatok megoldására több gép összefűzéséből áll. Ezen fürtöknél a cél a az alkalmazás skálázhatóságának növelése. Itt egy jól párhuzamosítható programra van szükség a megfelelő teljesítményhez, mely elsősorban a tudományos alkalmazására jelent megfelelő megoldást. Üzleti megoldások számára azonban általában nem megfelelő.

1.2. Technikai jellemzők

A Beowulf szoftver a párhuzamosan megírt alkalmazás számára biztosít osztott környezetet. Az adatokat vagy az ütemező vagy egy NFS szerver szolgáltatja. Innen történik az alkalmazás vezérlése is. Az egyes szerverek kiesése csökkenti a számítási kapacitást. Az ütemezőt és NFS-t futtató szervert célszerű HA fürtbe kötni.

Egy példa Beowulfra a németországi Chemnitz Műszaki Egyetemen működő 528 db PIII-800-as processzorral és 20 GB merevlemezzel rendelkező fürt.

2. Terhelésmegosztó klaszter

2.1. Tulajdonságok

A terhelésmegosztó fürt elsősorban webszerverek esetében illetve az elektronikus kereskedelmi alkalmazások számára nyújt kedvező megoldást. Ilyen például a Linux Virtual Server (LVS), ahol több összefogott szerveren ugyanaz az alkalmazás fut, statikus adatokkal, vagy közös adatbázist használva. Felmérések szerint jelenleg az LVS a legnépszerűbb ilyen jellegű implementáció.

2.2. Technikai jellemzők

A terhelésmegosztó fürt működésének alapelve, hogy a bejövő kéréseket a terhelésmegosztó fogadja, majd továbbítja a megfelelő webszerver felé. A válaszokat egyből az ügyfél kapja meg. A HA megoldással kiegészült terhelésmegosztó fürtnél redundánssá tesszük a terhelésmegosztó szervereket. A terhelésmegosztók mögötti szerverfarm menet közben változtatható, azaz kivehetők-berakhatók a szerverek.

3. Magas rendelkezésre állást biztosító klaszter

3.1. Tulajdonságok

Itt több számítógép és periféria összekapcsolásáról van szó, oly módon, hogy azok egyetlen rendszert alkotnak és ez a rendszer működőképes marad akkor is, ha valamelyik rendszerkomponens meghibásodik (No Single Point of Failure). Ilyen fürtökből korábban gyártófüggő megoldások születtek, az adott gyártó hardveréhez kapcsolódóan (pl. VMS, TrueCluster, SGI Failsafe). Később kezdtek megjelenni a PC alkatrészekből, Linux alapon, jóval olcsóbban megépíthető HA fürtök. Ilyen például a nyílt forráskódú Kimberlite fürt.

3.2. Technikai jellemzők

HA fürtnél általában 2 szerver van összekötve. Van egy közösen használt SCSI vagy Fibre channel lemez alrendszer, ahol a szerveren futó alkalmazások és a klaszter szoftver által létrehozott adatbázis adatai kerülnek tárolásra. A két gépet heartbeat csatornák kötik össze, melyek lehetnek soros vonaliak vagy LAN-on (elsősorban Ethernet) keresztül működők. Továbbá speciális raw device-okon keresztül is kommunikálnak egymással a fürt tagok, az előbb említett külső lemez-alrendszeren keresztül. Fontos tulajdonság, hogy a szerverek le tudják kapcsolni a másik fürt tagot. Az alkalmazások módosítása nélkül lehetséges úgynevezett szolgáltatásokat (service) meghatározni.

Fontos a fürt jó menedzselhetősége, ezt a Kimberlite fürt esetében karakteres felületről tehetjük meg. Ugyanezen szoftver dobozos verziójánál, a Convolonál létezik egy Babel névre hallgató webes, grafikus beállító felület. Mindkét helyen módosíthatjuk a fürt konfigurációját, állíthatjuk a naplózási szinteket, hozzáadhatunk, törölhetünk és letilthatunk szolgáltatásokat. Ezen felül természetesen különféle információkat is kaphatunk a fürt egyes tagjairól.

3.3. HA fürtök fejlődési szakaszai

- Átkapcsolásos fürtök (Failover Cluster): Egy adott alkalmazás egyszerre csak egy tagon futhat. A fürt tagjai figyelik egymás állapotát és hiba esetén átveszik a másikon futó alkalmazásokat.
- Párhuzamos fürtök (Parallel Cluster): Az alkalmazás párhuzamosan futhat mindegyik fürt tagon. Ehhez az alkalmazás módosítására van szükség.
- SSI - Single System Image Cluster: Egyetlen rendszernek látszik. Általában nem igényel alalmazás módosítást.

4. SSI klaszter

4.1. Tulajdonságok

A Single System Image klaszter jellemzője, hogy egyetlen rendszernek látszik az a több, akár különböző gép, amiket magába foglal. Tehát ideális esetben transzparens az alkalmazások számára és így azok nem igényelnek módosítást. Itt is előny a jó skálázhatóság és a magas rendelkezésre állás. Az SSI fürtök esetében is cél a rendszer erőforrásainak optimális kihasználása. Ilyen fürt például a MOSIX (Multi-computer Operating System for unIX).

4.2. Technikai jellemzők

A Mosix előnyei között sorolhatók fel a naprakész frissítések (2.4.19-es kernel támogatás), az, hogy – jó esetben – nem igényli az alkalmazás módosítását, a közös fájl és processz tábla, valamint az automatikus processz relokálás.

5. Összefoglalás

A Linux-alapú fürtök segítségével kisebb méretű cégek, alkalmazások is részesülhetnek a klaszter-technológia nyújtotta előnyökben. A Beowulffal szuperszámítógép tel-

jesítményt hozhatunk ki rendszerünkből, az LVS-sel kiszolgáló-farmokat építhetünk, a HA klaszterek révén pedig megfelelő megbízhatóságú platformot nyújthatunk akár nagyvállalati alkalmazások számára is.

6. Hivatkozások

http://www.beowulf.org/
http://www.linuxvirtualserver.org/
http://oss.missioncriticallinux.com/
http://www.missioncriticallinux.hu/
http://www.mosix.org/
http://www.openmosix.org/
http://www.debian.org/
http://www.steeleye.com/
http://www.resonate.com/
http://www.redhat.com/
http://www.turbolinux.com/

Az *LME* jelene és jövője

Sári Gábor
<saga@lme.linux.hu>

2002. október 21.

Kivonat

Köszöntöm önöket a Konferencia kiadvány olvasói között. Előadásomban a Linux-felhasználók Magyarországi Egyesületével (továbbiakban *LME*) kapcsolatban szeretnék egy kicsivel részletesebb információkat megosztani önökkel. Röviden a múltról, bővebben a jelenről és a jövőbeni terveinkről.

Tartalomjegyzék

1. A dicső múlt

Az Egyesület négy évvel ezelőtt 1998 őszén alakult, amit közel fél éves jogi eljárás követett, emiatt a végleges Bírósági bejegyzés 1999. tavaszán történt meg.

Címszavakban a múlt jelentősebb eseményeiről:

1999

- INFO ’99
- ETB Linux Konferencia
- I. Linux Konferencia
- Compfair ’99
- Linux Mikulás

2000

- II. Linux Konferencia
- Compfair 2000
- Linux Mikulás

2001

- Az első eredményes MEH pályázat. Célja az *OpenOffice.org* honosítása, az *FSR* (Fordítás Segítő Rendszer) és a *Pingvin Füzetek* megvalósítása.
- III. GNU/Linux Konferencia
- INFO 2001
- 11.14:
  A második MEH pályázat eredményeként megkezdődött az *Infoszféra Kft*-vel közösen megrendezésre kerülő, hat alkalomból álló Linux Konferenciasorozat. Az elhangzott előadások követőkiadványai letölthetők a következő címről: `http://www.lme.hu/konfsorozat/program.html`

2. Napjaink eseményei

2002

- 01. 08:
  Az *LME* által megrendelt *OpenOffice.org* honosítás első verziója nyilvánosságra kerül.

- 02.01-04.
  Az *FSF Magyarország Alapítvány* magja és az általuk összetoborzott több mint száz fős csapat létrehozza azt, amit az LME által megbízott cég nem tudott határidőre elvégezni. Emiatt az *LME* új vezetősége 04.16-án felmondta a szerződést a honosítás feladatát elvállaló Betéti társasággal. Az ílymódon felszabaduló összeget az *FSF Magyarország Alapítvány* és a később megalakult *LME Honosítás Csoport* feladatainak finanszírozására szántuk.

- 03.23.

  Lezajlott az *LME* 2002. évi Küldött Közgyűlése, megválasztásra került a máig is funkcionáló új vezetőség.

- 04.17.

  Megkezdődött a Fix.tv-ben Magyarország első Linux témájú műsora, ami azóta is minden szerdán 16-17 óra között jelentkezik. A műsor vezetője Gibizer Tibor az *LME* egyik aktív tagja.

- 04.24

  Befejeződött a fél éve tartó Linux Konferenciasorozat.

  A technikai fejlődés gyors ütemét és a szakma igényeit figyelembe véve a szakmai rendezvénysorozatot a jövőben is mindenképpen folytathatónak tartjuk.

  Terveink szerint a rendezvénysorozatot az *LME* és az *Infoszféra Kft.*, hasonló együttműködési feltételekkel folytatni fogja.

  A rendezvénysorozat követőkiadványaiból könyvet szándékozunk összeállítani, melyet a nyílt forrású dokumentációknál már bevált modell szerint terjesztünk: a könyv FDL licenc alatt elérhető elektronikus formában, és bármely könyvkiadó kiadhatja papír alakban. A könyvkiadó a szokásjog és saját eladásainak fellendítése érdekében a keletkező profit egy részét támogatás formájában visszaforgatja az Egyesületbe, így elősegítve a rendezvénysorozat továbbvitelét.

- 05.04.

  Az *LME 2002.* évi első Rendkívüli Küldött Közgyűlése a 2002. évi módosított költségvetés ismételt benyújtására és elfogadására.

- 06.20.

  Kikerülnek a `http://www.lme.hu/meh/pingvinfuzetek.html` oldalra a Pingvin Füzetek első, nyilvánosságra hozott darabjai.

- 06.30.

  A Konferenciasorozattal kapcsolatos elszámolások benyújtásra kerültek a *Miniszterelnöki Hivatal Informatikai Kormánybiztosságához*. Egy apró formai hiányosságot júliusban pótoltunk. A Konferenciasorozat utólagos finanszírozása miatti 4,5 milliós támogatási igényünk átutalása elől így minden akadály elhárult.
  30-60 napra ígérték az átutalást, de sajnos a kézirat írásának idejekor még nem került rá az Egyesület számlájára a pénz.

- 08.13.

  Az *LME Honosítás Csoportja* Dologh Ervintől adományként 50.000.- Ft támogatást kapott.

- 09.09.

  Az *APEH* értesítése alapján a Személyi Jövedelemadó egy százalékos felajánlásából az *LME* részére felajánlott összeg: 225.067.- Ft.

- 09.13-19.

  Az *INFOmarket Információtechnológiai és Telekommunikációs Vásáron* Gibizer Tibor képviselte az *LME*-t a *Kiskapu Kft.* standján.

- 09.25-26.

  Civiliáda 2002

  A rendezvény Internet Kávézójának üzemeltetését Balázs Tibor, az *LME* titkára biztosította.

- 09.30.

  Az Egyesület 2002. évi költségvetési tervében 05.04-én jóváhagyásra került az `ftp://ftp.fsn.hu/` (a talán legnagyobb forgalmat lebonyolító hazai szabad szoftvereket tartalmazó FTP kiszolgáló) diszkbővítésének finanszírozása.

  Ezen a napon megkezdődött az összesen közel 1,6 Terrabájt[1] méretű lemezegységek beszerzése.

3. A szép jövőről

Jövőbeni terveinkkel kapcsolatban a következőket tudom megemlíteni:

- Az ez évi, remélhetőleg nagy sikerű Szakmai Konferencia után a jövő évi (jubileumi) V. Szakmai Konferenciát már a 2002. év végén elkezdjük szervezni.
- Amennyiben komoly igény merül fel rá, és a „társszervezetek” is beleegyeznek, 2003. első felében rendezünk egy közös *Szabad Szoftver Konferenciát* az *LME*, az *FSF Magyarország* és a *BSD Egyesület* közreműködésével. Ez ügyben nagyon fontosak lesznek a mostani Konferenciával kapcsolatos szervezési, üzleti tapasztalatok.
- Az *LME* költségvetésében komoly tétel az *OpenOffice.org* honosításával kapcsolatos összeg. Ennek felhasználási módjáról az *FSF Magyarországgal* és az *LME honosítási Csoportjával* meg kell kezdenünk az egyeztetést.
- Célunk a *Honosítás Csoport* feladataihoz kapcsolódó ez évi adományhoz hasonló felajánlások számának jövőbeni növelése.

4. Hivatkozások

`http://www.lme.hu/` – Az *LME* honlapja
`http://www.linux.hu/` – Az *LME* által üzemeltetett Linux-os híroldal
`http://www.bsd.hu/` – A *Magyar BSD Egyesület* honlapja
`http://www.fsf.hu/` – Az *FSF.hu Alapítvány* honlapja
`http://office.fs.hu/` – Az *OpenOffice.org* honosított változatának honlapja

[1] $1,6 \cdot 10^9$ bájt.

Webszerver védelme határvédelmi eszközökkel

Scheidler Balázs
<bazsi@balabit.hu>

2002.09.30.

Kivonat

Előadásomban egy valós eset tapasztalatait igyekszem ismertetni, illetve az alkalmazott megoldásokat ismertetni.

A feladat szerint egy portál jellegű oldalt kell védelmi rendszerrel ellátni. A portál fejlesztése a végéhez közeledett, de tervezése során biztonsági szempontokat nem vettek figyelembe.

Mivel a honlapok feltörése és tartalmuk megváltoztatása látványos, ezért crackerek körében általában kedvelt célpont egy webszerver. A kockázatot jelen esetben emelte, hogy az oldal várt látogatottsága magas, továbbá rendszeres sorsolásokon értékes ajándékok nyerhetők.

Tartalomjegyzék

1. Követelmények

A honlapot kiszolgáló szerver egy internet szolgáltató szervertermében található, anyaggal való töltése szintén az interneten keresztül, egy adminisztrációs felületen keresztül történik. Legfontosabb követelmények:

- Az oldalon viszonylag magas látogatottság, napi kb. egymillió találat várható.
- Az adminisztrációs felületet az internetre bárhonnan csatlakozó emberek használhatják
- Az alkalmazás Microsoft IIS platformon fut, ASP oldalakat használva
- A teljes rendszert távolról kell tudni menedzselni.

2. Célok

A védelmi rendszer kialakításakor a következő célokat követtük, a fenti követelmények figyelembe vétele mellett:

- Operációs rendszer védelme
- Webszerver szoftver védelme (IIS)
- Webes alkalmazás védelme (információ rejtés, támadási kísérletek védelme)
- Adminisztrátorok erős hitelesítése (X.509 tanusítványok)
- A nem megbízható protokollok titkosítása és hitelesítése (IPSec VPN)
- Üzembiztos működés

3. Megvalósítás módja

Mivel az alkalmazás rendszerterve már elkészült, és mivel követelmény volt az eredeti rendszertől független működés, ezért védelem egyik legfontosabb eszköze egy határvédelmi eszköz, egy tűzfal lett. A maximális biztonság érdekében alkalmazás szintű tűzfalat, a Zorpot alkalmaztuk.

További eszközként felhasználtunk erős hitelesítést lehetővé tevő hardware tokeneket. Egy ilyen token egy vagy több X509 tanúsítványt illetve a hozzá tartozó titkos kulcsot tárol. A felhasználót úgy képes azonosítani, hogy a titkos kulcs a tokent nem hagyja el, azaz a digitális aláírást maga az eszköz végzi.

Az egyes komponensek védelmét a következő módon valósítottuk meg a fenti eszközök segítségével.

3.1. Operációs rendszer védelme

A webszerverként beállított gépen gyakran futnak olyan szolgáltatások, melyek a rendszer üzemeltetése szempontjából nem feltétlenül szükségesek. Ezeket a szolgáltatásokat a tűzfalon nem engedjük át, kívülről elérhetetlenné tesszük, így nem jelentenek kockázatot.

Ezt a tűzfal ún. „default deny” hozzáállása segíti, azaz nem azt soroljuk fel amit tilos, hanem minden tilos, és az engedélyezett szolgáltatásokat soroljuk fel egyesével.

3.2. Webszerver szoftver védelme

A webszerver által nyújtott HTTP szolgáltatást a tűzfalon engedélyeznünk kell, hiszen enélkül a honlapunk nem működőképes. Csomagszűrő tűzfal esetén a lehetőségeink végére érnénk, azonban alkalmazás szintű tűzfallal, mint amilyen a Zorp is, tovább szűkíthetjük ezt a csatornát a biztonság növelése érdekében.

Az első ötlet az, hogy próbáljunk meg minden lehetséges - a cracker számára hasznos - információt elrejteni. Itt fontos megjegyezni, hogy az információ rejtés (angol kifejezéssel security by obscurity) önmagában nem emeli a biztonságot, viszont a cracker dolgát nehezíti, amennyiben az elrejtett információval más forrásból nem rendelkezik.

„Server” fejléc szűrése

A legszembetűnőbb adat, amit elrejthetünk a kíváncsi szemek elől az a szerver típusa, melyet a kiszolgáló minden kérésre adott válaszában elküld. Ez nagyon gyakran olyan adatokat is tartalmaz, melynek birtoklása a szerver feltörése szempontjából elengedhetetlen (pl: verziószám, beépített kiegészítések, stb).
Ezt a következő minta is mutatja:

```
HTTP/1.1 200 OK
Date: Sat, 28 Sep 2002 18:35:51 GMT
Server: Apache/1.3.9 (Unix) Debian/GNU PHP/4.0.3pl1 mod_ssl/2.4.10
OpenSSL/0.9.5a
Last-Modified: Sat, 29 Apr 2000 07:07:45 GMT
ETag: "ff87-ffe-390a8a41"
Accept-Ranges: bytes
Content-Length: 4094
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

A „Server:” fejléc szűrése Zorpban:

```
from Zorp.Core import *
from Zorp.Http import *

class PublicHttpProxy(HttpProxy):
        def config(self):
                HttpProxy.config(self)
                self.response_headers["Server"] = (HTTP_HDR_DROP)

def inter():
        Service("inter_HTTP", PublicHttpProxy)
        # szükséges még egy ilyen ipchains rule
        # ipchains -A input -d 0/0 80 -j REDIRECT 50080
        Listener(SockAddrInet("1.2.3.4", 50080), "inter_HTTP")
```

Hibák szűrése

Dinamikus weboldalaknál hiba esetén, gyakran kerül ki olyan adat a szerverről, ami a támadás során felhasználható:

- tartalmazhatja a weboldal gyökerét a szerveren belül
- tartalmazhat konkrét lekérdezést, melyet az adatbázisnak nem sikerült végrehajtania
- tartalmazhat utalást belső változókra, stb.

Az információrejtést szolgálja, ha minden válasz hibaoldalt letiltunk, és egy egységes 404-es hibakódot (tartalom nem található) adunk vissza, attól függetlenül, hogy pontosan milyen hiba történt.
Ezt a következő Zorp konfiguráció szabályozza:

```
class PublicHttpProxy(HttpProxy):
  def config(self):
    HttpProxy.config(self)
    self.response["*", "4"] = (HTTP_RSP_POLICY, self.filterError)
    self.response["*", "5"] = (HTTP_RSP_POLICY, self.filterError)
    self.response_headers["Server"] = (HTTP_HDR_DROP)
    self.error_silent = TRUE

  def filterError(self, method, url, version, response):
    self.error_code = 404
    return HTTP_RSP_DENY
```

„Host” fejléc szűrése

A böngésző minden kérésnél megadja, hogy a szerveren belül melyik virtuális szervert kívánja megszólítani. Erre szolgál a „Host” fejléc, ahogy azt a következő minta mutatja:

```
GET / HTTP/1.0
Host: www.akarmi.com
```

A legtöbb cracker programokat használ a megfelelő célpont megkeresésére. Ezek általában egy IP tartományon mennek végig, kérést intéznek a feltételezett webszerverhez, majd ha az nem válaszol, vagy nem megfelelően válaszol, akkor mennek tovább, másik szervert keresve. Ezek a programok általában kizárólag a gép IP címét tudják, a géphez tartozó nevet nem, egyedül a DNS alapján próbálhatja meg az IP címhez tartozó nevet megkeresni. (ide bejegyezhetünk egy teljesen független nevet)

Ebből az következik, hogy a Host fejlécbe IP címet, vagy egy általános nevet adnak meg (pl: www). Ha mi pontosan ismerjük a védett szerveren található virtuális gépeket, akkor ismeretlen névre érkező kérést nyugodtan megtilthatjuk. Ezzel az ilyen kereső programokat megtéveszthetjük.
Ennek megvalósítása:

```
class PublicHttpProxy(HttpProxy):
  def config(self):
    HttpProxy.config(self)
    self.response["*", "4"] = (HTTP_RSP_POLICY, self.filterError)
    self.response["*", "5"] = (HTTP_RSP_POLICY, self.filterError)
    self.response_headers["Server"] = (HTTP_HDR_DROP)
    self.request_headers["Host"] = (HTTP_HDR_POLICY, self.filterHost)
    self.error_silent = TRUE

  def filterError(self, method, url, version, response):
    self.error_code = 404
    return HTTP_RSP_DENY

  def filterHost(self, hdr_name, hdr_value):
    if hdr_value == "www.valami.com":
      return HTTP_HDR_ACCEPT
    return HTTP_REQ_REJECT
```

URL szűrés

Az IIS szerverek jellemző hibáihoz jellemző kérések tartoznak, melyeket mindenképpen érdemes megtiltani és naplózni, hogy a támadási kísérletekről is tudomást szerezzünk.

Szűrhető például minden olyan URL, amelyik „cmd.exe”-t vagy „../”-t tartalmaz.

```
class PublicHttpProxy(HttpProxy):
  url_matcher = FileRegexpMatcher("/etc/zorp/url.black", "/etc/zorp.url.white")

  def config(self):
    HttpProxy.config(self)
    self.response["*", "4"] = (HTTP_RSP_POLICY, self.filterError)
    self.response["*", "5"] = (HTTP_RSP_POLICY, self.filterError)
    self.response_headers["Server"] = (HTTP_HDR_DROP)
    self.request_headers["Host"] = (HTTP_HDR_POLICY, self.filterHost)
    self.request["GET"] = (HTTP_REQ_POLICY, self.filterURL)
    self.error_silent = TRUE

  def filterError(self, method, url, version, response):
    self.error_code = 404
    return HTTP_RSP_DENY

  def filterHost(self, hdr_name, hdr_value):
    if hdr_value == "www.valami.com":
      return HTTP_HDR_ACCEPT
    return HTTP_REQ_REJECT

  def filterURL(self, method, url, version):
    if self.url_matcher.checkMatch(self.request_url_file):
      return HTTP_REQ_REJECT
    return HTTP_REQ_ACCEPT
```

3.3. Alkalmazás védelme

Miután az operációs rendszerrel illetve a web-kiszolgálóval kapcsolatos protokoll szűkítéseket megtettük, érdemes az alkalmazáshoz igazítanunk a beállításainkat.

Ha például tudjuk, hogy az adminisztrációs felület a /admin alkönyvtárban található, akkor ezt az URL-t kliens IP címhez köthetjük. Esetleg, amennyiben egy alkalmazás csak .php, .png, .jpg valamint .html oldalakat tartalmaz, akkor letilthatjuk az összes egyéb kiterjesztést.

Ezzel még inkább azonosítani tudjuk az automatikus scannereket illetve a támadási kísérleteket.

Természetesen az elfogott kéréseket folyamatosan ellenőriznünk kell.

3.4. Adminisztrátorok hitelesítése

Az érzékenyebb területeket mindenképpen érdemes titkosítással, valamint erős hitelesítéssel védeni. Ilyen érzékeny terület a honlap adminisztrációs felülete, melyen keresztül a cikkek feltöltése folyik.

Mostani megoldásunkban a tűzfal terminálja a bejövő HTTPS kéréseket, ellenőrzi, hogy a kliens rendelkezik-e tanúsítvánnyal, majd ha igen, akkor azokat titkosítás nélkül küldi be a webszervernek. Ezzel a módszerrel munkát veszünk le a webszerver válláról, így rendszerünk nagyobb teljesítményű is lesz.

Az adminisztrátorok a követelményeknek megfelelő számos token egyikét kapják, mely a böngészőjükbe beépülve kényelmesen használható.

3.5. Nem megbízható protokollok

A webszerverhez sajnos számos olyan protokollt kell beengedni, melynek biztonságában nem bízhatunk meg. Ilyen például a távoli menedzsmentet lehetővé tevő Terminal Services, valamint a fájlmegosztásokért felelős SMB protokoll.

Ezeket a csatornákat csak titkosított és hitelesített kapcsolaton keresztül nyitjuk meg. A megvalósítás pont-pont IPSec VPN kapcsolatokat használ, ahol a két felet szintén X.509 tanúsítvány segítségével azonosítjuk.

4. Konklúzió

Bár a rendszer utólag is hatékony védelemmel látható el, jobb eredményt érhettünk volna el, amennyiben már az alkalmazás fejlesztése kapcsán is figyelembe vehettük volna a biztonsági szempontokat.

A védelem gyenge pontja az alkalmazás, utólag ennek biztonságáról kizárólag audit vagy penetration test során győződhetünk meg.

Láthattuk, hogy a technikai eszközök széles skáláját vonultattuk fel, ezek megfelelő kiválasztása garantálja az egész rendszer biztonságát.

Grid Rendszerek

Szalai Ferenc
szalai@sztaki.hu

2002. október 17.

Kivonat

Az előadás elsődleges célja, hogy bemutassa a GRID technológiát és az ehhez kapcsolódó szabad szoftver fejlesztéseket. Az előadásban röviden összefoglaljuk a GRID rendszerek fejlődését kezdve a párhuzamos programok írását elősegítő üzenetküldő rendszerektől (PVM[1], MPI[2]) egészen a jelenleg is fejlesztés alatt álló technológiákig.

Tartalomjegyzék

[1] http://www.csm.ornl.gov/pvm/pvm_home.html
[2] http://www.mcs.anl.gov/mpi/

1. Bevezető

A Grid rendszerek eredete valahova a kilencvenes évek közepéig nyúlik vissza, és főleg a párhuzamos és elosztott számítási rendszerek kutatásaiban keresendő a gyökere. Nagy lökést adott a kutatásoknak, hogy a nagyteljesítményű szuperszámítógépek mellett megjelentek a különböző klaszter rendszerek is, melyek jóval alacsonyabb áron biztosítottak megközelítőleg azonos teljesítményt. A Grid kutatások előre törésének további oka, hogy nyilvánvalóvá vált, hogy a jelenlegi technika fejlődési üteme nem bírja a lépést a műszaki, tudományos téren újonnan keletkező igényekkel.

A fenti problémákra természetes válaszként érkezett a Grid kutatások létrejötte, melyet szigorúan véve Ian Foster és Carl Kesselman által 1998-ban szerkesztett *The Grid*[5] című könyvtől datálhatunk.

2. Mi az a Grid?

A párhuzamos számítási rendszerek „ősatyáinak” a PVM, MPI könyvtárakat tekinthetjük. Ezek lehetővé teszik általában FORTRAN, C, C++ nyelven írt programjaink számára, hogy bizonyos feladatokat párhuzamosan hajtsunk végre. Ezek a könyvtárak elsősorban a párhuzamos végrehajtáshoz szükséges kommunikáció, adatcsere lebonyolításában segítenek. Az utóbbi időben az MPI szabvány kezdi átvenni az uralkodó szerepet köszönhetően annak, hogy ezt a különböző szuperszámítógép gyártók is támogatják és elkészítik a saját architektúrájukra optimalizált változatukat. Ezek mellett számos szabad forráskódú implementáció is elérhető (PVM, MPICH[3], LAM[4]).

A fenti programozási paradigmán igyekezett túllendülni Foster és Kesselman mikor a jelenlegi informatikai infrastruktúrát az 1910-es évek elektromos hálózati (electric power grid – innen az elnevezés) szerkezetéhez hasonlította, mondván hogy akkoriban a megbízható, konzisztens, teljes körű és olcsó elektromos szolgáltatáshoz való hozzáférés ugyanolyan nehéz volt, mint ma a hasonló tulajdonságokkal rendelkező számítási kapacitáshoz való hozzáférés.

A Grid kutatások tehát azt a célt tűzik ki maguk elé, hogy az informatikai infrastruktúrában található erőforrásokat (számítási teljesítmény, merevlemez-kapacitás, adatok, emberek, vizualizációs eszközök stb.) egy világméretű rendszerben az arra jogosultak számára egységesen hozzáférhetővé tegye.

A rendszer legfőbb eleme a heterogenitás és központi fogalma az absztrakt erőforrás. Míg a 80-as években a párhuzamos és elosztott alkalmazások legtöbbje homogén számítási környezetben általában szuperszámítógépeken futott, addigra a 90-es évek végére a heterogén környezetek felé fordult a figyelem. A homogén architektúrákon sikerrel alkalmazott technológiák, mint a PVM és az MPI vagy egyéb üzenetküldő rendszerek esetleg automatikus párhuzamosítást lehetővé tevő programozási nyelvek (HPF) rendre kudarcot vallottak nagy, elosztott, heterogén környezetekben.

3. Mire használjuk?

A Grid rendszerek minden olyan esetben használhatóak ahol nagy mennyiségű közös erőforrást (számítási kapacitás, mérő eszköz stb.), adatot, numerikus szimulációk

[3] http://www.mcs.anl.gov/mpi/mpich/
[4] http://www.lam-mpi.org/

eredményét kell megosztani egymás között kutatóknak. Az ilyen típusú munkakapcsolatokat nevezzük *Virtuális Organizáció*nak.

Természetesen megfelelő analógiák figyelembe vételével az iparban is használható ez a technológia. Egyfelől az eredendően elosztott problémák (pl.: vállalat irányítás) másfelől a nagy rendelkezésre állású rendszerek esetén. Mindezek ellenére manapság inkább a tudományos-műszaki élet igényeit vesszük figyelembe a Grid rendszerek tervezésnél.

Egy életszagú felhasználási példa [2] a gravitációs hullámdetektorok adatainak feldolgozása. Itt az adatok egyfelől nagyszámú mérési és szimulációs eredményből születnek a világ több különböző pontján, néhány Petabájt nagyságrendben. Ennek sem tárolása sem feldolgozás nem oldható meg egy rendszeren, már csak azért sem mert a kutatók fizikailag nincsenek egy helyen. Tegyük fel ugyanis, hogy a Hannoverben felállított detektor adatait felhasználva Japánban egy kutató csapat szimulációkat indít annak vizsgálatára, hogy az észlelt esemény nem egy szupernova robbanás vagy egy neutroncsillag jelei-e. Amerikában ennek a szimulációnak az eredményeit vizualizálni kívánják egy Cave-ben, ahol a várható fizikai folyamatokat is vizsgálni kívánják. Látható, hogy rengeteg hely különböző eszközeinek, számítási architektúráinak és programjainak jól összehangolt munkája ez az egyébként a jövőben mindennaposnak nevezhető tevékenység.

4. OGSA - Open Grid Services Architecture

4.1. Globus

A korai Grid infrastruktúrák egy példánya az Argone National Laboratory és az ISI által fejlesztett szabad forrású Globus[5] rendszer. A rendszer legnagyobb hozománya a *szolgáltatás* orientált megközelítés. Ennek lényege, hogy a Grid erőforrásokat különféle jól meghatározott szolgáltatások segítségével érhetjük el és használhatjuk. Ezek a szolgáltatások a következőek lehetnek (legfontosabbak):

1. *Biztonság - Grid Security Infrastructure (GSI)*: Feladata, hogy egy elosztott heterogén autentikációs, kódolási és autorizációs megoldást nyújtson a virtuális organizáció minden eleméhez és az abban végrehajtható műveletekhez. Jelenleg a X.509 Certificate-et használó nyílt kulcsú titkosítási eljárások használatosak.

2. *Információs rendszer - Grid Information System (GIS)*: A rendszer állapotáról, a rendelkezésre álló erőforrásokról az *erőforrás brókerek* számára állít elő információkat.

3. *Erőforrás bróker - Resource Broker*: Az alkalmazói programok és felhasználók erőforrás igényeit igyekszenek kielégíteni úgy, hogy az *erőforrás allokátor*ok segítségével egy adott feladatot egy adott erőforrás csoportra helyezik el és ha szükséges menet közben változtatják azok elhelyezését.

4. *Erőforrás allokátor - Resource Allocator*: Olyan szolgáltatás, mely lehetővé teszi egy program adott erőforráshoz való hozzáférését, onnan való elmozgatását, és ha szükséges, jegyzi az alkalmazások adott erőforrásra vonatkozó előjegyzéseit is. Számítási erőforrás esetén ez azt jelenti, hogy el tudja indítani, le tudja állítani, „checkpoint-olni” tudja az alkalmazásokat.

[5] http://www.globus.org/

5. *Monitorozó rendszer - Grid Monitoring System*: Feladata, hogy folyamatosan nyomon kövesse a rendszer állapotváltozásait. A rendszer működése során keletkező adatokat, log fájlokat, teljesítmény elemzéshez szükséges adatokat eljuttatja az igénylőhöz.

6. *Adat és fájl hozzáférés - Storage Management*: a számítások megkezdéséhez szükséges, és az azok során keletkező fájloknak, valamint magunknak a bináris, futtatható állományoknak az egységes elérését biztosítja. Ezenkívül az adattároló erőforrásokhoz (adatbázis szerverek, memória silók stb.) is egységes hozzáférést nyújt.

A Globus rendszer a fenti szolgáltatásokra mutatott egy lehetséges megoldást.

Használata során a felhasználó a GSI segítségével autentikálta magát egy kiválasztott erőforráshoz (a Globus erőforrás brókert nem tartalmaz). Ott az erőforrás allokátor (Grid Resource Allocation Management (GRAM)) segítségével elindított egy alkalmazást. Ezzel együtt regisztrálta az alkalmazást a GIS-ben – mely a Globusban egy szimpla LDAP adatbázis – a későbbi elérhetőség miatt. Meg kell jegyezni, hogy a Globus a Monitorozást is az említett LDAP adatbázisban kívánta megvalósítani – szerény sikerrel.

A felhasználás során kiderült azonban, hogy ezek legtöbbje alig vagy egyáltalán nem megfelelően működik a fentebb vázolt felhasználási igények esetén. Ezekből a tapasztalatokból kiindulva kezdek a Globus fejlesztők is új irányba mozdulni és ebből született meg az OGSA.

4.2. OGSA

Az OGSA valójában az IBM által bevezetett *Web Services* technológia és a Globus rendszer keveréke.

A Web services egy módszer arra, hogyan lehet XML alapokon leírni és hozzáférni a szoftver komponensekhez heterogén elosztott rendszerben. Az IBM eredetileg ténylegesen web szerverek szolgáltatásinak definiálására szánta. Az OGSA három komponenst használ a technológiából:

1. *SOAP - Simple Object Access Protocol*: XML alapú üzenetküldési mechanizmus.

2. *WSDL - Web Service Description Language*: Egy XML dokumentum mely meghatározza a Grid szolgáltatás tulajdonságait.

3. *WSIL - Web Service Inspection Language*: WSDL-ek elérését írja le. Általában egy URI a WSDL-re vagy egy UDDI (Universal Description, Discovery and Integration) tárolóra.

Ezeknek az eszközöknek a segítségével definiálhatjuk a OGSA Grid Szolgáltatás modellt. A szolgáltatások olyan entitások, melyek egy adott mechanizmus szerint érhetők el, és egy vagy több jól meghatározott protokollt használnak. A szolgáltatásoknak több különböző példánya lehet jelen a rendszerben egyszerre. A szolgáltatás példányok dinamikusan jönnek létre és szűnnek meg az igények és a teljesítmény-elvárásoknak megfelelően. A Grid szolgáltatás egy adott példányának az eléréséhez szükség van egy WSIL segítségével megadott GSH-ra (Grid Service Handler). Ezt a szolgáltatás használója vagy tudja vagy egy UDDI rendszerből szedi. A GSH minden szolgáltatás példányra egyedi és akkor is létezik, ha maga a szolgáltatás példány nem. A GSH tartalmazza a szolgáltatást biztosító host elérését. Mivel a GSH a szolgáltatás példány lététől

függetlenül megtalálható, és nem tartalmazza a szolgáltatás által használt protokollok leírását, szükség van egy GSR-re (Grid Service Reference). Egy GSR-t a GSH segítségével kaphatunk meg egy GSH$\Longleftrightarrow$GSR leképező szolgáltatástól. A GSR egy olyan kiterjesztett WSDL leírás mely a szolgáltatás használatához szükséges adatokat tartalmazza. Többek között az elérési protokoll leírását is, ami ajánlottan SOAP, valamint azoknak az interfészeknek a leírását melyeket az adott szolgáltatás implementál.

Az OGSA bevezeti a *Notification*-okat, melyek segítségével egyszerű értesítéseket küldhetnek a szolgáltatások egymásnak. Egy szolgáltatás feliratkozhat egy notification-re amennyiben megvalósítja a hozzá tartozó *NotificationSink* interfészt. Egy szolgáltatás tud kibocsátani notification-okat, amennyiben megvalósítja a *NotificationSource* interfészt.

Az OGSA tehát elsősorban a Grid szolgáltatások specifikálásával és azok egységes elérését határozza meg. A rendszernek jelenleg egy demo szintű Java alapú implementációja érhető el.

5. GridLab

A GridLab[6] projekt a Supercomputing 2000 konferencia sikerén[1] felbuzdulva jött létre 2001-ben. Az akkori konferencián bemutattunk egy olyan alkalmazást, mely a Cactus[7] nevű környezet segítségével rácsszámításokat végzett 9 nagy európai szuperszámítógép központ összes gépen valamint képes volt worm szerűen közlekedni a különböző erőforrások között.

Az akkori tapasztalatok nyilvánvalóvá tették, hogy a Grid rendszerek összes hasznos és jó tulajdonságát kihasználni képes programok készítése még a hagyományos párhuzamos programokénál is jóval nehezebb feladatot ró a programozókra. A Gridlab projekt tehát egy olyan architektúra elkészítését tűzte ki maga elé, mely elsősorban a Grid alkalmazások készítőit segíti. Ezt nevezzük GAT-nak. (Grid Application Toolkit)

A rendszer négy, egymásra épülő rétegből áll. Az *Alkalmazás* rétegben a Grid igénylő és korábban már bemutatotthoz hasonló alkalmazások találhatóak, melyek az *GAT API* réteget használják a Grid erőforrásainak elérésére. A GAT API egy *GAT Engine*-ből és több úgynevezett *GAT Adaptor*-ból áll. Az Engine szolgáltatja az API-t az alkalmazó programok felé és tartalmazza azoknak a tulajdonságoknak és képességeknek a leírását mely a Grid használatához szükségesek, míg az Adaptorok az API elemeit képezik le egy konkrét Grid infrastruktúrára. Egy adott Grid infrastruktúra az előző fejezetben említett *szolgáltatás*-halmazt definiál. Ezen a szinten támaszkodhat a GAT a már meglévő Grid rendszerek egyes szolgáltatásaira, mint az OGSA, Globus, Legion stb. Ugyanakkor ezek a szolgáltatások valamilyen *rendszer* szintű megvalósítással kell, hogy rendelkezzenek.

Ez a szerkezet is jól mutatja, hogy a GridLab projekt elsősorban az alkalmazás programozói oldaláról közelíti meg a Grid-et, csak nagyon keveset mond a tényleges Grid szolgáltatásokról ill. azok implementációjáról. Bizonyos esetekben azonban, úgy mint például a Monitorozó szolgáltatás, a projekt saját megoldással kíván előrukkolni.

A rendszer jelenleg intenzív fejlesztési és tervezési stádiumban van, úgy másfél – két éven belül várhatóak az első széles körben is használható eredmények.

[6] http://www.gridlab.org/
[7] http://www.cactuscode.org/

6. Grid Undergound MESH rendszer

Bár a fent vázolt két rendszer „csak” két reprezentatív képviselője azoknak a Grid implementációknak, melyek jelenleg fejlesztés alatt állnak, ám ezek igyekszenek lefedni a Grid teljes spektrumát. Mindezek és a többi fejlesztés erőfeszítéseinek ellenére jelenleg nem léteznek jól használható stabilan működő Grid szolgáltatások. Az ez év közepén alakult *Grid Underground*[8] fejlesztői csoport munkájával elsősorban ezt a hiányt igyekszik megszüntetni. Mindemellett egy alternatív Grid infrastruktúra kidolgozására is vállalkozott, melyben mind az OGSA, mind a GridLab projektek eredményeit próbálja egyesíteni.

A rendszer alapvetően egy *MESH*-nek nevezett könyvtárhalmazból áll és az ezek segítségével elkészített szolgáltatás-modulokból, valamint a szolgáltatásokhoz tartozó API-kból.

A rendszer egyik nagy előnye, hogy az absztrakt Grid szolgáltatások mellett bevezeti az absztrakt protokoll valamint az *event* fogalmát is. Ez utóbbi lehetővé teszi a transzparens kommunikációt egy erőforráson belül létező szolgáltatások és a távoli erőforráson lévő szolgáltatások között egyaránt.

6.1. Event Rendszer

Az event-eket a szolgáltatások regisztrálják és iratkoznak fel rájuk. A rendszerben több féle típusú event-et is kibocsáthatnak:

1. *point-to-point*: Egy szolgáltatás adott példánya egy másik szolgáltatás adott példányának küld üzenetet.

2. *multicast*: Egy szolgáltatás példány egy szolgáltatás csoportnak küld üzenetet. A csoport lehet virtuális organizáció, erőforrás, host és szolgáltatás szintű egyaránt.

3. *anycast*: Egy szolgáltatás osztály egy tetszőleges példányának küldünk üzenetet. Tipikusan akkor lehet ilyenre szükség, ha nem érdekel minket, hogy az adott szolgáltatás mely implementációja hajtja végre a kérdéses műveletet.

6.2. Kommunikációs szolgáltatás

A legfontosabb újítás, hogy a kommunikáció lehetőségét a virtuális organizációk tagjai között (alkalmazások, szolgáltatások, felhasználók stb) egy külön speciális Grid szolgáltatásra bízza. Így például a többi Grid szolgáltatás ennek a speciális szolgáltatásnak a segítségével tud kommunikálni a fent említett event-ek segítségével.

A jelenlegi elképzelések szerint a kommunikációs szolgáltatás peer-to-peer hálózaton keresztül juttatja célba az event-eket. Mivel a Grid erőforrások dinamikusan lehetnek jelen a rendszerben (hol vannak, hol nincsenek), továbbá az alkalmazó programok erőforrás igénye is rapszodikusan változhat, így a P2P hálózatnak követnie kell a rendszer dinamikus viselkedését. Itt tehát a legfontosabb kérdések egyike, hogy hogyan lehetséges a peer-ek szomszédsági viszonyait lokális adatokra alapozva úgy megváltoztatni, hogy közben a minimális utat tudjuk biztosítani a csomagok számára.

[8] http://gug.sf.net/

6.3. Egyéb szolgáltatások

A fenti két újszerű szolgáltatás mellett természetesen megjelennek a már klasszikusnak mondható Grid szolgáltatások is. Itt elsősorban a Grid Információs rendszert emelnénk ki, mely egy az eddigiekhez képest teljen új koncepció szerint készült [3].

7. Összefoglalás

Összefoglalva kijelenthetjük, hogy amennyiben a fent bemutatott és egyéb Grid kutatások sikeresek lesznek, úgy a jövő elosztott rendszereit Grid infrastruktúrán képzelhetjük el. Ezt látszik bizonyítani az is, hogy az utóbbi fél évben a legnagyobbak (IBM, SUN, SGI, Microsoft stb.) is felsorakoztak a technológia mellett. Mindebből a hazai Open Source közösség is kiveszi a részét az olyan projekteken keresztül is, mint az itt bemutatott Gug-Mesh. Természetesen még a munka elején járunk és nem beszélhetünk kiforrott technológiákról, de a létező megoldások már elérhetőek a Linux-os közösség számára is.

Hivatkozások

[1] G. Allen, T. Dramlitsch, T. Goodale, G. Lanfermann, T. Radke, E. Seidel, T. Kielmann, K. Vertoep, Z. Balaton, P. Kacsuk, F. Szalai, J. Gehring, A. Keller, A. Streit, L. Matyska, M. Ruda, A. Krenek, H. Frees, H. Knipp, A. Merzky, A. Reinefeld, F. Schintke, B. Ludwiczak, J. Nabrizyski, J. Pukacki, H.-P. Kersken, and M. Russell. Early experiences with egird testbed. *IEEE International Symposium on Cluster Computing and the Grid*, 2001.

[2] Gabrielle Allen, Dave Angulo, Tom Goodale, André Merzky, Jarek Nabrizyki, Juliusz Pukacki, Michael Russel, Thomas Radke, Ed Seidel, John Shalf, and Ian Taylor. Gridlab: Enabling applications on the grid. *Proceedings of HPDC 11*, 2002.

[3] Z. Balaton, G. Gombás, and Zs. Németh. A novell architecture for grid information system. *Proceedings of 2nd IEEE International Symposium on Cluster Computing and Grid, CCGrid*, 2002.

[4] Kelly Devis and Tom Goodale. D1.2 techical specification, 2002.

[5] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann, 1998.

[6] Ian Foster, Carl Kesselman, Jeffery M. Nick, and Steven Tuecke. The physiology of the grid, 2001.

[7] Steven Tuecke, Karl Czakjowski, Ian Foster, Jeffry Ferey, Steve Graham, and Carl Kesselman. Grid service specification, 2002.

Ördög és pokol

(Gondolatok a BSD-ről egy Linux-Konferencia ürügyén)

Zahemszky Gábor
<Gabor at Zahemszky dot HU>

2002. október 17.

Kivonat

Valószínűleg többekben felmerül a kérdés: mit keres itt egy ilyen ide nem illő előadás? Természetesen igazuk van. Rövid választ sajnos nem tudok adni, de remélem az előadás végére a hosszú választ már mindenki maga meg tudja fogalmazni.

Tartalomjegyzék

1. Mi az a BSD?

Hogy mi az a BSD, pláne mik azok a BSD-k, ezt napokig lehetne tárgyalni. Megközelíthető jogi, nyelvészeti, valószínűleg etnográfiai alapon is, mi mégis informatikusi oldalról tesszük. A továbbiakban – nem túl meglepő módon – ez utóbbi következik.

A BSD-k gyakorlati szempontból UNIX-nak tekinthető operációs rendszerek, amelyek többsége a Linuxhoz hasonló funkcionalitással rendelkezik, és fejlesztésük is ahhoz hasonlóan alakult. Hasonlítanak a Linuxhoz pl. abban, hogy többféle BSD létezik, ahogy Linuxból is több (disztribúció) létezik. Hasonlít abban, hogy jó-rossz programozók tucatjai, százai és ezrei fejlesztik, akik döntő többsége nem ezen programok fejlesztéséből tartja fenn magát és családját.

Ezzel szemben a BSD-k komplett rendszert jelentenek, a kernelt és a felhasználói programokat (és nem csak a kernelt) együtt. Talán azt lehetne mondani, olyan komplett rendszerek, mint egy tetszőleges Linux-disztribúció. Csak kicsit kevesebben vannak.

Különbség, hogy a szabad BSD-k esetében általában több ember döntése határozza meg, hogy valamely funkció, kód bekerül-e a hivatalos terjesztésbe. Általában mielőtt ez megtörténne, a programot, programrészletet, esetleg az elvi megvalósítás vázlatát hosszan köpködik az adott terület (ön-) és mások által kijelölt szakértői. Sok esetben egy amúgy kívánatos funkció hivatalos bekerülését pontosan az akadályozza meg, hogy a fölkent szakértő(k) szerint a megvalósítás csak ideiglenes megoldás, melynek rendszerbe építése valójában csak hátráltatja a valóban üdvözítő megoldás előállítását. (v. ö. motiválatlanság. . .)

Egy másik, a felhasználók számára egyébként általában lényegtelennek minősített különbség Linux és BSD-k között a licenc, mely BSD rendszerek esetén általában az ún. BSD-licence, mely szerint mindenki azt csinál a forrással amit akar, pénzért árusíthatja (ha talál olyan őrültet, aki az ingyenesen is elérhető programért fizet), illetve nyugodtan módosíthatja, és nem kell a módosításokat visszaadni a fejlesztőközösségnek. Kb. az egyetlen megszorítás, hogy az eredeti Copyright szöveget benn kell hagyni.

Felmerülhet a kérdés, miért jó ez nekünk. Fejlesztőcégként nyilván kellemesebb dolog olyasmit ellopni, aminek lopása nem törvénybe ütköző (tehát nem is lopás), és nem kísérteni a szerencsét, hogy felismerik-e az eredeti program szerzői, hogy az ő GPL-es kódjukat használtuk fel (volt ilyen a múltban nem is egy, és valószínűleg lesz is még). Otthoni felhasználóként általában mindegy. Céges felhasználóként pedig szintén, hiszen ugyanúgy ingyenes. Ez az oka annak, hogy lényegtelennek minősítettem a Linux egy komoly jellemzőjét.

2. Milyen BSD-k vannak, és mire jók?

Különböző BSD-rendszerből kevesebb van, mint Linux-disztribúcióból, de a precíz, név szerinti felsorolás itt sem túl könnyű. Van egy kereskedelmi termék, a BSD/OS, mely Intel platformon létezik, elsősorban webszervernek és tűzfalaknak ajánlják. De mivel itt szabad szoftverekről van szó, kevesek érdeklődésére tarthat számot – ráadásul én se nagyon ismerem.

A nem kereskedelmi verziók közül az első a FreeBSD nevű, melynek fejlesztésénél a fő cél a PC-platform minél teljesebb támogatása – megtartva a BSD-k régi (lassan 25 éve létező) tulajdonságait – úgyis mint: jó skálázhatóság, kiváló hálózati támogatással, és kiemelkedő stabilitással párosulva. Noha a kezdeti cél az i386 architektúra volt, azóta Alpha platformra, és Sparc64-re készült el, és folyamatban van több más (többek között az Intel, illetve az AMD 64-bites) rendszerre való átültetése. Ez a BSD

ág rendelkezik a legnagyobb felhasználói és fejlesztői körrel, és általánosságban a legteljesebbnek tekinthető PC-s hardver támogatással. Az elmúlt években elismertsége egyre jobban nő, lassan komolyabb cégek is látnak fantáziát abban, hogy termékeik FreeBSD-re is megjelenjenek (Valószínűleg a legismertebb példa az Opera böngésző, melynek szeptember végén jelent meg az első, natív FreeBSD-s változata). Kiváló terep azoknak, akiknek kevés unixos, linuxos gyakorlatuk van, kellőképpen rugalmasak, és nem nagyon vágynak (vagy tudnak) napjaink PC-központúságából kitörni. Használata mindenütt javasolt, ahol a PC – mint hardverarchitektúra – teljesítménye és megbízhatósága elegendő. Kiváló web-, ftp-, és fájlszerver, jól használható routernek, tűzfalnak és webproxynak egyaránt, sőt valószínűleg a legközelebb van ahhoz, hogy elfogadható otthoni, desktop gépet lehessen vele összehozni. Hátulütője egy van, miszerint a legközelebb áll ahhoz, hogy a következő időszak felkapott operációs rendszere legyen azok számára, akik divatból választanak maguknak operációs rendszert, és akiknek a Linux már nem eléggé sikkes.

A sorban a következő a NetBSD, mely a világon a legtöbb különböző hardver architektúrán képes működni – palmtopoktól komoly teljesítményű szerverekig. „Fontolva haladó" fejlesztés, ritkaságszámba megy, ha egy funkciót úgy építenek az alap rendszerbe, hogy az csak egy-két architektúrán használható (ilyen pl. a legutóbbi időben bekerült többprocesszoros támogatás). Használatához kicsit több puritanizmus javasolt, előnyös olyan helyen, ahol a számítógép nem csak a PC-t (esetleg Mac-et) jelenti. Egy, a raktárban/pincében porosodó VAX, régi Sun masina, esetleg egy kiselejtezett HP vagy Digital munkaállomás megtáltosodik, szinte új életre kel ezzel az operációs rendszerrel, és mindenki megelégedésére ellátja egy útválasztó, egy nyomtató-, vagy fájlszerver minden funkcióját (Ezek persze kiragadott példák, hisz ugyanazokat a funkciókat is megemlíthetném, amelyeket feljebb tettem).

A nagyobbak között utolsóként az OpenBSD-t kell megemlíteni. A NetBSD-ből vált ki, és a multiplatform operációs rendszer cél mellé/helyére a biztonság került. Ezen operációs rendszer fejlesztőinek legfőbb célja a legteljesebb biztonság elérése. Ez jelenti egyrészt azt, hogy a fejlesztők folyamatosan a biztonságot növelő beállításokat, javításokat és újításokat építenek rendszerükbe, másrészt azt, hogy a biztonsággal akár csak távolról kapcsolatba hozható hardvernek szinte biztosan legelőször OpenBSD alatt jelenik meg a meghajtóprogramja. Ez, a biztonságot előtérbe helyező szemlélet azt eredményezi, hogy talán legnagyobb számban tűzfalaknak használják ezt a rendszert. Megfontolandó ugyanakkor, hogy egyéb szerver (munkaállomás) feladatok megoldására is ezt válasszuk, mint ahogy az ellenkezője is: NetBSD-ből csináljunk tűzfalat.

Megemlítendők még:

- egy másik kereskedelmi BSD – a MacOS/X alapját képező Darwin – amely, bár külsejével jól rejti, FreeBSD leszármazott; valamint a
- gombamód szaporodó (kicsit linuxos érzést keltő) egyéb szabad verziók (PicoBSD, MicroBSD, SecureBSD, TrustedBSD).

Ez utóbbiak általában az előbb felsorolt három nagy valamelyikének leszármazottai, eddigi tapasztalatok szerint nem túl hosszú élettartammal. A nevekből egyébként látszik, hogy vagy a méretcsökkentés, vagy a nagyobb biztonság elérése (volt) – az általában nem túl nagyszámú – fejlesztők fő célja.

Kell-e egyáltalán ennyi? Miért nem állnak össze a fejlesztők, és hoznak létre egy igazi nagy BSD rendszert? Egy olyat, amely elmegy minden vason, minden PC-s vackot támogat PCMCIA-kártyától WinModemig, és természetesen atombiztos. Nevezzük

el The One Tru(e) BSD-nek, és minden szép lesz! Ez – noha voltak ilyen kezdeményezések, és időről-időre visszatérő téma a különböző flame-, chat- és egyéb levelezőlistákon – reálisan sosem valósul meg (Never say never! – lásd Digital-Compaq-HP-merge). Oka: a fejlesztést nem a pénz és a marketing, hanem a fejlesztésben részt vevő egyének egója és érdeklődési köre irányítja (és csak meglehetősen ritkán engednek az ún. felhasználói nyomásnak).

Ha egyszer azt állítottam, hogy a BSD-k a Linuxhoz hasonló rendszerek, akkor persze kérdéses, hogy érdemes-e velük foglalkozni. Természetesen igen. Azt persze mindenkinek magának kell eldöntenie, hogy milyen szinten, és főleg miért. Lehet ok a valódi szabadság elérése (itt arra gondolok, hogy bizony a BSDL több mindent engedélyez, mint a GPL). Nyilván az ritkaságszámba megy, hogy ez legyen a fő szempont, de elképzelhető. Lehet az ok egyszerűen az, hogy a környezetünkben ezt látjuk (bár ez egyelőre nem tekinthető általános jelenségnek). Az is lehet, hogy egyszerűen nem akarunk átlagfelhasználók lenni (vagy legalábbis annak látszani), és ha egyszer mindenki vagy Windowst, vagy Linuxot használ, akkor valamely BSD használata elegendő ok ahhoz, hogy a szomszéd szobában dolgozó, hosszú hajú gyönyörűség felfigyeljen ránk (Sajnos ez utóbbi reális lehetősége csekély, meglehetősen kevés bombázó mutat élénk érdeklődést a számítástechnika (ilyen) mélységei iránt. Már az is kész csoda, ha egyáltalán tud a Windowson kívül egyéb operációs rendszerről, sőt már az is, hogy ismeri és tudja is, mit jelent az operációs rendszer).

Végül, de nem utolsósorban használhat valaki azért BSD-t, mert egy vagy több, számára szavahihető ember azt mondja neki: „az jó”. Kipróbálja, és kiderül: tényleg.

3. Mit kapok egy BSD rendszerrel?

Mindent. Mint említettem, a BSD-k teljes rendszerek. Az elérhető BSD-k mindegyike tartalmazza azokat a funkciókat (de legalábbis többségét), melyek egy normálisnak tekinthető számítógépes rendszertől elvárhatók. Természetszerűleg vannak szövegszerkesztők, programfejlesztő eszközök, apróbb-nagyobb játékocskák. Megtalálhatók a szerver funkciók ellátásához szükséges programok, akár levelező-, nyomtató-, web- vagy fájlszerver felállítása a cél. Nem túl extra (vagy gagyi) hangkeltő hardverek kivételével – legalább az alap – zenefunkciók elérhetők, azaz meglevő mp3, ogg, wav, midi és mod fájljainkat nem kell eldobni, meghallgathatók. Azokon a hardvereken, ahol ez egyáltalán fizikailag lehetséges, elérhető a UNIX/Linux rendszereken megszokott grafikus alrendszer, az X Window System, és természetesen az ezekre kifejlesztett különböző desktop felületek (GNOME, KDE, IceWM, ROX, stb.). És ha egyszer van GNOME és KDE, akkor természetesen a legtöbb Linuxra (illetve ezen felületek valamelyikére) elkészített program szintén elérhető.

A szoftverek többsége bináris formában elérhető, a telepítő CD-készleten vagy valahol a hálózaton megtalálható. Ezek általában tar.gz ill. tar.bz2 formájú csomagok – BSD-s elnevezésük: package. Természetesen vannak programok, melyek hiába ingyenesek, nem engedélyezett CD-ken, vagy esetleg bináris formátumban való terjesztése. Ezen szoftverek BSD-khez illesztésére BSD rendszereken egy fura, portnak (NetBSD-n pkgsrc-nek) nevezett módszert használunk. Ez annyit tesz, hogy valaki, akinek az adott program feltűnt és BSD-n használni szeretné (jobb esetben maga a szerző), elkészít egy picike katalógus-struktúrát, ebben különböző leírófájlokat (esetleg az adott rendszerben fordításhoz szükséges javításokat) helyez el. Az adott program telepítése - amennyiben előre fordított csomag nincsen -, annyit igényel, hogy a felhasználó belép az adott könyvtárba, itt elindítja a (minden BSD-ben alapban létező) „make” parancsot,

és vár. A továbbiakban letöltésre kerül a szükséges forrás, a rendszer patch-eli, konfigurálja, majd lefordítja azt. Amikor ez kész, rendszeradminisztrátorként egy „make install" – és a program feltelepült. Mielőtt mindenki visszariadna: a CD-ken több ezer csomag bináris, azonnal telepíthető formában elérhető, azaz nem az a jellemző, hogy BSD-n mindent forrásból telepítünk (Hozzá kell azonban tenni, ilyenre is van példa, hisz a bináris csomagok frissítése lassan halad, ráadásul egyáltalán nem biztos, hogy a csomag előállítója ugyanazokkal a fordítási beállításokkal dolgozik, amivel mi szeretnénk).

A port-fa frissítése Internet kapcsolat esetén viszonylag könnyen megtehető, így akár naponta, hetente aktualizálhatjuk rendszerünket.

Hasonlóan a kiegészítő szoftverekhez, maga az operációs rendszer is folyamatosan frissül – viszont ellentétben a Linux-szal, itt sem bináris csomagok, hanem a forrás az, amely elérhető. (Félreértések elkerülése végett megjegyzem, tudom, hogy a Linux-kernel forrásban gyűjtendő be, meg azt is, hogy X, Y és Z alrendszert az ember forrásban gyűjti be és úgy fordítja, ennek ellenére azt hiszem, a felhasználók többsége amikor megjelenik a GNU tar legutolsó ismert biztonsági hibáját javító csomag, szó nélkül szedi le a régi csomagot, és telepíti az újat – természetesen az elérhető bináris formából, még akkor is, ha forrásban is elérhető lenne.)

Ezzel kapcsolatban egy rövid idézet: „Először majd elsírtam magam, mert annyira fapados volt. Aztán rászoktam a CVS lista olvasására, a cvsup, make buildworld, make buildkernel, portupgrade használatára és azóta tudom, hogy kell-e, érdemes-e frissíteni valamit és nagyjából előre tudom, hogyha frissítek, akkor mire kell figyelnem." (Az idézet Nagy Attila (Bra) egy leveléből való, és a sokak által ismert ftp.fsn.hu rendszer Debianról FreeBSD-re való átállásának szubjektív leírása. A rendszer 2.2-es kernelt használó Debian volt, az átállás valamikor 2001 februárja és áprilisa között zajlott le.)

A Linux-közösség számottevő része úgy tekint a Linuxra, mint valami mondabeli hősre, vagy a mesék legkisebb fiújára. A zseniális finn egyetemista agyában megszületett, hobbinak indult szoftver, az Interneten élő-haló egyszerű emberek összefogásával és segítségével felkel, és lerázza a számítógépet használókra erőszakolt láncot – a Microsoftnak a szoftverpiacon érezhető uralmát. A M$ gonosz, termékei használhatatlanok, és az ezeket a rendszereket – önkéntesen – használó felhasználók csökött agyú hülyék, akiknek nemhogy számítógépet, de valószínűleg piros üveggolyót sem lenne szabad a kezükbe adni, hisz azt is elrontják.

Ezzel szemben a BSD-felhasználók – engem kivéve – mind komoly emberek, akik tudják, hogy a gonosz uralmát megszüntetni nem lehet, sőt nem is kell – annak is megvan a maga szerepe (Például nagyon sok ember számára kézzelfogható közelségbe került a computer – és ez valahol a DOS/Windows érdeme is). Próbálkozni lehet és kell is – így jönnek létre az egyre jobb, ténylegesen felhasználóbarát szoftverek – és ez majdnem mindenkinek jó.

Azok, akik úgy érzik, ez nem a létező világok legjobbika, és keresik a lehetőséget a jobbításra, rengeteg lehetőségük van. Ezek egyike a Linux, de valahol a listában ott szerepel a BSD is.

Az előadás címe egyfajta összehasonlítást sejtet. Nem célom elpanaszolni azokat a dolgokat, amelyek számomra kényelmetlenek, esetleg egyenesen visszataszítóak bizonyos operációs rendszerekben. Ugyanígy nem cél mindenkit meggyőzni arról, hogy a BSD-k jobbak. Akinek kell, úgyis tudja. A többieknek pedig csak azt szeretném mondani, próbálják ki.

4. Hivatkozások

http://www.bsd.hu/
http://www.freebsd.org/
http://www.netbsd.org/
http://www.openbsd.org/

Sun Microsystems – elkötelezetten a nyílt forráskódú közösségek iránt

Fischer Erik <erik.fischer@sun.com>

2002. október 21.

A peremhálózati számítástechnika kulcsa a horizontális méretezhetőség. A folyamatok gyors átfutása érdekében az infrastruktúra különböző pontjain kellően nagyszámú szervert szükséges munkába állítani a legjobb hatásfok eléréséhez. Ugyanakkor a menedzsmentnek maximalizálnia kell a beruházás megtérülését.

A Sun tökéletes megoldást talált. A Sun UNIX rendszerét, hardverét és szolgáltatási szakértelmét kitűnően hasznosító Sun LX50 szerver felbecsülhetetlen segítséget nyújt a vállalatok peremhálózatai számára. Egyedülálló előnyei közé tartoznak:

- Kiváló ár-teljesítmény arány. Kedvező ára féken tartja a költségeket, a két 1,4 GHz-es Intel Pentium III processzor pedig biztosítja a kellő teljesítményt. A szerver rackbe szerelhető, kompakt 1U kialakításának köszönhetően az alapterület-kihasználtság javításával is csökkenthetők a költségek.
- Egyszerűsített felügyelet. Az integrált Sun Control Station felügyeleti eszköz segítségével egyetlen képernyőről is biztonságosan kezelhetők a Linux szerverek, és áramvonalassá tehetők az olyan feladatok, mint a szoftver- vagy javítócsomag-elosztás, a rendszerfelügyelet és a leltárnyilvántartás.
- Fokozott megbízhatóságú Sun Linux. A Sun sokéves UNIX-piaci vezető szerepének köszönhetően rendkívül stabil, tökéletesen a hardverplatformra hangolt Linux rendszert tud kínálni.
- Vezető szolgáltatási platform. A Sun iparágvezető szolgáltatási platformot nyújt a hagyományos operációs rendszer jellegű működést, az alkalmazásszolgáltatásokat és a személyazonosság-kezelést ötvöző Solaris operációs rendszert futtató Sun LX50 szerverrel.
- Előkonfigurált, előtesztelt alkalmazások. A Sun LX50 szervert nagyszámú előre konfigurált alkalmazással együtt szállítjuk, így a vásárlók gyorsabban állíthatják üzembe a kívánt megoldásokat.
- A Sun Linux rendszerű Sun LX50 szervernek részét képezik a Sun ONE alkalmazások és fejlesztői eszközök, a Java 2 Platform, Standard Edition (J2SE platform), a Sun Chili!Soft ASP plug-in webszerver szoftver, a Sun Grid Engine szoftver, valamint a mySQL adatbázis.
- A Solaris operációs rendszert futtató Sun LX50 szervernek része a J2SE platform, a SunScreen tűzfalszoftver, a Sun Grid Engine szoftver, a mySQL adatbázis és az Apache webszerverszoftver. A Solaris operációs rendszer emellett a független szoftverszállítók által támogatott termékek legnagyobb csomagjainak egyikét kínálja.

Mivel mind a hardvert mind a Sun Linux operációs rendszert a Sun fejlesztette ki és a Sun támogatja, így gyors, integrált karbantartást tud biztosítani a Sun LX50 szerverek számára. Egyetlen telefonhívással elintézhető az összes probléma, nem irányítják minduntalan újabb és újabb helyekre. Ráadásul a szerver hardver- és szoftver-összetevőinek előre konfigurálásával és előtesztelésével annak az esélyét is minimálisra csökkentettük, hogy egyáltalán telefonálnia kelljen.

Az LX50 előnyei a peremhálózati megoldásoknál

A Sun Linuxszal szembeni nagymértékű elkötelezettsége mellett a vásárlóknak nyújtott beruházásvédelmet is fokozza, és szélesre tárja kapuit a fejlesztők előtt. A Sun most a UNIX-on, a Linuxon, a Java technológián és az XML-en alapuló termékek teljes, integrált sorát kínálja az operációs rendszer, az eszközök és az alkalmazott szoftverek szintjén, soha nem látott rugalmasságot biztosítva így a vállalatoknak. Azok a vásárlók, akik már beruháztak a Solaris operációs rendszerbe, folytathatják annak üzembe állítását a Sun LX50 szerveren, s a Sun vezető UNIX platformján rendelkezésre álló szoftvermegoldások ezreiből válogathatnak. A Sun LX50 szerveren emellett olyan vezető szállítóktól származó Linux alkalmazások széles köre is üzembe állítható, mint az Oracle, a BEA Systems, a Veritas vagy a Check Point.

(x)

MagyarOffice vagy OpenOffice.org?

Koleszár Kázmér

2002. október 21.

Tartalomjegyzék

1. Bevezetés

Míg a Linux az otthoni számítógépek területén eredményesen terjed, az oktatási szférában pedig máris komoly részesedést ért el, sokáig nem tudott megjelenni a munkahelyi számítógépek piacán. Ennek több oka közül az egyik leggyakrabban emlegetett a megfelelő irodai alkalmazáscsomag hiánya volt. Ezzel persze lehetett vitatkozni, hogy szükség van-e egyáltalán egy Microsoft Office típusú *„What You See Is All What You Get"* jellegű programcsomagra, nem nyújtanak-e sokkal nagyobb rugalmasságot és igazi alkotói szabadságot a Unix világban elterjedt megoldások. Bebizonyosodott azonban az, hogy a felhasználók roppant tömegei semmiféle vonzódást nem éreznek a Unix eszközök iránt, hiszen számítógépes ismereteiket a Windows világban szerezték, ennek a megoldásait fogadták el, mint egyedül üdvözítő utat.

Ezzel a helyzettel próbáltak megbirkózni a KDE, a GNOME és egyéb hasonló projektek, amikor intuitív módon kezelhető grafikus felületük mellé az irodai alkalmazások kifejlesztését is célul tűzték ki. Úgy tűnik azonban, hogy mégsem ezek a klasszikus szabad szoftver kezdeményezések járnak először sikerrel. Az amerikai Sun Microsystems bábáskodásával létrejött OpenOffice.org projekt, és annak kereskedelmi változata, a StarOffice jelenleg a Microsoft után következő leginkább elterjedt alkalmazáscsomag.

2. OpenOffice.org, StarOffice

A gyors sikerben talán a vegyes fejlesztési modell lehetett a döntő, hiszen nem kellett az egész szoftvert, a maga 8 millió sorával a semmiből létrehozni: a Sun egyszerűen megvásárolta a német fejlesztésű, Németországban jelentős piaci részesedéssel is rendelkező StarOffice programot, a fejlesztőgárda nagy részét is beleértve. A program további fejlesztése is vegyes stratégia szerint működik. Egyrészt a nyílt kódú fejlesztést projektvezetőkként általában Sun alkalmazottak tartják irányban, valamint a Sun rendszeresen készít az OpenOffice.org-ra épülő kereskedelmi verziókat is, legutóbb a StarOffice 6.0-t. Ezt a jogot ráadásul nem tartja meg magának: az OpenOffice.org ugyanis kettős licenc alatt áll. Ez azt jelenti, hogy a felhasználó szabadon megválaszthatja, hogy a klasszikus GPL vagy pedig az ún. Sun Industry Standard Source Licence (SISSL) rendelkezéseihez igazodik. Ha az SISSL licencet választja, akkor lehetősége van a forráskód továbbfejlesztésére, az abból fordított bináris változat önálló forgalmazására is, egy kikötéssel. A Sun által megszabott szabványokhoz ragaszkodni kell, ami az XML fájlformátumot, és az OpenOffice.org API-t jelenti. Ezeket megváltoztatni a leszármazottakban is csak úgy lehet, ha ahhoz teljes körű és nyilvános dokumentációt is csatol a továbbfejlesztő. Ezzel a kitétellel kerülhető el, hogy megjelenjen például egy „Kicsi Puha StarOffice", az eredetihez hasonló funkcionalitással, de azzal nem kompatibilis, titkos fájlformátumokkal…

3. A MagyarOffice

A MultiRáció Kft. MagyarOffice nevű termékének létrehozását is a Sun SISSL licenc tette lehetővé. Tehát jogilag rendben van a dolog, de miből gondoljuk, hogy a szűk magyar piacon létjogosultsága van egy ilyen, pénzért vásárolható szoftvernek az immár magyar fordításban is elérhető, ingyenes OpenOffice.org mellett?

Először is a piac különböző szegmenseit célozzák meg. Míg az OpenOffice.org

inkább a magán-felhasználók körében lehet népszerű, a MagyarOffice-t elsősorban a vállakozások számára készítettük, azzal a nem titkolt céllal, hogy a Microsoft termékekhez szokott, nem számítástechnikus felhasználók számára tegyen lehetővé minél zökkenőmentesebb áttérést.

Ennek az elsődleges célközönségnek az igényeit vettük figyelembe a MagyarOffice többletszolgáltatásainak, hozzáfejlesztéseinek kiválasztásában. Termékünk a magyar nyelvű OpenOffice.org-hoz képest a következő többletszolgáltatásokat nyújtja:

- magyar nyelvű súgó
- felhasználói kézikönyv
- a legújabb MorphoLogic nyelvi modulok (helyesírás, elválasztás)
- magyar nyelvű és hazai formátumú sablonok
- Magyarország térkép diagram
- többváltozós optimalizáló modul

Ezen kívül a vállalati vevők igénylik a személyes ügyfélszolgálatot és a garantált terméktámogatást is, amit a termék ára szintén magában foglal. Úgy gondoljuk, és az eddigi piaci tapasztalataink ezt alá is támasztják, hogy a vevőkör jelentős része ezeket a hozzáadott értékeket méltányolja, és nem tartja túlzottnak az ezekért kért árat, amely egyébként így is csak töredéke a piacvezető termék árának.

4. A MagyarOffice és a Linux

Azt hiszem nem okozok meglepetést, ha elmondom, hogy a MagyarOffice felhasználóinak több mint 95%-a a MagyarOffice Windows-os változatát használja. Hogyan és mennyiben járulunk akkor hozzá a Linux rendszerek térhódításához?

A MagyarOffice kifejezett célja, hogy lehetővé tegye a teljes megoldást az irodai használatban, ezért található a CD-n:

- egy MagyarOffice-ra hangolt *MinimálLinux disztribúció*,
- ezért csomagoljuk hozzá a *Mozilla* webböngészőt és levelezőt,
- a Professzionális változatnál ezenfelül a *MySQL* adatbázis szervert is.

Tehát ha a felhasználó vesz egy üres számítógépet, hozzá megveszi a MagyarOffice CD-t, akkor egy működőképes, teljes funkcionalitású irodai számítógépet kap a pénzéért, semmilyen más szoftverre nincs szüksége. A Linux mellett döntő felhasználóinknak mindemellett valamelyik „nagy” Linux disztribúció telepítését javasoljuk, ha komolyan gondolja, de első próbálkozásként megvan a lehetősége a CD-n megtalálható MinimálLinux telepítésére is.

Mivel cégünk is arra számít, hogy a nem túl távoli jövőben a Linux rohamos terjedése várható a munkahelyi számítógépeken is, kifejezetten ragaszkodtunk a MagyarOffice Linux-os és Windows-os változatának teljes funkcionális azonosságához (ez nem volt mindig egyszerű...). Így aki első lépésként áttért a MagyarOffice Windows-os változatára, annak a Linux-os változat sem fog meglepetést okozni, valamint a vegyes Linux/Windows környezetekben is könnyebb így a munka.

(x)

Az *Oracle – Linux* kapcsolat (kivonat)

Markovits Péter

2002. október 21.

Kivonat

A Linux már elérte a vállalati üzleti felhasználhatóság színvonalát. Kevesebb cég, de nagyobb szakértelemmel, vállalati szinten is elvárható támogatással áll a piacon elterjedt Linux disztribúciók mögött. Az Oracle az elsők között ismerte fel, hogy a Linux-szal érdemes az ő nagyvállalati piacán és termékkörében is foglalkozni. Az első szélesebb körben elérhető Oracle on Linux (8.0.5) már közel négy éve jelent meg. Megjelentek és teljes körű felhasználhatóságot kívánnak továbbá az Oracle Application Server széles körű komponens halmaza és maga az Oracle E-Business Suite is, ami a vállalatirányítási rendszerek teljes skáláját biztosítja.

A Real Application Clusters megjelenésével az Oracle szakított azzal a kizárólagos koncepcióval, hogy a komoly nagyvállalati rendszerek alá csak nagy, monolit hardvert szabad értékesíteni. A RAC szoftver architektúrája lehetővé teszi, hogy egyszerűbb, olcsóbb sztenderd számítógépekből (tipikusan akár PC szerverekből) is lehessen nagy összteljesítményű, hibatűrő szerver hardvert építeni fürtözéssel (klaszter) és ezt jól ki is lehessen használni. A Linux pedig egyik operációs rendszer alternatívája ezeknek a PC alapú szerver környezeteknek.

E forradalmian új fürtözési technológiának bemutatása az előadás leghangsúlyosabb mondanivalója.

(x)