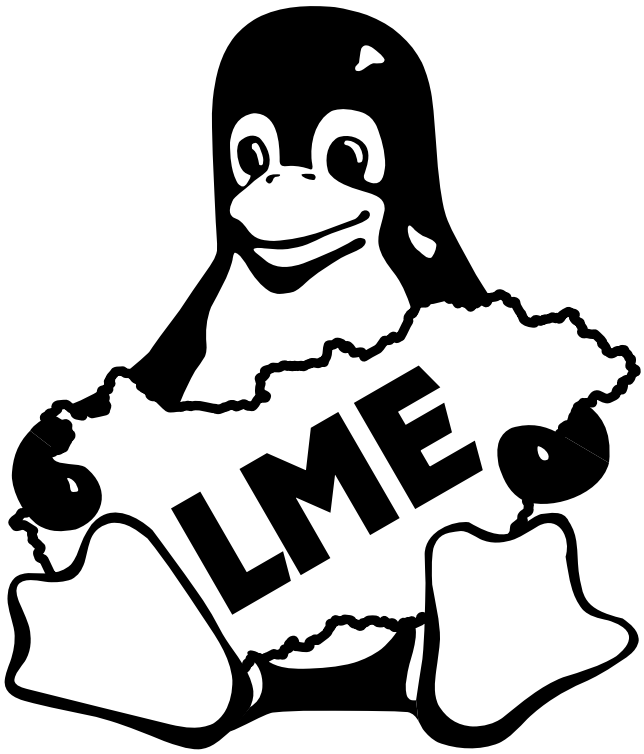
V. GNU/Linux Szakmai Konferencia



Budapest, 2003. november 8 .

A kiadvány tördelése a TEX 3.14159 verziójával készült,
GNU/Linux operációs rendszeren.
Helyesírás-ellenőrzés: Hunspell 0.9.6.2 verzió
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*

Minden jog fenntartva. Jelen kiadvány elektronikus verziója módosítás nélkül szabadon terjeszthető. A nyomtatott változat terjesztése, másolása, informatikai rendszerben történő további feldolgozása, tárolása csak a szerzők írásos hozzájárulásával lehetséges.

Tartalomjegyzék

A konferencia támogatói

Fő támogató:

Konzumbank Rt.

Fő médiatámogató:

Computerworld Számítástechnika

Médiatámogató: Prim Online

Arany fokozatú támogatók:

- IBM Hungary
- Novell Magyarország
- Oracle Hungary Kft.
- Sun Microsystem Kft.

Ezüst fokozatú támogatók:

- Linuxvilág
- Linux Support Center Kft.
- Matáv
- ULX Kft.

Bronz fokozatú támogatók:

- Balabit IT Biztonságtechnikai Kft.
- Direkt Kft.
- Lafisoft
- Mission Critical Linux Kft.
- Software Station

Előszó

Kedves vendégünk, tisztelt Olvasók!

Üdvözöljük a Linux-felhasználók Magyarországi Egyesületének immár V. alkalommal megrendezett szakmai konferenciájának alkalmából.

Egyesületünk 1998 őszén azzal a fő céllal alakult, hogy összefogja a Linuxszal foglalkozó szakembereket és cégeket, szakmai fórumokat teremtsen, terjessze a Linuxszal kapcsolatos ismereteket, szükség esetén jogi személyiségként képviselje a „Linux-hívők" hazai társadalmát.

Öt év a számítástechnika világában hihetetlenül hosszú idő, de talán egy egyesület életében éppen elég, hogy gyermekbetegségeit kinője, az ifjonti hév keveredjen a felnőtté válás komolyságával.

Hosszú volt az út, mely az első 5-8 fős beszélgetésektől az egyesület megalakulásáig vezetett, és még hosszabb, amíg eljutottunk odáig, hogy szakmai konferenciáinknak országos elismertséget szereztünk.

Az eltelt idő alatt több komoly eredményt értünk el. Egyesületünk taglétszáma ha nem is drasztikus mértékben, de folyamatosan nő, az őszi Konferencia időszaka mindig kiugró csúcsként jelentkezik az ezzel kapcsolatos kimutatásokban.

Idén a még alakulóban lévő Nemzeti Szabad Szoftver Stratégia előkészítésében való aktív szerepünk talán leglátványosabb eredményünk.

Érdekképviseletünk a Szoftver Szabadalmakkal Kapcsolatos direktívatervezet miatt első alkalommal komoly formában öltött testet.

Az e téren kifejtett komoly ismeretterjesztésünk kifejezetten sikeres volt. Az aktivistáink munkája, valamint a civil személyek/szervezetek közreműködése megfelelő mértékben fordította a problémára a figyelmet.

Egyesületünk szinte minden fórumon igyekszik megjelenni, ahol lehetősége nyílik a szabad szoftverek népszerűsítésére. Többek között részt vettünk az Ózon fesztiválon, a Sziget fesztiválon, a Civiliáda rendezvényen.

Egyesületünk minden évben kiemelt feladatának tekinti a GNU/Linux szakmai konferencia megrendezését, mely a tavalyihoz hasonlóan színvonalas környezetben került idén is megrendezésre.

Reményeink szerint a Konferencián öt időpontban összesen huszonöt előadás kerül lebonyolításra. Első alkalommal sikerült azt megoldanunk, hogy minden, a Kiadványban szereplő előadást meg is tudunk tartani.

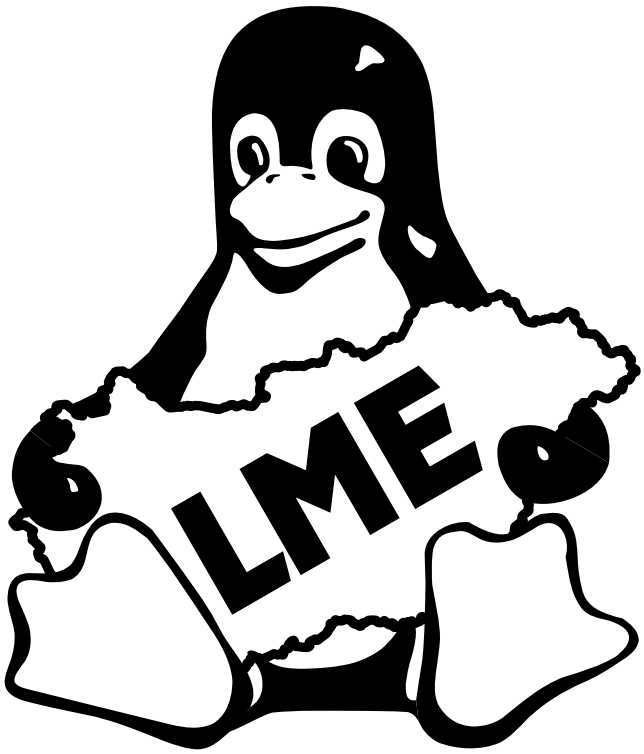
Ez évben szinte minden olyan cég képviselte magát Támogatóként, előadóként, aki valamilyen formában már zászlajára tűzte a Linux támogatását. Bízunk benne, hogy az előadások, és a kiállítók kínálata találkozik vendégeink érdeklődésével.

Észrevételeit, javaslatait bátran ossza meg velünk a Konferencián személyesen az LME standján, vagy ezt követően elektronikus levélben az *info@lme.linux.hu* címen.
E rövid bevezető lezárásaként azt kívánjuk, hogy forgassák minél többet kiadványunkat, és látogassanak el 2004-ben a következő GNU/Linux Szakmai Konferenciára!

Tisztelettel,

A konferencia szervezői

Előadások



Megjelenés-független dokumentáció készítése szabad szoftverekkel

Baráth Gábor

2003. október 27.

Kivonat

- A dokumentumkódolási alapjai, SGML és XML összehasonlítása.
- Szerkesztőprogramok (emacs, jedit, OpenOffice.org), docbook, TEI2, docbook website DTD-k bemutatása.
- Megjelenés-vezérlési alapok, DSSSL és XSL bemutatása, parser programok áttekintése (jade, openjade, saxon, xalan).
- Különböző kimentetek előállítási lehetőségei a gyakorlatban: segédprogramok (sgml2x, docbook2x, saját makefile készítése) nyomtatáshoz használt kimenetek (pdf, ps, dvi), elektronikus kimenetek készítése (html, különböző e-book formátumok).
- Nehézségek: képek elkészítése, kimeneti formátumok enkódolása, stb.

Tartalomjegyzék

1. Bevezető

Jónéhány, dokumentumokkal kapcsolatos feladat esetén, amennyiben:

- nagy mennyiségű dokumentum egységes formátumban való elkészítése,
- egy dokumentum ugyanazon a médián különböző megjelenésben való publikálása,
- egy dokumentum különböző médián vagy formátumban való publikálása

a cél, érdemes elgondolkodni újrahasznosítható dokumentumok készítésén.

Mindegyik feladat ellátásához alapvetően szükséges, hogy a dokumentumot két részre válasszuk szét. Az egyik résznek a dokumentum szerkezetét kell leírnia, hogyan tagozódik részekre, fejezetekre, stb., míg a másik résznek ennek a struktúrának és a megjelenítő eszköznek az ismeretében elő kell tudnia állítania a megfelelő kimenetet.

Az így szétválasztott dokumentumot aztán a tartalom változtatása nélkül lehet különböző megjelenítő eszközökre publikálni vagy fordítva, a dokumentum változásakor – mint például új fejezet beszúrása – a megjelenítő réteg változtatása nélkül lehet újra publikálni.

Sőt, mivel a megjelenítő réteg a lehetséges struktúra alapján működik, így a szerkezet betartásával készült dokumentumok ugyanazzal a megjelenítő réteggel működhetnek.

Ezt a gyakorlatban SGML vagy XML kódolással és a hozzájuk tartozó transzformációkkal lehet a legegyszerűbben és ami igen fontos, a szabványok betartásával elérni.

2. Dokumentumkódolási alapok

2.1. SGML és XML összehasonlítása

Már a 60-as években felmerült egy általános dokumentum-jelölő nyelv kidolgozásának szükségessége, de az első ilyen szabványosított nyelv csak 1986-ban készült el SGML (Standard Generalized Markup Language) néven. Az SGML egy nagyon átgondolt, széleskörűen használható szabvány, amit az is jól mutat, hogy az első tíz évben gyakorlatilag semmit nem változtattak rajta. Viszonylagos bonyolultsága miatt azonban kevés szoftver készült hozzá és ezek többsége is a teljes szabványnak csak kis részhalmazát támogatta.

1998-ban a W3C elfogadta szabványnak az XML (eXtensible Markup Language) 1.0-t amelyet az SGML és HTML jó tulajdonságainak figyelembevételével alakítottak ki. Az SGML-hez hasonlóan DTD (Document Type Definition) alapú, önleíró, tetszőlegesen kiterjeszthető nyelv. Alapjában véve az XML megfeleltethető az SGML egy részhalmazának, tehát minden XML dokumentum SGML dokumentum is egyben. Ellenben az SGML dokumentumok XML-lé alakítása egyáltalán nem triviális feladat, sőt bizonyos esetekben nem is lehetséges.

2.2. Dokumentum típusok (DTD)

Minden esetben, amikor egy dokumentumot készítünk, először meg kell határozni a dokumentum típusát, tehát el kell készíteni a DTD-t. Szerencsére nem kell mindent elölről kezdeni, hiszen jó néhány dokumentumfajtához találhatunk már elkészített DTD-ket.

Amennyiben szoftverdokumentációt készítünk, az egyik legalkalmasabb választás a docbook DTD [1]. Ebben készítik többek között a GNOME desktop teljes dokumentációját is. Nem elhanyagolható szempont, hogy rendkívül jól konfigurálható, sok kimenet előállítására alkalmas stíluslap-csomag is rendelkezésre áll hozzá, melynek segítségével számos formátumú végdokumentum készíthető postscripttől a javahelpig.

Szépirodalmi művek kódolásához használható a TEI2 (Text Encoding Initiatíve) DTD [2], mely a verseskönyvtől a színdarabig mindenféle műre alkalmazható. Sajnos a stíluslap csomag egyelőre elég kezdetleges, így a megjelenések testre szabása nagyobb energiaráfordítást igényel mint a docbook esetén.

Létezik egy docbook alapú, kifejezetten weboldalak leírására tervezett DTD, melynek docbook-website a neve. Ennek segítségével weboldalunk tartalmát az ismert docbook DTD szerint írhatjuk le, kiegészítve néhány, az oldalak struktúrájára és a weboldalra vonatkozó információval.

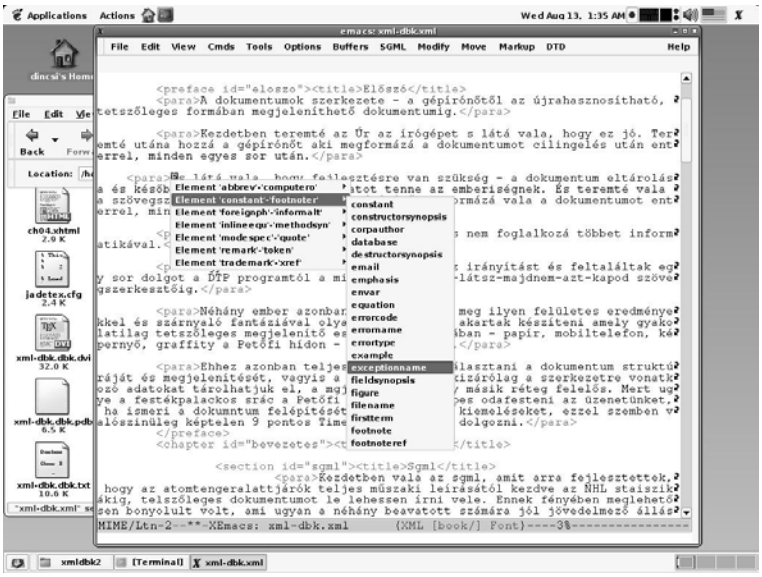
Szót kell ejtenünk még egy DTD-ről, amelyet kifejezetten a dokumentum formázásának – tehát nem a struktúrának, hanem a megjelenésnek – leírására fejlesztettek ki. Ez a dokumentum fajta az xsl formating objects (xsl:fo). Mi kizárólag köztes formátumként fogjuk használni, hiszen a bevezetőben már említettük a struktúra és a megjelenés szétválasztásának fontosságát, amely az xsl:fo esetén nincs meg.

Végezetül az érdekesség kedvéért megemlítjük, hogy természetesen található DTD a weboldalak leírásáért felelős HTML 3.2 és HTML 4.0 szabványhoz, illetve a HTML xml-kompatibilis verziójához, az XHTML-hez is.

2.3. Szerkesztőprogramok

Emacs és Xemacs

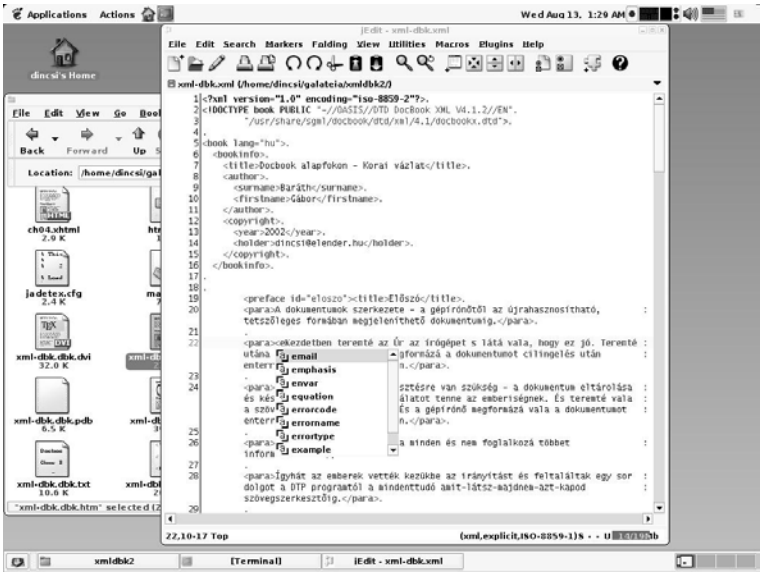
Az emacs és a psgml kiegészítő csomag segítségével komplett SGML szerkesztő programot kapunk gyakorlatilag az összes Linux terjesztéssel, amely természetesen XML szerkesztésre is kiválóan használható. A program képes szintaxis kiemelésre, DTD alapján a lehetséges elemek beszúrására, SGML és XML érvényesség ellenőrzésre, tagok elrejtésére és kibontására.



További előnyt jelenthet bizonyos helyzetekben, hogy akár konzolon is képes futni, így akár távoli elérés esetén, grafikus környezet nélkül is használható.

jEdit

A jEdit [3] egy nagy tudású, java alapú programozói szövegszerkesztő, melyhez letölthető többek között XML bedolgozó modul is. Ennek a modulnak a segítségével a jEdit átalakul egy igazi HTML és XML szerkesztővé, mely mindent tud amit az emacs-nál említettünk, ezen felül képes a tagok struktúráját faszerkezetben megjeleníteni.



Hátránya, hogy a java virtuális gép miatt viszonylag nagy erőforrásigénye van, így lassabb gépen nem igazán használható.

OpenOffice.org

Az OpenOffice.org [4] szövegszerkesztője saját DTD alapján XML-ben tárolja a dokumentumot, ezért adott a lehetőség, hogy megfelelő transzformációval a saját formátumát tetszőleges más DTD-nek megfelelő dokumentummá alakítsuk. Már az 1.0 változat támogatta a dokumentumok ilyen módon történő betöltését és kimentését, és az 1.1 verzióban már fejlesztett sdocbook (Simplified Docbook) támogatás található – de bárki készíthet a saját DTD-nek megfelelő transzformációkat hozzá.

3. Megjelenés-vezérlési alapok

3.1. DSSSL és XSL bemutatása

A dokumentumok megjelenését az úgynevezett stíluslapok segítségével vezérelhetjük egy transzformációnak nevezett eljárás során. Alapvetően kétféle stíluslap szabvány létezik, az SGML-hez kidolgozott DSSSL és az XML fájlok transzformációját végző XSLT.

A DSSSL nyelv és a hozzá tartozó elemző (parser) programok segítségével mind SGML mind XML fájlok transzformációja megoldható, az XSLT kizárólag XML fájlok transzformációjára használható.

Mindkét transzformációs eljárás esetén lehetőség van stíluslapok importálására, melynek segítségével a transzformáció egyes eljárásait felüldefiniálhatjuk. A gyakorlatban ezt úgy használjuk, hogy az általános stíluslapokhoz, melyek az alap DTD-khez letölthetők, saját vezérlő stíluslapot készítünk, melyben beállíthatunk paramétereket, vagy saját képünkre formálhatjuk a transzformáció bármely részét. Ugyanezzel a módszerrel készíthetünk különböző nyelvű stíluslap-változatokat.

3.2. Parser programok áttekintése

A DSSSL stíluslapokhoz két értelmező érhető el: a James Clark által írt jade [5] és a jade alapján készített openjade [6] értelmező. A jade sebességben, az openjade tudásban múlja felül társát. Ha például nem akarunk kétoldalas nyomatot, vagy könyvjelzővel ellátott pdf-et készíteni, használhatjuk a jade-t, ellenkező esetben marad az openjade. Fontos megjegyezni, hogy az openjade 1.4-es verziója egyelőre rengeteg hibát tartalmaz, ráadásul rendkívül lassú, ezért inkább a stabil 1.3 verzió használata javasolt.

XSLT vezérlőprogramból már lényegesen nagyobb választék áll rendelkezésre, kedvünkre válogathatunk a nagyszámú különböző nyelven írt program közül. Sebességben a C vagy C++ nyelven implementált programok viszik el a pálmát, ilyen az expat [7], a sablotron [8] vagy a xalan [9] régi, C++ verziója. Ezek azonban többségükben hiányosságokat mutatnak, például a különböző kódlapok kezelése illetve a néhány extra szolgáltatás (kimenet több fájlba írása, stb.) terén.

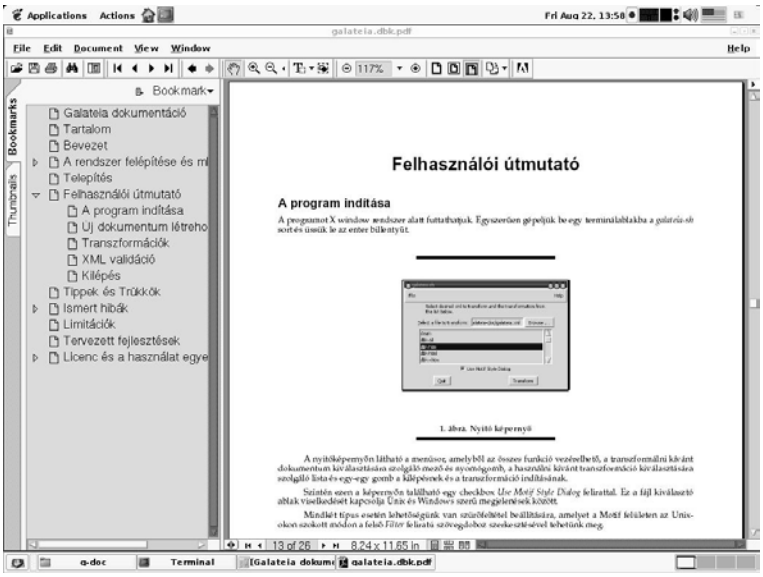
A java alapú szoftverek közül két kitűnő, gyakorlatilag hibátlan, sok extra szolgáltatást nyújtó implementáció létezik: a saxon [10] és a xalan [11] java-s verziója. Bizonyos stíluslap-csomagok ezekhez a transzformátorokhoz készített kiegészítőket tartalmaznak, melyekkel több lehetőséget tudunk kihasználni a transzformáció során. Ezért, amennyiben nem sebesség kritikus a rendszer, mindenképpen ezek valamelyikét ajánlott használni.

3.3. Különböző kimentetek előállítási lehetőségei a gyakorlatban

A következőkben a docbook DTD alapján elkészített dokumentumok transzformációit fogjuk áttekinteni. Természetesen megfelelő stíluslapok segítségével tetszőleges DTD-n alapuló dokumentumok hasonló módszerekkel transzformálhatók.

Nyomtatáshoz használt kimenetek

Linux rendszerünkkel háromféle nyomtatáshoz optimalizált kimenetet készíthetünk, DVI, PS és PDF formátumokat. Ezen kimenetek előállításához docbook dokumentumok esetén a DSSSL stíluslapokat használjuk mellyel TEX-forrást készíthetünk dokumentumunkból, majd a jadetex vagy pdfjadetex makró segítségével DVI vagy PDF fájlt készítünk belőle. A DVI fájlból a szokásos módon a dvips programmal PS fájlt készíthetünk. Ezzel a módszerrel nagyon jó minőségű, akár nyomtatásban is megjelenő kiadványok készíthetők, automatikus magyar elválasztással. Bizonyos esetekben jól jöhet, hogy a jade és az openjade rendelkezik egy backenddel mellyel RTF formátumú kimenet készíthető.



Egy másik, szintén TEX -alapú megoldás segítségével, a megfelelő XSLT stíluslappal először xsl:fo fájlt készítünk, majd ezt az xmltex makró segítségével feldolgozzuk. Az így kapott DVI fájlt a dvips és a dvipdf programmal PS és PDF formátumra konvertálhatjuk. Ez a módszer használható többek között TEI2 dokumentumokra is, azonban néhány bosszantó hibája van, többek között a margókat nem helyesen kezeli és nem kompatibilis az újabb docbook stíluslapokkal.

A harmadik módszerhez szintén xsl:fo fájlra van szükség, amelyet a fop nevű java alapú szoftverrel alakíthatunk többek között PDF-fé. Ez a program azonban még csak a fejlesztés korai fázisában van, többek között a nyelvi támogatás sem teljes benne, így nagyobb dokumentumok feldolgozására nem ajánlatos használni.

Elektronikus kimenetek készítése

A docbook xsl stíluslap csomag segítésével egyetlen transzformációval készíthető HTML és XHTML kimenet, akár egyetlen fájlba, akár több, fejezetekre tördelt automatikus tartalomjegyzékkel ellátott fájlba.

Ezeken a stíluslapokon alapulnak a docbook xsl stíluslapokhoz tartozó htmlhelp és javahelp formátumokat előállító stíluslapok.

Amennyiben dokumentumunkat PDA-kra szeretnénk publikálni, általában több lépésre van szükség. Palmdoc készítéséhez használhatjuk a pyrite publisher programot [12], mely HTML fájlból készít PDB-t, amelyet aztán a Palmra feltöltve a legtöbb e-book olvasó képes megjeleníteni. A plucker nevű programmal [13] szintén készíthetünk HTML oldalakból Palmon olvasható dokumentumot, ez azonban csak a saját – egyébként szabad szoftver – olvasójával tekinthető meg.

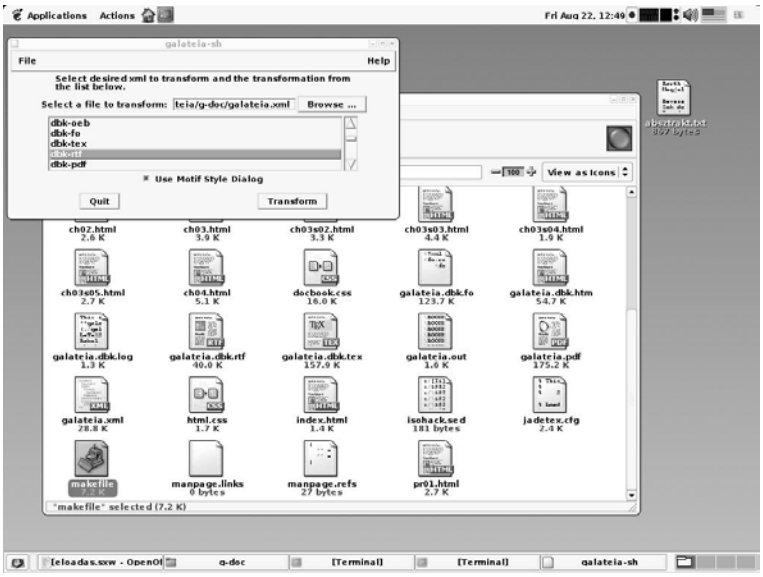
Az openebook e-book formátum, több olvasó, többek között a Microsoft Reader valamint néhány fogyatékosoknak készülő hangos könyv [14] kiinduló formátuma, melyet XHTML kimentből saját stíluslappal állíthatunk elő.

Végső esetben a HTML kimenetből a lynx segítségével szépen tördelt szövegfájlt készíthetünk. Ez természetesen megfelelő stíluslappal egy lépésben is elkészíthető a dokumentumunkból.

Segédprogramok

SGML docbook transzformációkhoz két segédprogram csomag is létezik az sgml2x és a docbook-utils. Ezek gyakorlatilag olyan héjjprogramok, melyek segítségével egyrészt csökkenthetjük a legépelt karakterek számát, másrészt figyelnek bizonyos tipushibákra, például a TEX -makrók esetében. Mindkét programcsomag hasonló tudású, a docbook-utils csomag tartalmaz egy perl helpert, amellyel man oldalt készíthetünk.

A legépelt karakterek számának csökkentésére egy másik lehetséges mód egy saját Makefile készítése, melyben elhelyezzük az összes saját szánk íze szerinti transzformációt.



Akit a téma bővebben érdekel tanulmányozhatja a galateia programot [15], mely egyrészt egy Makefile-alapú grafikus frontend, másrészt pedig egy réteges transzformációs rendszer, melynek segítségével mind a dokumentumok egységes megjelenése, mind pedig az egyes dokumentumok ettől eltérő megjelenése beállítható.

4. Nehézségek

A dokumentumok kódolásakor felmerülő első nehézség a magyar nyelvvel kapcsolatosan a különböző kódlapok használata. Magyar nyelvű szöveghez alapvetően az ISO8859-2 és az UTF-8 használatos. Sajnos ez a különböző megjelenítő eszközök esetén más és más lehet, gondoljunk csak a különböző webböngészőkre, vagy mondjuk a Palm PDA-ra amely egyiket sem támogatja.

Sajnálatos módon általánosan jó megoldás nincs, a legtöbb esetben azt lehet mondani, hogy a SGML transzformációk esetén érdemes az ISO8859-2 kódolást használni, ha XML fájlt transzformálunk DSSSL stíluslappal akkor a speciális XML kódolást használjuk az SP_ENCODING környezeti változó beállításával, míg XSLT esetén használjunk UTF-8 kódolást.

A másik sarkalatos pont a képek elkészítése, hiszen ha ugyanazokat a képeket szeretnénk használni nyomtatásban mint egy weboldalon akkor elég behatároltak a lehetőségeink mint formátum, mint méret szempontjából. Ilyen esetben, 150 vagy 300 dpi felbontású maximum 500 pixel nagyságú jpeg képet használjunk. Ez felbontástól függően 5, 10 centiméteres képet eredményez, amely ábrának elég nagy, de a webről még gyorsan letölthető, és mind PDF-be mind pedig PostScript dokumentumba beilleszthető.

Hivatkozások

[1] http://www.oasis-open.org/docbook/

[2] http://www.tei-c.org/

[3] http://www.jedit.org/

[4] http://www.openoffice.org/

[5] http://www.jclark.com/jade/

[6] http://openjade.sourceforge.net/

[7] http://www.jclark.com/xml/expat.html

[8] http://www.gingerall.com/charlie/ga/xml/p_sab.xml

[9] http://xml.apache.org/xalan-c/

[10] http://saxon.sourceforge.net/

[11] http://xml.apache.org/xalan-j/

[12] http://www.pyrite.org/

[13] http://www.plkr.org/

[14] http://www.daisy.org/

[15] http://galateia.fsf.hu/

A Common Criteria

Bálint Sándor

2003. október 27.

Kivonat

A cikk célja a „Common Criteria for IT Security Evaluation” szabvány (röviden CC vagy CCITSE, egyben ISO15408 szabvány is) ismertetése, különös tekintettel arra, hogy miként hasznosítható az általános nyílt forráskódú fejlesztések során.

Sor kerül a Common Criteria fő dokumentumainak (védelmi profilok, biztonsági rendszertervek) ismertetésére, szó esik olyan kifejezésekről, mint értékelési garanciaszint, biztonsági garanciális intézkedések és biztonsági funkcionális követelmények.

Az előadás áttekintést nyújt arról, hogy az USA Nemzetbiztonsági Hivatala (NSA, National Security Agency) által elkészített, az ITSEC B1 minősítésével ekvivalens Labeled Security Protection Profile (LSPP) funkcionális követelményei miként teljesíthetők Linuxon ill. más nyílt forráskódú szoftverekkel, illetve hogyan fejleszthetünk olyan alkalmazásokat, rendszereket, amelyek megfelelnek az abban foglalt biztonsági funkcionális és garanciális követelményeknek.

Az értékelési garanciaszintek kapcsán szó esik arról, hogy CC-s szempontból milyen korlátai vannak a nyílt forrású fejlesztésnek, valamint a szoftverek biztonsága és az értékelési garanciaszint közötti – nem teljesen egyértelmű – összefüggésről.

Tartalomjegyzék

1. A Common Criteria – bevezetés

A Common Criteria (teljes nevén Common Criteria for Information Technology Security Evaluation, rövidítve CCITSE vagy CCIMB-99-031) nyílt szabvány, jelen pillanatban a 2.1 verziónál tart, melyet 1999-ben adtak ki. 2.0 verziója ISO/IEC 15408:1999 néven ISO szabvány is lett. Immár magyar nyelven is elérhető, azoknak javasoljuk, akik számára problémát jelent a többszörösen rekurzív angol nyelvű rövidítésekkel megpakolt szakszöveg olvasása.

A Common Criteria készítői szándéka szerint lehetőséget teremt arra, hogy az informatikai rendszerek felhasználói az általuk használt vagy használni kívánt termékek (eszközök, rendszerek, szoftverek és hardverek egyaránt) IT biztonsági szintjét egy konkrét, előre elkészített követelmény- vagy szempontrendszer – innen a Common Criteria=„egységes követelményrendszer” elnevezés – alapján ítélhessék meg. Ezen követelményrendszer segítségével egyértelművé válik, hogy az adott termék egy bizonyos célnak megfelel vagy sem, illetve az is, hogy a megfelelőséghez milyen biztonsági követelményeknek kell még eleget tennie.

A Common Criteria (CC) a fentiekből is látható módon hasznos lehet mind a felhasználók, mind a fejlesztők számára. A felhasználók (ahol felhasználó alatt vállalati környezetben elsősorban az informatikai rendszerek bevezetését irányító döntéshozót értjük) a CC rendszer alapján történő értékelés eredményeként láthatja, a használni kívánt rendszer mely biztonsági követelményeknek tesz eleget, s ezek ismeretében hozhatja meg döntését. A fejlesztő számára pedig oly módon jelent segítséget a CC, hogy a felhasználó nem csak homályos utalást tesz az általa megrendelt eszközzel/rendszerrel szemben elvárt biztonsági funkcionalitásra, hanem pontos követelményrendszert tud a fejlesztő rendelkezésére bocsátani, aki ily módon azt a fejlesztés során mintegy vezérfonalként használva bizonyos lehet abban (mellesleg: bizonyítani is képes), hogy a fejlesztés végén előálló rendszer megfelel azoknak a biztonsági elvárásoknak, amelyeket a fejlesztés megkezdésekor a felhasználó vele szemben előírásként támasztott.

A magukra biztonsági szempontból valamit adó fejlesztők számára a fentieken túl azért is fontos lehet a CC módszertanával mielőbb megismerkedni, mert – mint az egy, a közelmúltban megrendezésre került konferencián elhangzott – 2003 év során Magyarország befogadó államként csatlakozni kíván a Common Criteria országaihoz. Ez azt jelenti, hogy az államigazgatási szervek számára történő szoftver- ill. rendszerfejlesztések során jó eséllyel előbb vagy utóbb így is, úgy is szemben találják magukat a követelményrendszerrel. A Common Criteria szabvány teljes szövege szabadon letölthető a `http://www.commoncriteria.org/` webcímről.

2. A Common Criteria szabvány felépítése

A Common Criteria mint szabvány, igencsak vaskos dokumentum: három kötetből áll, melyek a következők:

1. Bevezetés és általános modell
2. Biztonsági funkcionális követelmények
3. Biztonsági garanciális követelmények

Az első kötet tartalmazza a CC szabvány alkalmazásával kapcsolatos alapvető tudnivalókat, a szabvány felépítését, az egyes dokumentumok szerepét és az azokra vo-

natkozó formai és tartalmi elvárásokat. A második és harmadik fejezet rendre a funkcionális és garanciális követelmények (úgynevezett komponensek) listáját tartalmazza hierarchikus rendbe szedve. Ez utóbbi köteteket szokás komponens-katalógusnak is nevezni. A harmadik kötet tartalmazza a biztonsági funkcionális követelmények listáján túl az úgynevezett értékelési garanciaszintek (angol rövidítésük után EAL szinteknek is nevezett biztonsági szintek) által megkövetelt garanciális elvárások körét is. Ez utóbbit szokás a régi TCSEC (Trusted Computing Systems Evaluation Criteria) szabványban megfogalmazott biztonsági szintek, úgymint B1, C2 Common Criteria szerinti utódjának tekinteni. Később látni fogjuk, hogy ez nem teljesen valós, azonban az elmondható, hogy a 7 előre definiált EAL szintből minél magasabbra van értékelve egy adott rendszer vagy eszköz, annál biztosabbak lehetünk az adott termék biztonsági funkcióinak működőképességében.

3. A Common Criteria módszertana – áttekintés

A Common Criteria módszertana a következő modellt követi:

1. A rendszer biztonsági környezetének leírása
2. Biztonsági célok meghatározása
3. Biztonsági követelmények megállapítása

A fenti rétegek egymásra épülnek, és a CC módszertana előírja, hogy mindent indokolni kell. A majdani rendszer biztonsági környezetének (feltételezések, fenyegetettségek, érvényben levő szervezeti szabályozások) ismeretében kell meghatározni a biztonsági célokat, oly módon, hogy egyrészt minden cél egyértelműen visszavezethető legyen a környezettel kapcsolatos megállapításokra, másrészt minden, a környezettel kapcsolatos megállapításunkhoz tartozzék legalább egy olyan biztonsági cél, amely azt lefedi. Hasonló módon, a biztonsági követelmények megállapítása a már korábban meghatározott biztonsági célokhoz illeszkedik. A módszertan helyes használatával a rendszer koherens, egyenszilárd védelme megoldható, valamint biztosak lehetünk abban, hogy

- a fejlesztés végén előálló rendszer az általunk meghatározott környezetben való működésre megfelelően felkészített, valamint
- a rendszer egyetlen biztonsági funkciója sem felesleges.

Ez utóbbi ugyan furcsán hangozhat, ám az alábbi kép talán segít megláttatni, miért is fontos ez.

Biztonság	Használhatóság
Menedzselhetőség	Rendelkezésre álló anyagi források

Lássuk be: a gyakorlat azt mutatja, hogy a fenti szempontok egymással szinte kibékíthetetlen ellentétben vannak (a biztonságos és jól használható rendszerek általában menedzselhetetlenek, a jól menedzselhető és használható rendszerek többnyire nem igazán biztonságosak, a biztonságos és jól menedzselő rendszerek gyakran használhatatlanok, a biztonságos, jól használható és könnyen menedzselhető rendszerek pedig

többnyire megfizethetetlenek). Márpedig, ha egy rendszert kizárólag a biztonság szempontjából közelítve építünk fel, az mind a használhatóság, mind a menedzselhetőség kárára mehet, azon túl, hogy olykor drágábbá is teszi a terméket. Ezért is fontos az, hogy a rendszerben azok és csak azok a biztonsági funkciók legyenek jelen, amelyekre valóban szükség van (amellett természetesen, hogy ezen funkciók megfelelő módon kerüljenek kiválasztásra és persze megvalósításra is).

A CC modellje – biztonsági szabvány lévén – nem foglalkozik ugyan külön sem a menedzselhetőség, sem a funkcionalitás, sem a rendszer fejlesztésének költségeinek kérdéseivel, azon túl, hogy az egymásra épülő rétegek egymásból történő származtatása révén biztosítja a kockázatarányos védelem megvalósítását, s lehetővé teszi a biztonsági követelmények tisztázását még a fejlesztés megkezdése előtt. Ez a projekt jellegű fejlesztések esetén mint a sikerkritériumok egyike hasznosítható.

4. A Common Criteria terminológiája

Az alábbiakban a fontosabb CC kifejezések rövid ismertetője található:

Angol rövidítés	**Angol nyelvű elnevezés**	**Magyar elnevezés**
CC	Common Criteria	Közös követelményrendszer
EAL	Evaluation Assurance Level	Értékelési garanciaszint (ÉGSz)
IT	Information Technology	Informatika
PP	Protection Profile	Védelmi profil
SF	Security Function	Biztonsági funkció
SFP	Security Function Policy	Biztonsági funkció politikája
SOF	Strength of Function	A funkció erőssége
ST	Security Target	Biztonsági rendszerterv
TOE	Target of Evaluation	Az értékelés tárgya
TSC	TSF Scope of Control	Az értékelés tárgya biztonsági funkcióinak hatóköre
TSF	TOE Security Functions	Az értékelés tárgya biztonsági funkciói
TSFI	TSF Interface	Az értékelés tárgya biztonsági funkcióinak interfészei
TSP	TOE Security Policy	Az értékelés tárgyának biztonsági politikája

5. A Common Criteria két fő dokumentuma: a védelmi profil és a biztonsági rendszerterv

A Common Criteria két fő dokumentummal segíti mind a fejlesztők, mind a megrendelők munkáját (akik belső fejlesztés illetve nem külső megrendelés esetében természetesen megegyezhetnek). Az egyik neve védelmi profil (angolul Protection Profile, rövidítése: PP), mely egy megvalósítás-független dokumentum, amely absztrakt módon fogalmazza meg a konkrét rendszerrel szembeni biztonsági követelményeket. A másik dokumentum a biztonsági rendszerterv (Security Target, rövidítése: ST). Ez utóbbi már a követelményeken túl a konkrét megvalósítás leírását is tartalmazza, úgymint a rend-

szer biztonsági funkcióit, valamint a rendszer fejlesztése során alkalmazott úgynevezett garanciális intézkedéseket.

A védelmi profilok és biztonsági rendszertervek elkészítésében az alábbi módokon nyújt segítséget a Common Criteria szabvány:

- megadja a dokumentumok formai követelményeit
- ismerteti az egyes szakaszok tartalmi követelményeit
- a komponens-katalógus (lásd alább) segítségével mintát szolgáltat a követelményeket tartalmazó szakasz elkészítésére
- úgynevezett csomagok (package) segítségével előre összeállított követelményhalmazokat nyújt, melyeket tetszés szerint felhasználhatunk (példa: EAL szintek)

A fentieken túl a Common Criteria módszertan alapján az Amerikai Nemzetbiztonsági Szolgálat (National Security Agency, NSA) elkészített néhány védelmi profilt, amelyeket szabadon hozzáférhetővé is tett. Ezek közül a két leglényegesebb a CAPP (Controlled Access Protection Profile, hozzáférés-vezérelt védelmi profil, a TCSEC C1 kategória Common Criteria szerinti megfelelője), a másik az LSPP (Labeled Security Protection Profile, magyarul talán kötelező hozzáférés-vezérelt védelmi profilnak lehetne fordítani, ezen védelmi profil nagyjából a TCSEC B1 besorolásnak felel meg). A `http://www.commoncriteria.org/` weboldalon további védelmi profilok érhetőek el.

Rövid kereséssel ugyancsak találhatunk kész biztonsági rendszerterveket is az Interneten. Ezek közül talán a többek érdeklődésére tarthat számot a SUN Solaris 8 operációs rendszerének, illetve a Microsoft Windows 2000 operációs rendszerének biztonsági rendszerterve. Mindkét dokumentum szabadon letölthető a fejlesztő weboldalairól.

6. Biztonsági funkcionális és garanciális követelmények

A CC filozófiája szerint szükség van a biztonságot fenyegető tényezők, valamint az érvényes szervezeti biztonsági szabályozások tisztázására, a biztonsági intézkedéseket ezek alapján kell levezetni, valamint igazolnunk kell, hogy az előírt intézkedések elégségesek a célok eléréséhez.

Ezen kívül olyan intézkedésekre is szükség van, amelyek használatával csökkenthető a sebezhetőségek esélye, azok szándékos kihasználásának lehetősége, valamint a kár mértéke, amely egy létező sebezhetőség kihasználása esetén érhet bennünket. Olyan intézkedésekre is szükség van, amelyek elősegítik a fejlesztés lezárultát követően is a sebezhetőségek felismerését, azok megszüntetését, a károk csökkentését és/vagy jelzik azt, hogy egy adott sebezhetőség kihasználásra került.

A biztonsági követelményeket a Common Criteria két kategóriába sorolja: egyrészt funkcionális követelményekre, melyek elsősorban technikai módon megvalósított biztonsági funkciókat illetve a funkciók által érvényesített szabályokat tartalmaznak, másrészt pedig garanciális intézkedésekre, melyek azt hivatottak garantálni, hogy a rendszer biztonsági funkciói valóban megbízhatóan és rendeltetésszerűen működnek.

A könnyebb eligazodás érdekében az egymáshoz hasonló követelmények (mind a funkcionális, mind a garanciális követelmények esetében) kategóriákba, úgynevezett családokba sorolódnak, a családok pedig úgynevezett osztályokat alkotnak – hasonlóan a fajok darwini rendszeréhez.

A könnyebb hivatkozás érdekében az osztályok három karakterből álló kódot kaptak, melyek első karaktere utal a követelményosztály típusára (A=funkcionális. ill. A=assurance, vagyis garanciális), a második és harmadik karakter pedig az osztály nevének rövidítése.

Pl: FAU, ADV

Az osztályokon belüli családok elnevezése a 3 betű + alulvonás + 3 betű formulát követi, ahol az első három karakter azon osztály azonosítója, amelybe az adott család tartozik, míg a második három betű az osztály angol nevének rövidítése.

Pl: FAU_GEN, ADV_FSP

A konkrét követelményeket megfogalmazó egységek, a komponensek jelölése a család kódjából származik, melyet egy ponttal elválasztva és egy számjeggyel (az adott családon belüli komponens sorszámával) kiegészítve jelöl a CC:

pl: FAU_GEN.1; ADV_FSP.4

A Common Criteria második és harmadik kötete rendszerezve tartalmazza a komponenseket, ezért szokás komponens-katalógusként is utalni rájuk.

Funkcionális osztályok

Kód	**Osztály angol elnevezése**	**Magyar elnevezés**
FAU	Security Audit	Biztonsági naplózás
FCO	Communication	Kommunikáció
FCS	Cryptographic support	Kriptográfiai támogatás
FDP	User data protection	Felhasználói adatok védelme
FIA	Identification and authentication	Azonosítás és hitelesítés
FMT	Security management	Biztonságmenedzsment
FPR	Privacy	Személyes adatok védelme
FPT	Protection of the TOE security functions	Az értékelés tárgya biztonsági funkcióinak védelme
FRU	Resource utilization	Erőforrás-felhasználás
FTA	TOE access	Hozzáférés
FTP	Trusted path/channels	Megbízható csatornák

Garanciális osztályok

Kód	**Osztály angol elnevezése**	**Magyar elnevezés**
ACM	Configuration management	Konfiguráció-menedzsment
ADO	Delivery and operation	Szállítás és működtetés
ADV	Development	Fejlesztés
AGD	Guidance documents	Dokumentáció
ALC	Life cycle support	Élettartam-támogatás
ATE	Tests	Tesztelés
AVA	Vulnerability Assessment	Sebezhetőség-vizsgálat
AMA	Maintenance of Assurance	Garanciaszint fenntartása

6.1. Funkcionális követelmények

A funkcionális komponensek célja a biztonsági funkcionális követelmények definiálása, vagyis annak ismertetése, hogy milyen viselkedést várunk el az értékelés tárgyától. A komponenseket a védelmi profilban illetve biztonsági rendszertervben meghatározott biztonsági célok elérésére kell és lehet használni. Ezen követelmények célja az, hogy segítségükkel kézzelfoghatóvá, egyértelműen megfogalmazhatóvá váljanak az azonosított fenyegetettségek elleni védelem követelményei, valamint azok a feltételek, amelyeknek megfelelve eleget tehetünk a ránk érvényes szervezeti szabályozásoknak – mindezt az általunk elvárt működési környezetben.

A Common Criteria funkcionális követelményeket tartalmazó 2. kötetét haszonnal forgathatják mind az informatikai rendszerek felhasználói (döntéshozók), akik egy védelmi profilban vagy biztonsági rendszertervben megfogalmazott biztonsági célok megvalósításához keresnek maguknak eszközöket, mind a fejlesztők, akik szabványosított, jól rendszerezett módszertant kapnak annak biztosítására, hogy az általuk fejlesztett rendszerek valóban képesek a kitúzött biztonsági célok elérésére – no, meg aztán egyre több esetben fordul elő az is, hogy egy megrendelő eleve meg is adja a fejlesztők számára az általa megfelelőnek vélt védelmi profilt, mint követelményrendszert, és ilyenkor nem árt tudni, hova is forduljunk segítségért. Ez után fejlesztőként nem kapkodni fogunk, mint Bernát a ménkűhöz, hanem ehelyett módszeresen kikeressük a komponenseket a komponens-katalógusból, megértjük, milyen célt szolgálnak, s így végül minden bizonnyal azt fogjuk megvalósítani, amit a kedves megrendelő elvárt. Ily módon a reklamáció egy jó részétől is megszabadulhatunk – ráadásul nyugodtan alhatunk, mert a rendszerünkből jó eséllyel nem maradt ki egyetlen fontos védelmi funkció sem.)

6.2. Garanciális követelmények

Míg a funkcionális követelmények elsősorban IT eszközökkel megvalósítható funkciókat írnak elő meglehetős részletességgel (például: legyen olyan naplózó funkció, amely a naplózás elindulását és leállítását rögzíti), addig a garanciális követelmények a tervezés, megvalósítás és üzemeltetés menetére (egyebek között a teszteléssel szemben támasztott követelményekre, adminisztrátori és felhasználói dokumentáció elkészítésére) tartalmaznak útmutatást.

A biztonsági garanciális követelmények elsődleges célja a sebezhetőségek csökkentése. Sebezhetőségek felbukkanására abban az esetben kell számítanunk, ha a követelmények, megvalósítás, vagy éppen az üzemeltetés területén hiányosságok mutatkoznak, ennek megfelelően a garanciális követelmények is külön hangsúlyt fektetnek e területekre.

A garanciális követelmények alapja az értékelés: minél alaposabb az értékelés folyamata, minél több területre terjed ki, s minél szigorúbb követelményrendszer alapján történik, annál nagyobb bizonyosságot (garanciát) szerezhetünk az elkészült termék jóságát illetően. E mellett fontos cél az is, hogy minél magasabb garanciaszintet érjünk el úgy, hogy emellett az értékelésre csakis a minimálisan szükséges energiát kelljen fordítanunk. Többek között ebben segítenek az értékelési garanciaszintek.

6.3. Értékelési garanciaszintek

Az értékelési garanciaszintek a CC terminológiája szerint vett csomagok, melyek 7 egyre szigorúbb kategórában sorolnak fel biztonsági garanciális követelményeket. Fon-

tos azonban megjegyezni, hogy az EAL csomagok nem tartalmaznak minden biztonsági garanciális osztályból komponenseket: a legtöbb esetben tehát szükséges lehet a kiegészítésük. A 7 EAL szint a következő:

EAL szint	**Angol elnevezés**	**Magyar elnevezés**
EAL1	Functionally tested	Funkcionálisan tesztelt
EAL2	Structurally tested	Strukturálisan tesztelt
EAL3	Methodically tested and checked	Módszeresen tesztelt és ellenőrzött
EAL4	Methodically designed, tested and reviewed	Módszeresen tervezett, tesztelt és áttekintett
EAL5	Semiformally designed and tested	Félformálisan tervezett és tesztelt
EAL6	Semiformally verified and tested	Félformálisan igazoltan tervezett és tesztelt
EAL7	Formally verified design and tested	Formálisan igazoltan tervezett és tesztelt

Általánosságban elmondható, hogy

- a CC nem ismer részleges EAL megfelelőséget. Egy termék vagy teljesíti az EAL szint által megkövetelt garanciákat, vagy nem.
- amennyiben az EAL követelményein túl egy termék további követelményeknek is megfelel, azt az EAL szint jelölése utáni + jellel szokás jelölni: pl. EAL2+, EAL4+. A fentiekből következik, hogy nincs EAL4- vagy EAL7-.
- egy már meglevő termék CC szerinti értékelése során az elérhető legmagasabb minősítés az esetek többségében legfeljebb EAL4, valamint
- EAL5-nél magasabb értékelésre csak nagyon kevés, általában korlátozott funkcionalitású terméknek van esélye (például chipkártyák)
- minél magasabb EAL szintet célzunk meg, annál több időbe telik – és több pénzbe kerül – a termék elkészítése
- végül pedig: magasabb EAL szintből nem következik egyenesen magasabb általános biztonsági szint, tekintve, hogy az EAL csomagok önmagukban nem tartalmaznak egyetlen biztonsági funkcionális követelményt sem.

7. Common Criteria és a fejlesztők

Ha valaki azt mondja, hogy a Linux soha nem lesz képes elérni azt a biztonsági értékelési garanciaszintet, amelyet a kereskedelmi, zárt forrású operációs rendszerek, minden elfogultság ellenére (vagy nélkül?) azt kell mondanunk, hogy lehet benne némi igazság. Ez persze nem jelenti azt, pl. a Microsoft Windows 2000 operációs rendszer, amely megkapta az EAL4+ minősítést, bizonyítottan biztonságosabb, mint pl. a SuSE Linux Enterprise Server 8, amely EAL2 szinten lett értékelve.

De vajon miért van az, hogy míg a Windows 2000 EAL4+ szinten került minősítésre, a SuSE Linux pedig csak EAL2-n? Az alapvető különbség a fejlesztési modellben rejlik.

Mint azt már láttuk, a CC biztonsági funkcionális követelményei az értékelés tárgya biztonsági funkcióival kapcsolatos követelményeket tartalmazzák, a garanciális követelmények pedig a funkciók megfelelő működését hivatottak garantálni a fejlesztés, szállítás, üzembe helyezés illetve üzemeltetés menetére vonatkozó követelmények definiálásával.

Hogy a fejlesztésre vonatkozó követelményeknél maradjunk, gondoljuk magunk elé a következő helyzetet. Cégünk egy ügyfélkapcsolati és -nyilvántartási rendszerszoftvert készül bevezetni. A fejlesztő el is készítette a szoftvert a mi igényeink szerint, azonban a bevezetést követően meglepetten tapasztaljuk, hogy rövid idő leforgása alatt versenytársaink hosszú ideje hűséges ügyfelek tömegeit hódítják el tőlünk. Hosszas nyomozást követően kiderül, hogy a programban általunk nem kért funkció is található; nevezzük mondjuk trójai kódrészletnek. A fejlesztő természetesen tagadja, hogy szándékosan került oda a kód, mi több: azt állítja, hogy nem az ő fejlesztői keze munkáját dícséri az ominózus programrészlet.

Hogyan segít az ilyen helyzetek megelőzésében a Common Criteria?

Először is a fejlesztésre vonatkozó garanciális követelmények olyan dolgokat írnak elő, mint például hogy a fejlesztés során konfiguráció-menedzsmentet kell használni. Mi is ez? A talán legtöbb helyen alkalmazott konfiguráció-menedzsment eszköz a CVS (Concurrent Versioning System), amely a konfiguráció-menedzsment egyik legelterjedtebb eszköze. Helyes használatával bármilyen kódrészletről meg lehet mondani, hogy ki, mikor adta hozzá az alkalmazáshoz, ezáltal a hibák, esetleges szándékos „extrák” hozzáadása letagadhatatlanná válik – valamint bizonyítható, hogy a nyilvántartott változtatásokon kívül senki nem nyúlt bele a program forrásába. Innen már csak arra kell ügyelni, hogy a lefordított binárist ne tegye valami rosszindulatú cracker a magáévá.

Másodszor: további garanciális követelmények foglalkoznak az értékelés tárgya üzembe helyezésével, illetve (le)szállításával. Ezek azt hivatottak biztosítani, hogy valóban az kezdjen el üzemelni, amit a fejlesztő saját telephelyén elkészített. Megoldási lehetőség számos akad, talán a kriptográfia (digitális aláírás) használata tűnik a legkönnyebben kivitelezhetőnek, ezáltal ugyanis a megrendelő minden esetben ellenőrizni képes, hogy az általa megkapott/letöltött/megvásárolt termék valóban az-e, aminek látszik – illetve egészen pontosan csak azt, hogy valóban a fejlesztő írta-e alá.

Harmadszor: a biztonsági garanciális követelmények egyike éppen a módszeres fejlesztésre vonatkozik. Ennek egy kissé kevésbé emberbarát (illetve fejlesztőbarát) változata az, amely egy magasabb értékelési garanciaszint alapját szolgáltatja – ez pedig nem más, mint a formális programhelyesség-bizonyítás. Ettől a fejlesztők hátán a szőr feláll, szemük kigúvad, szájuk széle remegni, majd habzani kezd. Kicsit megnyugodhatnak: a formális programhelyesség-bizonyítás csak EAL7 szinten követelmény.

De vajon milyen plusz terheket ró a nyílt forrású szoftverek fejlesztőire az, ha a Common Criteria módszertanát követve akarnak fejleszteni? Először is nagyobb fegyelmet, rendszerezettséget (a konfiguráció-menedzsment szoftvert kell használni mindenfajta módosításra, amihez célszerűen csak a vezető fejlesztők férhetnek hozzá írási joggal, stb.), valamint olyan intézkedéseket, amelyek megakadályozzák a fejlesztés során történő kódkorrupciót (pl. tűzfal mögött történő fejlesztés, personal firewall alkalmazása) ami, mint az már elhangzott, egyfajta – ráadásul rendkívül hatékony – módja a programbeli sebezhetőségek kezelésének.

7.1. Továbbfejlesztett alkalmazások, rendszerek és a Common Criteria

A fentiekből talán sejlik, hogy a Common Criteria módszertanának teljes mértékben megfelelő módon csak úgy lehet fejleszteni, hogy már a kezdetektől fogva a megfelelő követelmények szem előtt voltak. Mi a helyzet azonban azokkal a projektekkel, amelyek nem eleve a CC módszertanának figyelembe vételével indultak el (arról nem is beszélve, hogy számos program már azelőtt létezett, hogy az egységes szabványrendszer pajzán gondolatként megfogant ősatyjai fejében)? Igen, ilyen projekt a Linux kernelének fejlesztése is.

A Common Criteria módszertana szerint természetesen nem csak olyan termékeket/rendszereket lehet értékelni, amelyek kifejezetten a CC módszertana szerint készültek. Sőt: ha egy adott termék elbukna a CC értékelésen, lehetőség van javítására és szükség esetén újbóli vizsgálatokra. A tapasztalat azt mutatja, hogy a Common Criteria szerinti legmagasabb értékelési garanciaszint, amelyet egy már meglevő projektre viszonylag értelmes erő-, energia- és pénzráfordítással rá lehet húzni (más szóval: amelyre iszonyatos költekezés nélkül fel lehet hozni a rendszert), valahol az EAL4 környékén megáll.

7.2. Érdemes-e egyáltalán Common Criteria szerint fejleszteni?

A kérdésre adandó válasz – bármily meglepő – nem olyan nevetségesen egyszerű, mint amilyennek első látásra tűnik. A Common Criteria szerint fejleszteni ugyanis nem olcsó: mind a fejlesztő, mind a felhasználó megérzi a bőrén (vagy a pénztárcája vastagságán). Előző azért, mert az egyébként máskor talán csak összecsapott munkát most sokkal gondosabban kénytelen elvégezni, utóbbi azért, mert szívinfarktust kap a felhasználó az elé letett árajánlaton szereplő összegtől. Amennyiben fontos a rendszer rendelkezésre állása, a rendszerben tárolt adatok bizalmassága, hitelessége, a műveletek letagadhatatlansága, illetve amennyiben fejlesztőként garantálnunk kell (esetleg mi magunk szeretnénk bizonyítani), hogy amit elkészítettünk, abból nem maradt ki egyetlen biztonsági szempont sem, úgy igen, van értelme a megnövekedett munkaterhek és költségek ellenére. Amennyiben a majdani értékelés tárgya semmiféle biztonsági funkcionalitást nem lát el, úgy bizonyos esetekben bátran eltekinthetünk a CC szerinti fejlesztéstől. Jó tudni azonban, hogy a CC által támasztott követelmények közül számos – ha nem is olyan részletességgel és precizitással, mint ahogyan az a CC-ben megtalálható – minden fejlesztési projekt során jól jöhetnek. Az adminisztrátori és felhasználói dokumentáció, amely nélkül (felhasználók és megrendelők, figyelem!) terméket átvenni nem illik, kitűnően illusztrálja ezt a megállapítást. A követelmények jelentős része pedig a fejlesztés munkáját segíti.

A kérdést átfogalmazva tehát egyértelműbb választ kapunk: arra a kérdésre, hogy az általános fejlesztések során hasznos-e a Common Criteria módszertanának szem előtt tartása, a válasz a leghatározottabb igen. Ugyanez vonatkozik a többi, már meglevő projekt továbbfejlesztésére is. Már persze, ha fontos számunkra az, hogy az alkalmazás vagy rendszer, amelyet a kezünkből kiadunk, jól dokumentált, megfelelő mélységben végiggondolt és megtervezett, a felesleges sallangoktól mentes, ennek ellenére az alapvető biztonsági elveket követő legyen.

openMosix: Live free() or die()

Bodnár Csaba

2003.11.08.

Kivonat

Az openMosix napjaink egyik legizgalmasabb Linux-alapú telep-projektje (cluster projekt). Más telep-típusoktól eltérően megpróbálja teljes egészében egyetlen nagy Linux-gépként láttatni a tetszőleges számú csomópontból álló telepet. Az előadás áttekinti az openMosix [1] projekt –az eredeti Mosix szabad leágazása (fork)– jelenlegi állását.

Tartalomjegyzék

1. „Live free() or die()”

1-2 évvel ezelőtt a BSD-ről tartott valaki előadást hasonló címmel. Ennek ellenére én is ezt a címet adtam az előadásnak, egyrészt mert maga az openMosix Howto is ezt a jelmondatot használja, másrészt mert a Mosix „szabadsága” 2001-ben valóban veszélybe került, amikor a projekt két vezetője nem tudott megegyezni annak további licencpolitikáját illetően. Az eredmény a fejlesztés kettéválása lett: Dr. Moshe Bar – mindnyájunk örömére– úgy döntött, hogy továbbra is GPL licenceléssel viszi tovább a kódot, openMosix [1] néven.

2. Az openMosix projekt történetének mérföldkövei

- A 80-as években született PDP-11/70-en. Egy teljes és egy „lemeztelenített” (diszkless) PDP, innen jött a processz migrálás ötlete.
- 1997 óta fut GNU Linux alatt.
- 1999: Az első nagy „pofon” a nyílt forráskódú közösségnek: kijönnek egy csak bináris verzióval. . .
- 1998 és 2001 között Prof. Barak és Dr. Moshe Bar közösen vezetik a projektet a Hebrew University-n.
- 2001-ben Mosix/openMosix szétválás licenc problémák miatt.
- 2002 júliusára a Mosix installációk 97%-a openMosixra váltott.
- 2002 augusztusa óta a legaktívabb Linux-telep projekt.
- 2002 augusztus: HP-partneri kapcsolat.

3. A különböző telep-típusok áttekintése

Az openMosix egy Linuxos telep. Maga a telep szó (angolul cluster, fürt) nem takar mást, mint hogy valahány számítógépet valamilyen cél érdekében „egybefogunk”, összetartozónak tekintünk. Ha ez a cél a:

- *magas rendelkezésre állás*, akkor magas rendelkezésre állást nyújtó telepről (High Availability, HA, pl. Kimberlite, Red Hat Cluster Manager, . . .)
- *terhelésmegosztás*, akkor terhelésmegosztó telepről (pl. LVS)
- *szuperszámítógép teljesítmény elérése*, akkor tudományos-technikai célú telepről (High Performance Technical Computing, HPTC, pl. Beowulf)
- *egy nagy SMP-gépnek látsszon*, akkor SSI-telepről (Single System Image, még nem találtam rá jó magyar kifejezést :) (pl. openMosix, Qlusters [2], OpenSSI)

beszélünk.

4. Mi az openMosix?

Fenti terminológia szerint az openMosix egy olyan SSI-telep, amely elsődlegesen tudományos-technikai feladatok megoldására szolgál. Automatikusan kiegyenlíti a terhelést a telep különböző tagjai között, valamint az egyes tagok menet közben csatlakozhatnak hozzá vagy hagyhatják el a telepet. A terhelést adott szempontok szerint (CPU-sebesség, hálózati sávszélesség, ...) osztja szét a tagok között.

Hasonlóan a klasszikus VMS-telepekhez (a DEC 20-30 évvel ezelőtt csinált elsőként ilyen telepeket VMS operációs rendszer alatt), a telep bármely tagjára belépve a felhasználó által indított processz azon a tagon fog futni, amelyik az adott pillanatban legjobban ki tudja szolgálni annak igényeit, vagyis a rendszer megpróbálja optimálisan szétosztani a tagok között a felhasználók futó processzeit.

Fontos hangsúlyozni, hogy az erőforrások megosztása a processzek szintjén történik, vagyis egy processz egyszerre csak egy teleptag erőforrásait használhatja. A felhasználó úgy jár jól, ha munkáját egymással párhuzamosan futtatható részekre bontja és azokat külön processzként futtatja. A későbbiekben lesz majd lehetőség szálankénti (thread) szétosztásra is.

5. Hogyan működik az openMosix?

Moshe Bar szavaival élve: „A processz migráció maga elég egyszerű, egyszerűen átrakjuk a processzt az egyik gépről a másikra, és –szükség esetén– még több processzt rakunk át. A rendszerhívásokat pedig vagy visszaküldjük az UHN-re (Unique Home-Node), vagy –néhány esetben– helyileg hajtódnak végre (pl. DFSA, közvetlen fájlrendszer elérés esetén). Ez ennél nem lesz sokkal egyszerűbb. Az ördög a részletekben lakozik."

5.1. Az erőforrás-megosztó algoritmusok

Az egyik –az erőforrások felhasználására hatással levő– algoritmus nagyon egyszerű: ha egy adott tagon elkezd elfogyni a memória, a rendszer elkezdi átpakolni a processzeket a többi tagra.

A másik algoritmus dönti el, hogy egy adott processz melyik tagon fusson. Az algoritmus matematikai modellje közgazdasági kutatások eredménye. Annak eldöntése, hogy egy feladatnak melyik tag a legmegfelelőbb helye, igen bonyolult probléma. A legnagyobb baj az, hogy az egy telepen elérhető erőforrások nagyon különbözőek lehetnek: A memória, a CPU, a processz-kommunikáció, stb. nehezen összevethetőek, még csak nem is ugyanazokban az egységekben mérhetőek. Az algoritmus megpróbálja ezeket a heterogén erőforrásokat egységes formára hozni, költséget rendelni hozzá. A feladatok pedig mindig azokon a gépeken fognak futni, ahol a legalacsonyabb a „költségük".

5.2. Processz-migrálás

Minden egyes processznek van egy úgynevezett UHN-je (Unique Home-Node): ez az a teleptag, amelyen a processzt eredetileg elindították. Ezen az UHN-en fut a processznek egy „helyettese" (deputy), amely kapcsolatban van a processz tényleges, valószínűleg valamelyik másik tagon futó példányával, a „távolival" (remote). A legnagyobb probléma a rendszerhívások kérdése: minél több rendszerhívást kell a „távoli-

nak” visszaküldenie a „helyettes” felé, annál lassabb lesz az egész, hiszen a hálózaton keresztüli kommunikáció nagyságrendekkel lassabb a gépen belülinél.

A fejlesztők mindent elkövetnek azért, hogy minél több rendszerhívás tudjon helyileg végrehajtódni: ezt célozta meg az oMFS hálózati fájlrendszer kifejlesztése és a socket migrálás (ha egyszer lesz majd) is.

5.3. openMosix fájlrendszer (MFS), közvetlen fájlrendszer elérés (DFSA))

Az openMosix telep vagy hálózati fájlrendszer kifejezetten a DFSA részére lett kifejlesztve. Segítségével bármely tag el tudja érni a többi tagon lévő fájlrendszerek fájljait, így a migrált processzek anélkül érhetik el a hozzájuk tartozó adatokat, hogy ehhez a rendszernek távoli rendszerhívásokat kellene végrehajtania (nyilván így is lesz hálózati forgalom, de jóval kisebb, mert ez kifejezetten erre a helyzetre van optimalizálva).

5.4. Socket migrálás

Ha a processzel együtt át lehetne küldeni magát a socketet is (vagyis a meglévő TCP kapcsolatot is) valamely másik tagra, az szintén nagyban gyorsítaná a telep működését. Sajnos a megfelelő kódot még nem írta meg senki... illetve dehogynem! Amit Shah, az egyik fő fejlesztő saját bevallása szerint tavaly ezen dolgozott, a keletkezett kód azonban a Qlusters Inc. tulajdonába került :(A Qlusters nevű cég egy –az openMosixhoz hasonló– üzleti célú telepszoftver fejlesztésével foglalkozik, techológiai vezetője (CTO, Chief Technology Officer) Moshe Bar, az openMosix projektvezetője...

5.5. Telep-szinten elosztott memória (DSM, Distributed Shared Memory)

A fenti Amit Shah instrukciói alapján öt indiai diáklány –a MAASK-csoport [3] – diplomamunkának a DSM létrehozását választotta. Jelenleg ugyanis az openMosix egyik komoly hiányossága, hogy az osztott memóriát használó illetve a többszálú (multithreaded) alkalmazások nem migrálhatók, így nem részesülhetnek annak „áldásaiból”. A MAASK-csoport által fejlesztett Migshm (Migration of shared memory) segítségével mind a shared memóriát használó processzek (shmget(), shmat(), shmdt() and shmctl() rendszerhívások), mind pedig a többszálú processzek (clone() rendszerhívás) migrálhatóvá válnak.

Moshe Bar egyelőre nem tette be a Migshm-et az openMosixba, mondván hogy még nem elég stabil („I would rather have no DSM than bad DSM”).

6. Kezelőfelületek

6.1. karakter-orientált felület

A klasszikus karakter orientált segédprogramok (*mosmon*, *mosctl*, *mosrun*, *moslimit*, *migrate*) mind rendelkezésünkre állnak, emellett többféle grafikus eszköz is segíti munkánkat.

6.2. OpenMosixView-készlet

Az alábbi X-window alapú segédprogramok OpenMosixView-készlethez tartoznak:

- *openMosixview* - A fő figyelő és adminisztrációs alkalmazás
- *openMosixprocs* - Processz kezelő alkalmazás
- *openMosixcollector* - Telep és teleptag információ gyűjtő alkalmazás
- *openMosixanalyzer* - Alkalmazása a gyűjtött adatok elemzésére
- *openMosixhistory* - Telep-szintű processz-történet alkalmazás

6.3. openMosixWebView

Az openMosixWebView egy web alapú monitorozó eszköz az openMosixhoz.

7. openMosix előnyök és hátrányok

7.1. Előnyök

- Általában nincs szükség az alkalmazás módosítására.
- Nincs szükség kiegészítő csomagokra.
- Egyszerű installálás és konfigurálás (pl. rpm -Uvh openMosix*.rpm).
- openAFS integráció.
- Portolva IA-64-re, AMD-64 port folyamatban.
- Telep-szintű hálózati fájlrendszer (oMFS).
- Több más termék is az openMosixon alapul: openMosixView, openMosixWebView, openMosixApplet, RxLinux, PlumpOS, K12LTSP, LTSP, ...
- Maguk a felhasználók fejlesztik, a felmerülő igényeknek megfelelően.
- Teleptag automatikus felfedezése (Node autodiscovery/fail-over daemon).
- Telep „maszkolás” (Cluster Mask).
- Az automatikus hangolást lehetővé tevő segédeszköz.

7.2. Hátrányok

- Kernelfüggő.
- Telep-szintű osztott memória támogatás még nem stabil.
- Többszálú alkalmazások támogatása még nem stabil.
- Nincs még socket migráció.
- Ha az alkalmazásunk egyetlen processzből áll, nem nyerünk semmit vele.

8. Összefoglalás

Az openMosix már most is egy teljes értékű, sokak által használt SSI-telep szoftver. A várható további fejlesztésekkel kiegészítve nem csak tudományos-technikai számítások, hanem egészen más típusú feladatok (hálózati alkalmazások, adatbázis-kezelők, stb.) terhelésének megosztására is alkalmassá válik.

Hivatkozások

[1] openMosix: `http://openmosix.sourceforge.net/`

[2] Qlusters: `http://www.qlusters.com/`

[3] A MAASK-csoport: `http://mcaserta.com/maask/`

Mitől megy a villamos, avagy a HA-telepekben használt technológiák

Bodnár Csaba, bocs@missioncriticallinux.hu

2003.11.08.

Kivonat

Az előadás először áttekinti a Linux-alapú telepek (clusterek) típusait, különös tekintettel a HA-telepekre. Ezután az ezekben használatos különböző technológiákat veszi sorra; vagyis azokat a funkciókat, amelyektől egy rendszer HA-teleppé válik. Végül néhány ismertebb HA-telep szoftvert hasonlítunk össze a fenti funkciók megléte illetve implementációjának minősége szerint.

Tartalomjegyzék

1. A különböző telep-típusok áttekintése

Maga a telep szó (angolul cluster, fürt) nem takar mást, mint hogy valahány számítógépet valamilyen cél érdekében „egybefogunk”, összetartozónak tekintünk. Ha ez a cél a(z)

- magas rendelkezésre állás, akkor magas rendelkezésre állást nyújtó telepről (High Availability, HA, pl. Kimberlite[1], Red Hat Cluster Manager[3], SGI FailSafe[4]. . .)
- terhelésmegosztás, akkor terhelésmegosztó telepről (pl. LVS)
- szuperszámítógép teljesítmény elérése, akkor tudományos-technikai célú telepről (High Performance Technical Computing, HPTC, pl. Beowulf)
- hogy egy nagy SMP-gépként látsszon, akkor SSI-telepről (Single System Image) (pl. openMosix, Qlusters, OpenSSI)

beszélünk.
Mivel a legtöbb vállalatnál igazi jelentősége a HA-telepeknek van, a továbbiakban erre koncentrálva áttekintjük, melyek azok a technológiák, amelyek révén egy HA-telep több, mint néhány gép egymás mellé rakva:

2. Teljes redundancia (NSPOF, No Single Point Of Failure)

Bár nem technológia, hanem inkább alapelv, érdemes beszélni róla. Mivel HA-telepről van szó, itt a leglényegesebb a magas rendelkezésre állás, vagyis az hogy a rendszer folyamatosan működjön, az esetleges kiesés minimális időtartamra korlátozódjon.

Ennek egyik alapfeltétele, hogy ha valamely hardver eszköz meghibásodik a rendszerben, legyen helyette egy másik, amelyik automatikusan átveszi a szerepét; vagyis ne legyen egyetlen olyan eszköz sem a rendszerben, amelyből csak egy van (Single Point Of Failure).

Ez bizonyos eszközök esetén kis erőfeszítéssel elérhető, más eszközök (pl. az, hogy egy külső RAID-alrendszerben a külső RAID-vezérlő is redundáns legyen) esetén súlyos százezrekbe vagy akár milliókba is kerülhet. Azt, hogy a tartalék eszközre az átkapcsolás automatikusan megtörténjen, vagy maga a hardver, vagy pedig a HA-telep szoftvere intézi el.

3. Többségi szavazás (quorum)

Maga a quorum szó latin eredetű (jelentése: akié, akinek a. . .), angolszász nyelvterületen a következő jelentéssel bír: Egy szervezet tagjainak olyan gyűlése, amely elég nagy ahhoz, hogy döntőképes legyen.

Telepek esetén is hasonló értelemben használjuk: A telep csak akkor életképes, ha a quorumhoz szükséges számú tag elérhető és működik. Általában a quorumot 50%+1-ben szokták meghatározni, vagyis a telep tagjainak több mint felének működnie kell. E mögött az a logika rejlik, hogy ha valami miatt a kommunikáció megszakad a telep két része között, akkor ne tudjon elindulni mindkettő önálló telepként.

Két tagból álló telepek esetén (ami HA-telep esetén tipikus) a fenti szabályt lehetetlen betartani, hiszen ekkor ha az egyik tag kiesik, akkor leáll a telep; márpedig a

HA-telep célja elsődlegesen az, hogy valamely tag esetén a másik átvehesse a szerepét. Ezért ilyenkor a quorum 50% szokott lenni (vagyis egy tag megléte esetén a telep már életképes).

Gyakran van egy ún. quorum-partíció is egy – az összes tag által elérhető – közös lemezterületen, ezen keresztül jelzik a tagok, hogy el tudják érni a telepet.

4. Figyelő csatornák (heartbeat)

Figyelő csatornák segítségével tudja 2 teleptag figyelni egymás elérhetőségét. Az egyik tag rendszeresen jeleket küld a másik felé, ezzel jelezve életképességét. Ha egy ezek a jelek megszűnnek, az azt jelenti, hogy valami probléma van a géppel.

A figyelő csatornák különböző médiákat használhatnak erre a célra. Leggyakoribb a soros vonali és az ethernet-hálózati figyelés, de általában bármilyen TCP/IP-t támogató médián mehet az efféle „életjel”. Akár az is elképzelhető, hogy közösen elérhető lemezterületen jelezzék elérhetőségüket egymásnak a teleptagok. Érdemes legalább 2, különböző típusú egyidejű figyelést alkalmazni, így ha valamelyik meghibásodik (pl. kicsúszik a soros kábel), a másikon keresztül még mindig működik a kommunikáció.

Milyen paraméterezési lehetőségek vannak egy figyelő csatorna esetén? Általában állítható a jelek küldésének „gyakorisága”, továbbá a fogadó tag „érzékenysége”, vagyis hogy hány egymás után következő jel elmaradása után kell megkezdenie a beavatkozást.

5. Node „lelövésének lehetősége” (STONITH, Shut The Other Node In The Head)

A STONITH szintén gyakran használt módszer HA-telep környezetben, különösen akkor, ha osztott lemezes alrendszer elérést is alkalmazunk. Mit is jelent ez? Segítségével egy teleptag le tud kapcsolni valamely másik teleptagot, ha úgy látja, hogy az már nem elérhető.

Miért hasznos ez számunkra? Tegyük fel, hogy van egy kételemű HA-telepünk, amely megosztva használ egy „hagyományos” fájlrendszert (nem telepfájlrendszert!) osztott SCSI-buszon keresztül. Ebben az esetben fontos, hogy egyszerre csak egy tag érhesse el a fájlrendszerünket, hiszen ha egyszerre ketten írnának, az adatsérülést okozna.

Előfordulhat, hogy (pl. lefagyás miatt) egyik tag úgy látja, hogy a másik már nem elérhető, átveszi annak szerepét, és – többek között – felcsatolja a közös fájlrendszert. Fontos, hogy a lefagyott tag ne tudjon újra „feléledni” és írni a – már más által átvett – fájlrendszerre. A STONITH-technológia éppen ezt garantálja számunkra, azáltal, hogy az átkapcsolás első lépéseként lekapcsolja a lefagyott tagot.

Hogyan kerülhet sor a STONITH fizikai megvalósítására, vagyis a géplekapcsolásra (a STONITH-implementációkban ezek többségéhez van támogatás):

- Speciális soros vonali kapcsolókkal (a gép ezen keresztül kapja az áramot) (pl. WTI RPS-10, APC)
- Speciális ethernet hálózati kapcsolókkal (a fentihez hasonló, csak a vezérlés nem soros vonalon, hanem etherneten keresztül történik; ezekkel egyszerre több eszköz is vezérelhető, ez viszont egy HA-telepben SPOF-fé válhat) (pl. WTI NPS-230, APC, NetBay)

- Nagyobb IBM-kiszolgálók esetén a szervízprocesszorhoz tartozó Service Management Porton keresztül
- Intel alaplapok esetén az EMP-port programozásával
- A reset kapcsoló kivezetésével

6. Rendszer működőképesség figyelés (szoftveres ill. hardveres watchdog)

Watchdognak nevezzük azt a mechanizmust, amelynek révén egy gép figyeli a saját magán futó alkalmazás által küldött életjeleket; ha a jelek megszűnnek, a gép újraindítja magát. Ezzel szintén kiküszöbölhető a fentebb, a STONITH-nál már említett, közös fájlrendszer eléréssel kapcsolatos probléma. Mivel mindkét megoldás (STONITH, watchdog) kb. ugyanarra a problémára nyújt valamiféle, egymástól különböző megoldást, az ember vagy egyiket, vagy másikat használja, ízlése és lehetőségei szerint.

Maga a Linux-kernel is biztosít egy watchdog szolgáltatást (insmod softdog, mknod /dev/watchdog c 10 130), amelyet szoftveres watchdognak nevezhetünk. Ezen túlmenően vannak ún. watchdog kártyák, amelyek szintén ugyanilyen elven működnek. Általában elmondható, hogy a hardveres kártyák megbízhatóbbak a szoftveres megoldásnál, hiszen a szoftveres megoldás életképességéhez a kernelnek még valamilyen szinten működnie kell.

7. Erőforráscsoportok (resource group, service)

A HA-telep infrastruktúra általában biztosít számunkra egy olyan keretet, amelyben alkalmazásunkat, a hozzá tartozó adatokat és egyéb erőforrásainkat elhelyezhetjük. Ezt az egyik terminológia szerint erőforráscsoportnak, a másik szerint egyszerűen szolgáltatásnak nevezzük. Mik tartoznak tipikusan bele egy ilyen erőforráscsoportba?

- Alkalmazás indító/leállító scriptek,
- Maga az alkalmazás, a hozzá tartozó adatokkal,
- IP-címek,
- Lemezes eszközök, amelyeken az alkalmazás és az adatok elhelyezkednek,
- Csatolási pontok.

8. Alkalmazásfigyelés (Application monitoring)

Gyakori elvárás egy HA-telepszoftverrel szemben, hogy figyelje a rajta futó alkalmazások állapotát, és probléma esetén indítsa újra az alkalmazást. Ezt a legtöbb HA-szoftver támogatja, vagy tervbe van véve a támogatás elkészítése.

Általában az elterjedtebb alkalmazásokhoz (Oracle, mysql, apache, ...) adnak példa figyelő scripteket, amelyek minimális módosítást igényelnek; egyéb alkalmazások esetén pedig nekünk kell megírni a scriptet. Miből is állhat egy ilyen script? Pl. bejelentkezik egy adatbázisba, lekérdez 1-2 rendszertáblát és ha mindent rendben talál, akkor a megfelelő visszatérési értékkel lép ki.

9. Telep-állományrendszerek (cluster filesystem)

Telep-állományrendszereknek nevezzük azokat a fájlrendszereket, amelyeket egyszerre több teleptag – egyidejűleg! – is használhat. Míg egy hagyományos fájlrendszert egyszerre csak egy gép csatolhat fel (mount), egy osztott fájlrendszerrel ezt többen is megtehetik.

Az utóbbi években jelentős fejlődésen ment keresztül ez a fajta technológia is, bár még általában nincs integrálva a HA-szoftverekbe – viszont különösebb integrációra nincs is szükség, hiszen a fájlrendszer, amit eddig a HA-szoftvernek kellett külön csatolnia erőforráscsoport indítása estén, az most mindig minden teleptagon rendelkezésre áll, tehát nem kell külön foglalkozni vele.

A legismertebb GPL-licenszes implementáció az OpenGFS[5].

Az egyik legfontosabb probléma maga a tény, hogy egyidejűleg több teleptag érheti el ugyanazt a csatolt fájlrendszert, vagyis a – tagok közötti – zárolás (locking) megoldása. Az OpenGFS-ben az egyes tagok memexp moduljai működnek együtt a zárolás folyamatos nyilvántartása érdekében. A zárolásokat egy közös helyen tárolják, ami lehet:

- Egy közös lemezterület (elkülönítve a fájlrendszertől és a naplózástól (journal)), amelyet DMPE-eszköznek (Device Memory Export Protocol) neveznek
- Egy memexpd hálózati démon, amely vagy memóriában, vagy a helyi lemezen tárolja a zárolásokat

Az OpenGFS felhasználók többsége az utóbbit használja, bár szerintem az előbbi HA-szempontból jobb megoldásnak tűnik, hiszen nincs egyetlen olyan kitüntetett gép, aminek kiesése esetén a zárolási információk elvesznek.

10. Összehasonlító adatok

Az 10 táblázatban néhány ismertebb, GPL-licenszelésű HA-szoftver tulajdonságait hasonlítjuk össze, a fenti szempontok szerint.

Jellemzők	Heartbeat	Kimberlite	RH Cluster Manager	Failsafe
Quorum van?	nem	igen	igen	nem
Quorum-eszköz van?	nem	igen	igen	nem
Figyelő csatornák	igen	igen	igen	igen
STONITH támogatás	igen	igen	igen	igen
watchdog támogatás	igen	nem	igen	nem
Erőforrás csoportok	igen	igen	igen	igen
Alkalmazás figyelés	nincs	nincs	van	van

1. táblázat. HA-szoftverek összehasonlítása funkcionalitás szerint

11. Összefoglalás

A fenti technológiák alkalmazásával komplett HA-megoldások építhetők. A HA-telepszoftverek fejlődése során – egyre újabb és újabb technológiák beépítése révén – a

végső cél egy SSI-típusú telepszoftverré válás lehet, amely egyetlen nagy rendszernek láttatja a telepet és egyaránt rendelkezik a HA és a terhelésmegosztó képességekkel.

Hivatkozások

[1] http://oss.missioncriticallinux.com/projects/kimberlite/

[2] http://www.linux-ha.org/

[3] http://www.redhat.com/

[4] http://oss.sgi.com/projects/failsafe/

[5] http://opengfs.sourceforge.net/

Hálózatfelügyeleti megoldások Linuxon

Deim Ágoston

2003.11.08.

Kivonat

A cikk célja bemutatni néhány nyílt forrású, hálózati távfelügyeleti szoftvert, valamint azt, hogy egyszerű eszközökkel is milyen hatékonyan felügyelhető egy hálózat.

Felhívjuk a távfelügyeleti szoftverek hátrányaira és előnyeire a figyelmet, ahol szükséges, ott tanácsot adva. A szoftvereket módszereik alapján is osztályozzuk – démon vagy adott időnként futó programok –, különbségeiket, előnyeiket és hátrányaikat elemezzük.

A szabad szoftverek közül először egy teljesen kidolgozott, professzionális programot, a Nagiost [1] (a Netsaint utóda) vizsgáljuk meg, külön kiemelve annak webes és wap-os rendszerét. Alternatív megoldásként röviden ismertetésre kerül a Big Sister [2] alkalmazás is.

Tartalomjegyzék

1. Mi is a hálózatfelügyelet?

Hálózatfelügyelet alatt kiszolgálók és kliensgépek felügyeletét értjük, melynek célja, hogy folyamatosan elérhetőek és használhatók legyenek az azokon kiajánlott szolgáltatások. Ennek során figyelni kell az operációs rendszert éppúgy, mint a futó szolgáltatásokat és a használt hálózatot.

2. Hálózatfelügyeleti megoldások

Az alábbiakban vizsgáljuk meg a leggyakrabban használt megoldásokat, azok előnyeit és hátrányait is számba véve. A két használt megoldás a felügyelt szolgáltatásokhoz történő csatlakozás, míg a másik kliens – démon módszer szerint megszerzett információk elemzése. A távolról, közvetlen csatlakozással működő felügyeletet nevezik *aktív ellenőrzésnek*, míg a felügyelt gépen futó alkalmazások eredményeit feldolgozó ellenőrzést *passzív ellenőrzésnek* nevezzük.

2.1. Csatlakozás távoli kiszolgálóhoz

Ez a legnépszerűbb módszer, mivel könnyen ellenőrizhetők a futó szolgáltatások anélkül, hogy bármit is kellene futtatni a távoli rendszeren. Ennek a módszernek két hátránya van. Az első, hogy nem képes információt adni távoli rendszer erőforrás-kihasználtságáról és terhelési adatairól, mint például a processzor terhelés, merevlemez telítettség, szabad memória mennyisége. A másik hátrány, hogy ha a felügyelt számítógép és a felügyeletre használt számítógép között tűzfal helyezkedik el, akkor nincs más lehetőségünk, mint a felügyeleti gépről érkező forgalom számára utat nyitni a tűzfalon. Van aki idegenkedik ettől a megoldástól. Hogy okkal vagy ok nélkül, azt mindig az adott helyzet ismeretében lehet megítélni.

A felügyeleti módszere a következő: általában egy központi számítógépről időzítetten elindulnak a felügyeleti programok, melyek a távoli gépen futó szolgáltatásokhoz csatlakoznak, ellenőrizve állapotukat. A hibakezelésre általános érvényű, hogy egy jó minőségű szoftver nem csak a felhasználói felületen üzen, hanem minimum e-mailben is értesítést küld a hibáról, de képes lehet például sms-en keresztül üzenni – persze csak akkor, ha megfelelő eszköz van a kiszolgálóhoz csatlakoztatva.

2.2. Távoli gépen futó alkalmazás

Egy távoli gépen futó alkalmazás rengeteg olyan dolgot képes közölni, amire a távoli ellenőrzés nem képes. Ilyen az előzőekben említett erőforrás-kihasználás minden jellemzőjének megjelenítése. Működésüket tekintve a szoftverek egy része kliens és démon keveréke, míg másik részük csak démon. A kevert működésű szoftverek az adatokat helyileg gyűjtik, majd kliensként egy – vagy több – központi géphez csatlakozva küldik át az adatokat egy ott lévő démonnak, mely elemzi az adatokat. A szoftverek másik körébe tartoznak azok, melyek csak démonként futnak, a központi géptől csatlakoznak rá és kérik le az adatokat további elemzésre. Itt általános érvényű, hogy érdemes titkosítani a felügyelt számítógép és a felügyeleti szerver közötti kommunikációt. Ezek a szoftverek képesek elkerülni az előzőleg ismertetett módszer hátrányait.

2.3. A kompromisszumos módszer

Azért, hogy lehetőleg mindkét módszer előnyeit ki tudják használni a rendszergazdák, és pontos képet kapjanak a kiszolgálók teljes állapotáról, a szoftverek írói vegyesen kezdték alkalmazni a módszereket, így adva teljes megoldást a problémákra. Túlsúlyban vannak a távoli megfigyelések adatai, de megtalálhatók kiegészítésként a felügyelt gépen futó szoftverek is.

3. A Nagios

3.1. A „sikersztori”

A legnépszerűbb felügyeleti eszköz a NAGIOS[1], mely hosszú évek óta jelen van a nyílt forrású szoftverek között. Nem is olyan régen még NetSaint-nek hívták, azonban az 1.0-s verziót elérve a vezető programozó úgy döntött, hogy a rengeteg változás miatt új nevet ad a szoftvernek, mivel az szinte nem is hasonlított a régi verziókra.

Miért népszerű ez az eszköz? Fut az összes jelentősebb kiszolgáló rendszeren, mint amilyenek a Linux disztribúciók, a BSD-k, vagy akár az Apple UNIX és FreeBSD alapú szervertermékén, az XServer + MacOS X pároson is. Kiegészítőként kapható a Nagios-szal tökéletesen integrálható hardver, melynek segítségével többek között nyomon követhető akár a szerverszoba hőmérséklete és sok más adat is. A szoftver maga pedig tökéletesen moduláris és nem elhanyagolható előny a kitűnő webes felület, melyet hatékonyan egészít ki a WAP-os felület, hogy minden igényt kielégítsen.

Lehetőségeinek rövid összefoglalója:

- szolgáltatások távoli felügyelete, kiszolgálók erőforrásainak megtekintése,
- adatbázis-kapcsolat a konfigurációs fájlok tárolására,
- csoportosítható szolgáltatás és kiszolgáló felügyelet,
- szintén csoportosítható felügyeleti személyek, csoportok,
- értesítés nem csak e-mailen, de akár sms-en vagy azonnali üzenetküldő szolgáltatáson keresztül mint például az ICQ,
- korszerű bővítménykezelés további szolgáltatások felügyeletére,
- eseménykezelő rendszer a problémák azonnali elhárítására,
- Web és WAP-alapú megjelenítési rendszer,
- hardver integráció,
- elosztott felügyelet,
- klaszterek felügyelete.

3.2. A Nagios felépítése és működése

Miből is áll a Nagios?

A Nagios tulajdonképpen egy mag, mely önmagában nem lenne használható. Ez a mag nem képes szolgáltatások ellenőrzésére, adatok elemzésére, egyszerűen „csak” a megkapott adatokat jeleníti meg illetve reagál ezek szerint. Hogyan lesz mégis használható? A munka legnagyobb részét a modulok végzik, melyek beépülnek a Nagiosba, ezek ellenőrzik a szolgáltatásokat és jelentenek a Nagiosnak. Ezeken kívül, gyakorlatilag a legfelső szinten jelenik meg a Web és WAP-alapú felület, ahol a legjobban értelmezhető módon, vizuálisan jelenik meg az információ. Opcionális a hardver elem, mely szintén bővítmény használatával adja át az adatokat. Szintén külön osztályt alkotnak a *kiegészítők*, melyek nem a felügyeleti gépen futó alkalmazások – vagy csak részben azok – azonban mégis a Nagios-hoz tartoznak és a távoli gépek erőforrás-kihasználtságát vagy a szolgáltatások állapotát közvetítik.

A bővítményekről

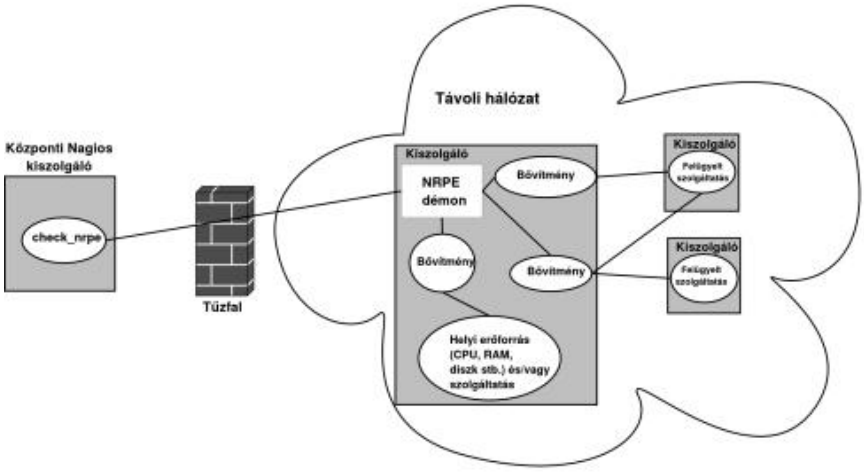
A bővítményeknek a Nagios által elvárt módon, bizonyos szabályok betartásával kell adatokat közölniük. A bővítmények számára külön oldalt hoztak létre, itt megtalálható a teljes leírás, hogy milyen elvárásoknak kell megfelelnie annak, aki azt szeretné, hogy bővítménye a hivatalos csomagba kerüljön. A projekt irányítóinak felhasználó/rendszergazda-barátságáról sokat elárulhat, hogy nem lehet a csomag része olyan alkalmazás, mely nem rendelkezik *--help* kapcsolóval. Ezenkívül a legfontosabb meghatározás, hogy tartani kell magát mindenkinek egy egységes kódtáblához, mely a felügyelt szolgáltatás állapotára vonatkozik és ettől eltérni nem lehet. A bővítmények bármilyen programozási nyelven íródhatnak, előnyben részesítik azonban a lefordítható programokat az interpretáltakkal szemben, mely teljesítmény adatokkal magyarázható – erről részletesen később –, illetve a Nagiosba beépített, módosított Perl értelmező számára írt alkalmazásokat kedvelik. Természetesen használható akár kedvenc shell szkriptünk is, ha megfelel a követelményeknek.

A kiegészítőkről

A két legelterjedtebb kiegészítő az *NRPE* és az *NSCA*. Működési elvük szerint mindkettő hasznos lehet, ha olyan adatokhoz szeretnénk hozzáférni, melyeket távolról nem tudunk ellenőrizni – tehát nem szolgáltatások, hanem erőforrások –, vagy nincs lehetőségünk közvetlenül a szolgáltatásokhoz kapcsolódni. Mindkét alkalmazás tartalmaz egy démont és egy klienst is, azonban fordított logikával működnek. Vizsgáljuk meg, hogyan.

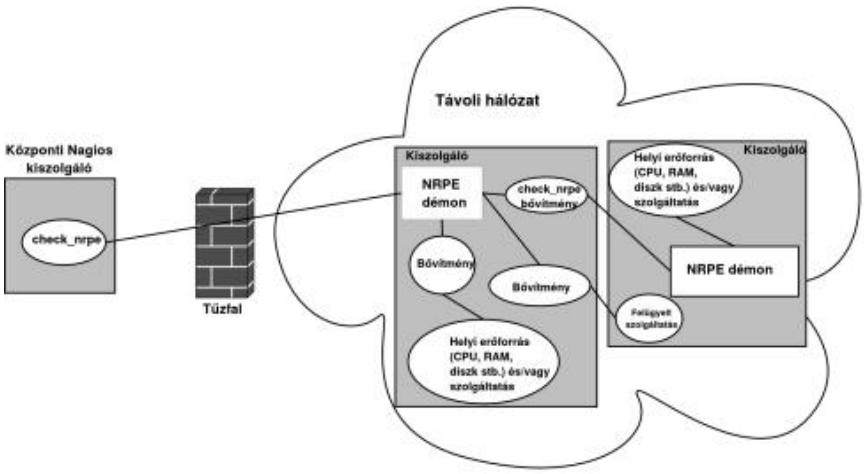
Az NRPE démon neve – mely rövidítés – is sokat elárul már. Magyarul és kifejtve a következőt jelenti: Nagios Távoli Bővítmény Végrehajtás. Ez gyakorlatilag egy közbülső réteg. Egy külön felügyeleti gépen is futhat a démon – inetd vagy xinetd alól is indítható – és időnként ehhez kapcsolódik a központi felügyeleti gépről a check_nrpe program, hogy ott olyan módon hajtsa végre a meghatározott bővítmény(eke)t, mintha azt a felügyeleti gép tenné meg. Az NRPE démon végrehajtja a szükséges bővítményeket, majd az eredményeket a check_nrpe programnak adja át, mely továbbadja azt a Nagios központi részének a megfelelő kóddal és program kimenettel. Az NRPE démonnak megszabható, hogy mely IP címekről fogadjon el kapcsolatot. Kommunikációs része támogatja a titkosítást. Az NRPE démon rendelkezik Microsoft platformon futó

verzióval, melyet szolgáltatásként kell ott futtatni. Az NRPE tehát tökéletesen alkalmas arra, hogy egy tűzfallal elzárt alhálózaton kinevezzünk egy központi gépet, erre telepítsük az NRPE démont, majd beállítsuk úgy, hogy ez ellenőrizze az alhálózatban lévő többi gépet és az eredményeket visszaadja a központi Nagios kiszolgálónak. Ezt szemlélteti az 1. ábra. Ezzel csak egyetlen csatornát nyitunk a tűzfalon, ahol a kom-



1. ábra. NRPE-Nagios kapcsolat

munikációt remélhetőleg titkosítottan végezzük. Természetesen nyugodtan futtatható egyetlen gépen is az alkalmazás. További lehetőség az ilyen kiszolgálók láncba fűzése, melyet a 2. ábra mutat be.



2. ábra. Kiszolgálók láncba fűzése

Az NSCA szintén több részből áll, itt azonban a kliens rész az, mely a felügyelt

gépen összegyűjti az adatokat a konfigurációs fájl alapján és elküldi azt a központ Nagiost futtató kiszolgálónak, ahol az NSCA démon feldolgozza az adatokat és továbbítja azt a Nagios számára. Amennyiben a résztvevő feleken telepítve van az *mcrypt* függvénykönyvtár a kommunikáció titkosítottan is történhet. Ennek a kiegészítőnek a segítségével végezhetünk *passzív* ellenőrzéseket.

Konfigurációs lehetőségek és osztályzásaik

A Nagios rendkívül sok konfigurációs lehetőséggel rendelkezik, de a konfigurációs állományokat is több részre osztja. Ezek: a központi állomány, az objektumok – ezek felelnek a szolgáltatásokért, kiszolgálókért is, mivel ezek az objektumok –, a CGI (webes) felület beállító fájlja és más kiegészítő fájlok. A központi állományban hivatkozhatunk több, különálló konfigurációs állományra, így mindent külön tudunk kezelni, ezzel is növelve az átláthatóságot. A legjobb lehetőség azonban a *sablonok* használata. Segítségükkel rendkívüli mértékben lecsökken a szolgáltatások felügyeletének konfigurációs ideje és karbantarthatóbbá válik az egész struktúra.

Hogy megtapasztaljuk milyen hasznos egy ilyen sablon, alakítsuk ki a következő struktúrát:

```
define kiszolgalok{
check_command check-host-alive
notification options d,u,r
max_check_attempts 5
name kiszolgalosablon
register 0
}
define host{
host_name www.belsohalo.hu
address 192.168.0.126
use kiszolgalosablon
}
define host{
host_name www2.belsohalo.hu
address 192.168.1.253
max_check_attemtps 2
use kiszolgalosablon
}
```

Ez kifejtve – a Nagios szemszögéből – a következőképpen néz ki:

```
define kiszolgalok{
check_command check-host-alive
notification options d,u,r
max_check_attempts 5
name kiszolgalosablon
register 0
}
define host{
host_name www.belsohalo.hu
address 192.168.0.126
check_command check-host-alive
notification options d,u,r
max_check_attempts 5
}
define host{
host_name www2.belsohalo.hu
address 192.168.1.253
chec\_command check-host-alive
notification options d,u,r
max_check_attemtps 2
```

```
}
```

Mi derül ki a fenti mintából? Mint látható a sablonok egymásba is ágyazhatók és a lokális változók felülbírálják a sablonban beállítottakat. A fenti példa sablonban a max_check_attempts értéke 5, melyet át is vett az első kiszolgáló, de második beállításnál helyileg felülbíráltuk azt, így 2 lett az értéke.

A fentieken túl a sablonokkal – vagy akár azok nélkül is – még csoportokat is alakíthatunk szolgáltatások vagy kiszolgálók szerint is beállítva. Bővebb példákat találhatunk a Nagios terjesztésében és dokumentációs kézikönyvében.

3.3. Hibák meghatározása

A felügyelt szolgáltatásoknál fontos kérdés, hogy ha nem tudunk egy adott szolgáltatásról információt szerezni, akkor az a kiszolgáló, az adott szolgáltatás vagy a hálózat hibája? Ha egy kiszolgálón futó szolgáltatás hibáját érzékeli a Nagios, akkor megvizsgálja, hogy a kiszolgáló egyáltalán működik-e? Amennyiben nem, akkor felesleges a szolgáltatások hibájáról értesítést küldenie, hiszen mindegyik „hibás” lesz. Amennyiben a kiszolgáló működik, kiküldi a hibajelentést. Ha elérhetetlen a kiszolgáló, akkor felmerül a kérdés, hogy hardver vagy hálózati hiba okozza-e a kiesést? A Nagios számára a hálózat felépítése a következő: amelyik felügyelt géppel azonos hálózaton van illetve közvetlenül eléri azt, azt helyi gépnek tekinti. Minden más felügyelt gép távoli. Miért fontos ez? Mert ha több számítógépen keresztül érjük el a felügyelt kiszolgálókat, akkor lehetséges, hogy egy közbülső, összekötő gép „esett ki”. A gépek közötti reláció meghatározására a Nagios bevezette a számítógépek közötti függőségeket és a szülő – gyermek kapcsolatot. Ez által, ha kiesik egy kiszolgáló, de már a szülője sem érhető el, akkor feltételezhetjük, hogy a szülő kiesése okozta a problémát. Ha több úton is el tudunk jutni egy adott szolgáltatásig, akkor ennek segítségével még könnyebben meghatározható, hogy valóban egy közbülső számítógép esett-e ki. A hibák meghatározását elvégzi számunkra a Nagios, ennek eredményét megtekinthetjük a webes felületen.

3.4. Állapotok

Amennyiben hiba lép fel, akkor felmerül a kérdés, hogy miként is osztályozhatók ezek a hibák. A Nagios két hibatípust ismer.

Soft: az enyhe hiba kategória, amikor még nem bizonyosodott be teljesen, hogy hibás a szolgáltatás, valamint amikor a szolgáltatás vagy kiszolgáló visszatér az enyhe hiba állapotából.

Hard: súlyos hiba, amikor bebizonyosodott, hogy a szolgáltatás vagy kiszolgáló hibás, illetve amikor ebből az állapotból tér vissza enyhébb kategóriába, azaz „meggyógyult”.

Az, hogy mikor válik bizonyossá egy szolgáltatás vagy kiszolgáló hibája, konfigurációs paraméterek beállításával szabályozható. Ezekhez is segítséget nyújt a Nagios dokumentációja.

3.5. Eseménykezelők

Amennyiben a Nagios valós hibát észlel, akkor még az értesítés előtt megpróbál proaktív módon beavatkozni. Erre valók az eseménykezelők. Amennyiben azonos gépen fut

a Nagios és a szolgáltatás, vagy más módon van lehetőségünk beavatkozni, akkor egy eseménykezelővel rengeteg dolgot el tudunk érni. Futtathatunk további ellenőrzéseket és a válaszok alapján dönthetünk. Akár újra is indíthatjuk a szolgáltatást, hogy a szolgáltatás kiesési időt minél lejjebb szorítsuk. Ha egy szolgáltatáshoz eseménykezelő is hozzá van rendelve, akkor az *mindig* előbb hajtódik végre, mint a riasztás. Miért? Egyrészt a már említett proaktív mód miatt, másrészt ezzel a hibáról küldött üzenetben benne lehet, hogy már el is lett hárítva – amennyiben sikeres volt az esemény kezelése.

3.6. A riasztásokról

A Nagios minden olyan esetben riasztást küld ki amikor a hiba súlyos minősítést kap vagy a súlyos hiba minősítést kapott szolgáltatás/kiszolgáló adott – konfigurálható – időn belül nem kerül enyhe hiba vagy minden rendben állapotba. Ki és hogyan kap ekkor értesítést? Az adott szolgáltatáshoz vagy kiszolgálóhoz rendelt személy(ek). Azonban mielőtt egy levél kimenne a következő ellenőrzések zajlanak le:

- Senki nem kap meg kétszer egy üzenetet.
- Engedélyezve van-e a riasztások küldése globálisan?
- A szolgáltatás/kiszolgáló „ugrál”, azaz váltogatja az állapotát, ha igen, akkor nincs riasztás!
- Ellenőrzi az állapotokra vonatkozó beállításokat, mikor kell értesítést küldeni?
- Engedélyezett időpontban küldi-e a riasztást? Ha nem, akkor az első érvényes időpontra tolja azt.
- A megadott intervallumon belül esik-e a riasztás, azaz nem túl gyakori-e?

Azonban az, hogy ezeken a szűrőkön „átment” egy riasztás, nem jelenti azt, hogy minden, az értesítési listában szereplő személy megkapja, mivel személyenként is beállíthatók a fenti szűrések. Az a riasztás azonban, mely képes minden szűrést kikerülni, a következő módokon juthat el a címzettekhez:

- SMS
- Pager
- ICQ, Yahoo, MSN(!) azonnali üzenetküldés
- Hang alapú riasztás
- stb.

3.7. Időzített kimaradások

Természetesen a Nagios képes kezelni az olyan eseteket, melyben mi magunk állítjuk le a szolgáltatásokat és a kiszolgálókat. Ha tudjuk előre, hogy le fog állni, miért is küldjön értesítést hibáról? Minden ilyen időzített leállítást beállíthatunk előre. Ez pontosabb statisztikákat is eredményez – például nem jelennek meg az akár több órás időzített kiesések hibaként.

3.8. Web felület

A Nagios rendkívül jól „felszerelt” webes felülettel rendelkezik, melynek segítségével egy böngészőn keresztül nyomon tudjuk követni a szolgáltatások és kiszolgálók állapotát, készíthetünk kimutatásokat rengeteg más lehetőség mellett. Érdemes ezen keresztül nyomon követni a szolgáltatások állapotát.

3.9. Összefoglalás

Bár a fenti összefoglaló sok kérdésben segít, de terjedelménél fogva nem képes mindenre választ adni, nem is volt célja. Rengeteg egyéb lehetőség van a Nagios-ban, mint az elosztott felügyelet, melyben több Nagiost fogunk össze, klasztereket és más redundáns szolgáltatásokat felügyelve. A teljes referencia lista eléréséhez letölthető a Nagios mintegy 300 oldalas angol nyelvű kézikönyve.

4. A kihívó: Big Sister

A Nagios nem egyedül tartózkodik a felügyeleti szoftverek piacán. Rengeteg más lehetőség mellett a Big Sister-t[2] emelhető ki, mely az időközben kereskedelmi termékké vált Big Brother szoftver nyílt forrású folytatása. Sok vonzó tulajdonsággal rendelkezik, melyek közül itt is fontos a webes felület, a szolgáltatások megfigyelhetőségének sokfélesége. Hátránya, hogy a Nagios-hoz képest sokkal kevesebb rendszeren működik. Nem szabad azonban lebecsülni, rendelkezik például Oracle ellenőrző modullal is. A webes felület jól átgondolt, informatív. Felépítése nagyon hasonlít arra, ahogyan a Nagios az NSCA kiegészítővel működik. Két részből áll: a *Status Collector*-ból illetve egy vagy több távoli gépen futó *Agent*-ből. Ez utóbbi a lokális gépről gyűjti annak erőforrás-adatait, a környező gépekről pedig a szolgáltatások adatait és időnként ezt elküldi a Collector-nak. A Collector elemzi, tárolja és megjeleníti az adatokat, valamint riasztásokat küld, amennyiben szükséges. Jelenleg a következő szolgáltatásokat és erőforrásokat képes monitorozni (kivonat): Oracle, TripWire, DNS, HTTP, LDAP, CPU terhelés, memória állapota, szabad lemezterület, UPS állapota stb.

5. Összefoglaló

Bár most csak két szoftver került bemutatásra, de sokkal több nyílt forráskódú felügyeleti ezsköz létezik. A részletesen bemutatott Nagios-t tanulmányozva felismerhetjük, hogy a nyílt forrású hálózatfelügyeleti eszközök rendkívül erősek, átgondoltak. Ha valamilyen szolgáltatáshoz nem érhető el bővítmény, azt saját magunk számára megírhatjuk és beilleszthetjük a Nagios rendszerébe, gyakorlatilag végtelenítve annak lehetőségeit és megkönnyítve saját életünket. Nem maradt más hátra, mint jó használatot kívánni a felhasználáshoz!

Hivatkozások

[1] `http://www.nagios.org/`

[2] `http://bigsister.graeff.com/`

Csoportmunka/Együttműködés/Projekttámogatás linuxos eszközökkel

Fejős Tamás <tms@dunaferr.hu>

2003.11.08.

Kivonat

Rövid bevezetés és a megvalósítandó feladatok felvázolása után, bemutatok néhány eszközt. Ismertetem ezek legfontosabb előnyeit, hátrányait. Röviden összehasonlítom a Windows platformon elérhető lehetőségeket a Linuxon elérhetőkkel.

Érintett szoftverek: TUTOS[1], phpGroupWare[2], PHProjekt[7], Lotus Domino/Notes, MS Exchange, MS Project. Részletesen ismertetem a Linuxos megoldások előnyeit, hátrányit. Gyakorlati útmutatóként szolgálnak egy konkrét rendszer megvalósítási lépései.

Bemutatom a kapcsolódási pontokat, felhívom a figyelmet, hogy mely más előadások nyújthatnak hasznos segítséget a témában. Az érdeklődőknek az elinduláshoz „kapaszkodóként" információforrásokat is megadok.

Tartalomjegyzék

1. Bevezetés

Képzeljük el, hogy 15 ember együtt kíván dolgozni munkacsoport vagy projekt keretében. Adott tehát egy lelkes csapat melynek célja, hogy a Debian terjesztés honosítása elkészüljön (Debian Honosító Projekt – DHP). Munkájuk során rengeteget fognak fordítani és rengeteget fognak szervezni, egyeztetni, hogy a szükséges erőforrások rendelkezésre álljanak. Néhány aktivista Budapesttől távolabb lakik és csak ritkán engedheti meg magának, hogy felutazzon a megbeszélésekre, vagy hosszadalmas telefonkonferencián vegyen részt. Emiatt más kommunikációs csatornára van szükség. A személyes kontaktus ötletek, projektek kidolgozásakor továbbra is sokszor hasznos.
Emeljünk ki néhányat munkájuk eredményességének feltételei közül:

- szakmai tudás,
- megfelelő kommunikáció, információáramlás,
- elegendő idő a munkához,
- elegendő pénz az eszközök beszerzésére.

Nézzük közelebbről, hogy mit is takar a megfelelő kommunikáció.

1.1. Információ vagy utasítás: áttekinthetően, visszakereshetően és számon kérhetően

A projektben résztvevők mindennapi munkájuk során gyakran akarnak *megosztani egymás közt információkat* (pl. egy esemény időpontja, megbeszélések eredménye), *továbbítani kéréseket* (hozd el a csomagot a postáról, állítsd össze a pályázati anyagot), *ötleteket* (vegyünk részt az idei Linux konferencián). Mindez egyfajta kommunikáció, méghozzá olyan, amelyben az elhangzottak *rendszerezetten megőrzendőek az áttekinthetőség és a gyors visszakereshetőség* érdekében. A DHP-nál maradva példaként nézzük annak egy részprojektjét. A hetekben volt ez első hazai DDTP (Debian Csomagleírás Fordító Parti). Tallózzunk a gördülékeny munka feltételei között! Egy adott munkatárs – hívjuk őt Miklósnak – az esemény szervezése folyamán:

- mindig *időben* tudja, hogy mik a teendői,
- ha *információra van szüksége* (pl. egy telefonszám, következő találkozó időpontja és helye, az eddigi megbeszélésekről készített összefoglalók, teendők, ötletek, javaslatok) *anélkül hozzájuthasson, hogy ezzel másokat megzavarna*,
- munkatársai, „felettesei” *naprakészen tudják*, pillanatnyilag *min dolgozik*, és *hol tart* vele, hogyan tudják előremozdítani ügyeit. Ha nem végez határidőre, *számon kérjék* tőle, esetleg átadják másnak a munkát,
- mindez a lehető *legkevesebb ráfordítást* igényelje.

Látni és láttatni, Miklós hozzájut munkájához szükséges információkhoz (pl. hol lehetne az összejövetelt tartani, lehetséges támogatók elérhetősége), valamint másokat információval lát el (pl. saját tevékenységéről, mikor és hol lesz az összejövetel, a várható programok, milyen segítségre van szüksége). Mindez akkor kezd izgalmas lenni, amikor már nem 1-2 ember, hanem 6-8 próbál együtt gondolkodni és dolgozni. Döntő kritérium, hogy mindezt a *lehető legkevesebb adminisztrációs munkával* kívánjuk elérni (hiszen az egy lelkes aktivistának is nyűg, hát még egy hajszolt programozónak,

rendszergazdának). Ahol lehet, ott *egy adatot csak egyszer vigyünk fel* és a rendszert a felhasználók *elfogadják*. Ha nem látják át, és ez összezavarja őket, vagy az adatbevitel elveszi a(z önkéntes) munkavégzésre fordítható idejük nagy részét, ne csodálkozzunk rajta, ha megutálják, elutasítják, nem használják, és a belső kommunikációs problémák megmaradnak.

A hagyományos csatornák (pl. élő vagy telefonos beszélgetés, levél, FAX) sok esetben nem teljesítik az előbb említett kívánalmakat. Archiválásuk, visszakereshető rendszerezésük, egymás közötti megosztásuk nehézkes, főleg, ha az érintettek földrajzilag távol vannak.

2. Szempontok és lehetőségek

2.1. Funkciók, szolgáltatások

Egy csokorra való abból, amit a szabad PMS-ek ma nyújtanak:

- Címjegyzék,
- csoportok kezelése,
- projektkövetés,
- feladatlista,
- hibakövetés, (hibajegyek),
- levelezés (webmail),
- időnyilvántartás (timetracking),
- erőforrások kezelése,
- költségelszámolás,
- jegyzet készítés,
- fájlkezelő, (dokumentumok, kapcsolódó anyagok tárolásához),
- kereső, full text ill. teljes site,
- értesítések automatikus küldése

Néhány „extra” funkció:

- Chat,
- fórum,
- installáció követés (nyomon követhető vele, hogy mely termékünket, hol telepítettük, milyen hibák voltak ott),
- telefon napló,
- csoportos könyvjelzők,
- szavazó modul,

- webtartalom-kezelő,
- tudásbázis,
- webboltkezelő,
- FAQ (GYIK) modul.

2.2. Ár

A kereskedelmi szoftverek költsége 1-2 felhasználóra még nem olyan megterhelő, de több tucat licenc beszerzése már komolyabb akadályokba ütközhet, mint ahogy van is rá nem egy példa. A szabad szoftverek „árai” viszont önmagukért beszélnek. . . :)

2.3. Ami még hosszú távon fontos lesz

- Rugalmasság, bővíthetőség. A szabad szoftverekhez a forráskód is elérhető így bármikor hozzáfejleszthetünk. Egyre több a PHP programozó.
- Biztonság. Az elérhető forráskód miatt bármikor auditálhatjuk a forráskódot. Szempont az alkalmazáson belül megvalósított jogosultsági rendszer is.
- Egyszerű, intuitív használat. Ne igényeljen „pilótavizsgát” egy új projekt vagy feladat létrehozása, mert akkor csak a potenciális felhasználók igen szűk köre lesz hajlandó bajlódni vele. Fontos a jó dokumentáció, ami ideális esetben honosított és az általunk használt szoftver változathoz (verzióhoz) készült.
- Gyorsaság. Fontos, hogy a felhasználó minél kevesebbet várjon a képernyő előtt ülve, míg megjelenik az általa kért adat.
- Hatékony adminisztráció .
- Honosítottság. Nem tudhat mindenki angolul, németül, franciául vagy éppen hollandul.

Az egyes szoftvereket „gyári” állapotban vizsgáltam, hozzáadott fejlesztések nélkül.

A szoftverek kipróbálásakor javaslom az alábbi példát végig csinálni. Alaphelyzet: adott egy közösség, vannak céljai, melyeket projekteken keresztül valósít meg, a projekteken belül vannak feladatok, az egyes feladatokon belüli tennivalókat hibajegyekként kezelve végzik el. A DDTP-nél maradva először egy ddtp nevű csoportba felveszünk felhasználókat. Létrehozunk ddtp néven egy projektet, (megírjuk a projektalapító okiratot), definiálunk feladatokat, ezeken belül hibajegyeket (melyek még hiányzó vagy hibás részfeladatok emlékeztetőiként is felfoghatók). Természetesen szeretnénk, hogy minderről az érintettek e-mailben értesítést kapjanak. Közben a publikus híreket közzétesszük a weben. A projekt tagjai a néhány dokumentum szerkesztését közösen végzik. Majd eldönthetjük mennyire felel meg a felvitel módja, a későbbi megjelenítés, módosíthatóság, visszakereshetőség. Átlátjuk a futó projekteket, feladatokat, hibajegyeket?

-	TUTOS	phpGroupWare	PHProjekt
Ár	ingyenes	ingyenes	ingyenes
Verzió	1.1.20030715	0.9.14	4.0.4
További info	`www.tutos.org`	`www.phpgroupware.org`	`www.phprojekt.org`
Funkciók	projektkezelés első fokon		a sokoldalú eszköz
Projektkövetés	kiválóan alkalmas	alkalmas	alkalmas
Bővíthetőség	igen	igen	igen
Intuitív?	igen	igen	igen
Kliens	böngésző	böngésző	böngésző
Honosított?	folyamatban	n.a.	igen
Dokumentáció	bőséges	bőséges	bőséges
Saját API?	igen	igen	igen
LDAP?	igen	igen	igen
Egyéb Integráció	adat import		Postnuke, adat export-import
Próba?	online	helyi install	online

-	Lotus Domino	Exchange + Outlook	MS Project
Ár	túl magas	túl magas	túl magas
Verzió	6.0	2000	2000
További info	`www.lotus.com`	`www.microsoft.com`	`www.microsoft.com`
Funkciók	kommunikáció, projekt	kommunikáció	projekt
Projektkövetés	TeamRoom	n. a.	alkalmas
Bővíthetőség	igen	igen	kevésbé
Intuitív?	igen	igen	igen
Platform	Linux, Windows	Windows	Windows
Kliens	natív + böngésző	natív	natív + böngésző
Honosított?	igen	igen	n.a

3. A szabad szoftveres megoldásokról bővebben

Az igények és a lehetőségek mérlegelése után anyagi (és „világnézeti” :)) okok miatt végül a DHP szabad szoftvert választ, amely eleinte kényelmetlenebbnek, kisebb tudásúnak tűnhet fizetős társainál. Tény azonban, hogy az e területen beszerezhető kereskedelmi alkalmazások gyakran többet tudnak még annál is, amire szükség van. (Tipikusan az „ágyúval lövünk verébre” eset.) Másik példával élve: nekünk csak egy pohár tejre van szükségünk, mégis az egész tehenet meg akarják vetetni velünk.

3.1. A PMS rendszer kiépítése

Tekintsük át vázlatosan a Project Management System kiszolgáló kialakításának lépéseit.

- a vezetés támogatásának megszerzése (e nélkül gyakorlatilag használatba vehetetlen),
- a hardver beszerzése, elhelyezése,
- hálózati kapcsolat biztosítása,
- operációs rendszer (pl. Debian GNU/Linux) telepítése és beállítása,
- webkiszolgáló (Apache) telepítése (a megfelelő modulokkal) és beállítása,

- PMS-be kezelendő felhasználók, projektek, feladatok, címek, ügyfelek összegyűjtése (legyen egy alap az induláshoz) ,
- PMS szoftver kiválasztása, mérjük fel alaposan igényeinket, szükségleteinket. Milyen modulokat tudnánk használni, mi az elengedhetetlen, és eszerint döntsünk (pl. vegyük figyelembe, hogy az adott termékhez támogatás elérhető-e),
- a választott adatbázis-kezelő telepítése, beállítása,
- PHP4 telepítése (igényeinknek megfelelő kiterjesztésekkel), beállítása,
- PMS telepítése, beállítása,
- a PMS feltöltése adatokkal.

Ugyanez ügyfél oldalon

- webböngésző biztosítása,
- oktatás (ez általában hiányos vagy elmarad és nem képesek a felhasználók megfelelően kezelni a rendszert).

A böngészőben a nyelvi preferenciák legyenek beállítva, azt használják a programok. Nálam az elsődleges a magyar [hu], majd az angol [en] (amíg nem így volt a tesztelés során angol, ill. magyar helyett német nyelvű oldalakkal is találkoztam).

4. TUTOS

Átlátható nyitóoldal. Nagyon jól strukturált, amit tud azt mélységekbe menően, viszont hiányoznak belőle bővítő funkciók, „extrák”. Interneten történő használata is gyors, 1-2 helyen adatbevitelnél apróbb kényelmetlenségek előfordulhatnak. Az LME-hez hasonló szervezeteknek ajánlom, ahol a többi szoftver extráira valójában nincs szükség (mert nem kell, vagy az adott funkció másképpen lett megoldva). Az LME jelenleg ismerkedik a rendszerrel.

4.1. Előfeltételek

- web kiszolgáló: Apache,
- adatbázis kiszolgáló,
  - PostgreSQL,
  - MySQL,
  - Oracle,
  - Borland Interbase 5.
- PHP 4 (minimum php 4.1.0.),
  - a választott adatbázis-kezelő támogatásával,
  - igény szerint LDAP kiterjesztéssel (természetesen LDAP kiszolgáló is szükséges),
  - igény szerint PDF kiterjesztéssel,
  - igény szerint JGRAPH kiterjesztéssel,
- igény szerint overlib javascript csomag.

4.2. Funkciók

- Feladatok listája,
- költségek könyvelése,
- erőforrások kezelése,
- időnyilvántartás,
- számlák (számlakészítés),
- jegyzetek (kezelése),
- dokumentumok (kezelése, saját verziókövetővel),
- naptár,
- címek, (címjegyzék),
- hibakezelés (hibajegyek, bejelentések felvétele szinte bármire),
- csoportok,
- levelezés,
- könyvjelzőkezelés.

Beállításánál érdemes odafigyelni arra, hogy a felhasználótól mikor kérjen azonosítást (a bejelentkezéskor és bizonyos inaktívan töltött idő után), mert képes minden kattintás után is megtenni, ami roppant zavaró. A honlapján kipróbálható demó kiváló, teljes körű hozzáférés nyújt a teszteléshez (és nagyjából tud magyarul is). További teszteléshez kiváló lehetőséget nyújt, hogy képes az adatbankjait (adatbázisait) szinkronizálni, így könnyen nyerhetünk tesztadatokat. A fejlesztők a számlakészítő rész jelentős fejlesztését tervezik.

5. phpGroupWare

A szabad PMS-ek egyik úttörője, funkciókínálata széleskörű.

5.1. Előfeltételek

- web kiszolgáló: Apache,
- adatbázis kiszolgáló,
  - PostgreSQL 7.2.x,
  - MySQL 3.23.x,
  - Későbbiekben Oracle is.
- PHP 4 (minimum php 4.1.0.),
  - a választott adatbázis támogatásával,
  - igény szerint LDAP kiterjesztéssel.

- LDAP (OpenLDAP és PHP-ben kell az LDAP kiterjesztés) a felhasználói adatok és a címjegyzék tárolásához,
- Levelező kiszolgáló (PHP-ben IMAP kiterjesztés).
  - Courier-IMAP,
  - Cyrus-IMAP,
  - UW-IMAP,
  - MS Exchange IMAP (5.5).
- Egyéb (Sablotron a HEAD-hez).

5.2. Funkciók

AddressBook: címtár,

- LDAP és SQL adatbázis támogatása,
- személyes és szervezeti adatok elkülönítése.

Calendar: naptár,

Mail: levelezés,

Infolog: az AddressBookkal integrált CRM alkalmazás, telefonnaplóval. Minden tevékenység naplózására alkalmas, képes CSV importra,

FileManager: a saját VFS-ében levő állományok tárolása fájl, sql-db vagy webdav alapon,

Bookmarks: könyvjelzők megosztásához,

siteMgr: több webkikötő kezelésére is alkalmas webhelykezelő,

property: tulajdon/létesítmény kezelés,

phpBrain: saját tudásbázis elkészítéséhez,

fax: HylaFax-hoz felület,

Netsaint: felület a netsaint hálózatfigyelő használatához,

Stock Quotes: részvény-árfolyam kezelő,

Backup: cron alapú parancsfájl, mentéshez.

5.3. További elérhető modulok

Anglemail: az alapértelmezett levelező fejlesztői ága,

axisgroupware: a phpGroupware-re épülő bővítőmodul gyűjtemény. Magas minőségre és a használhatóságra összpontosító testreszabott phpGroupWare terjesztés.Pl. music library, saleslog, Axis Links,

CK-Ledger: A phpGroupWare-re épülő pénzügyi program,

Double Choco Latte: alapvető projektkezelési funkciók, tevékenységek idő követése, telefonnapló, e-mail értesítő, statisztikai riportok, riport motor,

JiNN: adatbázis alapú webes tartalomkezelő a phpGroupWare-hez.

A phpGroupWare saját API-jának felhasználásával további kiegészítőket készíthetünk.

6. PHProjekt

Belépéskor azonnal megkapjuk az aktuális emlékeztetőket. Interneten keresztüli használata is gyors. Átlátható.

6.1. Előfeltételek

- web kiszolgáló: Apache,
- adatbázis kiszolgáló,
  - MySQL
  - PostgreSQL 7.2.x,
  - Oracle
- PHP 4

6.2. Funkciók

Összesítő: ezt kapjuk bejelentkezés után, Tartalma: mai események, új jegyzetek, futó projektek, aktuális teendők, legfrissebb fórum üzenetek, ügyfélszolgálati kérések (hibajegyek), szavazások, új fájlok,

Naptár: események nyilvántartása, ismétlődő események, kapcsolattartó megadása, privát, normál, nyilvános események. Kényelmes, gyors, megadott projekthez hozzárendelhető, erőforrásigény megjelölhető, írhatunk megjegyzést, csoport nézet,

Kapcsolatok: exportálható XML, XLS, DOC, CSV, HTML, RTF formátumba, hozzátársított ToDo-t kilistázza (ahol az adott személy a kapcsolattartó),

Csevegés: csevegés :),

Fórum: egy hozzászólás egy oldal, azonnali válaszadási lehetőséggel. A főoldalon csak a hozzászólások címe, a hozzászóló és a dátum található,

Fájlok: fájlokat tölthetünk fel, melyeket projekthez rendelhetünk, hozzáférés-vezérlés megoldott,

Projektek: részprojektek, prioritások, egy oldalon vannak a hozzákapcsolódó megjegyzések, fájlok, naptárbejegyzések,

Időkártya: időráfordítások,

Feljegyzések: jegyzetek,

Helpdesk: Tudásbázis kezelése, hibajegyek kezelése (korlátozottan),

Levél: jól használható, testre szabható, levéliráskor adatbázis mezők beszúrhatóak, címzéskor csoporttagok, külső kapcsolatok kijelölhetőek, csatolások kezelése,

Teendők: kereshető, név, felelős (saját, vagy más teendő listájába kerüljön?), projekthez társítás, kapcsolattartó, státusz, határidő, prioritás,

Egyebek: csoportos könyvjelzők, szavazó rendszer maximum 3 választható elemmel. Kulcsszavas keresés a teljes rendszerben, ill. az egyes modulokban.

Opciók: beállítható a nyelv, a bőr (skin), a képernyő felbontása!, időzóna. A naptárban a munkanap első és utolsó órája.

A rendszer integrálható a Wiki-vel, mely nagyban megkönnyíti információs honlapok létrehozását. A fejlesztők jövőbeni terveiről. ImportExport: csatlakoztatható adatforrások és célok alkalmazása pl. csv-fájl, sql-db. WorkFlow: munkafolyamat-követő (workflow) modul készítése a phpGroupWare-hez. A honlapon kipróbálható demó adatai sajnos túl rövid időközönként törlődnek ahhoz, hogy mélyebben megismerhessük a rendszert.

Ajánló

Nincsen egyértelmű nyertes, az alapfunkciókat mindegyik tudja, a szükséges „extrák” határozzák meg, hogy mikor melyiket érdemes választani. A tesztelésre, kipróbálásra érdemes időt fordítani. A munka során keletkező dokumentumok kezeléséhez ajánlom az OpenOffice.org programot, mellyel „Az OpenOffice.org múltja és jövője” c. előadás foglalkozik.

A jövő

Véleményem szerint egyre több helyen fognak PMS-t alkalmazni, melyek tudása egyre bővül. A cégek a szabad szoftverként elérhető alkalmazás mellé üzleti alapon támogatást nyújtanak. Hamarosan elérhetők lesznek fizetős szolgáltatásként a webes munkacsoport és projektszervező funkciók megfizethető díjért. Olyan kisebb csoportok számára ideális, melyek gyors internet kapcsolattal rendelkeznek. Ennek egy szép példája az Ace Project[12].

Hivatkozások

[1] A TUTOS honlapja: `http://www.tutos.org/`

[2] A phpGroupWare oldala: `http://www.phpgroupware.org/`

[3] TUTOS a SourceForge-on: `http://sourceforge.net/projects/tutos/`

[4] TUTOS Debian csomagok: `http://tapoueh.org/debian/` vagy a `http://packages.debian.org/unstable/web/tutos.html`

[5] a CK-Ledger honlapja: `http://ck-ledger.sourceforge.net/`

[6] Az Axis Groupware honlapja: `http://axisgroupware.org/`

[7] A PHProjekt honlapja kipróbálható demóval: `http://www.phprojekt.com/`

[8] A Lotus honlapja: `http://www.lotus.com/`

[9] Egy online PMS: `http://www.freetaskmanager.com/`

[10] Nagyon jól használható link gyűjtemény:
`http://www.comp.glam.ac.uk/pages/staff/dwfarthi/projman.htm`

[11] Free Software Project Management HOWTO:
`http://yukidoke.org/~mako/projects/howto/`

[12] Ace Project: `http://www.freetaskmanager.com/`

A tűzfalon túl – A kéretlen levelek szűrésétől a teljes tartalomszűrésig

filter:max
http://www.filtermax.hu

2003.11.08.

Tartalomjegyzék

Az Internet mindennapos, rutinszerű használata szükségessé teszi a vállalati informatikai rendszer kérdéskörének komplex átgondolását. Sok vállalat számára ugyanis még ma is csupán a vírusellenes megoldások és a tűzfalak jelentik a hálózati biztonságot. A tűzfal valóban elengedhetetlenül szükséges, első számú védelmi vonal, hiszen segítségével kontrollálható a helyi hálózat kívülről történő elérése. A legnépszerűbb biztonsági elemet a vírusirtó programok jelentik. A vírusok, férgek és más ártó kódok rendkívüli mértékű terjedésének köszönhetően ma már szinte nincs olyan vállalat, ahol ne találkoznánk valamilyen vírusellenes megoldással. Azonban sem a tűzfal, sem a vírusirtó szoftverek nem csodafegyverek. Nem is kell annak lenniük. Azokat a feladatokat kell ellátniuk minél magasabb szinten, amelyekre valójában valók. Feladatkörük kibővítése – például tartalomszűrési lehetőségek kihasználása – félmegoldásokat szülhet, valamint csökkentheti az adott eszköz eredeti funkciójának hatékonyságát. Szükséges tehát a tűzfal és a vírusirtó program kettősének kiegészítése egy harmadik elemmel, a tartalomszűréssel, a hálózaton belüli fenyegetések, problémák megoldásának céljából. Az így trióvá kiegészült biztonsági rendszer immár megnyugtató megoldást nyújt minden hálózati biztonsági problémára, legyen az külső vagy belső.

A tartalomszűrés oly módon bővíti ki a vállalati informatikai rendszerek biztonságának kérdését, hogy a tárgyi és emberi erőforrások hatékony felhasználását is biztonsági kérdésnek tekinti. Létjogosultságát elsősorban a munkahelyi Internet-használat módjában, jellegzetességeiben kell keresni. Be kell látni, hogy a kívülről érkező támadások mellett jelentős kockázatot hordoznak a – közhely-szerűen a támadások 80-90%-áért felelősnek tartott – belső támadások illetve a helytelen Internet-használat.

A mára komoly iparággá fejlődött tartalomszűrés kezdeteit részint a bizalmas információk, részint a kéretlen levelek kiszűrésére irányuló technikák jelentették. Tekintsük át tehát a különböző levélszűrő eljárásokat.

1. A kéretlen levelek szűrése

A kéretlen levelek (UBE[1], UCE[2], röviden és pontatlanul spam [1]) vagyis a tömeges, általában kereskedelmi célú e-mail „informatikai időszámítás” szerint már-már történelmi jelenségnek számít. Az e-mail, mint kommunikációs eszköz tömeges elterjedése hívta életre. A kéretlen levelek számának rendkívüli növekedésével párhuzamosan, természetes folyamatként indult meg az ezeket kiszűrni szándékozó megoldások fejlesztése. Nyilvánvalóan ezek a szűrési technológiák is egyre kifinomultabbak lettek, amint a kéretlen levelek írói is igyekeznek egyre jobban álcázni küldeményeiket ezek elől.

Az elektronikus levelezést nap mint nap használó vállalatok, szervezetek számára egyre sürgetőbb feladat hatékony levélszűrés kialakítása. A legjobb levélszűrő megoldás kiválasztásához először azonosítani és értékelni kell a kéretlen levelek által okozott költségeket és kockázatokat. A kéretlen levelek több alapvető módon is érintik a szervezeteket:

- csökken a munkatársak hatékonysága,
- növekednek az informatikai és hálózati költségek,
- biztonsági és jogi kockázatok merülnek fel.

A Ferris Research 2003-as adatai szerint a kéretlen levelek miatti hatékonyságcsökkenés egyedül 874 dollárjába kerül a vállalatoknak munkatársanként, ehhez hozzáadva az

[1] Unsolicited Bulk Email, vagyis kéretlen tömeges e-mail
[2] Unsolicited Commercial Email, vagyis kéretlen üzleti célú e-mail

IT költségeket, a kéretlen levelek összesen több, mint 10 milliárd dollárjába kerülnek az amerikai vállalatoknak.

1.1. Hol történjen a kéretlen levelek szűrése?

A kéretlen levelek megállítására, kiszűrése tett erőfeszítések első lépcsőjeként meg kell határozni, hogy hova is szeretnénk felépíteni a védelmünket. A vírusokkal ellentétben – szerencsére – a kéretlen levelek csak egyetlen helyen, az Internet átjárón keresztül juthatnak be. Ezért több lehetőség is kínálkozik a levélszemét-áradat megállítására.

A védelmet tehát felállíthatjuk a munkaállomások szintjén, az e-mail szervernél, a hálózat szélén vagy éppen azon kívül.

1.2. Munkaállomás szintű szűrés

A kéretlen levelek szűrésére számos, egészen jól működő egyéni szűrő érhető el, amelyek elsősorban az otthoni felhasználók számára nyújthatnak megoldást. Sajnos számos otthoni Internet-használónak nehézséget okoz ezen levélszűrők által nyújtott lehetőségek maximális kihasználása. Vállalati környezetben (kivéve néhány fős kisvállalatokat) szinte elképzelhetetlen ezek hatékony használata, mivel az egyes munkaállomásokra történő telepítés, beállítás, karbantartás valamint a felhasználók képzése talán több időt és pénzt emészt fel, mint maga a kéretlen levelekkel kapcsolatos probléma. Ezen kívül az egyéni levélszűrők általában nem rendelkeznek olyan funkciókkal, mint a központosított menedzsment, kontroll és jelentéskészítés, amelyek nélkülözhetetlenek egy vállalat számára. Vagyis egy teljes vállalati kéretlenlevél-házirend aligha valósítható meg ezekkel. A legtöbb vállalat tehát központosítottabb megoldást keres.

1.3. Kéretlen levelek Csoportmunka-szerveres szűrése

A kéretlenlevél-szűrők következő csoportja a vállalati csoportmunka szervereken, vagy azokhoz kapcsolódva működik, általában Microsoft Exchange, Lotus Notes vagy Novell Groupwise kiegészítő moduljaként. Ezen szerverszintű szűrés esetén nyilvánvalóan a végfelhasználóknak már nem kell időt vesztegetniük a kéretlen levelek kezelésére, jelentősen növelve így munkájuk hatékonyságát. Kérdés ugyanakkor, hogy mennyiben lassítja a szoftvereknek helyet adó szerverek működését.

1.4. Kéretlen levelek Gateway szűrése

Több e-mail szerverrel rendelkező, vagy nagy terhelésnek kitett nagyvállalatok esetében a csoportmunka-szerveres e-mail szűrés nem megfelelő, számukra az átjáróban történő szűrés lehet a legjobb megoldás. A vállalati tűzfal és az e-mail szerver közé ékelődő kéretlenlevél-szűrő megoldásoknak három formája van:

Önálló Relay: Mind a szervezetből kifele, mind – igény szerint – befele haladó e-mail forgalmat kezelő (elemző és szűrő, fejlettebb esetben kategorizáló) szoftverek tartoznak ebbe a csoportba. Ezen szoftverek saját SMTP motort tartalmaznak, önálló MTA-funkcionalitásuk azonban csak a feltétlenül szükséges funkciókra terjed ki.

E-mail Relay Plug-in: Az SMTP relay-hez kapcsolódva szűri a beérkező üzeneteket, pl. a Sendmail „filter” funkcióján keresztül, így akár az SMTP válaszüzeneteket

is meghatározhatják (pl. SPAM-nek minősülő levél esetén „No such user” válasz adásával).

E-mail Firewall: Egy különálló eszköz, amely több, különböző e-mail kezelési funkciót (kéretlenlevél- és vírusszűrés, kimenő levelek szűrése, e-mail relay szolgáltatások) is ellát.

Jelen sorok szerzőjének véleménye szerint az e-mail „tűzfalak” az átlagos felhasználó számára gyenge átlagos teljesítményt és minőséget nyújthatnak, a szakmája iránt igényes rendszergazda mindenképp az első két megoldásból választ magának.

1.5. Kéretlen levelek külső szűrése (outsource)

A kívülálló által szolgáltatásként végzett kéretlenlevél-szűrés leveszi annak terhét a vállalatról. Ezek a megoldások úgy működnek, hogy a vállalat teljes beérkező e-mail forgalma egy külső szolgáltatón keresztül érkezik be a rendszerbe. A jobb SPAM-szűrő szolgáltatók vírus-, sőt, akár pornókép- vagy alapvető tartalomszűrési szolgáltatást is nyújtanak, így a káros e-mailek a vállalati hálózatnak még a határához sem érkeznek meg.

Megtérülés tekintetében egyértelműen ez lehet a legjobb megoldás. Ezt a lehetőséget választva ugyanis a vállalatnak nem kell többé belső erőforrásokat fordítania a kéretlen levelek áradatának kezelésére. Természetesen mérlegelni kell ennek a megoldásnak a kockázatait is egy saját belső rendszer működtetésével szemben.

A kéretlen levelek szűrésének két alapvető eleme: a kéretlen levél azonosítása és az azonosított üzenetek kezelése. A kéretlen levél azonosítására újabb és újabb technológiák születnek, próbálva lépést tartani a kéretlen levelek küldőinek (spammer) azon törekvésével, hogy megtévesszék, kicselezzék a szűrőket. Eleinte, az egyszerűbb technikákkal csak bizonyos szavakra vagy a kéretlen levelekben előszeretettel használt karakterekre (pl.:!!!, nagybetűs FREE, stb.) kerestek. Manapság ennél sokkal furfangosabb trükköket is ki kell tudni szűrni. Például rejtett HTML tartalmakat, vagy szándékosan hibásan írt szavakat (F_R_E_E, V1AGRA stb.)

Egy szűrési technológia kéretlenlevél-azonosítási hatékonyságát két érték írja le: a kiszűrt kéretlen levelek aránya az összes beérkező kéretlen levélhez képest és a téves szűrések aránya. A kéretlen levelek 100%-os azonosítása és 0%-os téves szűrés természetesen nem létezik, azonban 80-90%-os „elfogás” és 0,1%-os téves szűrés reális cél lehet.

Egy hatékony kéretlenlevél-szűrő megoldás lényege a többszintű szűrés, ebbe tartozhat: rendszeresen frissülő kéretlenlevél-adatbázis, szószűrés (pontozórendszerrel, pl. SpamAssassin vagy LexiMatch), figyelembe véve a szószűrés kicselezésére szolgáló trükköket (pl.: F_R_E_E, V1AGRA stb.), HTML levélszemét felismerése, képfelismerés (a kéretlen levelekben gyakran előforduló pornóképek miatt), HTML tartalom eltávolítás, valós idejű fekete lista (bár ezek alkalmazása gyakran többet árt, mint használ), tartománynév feloldás.

2. Tartalomszűrés

A kéretlen levél azonban csak a „jéghegy csúcsa” abban a tekintetben, hogy milyen kockázatokat vet fel az Internet munkahelyi használata. Nem megfelelő használat esetén a vállalati *e-mail* rendszereken keresztül vírusok, férgek támadhatják meg a hálózatot, titkos, bizalmas dokumentumok kerülhetnek illetéktelen kézbe, nem munkahelyre

való tartalmak küldhetők, fogadhatók, tárolhatók. A *web böngészése* hasonló veszélyeket jelent a vállalatok számára: illegális, ártó, nem kívánatos tartalmak érhetők el, jogvédett tartalmak tölthetők le. A letöltött tartalmak pedig *fájlcserélőkön* keresztül oszthatók meg, ráadásul ezen eszközök használata során már a letöltés megkezdésének a pillanatában is terjesztővé válhat a felhasználó, ami egyszerű jogvédett tartalmak (pl. zenék) esetében is veszélyes, pedofil anyagok esetében pedig már vállalhatatlan kockázatot jelent – miközben a letöltés elején a tényleges tartalom még nem, legfeljebb egy fájlnév ismert.

Az informatikai hálózat ugyanolyan *erőforrása* egy vállalatnak, mint bármely más anyagi vagy szellemi erőforrása. Ennek fényében teljesen jogos az az igény és törekvés, hogy ezzel az erőforrással is hatékonyan gazdálkodjunk. Tartalombiztonsági megoldások bevezetésének mozgatórugója tehát az, hogy *a dolgozók ne pazarolják a szervezet erőforrásait (így saját munkaidejüket se)* magáncélú e-mailezéssel és böngészéssel. Természetes persze az az igény, hogy az emberek -bizonyos mértékben- használhassák munkahelyükön az Internetet személyes célból is. A vállalati tartalombiztonsági megoldások pontosan ezt hivatottak szabályozni, annak érdekében, hogy mindkét fél számára elfogadható megoldás szülessék. Ezen erőforrás felhasználási kérdések mellett különböző *jogi problémák* elkerülése céljából is indokolt tartalombiztonsági rendszer felállítása. Magyarországon és Európában szerencsére nem elsősorban azoktól a szexuális zaklatási perektől kell egyik vagy másik félnek tartani, amelyek az Egyesült Államokban mindennaposak. Komoly jogi kockázatokat rejtenek azonban például az Interneten megtalálható jogvédett vagy illegális tartalmak letöltésének lehetősége. Hazánkban, ahol az egyik legszigorúbb az adatvédelmi törvény, elsősorban az *adatvédelmi és a munkajogi viták* lehetnek jogi szempontból ezen megoldások bevezetésének mozgatórugói.

A tartalomszűrés tehát olyan biztonsági megoldás, amely az Internet használatával kapcsolatos *vállalaton belüli kockázatok*ra kíván válaszolni. Eszközeivel előzetesen kialakított elvek mentén *kontrollálható vagy monitorozható* az Internet használata, annak érdekében, hogy egy szervezet informatikai erőforrásainak felhasználása biztonsági és munkahatékonysági szempontok szerint szabályozható legyen. Teljes körű tartalomszűréssel többek között kiszűrhető a nem kívánatos tartalom, a kéretlen levél, megakadályozható bizalmas *információk véletlen vagy szándékos kiszivárogtatása*, szabályozható a fogadható és küldhető e-mailek száma, mérete, típusa, küldésének ill. fogadásának időpontja és biztosítható a tényleges üzleti Internetes forgalom elsőbbsége.

Egy hatékony *e-mail* szűrési megoldásnak biztosítania kell, hogy akár egyénenként is szabályozható legyen, mit küldhet el és kinek. Valós idejű monitorozás, a gyanús levelek elkülönítése, távoli menedzsment, automatikus értesítések szabályszegéskísérletek esetén szintén szükségesek. A bizalmas információk nem kívánatos kiküldésének megelőzésére a teljes tartalom szerinti szűrés is rendelkezésre áll, mely a csatolt állományokra is kiterjed. A szövegkörnyezetet figyelembe vevő szűrés természetesen sokkal hatékonyabb, mint az egyszerű szószűrés, mely számos téves szűrést eredményezhet. Az e-mail szűrés legfontosabb technikái:

- Tartalomszótárak: több nyelvű, kategorizált szótárak (pl.: Felnőtt, Gyűlölet, Kéretlen levél, Szórakozás stb.).
- Több rétegű kéretlenlevél-szűrés: kéretlenlevél-adatbázis, szószűrés, HTML eltávolítás, intelligens tanuló rendszer, képfelismerés, tartománynév feloldás, blokkolás tartomány, e-mail cím és IP-cím alapján, valós idős feketelista.
- Csatolmány ellenőrzés, tömörítés, mentesítés, titkosítás felismerésére .

A *webszűrés* esetében pedig annak a szabályozása szükséges, hogy ki, mikor, mely oldalakat látogathat, onnan tölthet-e le adatot, ha igen mekkora időtartamban, mennyiségben , illetve a nap mely szakában. Az eredményes tartalomszűrés (legyen az e-mail vagy webszűrés) kulcsa a felállítható szabályrendszerek rugalmasságában rejlik, és lényeges eleme, hogy nem a konkrét weboldalak listája alapján, hanem azok tartalma szerint osztályoz. Természetesen korrekt webszűrő megoldásnál különböző szabályok határozhatóak meg a vállalat, adott csoportok vagy egyének szintjén, akár a vállalati címtárhoz kapcsolódva.

Nem megoldás az, hogy egyáltalán nem tesszük lehetővé az Internet használatát. Olyan szabályzatot és azt kikényszerítő eszközrendszert kell kialakítani, amelynek a segítségével ésszerűen szabályozható az Internet használata: mindenki számára elérhetővé kell tenni azokat az oldalakat, amelyekre munkájához valóban szüksége lehet, de korlátok közé kell szorítani a nem ilyen jellegű oldalak elérhetőségét. Bizonyos oldalak (pornó, rasszista, gyűlöletkeltő stb.) teljes blokkolása is indokolt lehet. Egyéb, kifejezetten személyes céllal felkeresett oldalak (sport, szórakozás, utazástervezés, pénzügyi szolgáltatások stb.) esetében pedig részleges szűrésre van szükség.
A webes forgalom szűrésének leggyakoribb technikái:

- URL „fekete/fehér” lista és/vagy adatbázis,
- intelligens kategorizálás,
- szövegrészletek szűrése,
- vírus szűrése,
- web-tartalmon belüli QoS – prioritások a tartalom szerint,
- idő szerinti korlátozás, sávszélesség beosztás, priorizálás,
- letölthető állományok típusa és mennyisége szerinti korlátozás,
- felugró ablakok, reklámcsíkok szűrése,
- valós idejű monitorozás.

A webes és e-mailes tartalmak szűrése mellett egyre fontosabb az *azonnali üzenetküldő és fájlcserélő rendszerek* használatának kontrollálása is.

Hatékony szűrési eredmény elérése érdekében a kiválasztott szűrési technológiának a lehető legrugalmasabbnak kell lennie abban a tekintetben, hogy lehetőség legyen szabályokat felállítani felhasználókként vagy tetszőleges csoportokként is. Nem létezik ugyanis „egyenméterű” megoldás. Minden vállalatnak saját, egyedi tartalomszűrési igényei vannak, sőt vállalaton belül is eltérő igények léphetnek fel, egyes osztályok, részlegek között is.

Összefoglalva azt mondhatjuk, hogy míg a vírusellenes programok és a tűzfalak a hálózaton kívülről befelé irányuló fenyegetések ellen nyújtanak védelmet, addig a *tartalomszűrés* a *belülről kifelé tartó forgalmat kontrollálja*, elemzi és az előzetesen kialakított biztonságpolitikai elvek szerint kezeli.

Hivatkozások

[1] Eric Raymond: The Jargon dictionary.
`http://catb.org/~esr/jargon/html/S/spam.html`

Az SQL-Ledger integrált ügyviteli rendszer

Kabai József

2003.11.08.

Kivonat

Az SQL-Ledger [1] egy olyan integrált ügyviteli rendszer, amellyel cégek könnyen és rugalmasan végezhetik el a megrendelések, számlák feldolgozását illetve azok egyidejű kettős könyvelését, valamint figyelemmel kisérhetik készletmozgásaikat. A program felhasználása rendkívül sokrétű, működik Windows, Linux, Mac operációs rendszerek alatt, valamint a legfejlettebb adatbázismotorokat (Postgres, Oracle) használja. Használható önálló gépeken, intraneten, esetleg igény szerint interneten is.

Tartalomjegyzék

1. Az általános jelenség

1.1. Elszigetelt alkalmazások

A vállalkozások működésük során folyamatosan szembesülnek azzal a problémával, hogyan egységesítsék a különböző alkalmazásokon futó adataikat, amelyek szigetként különülnek el a többi adatbázistól. A szállító/vevőszámla-nyilvántartás a pénzügyi osztályon szerepel valamelyik népszerű táblázatkezelőben. Ezeket az adatokat a könyvelés duplikálja a számlák feldolgozásakor, és a raktár is használ (évek óta) valami „ideiglenes" megoldást a raktárnyilvántartásra és a számlázásra. Ezekből az adatokból fáradtságos munkával lehet csak a vezetők számára értékelhető, és a valósághoz közeli jelentéseket létrehozni. Hogyan lehet a legfontosabb információkat, mint például az árréselemzést elvégezni, amikor a raktármozgások rögzítése elkülönül az árbevétel-nyilvántartástól?

1.2. Integrált rendszerek

Adódik a megoldás: a különböző osztályok ugyanabból az adatbázisból dolgozzanak, amelyet az integrált rendszerek biztosítanak. A legismertebb a nagyvállalatok részére készült SAP, amely a piac úttörője, és a legnagyobb piaci szereplője is egyben. Ezek a rendszerek már olyan fejlettségűek, hogy lehetőséget adnak nemcsak a múltbeli teljesítmény értékeléséhez, hanem a vállalati tervezésben is segítséget nyújtanak (ERP rendszerek: Enterprise Resource Planning). Tipikus példa egy termelő vállalat, amely számára létfontosságú a jelenlegi és a jövőbeni megrendelések alapján a termeléshez szükséges anyagbeszerzés ütemezését megtervezni.

1.3. Nincs rá keret

Az integrált rendszerek hasznosak, csak drágák, ráadásul a betanítást és a testreszabást is sok pénzért lehet csak megkapni. A rendszerek döntő többsége zárt forráskódú, ezért a felhasználó a program fejlesztőjéhez van láncolva: nincs alternatíva, nincs versenyhelyzet. Mit tehet az ember, ha nincs elég pénze és nem akar kiszolgáltatott helyzetbe kerülni? Körülnéz a *szabad világban*.

2. Reális alternatíva: szabad szofver

2.1. Szabadság, szeretem

A szabad szoftverek fejlesztése ebben a témában csak az utóbbi években gyorsult fel, és a SourceForge [2] oldalait böngészve szép számmal találhatóak szabad ügyviteli rendszerek. Jellemzően egyedi megbízásokból fejlődtek szabaddá ezek a megoldások. Mit jelent számomra tágabb értelemben a szabadság?

- használatához szükséges környezet is ingyenes (programnyelv, adatbázis stb.);
- szabadon letölthető: ingyenes;
- szabadon módosítható, nyílt forráskódú, amely könnyen átlátható, logikus;
- szabadon eldönthetem, hogy energiát és tanulást belefektetve magam módosítom az igényeim szerint, vagy mással végeztetem el;

- szabadon eldönthetem melyik operációs rendszeren használom.

Az SQL-Ledger maradéktalanul megfelel a fenti elvárásoknak:

- Az adatok a legjobb GPL licenszes adatbázisban a PostgreSQL-ben tárolódnak (a fizetős Oracle is választható);
- A kliens bármelyik böngésző (Internet Explorer, Mozilla, Lynx, Links, W3M) lehet. Javascript kódot nem használ, kompatibilitási problémák nem jelentkeznek;
- az alkalmazás Perl nyelven íródott;
- Apache webszerver kommunikál az adatbázis és a böngésző között;
- Linux, Windows, Macintosh rendszereken is fut.

3. Az SQL-Ledger könnyen módosítható

3.1. Jó tervezés

Ha valaki úgy dönt, hogy szabad szoftvert ír, az óriási kitartás mellett arra is szükség van, hogy a program alapjaiban jól megtervezett, és a kódja jól olvasható legyen: számos példát láthatunk mennyi fejlesztés jutott zsákutcába a programkód állandó foltozása miatt. Az ügyviteli rendszerek tipikusan olyan szoftverek, amelyeket gyakran kell testre szabni a különböző vállalati igényekből fakadóan. Egy jól olvasható, és érthető kód jelentős mértékben megkönnyíti a fejlesztők munkáját.

3.2. Módosított állományok

Az SQL-Ledger készítői a fejlesztőket is megcélozva óriási hangsúlyt fektettek a módosíthatóságra. A módosított Perl állományt két szinten lehet érvényesíteni: globális szinten, valamint felhasználói szinten, amelyet a Perl állomány nevének változtatásával lehet szabályozni. Például ha egy új dátummezőt illesztünk be a vevőszámlák feldolgozása során, a változtatás globális szintű lesz, így az állomány nevét *custom_ar.pl* névre kell változtatni. Egy másik esetben például dönthet egy könyvelőiroda, hogy a pénzügyi jelentések megtekintésekor az ügyfél még véletlenül se módosíthasson, törölhessen adatot (hiszen az a könyvelők feladata). Tehát a módosítást tiltó állomány neve (az ügyfél bejelentkező neve: client) a *client_ar.pl* lesz, amely tehát csupán a client bejelentkezés esetén fut le. A módosított nevű állományok így nem vesznek el új verziók telepítésekor sem, hiszen csak az eredeti *ar.pl* állomány íródik felül. Természetesen az új verzió állományát ismét módosítani kell saját kódrészeinkkel, ha élvezni akarjuk az előnyeit a módosításokkal együtt.

3.3. Új állományok létrehozása

A módosíthatóságon túl, lehetőség van új menüpontok létrehozására és ezek menürendszerhez történő könnyű csatolására. Például, ha egy határidőnaplót csatolunk a rendszerhez, az alapmodulokhoz hasonlóan létre kell hoznunk egy szimbolikus linket a gyökérkönyvtárban, a megjelenítésért felelős állományt a *bin/mozilla* – vagy lynx böngésző esetén *bin/lynx* – könyvtárba, és az adatbázis-műveletekért felelős Perl modult az SL könyvtárba.

A fentiekre konkrét példák találhatók a következő címen:

```
http://www.sql-ledger.org/cgi-bin/nav.pl?page=misc/api.html

                                                    &title=Customization
```

4. Választható felhasználói felületek

4.1. Kliensprogram: melyiket válasszam?

A tervezés során valószínűleg a legjobb választás a szabványos böngésző melletti döntés volt. Böngésző bármelyik gépen található; nem kell vesződni a kliensprogram telepítésével, illetve karbantartásával. Természetesen az alapverzióban nyomokban sem található a webprogramozók életét megkeserítő javascript kód, így minden böngészőben kiválóan működik.

A gyors adatbevitelt megnehezítő egérhasználatot grafikus böngészőkben is ki lehet váltani egyedi testreszabás során, a felhasználó kérésére. Erre az *accesskey* eszköz használható, amely lehetőséget ad egérműveletek helyettesítésére billentyűkombinációkkal, mint például a `<a href= ref.html ACCESSKEY="r">ref</a>` esetében, ahol az *ALT+r* lenyomásával aktívvá tehetjük a hivatkozást.

4.2. Más kliensprogramok választása

Ha valakinek nem tetszik a böngészős környezet, akkor a program rugalmasságának köszönhetően más felhasználói felület is húzható a program elé, mivel a folyamatok parancssoros üzemmódban is működnek a változók paraméterként való átadásával. A következő példa a szállítók listájának keresőfelületét jeleníti meg parancssorosan:

```
$ cd /usr/local/sql-ledger
$ ./ct.pl "login=name&path=bin/mozilla&password=&action=search&db=vendor"
```

A fenti példához hasonlóan, ismétlődő könyvelési tételek sokkal gyorsabb bevitelét is el lehet érni: táblázatkezelőben beírt tételeket (szöveges állományként) egy egyszerű szkripttel be lehet tölteni.

4.3. Együttműködés más alkalmazásokkal

A testreszabás nem feltétlenül történhet az alapprogram szabályai alapján. A *PostgreSQL* adatbázismotor közvetlenül is elérhető, amely lehetőséget ad nagy tömegű adat közvetlen bevitelére. Egy régi rendszer törzs- és záró adatait (cikktörzsállomány, nyitott vevőszámlák) nagyságrendekkel könnyebb így feltölteni, mint kézzel bepötyögni. *ODBC* adatkapcsolaton keresztül pedig saját egyéni lekérdezéseket írhatunk, amelyek eredményét tovább elemezhetjük kedvenc táblázatkezelőnkben.

Egy hazai nagykereskedelmi cégnél is hasonló megoldás született. A logisztikai osztály dolgozói nehezen váltak volna meg az évek óta használt MS Access felületüktől, így a rendelések feldolgozása, a szállítástervezés, illetve a szállítólevelek kinyomtatása Accessben történik. A megvalósult kiszállításokat az SQL-Ledger *ODBC* kapcsolaton keresztül automatikusan megkapja, és innentől kezdve a termék életútját már a pénzügyi osztály kíséri figyelemmel. A feladás a raktárkészletet módosítja, és a megvalósult vevőrendelésekről egy gombnyomásra számla készül.

5. A felhasználói kör

5.1. Klasszikus felhasználók

Az integrált ügyviteli rendszert gyakorlatilag bármilyen iparágban tevékenykedő cég használhatja, attól függően, mennyi energiát és fejlesztési munkát áldoz a testreszabásra. Mégis a legkevesebb beavatkozással a szolgáltató- és kereskedőcégek tudják az SQL-Ledger-t használni. A szoftver logikája a termék/szolgáltatás általános életútját követi:

Értékesítés esetén:

Ajánlat → Vevőrendelés → Kiszállítás → Számlázás → Pénzbefolyás

Beszerzés esetén:

Ajánlatkérés → Beszerzési rendelés → Beszállítás → Beszerzési számla → Pénzkifizetés

Az alapbeállítás során meg kell határozni a termék/szolgáltatás mozgásainak könyvelési szabályait, így a tranzakciók automatikusan könyvelésre kerülnek. A programot tehát olyan is tudja használni, aki nem ismeri a kettős könyvelés rejtelmeit.

5.2. Könyvelőcégek

A program ideális lehet könyvelőcégek számára is. Korlátlan ügyfél könyvelhető, minden ügyféladatbázishoz korlátlan felhasználó kapcsolható különböző jogosultságokkal. A tranzakció-kezelés megakadályozza az adatvesztést, és a nap bármelyik pillanatában biztonsági másolat készíthető a lekönyvelt tételekről. Megfelelő biztonsági beállítások mellett Interneten keresztül távmunkások alkalmazása is lehetséges, illetve az ügyfelek Interneten keresztül megtekinthetik cégük pénzügyi jelentéseit. Egyedi javascript alkalmazásokkal a könyvelőcég alkalmazottainak munkája automatikusan ellenőrizhető, illetve a hibás könyvelések száma jelentősen csökkenthető.

5.3. Rendelési rendszer

A `http://www.sql-ledger.com/cgi-bin/nav.pl?page=feature/nhc.html&title=Application%20Interface%20Example` címen a program olyan alkalmazása látható, amely során az Interneten keresztül a franchise partnerek továbbítják a rendeléseket a központi szerver felé, ahol a beküldött rendelések alapján a számlázás történik. Hasonló elven egy webáruház mögöttes adminisztrációjára is fel lehet készíteni a programot.

6. A magyar verzió

6.1. A program honosítása

Egy idegen nyelven fejlesztett szoftver honosításának első lépése a magyarra történő fordítás. A szoftver készítője könnyen használható eszközt kínál a fordításra, csupán

egy hash változóba kell beírni (a *locale/hu/all* állományban) a szó illetve kifejezéspárokat. A *locale/hu/locale.pl* elindításával a szövegpárok a megfelelő helyre kerülnek, és a szoftver közben folyamatosan be tudja olvasni a magyar fordításokat.

Egy könyvelőprogram használatánál a leggyakrabban felmerülő kérdés, hogy vajon megfelel-e a törvényi szabályozásnak. Az erre vonatkozó passzusok a számlázás miként jét szabályozzák, szó nincs arról, hogy az APEH-nál regisztráltatni kell a könyveléshez használt szoftvert (vagy hasonló tévhitek). A helyes számlakibocsátás követelményeinek megteremtéséhez – mint például kihagyás és ismétlés nélküli sorszámozás, számla kötelező adatai – módosítani kell a programot, amelyet külön csomagként lehet letölteni az éppen aktuális verzióhoz.

6.2. Ki vállal felelősséget?

Nehezebb probléma a törvény által megkövetelt felelősségvállalás kérdése, amelyben a program készítőjének vagy terjesztőjének nyilatkoznia kell a vonatkozó rendeletek betartásáról. Egy szabad forráskódú programnál ez első ránézésre megoldhatatlan feladat, hiszen felelősséget vállalni egy könnyen módosítható programért – enyhén szólva – felelőtlenség.

Mint minden ez sem lehetetlen: felelősséget lehet vállalni egy programért, amelyhez digitális ujjlenyomatot mellékelnek. Mindenféle módosítás a felelősségvállalás elvesztését vonja maga után, amelyet az ujjlenyomat igazol.

7. Mit hoz a jövő?

Idehaza még nincs köztudatban, mi is az a szabad szoftver. Néhányan már összefüggésbe hozzák a Linux-szal, de a szabad szofver mozgalom lényegét kevesen értik, illetve gyanakvással méregetik. A szoftver elterjedését jelentősen elősegítheti a szabad szoftver filozófiájának terjesztése, (amelynek érdekében jó néhány egyesület már évek óta kampányol pl. LME, fsf.hu), mégis a döntő lépés a megfelelő szakembergárda kialakulása lenne. Szükség van (döntően Linux) rendszergazdákra, programozókra, valamint alkalmazásban jártas tanácsadókra, akikre biztonsággal támaszkodhat a felhasználó. Ezeknek az embereknek a megtalálása lesz a közeljövő legfontosabb feladata.

Hivatkozások

[1] `http://www.sql-ledger.org/`: a szoftver oldala

[2] `http://www.sourceforge.net/`: szabad szoftverek gyűjteménye

[3] `http://www.investor.hu/`: a honosított verzió oldala

Tűzfal teljesítmény-tesztelés

Kadlecsik József

2003.11.08.

Kivonat

Ez a cikk a netfilter[1] csomagszűrő-rendszer kapcsolat-nyomkövető alrendszerének egy teljesítmény-tesztelését írja le. Megtárgyaljuk a tesztelés során felmerült nehézségeket, és bemutatjuk a mérések során kapott adatokat.

Tartalomjegyzék

1. Bevezetés

Egy tűzfal-rendszer a védett hálózat számára abszolút felső korlátot jelent az elérhető sávszélességben, a párhuzamosan igénybe vehető és kiszolgálható kapcsolatok számában. Ugyanakkor ha megpróbálunk utána járni annak, vajon adott szoftver-verzió mellett mekkora hardver képes megbízhatóan működni az általunk megszabott teljesítmény határértékig, lényegében légüres térbe kerülünk: publikált mérési adatok hiányában megérzésekre, *ökölszabályokra* vagyunk utalva.

A következő fejezetben pontosan lerögzítjük, mit is szeretnénk mérni. Ennek alapján megtárgyaljuk a teszteléshez kiválasztott szoftvereket, a mérés során felmerülhető nehézségeket és kiküszöbölési módjaikat. Leírjuk a tényleges tesztelési hardverkörnyezetet és topológiát. A harmadik fejezetben a kalibrálást tekintjük át, míg a negyedikben a tesztelési adatokat ismertetjük. Végül összefoglaljuk az eredményeket és a tanulságokat.

2. Mit vizsgáltunk, mivel és hogyan mértünk

2.1. Mit vizsgáltunk

Egy tűzfal esetén több teljesítmény-paraméter mérése mellett dönthetünk:

- csomag per másodperc (pps), különböző hosszúságú szabálylisták esetén

  Az ilyen tesztelés során azt vizsgáljuk, vajon mennyire lassítja le a csomagfeldolgozást az, ha a vizsgált tűzfal rendszernek egyre hosszabb és hosszabb nem egyező szabálylistán kell *átrágnia* magát ahhoz, hogy eljusson a csomaggal egyező szabályhoz. (Néhány összehasonlító mérés már történt netfilter/iptables és nf-hipac[2], valamint OpenBSD pf és Linux iptables[3] között.)

- kapcsolatkezdeményezés per másodperc (request/s), amikor a rendszer által lekezelt kapcsolatokat számláljuk egyre növekvő másodpercenkénti kapcsolatkezdeményezés mellett

  Ekkor nem a nyers csomag-átvitelt vizsgáljuk, hanem azt, hogy a kapcsolatnyomkövető rendszer milyen jó hatásfokkal működik.

A kapcsolatok számának mérését megnehezíti az, hogy nem elég csupán adott ütemben csomagokkal bombázni a rendszert – teljes kapcsolat-felépítést és bontást kell végrehajtanunk egy-egy kliens és szerver között. Vannak gyártók, amelyek közölnek kapcsolat-nyomkövetési adatokat, de UDP-vel mérve. Mivel az UDP **nem** kapcsolat-orientált protokoll, ezért meglehetősen kétséges, hogy értelmezhetők-e egyáltalán az ilyen adatok. Mindkét teszttípus esetén elvárható, hogy a teszteléseket különböző csomagméretek mellett is elvégezzék, nem csak az Ethernet-en elérhető maximális 1500 byte-os csomagméret mellett.

Méréseink során a netfilter kapcsolat-nyomkövető képességeinek vizsgálatát céloztuk meg TCP[4] protokoll és különböző csomagméretek mellett.

2.2. Mivel mértünk

Miután a célunk az, hogy kontrollált körülmények között TCP kapcsolatokat generáljunk, ezért egy arra alkalmas szoftvert kellett keresnünk. Végül a választásunk a

httperf[5] programra esett, amelyet HTTP szerverek teljesítmény-vizsgálatára fejlesztettek ki, de a mi céljainknak is megfelelt.

Hogy megértsük a *httperf* által nyújtott lehetőségeket és a működését, nézzük a tipikusan használt kapcsolóit:

```
httperf --server hostname \
        --port 80 --uri /test.html \
        --rate 150 \
        --num-conn 27000 \
        --num-call 1 \
        --timeout 5
```

Ezzel a paranccsal a *httperf* a *hostname* névvel megadott szerver 80-as portjához kapcsolódik, és onnan a `/test.html` oldalt tölti le, majd ezt ismétli újra és újra. Másodpercenként 150 TCP kapcsolatot kezdeményez, az összesen kezdeményezendő kapcsolatok száma 27000. Egy TCP kapcsolatban egy HTTP kérést hajt végre. Végül egy-egy kapcsolatkezdeményezésnél maximálisan 5 másodpercig vár, hogy megérkezzék a szerver válasza. Ha ez nem történik meg időben, akkor a *httperf* a kapcsolatfelépítést sikertelennek könyveli el. Mivel összesen 27000 kapcsolatot kezdeményezünk 150/s-os „sebességgel”, ezért a teszt teljes ideje megközelítőleg 180 másodperc lesz.

Azonban van néhány kliens-oldali korlát, amely limitálhatja a másodpercenként generálható kapcsolat-kérések számát, amelyekre ezért ügyelni kell[6]:

- a TCP portok száma 2^{16}, amelyből az első 1024 fenntartott

  A kliens számára lehetséges portok száma így 64512. Mivel egy adott TCP portot addig nem lehet újra használni, amíg az utolsó csomag utáni TCP_TIME_WAIT állapotból ki nem került, ez jelentősen korlátozza az egy kliens által (ugyanazon szerver ugyanazon portjára) másodpercenként kezdeményezhető kapcsolatok számát. Linux esetén a TCP_TIME_WAIT timeout értéke körülbelül egy perc, ami maximálisan 1075 kérés/s-t jelent.[1]

  Szerencsére a Linux kernelben ez a korlát megfelelő sysctl paraméterezés segítségével kiküszöbölhető:

  ```
  # A lokális port-tartomány fedje le a teljes
  # lehetségeset:
  echo "1024 65535" > \
  /proc/sys/net/ipv4/ip_local_port_range

  # TCP_TIME_WAIT állapotban levő portokat
  # használjuk fel újra:
  echo 1 > /proc/sys/net/ipv4/tcp_tw_recycle
  ```

- a processzenként egyszerre megnyitott fájlok (open file descriptors) száma korlátos, Linux esetén 1024

  Mivel egy felszabaduló file descriptor-t azonnal újra lehet használni, a kapcsolatkezdeményezési kérés timeout-ja érdekes ebben az esetben. Öt másodperces timeout értéket feltételezve ez maximálisan 204 kapcsolatkérés/s-ot jelentene, ha nem érkezik válasz a szervertől. Hogy ezt a korlátot kikerülhessük, a *httperf* forráskódját módosítva a limitet 65536-ra emeltük fel[7].

[1] Az RFC793 által ajánlott 4 perces timeout esetén ez 268 kérés/s-ra csökkenne le.

- TCP küldő és fogadó socket buffer memória

  A *httperf* esetében az alapértelmezett küldő és fogadó socket buffer memória 4KB és 16KB, így amikor a bufferek tele vannak, a lehetséges kapcsolatok számát a rendelkezésre álló memória is korlátozni tudja. Ezért megemeltük a rendszerbufferek méretét, hogy a kernel a legrosszabb esetben is legalább 20480 kapcsolatot legyen képes kezelni:

```
# Send buffer: 4K * 20480 = 83886080
echo 83886080  > /proc/sys/net/core/wmem_max
echo 83886080  > /proc/sys/net/core/wmem_max_default

# Receive buffer: 16K * 20480 = 335544320
echo 335544320 > /proc/sys/net/core/rmem_max
echo 335544320 > /proc/sys/net/core/rmem_max_default
```

A teljesítmény fokozása érdekében megnöveltük a SYN backlog és a TW bucket méreteket is:

```
# SYN backlog (szerveren): azon kapcsolatok maximális
# száma, amelyekre nem jött még válasz a kliensektől
echo 65536 > \
/proc/sys/net/ipv4/tcp_max_syn_backlog

# TW buckets: azon socketek maximális száma, amelyek
# a TCP_TIME_WAIT állapotban vannak
echo 500000 > \
/proc/sys/net/ipv4/tcp_max_tw_buckets
```

Hasonló okból megemeltük a tűzfalon az interfészek küldő queue-ját valamint a netdev backlog-ot:

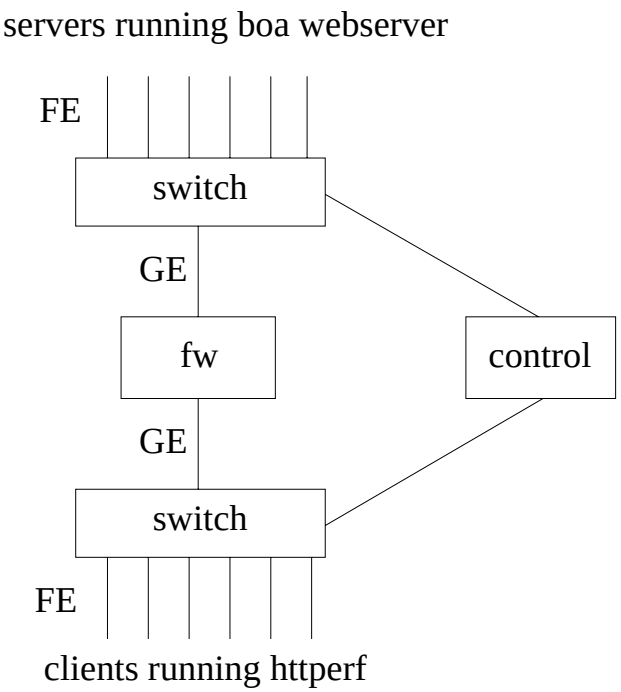
```
# Interface transmit queue hossza
ifconfig eth0 txqueuelen 1000
ifconfig eth1 txqueuelen 1000

# netdev backlog: az INPUT oldalon a queue-ben levő
# csomagok maximális száma, amikor az interfész
# gyorsabban kapja a csomagokat, mint ahogy a kernel
# fel tudná azokat dolgozni
echo 300000 > \
/proc/sys/net/core/netdev_max_backlog
```

A klienseken futó *httperf*-ek HTTP kéréseire a szervereken webszervernek kell válaszolni. Hogy a szerver program minél kevésbé befolyásolja az eredményeket, ezért az *Apache*-nál egyszerűbb és nagyobb teljesítményű *boa*[8] webszervert telepítettük, minden naplózás, CGI, MIME támogatás kikapcsolása mellett.

2.3. Hogyan mértünk

A teszteléshez egy mini-hálózatot építettünk fel (1. ábra), amely a következő elemekből állt:

- két 3Com 4228G switch
- tűzfal: Serverworks chipset, dual Xeon CPU, 2GB RAM, Intel e1000 Gb interfészek



1. ábra. A teszthálózat topológiája

- 5-5 kliens és szerver: dual AMD Athlon MP, 1GB RAM, 3c59x FE interfészek
- vezérlő-adatgyűjtő gép

Minden beállítást, paraméterezést, program-indítást a vezérlő gépről indítottunk, és ott gyűjtöttük össze az adatokat:

- a kliensekről:
  - *httperf* outputját
- tűzfalról:
  - `/proc/net/dev` teszt elején majd végén elmentett tartalmát az interfészek statisztikai adataival
  - *dmesg* outputot a kernel ring buffer tartalmával
  - *vmstat* outputját, amely a teszt alatt percenként írta ki a memória, CPU stb kihasználtsági adatokat
  - `/var/log/kern.log` tartalmát a kernel üzenetekkel

A teszteket a vezérlő gépről a következő shell-script töredék szerint indítottuk:

```
for each firewall-kernel-type
  reboot firewall with the given kernel
  for each filesize
    for each rate from 5000 to 50000, step by 5000
```

```
for each repeat in set (0 1 2)
  for each client
    start httperf &
  sleep 300
  collect logs from firewall
  for each client
    collent httperf logs
```

A letöltendő fájlok méretét úgy választottuk meg, hogy a fájlt átvivő Ethernet szegmens mérete a következő értékeket vegye föl[9]: 128, 256, 512, 1024, 1280, 1518, 64∗1518 byte (azaz utóbbinál 64 maximális Ethernet szegmens). Minden tesztet háromszor ismételtünk meg és az eredményeket a kiértékeléskor átlagoltuk. Két teszt között két percet vártunk a késleltetett csomagoknak a hálózatból való kiürülésére és a rendszerek alapállapotba való kerülésére.

Az adatokból a grafikonokat egy perl script készítette gnuplot segítségével.

3. Kalibrálás

A kalibrálás érdekében egy teszt-sorozatot indítottunk úgy, hogy nem volt betöltve a kapcsolat-nyomkövetés, vagyis a tűzfal közönséges router-ként funkcionált. A következő kernel verziókat vizsgáltuk így:

- 2.4.21
- 2.5.74
- 2.5.74 és az e1000 driverében bekapcsoltuk a NAPI támogatást
- 2.5.74 és az e1000 driverében bekapcsoltuk a NAPI támogatást valamint az interfészeket külön-külön hozzákötöttük egy-egy CPU-hoz (IRQ affinity)

A 2-5 ábrák jól mutatják, hogy a tesztelt szoftver/hardver konfigurációk gyakorlatilag érzéketlenek voltak a csomagméretekre: nagy méretű csomagokat épp olyan jól továbbítottak, mint kicsiket. A 2.5-ös kernel használata – különösen NAPI bekapcsolása mellett – teljesítmény-növekedést eredményezett a 2.4-es kernel sorozathoz képest. Az IRQ affinitás érdemben sem nem javított, sem nem rontott a teljesítményen.

4. Kapcsolat-nyomkövetés tesztelése

A kapcsolat-nyomkövetéshez alapkernelként a kalibráció során legjobb teljesítményt nyújtó 2.6-os kernelt vettük alapul, NAPI bekapcsolásával, IRQ affinity használata nélkül.[2] A következő kernel változatokat vizsgáltuk:

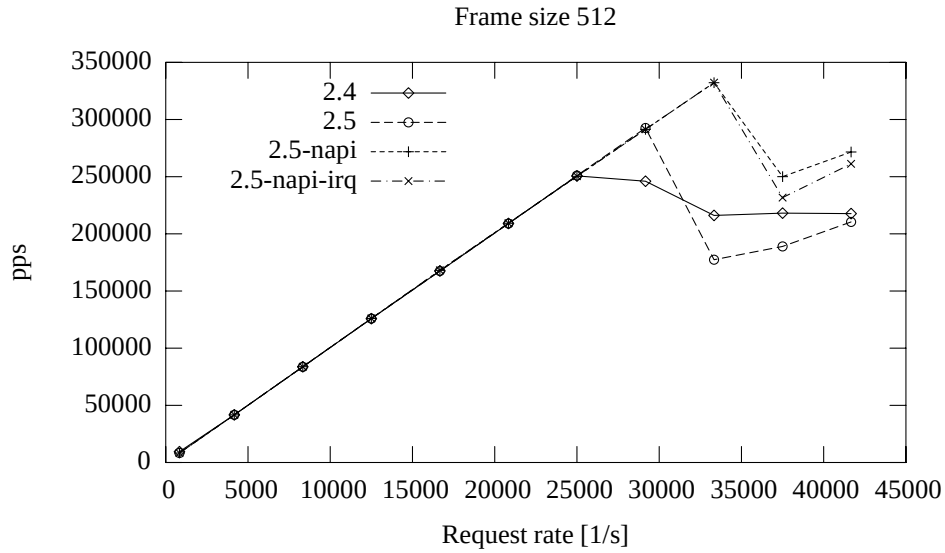
- 2.6.0, default netfilter conntrack

  2.6.0-ás kernel, az alapértelmezett kapcsolat-nyomkövető rendszerrel.

- 2.6.0 + locking patch

  Az előző kernel, de a globális ip_conntrack_lock helyett hash bucket szintű lockolást vezetünk be.

[2] A conntrack tesztelés során fölfedeztünk egy slab cache bug-ot, amelynek tisztázása a a tesztelésre fordítható időt lecsökkentette – a szerver/kliens gépeket a KFKI RMKI Grid klaszteréből meghatározott időre kaptuk kölcsön.



2. ábra. 512 Byte Ethernet frame, pps

- 2.6.0 + locking patch + TCP window tracking patch

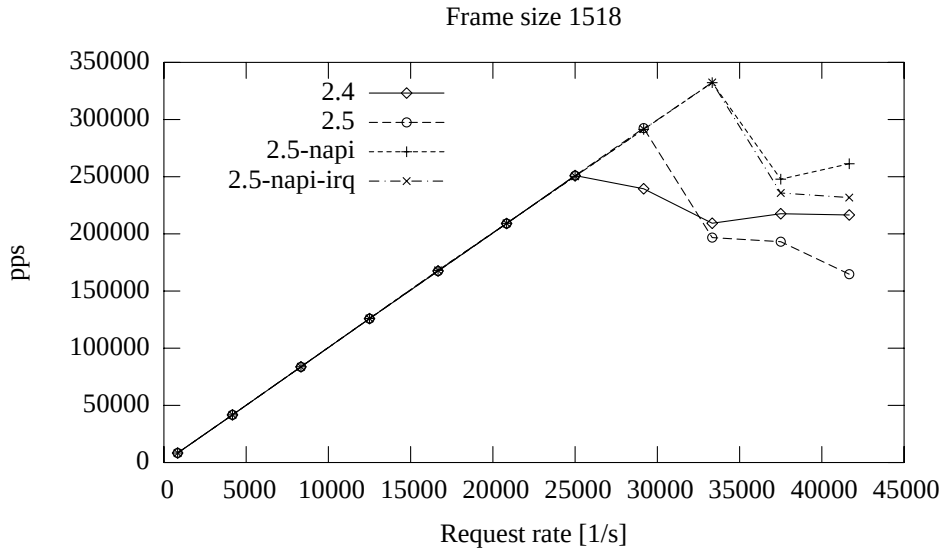
  Az eddigieken felül hozzáadjuk a TCP window tracking kódot, amely a kapcsolat-nyomkövetés során a TCP SEQ, ACK valamint window paramétereit is figyelembe veszi.

Amint látható, az alapkernelből néhány, a jövőben bevezetésre kerülő patch alkalmazásával kapott változatokat vetettük össze egymással.

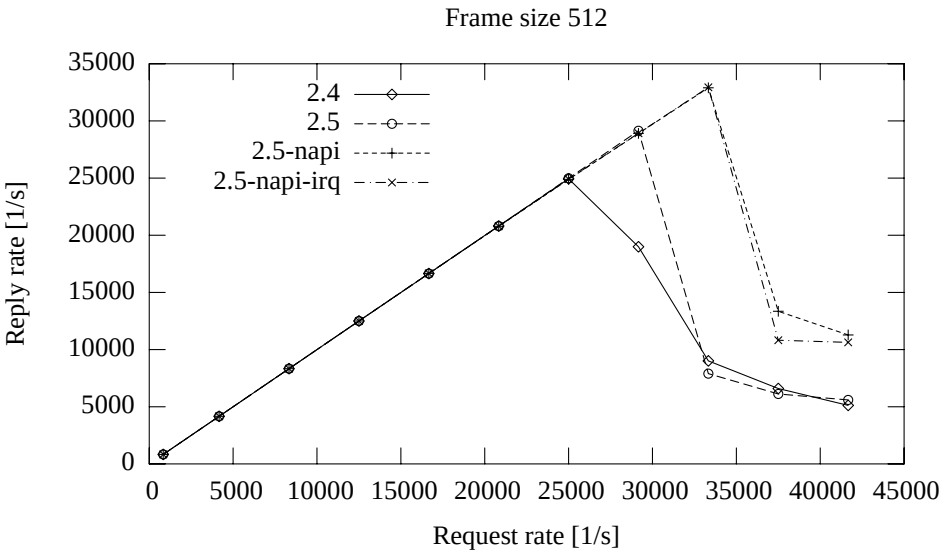
Az eddigi pps és request rate/s mellett vizsgáltuk az egy-egy tesztfuttatás alatt a rendszer által kezelt kapcsolatok számának időbeli változását is.

A mérési eredményeket a 6-10 ábrák foglalják össze. A pps adatokból kitűnik, hogy amint az várható, a kapcsolat-nyomkövetés valamennyit ront a csak route-olást végző rendszer teljesítményén, a locking patch pedig szignifikánsan javít a conntrack teljesítményén.

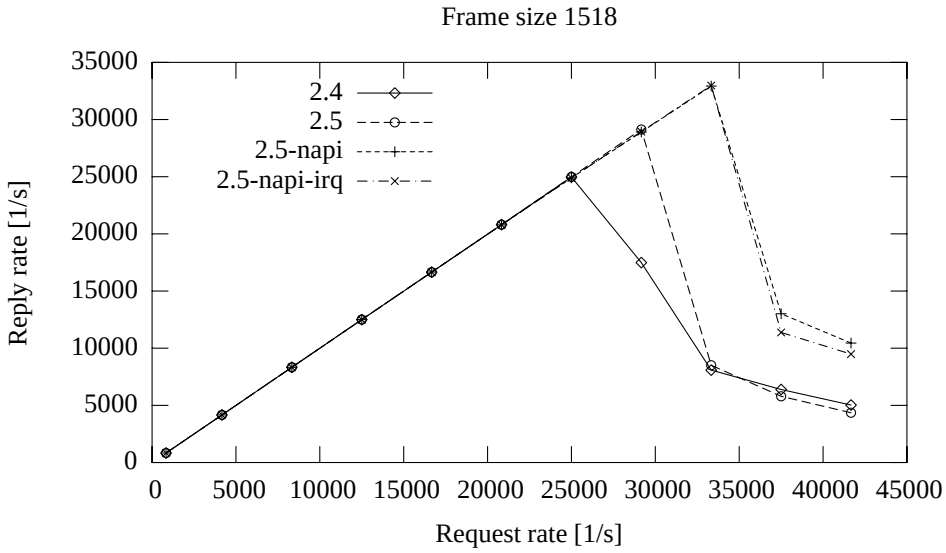
Meglepő módon a TCP window tracking kód esetén kicsi, de további **javulás** észlelhető. Ennek magyarázata összefügghet a kapcsolatok számának alakulását ábrázoló grafikonoknál a TCP window tracking görbe hullámszerű alakjával: a teszt mesterséges körülményei között a minél magasabb másodpercenkénti kapcsolat-kérés elérése érdekében **szándékosan** nem tartjuk be, hogy a kapcsolatok bontása után egy-egy port a TCP_TIME_WAIT ideig nem használható fel újra. A TCP window tracking kód a szabványos működést követeli meg, és az ugyanazon portra jövő új (szabálytalan) csomagokat nem engedi át mindaddig, amíg a szabvány szerinti timeout le nem telik. Ez egyrészt az általa nyilvántartott kapcsolatok számának csökkenésében (hullám-alak), másrészt a rendszer teljesítményének pici javulásában (csomagok hamarabb dobódnak el) nyilvánul meg.



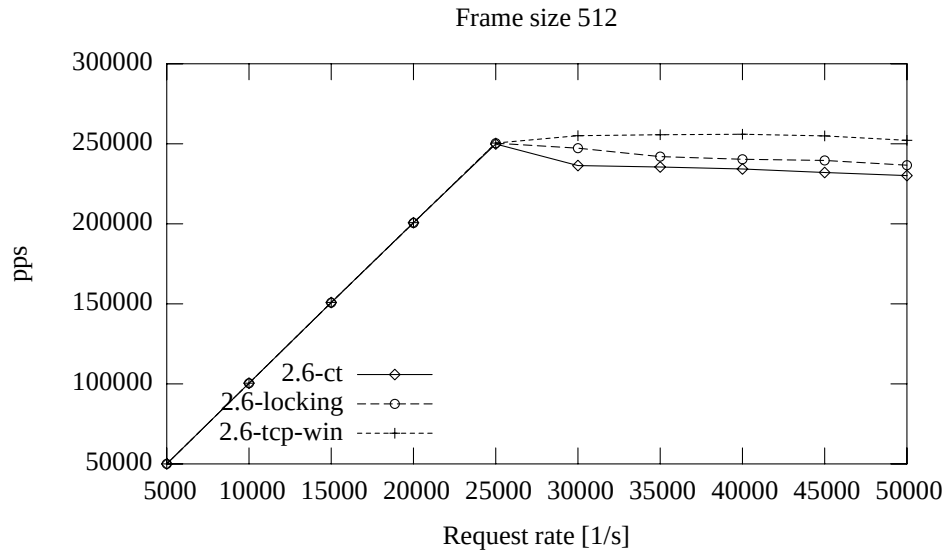
3. ábra. 6∗1518 Ethernet frame, pps



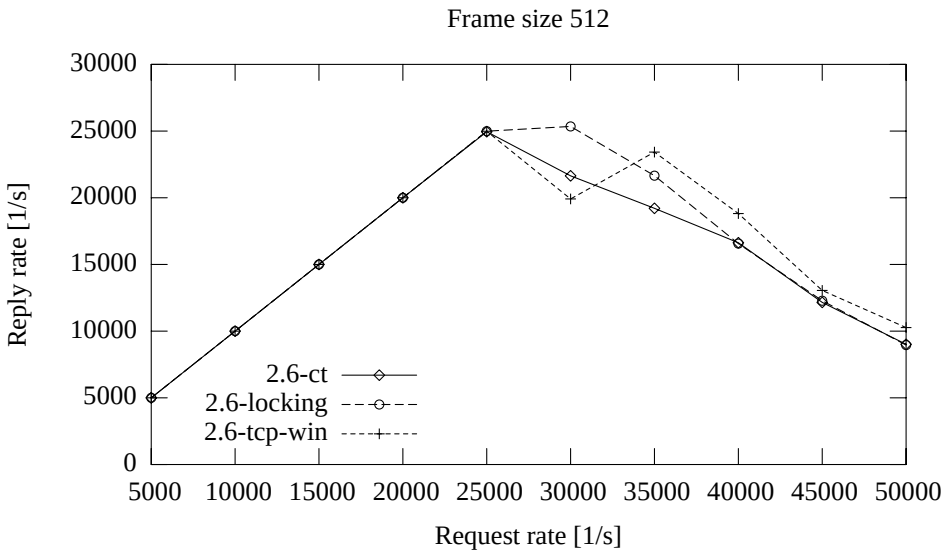
4. ábra. 512 Byte Ethernet frame, request rate/s



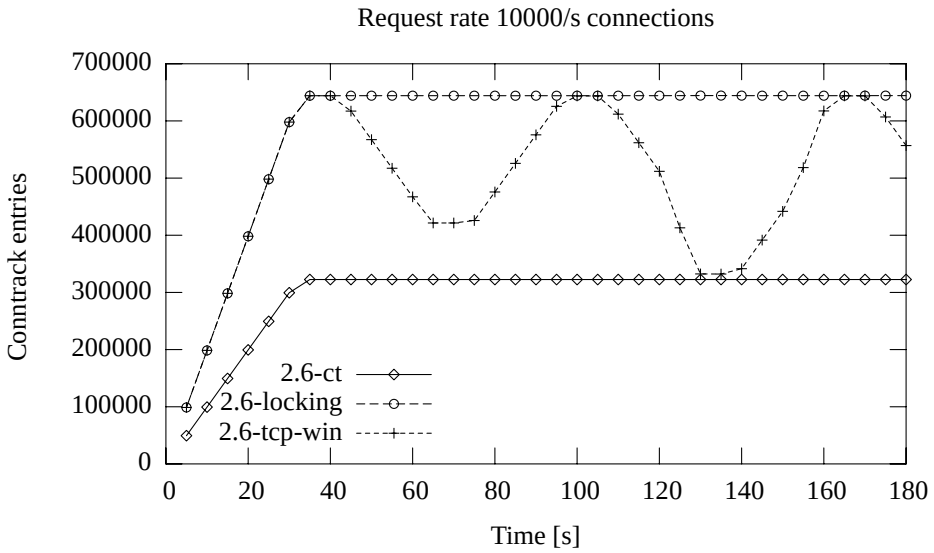
5. ábra. 6∗1518 Ethernet frame, request rate/s



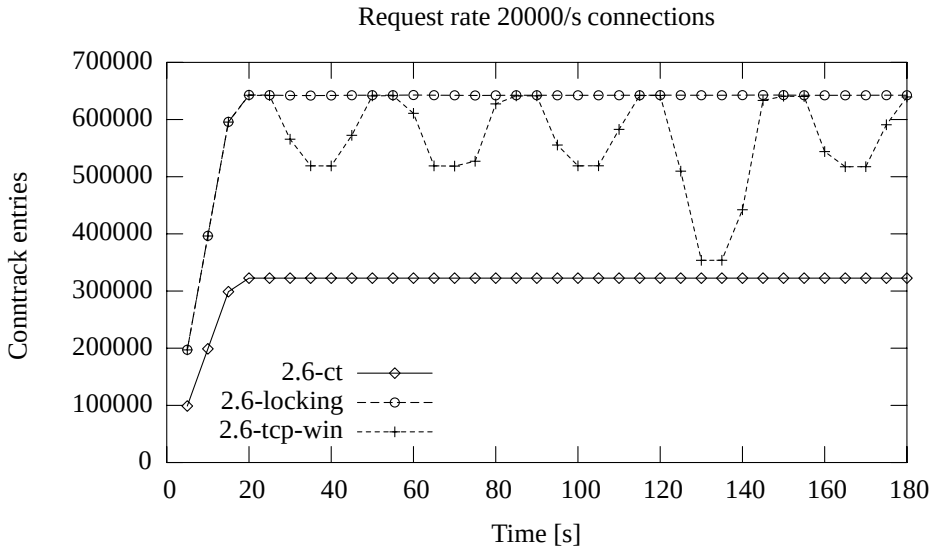
6. ábra. 512 Byte Ethernet frame, pps



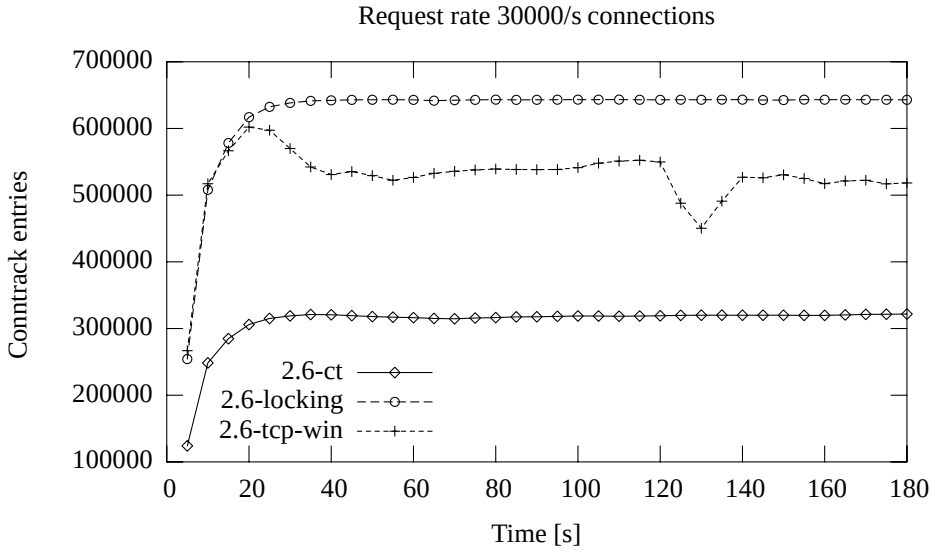
7. ábra. 512 Byte Ethernet frame, request rate/s



8. ábra. Kapcsolatok száma 10000 request/s mellett



9. ábra. Kapcsolatok száma 20000 request/s mellett



10. ábra. Kapcsolatok száma 30000 request/s mellett

5. Összefoglalás

Ebben a cikkben a kapcsolat-nyomkövetési alrendszer tesztkörülmények közti vizsgálatát mutattuk be. Az eredmények azt mutatják, hogy meglehetősen nagy teljesítményű Linux alapú állapottartó (stateful) csomagszűrő tűzfalak építhetők már a rendelkezésre álló kernel verziókból is, az új fejlesztések pedig a teljesítmény további javulásához vezetnek majd.

A teszt mesterséges körülményei között kapott eredmények mutatják, hogy további, életszerűbb (még több kliens/szever gépet használó) mérésekre van szükség ahhoz, hogy pontosabb adatokat kaphassunk a rendszerek viselkedéséről. A más rendszerekkel való összehasonlító mérések szintén még nyitott feladatot jelentenek.

Köszönetnyilvánítás

Köszönjük a KFKI RMKI Grid csoportjának, hogy a tesztelés megvalósíthatósága érdekében nélkülözhető számítógépeiket kölcsön adták.

Hivatkozások

[1] Netfilter
`http://www.netfilter.org/`

[2] Nf-hipac
`http://www.hipac.org/`

[3] Daniel Hartmeier, *Design and Performance of the OpenBSD Stateful Packet Filter (pf)*
`http://www.benzedrine.cx/pf-paper.pdf`

[4] RFC 793, Postel, J.,*Transmission Control Protocol*

[5] httperf
`ftp://ftp.hpl.hp.com/pub/httperf/`

[6] *httperf – A Tool for Measuring Web Server Performance*
`http://www.hpl.hp.com/personal/David_Mosberger/httperf.html`

[7] Patrick O'Rouke, Mike Keefe, *Performance Evaluation of Linux Virtual Server*
`http://www.missioncriticallinux.com/products/lvs.pdf`

[8] Boa
`http://www.boa.org/`

[9] RFC 1944, Bradner, S, McQuaid, J,*Benchmarking Metodology for Network Interconnect Devices*

Biztonságos hálózati elérés vezeték nélküli kapcsolat felett

Keszei Csaba

2003.11.08.

Kivonat

Napjainkban a vezeték nélküli hálózatok (WLAN, Wi-Fi, IEEE 802.11b[a,g,e]) praktikus és egyre olcsóbb megoldást nyújtanak hordozható eszközök csatlakoztatására a kiépített vezetékes hálózatokhoz, vállalati intranetekhez.

Olyan esetekben, ahol nem az elérendő sávszélesség a fő szempont, hanem a létesíthető végpontok száma fontos, talán a WLAN megoldások versenyképesebbnek is bizonyulhatnak akár fix végpontok estén is.

Biztonsági szempontból a WLAN kapcsolat vitathatatlanul sérülékenyebb a vezetékes alternatívánál, mivel lehallgatása gyakorlatilag észlelhetetlen. Erre kihívásra a készülékgyártók a rádiós csatorna titkosításával válaszoltak, amihez csak a WLAN-kártyákon található szűkös erőforrások használhatók. Ebből született meg a WEP, mely manapság bizonyítottan nem nyújt semmilyen védelmet mivel alacsony számítási kapacitást igényel ugyan, de kriptográfiai szempontból gyenge algoritmus.

Cikkemben áttekintem, hogyan lehet egy valóban biztonságos WLAN hozzáférési pontot kialakítani 1 WLAN kártya és egy közepes teljesítményű PC felhasználásával. Az adatkapcsolati réteg funkcióit a hostap csomag végzi, WEP-et eleve nem is használunk, a biztonságos kapcsolatért az IPSec protokoll felel.

Tartalomjegyzék

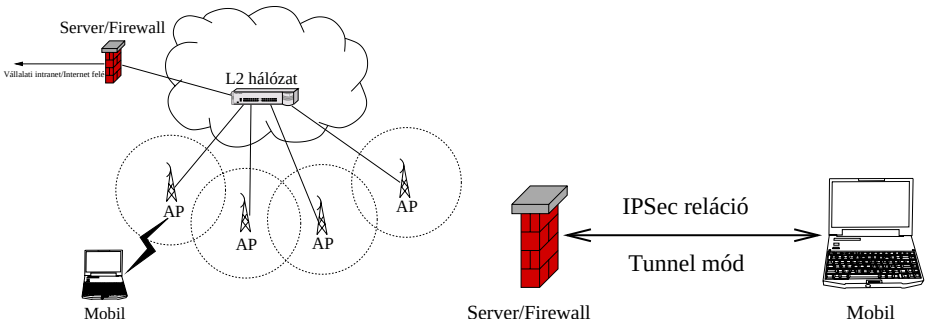
1. Bevezetés

A cikk témája a biztonságos hálózatelérés vezeték nélküli hálózati technika segítségével. Ezen dokumentum egy gyakorlatban is létező és élesben működő rendszer kialakítása során szerzett információkat és tapasztalatokat hívatott összegezni.

A felhasználók terminájai (továbbiakban *mobilok*) mozgó vagy hordozható eszközök (pl. laptop, PDA) melyek rendelkeznek IEEE 802.11b szabványt követő hálózati interfésszel (WLAN kártyával). Bár a mintarendszer a 802.11b rádiót használja, más típusú elérésekre is át lehet alakítani a biztonság szintjének érdemi változtatása nélkül.

Rendszerünk biztonságáról az *IPSec* protokoll gondoskodik amely független az alatta található link layer technológiától, így a 802.11b-től is. Az a tény, hogy az adatátvitel biztonságáról IP-szinten gondoskodunk megkönnyíti az új és fejlettebb rádiós technikák zökkenőmentes beépítését a rendszerbe. Feltételezzük, hogy az adatátvitel csak az IPv4-es protokollt használja. A mintarendszerben minden eszköz Linux alapú és telepítve van a *FreeSWAN* [1] csomag, ami a Linux rendszer egyik IPSec megoldása. Mivel mind a rádió, mind az IPSec szabványos protokollok, elképzelhető nem linuxos mobilok csatlakoztatása is a rendszerhez.

Feltételezzük továbbá, hogy egy felhasználóhoz egyértelműen hozzá lehet rendelni néhány fix adatot. Ezen adatok a WLAN-kártyájának ú.n. MAC címe (48 bites hardware vagy ETHERNET cím), használt IP címe, felhasználói neve és egy RSA aláírás (signature) ami egy publikus kulcs. Ezen adatokat a *felhasználói adatbázis*ban fogjuk tárolni.



(a) Fizikai topológia (b) Logikai kapcsolatok

1. ábra. Reference Architecture

Az 1. ábra a cikkben használt referencia architektúrát mutatja be. A Szerver/Firewallként jelölt eszköz a rendszer legfontosabb eleme (továbbiakban *tűzfal*). A mobilok ezzel az eszközzel lépnek u.n. IPSec relációba (1(b) ábra). További feladata a tűzfalnak mind az egész hálózat, mind pedig a kapcsolódó mobilok védelme az illetéktelen adatcsomagoktól.

A mintahálózat egyetlen rádiós cellából áll így az nem tartalmazza az ábrán jelölt L2 alhálózati „felhőt” és az ehhez kapcsolódó több hozzáférési pontot (AP-t).

2. A rádiós cella kialakítása

Egy 802.11b WLAN rendszer legalább három féle módon működhet (két ad-hoc mód és az infrastruktúra üzemmód). Ezen működési módok nem kompatibilisek, ráadásul az egyik még nem is szabványos. Legjobban az infrastruktúra üzemmódot támogatják a különféle gyártók eszközei; ez az az üzemmód, amit minden 802.11b eszköz ismer. Ebben az esetben beszélhetünk celláról és hozzáférési pontról (továbbiakban *AP*, azaz *Access Point*). Ebben a fejezetben egy ilyen AP kialakítását tekintjük át.

A mobilokra a használt WLAN kártya eszközmeghajtó modulját kell csupán telepíteni ezen a szinten. Ezzel a témakörrel a továbbiakban nem foglalkozom. A szükséges moduloknak széles választéka található meg a *pcmcia-cs* csomagban [3].

2.1. A hostAP csomag

802.11b AP-t kialakíthatunk egy – kliensekbe szánt – megfelelő WLAN-kártyából és egy PC-ből is. Mivel a rendszerünkhöz amúgy is szükséges egy Linuxos PC a tűzfal szerephez, nem szükséges beruházni egy drágább AP-be. Az AP funkcióit ellátja a *hostAP* [2] csomag, amely az Intersil által gyártott Prism chipsettel rendelkező WLAN kártyákat támogatja elsősorban. Ilyen, belülről roppant hasonló kártyákat forgalmaz számos „gyártó” (rengeteg Linksys, SMC és D-link kártya ilyen).

A hostAP csomag tartalmaz egy eszközmeghajtó kernelmodult (`hostap_cs`) és egy WLAN kártya konfiguráló démont (`hostapd`). (Illetve sok minden mást is, amit nem használtunk fel a mintarendszer kialakításához.) A csomag fordításakor ügyeljünk arra, hogy a leírásokban megadott módon engedélyezzük a *hostapd* támogatást. Figyeljünk arra is, hogy a tűzfal gépbe dugott WLAN kártyát a `hostap_cs` kernelmodul kezelje.

2.2. Egyszerű hozzáférés-kontroll

A `hostapd` program segítségével alapvető beállításokat lehet elvégezni az AP-n. A programnak paraméterként egy konfigurációs fájlt adva be tudjuk állítani a WLAN interfészt (`interface=wlan0`), WLAN SSID-t (`ssid=akarmi`) és egy egyszerű hozzáférés kontrollt (`macaddr_acl=1`). Ekkor egy külön fájlban fel lehet sorolni az engedélyezett mobilok MAC címeit. Másokat ez után nem fog beengedni a cellába.

Fontos megjegyezni, hogy ez csak minimális védelmet nyújt. Bővebben az 5.2 fejezetben.

3. IPSec telepítés

A *FreeSWAN* [1] egy igen komplex rendszer. Két fő részből áll, egy kernelmodulból (`ipsec.o`) amit KLIPS-nek neveznek a dokumentációkban és egy `pluto` nevű démonból ami a kulcsmenedzsmentért felelős. Egy parancssori eszközzel (`ipsec`) lehet elvégezni a különféle műveleteket.

3.1. Kernel-fordítás

Amennyiben a használt Linux kernel nem tartalmazza a FreeSWAN csomagot (a standard („vanilla”) kernel nem tartalmazza) akkor azt bele kell fordítani. A csomag egy

automatikus patch-elő Makefile-al érkezik, amit elindítva a kernel forrásunk konfiguráló és fordító rendszerébe jutunk. Aki fordított már Linux kernelt, annak nem okoz ez meglepetést.

A kernel konfigurálóban a „Networking Options” menüben válasszuk ki modulként az IPSec támogatást. Ezután automatikusan újrafordítja a kernelmodulokat továbbá lefordítja és telepíti a FreeSWAN-hez járó programokat is.

A FreeSWAN csomag tartalmaz egy jó leírást a csomag telepítéséről arra az esetre, ha valaki elakadna benne. A fájl neve `INSTALL` és a FreeSWAN főkönyvtárában van.

3.2. FreeSWAN beállítások

A FreeSWAN csomag rengeteg konfigurációs példát tartalmaz különféle helyzetekre. Ezekből sok mindent meg lehet érteni, most csak 1 db IPSec kapcsolat felépítésére koncentrálunk.

Egy mobil és a tűzfal között kell felépíteni egy IPSec kapcsolatot. A mobil teljes kimenő és bejövő forgalma ezen az IPSec kapcsolaton keresztül fog bonyolódni. Ez azt jelenti, hogy olyan IP csomagok titkosításával is foglalkoznia kell a tűzfalnak aminek sem nem forrása, sem nem nyelője. Ilyen helyzetekre az IPSec u.n. *tunnel* (alagút) üzemmódja alkalmas. A mobil összes forgalma egy IPSec alagútban fog utazni a tűzfalig így védve marad a rádiós átvitel során is.

A FreeSWAN rendszer fő konfigurációs állománya a `/etc/ipsec.conf`. Érdemes egy példát átszerkeszteni. A fájl általános dolgokkal kezdődik. Meg kell adni, hogy melyik fizikai interfészhez kapcsolódjon a virtuális IPSec interfész:
(`iterfaces="ipsec0=wlan0"`)
erről bővebben a 4. fejezetben lesz szó. Fontos még a `conn %default` szekció, itt kell pl. beállítani a használt hitelesítési eljárást.
Esetünkben ez RSA alapú lesz: (`authby=rsasig`).

Egy IPSec kapcsolatnak két egyenrangú oldala van. Definíciójakor a kapcsolat neve mellett definiálni kell az egyes oldalak tulajdonságait is. A FreeSWAN a „left” és „right” kulcsszavakkal különbözteti meg az egyes oldalakat. Legegyszerűbb eljárás az, ha egy adott „oldal” mindig ugyan arra az eszközre vonatkozik. Esetünkben pl. a „right” kulcsszó jelenti a mobilt minden esetben. Ez által a tűzfalon és a mobilon található konfigurációs fájlok nagyon hasonlóak lesznek.

Egy kapcsolat definíciója (avagy egy mobil „előfizetése”) a következőképpen néz ki a mobil oldalán:

```
conn lap
 right=10.1.1.2  #mobil IP címe
 rightsubnet=10.1.1.2/32
 rightid=@local.info
 rightrsasigkey=0sAQPajMWa+wCxtKUt2eqOoXg+y0aKrKfBJ...
 left=10.1.1.1   #tűzfal IP címe
 leftsubnet=0.0.0.0/0
 leftid=@firewall.hu
 leftrsasigkey=0sAQN+yrpSr2WMRkaNhX24E+GMwWeuOCf3k...
 auto=start
```

Ez definiál egy „lap” nevű kapcsolatot (`conn lap`). A mobil fix IP címe 10.1.1.2 (`right=10.1.1.2`) subnetje saját maga (`rightsubnet=10.1.1.2/32`). A mobil publikus RSA kulcsát a mobilon kiadott `ipsec showhostkey --right` paranccsal lehet lekérdezni, és ezt kell a konfigurációs állományba bemásolni.

A „left” kezdetű sorok az előbbi adatokat adják meg a tűzfal oldalára. A teljes szekció a mobilban és a tűzfalon található konfigurációs állományokban csupán egyetlen sorban tér el. Az `auto=` sor azt állítja be, hogy mi történjen a teljes IPSec rendszer indulásakor.

Ez a mobil oldalon `auto=start` ami azt jelenti, hogy az IPSec rendszer indulásakor azonnal próbálkozzon is a kapcsolat felépítésével. A mobil úgy lett kialakítva a mintarendszer során, hogy az IPSec rendszer (kernelmodulok, stb. . .) csak akkor töltődnek be, ha a mobil ezen a biztonságos WLAN kapcsolaton keresztül akar kapcsolódni. Több bekonfigurált IPSec kapcsolat esetén ez a módszer nem megfelelő. Az egyes kapcsolatok között az `ipsec` parancs segítségével lehetne váltani.

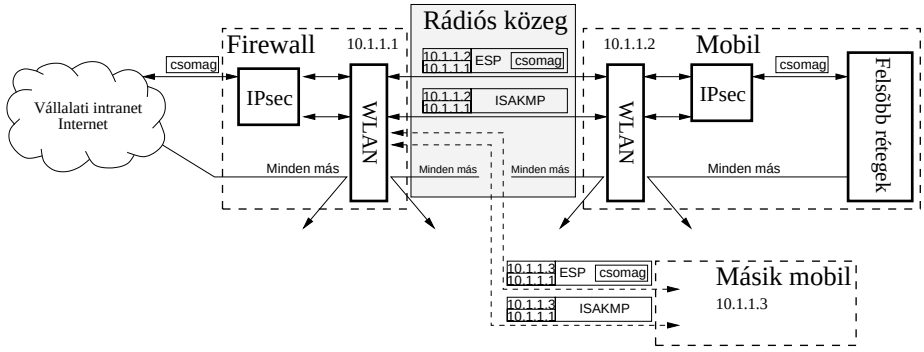
A tűzfal oldalon a megfelelő beállítás az `auto=add`, ez azt jelenti, hogy a tűzfal passzívan figyel és elfogadja a mobil kapcsolatfelépítési kérelmét. Ezért a mobil tetszőlegesen felépítheti és lebonthatja az IPSec kapcsolatot. Minden kapcsolatfelépítéskor lezajlik az IPSec protokollban meghatározott hitelesítés és átmeneti kulcsgenerálás.

4. Linux kernel tűzfal-szabályok

Ebben a fejezetben részletesebben áttekintjük, hogy mi is történik a csomagokkal egy IPSec rendszerben. Továbbá sorra vesszük, milyen triviális és kevéssé triviális beállításokat kell elvégezni a csomópontok tűzfal-rendszereiben.

4.1. Az IPSec működése

Az IPSec rendszer egy új, virtuális interfészként jelenik meg az operációs rendszer számára. Minden ezen interfészen keresztül route-olt csomag a neki megfelelő IPSec kapcsolaton keresztül fog továbbításra kerülni. Mintarendszerünk esetén a mobil IP feletti protokollrétegeiben keletkező összes csomag az IPSec interfészen keresztül továbbítóik, ami azt jelenti, hogy bekerül egy IPSec alagútba a rádiós közeg fölött. A 2. ábra szemlélteti a folyamatot.



2. ábra. Csomagutak és tűzfal-működés

Megfigyelve az egyes interfészeken a forgalmat (pl. a `tcpdump` programmal) a WLAN interfészeken – ha mindent megfelelően beállítottunk – olyan csomagok közlekednek, amiknek forrás és cél IP címei megegyeznek vagy a regisztrált mobilok vagy a tűzfal IP címeivel. A csomagok tartalma egy u.n. ESP-csomag ami egy erősen titkosított formában tartalmazza az eredeti csomagot. A titkosítatlan csomagokat az IPSec

interfészeken lehet megfigyelni, természetesen erre csak a mobilokon belül illetve a tűzfalon túli hálózatban van lehetőség ahogy ez a 2. ábrán is látható.

4.2. RP-filter beállítása

Létezik egy Internetes szabvány (RFC2267 [4]), ami megvéd számos olyan támadástól ami hamisított forrás IP címek felhasználásával működik.

Röviden ez az RFC azt mondja ki, hogy „ne vegyünk át olyan csomagot egy adott interfészen, aminek a forrás- és célcímét felcserélve, az így kapott csomagot egy másik interfészen keresztül továbbítanánk."

Sajnos az IPSec rendszerben ez üzemszerű, mivel minden csomagot az IPSec interfészen keresztül route-olunk, de a bejövő titkosított csomagok a WLAN interfészen érkeznek. Ezt a szabályt tehát ki kell kapcsolni:

```
echo 0 >/proc/sys/net/ipv4/conf/wlan0/rp_filter
```

4.3. Linux tűzfal beállítások

Ha a WLAN-interfészen nem ESP csomag érkezik, akkor az feldolgozásra kerül, megkerülve az IPSec rendszert. Ez teszi lehetővé, hogy olyan csomópontokkal is kommunikáljunk, akikkel nem vagyunk IPSec relációban. Ez a mi rendszerünk esetében nem megengedett, hiszen így bárki könnyedén kicselezheti a biztonságosnak hitt WLAN szolgáltatásunkat. Minden olyan csomagot szigorúan el kell tehát dobni, ami nem szükséges a rendszer működéséhez. Ezt meg kell tenni mindkét irányban szabályokat elhelyezve a PREROUTING és POSTROUTING láncokba a Linux kernel csomagszűrő rendszerében. Fontos ezt megtenni a mobilokon is. Az 5.4. fejezet foglalkozik egy lehetséges támadással arra az esetre, ha ezt elmulasztjuk.

Az ESP-csomagokon kívül át kell engednünk az IPSec protokoll jelzéscsomagjait is. Ezek az u.n. ISAKMP csomagok az 500-as UDP port-ot használják. Mindez összegezve látszik a 2. ábrán.

Egy lehetséges tűzfal oldali `iptables` script a következő:

```
IF=wlan0
SR=10.1.1.1
IPT="iptables -t mangle"
$IPT -N wlanin
$IPT -N wlanout
$IPT -F wlanin
$IPT -F wlanout
$IPT -A PREROUTING -i $IF -j wlanin
$IPT -A wlanin  -d ! $SR -j DROP
$IPT -A wlanin -p udp --destination-port 500 -j ACCEPT
$IPT -A wlanin -p esp -j ACCEPT
$IPT -A wlanin -j DROP
$IPT -A POSTROUTING -o $IF -j wlanout
$IPT -A wlanout  -s ! $SR -j DROP
$IPT -A wlanout -p udp --destination-port 500 -j ACCEPT
$IPT -A wlanout -p esp -j ACCEPT
$IPT -A wlanout -j DROP
```

4.4. IPv6 csomagszűrés beállításai

Az IPv6 protokollban minden aktív interfésznek van egy u.n. *link local* IPv6 címe. Amennyiben a futó Linux-kernel támogatja az IPv6-ot a WLAN interfészünknek lesz

link local címe. Az egyes alkalmazásoktól függ, támogatják-e az új IP protokollt, illetve fogadnak-e kapcsolatot link local címekről. Minden estere az `iptables` programmal beállított tűzfal-szabályok *csak* az IPv4 protokollt használó csomagokra érvényesek. Tehát IPv6-on a rendszer akár átjáróház is lehet.

Mivel a mintarendszerben nem akartuk használni az IPv6-ot, legjobb az összes IPv6-os csomagot eldobni. Ezt az `ip6tables` paranccsal kiadott szabályok segítségével tettük meg.

4.5. ARP beállítások

A fenti szabályokkal kiszűrtük az ARP (address resolution protocol) csomagokat is, amik nélkül egy pont-multipont IPv4 rendszer működésképtelen. A mi rendszerünk inkább pont-pont az IPSec reláció miatt, így az ARP-t nyugodtan letilthatjuk (`ifconfig wlan0 -arp`) és statikusan bejegyezhetjük a szükséges MAC-címeket (`arp -s -i wlan0 IP cím MACcím`).

5. Támadások

Ebben a fejezetben néhány támadási példát tekintünk át. A különböző példák másmás biztonsági szinttel ellátott WLAN-rendszereket feltételeznek. Az általuk jelentett kockázat is különböző.

5.1. Passzív lehallgatás

Egyes kártyák (pl. Prism chipset) speciális üzemmódja alkalmas erre, ezt a WLAN-terminológiában *monitor mód*-nak nevezik. Ilyen üzemmódban a kártya a beállított frekvencián vett *összes* csomagot továbbítja a felsőbb rétegek felé további feldolgozásra. A kártya felépítéséből adódóan ilyenkor nem képes adásra, így a lehallgatás teljesen észlelhetetlen.

Az üzemmód kiválasztása `hostap` driver esetén az `iwpriv wlan0 monitor 2` paranccsal lehetséges. Ezek után triviális, hogy szükséges valamilyen védelem a WLAN rendszerek esetén.

5.2. Tetszőleges MAC cím beállítása

Az `ifconfig` parancs leírását tanulmányozva kiderül, hogy az MAC vagy hardware cím is egy szabadon megválasztható azonosító, hasonlóan az IP címhez. Ezt az eljárást számos ETHERNET- és WLAN kártya meghajtóprogramja támogatja. Saját tapasztalatom szerint a Prism alapú valamit a Lucent/Avaya Orinoco Silver kártyákon a MAC-cím átmeneti megváltoztatásának semmi akadálya.

```
ifconfig wlan0 hw ether 00:0a:0b:0c:0d:0e
```

Ez már komolyabb probléma, mivel a legtöbb AP támogatja az MAC-címeken alapuló hozzáférés kontrollt. Ez azonban hamis biztonságérzetet ad az adott szolgáltatónak, mivel a technika kijátszható [1].

[1] Valóban kijátszható. Hallottam erről panaszkodni szolgáltatót, illetve volt szerencsém ingyen élvezni egy hotel – így már – ingyenes WLAN szolgáltatását.

5.3. WEP kulcsok feltörése

A WEP-ben használt titkosítási algoritmus nem nyújt a DES-ben (Data Encryption Standard – adattitkosítási szabvány) megszokott szintű adatvédelmet. Ráadásul a WEP-implementációjába számos „egyszerűsítés” is becsúszott, így az általa nyújtott biztonság kérdéses.

Gyakorlatilag a WLAN forgalom passzív lehallgatásával – megfelelő mennyiségű adat összegyűjtése után – megfejthető a rádiós cellában használt WEP-kulcs. Erre kész applikációk tölthetőek le az Internetről.

5.4. „Nyílt-rejtett szöveg” támadás

Klasszikus támadási módszer az, hogy egy titkosító gépet a támadó által ismert szöveg titkosítására kényszerítenek és így rendelkezésre áll a szöveg nyílt és rejtett (titkosított) verziója is. Ez az általunk használt 3DES eljárás eseteben nem jelent kritikus problémát. Viszont alkalmas a mobil terminál processzor-terhelésének növelésére.

Helytelen tűzfal-beállítások esetén a támadás nagyon egyszerű: a mobilba be kell juttatni olyan csomagokat, amire az válaszolni próbál. Ha engedélyezett tetszőleges csomag bejutása a mobilba akkor erre válaszként egy titkosított csomagot fog kisugározni. ICMP echo (ping) csomagok esetén elő is állt a klasszikus támadás.

Hivatkozások

[1] FreeSWAN an Open Source IPSec Implementation (v1.99): *http://liberty.freeswan.org/*

[2] HostAP Driver for Prism Chipsets (v0.0.1): *http://hostap.epitest.fi/*

[3] PCMCIA Card Services Package (v3.2.4): *http://pcmcia-cs.sourceforge.net/*

[4] Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing, rfc 2267: *http://www.ietf.org/rfc/rfc2267.txt*

Nyílt forráskódú szoftverek és a kormányzatok

László Gábor

2003.11.08.

Kivonat

Az előadás célja a trendek és azon okok bemutatása, amelyek közrejátszanak abban, hogy a gazdaságilag vezető nagyhatalmaktól kezdve, a fejlődő országokon keresztül a világ nemzeti kormányai miért fontolják meg, illetve támogatják – másmás megközelítésből ugyan – a nyílt forráskódú szoftverek használatát a kormányzati munkában.

Tartalomjegyzék

1. A közigazgatás speciális feladatai

A felgyorsult technológiai fejlődés, a hálózatosodás új kihívások elé állítják a kormányzatokat is, amelyek különböző mértékben, de törekednek a „szolgáltató állam” megvalósítására. Az üzleti szférához viszonyítva a közigazgatásnak speciális feladatok ellátására – az integritás, a biztonság és a közadatokhoz való hozzáférés biztosítása – kell alkalmasnak lennie, és ezen speciális feladatok ellátására nem mindig alkalmasak a több célra kifejlesztett kereskedelmi szoftverek. Ezért az adatok tárolása és visszanyerése egy titkos és szabadalommal védett zárt adatformátum esetén különösen problematikus. Továbbá az állampolgároknak joguk van az átláthatósághoz, amelyet akadályozhatnak a kereskedelmi szoftverek titkosságuk által. Jó példa erre az e-szavazás szoftver. Valószínűleg senki sem akarja komolyan megvédeni a szabadalmaztatott szoftvercégek jogait, amely jogok megakadályoznák azt, hogy a politikusok megvizsgálják a választások eredményeit értékelő szoftvert[1].

A végbemenő technikai változások által generált igények következtében, a kormányzatok munkájának megújulásaként megalkotásra kerülő e-kormányzati stratégiák esetében számos szempontrendszert kell figyelembe venni. Egyrészről a technikai fejlettség és az alkalmazható technológia, másrészről pedig a szolgáltatást igénybe vevő állampolgárok igényeire is tekintettel kell lenni. Michel Sapin francia közigazgatási miniszter szerint: „Az e-kormányzat következő generációjának két követelménye van: az interoperabilitás és az átláthatóság. Ez a kettő tulajdonság az erőssége a nyílt forráskódú szoftvereknek.”

2. Miért fontolandó meg az OSS használata a kormányzati munkában?

Állandó kérdés, hogyan lehetnek a nyílt forráskódú szoftverek egyenrangú versenytársai a szabadalmaztatott szoftvereknek. A nyílt forráskódú modell alapjában véve „pull”, azaz húzó stratégiát követ: a közbeszerzésben döntő hivatalnoknak először tisztában kell lennie az alapokkal, hogy fel tudja tenni első kérdéseit. A nyílt forráskódú modell, szemben a hagyományos szoftverek gyártóival – ahol egy átlagos nagy cég esetében a cég alkalmazottainak 1/3-a fejlesztő, 2/3-a értékesítő, piacelemző, vezető és jogász – nem rendelkezik hasonló erőforrásokkal. Ez által felmerül egy nagyon találó kérdés: ha a szabadalmaztatott szoftverek olyan jók, és minden fejlesztőre 2 plusz „piaci” ember jut, akkor miért próbálják meg a kereskedelmi szoftverek gyártói eltántorítani a kormányzati alkalmazottakat a lépésről lépésre történő összehasonlítástól?

2.1. Szabadalmaztatott szoftver versus nyílt forráskódú szoftver

Az elkövetkezendő években a mobil eszközök és az ipari számítástechnika területein várható a nyílt forráskódú szoftverek további előretörése. A nyílt forráskódú szoftverek növekvő térhódítása lehetővé teszi, hogy az USA-n kívüli piaci szereplők is hatást gyakorolhassanak az IT infrastruktúra jövőbeli alakulására.

Az előnyök és hátrányok elemzésekor (bővebben: [2], [3]) tekintetbe kell venni, hogy ami az egyik oldalon előnyként jelentkezik, az a másik oldalon általában hátrányként mutatkozik meg. Az összehasonlítás nehézségeit jól érzékeltetik Lord Kelvin szavai: „Amennyiben mérni tudjuk, amiről beszéltünk, és számokban tudjuk kifejezni, tudunk róla valamit. Amennyiben nem tudjuk mérni, amennyiben nem tudjuk számokban kifejezni, a róla alkotott ismereteink szegények és nem kielégítőek; ez esetleg a

tudás kezdete, de aligha jutottunk el a gondolatainkban a tudomány szintjére." A hagyományos kereskedelmi és a nyílt forráskódú szoftverek összehasonlítása elég nehéz. Léteznek különböző indikátorok (megbízhatóság, teljesítmény, skálázhatóság, biztonság, TCO stb... és vannak nem számszerűsíthető indikátorok), amelyek mentén az összehasonlítás elvégezhető, azonban ezen indikátorok megválasztásával is befolyásolható a végeredmény.

Mindazonáltal a nyílt forráskódú programok előnyei világosak a – különösen az Amerikán kívüli – kormányok számára. Egy kormány sem akarja, hogy a számítógépes infrastruktúráját egy hatáskörén kívüli, teljesen idegen cég ellenőrizze.

2.2. Függőség

A függőség azt jelenti, hogy a felhasználó mennyire van kiszolgáltatva a szoftver szállítójának. Ez az a kérdés, ami miatt egyre több kormány teszi le a voksát a nyílt forráskód mellett. Nem akarnak egy külföldi, idegen vállalat piaci megfontolások szerint változó érdekeinek csapdájába kerülni. A függőség szorosan összefügg a biztonság kérdésével is, hiszen felmerül a kérdés, mennyire biztonságos olyan szoftvert használni, ami beépített hátsó bejáratot tartalmazhat, amelyen keresztül elérhetőek a felhasználó személyes és akár bizalmas adatai is. Megfontolást igényel az a kérdés is, mennyire biztonságos egy zárt formátumú állományban tárolni az adatokat. Jiang Guangzhi a Shangha-i Szoftver Fejlesztési Központ igazgatója azt hangsúlyozta, hogy a kínai kormány nem akar egy olyan vállalatot, amely „manipulálja, vagy uralja a kínai szoftverpiacot". Otto Schilly német belügyminiszter szerint – miután a minisztérium aláírt egy Linux megoldásról szóló megállapodást az IBM-mel – a változás segít lefaragni a költségeket, fejleszteni a nemzeti számítógépes hálózat biztonságát és alacsonyabb függőséget biztosít minden egyes szállítóval szemben.

2.3. Interoperabilitás

Az Európai Bizottság véleménye szerint az együttműködő-képesség munkálatainak a nyílt forráskódon kellene alapulnia. A munkaanyag[4], „Európa összekapcsolása: az együttműködő-képesség fontossága az e-kormányzati szolgáltatásokért" azt hangsúlyozza, hogy a tervezett európai együttműködő-képesség keretprogramjának „nyílt szabványokon kellene alapulnia, és ösztönöznie kellene a nyílt forráskódú szoftverek használatát."

2.4. Munkahelyteremtés

A nyílt forráskód üzleti modellje hasonló a specializált szolgáltatóiparhoz, a jogi, az orvostudományi, vagy a mérnöki tudományokéhoz. A kormányzati rendszerek nyílt forráskódú rendszerek felé történő elmozdulásának eredményeként több helyi, jól fizetett IT állás keletkezik az országon belül. A szoftver beszerzésre fordított pénzek, amelyeket az országon belül saját fejlesztésű szoftverekre költenek és nem fizetnek ki külföldi cégeknek, nyilvánvalóan növelni fogják az állam adóbevételeit, miközben költségmegtakarítás is elérhető vele.

3. Kormányzati trendek

Néhány kormány egyértelműen támogatja a nyílt forráskódú szoftvereket, míg néhány kormány semleges álláspontra helyezkedik. Sok kormány azért szereti a Linuxot, mivel így költséget tudnak megtakarítani, továbbá a nyílt forráskódú szoftvereket sokkal biztonságosabbnak tartják. Mindeközben a világ számos kormányát igyekszik meggyőzni a Microsoft arról, hogy veszélyes dolog nyílt forráskódú programokat használni az államigazgatásban. A Microsoft – mint képviselői mondják – nem ellenzi a nyílt szabványok alkalmazását, pusztán amellett érvel, hogy a kormányzati szervek és a központi költségvetésből finanszírozott akadémiai intézmények ne használjanak a GPL licenc betartásával készült programokat.

3.1. Magyarországi helyzet

Hazánkban a legnagyobb informatikai fogyasztó az állam. Emiatt nagy felelőssége van abban, hogy hogyan és mire költ e téren, ugyanis ezzel a piacot meghatározó módon képes befolyásolni. A jelenlegi magyarországi helyzet azt mutatja, hogy a nyílt forráskódú programok még nem kerültek a döntéshozók, politikusok látókörébe. Sok a félreértés a szabad szoftverekkel kapcsolatban, aminek eloszlatása az elsődleges feladata a szabad szoftveres közösségeknek. Erősen él az a tévképzet, hogy a szabad szoftver egyben ingyeneset is jelent, illetve, ami ingyen van, az nem is lehet olyan jó. 2003. elején a 11,25 milliárd forintos szoftverbeszerzési tender esetében sem került szabad szoftveres megoldás a győztesek közé. A legfontosabb lépés, a szabad szoftverek megismertetése, előnyeinek-hátrányainak bemutatása a döntéshozók részére. Az eddig történt események, a Parlament Informatikai Bizottságának meghallgatása, illetve az Informatikai és Hírközlési Minisztérium által szervezett civil fórum jó kiindulási pontot jelentenek.

3.2. Nemzetközi kitekintés

Inkább az Amerikán kívüli cégek és kormányok keresnek alternatívákat a Microsoft termékek helyett szoftver igényeikre, állapította meg a Gartner egyik jelentése. Az okok között szerepelnek a licencelési problémák és a biztonság miatti aggodalmak. A további okok között az anyagi megfontolások mellett, a legfontosabb kérdés a szállítóktól való függőség kérdése, illetve az interoperabilitás[5].

Az ázsiai, csendes-óceáni régióban Kína, Szingapúr, Malajzia és Ausztrália kormánya kifejezetten bátorítja a kormányszerveket és a vállalkozásokat a Microsoft termékek Linux és más nyílt forráskódú termékekkel és a helyi gyártók termékeivel történő kiváltására. Európa és Latin-Amerika sok országában is el akarják kerülni, hogy a GDP egyre nagyobb részét egy amerikai cég szerezze meg. Miután egyre több kormány fordul az olcsóbb és biztonságosabb nyitott forráskódú szoftverek felé, egyre szervezettebbé válnak a szoftvercégek, hogy ellenezzék a használatukat. 2002-ben Németország, Franciaország és Finnország voksolt a Linux mellett, de Anglia is fontolgatja az áttérést, a Microsoft új licencpolitikája miatt. A Microsoft pedig kényszerből már több helyen is árengedményre kényszerült – bár még nem közeli a veszély, hogy kiszorulhat a kormányzati piacról.

A következőkben az Európai Unió, az USA és Kína nyílt forráskódú programokkal kapcsolatos főbb erővonalai kerülnek rövid bemutatásra.

Erópai Unió

Nyitott forráskódú szoftverbank az e-kormányzati szoftverek területén

2002. július közepén jelent meg az Európai Bizottság tanulmánya a tagállamok elektronikus kormányzásban (eGovernment) használt szoftvereiről. A tanulmány, amelyet a kormányzatok közötti elektronikus kommunikációval és adatcserével foglalkozó EU-s program (IDA) finanszírozott – angol címe: „Pooling Open Source Software". A tanulmány megállapítja, hogy 28%-kal emelkedett a kormányzatok és az önkormányzatok információs technológiákra fordított kiadása, amely 2001-ben meghaladta a 6,6 milliárd eurót. A költségek korlátok között tartására érdekében javasolta a speciálisan e-kormányzat célokra kidolgozott szoftverek újrahasznosítását egy közös szoftverbankon keresztül. Várhatóan szükség lesz a hasznosításra felajánlott szoftverek nyelvi módosítására és nemzeti jogi környezetbe való beillesztésére, de így a költségkímélés mellett lehetővé válik az e-kormányzat szolgáltatások minőségének és hatékonyságának javítása. A szélesebb körű hasznosításnak előfeltétele az, hogy a kormányzatok számára speciálisan kifejlesztett szoftverek nyitott forráskódúak és szélesebb körben használhatóak legyenek. A szoftverkészítőkkel meg kell állapodni a felelősségi kérdésekről és a minőségi garanciákról is. A tanulmány a javasolt szoftverbank fokozatos felállítását ajánlja. Első lépésként fontosabb az e-kormányzat programokkal kapcsolatos tapasztalatok és a legjobb gyakorlatok átadása. A dokumentum javasolja, hogy a szoftverbank segítse a szoftverfejlesztők, felhasználók és a politikusok párbeszédét és együttműködését is[7] [8].

Az EU újabb szerződést kötött a Microsofttal

Az ingyenes operációs rendszer használatával Brüsszel a továbbiakban nem szorult volna a Microsoftra, miközben folyamatosan vizsgálatokat folytat ellene különböző trösztvádakra alapozva[10]. Bár az Európai Parlament és az Európai Bizottság tavaly év végétől több hónapon keresztül tesztelte a Linuxot, és nem zárják ki, hogy a jövőben alkalmazni fogják, azonban a nyílt forráskódú rendszer egyelőre alulmaradt a Microsoft programjaival szemben. Az EP egyik tisztviselője szerint, az nem elég biztonságos, és nem lenne képes rövid időn belül húsz nyelven működni. Az uniós számítógépes szakembereknek viszont nem voltak kifogásaik a Linux-szal szemben, sőt több tagállam – köztük Németország, Finnország és Franciaország – kormányzata és közigazgatása is áttért már az ingyenes rendszerre. A Microsoft szoftverei azonban hosszú távon lehet, hogy eltűnnek az EU több tízezer komputeréről, ám úgy tűnik, egyelőre nem sikerült feloldani az uniós elvek és a gyakorlat közti ellentmondást: a kényelem – vagy a Microsoft-lobbi – győzött az újító szellem felett.

USA

Fritz Schultz, a Szövetségi Információrendszer-védelmi Irodájának tisztségviselője azt mondta, hogy a nyílt forráskódú szoftverekkel kapcsolatos kormányzati politika leginkább úgy foglalható össze, „hagyd érvényesülni a versenyt mindaddig, amíg a nyílt forráskódú és szabadalmazott rendszerek egyenlő elbírálás alapján vannak értékelve. Inkább, mint hogy kifejezd az értékítéletedet a nyílt forráskód mellett, vagy ellene". Hozzátette, hogy „a minisztérium a legjobb szoftvert szeretné használni a munkájához".

A Microsoft időnként közvetlenül is beavatkozik a kormányzati szerveknél folyó munkákba. Az Initiative for Software Choice (ISC) elnevezésű amerikai ágazati szövet-

ség figyelmeztette az USA védelmi minisztériumát (DoD), gondolja újra az ingyenes és nyitott forráskódú szoftverekkel kapcsolatos álláspontját. Az ISC webhelyén közzétett figyelmeztetés a MITRE[6] által 2002. november 6-án „Nyílt forráskódú szoftverek használatáról a Védelmi Minisztériumban" címmel megjelentetett jelentésre reagál, amely szerint a nyitott forráskódú szoftverek az egyedi szoftvereknél jobban használhatók a DoD kritikus fontosságú rendszereiben. Az ISC felhívja a Pentagon figyelmét, hogy „téves az a prekoncepció, amely azt sugallja, hogy a nyitott forráskódú szoftverek lényegüknél fogva biztonságosabbak". Az USA védelmi minisztériumának intézete, a Defense Information Systems Agency engedett a cég kérésének, hogy vizsgálja felül az intézetnek készült jelentést. Az ügynökség vezető informatikusa rábírta a jelentést készítő Mitre Corporationt a dolgozat említett következtetésének „finomítására".

2002. októberében a nyitott forráskód hívei támadást intéztek a CompTIA (Computing Technology Industry Association) és az ISC ellen, hogy megpróbálják kizárni a kormányzati piacról a nyitott forráskódú szoftvereket a kormányzati szoftverbeszerzések „tisztaságának" hirdetése ürügyén. Mindkét szervezet elutasította a nem is burkolt vádat, hogy a Microsoft szócsöveként működnének, mondván: ők csak azt a gondolatot népszerűsítik, hogy a kormányzat olyan szoftvereket vásároljon, amelyek a legjobban megfelelnek igényeinek. Az ISC közleménye a General Public License elvet is kifogásolta, mert kockázatosnak tartja olyan szoftverek használatát biztonságos környezetben, amelyek kódja szabadon módosítható.

Kína

Kormányszervezetek és tisztviselők Kínában és Indiában támogatásukat fejezték ki a Linuxnak, amely sokkal olcsóbb, mint a Windows és emiatt alkalmas a fejlődő országok számára. Nem csak az alacsony költség motivál néhány kínai hivatalnokot arra, hogy adjon egy esélyt a Linuxnak. A nyílt forráskód is jelentős ösztönzést jelent. A Microsoft hosszú ideje őrzi a Windows alkotóelemeit, és ez érzékeny kulturális pontot jelentett néhány kínai számára, akikben felmerült a gyanú az amerikai óriás megbízhatóságával kapcsolatban.

A Red Flag Linux[9] , – egy pekingi központú Linux szoftver- és szolgáltatás disztribútor – a központi kormányzat csúcskutató intézetéhez, a Kínai Tudományos Akadémiához kapcsolódott. Ennek ellenére a Red Flag aligha nevezhető idegengyűlölőnek. 2002. októberében partneri kapcsolatot jelentett be egy másik amerikai céggel, a Sybse-sel, hogy e-business termékeket fejlesszenek Linuxra. Miért választaná Kína a Linuxot? Red Flag magyarázata szerint: „A Windowst a végfelhasználóknak forráskód nélkül adják el. Ez azt eredményezi, hogy a végfelhasználók mit sem sejtenek a biztonsági hiányokról és résekről, amelyek a szoftverrel együtt járhatnak. Nyilvánvaló, hogy az átláthatóság és a biztonság ezen szintje nem találkozik a kormányzat adminisztrációs igényeivel. A rendszer lehetséges biztonsági réseinek rosszindulatú kihasználása, az ország fontos gazdasági és katonai információit a felfedés komoly kockázatának veszélyébe sodorhatja. Ennélfogva a kínai készítésű operációs rendszereknek kellene az elsődleges választásnak lenniük az e-adminisztráció számára."

„A kormányzat Kínában agresszív módon támogatja a Linuxot" – mondta Huang a Hewlett-Packard kínai és hongkongi szerver divíziójának elnöke. „Amikor szóba kerül a titkosítás és más biztonsági használat Kína a fejlődő világ előtt jár". A katonaság a legelsők között volt, akik adoptálták a Linuxot, a nulláról építve ki a rendszereket. Az egyik ok, ami miatt a kínai néphadsereg olyan aktív a Linux fellendítésében, mivel Kína korlátozásokkal néz szembe a high-end technológiák területén, mint például a szuperszámítógépek. A Linux használata alternatívát jelent a katonaság számára.

4. Gazdasági megfontolások

Közgazdászok felteszik a kérdést, hogy a piac hibázott-e és szükség van-e (kormányzati) beavatkozásra a piaci folyamatokba, vagy Adam Smith „láthatatlan kéz” elmélete alapján normális piaci folyamatok zajlanak. Abban egyetértés mutatkozik a kutatók között, hogy a hagyományos kereskedelmi és a nyílt forráskódú szoftverek közötti versenyt és az egyenlő bánásmódot biztosítani kell. Ez nem jelenti a piaci folyamatokba való beavatkozást. Egyes nézetek szerint azonban szükség van a központi kormányok támogatására a nyílt forráskódú szoftverek elterjesztését, és használatát illetően.

Biztosítani kell a nyílt forráskódú programok azonos elbírálását a közbeszerzés során, és azt abból nem lehet kizárni.

5. Egy mintastratégia elemei: Dél-Afrika

Lelkesen támogatja a dél-afrikai kormány a nyílt forráskódú szoftverek használatát, melyeket az egyedi szoftverek költséghatékony alternatívájának tekint a kormányzati és közszolgálati munkában. Nem csak a több milliárd randnyi megtakarítás a nyílt forráskód egyetlen előnye, hanem az is, hogy nincs kiszolgáltatva a felhasználó meghatározott szoftvergyártóknak, ugyanakkor segít az informatikai képzésben, és végső soron hozzájárul a növekedéshez és a fejlődéshez. A Közszolgálati Központ és a Kormányzati Informatikusok Tanácsa igyekszik felvenni a harcot a tévképzetekkel szemben, oktatást szervez, és keretet kínál az iparral együttműködve ahhoz, hogy lerakják az alapjait a valódi nyílt forráskódú szoftverek fejlesztésének. Ha van hátránya a nyílt forráskódnak, az csupán az, hogy a nyílt forráskódú szoftveripar Dél-Afrikában még fejletlen, megoldásra vár a támogatás, a képzés és a tanúsítás kérdése – ezt azonban a kormányzat, az ipar és a szakértői testületek közösen meg tudják oldani.

A nyílt forráskódú szoftverek kormányzati alkalmazását vizsgáló központ[12] Dél Afrika nyílt forráskódú stratégiáját állítja követendő példaként a kormányok elé a következő okok miatt:

- *A nyílt forráskódú szoftverek legitimációjának felismerése és ennek hivatalos kinyilatkoztatása.* A nyílt forráskód alkalmazása új jelenség, így a megfelelő stratégia első lépése nyílt forráskód szoftverfejlesztési módszertanának el- és felismerése.

- *Speciális kormányzati ügynökség létrehozása a nyílt forráskódú fejlesztések vezetésére, koordinálására.* A megfelelő nyílt forráskód stratégia magas kormányzati szintre delegálja a szakmai hozzáértést és elszámoltathatóságot. A hivatalnok felelős a koordinációért, a kommunikációért és a program gyakorlati megvalósításáért.

- *Azonos feltételek biztosítása a közbeszerzések során.* A kormányzati IT beszerzési stratégiának biztosítania kell a versenyt anélkül, hogy direkt, vagy indirekt módon előre eldöntené a győztest annak eredményeként, hogy nincs előre meghatározott közbeszerzési preferenciája. Miután az azonos feltételeket biztosította, a döntést a technikai szempontok alapján kell meghoznia mind a hagyományos kereskedelemi, mind a nyílt forráskódú szoftvereknek azonos elbánást biztosítva.

- *A nyílt forráskódú szoftverek társadalmi értékének elismerése.* A szociális haszon például az állampolgárok szélesebb-körű hozzáférése a kormányzati információkhoz, az átláthatóság, a hazai szoftveripar megerősítése, jobb oktatás és tréning a hazai IT szakemberek számára.
- *Megvalósítási szakasz.* A nyílt forráskódra történő átállás nehézségei miatt a programnak biztosítania kell a pilot programok lehetőségét, az oktatást és tréninget, a kísérletezést stb. . .

Hivatkozások

[1] Nyílt forráskódú szoftverek közbeszerzési politikája Tony Stanco előadása nyomán http://www.govtech.net/magazine/story.phtml?id=48258

[2] David S. Evans and Bernard Reddy: Governement preferencies for promoting open-source software: A solution in search of a problem http://ssrn.com/abstract_id=313202

[3] Analysis of the Impact of Open Source Software: http://www.govtalk.gov.uk/documents/QinetiQ_OSS_rep.pdf

[4] Linking up Europe: the importance of interoperability for e-governement services http://europa.eu.int/ispo/ida/export/files/en/1523.pdf

[5] Governments seek Microsoft alternatives http://www.electricnews.net/news.html?code=9360804

[6] A MITRE Corp. jelentése „A nyílt forráskódú szoftverek használatáról a Védelmi Minisztériumban” http://www.egovos.org/pdf/dodfoss.pdf

[7] Free / open source software actions in European programmes http://www.cordis.lu/ist/ka4/tesss/impl_free.htm

[8] EU IDA programja http://europa.eu.int/ispo/ida/

[9] Red Flag Linux http://www.redflag-linux.com/eindex.html

[10] Az unióba is betörhet a Linux? 2003. május 23. Magyar Hírlap

[11] Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers! http://www.dwheeler.com/oss_fs_why.html

[12] The Center of Open Source & Government http://www.egovos.org/

[13] Dél Afrika OSS stratégiája http://www.oss.gov.za/docs/OSS_Strategy_v3.pdf

Az összes felsorolt on-line hivatkozás 2003. szeptember 8-án elérhető volt.

Hazai pályázatok és a magyar részvételi lehetőségek az EU programjaiban

dr. Mlinarics József elnök
Magyar Tartalomipari Szövetség – MATISZ

2003.11.08.

Kivonat

A mai világban Magyarországon nagyon sokan úgy gondolják, hogy a sikeres vállalkozás nélkülözhetetlen feltétele az, hogy az összes lehetséges pályázaton indulnia kell. Az EU vállalkozások közül, azonban csupán 5% aki rendszeresen pályázik.

Az EU csatlakozással megnyíló új források (kb. 1300 milliárd Ft.), azonban 2004 évtől számunkra csak a pályázatokon keresztül érhető el.

Tartalomjegyzék

1. Hogyan lehet a szükségből erényt kovácsolni?

Mit tud tanácsolni a kissé „koros” (12 éve alakult) Magyar Tartalomipari Szövetség a fiatalabb „testvér” a Linux-felhasználók Magyarországi Egyesületének tagjai és a Linux Szakmai Konferencia részvevői számára.

Készítsük el a vállalkozás pályázati portfolióját és legyünk résen!

A pályázati portfolió tartalmazza a szervezet azon céljait, amelyek külső forrás (pályázati támogatás) bevonásával történő megvalósulása esetén olyan eredmény elérést segíti elő, amely a vállalkozás gazdálkodását, ismertségét javítja.

A pályázati portfolió tartalmazza a pályázati cél elérését biztosító források (pénz, eszköz, szaktudás, stb.) és a szükséges referenciákat és kompetenciákat is.

Az előadás megkísérli bemutatni a sikeres pályázatok készítésének alapelveit.

2. Alapelvek

1. A téma megválasztása
2. A pályázatok figyelése, kiválasztása
3. A pályázat elkészítése, benyújtása
4. A szerződéskötés, a projekt indítása
5. A projekt végrehajtása, részjelentés és elszámolás
6. A projekt befejezése, disszemináció
7. A projekt eredményeinek hasznosítása

3. Hazai programok

A pályázatírásra vállalkozók számára "rendelkezésre álló" hazai fejlesztési programok közül az alábbiakat mutatja be az előadó:

1. Informatikai és Hírközlési Minisztérium
2. Nemzeti Fejlesztési Terv
3. Oktatási Minisztérium
4. Nemzeti Kulturális Örökség Minisztériuma
5. Foglalkozáspolitikai és Munkaügyi Minisztérium
6. Gazdasági és Közlekedési Minisztérium

Az előadás külön foglalkozik az új K+F munkahely (EEF) és eszközbiztosítási (CSI) támogatásokkal is.

A EU programok közül több a hazai programok keresztfinanszírozásával (OM EUB, EUK) fejlesztési lehetőséget biztosíthat a szoftverfejlesztők számára is.

4. EU programok

A számításba vehető EU programok köre a következő:

1. eContent
2. 6. K+F keretprogram
3. CULTURE 2000
4. LEONARDO DA VINCI
5. SOCRATES
6. eLearning
7. MEDIA PLUS

5. Zárszó

Bővebb információk a MATISZ Hírlevelekben `http://www.matisz.hu/`
Feliratkozás hírlevélre: `info@matisz.hu`

Hivatkozások

[1] `http://www.magyarorszag.hu/`

[2] `http://www.kulugyminiszterium.hu/`

[3] `http://www.nfh.hu/`

[4] `http://www.vati.hu/`

[5] `http://www.ihm.hu/` (pályázatok,/eContent)

[6] `http://www.om.hu/` (kutatás-fejlesztés)

[7] `http://www.nkom.hu/`

[8] `http://www.fmm.gov.hu/`

[9] `http://www.gkm.hu/`

[10] `http://www.nive.hu/`

[11] `http://www.tetalap.hu/download.htm`

[12] `http://www.kulturpont.hu/`

[13] `http://www.tpf.iif.hu/`

[14] `http://www.euportal.hu/`

[15] `http://www.euinfo.hu/`

[16] `http://www.euoldal.hu/`

[17] http://eu.lap.hu/

[18] http://www.cordis.lu/

[19] http://europa.eu.int/

Scribus – DTP Linux alatt

Nagy Bence

2003.11.08.

Kivonat

Mintegy kétévnyi fejlesztés után ez év július 14-én jelent meg a Franz Schmid által készített Scribus tördelőprogram 1.0-ás verziója, mely hiánypótló alkalmazás a nyomdai előkészítés Linuxos lehetőségeit tekintve. Noha a TEX-rendszerrel már régóta lehetséges kiváló minőségű dokumentumok készítése, a Scribus teljesen WYSIWYG-rendszerű alkalmazás, lehetőségeit tekintve is a más platformokon népszerű QuarkXPress vagy Adobe PageMaker nyílt forrású alternatíváját jelenti. Az egységes szabványokra épülésen kívül a magyar kezelőfelület és elválasztás még az a többlet, amely némi előnyt is biztosít ezen programokkal szemben.

Tartalomjegyzék

1. Bevezetés

1984-ben az Aldus cég PageMaker szoftverével és az Apple LaserWriter nyomtatójával beköszöntött az asztali kiadványszerkesztés kora. Az ezt követő időkben a DTP és a Macintosh számítógépek szorosan kapcsolódtak egymáshoz, mire mintegy egy évtizeddel később a 32 bites Windows megjelenésével egy időben a PC-architektúra is egyenrangúvá vált. 2003-ban, két év fejlesztés után megjelent a Linuxon futó Scribus szoftver 1.0-s verziója, ezzel a DTP evolúciója a drága architektúra/drága szoftver kombinációtól az olcsó architektúra/drága szoftveren át eljutott az olcsó architektúra/olcsó szoftver változatig.

Visszaemlékezve a Macintosh-felhasználók öt-tíz évvel ezelőtti fanyalgására, hasonlóra számíthatunk ez esetben is, mivel a program használhatósága még nem éri el a kereskedelmi rendszerekét. Azt azonban már most el kell ismerni, hogy a Scribus a nyílt forrású alkalmazásfejlesztés egyik diadala.

2. Bemutatás

A Scribus [1] fejlesztője Franz Schmid. Általános célú oldaltervező (page layout) alkalmazás, a magyar nyelvben talán a tördelőprogram kifejezés használható még rá, de semmi esetre sem a túl sok mindenre ráhúzható kiadványszerkesztő fogalom. A Scribus ugyanis rendelkezik a professzionális nyomdai előkészítést támogató eszközökkel, és ez az a különbség, ami megkülönbözteti a „kiadványt szerkeszteni képes” egyéb programoktól.

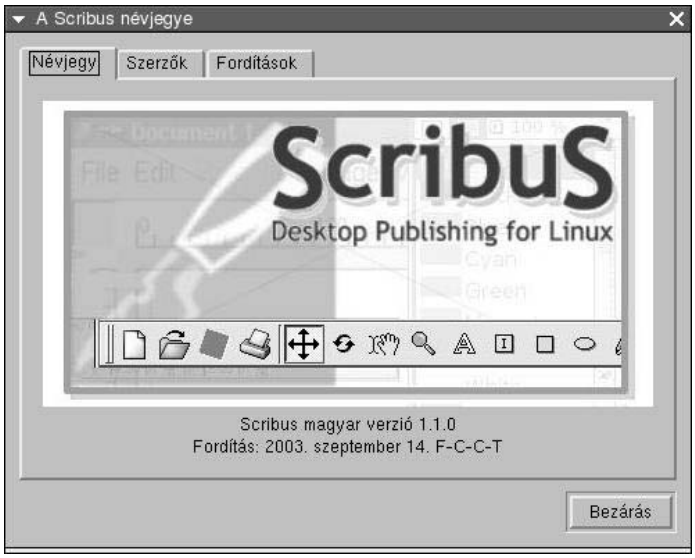
A nyílt forrás diadaláról azért beszélhetünk a Scribus esetében, mert számos nyílt szoftvert és fejlesztői könyvtárat használ fel működéséhez. A fordításhoz szükséges csomagok alapján könnyű felvázolni a program felépítését és ezek fényében képességeit is: Qt, XFree, libstdc++, libtiff, libpng, zlib, libfreetype2, lib-cups, liblcms, gimp-print, python-2.1 (az elnevezések alatt a disztribúciótól függő *-devel* csomagokat kell érteni, illetve saját fordítás esetén a komponens telepítése szükséges). A Scribus tehát igazi puzzle-alkalmazás, ennek következtében fejlesztése gyors, valamint csak a hozzáadott kód hibamentességének feladata hárul programozóira.

A program képességeinek leírásához jó út a beépített programkönyvtárak áttekintése. Az 1. ábrán látható a program névjegy ablaka, ahol a F-C-C-T felirat a beforgatott FreeType2, CUPS, LittleCMS és LibTiff komponensekről árulkodik.

A Qt a Trolltech által fejlesztett grafikus komponenskönyvtár, legfőbb alkalmazása a KDE felhasználói felület. Mivel a Scribus a Qt grafikai elemeit használja, ezért legjobb formáját KDE-környezetben hozza, itt futtatva valódi drag&drop másolási és beillesztési lehetőséget kapunk, de a program Gnome vagy más ablakkezelő alatt is használható. Teljes értékű WYSIWYG-megjelenést nyújt, és noha Linuxon a TEX/LATEX-rendszer is tördelőprogramnak tekinthető, oldaltervező alkalmazásként ez nem jöhet számításba, és így az úttörő szerep mindenképpen a Scribus-nak jut.

A FreeType2 programkönyvtár használatával a 4.3-as verziójú XFree86 alatt élsimított szövegmegjelenítést kapunk, valamint az Unicode-szabvány szerinti fontokban elérhetjük valamennyi karaktert és jobbról balra is írhatunk. A 2. ábra a Bigelow and Holmes által közzétett Luxi Serif betűtípus felhasználható karaktereit mutatja. A program jelenleg a Type1 és TrueType betűkészleteket kezeli, a jövő betűformátumát jelentő OpenType készleteket egyelőre nem. Ez a jelenleg opcionális csomag a jövőben szükséges csomaggá fog változni.

A libstdc++ csomag alapján nem nehéz kitalálni, hogy a program legnagyobb részét



1. ábra. A Scribus névjegye

2. ábra. Speciális karakter beszúrása

C++-nyelven írták, csupán néhány C-nyelvű programkönyvtár beépítése történt meg. A program kódja ezért jól áttekinthető, és csak minimális árat kell fizetni ezért a fordítási idő hosszában.

A libtiff és libpng csomagok arról árulkodnak, hogy a program kezeli a TIFF és a PNG formátumú képeket. Ezen evidens formátumokon kívül a Qt 3-as verziója által ismert összes képformátum is felhasználható.

A zlib csomag a PDF-exportálás funkciónál kap szerepet, a kimeneti fájlok egyes adatfolyamait tömöríti.

A CUPS és a Gimp-Print a kényelmes nyomtatáshoz szükséges, a programból elvégezhető a feltelepített nyomtató beállítása éppúgy, mintha webes felületről hajtanánk végre ugyanezt.

3. Képességek

A program saját dokumentációjának bevallása szerint is a QuarkXPress, a Corel Ventura és az Adobe Pagemaker/InDesign alkalmazásokra hasonlít, és képességei révén méltán állítható egy súlycsoportba ezekkel. Valamennyi jellemzőjének áttekintése helyett csak azokat emelem ki, amelyek a Scribus-t igazán professzionális alkalmazássá teszik.

3.1. Színkezelés

A Scribus a nyílt forrású LittleCMS színkezelő rendszert [2] tartalmazza. Professzionális oldaltervező szoftverek esetében nélkülözhetetlen a használata, ugyanis ezzel az eszközzel érhető el, hogy a számítógépen létrehozott és megjelenített színvilág nyomdai úton reprodukálva is ugyanúgy jelenjen meg.

A képernyőn megjelenő fényszíneket az RGB-színtér (vörös–zöld–kék) szerint írjuk le, és az additív keverés szabályai érvényesülnek, míg a nyomdai eljárás során keletkező festékszíneket a CMYK-rendszer (cián–bíbor–sárga–fekete) szerint, és itt szubsztraktív módon keverhetőek ki újabbak. A két rendszerben szereplő színek egy-egy másik rendszerbeli szín összeadásával kaphatóak meg.

A CMYK-rendszerben a külön fekete szín alkalmazását az teszi szükségessé, hogy a három alapszín keverésével a nyomdafestékek tökéletlen fényvisszaverő képessége miatt nem érjük el a teljes árnyalatterjedelmet. Tovább bonyolítja a helyzetet, hogy a három összetevő egyenlő arányban történő keverése szürke árnyalatot eredményez, melyet az adott mennyiségnek megfelelő árnyalatú feketével helyettesíthetünk, és figyelembe kell venni azt is, hogy a nyomathordozót annak anyagától függően csak meghatározott mértékig terhelhetjük festékkel.

A színkezelő rendszer feladata a két színtér közötti átalakítás vezérlése. A kimenet során biztosítania kell, hogy a különböző eszközök ugyanazon eszközfüggetlen színinformációkból a saját működésük szerinti kimeneti színeket állítsák elő, és ehhez az eszközt jellemző ICC-profilokra van szükség. Mivel általában a képernyő és a nyomtató által használt színtartomány eltér egymástól, a képernyőn jelezni kell, ha egy szín az adott kimeneti eszközön nem reprodukálható (gamut color), ezeket mind beállíthatjuk a Scribus-ban.

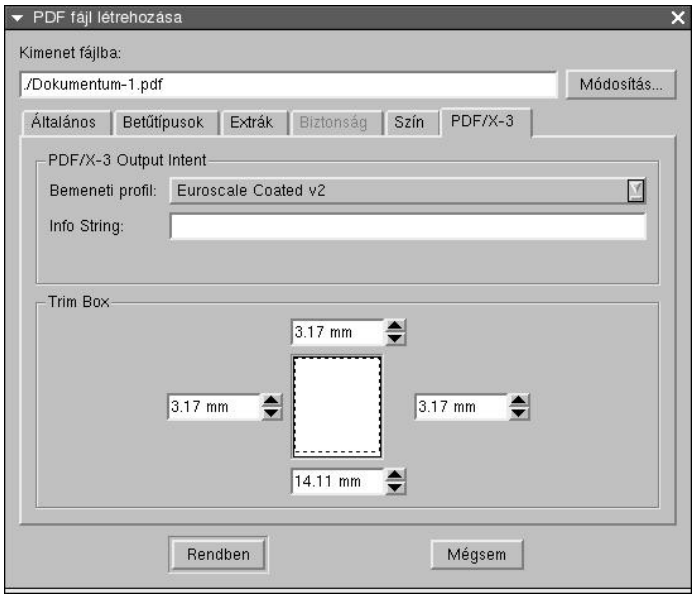
3.2. PDF-támogatás

A nyomdai előkészítés vezető formátumává váló PDF-formátum PDF/X-3 változata ISO 15930-3:2002 néven immár ISO-szabvány is.

A 3. ábra a PDF-exportálás ablakának egyik fülét mutatja, a fenti *Euroscale Coated v2* felirat arról tanúskodik, hogy a fájl az európai színgarnitúrának és a műnyomó papírnak megfelelő színprofilt fogja hordozni.

A program bementi és kimeneti formátumként is használni tudja a PDF-et, exportáláskor speciális funkciókat is igénybe vehetünk: védhetjük a dokumentumot bizonyos felhasználásoktól (megtekintés, nyomtatás, megváltoztatás) vagy akár prezentációs effektekkel is bővíthetjük.

A Scribus támogatja a PDF interaktív alkalmazását is, dialógusablakok, nyomógombok helyezhetőek el az oldalakon, és ezekhez Javascript-kódokat rendelhetünk.



3. ábra. PDF-exportálás

3.3. Színbontás

A Ghostscript 8.0-ás verziója révén a Scribus maximum 8 különböző színkivonat elkészítésére használható, ezzel felülmúlja a PageMaker 7-es verzióját, melynek Pantone Hexachrome rendszere mindössze 6-ra képes. Azt azért meg kell jegyezni, hogy a magyar nyomdaipar nem bővelkedik még hatwerkes gépekkel sem, és a fizetőképes kereslet sem nagy az ilyen extra minőség iránt.

3.4. Programozhatóság

Külön letölthető a programhoz a Python Scripting Plugin, amivel egy python-interpretert kapunk a programon belüli használatra. Ez által egyszerűbb makrók és bonyolultabb programok is írhatóak, mellyel az ismétlődő feladatokat automatizálhatjuk.

A programban teljes értékű python programokat írhatunk, vagy már meglévő szkripteket olvashatunk be, és a nyelv egyszerűségéhez hasonlóan egyszerű API áll rendelkezésre.

4. A program magyarítása

4.1. A magyarítás alapelvei

Egy szoftver magyarítása során két kritériumnak kell megfelelni: az első a magyar nyelvű kezelőfelület, a második a magyar sajátosságok szerinti működés, ez utóbbi tördelőprogramok esetében a helyes magyar elválasztás és a magyar tipográfiai hagyományok szerinti beállítás.

4.2. A kezelőfelület

A Scribus a nyelvi változatokat a gettext segédprogramjainak Qt-s megvalósításán keresztül támogatja. Működési mechanizmusa vázlatosan a következő: a programban megjelenő sztringek a `tr()` függvényen keresztül átengedve jelennek meg a képernyőn. Ahhoz, hogy a `tr()`-függvény egy adott sztringeket lefordítson, létre kell hozni egy olyan fájlt, melyben ezek a sztringek és fordításaik meghatározott szintaktika szerint szerepelnek. Ezt a `.po`-végződésű fájlt kézzel vagy célalkalmazások segítségével (pl. KBabel) szerkeszthetjük. Az `msg2qm` programmal lefordítva megkapjuk a program által használható `.qm`-végződésű fájlt.

Ez az eljárás a Qt 2-es verziójában volt használatos, és a program fordításához ajánlott 3.1-es verzióban a bináris formátum megváltozása miatt ez az alkalmazás már nem lelhető fel. A `qt3-dev-tools-compat` csomag szerencsére orvosolja ezt a problémát.

A Scribus eredeti verziója az operációs rendszer által használt nyelvi beállításokat veszi figyelembe, vagy a `scribus --lang xx` paraméteres formával futtatva a kívánt nyelven indul el. Nem csak egyszerű fordításról beszélhetünk abban az esetben, amikor egy feliratban valamely változó értéke is megjelenik. Noha a sztringeket lefordítja a program, de a szöveg és változó egymásutánisága gyakran eltérő a magyarban.

Egy példával szemléltetem: a dokumentum elején állva az angol változatban az állapotsor alján a *Page 1* felirat látható, erre kattintva egy felnyíló gyorsmenüből kiválasztva léphetünk másik oldalra. A *Page 1* felirat a *Page* sztringből és aktuális oldalszám változójából áll. A *Page* szó fordítása után magyar nyelvű beállítások esetén az *Oldal 1* felirat jelenik meg a nyelvi szokásaink szerinti *1. oldal* helyett. A *Page* szó több helyen is megjelenik a programban, ezért a nagybetűs *Oldal* fordítást nem cserélhetjük le kisbetűsre. A megoldást egy speciálisan magyar patch jelenti, amely kizárólag a magyar verzióba kerül beépítésre, ennek részlete:

```
-~LA->setText(tr("Page")+"~"+QString::number(Seite+1));
+~LA->setText(QString::number(Seite+1)+".~oldal");
```

Ennek a megoldásnak a hátránya, hogy hiába váltanánk át másik nyelvre, mivel a magyar feliratot beégettük a programba, ezért az látszana minden esetben. Mivel speciálisan magyar nyelvű verzió lett kialakítva, ezért kizárólag magyar nyelvű kezelőfelületet lehet használni, így az ehhez hasonló patch-ek nem zavarhatnak bele a más nyelvű használatba.

4.3. Beállítások

A számítógépes tördelőprogramok megjelenésével a Didôt-mértékrendszer eltűnt, és helyette az angolszász pica-rendszer vált egyeduralkodóvá. Noha a kettő közötti méretekben mindössze 7% a különbség, az egyes tipográfiai méretrendszerek a származási országaik mértékrendszereivel összhangban alakultak ki: a pica-pont az angol inch 72-ed része, 24 Didôt-pont pedig 9 mm-rel egyenlő.

Noha nálunk a tördelőprogramok alapbeállításainak hatására a betűméreteket általában pica-pontban adják meg, az oldal többi elemének leírásához a millimétert és nem a pica-pontot használják. Ezért a Scribus alapértelmezés szerinti mértékegysége a milliméterre lett változtatva.

4.4. Magyar elválasztás

A Scribus-ba épített elválasztó programkódot Raph Levien írta a TEX-rendszer megfelelő kódjának továbbfejlesztésével, és azt LGPL-licenc alatt nyílt forrásúvá tette. Ugyanez a programkód felelős az OpenOffice.org elválasztásaiért, viszont tördelőprogram esetén sokkal súlyosabb követelmény a helyes elválasztás, mint egy irodai alkalmazásnál.

Az algoritmus működése egyszerű: egy elválasztási mintafájlt kell létrehozni, melyben betűket és számokat tartalmazó karaktersorozatok szerepelnek, ahol a páratlan számok a lehetséges elválasztást, a párosok az elválasztások tiltását jelenti. Ha egy elválasztandó szóban szerepel egy elválasztási minta, úgy az abban lévő számokat rendeli a szó megfelelő betűihez úgy, hogy a magasabb számok felülírják az alacsonyabbakat.

Egy rövid példán szemléltetve: a *legelső* szó elválasztása esetén a `.le2g1`, a `1ge` és a `1ső` mintákat találja megfelelőnek, mivel ennek betűi egyeznek meg a szó egy-egy részével. A minta előtti és utáni pont azt jelzi, hogy csak a szó elején, illetve végén alkalmazható mintáról van szó. A három mintát illesztve szavunkra a `le2g1el1ső` karaktersorozatot kapjuk, ahol a páratlan számokat kötőjellé cserélve, a párosokat pedig törölve kapjuk meg a szó elválasztott alakját: *leg-el-ső*.

A Huhyphn projekt [3] keretében fejlesztett elválasztási mintagyűjtemény több mint 350.000 szó elválasztásának elemzésével készült, ráadásul ez az egyetlen olyan gyűjtemény, amely teljes egészében a Magyar Helyesírás Szabályai szerinti elválasztást nyújt. Ugyanez került alkalmazásra az OpenOffice.org-ban is, mely kiváló helyesírás-ellenőrzője révén egy Linux-alapú nyomdai előkészítő műhely másik fontos eleme lehet.

Hivatkozások

[1] Scribus: `http://web2.altmuehlnet.de/fschmid/`

[2] LittleCMS színkezelő rendszer: `http://www.littlecms.com/`

[3] Huhyphn projekt: `http://huhyphn.tipogral.hu/`

A Szószablya fejlesztés

Németh László

2003.11.08.

Kivonat

A Szószablya fejlesztés célja, hogy a magyar weboldalak szövegtartalma alapján magyar szógyakorisági-szótárat, majd ennek felhasználásával nyitott forráskódú morfológiai elemző programot készítsen.

A szótövezésre képes elemző megkönnyíti, illetve minden korábbi eszköznél jobban lehetővé teszi az internetes, vagy intranetes weboldalak, dokumentumok indexelését, ellenőrzését, egyéb automatikus kategorizálását, nem véletlen, hogy az ilyen célra történő vállalati felhasználása már elkezdődött.

A Szószablya fejlesztés alapjául szolgáló; kibővített, javított Magyar Ispell helyesírási szótár és Hunspell helyesírás-ellenőrző pedig a magyar OpenOffice.org révén hozzájárul a nyitott forráskódú programok hazai terjedéséhez. A magyar morfológia elemző pedig az informatika jövőt képviselő területein (gépi fordítás, mesterséges intelligencia) nyújt majd pótolhatatlan segítséget.

Tartalomjegyzék

1. A Szószablya fejlesztés

A Szószablya fejlesztés (`http://www.szoszablya.hu/`) 2003 márciusában indult a BME és a Matáv által létrehozott, a BME Szociológia és Kommunikáció Tanszék keretén belül működő Média Oktatási és Kutató Központ (MOKK) vezetésével.

Az egy évig tartó fejlesztés során alapvető, nyitott forráskódú (LGPL és GPL licencű) informatikai eszközök készülnek el a magyar nyelvhez. Ezek közül a legfontosabbak a *Hunspell* helyesírás-ellenőrző, a *Hunstem* szótövező, és a *Hunmorph* morfológiai elemző programkönyvtár és program. Elkészül a magyar weboldalak szövegtartalma alapján egy több millió szót tartalmazó *gyakorisági szótár* is. Elérhetővé válnak a a weboldalak feldolgozása során használt segédprogramok (*Hunnorm* szövegátalakító és -normalizáló, *Huntoken* mondatra és szóra bontó); a *Hunp* nyelvfelismerő alkalmazás és más kisebb eszközök.

A következőkben azoknak az eredményeknek az ismertetésére kerül sor, amelyek már most is elérhetők, szabadon felhasználhatók, és (különösen az indexelésre használható tövező esetében) sok Linux fejlesztőt érintenek.

2. Hunspell helyesírás-ellenőrző

A Szószablya fejlesztésnek köszönhetően a Hunspell helyesírás-ellenőrző a helyesírási szabályoknak való megfelelés, és a kezelt szókincs tekintetében is sokat lépett előre. A Hunspell szótárának 0.96-os (2003. február) és a 0.99.4-es (2003. szeptember) változatának összehasonlítása alapján megközelítőleg 99,4%-ról 99,8%-ra sikerült növelni a program szövegellenőrzési képességét, vagyis míg korábban 1000 átlagos szövegszóból több, mint 6 helyes szót nem ismert fel az ellenőrző, most ez a szám körülbelül kettőre, vagyis harmadára csökkent.[1]

A Szószablya hozadéka a mintegy félmillió kiváló minőségű weboldal szókincse alapján készült 4 millió szavas tesztszótár, amivel biztosítani lehet a helyesírási szótár minőségét a folyamatos fejlesztések során.

A Hunspell dokumentáció [10] részletezi az ellenőrző magyar helyesírási szabályoknak való megfelelését, összevetve a legelterjedtebb magyar helyesírás-ellenőrzővel is. Az MS Office XP-ben található Helyes-e? helyesírás-ellenőrző a következő nagyobb hiányosságokat mutatja a Hunspell-lel szemben:

1. Alapvető helyesírási szabályokat nem ismer. Helyesnek veszi a következő típushibákat: **lássd*, **írdd*, **fessd*, **késsd*; **mögémegy*, *elévág*, *föléüt*, *teliönt*; **melegvíz*, **szépvers*, **magyarállam*; **fázósgyerek*, **egércincogott*, **olymacska*; **észszerű*, **mészszerű*, **viaszszerű*; **feddd*, **pufffféle*, **frissség*, **fessség*, **izzzad*, **továbbbotorkál*, **játszza*; **Einsteinféle*, **Budapestszerű*.
2. A magyar nyelvtan kezelésében nem pontos, ezért elfogadja a következő típushibákat: **kézéé*, **vízéé* **tűzéé*; **bonttat*, **kértet*, **nyitat*, **oktatatás*; **hölgyök*, **tölgyök*, **szüzök*; **hatalomi*, **birodalomi*; **meghalja*, **történed*, **évődi*; **kézség*, **jelentősség*, **adóság*, **tartóság*; **újnyi*, **nyulnyi*, **lovnyi*, **bokrnyi*; **bokré*, **terhé*; **tesszél*, **visszétek*, **essz*; **tettról*, **hitthoz*, **vészja*; **fiája*, **híjája*, **őfenségéje*; illetve elutasítja: *Józsefné*, *Ferencné*; *Béláék*, *Péterék*, *unokámék*; *zsenijei*, *tepsijei*; *New York-iakat*; *1.-t*, *11.-et*.

[1] Kevesebb, mint 4% helyesírási hibát (beleérve ebbe az idegen és ismeretlen szavakat is) tartalmazó weboldalak szókincsére vonatkozik ez a becslés. Ez magában foglalja a magyar nyelvű napilapokat és a szépirodalmi műveket is.

3. Egyéb súlyos helyesírási hibákat is elfogad (például **estéji*, **karvaj*, **szeméji*, **ünnepéjes*; **csevely*, **bolyár*, **bolytár*, **súlytó*; **elitélt*, **hivő*, **izület*, **neutrinó*, **egyivású*, **tikfa*, **hivat*, **irogat*; **fűzek*, **színtű*, **bíztat*, **kompatíbilis*; **hiu*, **savanyu*, **ramazuri*, **harangbugás*; **tinórú*, **kultúrált*, **kultúrálatlan*, **kultúrálódik*; **egyűvé*, **szűntető*, **tűntető*, **kűzdőtér*; **diagramm*, **vállfaj*, **zsupptető*, **mennyhal*; **analfabétizmus*, **polémikus*, **siserehad*, **bédekker*), sőt néhol ezzel együtt a helyes alakokat el is utasítja (például *mésszerű*, *viasszerű*; *csevej*, *puzón*; *Einstein-féle*; *trabantos*; *Rio Janeiró-i*; *camembert-rel*).
4. Szókincse új szavakat nem tartalmaz. Nem ismeri például: *EU*, *ombudsman*, *klónozás*, *valóságshow*, *szafari*, *sztárol*, *kapucsínó*, *nonprofit*, *globalizáció*.
5. Szókincse a gyakori szavak tekintetében is hiányos: *karalábé*, *megnövekedett*, *mondotta*, *közzétesszük*, *legősibb*; *ebtartás*, *ebtenyésztés*; *hőszabályzó*, *hőháztartás*; *síkesztyű*, *síbaleset*; *sólepárlás*, *sómennyiség*; *ízérzékelés*, *ízfokozó*; *hókotrás*, *műhó*; *eperlé*, *léalma*; *aggaszt*, *apaszt*, *áraszt*, *ébreszt*, *fagyaszt*, *függeszt*, *horgaszt*, *kepeszt*, *lehiggaszt*, *sorvaszt*, *szállaszt* stb.
6. Szókincse nem terjed ki a magyar településnevekre (még a városokéra sem: *Gyomaendrőd*, *Szécsény*, *Vásárosnamény*, *Zalaszentgrót* stb.).
7. Szókincse nem terjed ki a szakszavakra. Hiányzik például: (matematika) *elempár*, *variancia*, *alterek*, *bijektív*, *attraktor*, *Cauchy*; (informatika) *alhálózat*, *élkiemelés*, *mikrocsip*, *flopi*, *Linux*; (biológia) *póc*, *domolykó*, *csilló*, *füzény* stb.
8. Elavult ékezet nélküli formában, vagy egyáltalán nem tartalmazza az idegen ékezetes szavakat. Hiányzik például: *Händel*, *Dvořák*, *Škoda*, *złoty*) stb.
9. Értelmetlen szavakat tartalmaz. Például: **veé*, **sepr*, **sodr*, **késsz*, **mis*, **fesz*, **süv*, **lépt*, **tel*.
10. Javítási képességei a tipikus hibák csak csekély részére terjednek ki, szemben a Hunspellel, ami minden egy karakter távolságra lévő hibát, valamint az összes tipikus több karaktert érintő hibát javítja.
11. Zárt program. Alapszókincse a felhasználók által nem bővíthető, szemben a nyitott forráskódú Hunspell-lel, aminek parancssori változata az olyan tőszavak ellenőrzés közbeni felvételét is támogatja, amelyeket az ellenőrző az alapszavakhoz hasonlóan képes toldalékolt alakban is felismerni.

Az összehasonlítás alapján határozottan javasolható az MS Office lecserélése a Hunspell programkönyvtárát és helyesírási szótárát tartalmazó OpenOffice.org-ra [3].

3. Hunstem szótövező

A Hunstem szótövező programkönyvtár és alkalmazás a Hunspell forráskódjának és helyesírási szótárának bővítésével készül. Első komolyabb változata a valós nagyvállalati igényeknek megfelelően fel van készítve a szálkezelésre, és képes ismeretlen szavak feltételezett töveinek megállapítására, ami elsősorban tulajdonnevek és szakszavak tövezése esetében hasznos. A tövezés sebessége átlagos szövegen, AMD XP 1800+-os gépen 40 000 szó/másodperc, ami egy nagy méretű vállalati intranet változásainak folyamatos nyomon követését is lehetővé teszi.

A tövező programhoz elkészített mintaalkalmazás, illetve CGI program példát mutat webhelyek indexelésére, és helyi dokumentumok keresésére. A tövezésnek köszönhetően az indexállomány mérete lényegesen kisebb, valamint a módosuló tövet tar-

talmazó szóalakok kezelése sem okoz többé gondot (macska/macskát, bokor/bokrot, híd/hidat stb.). Egy komolyabb webhely esetén esetén a Hunstem használata kikerülhetetlen, de természetesen a tövező felhasználási területe nem korlátozódik az intra- és internetes információ-visszakeresés támogatására.

4. A fejlesztés tapasztalatai

4.1. Hardver- és szoftverkörnyezet

A magyar weboldalak feldolgozásáért felelős központi gép Debian GNU/Linux operációs rendszert futtat, 1 TB-os XFS fájlrendszerű háttértárolóval, 2 GB memóriával, 4 darab 1,8 MHz-es Intel Xeon processzorral rendelkezik.

A Szószablya fejlesztés során nyitott forráskódú programok kerültek felhasználásra. A magyar weboldalak a Larbin webrobottal lettek összegyűjtve [6]. A Hunnorm, illetve Huntoken feldolgozó szűrősor C-ben, illetve Flexben készült. A hatékonyság növelése céljából a szűrősor első tagja egy állományba kötegeli a feldolgozni kívánt (több millió) weboldalt, ami a szűrősor végén igény szerint újra állományokra bontható.

A gyakorisági listák a Glibc programkönyvtár tsearch() bináris fát kezelő függvénycsaládja segítségével készültek. Egyéb feladatokat pedig legtöbbször unixos segédprogramokkal meg lehetett oldani. Kisebb gondot okozott, míg ki nem derült, hogy a fejlesztői gép alapértelmezett awkja a mawk, ami milliós nagyságrendű adatok hasítótáblás kezelésében lényegesen rosszabb teljesítményt nyújtott, mint a GNU awk.[2]

A Hunspell helyesírás-ellenőrző a BSD licenccel rendelkező Myspell függvénykönyvtáron alapul, ami eredetileg az OpenOffice.org része. A Myspell az Ispell program működése alapján készült. Az Ispell (és így a Myspell) működése egy olyan mintafelismerő algoritmuson alapul, ami egy orosz nyelvű folyóiratban jelent meg először, és a folyóirat angol nyelvű kiadásával került be a köztudatba még a hatvanas években [7]. Az orosz nyelvű cikk írója Dömölki Bálint matematikus, aki meghatározó szerepet játszott a legendás magyar M–3-as számítógép megépítésében. A Dömölki-algoritmus az M–3-on folyó magyar nyelvészeti kutatások során született meg, közelebbről magyar költők műveinek pszicholingvisztikai, fonetikai elemzésekor [8]. A Dömölki-algoritmust használó Hunspell most a mai magyar köznyelvi szókincset tartalmazó szótár elkészítését teszi lehetővé.

4.2. A magyar web és feldolgozása

A letölthető magyar web durva becslés alapján 20–25 millió oldalból áll, amelynek jelentős része szövegtartalom nélküli, idegen nyelvű, vagy duplikált oldal. A weboldalak

[2] Az awk több helyen fel lett használva a Szószablya fejlesztésben: duplikátumszűrés, trigram adatok számítása, küszöb szerinti vágás és gyakorisági osztályok kiszámítása a később Gnuplot programmal megrajzolt diagramokhoz stb. Például a következő – gyakoriságok összesítéséhez használható – kis héjprogram a bemenet adott (alapértelmezésként első) számoszlopát összeadja, a bc programnak köszönhetően akár több milliárd számjegy pontossággig:

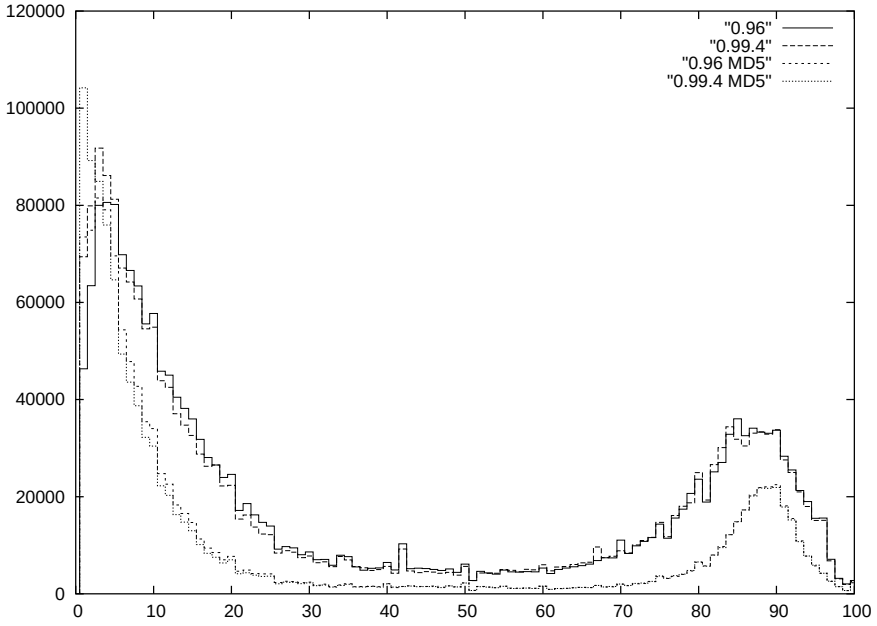
```
#!/bin/sh
awk '{ print "a+=" $'${1-1}' }
END {print "print a"} ' | bc
echo
```

A Unix, Linux és egyéb technológiákról, illetve történeti vonatkozásaikról figyelemre méltó áttekintést nyújt [9].

kódolása, nyelve, helyesírása különböző. Egy előzetes, a .hu tartomány alól letöltött 2,4 millió weboldalon alapuló vizsgálat érdekesebb számai a következőek voltak: Az oldalak mindössze 0,46%-a volt Unicode kódolású. A letöltött oldalak 6,5%-a nem tartalmazott egy szót sem, további 21% nem tartalmazott magyar ékezetes betűt. Átlagban a nem üres oldalak 347 szót tartalmaznak, 85%-uk legalább 20 szóból állt.

A magyar köznyelvi szókincs összegyűjtése céljából oldalszintű szűrésre került sor a Hunspell helyesírás-ellenőrző segítségével. Az oldalszintű szűrés hátterében az a feltételezés állt, hogy az oldalak szókincse konzisztens, és így a megfelelő (esetünkben az ellenőrző által biztosított köznyelvi) szókinccsel a jó oldalak kiválaszthatók [11].

A következő hisztogram a legalább 20 szövegszót tartalmazó weboldalak gyakoriságát ábrázolja a százalékos hibás szóarány függvényében.



Az ábra két csúcsot tartalmaz. A bal oldali csúcs a jó helyesírású oldalaknak felel meg, a jobb oldali pedig a jelentős részében idegen nyelvű oldalaknak. Az MD5-tel jelölt görbék egy olyan oldalszűrés eredményét mutatják, ahol csak a pontra végződő mondatok lettek meghagyva a letöltött oldalak szövegében (illetve a kérdő, és felkiáltó mondatok, ha volt mellettük pontra végződő mondat), majd ezután lett alkalmazva rajtuk egy MD–5 hasítókódot használó duplikátumszűrés. A cél elsősorban az automatikusan hozzácsatolt címek (menüpontok, hírek) okozta minőségromlás kiküszöbölése volt.[3] Ez a módszer azonban általában véve növelte a tényleges összefüggő szövegtartalom súlyát.

A módszer eredményességét jól mutatja a kevésbé jó oldalak számának nagy mértékű csökkenése, a duplikátumok okozta kiugró kis csúcsok eltűnése, illetve a jó oldalak számának növekedése. A fenti adatok tükrében, standard szövegek átlagos hibaaránya alapján kiválasztható a magyar web azon része, ami a mai magyar köznyelvi

[3]Ez legalább két nagy hibáért volt felelős: (1) duplikálás, amikor is a más tartománynéven, de ugyanazon helyről letöltött oldalak a különböző időpont miatt eltérő automatikus címeket tartalmaztak, bár tényleges szövegtartalmuk ugyanaz maradt; (2) szavak hamis gyakorisága, mivel az automatikus címekben szereplő szavak a tényleges gyakoriságuknál jóval nagyobb számban jelentek meg, például egy nagy szolgáltató minden egyes letöltött oldalán olvashatóak voltak.

szókincset tartalmazza. Természetesen nem kapunk hibátlan szókincset így sem, de arányában sokkal több helyes szót tartalmaznak ezek az oldalak. Jellemző, hogy az oldalszintű szűrésnek köszönhetően határozott javulás figyelhető meg a tipikus tévesztések arányában is: például míg a letöltött oldalak a szerviz szót az esetek 27%-ban hibásan (szervíz) tartalmazzák, ez az arány a minőség alapján kiválasztott oldalakon már csak 9%. A módszerekről és az eredményekről részletesen a Szószablya fejlesztés honlapja számol be.

5. Támogatók

A Szószablya fejlesztést az Oktatási Minisztérium és az Informatikai és Hírközlési Minisztérium ITEM pályázata, valamint a MATÁV támogatása tette lehetővé. A Szószablya alapját képező Magyar Ispell és Magyar Myspell fejlesztés kiemelt támogatói a Szószablya fejlesztés mellett az UHU-Linux Kft. és az IMEDIA Kft. A Szószablya fejlesztést megelőzően a TypoTEX könyvkiadó, Mátó Péter és a SuSE Linux Kft. nyújtott még anyagi támogatást a nyitott forráskódú magyar helyesírás-ellenőrző készítéséhez. Sokan hozzájárultak munkájukkal is a fejlesztéshez. Az együttműködők nevét a Hunspell dokumentáció, illetve forráskód sorolja fel. Köszönöm Kornai Andrásnak, hogy a cikk első változatait véleményezte, nagy segítséget adva a végleges változat elkészítéséhez.

Hivatkozások

[1] A Szószablya fejlesztés honlapja: `http://www.szoszablya.hu/`

[2] A Magyar Ispell fejlesztés honlapja: `http://magyarispell.sourceforge.net/`

[3] Magyar OpenOffice.org: `http://office.fsf.hu/`

[4] UHU-Linux: `http://www.uhulinux.hu/`

[5] IMEDIA Kft: `http://www.imedia.hu/`

[6] Larbin webrobot: `http://larbin.sourceforge.net/`

[7] Domolki, B., *Algorithms for the recognition of properties of sequences of symbols.* Журнал Вычислительной Математики и математической Физики 5, 1 (1965), 77–97, fordítás: USSR Computational & Mathematical Physics 5, 1, Pergamon Press, Oxford, 1967, 101–130.

[8] Kovács Győző: *Válogatott kalandozásaim Informatikában*, Gáma-Geo Kft. és Masszi Kiadó, Budapest, 2002, 235–236. o.

[9] Ny. Bezrukov: *Softpanorama: (slightly skeptical) Open Source Software Educational Society*, `http://www.softpanorama.org/`

[10] Németh László: *Magyar Ispell dokumentácó*, `http://magyarispell.sourceforge.net/magyarispell.pdf`

[11] Halácsy P., Kornai A., Németh L., Trón V.: *Cleaning large corpora*, kézirat, benyújtva: DIMACS Workshop on Data Quality, Data Cleaning and Treatment of Noisy Data, 2003

Az OpenOffice.org múltja, jelene és jövője

Noll János

2003.11.08.

Tartalomjegyzék

1. Bevezetés

„Az OpenOffice.org lehet az egyik legfontosabb szabad szoftveres projekt napjainkban. Az emberek ki fogják próbálni, és látják majd, hogy elvégezhetik a napi teendőket anélkül is, hogy a Microsoftnak még több pénzt juttatnának. Az emberek látni fogják – kockázat nélkül –, hogy a szabad szoftver nem csak hogy használható, de jobb megoldást is nyújthat."

A fenti szavak Miguel de Icaza-tól, a GNOME projekt alapítójától származnak, de kijelentésével a valószínűleg mindenki egyetért, aki rendszeresen használja az OpenOffice.org-ot. Napjainkban az otthoni és irodai számítógépek jelentős részén megtalálható egy szövegszerkesztő, táblázatkezelő program. Az üzleti partnerek egymás között egyre inkább elektronikus formában kommunikálnak, dokumentumokat, táblázatokat küldve egymásnak. Előadásoknál, bemutatóknál mind gyakrabban láthatunk számítógéppel készített prezentációkat. Nem túlzás tehát azt állítani, hogy az irodai programok piaca az egyik legnagyobb és legjelentősebb terület. Napjainkban ezt a piacot szinte kizárólagosan egyetlen nagy szoftvergyártó uralja, de ha a szabad szoftveres közösség képes valós alternatívát nyújtani, akkor ez megváltozhat, és több szereplő jelenhet meg a piacon. A több termék közötti választási lehetőség mindenképpen a felhasználók javára válik, még akkor is, ha néhányan a jelenleg is használt termékcsaládnál maradnak, hiszen a verseny várhatóan minden piaci szereplőt nagyobb teljesítményre sarkall.

Az OpenOffice.org sokkal több, mint egyetlen teljes, ingyenes, szabad irodai csomag, mely tartalmaz szövegszerkesztőt, táblázatkezelőt, bemutatókészítőt és rajzolóprogramot. Az OpenOffice.org egészében és egyes technológiai részeiben is alapot nyújt, amelyre többféleképpen lehet építkezni. Lehetséges módosított, továbbfejlesztett, egyedi felületű szövegszerkesztőt készíteni, lehet egy nagyobb alkalmazás egy alkotóelemeként használni, továbbá lehet „csak" egy-egy technológiát felhasználni belőle, mint pl. az UNO[1]. Az OpenOffice.org licence több, mint egy egyszerű szabad szoftveres licenc: mivel a forrásra a GNU GPL+GNU LGPL [1] vagy a Sun SISSL [2] vonatkozik (választhatóan), ezért egyaránt lehet zárt forráskódú és szabad szoftveres fejlesztést végezni a terméken. Bár sokan szkeptikusak a zárt forráskóddal szemben, az OpenOffice.org szerzői nem mennek bele ebbe a vitába, egyaránt lehetőséget adnak mind a zárt, mind a nyílt forráskódú közösségnek, hogy a szoftverre építkezzen. Az SISSL (a zárt felhasználást lehetővé tevő licenc) egy dolgot azonban szem előtt tart: bármi is legyen a zárt továbbfejlesztés eredménye, a fájl formátumok és „kapcsolódási felületek" esetleges módosításait szabadon közre kell adni. Ezzel kerülhető el a termékcsapda: az, hogy egy „átmenetileg jobb" zárt termék a speciális formátumbeli kiegészítésekkel az áttért felhasználók számára elvágja a jövőbeni szabad választás lehetőségét.

2. Történelem

A StarOffice nevű irodai csomagot egy német cég, a Star Division kezdte el fejleszteni, 1986-ban. Az eredeti program csupán egy szövegszerkesztőt tartalmazott és StarWriter néven futott. 1993-ban megszületett a termék Windowsos verziója, melyet egy évvel később az OS/2-es és a Macintoshos verzió követett. A StarOffice nevet 1995-ben vette fel a termék, ekkor már több jelentős komponenst tartalmazott: szövegszerkesztőt (StarWriter), egyszerű rajzprogramot (StarImage), táblázatkezelőt (StarCalc),

[1] Universal Network Objects - az OpenOffice számára kifejlesztett komponens modell, mint pl. a CORBA

grafikonkészítőt (StarChart) és egy vektoros rajzolóprogramot (StarDraw). A StarOffice 3.1-es veziója 1996-ban jelent meg, ez a változat már egy böngészőt és egy HTML szerkesztőt is tartalmazott. A csomag 1997-ben vált teljes irodai csomaggá, amikor már bemutatókészítőt (StarImpress) és adatbázis-kezelőt (StarBase) is tartalmazott.

A StarDivision cég története 1999-ben ért véget, amikor a Sun 73,5 millió dollárért felvásárolta a céget. A StarOffice 5.2-es verzióját a Sun ingyenesen letölthetővé tette, hogy így próbálja meg növelni a termék piaci részesedését. A későbbi változatok már fizetős, kereskedelmi termékekként kerültek a felhasználókhoz, jelenleg a 6.0-s kiadás a legfrissebb, de már bétatesztelés alatt van a következő, 6.1-es verzió is. A Sun 1999-től dolgozott egy Java nyelven írt, webről, előfizetéssel elérhető, távolról használható irodai csomagon és portálon, melyet StarPortal-nak nevezett el. Ez azonban soha nem eredményezett igazi, látható, eladható terméket. A StarPortal-t később felváltotta a Webtop név és koncepció (amely egy „internetes desktopot” képzelt el), majd, miután a Webtopból sem vált önálló termék, a technológia egyes részei bekerültek a Sun ONE Portal Serverbe.

A szabad szoftveres közösség számára a „nagy nap” 2000 október 13-án jött el (ami ironikus módon egyben „péntek, 13” volt), amikor a Sun OpenOffice.org néven szabaddá tette az irodai csomag forráskódját. Több, harmadik fél által készített, licencelt komponenst ki kellett venni, illetve szükség volt több átalakításra is, mielőtt megszülethetett volna az OpenOffice.org kiindulási forrása. Az OpenOffice.org körülbelül másfél év alatt érte el az első nagy mérföldkövet: az 1.0-s verzió 2002 május elsején jelent meg. Az 1.0 óta megjelent az 1.0.1, az 1.0.2 és az 1.0.3-as változat is, amely főleg hibajavításokat tartalmazott. Jelentősebb fejlődés 2003 őszén történt, amikor megjelent az OpenOffice.org 1.1-es kiadása. A következő igazi nagy lépést a 2.0-s változat jelenti majd, amely a tervek szerint a „szokásos” 18 hónapos verzió-ciklus elteltével, 2004 végén vagy 2005 elején várható.

Dan Kegel [3] szabadszoftveres programozó és aktivista szerint míg az OpenOffice 1.0 csupán 70 pontot érdemelne a 100-ból a Microsoft Office-al való kompatibilitásra, addig az 1.1 beta 1 már 90 pontot kaphat. Bár a kompatibilitás még jelenleg sem 100%-os, de a folyamatos javítások eredményeképpen egyre közelebb és közelebb kerül a tökéleteshez. Ugyanakkor közismert, hogy a Microsoft Office különböző verziói között sem 100%-os a kompatibilitás.

Amikor a StarOffice 5.2 forráskódját átalakították és átrendezték, hogy abból OpenOffice.org születhessen, fontos feladatott jelentett az egyes programrészek újra felhasználható komponenssé való átalakítása. A komponensekhez kifejlesztettek egy saját, UNO (Universal Network Objects) nevű, teljesen új modellt. Az UNO lehetővé teszi, hogy a komponensek különböző folyamatok és gépek között összekapcsolódhassanak és együttműködhessenek más komponensekkel. Fontos szempont volt az UNO modell kifejlesztésnél a sebesség és a teljesítmény, ez is szerepet játszhatott abban, hogy nem egy meglévő, létező modellt használtak (mint pl. a CORBA).

Az OpenOffice.org XML alapú [4], nyílt, teljesen dokumentált fájlformátumot használ. Az XML alapú formátum számos előnyt hordoz magában, hiszen lehetővé teszi a dokumentumok konvertálását, módosítását külső scriptekkel, programokkal, egyszerűbbé teszi az integrációt más rendszerekkel. Csupán apró példa, hogy egy néhány soros Perl nyelven írt scripttel egyszerűen kimazsolázhatjuk egy könyvtárban található OpenOffice.org dokumentumok címeit és szerzőit, anélkül, hogy a szükségünk lenne az OpenOffice.org programcsomagra. Jelentős eredménynek számít, hogy az OASIS [5] nonprofit, szabványokkal foglalkozó csoport egyik technikai bizottsága 2002 novemberében az OpenOffice XML formátumát választotta kiindulási alapnak egy nyílt dokumentum-formátum kialakításához.

3. Magyarítás

Amennyiben azt szeretnénk, hogy az OpenOffice.org minél több magyar felhasználó számára elérhető legyen, valós alternatívát nyújtson, fontos, hogy magyar nyelvű felületet, segítséget és eszközöket (pl. helyesírás-ellenőrző) tartalmazzon.

Az OpenOffice.org egy óriási programcsomag, mely sok, gazdag funkcionalitású komponenst tartalmaz, nem csoda, hogy a magyarítása is nagy erőfeszítést igényel. A lefordítandó tartalom három részre bomlik: grafikus felület kifejezéseire, a súgó oldalaira, és az egyéb kiegészítőkre. A leglátványosabb és legszembetűnőbb rész a program kezelői felülete, amely többek között a párbeszédablakokat és a menüket jelenti. Ez a rész körülbelül 22.000 kifejezést tartalmaz. A súgó ennél még nagyobb falat: 400.000 szóból áll a teljes, helyzetérzékeny segítség. Magyarra fordítandó emellett a programhoz mellékelt néhány sablon (levél, fax, stb.) és apróbb fájl is. Ugyan nem fordítási feladat, de mindenképpen a magyar változat használhatóságát növeli, ha van a csomagban magyar nyelvű helyesírás-ellenőrző, szinoníma-szótár és elválasztás is.

Az OpenOffice.org alapú programcsomag első, a súgó kivételével teljes magyarítása 2001 végére készült el, amikor a MultiRáció Kft. kiadta a MagyarOffice 1.0 nevű, kereskedelmi termékét. Sajnos ez a magyarítás nem került vissza a szabad szoftveres közösséghez, ezért a közösség megszervezte a saját, szabad magyarítás elkészítését. Az, akkor még leendő, FSF.hu Alapítvány szervezésében, 2002 februárjában, egy 3 napos maratoni fordító-hétvégén készült el az OpenOffice.org 1.0 felhasználói felületének fordítása. A helyszínen jelenlévő 20-30 ember és a távolról segítő további 120 ember segítségével elkészült az (akkor még lektorálatlan) fordítás. A fordítást egy speciálisan az OpenOffice.org fordítására kifejlesztett, egyszerű, ámde hatékony webes felület segítette. Saját fordítóeszköz kifejlesztésére azért volt szükség, mert az akkor létező fordítást segítő szabad eszközök egyike sem támogatta ennyi ember ilyen szintű párhuzamos fordítását, továbbá az eszközök telepítése és használata is sokkal bonyolultabb lett volna a webes felület használatánál. A sok önkéntes fordító előnyt jelentett a fordítási teljesítmény szempontjából, ugyanakkor problémát jelentett, hogy eltérő fordítások születhettek egy-egy fogalomra. Ezt jelentősen csökkentette ugyan a minden résztvevő számára weben elérhető terminológia oldal, azonban még emellett is akadtak hibás fordítások, félreértelmezések. A 3 napos fordító hétvége végére az OpenOffice.org-ban található körülbelül 22.000 kifejezés mindegyike le lett fordítva, de lektorálva, átnézve csupán a kifejezések 40%-a készült el. Ez persze nem jelenti azt, hogy a magyar nyelvű programcsomag ilyen állapotban ne lett volna használható, csupán arról van szó, hogy időnként kisebb-nagyobb hibákkal, félrefordításokkal találkozhatott a felhasználó. Mivel az aprólékos lektorálás, javítás időigényes és unalmas munkának számít, ezért nem meglepő, ha a lektoráltság szintje a fordító hétvége után csak lassan emelkedett. A lektorálás befejezését 2003 őszén a Gamax Kft. végezte el, megbízás alapján. Jelenleg az OpenOffice.org 1.1-es kiadása teljesen le van fordítva és lektorálva is van és ami nagyon fontos, ez a fordítás szabadon felhasználható.

Bár az OpenOffice.org felületének fordítása egyértelműen a közösség sikertörténetének mondható, a súgó fordítása azonban nagyobb falatnak bizonyult. Jelenleg a súgó mintegy fele van lefordítva, amely mintegy 200.000 szót jelent, de sajnos ebben a fordításban még bőven találhatóak kisebb-nagyobb hibák. Nehézséget jelentett, hogy míg a felület kifejezései önálló kifejezések voltak, amelyeket külön-külön lehetett fordítani, addig a súgó összefüggő oldalakból illetve oldalrészekből álló, speciálisan formázott és összekapcsolt szöveg. Emellett problémát jelentett az is, hogy a program súgója eredetileg német nyelven készült, amit később angolra fordítottak, emiatt sok helyen az eredeti, angol súgó sem fedte a program adott funkciójának pontos működését. A for-

dítóknak az is nehezített a munkáját, hogy először meg kellett találniuk a fordítandó súgóoldal-részt, az eredeti környezetben, meg kellett érteniük, hogy pontosan mit ír le, és meg kellett érteniük a leírást. A webes, súgóoldalak fordítását segítő felület a korábbi rendszer koncepciójának továbbfejlesztése volt, amely azonban még messze nem tökéletes, további átgondolásra és javításra szorul. Mindezek ellenére reményeim szerint a Writer modul teljesen magyarított súgója sokak számára segít kihozni a maximumot az OpenOffice.org-ból.

Sokáig az egyetlen, széles körben használható helyesírás-ellenőrző a Morphologic „Helyes-e?" nevű, kereskedelmi terméke volt. Bár a szabad licencű, alternatívát nyújtó Magyar Ispell [6] helyesírás-ellenőrző fejlesztése már 1998-ban elkezdődött, a program igazi lendületet csak 2002-ben kapott. A Magyar Ispell fő fejlesztője Németh László, de a szóadatbázis feltöltésében, javításában mások is közreműködtek, jelenleg a programt több mint 10.000 ragozási szabályt és több mint 100.000 szóból álló szókincset tartalmaz. A Magyar Ispell 2003-ban eljutott oda, hogy nem csak felveszi a versenyt a kereskedelmi „Helyes-e?" termékkel, de azt egyes területeken le is körözi [2]. Az OpenOffice.org helyesírás-ellenőrzője, a MySpell képes a MagyarIspell adatbázisának kezelésére, így minden felhasználó élvezheti a professzionális szintű helyesírás-ellenőrző előnyeit.

Az OpenOffice.org a magyar elválasztást a TEX-hez korábban készített elválasztási adatbázisból örökölte. A Nagy Bence által karbantartott HuHyphn [7] elválasztási adatbázis egy több, mint 330.000 szavas szótárból indult ki, és több mint 1.000.000 elemzett elválasztási helyet tartalmaz.

4. Termékvonalak

Az OpenOffice.org termékkel a legszorosabb kapcsolatban a Sun StarOffice csomagja áll. Bár eredetileg a StarOffice forrásából született az OpenOffice.org, azóta fordult a helyzet. A StarOffice 6.1 megjelenése előtt a korábbi verziók az OpenOffice.org egyes, stabil fázisaiból indultak ki. A két, párhuzamosan karbantartott vonal többletmunkát és sok gondot jelentett a fejlesztőknek, ezért a StarOffice 6.1 illetve OpenOffice.org 1.1-es kiadása óta azonos forráskódból készül mindkét programcsomag. Különbséget a hozzáadott adatok, programok és szolgáltatások jelentenek. A StarOffice-hoz kereskedelmi helyesírás-ellenőrző, professzionális szintű kép- (clipart) és sablongyűjtemény jár, emellett a termék minőségbiztosítását a Sun végzi, míg az OpenOffice minőségbiztosítását a közösség tagjai végzik. Az Adabas D adatbázis-kezelő, a licencelt betűtípusok, a teljes dokumentáció és a hivatalos támogatás olyan hozzáadott értékek, amelyek révén a Sun jogosan számíthat sikerre a StarOffice csomaggal céges körökben.

Bár szomorú hír lehet a magyar felhasználók számára, hogy egyelőre a Sun nem tervezi a StarOffice hivatalos magyar változatának megjelentetését, azonban nincs ok aggodalomra, hiszen létezik kereskedelmi változat itthon is.

A Multiráció Kft. által fejlesztett MagyarOffice az OpenOffice.org forráskódján alapul, és szintén a hozzáadott értékektől reméli, hogy sokan döntenek majd a terméke mellett. A MagyarOffice tartalmazza, többek között, a Morphologic-tól licencelt magyar nyelvű helyesírás-ellenőrzőjét, elválaszó programját és szinonímaszótárát. A csomag ezen felül tartalmaz magyar témájú kép- (clipart) és sablongyűjteményt, optimalizáló és térképes diagram modulokat, felhasználói kézikönyvet és terméktámogatást. Az másik magyar nyelvű, OpenOffice.org-on alapuló, boltban is kapható termék az EuroOffice, amelyet az Információ Digitálmédia Kft. jelentetett meg. Ez a termék leginkább

[2] Lásd Németh László cikkét.

a terméktámogatás révén kíván többletet nyújtani a magyar, FSF.hu OpenOffice.org-hoz képest.

A szabad magyarított OpenOffice.org-ot az FSF.hu Alapítvány gondozza, a legfrissebb Linuxos és Windowsos változatok az FSF.hu OpenOffice.org oldaláról [8] oldalról tölthetőek le. Természetesen a programok mellett lehetőség van a fordítás forrásának és a kiegészítéseknek a letöltésére is, továbbá az esetleges, magyarítással kapcsolatos hibákat szintén a weboldalon lehet jelenteni.

Bár nem jelenik meg önálló termékként, de mindenképpen említésre méltó, hogy mind az UHU-Linuxban, mind a magyar SuSE Linuxban megtalálható a magyar OpenOffice.org, amely tartalmazza a szabad magyar helyesírás-ellenőrzőt és elválasztást is.

A Ximian Desktop-nak része egy Ximian Office program, amely az OpenOffice.org kisebb-nagyobb fejlesztéseit tartalmazó változata. A nagyobb fejlesztések közé tartoznak a szebb eszköztár-ikonok (és az ikonrendszer kódjának továbbfejlesztése), a CUPS nyomtatórendszerrel való integráció illetve a Samba és NFS protokollok belső kezelése.

Az OpenOffice.org jelenleg Windows, Linux, Linux PowerPC, Solaris, Mac OS X, FreeBSD, IRIX, Linux/S390/HP-Unix rendszereken futtatható, azaz valóban többplatformos program, a használatával semmiképpen sem leszünk egyetlen operációs rendszerhez láncolva, sőt, bármikor válthatunk rendszert, a dokumentumainkat és a tapasztalatunkat ugyanúgy használni tudjuk majd.

5. A jövő

Az első OpenOffice.org konferenciát [9] 2003 márciusában rendezték meg, ahol több érdekes előadás foglalkozott az OpenOffice.org jövőjével és a programcsomagra épülő fejlesztésekkel. A NUSS projekt [10] azt kutatta, hogy miként lehet az OpenOffice.org segítségével interaktív bemutatókat, oktatást tartani egy-egy csoport számára. A rendszer az OpenOffice.org forráskódjának módosítása nélkül, kizárólag a hozzáfejlesztett külső modulokkal és Java nyelven írt távoli vezérléssel készült el. Egy másik érdekes felhasználása az irodai programcsomagnak a StarOffice 4kids projekt [11], amelyben egy Java nyelven írt grafikus felületbe ágyazták be a programot, úgy, hogy minden vezérlőelemet a Java program nyújt. A kísérleti projekt célja, hogy gyerekek által használható, csökkentett tudású és megjelenésű programokat lehessen készíteni a rendszerrel. A projekt nem csak egy-egy alkalmazást illetve játékot hozott létre, hanem egy általánosan felhasználható keretrendszert, amely segítségével bárki készíthet hasonló programokat. Fontos ismételten megemlíteni, hogy az OpenOffice.org forrásának módosítására itt sem volt szükség.

A 2003-as év első felében jelent meg az OpenOffice.org SDK [12] (programfejlesztő csomag), amely lehetővé teszi az OpenOffice.org bővítését új komponensekkel, anélkül, hogy szükség lenne a programcsomag forrásának módosítására. Az SDK tartalmaz egy több mint 1000 oldalas, részletes dokumentációt (elektronikus formában), Java, C++ és OpenOffice Basic példákat, tehát mindent, ami egy a fejlesztéshez szükséges.

Természetesen a fejlesztések jelentős része továbbra is az OpenOffice.org forráskódján történik, amely fejlesztést egy új technika tesz könnyebbé és problémamentesebbé. Korábban egyetlen fő fejlesztői ág létezett, ide kerültek az új funkcionalitások, nagyobb javítások vagy változtatások. Ez azért jelentett problémát, mert egy-egy rossz módosítás destabilizálhatta a teljes fejlesztői ágat, amelyből addig nem lehetett használható terméket előállítani, amíg a módosítás által okozott hibákat, mellékhatásokat

fel nem számolták a fejlesztők. Ugyanezen okból nehezen fogadtak be a fő vonalba nagyobb külső kiegészítéseket, fejlesztéseket. A problémára a „Child Workspaces” („Gyerek munkaterületek”) technológia hozott megoldást. Ezzel a módszerrel a fő ágról leválasztható egy-egy izolált mellékág, ahol bármilyen fejlesztés, javítás történhet, anélkül, hogy hatása lenne a fő vonalra. Amikor egy ilyen mellékág stabil állapotba kerül, a fejlesztés eredményei kemény tesztelés után visszakerülnek a fő ágba. Ez által bátrabban és szélesebb körben lehet fejleszteni az OpenOffice.org termék tudását, anélkül, hogy bármelyik különálló fejlesztés veszélyt jelentene a teljes forrás stabilitására.

Felmerülhet a kérdés, hogy mit tegyen az, aki nem ért a programozáshoz, de szeretné, ha az OpenOffice.org egy fejlődő, egyre jobb és nagyobb tudású programcsomag lenne?

Több lehetőség is nyílik arra, hogy segítsük a program fejlődését: az új OpenOffice.org programok tartalmaznak egy olyan komponenst, amely elszállás esetén opcionálisan jelenti a hibát (és annak okát) a fejlesztőknek. Emellett nagyon fontos, hogy amennyiben programhibát találunk, ha tudásunk és időnk megengedi, jelentsük azt az IssueZilla nevű hibakövető rendszerben [13]. Aki még több idővel és tudással rendelkezik, az segíthet a hibák kategorizálásában, ellenőrzésében és átfutásában, Dan Kegel OpenOffice.org oldala [14] részletes leírást tartalmaz erre vonatkozóan.

Hivatkozások

[1] GNU General Public License, GNU Lesser General Public License – szabad szoftveres licencek, `http://www.gnu.org/licenses/licenses.html`

[2] Sun Industry Standards Source License - zárt forráskódú felhasználást lehetővé tévő szoftver licenc, `http://www.openoffice.org/license.html`

[3] `http://www.kegel.com/`

[4] `http://xml.openoffice.org/`

[5] `http://www.oasis-open.org/`

[6] `http://magyarispell.sourceforge.net/`

[7] `http://gimb.freeweb.hu/`

[8] `http://office.fsf.hu/`

[9] `http://marketing.openoffice.org/conference/`

[10] `http://www.informatik.uni-stuttgart.de/ipvs/vs/en/projects/NUSS/`

[11] `http://www.kippdata.de/produkte/so4k/so4k_en.html`

[12] `http://api.openoffice.org/SDK/`

[13] `http://www.openoffice.org/issues/enter_bug.cgi`

[14] `http://www.kegel.com/openoffice/`

Permanens forradalom

Tomka Gergely

2003.11.08.

Tartalomjegyzék

1. Milyen egy forradalom?

Forradalmat élünk meg. Lassú forradalmat, és ezt belülről nehéz átlátni. Ha reggel kinézünk az ablakon és látjuk, hogy villamosokból és napernyőkből emelt barikádokat ostromol a karhatalom, miközben plüsspingvineket lóbáló fegyveresek harcolnak a jövőért, tudjuk: forradalom van, és mi is csatlakozunk elvtársainkhoz. A most is körülöttünk zajló (csordogáló?) forradalmat nehezebb észrevenni, és ezért talán kevesebben is csatlakoznak hozzá. Előadásomban szeretném megmutatni, hogy azok a viszonylag apró dolgok melyeket, hétköznapi munkánk során teszünk – egy GNU/Linux telepítés, egy iroda átállítása OpenOffice.Orgra, a világ legjobb csomagszűrőjének megírása –, egy nagyobb és csodálatos folyamat része. Szeretném, ha az itt jelenlévők a következő samba-konfigurálás alkalmával éreznék, hogy Gilgames hőseinek nyomában járnak, hogy Voltaire barátságosan mosolyog a válluk fölött, és hogy tízezer év történelme tekint le rájuk.

2. Információs forradalmak a múltban

Az információ, mint minden elvont fogalom, önmagában létezik jó régóta, és átvitele, tárolása, feldolgozása is számtalan változást ért meg. A lassú és állhatatos fejlődés néha felgyorsul, ha valami új felfedezés történik.

2.1. Írás

Egyszer volt, hol nem volt, a szóbeszéd. Évtízezredekig minden tudás szájról szájra öröklődött, és ez pontosan elég is volt. Jó emlékezni a Nagypapa hőstettére, amikor a sziklával megölte a barlangi oroszlánt, de semmilyen komoly érdekünk nem fűződik ahhoz, hogy *pontosan* emlékezzünk rá. A Nagypapának is jólesik hallani, amint az oroszlán egyre nagyobb lesz, minél többször mesélik el a történetet. Ráadásul a szóbeszéd meglehetősen pontos is lehet. Amikor a mormonok családfákat kutattak[1] Óceániában, évszázadokra visszamenőleg igen részletesen megismerhették a családfákat, házasságtöréseket és egyéb érdekességeket.

Új adatrögzítési fogásokra csak akkor lett szükség, amikor megszülettek az első királyságok. A törzsi vadászoknál mindenki jelen volt, amikor ettek, de a királynak adót kell szednie, kereskednie kell, és ez nem megy pontos nyilvántartás nélkül. Nem véletlen, hogy az első államok – Sumer, India, Kína – megszületésének állandó kísérőjelensége a kis jelekkel telerótt agyagtáblák, teknőspáncélok, falevelek megjelenése. Eleinte ezek igen egyszerűek voltak – egy ökör rajza és mellette 27 vonalka azt jelentette, hogy 27 ökörrel indult neki a pásztorlegény, és ez akkor bőven elegendő volt. A siker így is azonnali és átütő volt. Az erősebb és megbízhatóbb alá-fölé rendeltségi viszonyoknak, a pontos elszámolásnak (is) köszönhetően olyan gyorsan fejlődtek ki az első birodalmak a jelentéktelen földművesfalvakból, hogy a történészek azóta is sokat morfondíroznak a történteken, és Däniken barátunk egyre-másra gyárthatja az ufó-összeesküvéselméleteit. És ha belegondolunk, a ma leírt szövegeknek túlnyomó többsége szintén ebbe a kategóriába esik. Ügyvitel, elszámolás, könyvelés, ezek mindmind a sumerok óta nagyjából változatlan dolgok.

Két fő csapásirányon fejlődött tovább az írás. A királyok szerették volna, ha hőstetteiket tovább őrzi meg az emlékezet, mint Nagypapáét az oroszlánnal, és az sem árt, ha

[1] Egyik elképzelésük az, hogy minden valaha élt embernek megadják a végtisztességet – ennek eredménye a magyar családfakutatás ősforrása is.

az általuk hozott döntéseknek és ítéleteknek is marad valami halvány nyoma. Mást is szerettek volna – szigorúbban ellenőrizni alattvalóikat, nemcsak ökreiket és gabonájukat kezelni, de amennyire lehet, gondolataikat is. Ezért megszülettek az első egyházak, papokkal és szent iratokkal.

Törvények, teremtéstörténetek, história. Dúskálnak az olyan elvont fogalmakban, melyeket igen nehéz képregényként lerajzolni. A kereskedelmi feljegyzésekben azt, hogy „szerválbőr" nem nehéz odakarcolni – csak egy szervált és egy bőrt kell megalkotni. De hogyan rajzoljuk le azt, hogy „tulajdonába kerül"? Vagy hogy

> Láttad azt, aki egy fiút nemzett?
> Láttam őt.
> Mi a sora?
> Házának fala mellett könnyei hullanak.[2]

Az ilyen problémák leküzdésére született meg az, amit ma szótagírásnak nevezünk. A gondolat praktikus, képrejtvényeket kell írni képregények helyett. Ha van jelünk a „szervál", a „relé" és az „ember" szóra, akkor a 3 jelet egymás mellé írva hihetjük azt, hogy az mostantól azt jelenti, hogy „szerelem". Hogy miért nem „váléber"? Ez jó kérdés, és hogy meg tudjuk válaszolni, sok és néha igen-igen bonyolult szabályra van szükségünk, ezt 12 éven át tanítanunk kell gyermekeinknek, majd érettségizniük is kell belőle. A szótag- és betűírásoknál a helyesírás egy ma is virágzó és dinamikusan fejlődő iparág.

Kínában kicsit más utat választottak. Ott a nyelv is más, ezért járhatónak tetszett az a megoldás, hogy minden szónak van egy jele. Ez is magában hordozza büntetését – sok jelünk lesz, a kínai nyelvben például 21 ezer. Ráadásul ez is kevés, ezért egy jelnek sok értelme is lehet. Változatos mondatokat – „A mester szeret oroszlánnyálat nyalogatni" – lehet leírni egy jel ismételgetésével. De ez az ő problémájuk, elég nekünk a 300+ oldalas helyesírási szótár.

Akárhogy is, a kezdeti könyvelők, íródeákok, jogászok lehetővé tették az elvont fogalmak megörökítését, és szabadidejükben létrehozták azt, amit ma szépirodalomnak nevezünk. Úgyhogy ha legközelebb nem eredménykimutatásokat olvasgatunk este a kandalló előtt, hanem Rejtő Jenőt vagy Tolsztojt, gondoljunk rájuk szeretettel.

2.2. Könyvnyomtatás

Bár az írás alapvetően megváltoztatta a világ jellegét és fejlődésének irányát, évezredekig újabb áttörés nem történt. Az egyszeri ember nem igazán látott írást életében, ő maga sem tudott írni. A papok és könyvelők foglalkoztak az írással, fejlesztgették, csiszolgatták. Lerövidítették a képrejtvények elemeit, és végül áttértek a szótagokról a betűk írására, megváltoztatták a betűk formáját, attól függően, hogy agyagba nyomkodták, papírra írták vagy kőbe vésték a jeleket. És gyűjtögették a gondolatokat, tudományos munkák, evangéliumok, verseskötetek születtek és gyarapodtak. Természetesen a legtöbb gondolatot továbbra is a központi hatalom gyűjtötte be, értelmezte és használta föl, de ahogy egyre több ember tanult meg írni-olvasni, lassan megszülettek az eretnek gondolatok is. Egy levél messzebbre és gyorsabban eljut, mint maga az ember, aki írta, ráadásul a levelet többfelé is el lehet küldeni, míg a magányos térítő csak egyfelé mehet. Semmiképpen sem gyors folyamat ez a levelezgetés. Sokszor és kézzel kell lemásolni a szövegeket, és ezeket titokban csempészni szerteszét a világban. Nem csoda,

[2] Gilgames, Komoróczy Géza fordítása

hogy akkoriban egy eretnek gondolatnak több száz évébe került széles körben elismert gondolattá válnia.

Így mikor egy újabb eretnek gondolat született meg, és terjedt az írástudók körében, keresni kellett egy olyan eljárást, amellyel a fárasztó körmölés helyett egyszerűen és gyorsan sokszorosíthatjuk könyveinket. A már régóta ismert fadúcok elve egymástól elkülönített fémbetűkkel alkalmasnak tűnt arra, hogy ugyanazt a szöveget lássuk sokszor. Csak a könyv minden egyes oldalát ki kellett rakni kicsi fémbetűkből[3], a nyomóformát bekenni festékkel, majd papírt szorítani rá. Kezdetben ez nem ment gyorsan, az első könyvek pár száz példányban való kinyomtatása több év is lehetett, de a szellem kiszabadult a palackból. A sok könyv elolvasásához sok embert meg is kellett tanítani olvasni, és minél több ember tudott olvasni, annál több könyv kellett. Az eleinte a reformáció, majd az ellenreformáció iskolái behálóztak az országokat. A reformáció, az ellenreformáció, a felvilágosodás, a munkásmozgalom, a kommunizmus, a dianetika és más eszmék futótűzként szaladtak végig Európán, évszázadokig tartó vallási és egyéb háborúkba taszítva népeket.

Volt persze jó hatása is a könyvnyomtatásnak. A sokfelé eljutó könyvek egyre több embert vontak be a fejlődésbe, egyre többfelé jutottak el hasznos információk, a tudomány legújabb eredményei. Eleinte a mezőgazdaság, majd az ipar is nekilendült, amikor az addig jórészt szobatudósok által kiötölt új elméletek gyakorlatias, pénzhajhász emberek kezébe kerültek. Papen kísérleti gőzgépe pár évtizeddel később brit bányákból szivattyúzta a vizet, szimbólumává válva a ma is tartó ipari forradalomnak. Lassan szükségessé vált, hogy *mindenki* tudjon írni és olvasni, kialakultak az iskolarendszerek.

A könyvnyomtatással hamarosan a Hatalom is megbékélt. Sok pénze volt, sokat tudott kinyomtatni kincstári gondolataiból, az eretnekeknek kevesebb, ezért ők kevesebbet nyomtattak. Így a kezdeti felfordulás után ismét kialakult az egyensúly. A Hatalom ellenőrizte a nyomdákat – ezt nem nehéz, egy nyomda sok géppel dolgozik, zajos, nagy az energiaigénye, nem egyszerű elrejteni –, a mindenkori ellenzék pedig igyekezett mégis kinyomtatni eretnek gondolatait, és ezeket eljuttatni célközönségének. Az ellenzéki lét fontos része lett a „szamizdat”, elég csak a Joszif Visszárionovics Dzsugasvili szerkesztésével működő *Iszkra* újságra, vagy a *Beszélő* folyóirat Demszky Gábor csodálatos Volkswagenjében csempészett nyomdagépeire gondolni.

2.3. Rádió, tv

A Hatalom képviselői megbékéltek a könyvekkel, de ezeknek volt egy súlyos hibájuk. Befogadásukhoz *aktív tevékenység* kell. Oda kell menni a polchoz, leemelni a könyvet, értelmezni, emlékezni, ne adj isten, gondolkodni. Ez nem jó. Sokan vannak, akik úgynevezett *funkcionális analfabéták*, hozzájuk hogy jut el a Hatalom aktuális üzenete? Nekik született meg előbb a rádió, majd az igények növekedésével a televízió. Kezdetben a kultúra fontos részei voltak, gondoljunk például Orson Welles rádiójátékára, vagy a korosabbak által ismert *Egymillió lépés Magyarországon* televíziós sorozatra. De az elektronikus berendezések, adók, átjátszóállomások nagyon sok pénzbe kerülnek, és minél több emberhez akarunk beszélni, annál több pénz kell, annál nehezebb rejtetten, titokban, a Hatalom híre s tudta nélkül szolgáltatni. Nagyon kevés kalózrádiót ismerünk, és azok is legtöbbször valamilyen külső hatalom támogatásával működtek, mint az Amerika Hangja vagy a hatvanas évek Angliájának Radio Caroline-ja, kalóz tv-adóról meg nem is igen hallottam.

[3] Ez fárasztó, de jóval gyorsabb megoldás, mint minden egyes oldalt kifaragni.

3. Internetforradalom

Mint minden ezt megelőző információs forradalmat, ezt is egy technikai „ugrás”, a számítástechnika fejlődése indította el. És mint ahogy az írás is eredeti céljától (a kereskedelmi nyilvántartásoktól) elszakadva járta be útját, az internet is hamar megszűnt mint atombiztos kommunikációs csatorna.

3.1. Röviden az indulásról

1962-ben gondolta úgy az amerikai légierő, hogy szükségük van egy kommunikációs rendszerre atomtámadás esetén. Ilyenkor az ellenfél a központokat bombázza le először, ezért a kommunikáció is megszakadhat. Az első követelmény ezért az volt, hogy ne legyen központja a hálózatnak. A második követelmény a viszonylagos egyszerűség volt, ezt talán úgy lehetne szemléltetni, hogy míg az IBM SNA protokollnak megvan mind a 7 ISO/OSI rétege, komplex alrétegekkel, addig a széles körben használt TCP/IP protokollcsalád még nagy jóindulattal is csak 3-4 réteget valósít meg, minden egyetemi oktató nagy bánatára.

3.2. Az eredeti katonai célok emberivé válása

Van tehát egy egyszerű, egyre inkább terjedő hálózatunk, nyitott és aránylag egyszerű protokollal, atomháború esetére, de az atomháború elmaradt (mindannyiunk nagy szerencséjére). Valamit kellett kezdeni ezzel a sok felszereléssel, és az elméleti kutatások mellet hamarosan megjelent az első levelezőprogram (1972), majd 1979-ben az USENET is megszületett, a hálózati sakkpartik szervezését megkönnyítendő. Míg az egyetemek vezetői kutatási projekttervek vastag paksamétáival utaztak új kábelekre, szerverekre és routerekre pénzt kalapozni, a felépült hálózatokon viccek, hírek, ötletek kavalkádja utazott mindenfelé.

3.3. Mit változtat meg az internet?

Az írás a gondolatok örök életét biztosította. A nyomtatás azt, hogy ezeket a gondolatokat elterjeszthessék. Az 1985 táján már robbanásszerűen terjedő internet pedig azt, hogy bárki gondolatait el lehessen terjeszteni. Nincsen szükség stencilezőgépre, tv-toronyra, HíF engedélyre ahhoz, hogy egy általunk kitalált vicc elterjedjen – elég elküldeni a `HIX MÓKA` levelezőlistára, és ha jó a humorérzékünk, máris sikerünk van. Ráadásul ne felejtsük, hogy a hálózat központ nélküli, ebből a filozófiából következően ellenőrizhetetlen (a gyakorlatban csak „nehezen ellenőrizhető”). Ráadásul, bár a hálózat kiépítése költséges, mégis viszonylag egyszerű és olcsó csatlakozni rá. Ez a pár előny több új elemet is visz az emberek közötti kommunikációba, és mindegyik önmagában is kisebb forradalom.

Hírek terjedése

A már sokszor emlegetett Hatalom nem tudja ellenőrizni a terjedő híreket. Megszűnt a nagy állami vagy üzleti alapú hírügynökségek abszolút hatalma. Az iraki háború során kicsi, nem hirdetett, magánszemélyek által fenntartott fórumokon keresztül jelentek meg a friss hírek, ilyen helyeken cáfolták a CNN vagy a hadsergek propagandistáinak hazugságait, és hozzáértők rögtön elemezték is a híreket, a „hagyományos” sajtó képviselőinek már csak a helyesírást kellett rendbe tenniük. Szemben a pletykával, ahol a

hírek eredeti forrásától sok-sok lépés választ el minket, az interneten azonnal a hírek forrásával kerülhetünk kapcsolatba, például a szír–iraki határ mellett élő arab ifjúval.

Ellenőrizhetetlenség

Az internet alapjainak kifejlesztésekor még nem volt szempont a szereplők azonosítása, hiszen a terveken dolgozó pár tudós mind ismerte egymást. Az, hogy már az első megvalósítás ilyen sikeres lett, bizonyos tervezési hibák örök életével jár. A legtöbbet átkozott ezek közül az, hogy nincsen megbízható megoldás az azonosításra. Ha akarjuk, azonosíthatjuk magunkat (PGP, hitelesítésszolgáltatók), de ha ezt nagyon nem akarjuk, akkor senki sem talál meg minket. Roppant bosszantó, amikor ezt arra használják, hogy a legújabb pénisznövelő por csodájáról tájékoztassanak minket, de sok elnyomott országban élő, szabad lelkű embernek pontosan ez a bug/feature teszi lehetővé a szabad világhoz való csatlakozást.

Ismeretlen ismerősök

A tudás végtelen kavalkádja található meg az interneten. Egy újság évekig tartó cikksorozatot alapozott arra, hogy a „fogpiszkáló” szóra keresve megjelenő oldalak között kalandoztak. Mégis vannak kisebb-nagyobb csomópontok – levelezési listák, fórumok, irc csatornák, keresők –, ahol emberek összejöhetnek, elindulhatnak megismerni az internetet. Az általános ismerkedésen túl minden érdeklődési kör is összegyűlhet valahol – legyenek azok az Észak-Finnországi Fogpiszkáló-faragók vagy a kínai űrhajózás rajongói. Egy ember, aki a fehér éjszakákat fogpiszkáló-faragással tölti, talán soha életében nem találkozik hasonló bolonddal[4], de az interneten hamar egymásra találnak, és ez még a fogpiszkáló-faragás esetén is csodálatos dolog. Hát még ha hasonlóan hóbortos programozók találkoznak – de erről később.

Szabadság

Az interneten közel tökéletes szabadság uralkodik. Aki amit akar, általában közzéteheti. Ennek vannak sötét oldalai, nácizmus, gyermekpornó, és a Hatalom ezeket rendszeresen föl is hozza érvként az internet korlátozása mellett. De remélem, nem vagyok egyedül azzal az elképzelésemmel, hogy ennek a totális szabadságnak több a világos vagy szürke oldala, mint a sötét.

4. Mit ad az internetforradalom az emberiségnek?

Nem forradalom a forradalom forradalmi újdonság nélkül. Kell valami, ami eddig nem volt, és csakis ez a forradalom tette lehetővé. Természetesen az informatikai forradalom is hozott valamit az emberiségnek, ami talán jobbá teszi a világot, szépen lassan. Ez a szélesebb értelemben vett munkaszervezés egy új módszere.

4.1. Katedrálismodell

Kezdetben vala egy mező a város szélén. A Püspök elhatározta, hogy oda Katedrális épül. Megfizetett egy Építőmestert, az utasította az ácsok Mesterét, a kőművesek

[4] Itt kérünk elnézést minden fogpiszkáló-faragótól – a hobbi tényleg létezik, toothpick carving szavakra kell keresni

Mesterét, azok utasították a favágókat, kőfaragókat, és szépen, lassan, pár száz év alatt fölépült a Katedrális. Szoros, központi irányítás, szigorú tervezés, parancsláncolat, fegyelem. Ezzel a megoldással nagyszerű dolgokat hoztak létre, de nem illeszkedett minden problémára a nagybetűs életben. Közismertek a központi irányítás katasztrófái a gazdaságban, de különösen kínossá a szoftverfejlesztés terén vált a helyzet. Nagy projekthez sok fejlesztő kell, a sok fejlesztőhöz sok vezető, sok parancsnoki szint, és a végeredmény az az, hogy a vezetők semmit sem tudnak arról, hogy mit csinálnak azok, akik ténylegesen dolgoznak. Ezért nagyon sok perverz dolgot kényszerítenek a fejlesztőkre (például titokban kell tartaniuk, mit csinálnak), nem a hatékonyság vagy hibamentesség a fő szempont, és ezer más probléma. Jelentős részüket kiválóan szemlélteti a Nagynevű, De Ezen A Konferencián Soha Meg Nem Jelenő Redmondi Fejlesztő Cég legújabb, XP nevű állatorvosi lova.

4.2. Bazármodell

Ezzel szemben áll egy finn/svéd ifjonc, aki saját szórakoztatására megalkot valamit, amiből egyszer majd egy több területen piacvezető operációs rendszer lesz. Amikor kiáll vele a nyilvánosság elé, még nem hasonlít egy piacvezető operációs rendszerre. Nem is igen működik, nem lehet vele semmit sem csinálni, de kikerül a széles nyilvánosság elé, és ekkor valami, ami addig még nem nagyon történt, megindul. A hobbi operációs rendszer iránt érdeklődők lecsapnak a kezdeményre (gondoljunk az egymástól távol élő fogpiszkáló-faragókra), és a rút kiskacsából hamarosan hatalmas (továbbra is rút) sasmadár lesz.

Mindenki saját elképzeléseit építi bele – akinek GUS hangkártyája van, az ír hozzá meghajtót. Akinek több ezer processzoros számrágcsáló gépei vannak, az ír erre való rendszert. Az Eric S. Raymond által „kanonizált” bazármodell első fényes sikerét aratta. Azóta számtalan helyen alkalmazták egyszerű alapelveit – „Add ki korán, add ki gyakran”, „sok szem több hibát lát” –, köztük legalább egyszer tudatosan, ellenőrizendő a bazármodell elméletét.

Eric S. Raymond híres könyve, *A katedrális és a bazár* a fetchmail megalkotásának folyamatán mutatja be a bazármodell 19 szabályát, a követendő eljárásokat és magatartásformákat. Az elmélet bizonyítéka pedig maga a fetchmail, egy híresen kiváló, pótolhatatlan, és nagyjából versenytárs nélküli program.

4.3. A bazármodell nem informatikai használata

A bazármodell alapelvei, az, hogy használjuk ki sok ember közös erőfeszítését az internet adta kommunikációs lehetőségek révén, meglehetősen messzire visz. Néha egészen meglepő helyeken lehet használni, a legújabb példa erre a „tőzsde” jellegű közösségi „jóslat”, ahol interneten fogadhatunk jövőbeli eseményekre, árfolyamokra, terrorcselekményekre, és a kialakuló árfolyamok néha meghökkentően pontosan jelzik előre a történéseket. Egy másik érdekes kezdeményezés a Wikipedia, egy az olvasók által bővített és karbantartott enciklopédia. De kicsit távolabbról ide köthető a Rejtő Jenő utca közelmúltbeli megszületése, vagy a rajongók által megmentett utolsó magyarországi NOHAB mozdony története. Lassan, bátortalanul, apró lépésekben egyre több helyen találkozhatunk a közös bölcsesség erejével.

5. Zárszó: csillogó tekintettel büszkén a jövőbe

Remélem, mostanra mindenki kicsit egyenesebben tartja magát, mélyebbeket lélegzik, és amint hazamegy, megváltja a világot. De nem szabad pár apróságot elfelejteni. Például a bazármodell sem mindenható – eddig nem teremtett például tökéletes irodai programcsomagot, és több ezer interneten szavazó sakkozó nem erősebb, mint egy igazán jó sakknagymester. Jelenlegi életünket sokkal inkább meghatározza az, hogy az internet közel hozza az információt és gyorsabbá teszi a kommunikációt, a bazármodellel legtöbbünk csak távolról találkozott. Az igazán nagy változások később jönnek majd. Talán amikor a bazármodellel dolgozunk ki törvényeket és költségvetéseket? De jósolni nehéz forradalmak idején. Az egyszeri sumer kereskedő, aki az első öt vonást tette a puha agyagba a tehén ábrája mellé, nem tudhatta, hogy ötezer évvel később „Edvárd király léptet fakó lován”. Úgyhogy mi se nagyon kapálózzunk, inkább vitessük magunkat az árral, és csináljuk azt, ami jólesik. Mellesleg ez a bazármodell első alapelve is.

Néhány link, további olvasásra

- Ősi írásokról
  `http://www.viewzone.com/stantest22.html`
- Gilgames
  `http://www.zone.ee/aurin/vers/wsgilgames.html`
- Gutenbergről minden
  `http://www.gutenberg.de/english/`
- A magyar szamizdatról
  `http://www.bparchiv.hu/magyar/kiadvany/bpn/05/129-172.html`
- Egy kalózrádió
  `http://tilos.hu/history.html`
- Tömör internettörténelem
  `http://www.davesite.com/webstation/net-history.shtml`
- Levelezésről bővebben
  `http://www.let.leidenuniv.nl/history/ivh/frame_theorie.html`
- Eric S. Raymond: The Cathedral and the Bazaar
  `http://www.catb.org/ esr/writings/cathedral-bazaar/`
- A NOHAB mozdonyról
  `http://www.nohab-gm.com/hu/hu_000.html`
- Kínai és más űrhajókról
  `http://forum.index.hu/forum.cgi?a=t&t=9009306&uq=2049`
- Fogpiszkálók
  `http://www004.upp.so-net.ne.jp/mrkei/english/youji.htm`

Tartalomkezelő Rendszer(ek)
TYPO3

Tóth Sándor <toth.sandor@honlapgyar.hu>
OctaSoft MultiMedia Bt.

2003.11.08.

Kivonat

A Tartalomkezelő Rendszerek lehetővé teszik, hogy speciális programozói ismeret nélkül egy átlagos felhasználó saját maga tudja karbantartani a honlapját. A honlap egy egyszerűen átlátható felületen keresztül módosítható. Az előadás során több Tartalomkezelő Rendszer kerül összehasonlításra. A TYPO3 CMS rendszer pedig részletesebben is bemutatásra kerül.

Tartalomjegyzék

1. Honlapkészítési technikák

Rövid bevezetés

- Mi a honlap, mire jó?
- Mi szükséges egy honlap készítéséhez?
- HTML szerkesztő programok
- Kép szerkesztő programok
- Dinamikus weblapokról
- Honlapkészítés fázisai

2. Tartalomkezelő Rendszerek

2.1. Mit nevezünk Tartalomkezelő Rendszernek?

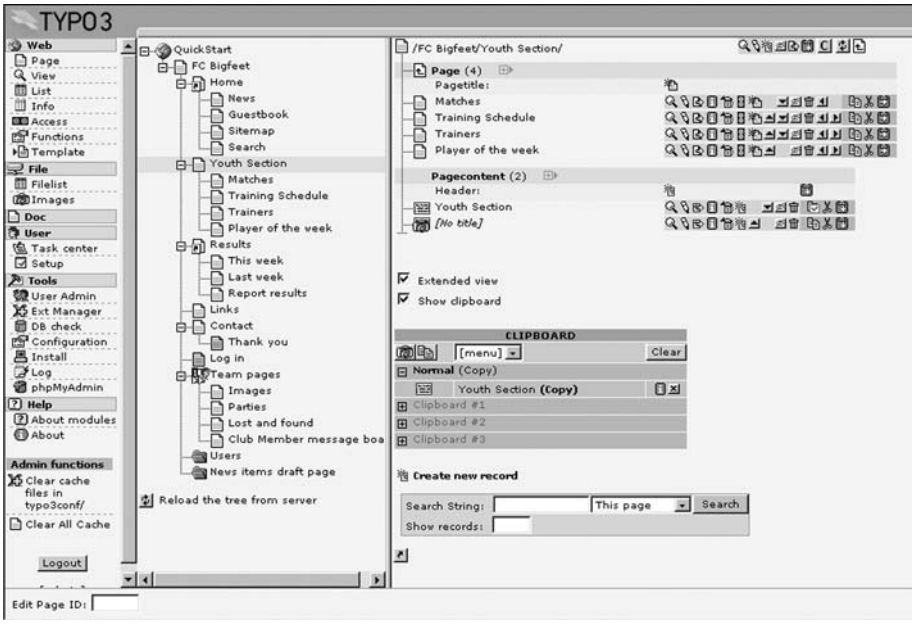
A Tartalomkezelő Rendszerekben a design és a tartalom elkülönítve kerül kezelésre. A honlap kinézetét a template (sablon) írja le. A design a honlap átadása után általában nem változik, de bármikor lehetőség van új arculat kialakítására, a meglévő tartalmi elemek megtartásával.

A tartalmi elemek általában adatbázisban vagy külön fájlokban tárolódnak, így a honlap szerkesztőjének csak a tényleges tartalmi részre kell odafigyelnie.

A legtöbb Tartalomkezelő Rendszer támogatja a csoportmunkát, így nagyobb portálok is hatékonyan menedzselhetők. Egy web-böngésző segítségével programozási ismeretek nélkül lehet bővíteni, törölni, frissíteni a honlap tartalmát. Egyszerűen alakítható a honlap struktúrája, oldalak beszúrásával, módosításával vagy törlésével.

2.2. Tulajdonságok

- egy web-böngésző segítségével kezelhető
- nincs szükség programozási ismeretekre vagy speciális szoftverre
- gyorsan megtanulható és könnyen kezelhető
- WYSIWYG szövegszerkesztő
- előzetes nézet
- több felhasználó kezelése
- csoportmunka támogatása
- jogosultság kezelés
- tartalom módosításainak követése, visszaállítási lehetőség
- tartalom publikálásának engedélyezése
- tartalom időhöz kötött megjelenése, eltüntetése



- külső felhasználók kezelése (egyes funkciók csak bejelentkezés után érhetők el, rejtett oldalak)
- honlap struktúra módosíthatósága
- hivatkozások (linkek) kezelése
- már meglévő külső oldalak beépíthetősége
- többnyelvű honlap támogatása
- látogatottsági statisztika
- fájl kezelés
- cserélhető template (design)
- moduláris felépítés, bővíthetőség, testre szabhatóság

2.3. Tartalomkezelő Rendszerek összehasonlítása

Néhány ismert és kevésbé ismert Tartalomkezelő Rendszer kerül összehasonlításra ebben a részben.

3. TYPO3

3.1. Erőforrásigény

www-kiszolgáló

- operációs rendszer: Linux, Windows vagy Mac

- web kiszolgáló: Apache, IIS
- futtató környezet (középső réteg): PHP4
- adatbázis (belső rendszer): MySQL
- adatbázis (felhasználói kiterjesztés): támogatás: Oracle, MS-SQL, ODBC, LDAP – gyakorlatilag bármilyen külső adatforrás (PHP4 segítségével)
- opcionális programok: (ImageMagick, GDlib/Freetype, zlib, Apache mod_gzip/mod_rewrite, PHP-cache (csak UNIX rendszerben pl: PHP-accelerator / Zend Accelerator)
- hardver: egy átlagos webszerver, modern processzorral, minél több memóriával

ügyfél (adminisztrátor)

- valamilyen grafikus böngésző program (Netscape, IE, Opera, Konqueror) bármilyen operációs rendszeren (Windows, Unix, Mac)
- hardver: aránylag modern számítógép

3.2. Telepítés

3.3. Főbb jellemzők

- helyzetérzékeny súgó
- többnyelvű kezelőfelület
- modulok
- kereső motor
- direct-mail támogatás
- vágólap
- WYSIWYG szövegszerkesztő
- moduláris felépítés (Extension manager)
- fájl kezelés
- gyorsan megtanulható és könnyen kezelhető
- design gyorsan cserélhető
- alkalmazásgenerátor (Extension kickstarter wizard)
- opcionális képfeldolgozás, képkezelés (GDlib, Freetype and ImageMagick)

3.4. TYPO3 bemutató

Egy minta honlap létrehozása, amely néhány oldalból áll

- oldalak létrehozása
- tartalom (szöveg, kép) szerkesztése
- design alakítása

3.5. Programozás

- TYPOSCRIPT nyelv (weboldal leírásához, pl. menü vagy grafikus fejléc beillesztése)
- PHP kód futtatása (meghívható a TYPOSCRIPT-ből)
- Extensions Wizard (alkalmazás generátor létrehoz egy minta programkódot (PHP), amelyet felhasználva gyorsan lehet kiegészítő modulokat fejleszteni)

4. Hivatkozások

TYPO3

- honlap `http://typo3.com/`
- demó oldal `http://demo.typo3.com/`
- fejlesztőknek szóló információk `http://typo3.org/`

és még néhány Tartalomkezelő Rendszer

- ezContents `http://www.ezcontents.org/`
- Mambo `http://www.mamboserver.com/`
- PHPNuke `http://www.phpnuke.org/`
- Plone `http://www.plone.org/`
- PostNuke `http://www.postnuke.com/`
- xaraya `http://www.xaraya.com/`

végül egy linkgyűjtemény

- `http://cms.lap.hu/`

az előadás teljes anyaga pedig itt letölthető

- `http://www.honlapgyar.hu/`

FSRv2: Fordítást Segítő Rendszer (v2)

Verók István

2003.11.08.

Kivonat

A szoftverek (magyar) nyelvre fordítása jelenleg sok technikai kihívást jelentő feladat. Csak címszavakban: sokféle formátum, nehézkes szövegszerkesztés, szókincs-egységesítés csak kézzel, a régi fordítások automatikus újrafelhasználása minimális, több fordító egyidejű munkájának összehangolása kevéssé megoldott. Egyes projektek saját igényei szültek eszközöket (pl. a magyar OpenOffice.org fordítási felülete), melyek a megcélzott gondokat a maguk módján orvosolják. Általánosan használható, formátumokon és projekteken átlépő, sok fordítót támogató szabad szoftveres rendszer azonban még nem lépett színre.

Ezzel próbálkozik meg a készülőben levő FSRv2 [1]. A csoportmunkát elsősorban folyamatosan frissített és közösen használt fordítási memóriák és szótárak rendelkezésre bocsátásával támogatja, így az egyre bejövő eredmények mindenki számára visszaterítése automatikus. A fordítási memória mondatonként (szegmensenként) visszakereshetően tárolja el forrás- és célnyelvű mondatok párjait. A szótár hasonló, de a frissítése kézzel, közmegegyezéssel történik. E két erőforrással a fordítási munkafolyamat több lépése is hatékonyan támogatható (pl. előfordítás, terminológiai egységesítés).

Az FSRv2 felépítésének ismertetése után röviden implementációs részleteket is érintünk, végül egy komplett munkafolyamatot végigkövetve a felhasználó oldaláról nézzük meg, hogyan teszi kényelmesebbé a rendszer a fordítói, lektori munkát.

Tartalomjegyzék

1. Problémafelvetés

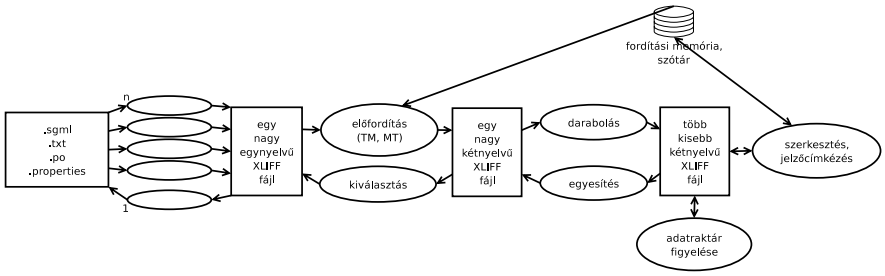
Ahány program, szinte annyiféle módon tárolják a felhasználó szeme elé kerülő szövegeket. C és C++ programoknál elterjedt a GNU .po (Portable Object) formátum. Java-kódban sokszor látni .properties fájlokban levő kulcs-érték szövegpárokat. Egyéb dokumentáció a DocBook-tól kezdve HTML-en át egyszerű szövegfájlokkal bezárólag bármiben lehet. Az emberi találékonyság határtalan, az SQL Ledger magyarításához például egy Perl asszociatív tömb forráskódszintű átírása kellett.

Egy mezei szövegszerkesztővel persze minden feladat megoldható – de milyen áron? Mennyi fáradságot és felesleges munkát lehetne megtakarítani például azzal, ha nem kéne minden egyes fordítást kézzel begépelni (vagy mondatonként kivágni és bemásolni), újra és újra, minden egyes új programverzió megjelenésekor? (A .po formátumot körítő eszközök ugyan adnak némi támogatást ilyen esetekre, a többit azonban ez nem segíti.)

Bonyolultabb feladatok pedig (pl. fordítás egységes szókincsének biztosítása, csoportmunka segítése) ma jórészt megoldatlanok. A szabad szoftverek fordítói sok olyan munkát kénytelenek emberi erőfeszítéssel, idővel és szorgalommal pótolni, amit egy hivatásos fordító (a maga eszközeinek birtokában) egyszerűen nem lenne hajlandó kézzel elvégezni. Arra való a gép, hogy dolgozzon, mondják. Ideje ebben a munkakörben nekünk is munkára fogni.

2. Az FSRv2 szerkezeti áttekintése

A mondás szerint egy kép többet ér ezer szónál: 1. ábra.



1. ábra. Adatmozgás az FSRv2 részei közt

A felső, balról jobbra vezető nyilak mutatják a kezdeti lépéseket: a forrásdokumentumokat valamilyen formátumból importálni kell egy közös XLIFF hordozóformátumba (l. alább), ilyenkor történik a kezdeti szegmentálás, majd (régebbi verzió megléte esetén) a félautomatikus előfordítás. Eztán lehet az egy nagy XLIFF fájlt több egységre bontani (célszerűen úgy, hogy nagyjából azonos mennyiségű lefordítatlan szöveg legyen bennük), és jöhetnek a fordítók/lektorok az egyes egységeken. A végtermék kinyerése ugyanez fordítva (alsó nyilak, jobbról balra): összefésülés, csak a célnyelvi verzió megtartása, és az eredeti formátumba visszaírás. A csoportmunka menete a közösen használt adatraktár (repository) figyelésével követhető. A segítő erőforrások (fordítási memória, szótár) láthatóan két fázisban is részt vesznek, először az előfordításban játszanak szerepet, majd a fordítók és lektorok munkáját segítik.

Az implementációról röviden

A szoftver legfontosabb jellemzője, hogy a fordítási memóriákkal és szótárakkal kommunikációra használt web service interfész teljesen elhatárolja a klienst és szervert, valamint az interfész megsértése nélkül bármelyikük lecserélhető ekvivalens funkciójú másik implementációra. (Web service: a hálózaton egy WSDL-ben [5] definiált interfész metódusainak megfelelő, SOAP-borítékokba helyezett XML-üzenetek forgalmazása zajlik, de esetünkben SOAP [4] helyett lehetne akár REST-stílusú is [6].) A jelen feladatra definiált web service nagymértékben épít a Translation Memory Exchange (TMX) szabványra [2], a fordítás alatt álló fájlokat pedig a széleskörűen támogatott XML Localisation Interchange File Format (XLIFF) szabvány [3] szerint tároljuk.

A protokoll jelen implementációja Javában készül, szerveroldali részei természetes módon servletekkel implementálhatók, a fordítási memória és szótár-szerverek API-felületei éppúgy, mint a fordító számára látható webes kliens HTTP-felülete. Futtatáshoz nem feltétlenül szükséges JVM – Linuxon létezik a GCJ frontend, amellyel a GCC számára emészthető szintaktikai fa készíthető Java forrásból és .class fájlokból is, ezt a fát pedig már ugyanaz a GCC backend kód fordítja és optimalizálja futtatható natív kóddá, mint mondjuk a C++ forrásból készült fát. A bemeneti adatformátumok leírása egy formális nyelvtannal történik, melyből a SableCC fordítógenerátor [7] készít a bemeneti formátumból olvasó Java kódot.

3. Tipikus felhasználás menete

Általában az alábbi munkafázisokat lehet elkülöníteni.

Új fordítási projekt indítása vagy egy meglevő frissítése

- A lefordítandó fájlok (.po, .txt, .sgml, stb.) importja.
- Ezek valamekkora egységekre darabolása (nem feltétlenül csak fájlhatárok mentén, lehet, hogy egy hatalmas méretű fájl több egységre bomlik, vagy ellenkezőleg, sok kisebb fájl is lehet egyetlen egységben).
- Ennek célja, hogy a fordítók jól tervezhető méretű terhelést kapjanak, pl. 3000-3600 szót, és az egységeket fogják kivenni.
- Felhasználandó erőforrások megadása (melyik fordítási memóriában gyűljenek a munka során keletkező szegmensek feljegyzései, milyen egyéb fordítási memóriákban keressen még további, már lefordított szegmensjavaslatokat, milyen szótárakat használjon, pl. közös GNOME-terminológia).
- A megadott erőforrások alapján a fordítandó szöveganyag előfeldolgozása: szegmensekre (~mondatokra) bontás, a szegmensek félautomata előfordítása a fordítási memóriák javaslatai alapján. (Előfordítás pl. úgy, hogy a teljes egyezéseket elfogadja (pláne, ha a környezetében is mondjuk előtte-utána levő szegmensek stimmelnek), a bizonyos jóságúnak ítélt fuzzy match-eket szintén.)

Fordítás

- A fordító kivesz (magáénak nyilvánít) egy egységet, végigmegy a benne levő fájl(darab)okon, azok szegmensein.

- A szövegösszefüggést is látva fordít.

- Munkáját segíti a dinamikusan kezelt és frissített fordítási memória (ami minden újonnan felvett szegmenst (eredetit és fordítást is) azonnal mindenkinek feldob, aki aztán hasonló eredetit talál) és a szótár (amibe konszenzusosan, nem automatikus módon, hanem kézzel vesznek fel újabb szavakat és kifejezéseket).

- A végén az egység visszakerül a közösbe.

Lektorálás

- A lektori teendőket általánosítva kezeljük, így a fordítók és a lektorok közt csak annyi különbség van, hogy ki milyen jelzőcímkét tehet rá az egyes szegmensekre (és vehet le azokról). Fordítónak is értelmes lehet pl. egy „segítség kell” jelzéssel ellátni azt, amiben bizonytalan, és szeretné, ha valaki utólag átnézné. Illetve „nyersfordítás” jelzőcímkével jelzi, hogy egyáltalán ott járt. Lektornak pedig pl. olyan jelzőcímkék állhatnak rendelkezésére, hogy „egyszer átnézve”, „GNOME-terminológiával kompatibilitás szempontjából átnézve”, stb. Ebből is látszik, hogy a jelzőcímkéket menet közben is lehet bővíteni, a projekt előrehaladásával felmerülő újabb igények esetén újabb saját jelzőcímkéket lehet definiálni.

- Érdemes lehet pl. a gép elé leüléskor elhatározni, hogy ha a lektor most a szókincs GNOME-kompatibilitását ellenőrzi, akkor minden láttamozott szegmenst önműködően lásson el „GNOME-terminológiával kompatibilis” jelzőcímkével, és akkor erre sem kell külön odafigyelni.

- Egyébként a lektor is ugyanazt teszi, mint a fordító: kivesz egy egységet, végigmegy a szövegen, szerkeszt, visszateszi az egységet. Ő is használ segédeszközöket (fordítási memória, szótár), esetleg még egy tipikusan lektori eszköze lehet: ha már van régebbi kész verzió is az adott fordítandóból, akkor diff alapján könnyebben kiszűrheti a figyelmére érdemes változásokat.

Végtermék elkészítése

- Lehet köztes leszállítandókat is készíteni (pl. az OpenOffice.org esetében: csak resource fájlok).

- És lehet komplett build is: pl. kész telepítőcsomag.

- Ilyenkor minden egység legfrissebb állapotát visszafésüli, összeállítja belőlük a kellő fájlokat (a több egységbe darabolt dolgokat sorrendben egybemásolja, stb., tehát minden a helyére kerül), majd ezeknek a vég-programrendszerbe beillesztése után indulhat a bináris fájlok, erőforrásfájlok, stb. lefordítása.

- Ezek támogatására lehetne egy „build gomb”, ami pl. egy Makefile vezényletével egy kattintással elvégzi a leszállítandók adott állapotú lefordítását.

4. Zárszó

A vázolt működés teljes megvalósításáig még sok munka van hátra (az aktuális verzió addig is megtalálható a projekt honlapján). Jelen pillanatban (2003 ősze) kevésbé látványos infrastrukturális alapok vannak készen, és a munka (bizonyos kihagyás után) ismét folyik. Stay tuned.

Hivatkozások

[1] FSRv2 honlap: `http://photon.hu/fsr/`

[2] Translation Memory Exchange szabvány (TMX):
`http://www.lisa.org/tmx/`

[3] XML Localisation Interchange File Format szabvány (XLIFF):
`http://www.oasis-open.org/committees/xliff/documents/xliff-specification.htm`

[4] Simple Object Access Protocol szabvány (SOAP):
`http://www.w3.org/TR/soap/`

[5] Web Services Description Language szabvány (WSDL):
`http://www.w3.org/TR/wsdl/`

[6] REpresentational State Transfer (REST) wiki:
`http://internet.conveyor.com/RESTwiki/moin.cgi/FrontPage`

[7] SableCC fordítógenerátor honlap: `http://www.sablecc.org/`

Programok fejlesztése Linux környezetben – a soros porti kommunikációtól a többszálas programozásig

Vomberg István, Dröszler Attila

2003.11.08.

Kivonat

Cikkünkben rövid áttekintést kívánunk adni a Linux operációs rendszeren futó, külső hardver eszközökkel kommunikáló, többszálas, többfolyamatos programozásról. A Linux operációs rendszer kifinomult kommunikációs, adatgyűjtő és feldolgozó programok írását teszi lehetővé, ennek a technikájához kívánunk segítséget nyújtani. Bemutatjuk az alap eszközöket, mint külső kommunikáció, multiprocessz illetve multithread technikák.

Tartalomjegyzék

1. Bevezetés

A GNU/LINUX környezetben is örvendetesen szaporodásnak indultak a felhasználói programok, ám sokan nem ismerik ennek a robusztus, nagy teljesítményű környezetnek a programozási fogásait. Ennek elsősorban az az oka, hogy magyar nyelvű dokumentáció elég kevés létezik, angol nyelven pedig némi utánjárás szükségeltetik hozzá. Nem mellékes az a körülmény sem, hogy akik unixos rendszereken tanultak programozni, azok általában nem a desktop világban dolgoznak, akik viszont kisebb alkalmazói programokat írnak, azok nem ismerősek a UNIX típusú operációs rendszerekben. Egy kicsit előreszaladva egy mondatban összefoglalva a tapasztalatainkat: hihetetlen könnyedséggel és eleganciával lehet nagyon komoly feladatokat megoldani a GNU/Linux operációs rendszer által adott környezetben.
Előadásom és ez a cikk pont ezért született, hogy röviden és vázlatosan ugyan, de ismertessem a fontosabb technikákat.

A feladatok

A cikkben bemutatott módszerekkel két nagyobb lélegzetű program is készült, melyekben – ha nem is teljes mélységében, de széles spektrumban – kihasználtuk az operációs rendszer nyújtotta fontosabb lehetőségeket.
Az egyik projektben ipari vezérlő és mérési adatgyűjtő rendszert kellett építenünk, beleértve a PLC-hez hasonló – ám annál bővebb – funkcionalitású vezérlőelektronikát, majd egy szerverrel kezelni a több mint 10 egységet, a gyűjtött adatokat tárolni, webes felületen hozzáférhetővé tenni, valamint email segítségével kommunikálni a gyári mérnökökkel. Minden egyes vezérlőegység külön mérőhelyen dolgozik, villanymotorok tesztelése a feladat, egymástól teljesen függetlenül kell a munkájukat végezniük. Miután egy ilyen rendszerben bármikor fennállhat annak az igénye, hogy távolról (általában egy karakteres terminálról) be lehessen avatkozni a folyamatba, a programot ennek a figyelembevételével kellett megtervezni.
A második projekt egy műszercsalád vezérlő program. A műszereket a kémiai analitika területén használják, a szoftver feladata a hardver vezérlése, az adatgyűjtés, a mért adatok kiértékelése, grafikus megjelenítése, számítások végzése, az eredmények tárolása nyomtatása. A szoftver ebből is láthatóan teljesen *desktop* jellegű.
A programokban sok a közös vonás. Az egyik nyilvánvalóan az alkalmazott operációs rendszer és a használt grafikus felület, azaz GNU/Linux és GTK.
Programozástechnikailag a soros port (RS232) kezelése, a processzek és az interprocessz kommunikáció használata, a többszálas programozás a kapcsolódási pontok. Jelen cikk ezeket a technikákat mutatja be az Olvasónak.

2. A soros porti kommunikáció

A kommunikációs hardver kiválasztásánál több szempont is érvényesült. Egyszerű legyen, platformfüggetlen, régebbi eszközeinkkel is tudjunk kommunikálni, viszont túl nagy adatátviteli sebességre nincs szükség. Ezen szempontoknak az RS232 port használata felel meg. Jelezni kívánom, hogy miután több GPL licencelésű projekt van, melyek mikrokontrollerek és Ethernet alapú TCP/IP átvitel összeházasításával foglalkoznak, ezen a téren rövidesen még komoly fejlődésnek lehetünk szemtanúi.

A soros porti meghajtó

Az alfejezet cím talán egy kicsit félrevezető lehet, hiszen a soros porti meghajtó az az operációs rendszerünk része. Igen ám, de ettől még nem tudunk kommunikálni az RS232 vonalon! Viszont akkor mit kell tennünk, kérdezheti joggal az Olvasó. Emlékezzünk csak a UNIX alapú rendszerek szlogenjére, miszerint *minden fájl*. Magyarul írnunk kell néhány függvényt, mely megnyitja a */dev/ttySx* fájlt, ír-olvas a fájlból, majd bezárja. Így viszont egy-két ügyesen megírt függvénnyel byte szinten tudunk adatokat kezelni a soros porton.

A pollozás

A soros porti kommunikáció egyik legrázósabb része az adathibák kezelése. Bármikor, bármilyen oknál fogva megtörténhet az, hogy az adatcsomag közepén megszakad a kommunikáció. Amennyiben ez vétel közepén következik be, a *repeat until* ciklusunk a világ végezetéig várni fog a soha be nem érkező jelre. Egy ipari vezérlés esetén ennek a kellemesebb kihatása az adatvesztés, sokkal kínosabb tud lenni, ha valamilyen fizikai meghibásodást okoz a kieső adat. Hogyan lehet ezt a problémát megoldani?
Kézenfekvő a megoldás, valamilyen *timeout*-ot kell használnunk a célra, amely idő leteltével az operációs rendszer gondoskodik a program továbbhaladásáról.
Erre a célra bemutatunk egy kis mintaprogramot. Ez praktikusan néhány soros port kezelőfüggvény és változó. Az egyszerűség kedvéért a header filet egybevontuk a C kódfilellal, illetve a hibakezeléseket, hibakiíratásokat kivettük.

A mintaprogram

```
#include <stdio.h>
#include <termios.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/poll.h>
#include <stdlib.h>
#include <string.h>

int port;
FILE *tty;
struct termios old_port_params, new_port_params;
int timeout_flag;
int timeout_value;

int tty_init (char *serial_port)
{
int result = 0;

  port = open (serial_port, O_RDWR | O_NOCTTY);
  if (port < 0) return -1;

  result = tcgetattr (port, &old_port_params);
  if (result != 0) return -1;

  new_port_params.c_cflag = B9600 | CS8 | CLOCAL | CREAD | CSTOPB;
  new_port_params.c_iflag = IGNPAR;
```

```
  new_port_params.c_oflag = 0;
  new_port_params.c_lflag = 0;

  result = tcflush (port, TCIFLUSH);
  result = tcsetattr (port, TCSANOW, &new_port_params);
  if (result != 0) return -1;
  tty = fdopen (port, "a+");
  setvbuf (tty, NULL, _IONBF, 0);
  timeout_flag=FALSE;
  timeout_value=100;
  return 0;
} // ttyinit

int tty_flush (void)
{
int result = 0;

  result = tcflush (port, TCIFLUSH);
  return 0;
} // tty_flush


int tty_reset (void)
{
int result = 0;

  result = tcflush (port, TCIFLUSH);
  result = tcsetattr (port, TCSANOW, &old_port_params);
  if (result != 0) return -1;
  return 0;
} // tty_reset

unsigned char readbyte(void)
{
struct pollfd pfd;

  timeout_flag=FALSE;

  pfd.fd=fileno(tty);
  pfd.events=POLLIN;
  if (poll(&pfd, 1, timeout_value)==0)
  {
  timeout_flag=TRUE;
  return 0;
  }
  else return (unsigned char)fgetc(tty);
}

int dputc(char databyte)
{
  fputc(databyte, tty);
  return 0;
} // dputc
```

3. A processzek, IPC

A processz egy programnak az éppen futó példánya. Ha két terminál ablakot megnyitunk a képernyőn, akkor két terminál processzünk lesz. Ezek természetesen futtatnak egy-egy shellt, úgyhogy alájuk rendelve azokból is lesz egy-egy. Ha megnézzük pél-

dául a `gnome-system-monitor` programmal, hogy milyen processzek futnak a gépünkön, meglepően sokat fogunk látni, de például az apache webszerver esetében több egyforma példányt is, külön-külön azonosítóval, PID-del. A programozáshoz elsősorban két dolgot kell ismernünk a processzek világából, egyrészt hogy tudjuk létrehozni őket (és miért), másrészt pedig hogy tudjuk kommunikációra késztetni az egyes processzeket egymással.

A fork(), exec

Processz minden esetben csak processzből tudunk indítani, legyen az egy terminál shellje – és akkor begépeljük a program nevét –, vagy akár egy program, mely maga végzi az indítást – így szaporodik például az `apache`. A processzekhez tartozik egy egyedi azonosító, ez a PID, mely megkülönbözteti egymástól a futó processzeket. A processzek közt van egy különleges példány, ennek azonosítója `1` és ez az `init`, mondhatni minden process ős-anyja. Ennek később még szerepe lesz a processzek tárgyalásánál. Jelen esetben a processznek a programból történő létrehozását vizsgáljuk közelebbről.
A `fork()`.
Egy új processzt úgy tudunk létrehozni – amellett hogy megmarad az eredeti is – hogy duplikáljuk, megkettőzzük az éppen futó példányt. Amennyiben a `fork()` rendszerhívást hajtjuk végre, a processzünkből hirtelen két példány fog létezni, a szülőprocessz (parent) és a gyermekprocessz (child). A PID és még egy-két egyéb változó kivételével a tökéletes másolatát kapjuk az eddig futott processznek. Jó-jó! De a programból hogy tudjuk eldönteni, hogy most éppen melyikben vagyunk a kettő közül? Az egyik a szülő a másik a gyermek, a fork()-olás után pedig két egyforma programban is ott vagyunk a következő utasításon... Természetesen a PID alapján, de nem olyan egyszerűen. Nézzünk meg ehhez egy fork()-olási folyamatot.

A fork mintaprogram

```
pid_t    ChildPID;

  ChildPID = fork ();
  if (ChildPID != (pid_t)0) {
  // Parent
  printf ("Parent process, PID = %d\n", getpid ());
  } else {
  // Child
  printf ("Child process, PID = %d\n", ChildPID);
  if (execl ("program2", "program2", NULL) == -1) {
    printf ("***Error at execve\n");
    exit (0);
  } // execve
  } // ChildPID
```

Ahol a „program2” az újonnan indított program neve.

A program értelmezése

A döntés lényege, hogy a `fork()` visszatérési értéke a gyermek processz PID-je. Azaz ha ez az érték nem `0`, akkor a szülő processzben vagyunk, ellenkező esetben pedig a gyermek processzben. Ezt egy `if` utasítással eldönthetjük, így tovább tudunk

lépni. A gyakorlatban ilyenkor a gyermek processz el szokott indítani egy másik folyamatot (az `apache` esetében például nem, ott megkapja az éppen beérkezett kérést és ő fogja kiszolgálni, míg a szülő processz várja a következő lekérést és ismét szaporodik), amit láthatunk is a mintaprogramban, ezt az `exec()` családdal tudjuk intézni. Az `apache` esetében például nem, ott a gyermek processz megkapja az éppen beérkezett kérést és ő fogja kiszolgálni, míg a szülő processz várja a következő lekérést és ismét szaporodik ha szükséges.
Az `exec()`.
Egy új processz indítását az `exec()` család valamely tagjával végezzük. Az egyes exec() variánsok között a különbség annyi, hogy a program parancssori paramétereit hogy kezelik, azaz programból milyen módon tudjuk átadni őket az új processz részére. Az alapfüggvény az `execve()`, a többi az front-end ehhez, különböző argumentumhívási technikákkal. Definíciója:

```
int execve(const char *filename, char *const argv [],
char *const envp[]);
```

Visszatérése a függvénynek csak az esetben van, ha nem sikerült indítani a kívánt programot, egyébként értelemszerűen nincs. Amennyiben sikerült indítani a `filename`-el jelzett programot, az eddig futó processz megszűnik és az új program foglalja el a helyét. A paraméterek értelmezése:
`filename` Az indítani kívánt új program neve
`argv []` A parancssori paraméterlista
`envp[]` A környezeti változók, ha újakat szeretnénk átadni
A `filename`-nek létező binárisra vagy scriptre kell mutatnia. Az `arg[]` egy string tömb, azonban figyeljünk arra, ha nem is adunk át paramétert, a 0. indexű paraméternek az új program nevének kell lennie, azaz rejtetten, de egy paraméter mégiscsak átadásra kerül. Az `envp[]` szintén string tömb, kulcs-érték páros módon kell a kívánt környezeti változókat átadni. Az `argv[]` és az `envp[]` a `main` függvény `int main(int argc, char *argv[], char *envp[])` definíciójának megfelelően kerül átadásra.

A jelek (SIGNAL)

Most már, hogy el tudunk indítani egy processzt, felmerül a kérdés, hogy a szülő és a gyermek processz milyen módon tud kommunikálni egymással. Szükség lehet például arra, hogy a szülő processz értesítse a gyermek processzt egy eseményről, vagy éppen a futásának megszüntetését kezdeményezze. Erre valók a JELek (SIGNAL), melyek a gyakorlatban egy szoftver interruptot kezdeményeznek az éppen megcímzett folyamatban, s a programozó szándékától függően kerülnek lekezelésre. JELet a `kill()` függvénnyel küldhetünk egy adott processznek. Definíciója:
`int kill(pid_t pid, int sig);`, ahol
`pid` a megcélzott processz PIDje
`sig` a kívánt jel sorszáma.
Csak néhány példát kiragadva a JELek közül:

- SIGKILL 9 – Azonnali megállítás és eltávolítás
- SIGSEGV 11 – Segmentation fault hiba jelzése, az operációs rendszer küldi a processznek
- SIGTERM 15 – Befejezésre felszólító jel, a program még kap lehetőséget a lezáratlan ügyeinek elintézésére

- SIGUSR1 30,10,16 – Felhasználói jel, szabad felhasználású
- SIGUSR2 31,12,17 – Felhasználói jel, szabad felhasználású
- SIGCHLD 20,17,18 – A gyermek processz befejezte futását, az operációs rendszer küldi a szülőnek
- SIGSTOP 17,19,23 – Processz megállítása, de nem megszüntetése
- SIGCONT 19,18,25 – A megállított processz folytatása

Amint látjuk, egy-egy jelhez (operációs rendszertől függően) több érték is tartozhat, ezért *ne* a közvetlen értékekkel hivatkozzunk rájuk, hanem az adott konstanssal, például `kill(25114,SIGUSR1)`.
Megoldottuk a JEL küldését, most megismerkedünk a beérkező JEL kezelésével a fogadó programban.

Mintaprogram a JEL elfogására

```
void SIGUSR1_handler (int signal_number);

void SIGUSR1_handler (int signal_number)
{
  printf ("SIGUSR1 received.\n");
} // SIGUSR1_handler

int
main (int argc, char *argv[])
{
struct sigaction SIGUSR1_sa;

  memset (&SIGUSR1_sa, 0, sizeof (SIGUSR1_sa));
  SIGUSR1_sa.sa_handler = &SIGUSR1_handler;
  sigaction (SIGUSR1, &SIGUSR1_sa, NULL);
/*
Itt jön maga a program
*/
}
```

A programhoz néhány megjegyzés

A JEL beérkezése egy szoftveres interrupt, annak minden következményével.

- Kernel folyamatot is megszakít, tehát ha rendszerhívást használunk valahol a programban, akkor a visszatérési értéket mindig vizsgáljuk le az EINTR értékre, mert lehet, hogy egy JEL érkezése megszakította azt! Ez esetben újra kell hívni. Ilyenek például az `open()`, `close()`, stb.
- Nem reentráns kódot ne helyezzünk el a JEL kezelőjében, a mintában adott `printf()` elvileg hibás, a gyakorlatban kicsi a valószínűsége hogy problémát okoz. Leggyakrabban csak egy flaget, mutexet, szemafort állítunk a kezelőben, a többiről a program eredeti függvényei célszerű, ha gondoskodnak.
- Egyes esetekben (például a SIGSEGV esetén egyes rendszereken) a régi kezelőt el kell tárolnunk és visszaállítanunk a program végeztével. Ilyenkor a `sigaction()` harmadik paramétere nem `NULL`, hanem egy másik sigaction adatstruktúra.

- Vannak olyan JELek, melyeket nem lehet kezelővel elfogni. Ilyenek például a SIGKILL.
- Többszálas programozás esetén a JELek kezelése külön figyelmet igényel.

Az Inter Process Communication természetesen nem csak a JELek kezeléséből áll, azonban terjedelmi okokból csak megemlíteni tudjuk a többi módszert:
Folyamatos adatkapcsolatra két processz közt a csövek (PIPE) szolgálnak.
A leggyorsabb adatmegosztási technika az osztott memóriakezelés, amikor egy memóriaszeletet több processz közösen tud használni.
A processzek közt szemaforokkal (semaphor) lehet szinkronizálni a folyamatokat, illetve például az osztott memóriához való hozzáférést, hogy egymás feje fölött átnyúlkálva ne tudják összekuszálni az adatok egységességét.

4. A többszálas programozás

Felmerülhet az igény, hogy a programban az adatokon egy-egy függvény műveleteket végezzen, de a „háttérben”, azaz magát a programot ne tartsa fel (mert például hosszas a számítás), tehát aszinkron módon, de a folyamatot kézben tudjuk tartani. Erre szolgál a szálkezelés.

Mi az a szál?

A szál (thread) egy könnyűsúlyú processz, ami azt jelenti, hogy egy szál megnyitásánál az adatterületek nem duplikálódnak csak a program, de a rendszer processzként tartja nyilván a szálat. Tehát közös adatterületen különböző folyamatok végezhetnek műveleteket.

A többszálas programozás jelentősége

Miért jó ez nekünk és „mibe kerül”? Egyrészt azért jó, mert nem kell külön processzt indítani egy művelet elvégzéséhez – ami rendszerszinten mérve „drága” tranzakció –, hanem csak egy threadet, ami egy egyszerűbb duplikálási folyamattal jár. Programozói szemmel nézve a nagy előny, hogy megmarad az adatterület, tehát rengeteg IPC-t takarítunk meg, ami szintén rendszer erőforrásokat köt le. Ez egyúttal egy hátránya is a szálkezelésnek, ugyanis az adatok konzisztenciájáról nekünk kell gondoskodnunk programozás közben, erre a `mutex` nevű zárak (lock) szolgálnak, amivel megakadályozható, hogy például egy karakter tömbbe (string) egyszerre ketten írjanak bele s a végeredmény egy siralmas zagyvalék legyen. Ez abból következik, hogy a szálak aszinkron módon kerülnek végrehajtásra, azaz a végrehajtási sorrendet nem áll módunkban befolyásolni, ezt az operációs rendszer végzi.

Szálak kezelése

Egy szál létrehozását majd megszüntetését a legjobban a becsatolt kis példaprogramon vehetjük szemügyre.

- `pthread_create()` a szálhoz tartozó függvényt csatoljuk be
- `pthread_cancel()` megszüntetjük a szál futását

- `pthread_join()` megvárjuk amig a szál teljesen kilép
- `pthread_mutex_init()` mutexet iniciálunk
- `pthread_mutex_destroy()` megszüntetjük a mutexet
- `pthread_mutex_lock()` zárjuk a mutexet
- `pthread_mutex_unlock()` nyitjuk a mutexet

A mutexek egy pár szót megérdemelnek külön is. Az adatok biztonságát csak úgy tudjuk garantálni, hogy mindazon utasítások, melyek egy változón változtatást végeznek, egy zárolási mechanizmust használnak. A mutexek következetes használatával elérhetjük azt, hogy egy változón, adatterületen egyszerre csak egy szál tudjon változtatást végrehajtani, s így az adatok konzisztenciája megmarad.

Gyakorlati példa, blocking IO kezelése

```
pthread_t        measure_thread;
pthread_mutex_t  measure_data_mutex;
int             counter;
char*           data_stream;

extern void*
measure_function (void* unused)
{
char dummy;
  for („) {
    dputc ('\0x00');
    dummy = readbyte ();
    pthread_mutex_lock (&measure_data_mutex);
    data_stream [counter] = dummy;
    counter++;
    pthread_mutex_unlock (&measure_data_mutex);
  }
  return NULL;
} // measure_function

int
main (int argc, char *argv[])
{
int end_flag;
  data_stream = alloca (1000);
  pthread_mutex_init (&measure_data_mutex, NULL);
  pthread_create (&measure_thread, NULL, &measure_function, NULL);

  end_flag = 0;
  while (end_flag == 0) {
    pthread_mutex_lock (&measure_data_mutex);
    /* Itt még csinálunk az adatokkal valamit */
    if (counter == 999) end_flag = 1;
    pthread_mutex_unlock (&measure_data_mutex);
  }

  pthread_cancel (measure_thread);
  pthread_join (measure_thread, NULL);
  pthread_mutex_destroy (&measure_data_mutex);

  printf ("%s", data_stream);
```

```
  return 0;
}
```

Megjegyzések a mintaprogramhoz

A program nem tartalmaz minden `include` hivatkozást, a főbb folyamatokat szemlélteti. A `measure_function`-ban a soros porti kommunikációból ismert függvényeket használjuk, melyek *blocking IO* jellegűek, azaz a `readbyte ()` akár perceket is állhat egy helyben a `timeout_value` értékétől függően. Szálkezeléssel végezve a soros kommunikációt azonban ez az időszakos „fagyás” a főprogram működését nem befolyásolja, ott a szükséges műveleteket közben el tudjuk végezni.

Felhasználói programok fejlesztése GTK+/Gnome környezetben

Vomberg István, Dröszler Attila

2003.11.08.

Kivonat

Cikkünkben rövid áttekintést kívánunk adni a Linux operációs rendszeren futó, Gnome/GTK+ grafikus környezet alá történő programozásról. A Linux operációs rendszer kifinomult desktop programok írását teszi lehetővé, ennek a technikájához kívánunk segítséget nyújtani. Bemutatjuk a grafikus rendszerek felépítését, használatát és a nyomtatást.

Tartalomjegyzék

1. Bevezetés

A GNU/LINUX környezetben is örvendetesen szaporodásnak indultak a felhasználói programok, ám a fejlesztők egy kicsit idegenkednek ennek a környezetnek a használatától. Ennek elsősorban két oka van. Az egyik a túl bőséges választék, mint például grafikus eszközkészletből (widget set) is legalább négy van szélesebb körben használatban (Motif, Athena, Qt, GTK+) bár az utóbbi időben csak a Qt és a GTK az, amelyik komoly szerepet játszik ezen a téren. A másik problémaforrás már komolyabb, az pedig a dokumentációk és segédletek (tutorialok) szűkebb volta. Itt szinte hallom felhorkanni a programozók egy részét és lelki füleimmel az *eufemizmus* szót hallom kiáltani, mert például a libgnomeprint könyvtár dokumentációja a teljesség tekintetében enyhén szólva is kívánnivalókat hagy maga után és akkor még finom voltam.
Előadásom és ez a cikk pont ezért született, hogy egy lehetséges utat mutasson ezen a területen, természetesen nem az egyedül üdvözítőt. Az általam ismertetett feladatokat más eszközökkel, más programozói technikákkal ugyanilyen jól és szépen meg lehetett volna oldani, de talán pont ez adja a szépségét ennek a rendszernek.

A feladatok

A GTK környezetben történő programozás technikáját egy konkrét feladat segítségével teszem kézzelfoghatóvá.
A projekt egy műszercsalád vezérlő program. A műszereket a kémiai analitika területén használják, a szoftver feladata a hardver vezérlése, az adatgyűjtés, a mért adatok kiértékelése, grafikus megjelenítése, számítások végzése, az eredmények tárolása nyomtatása. A szoftver ebből is láthatóan teljesen *desktop* jellegű.

2. Röviden a GLib-ről

A GLib egy keresztplatformos alapkönyvtár, amely egyrészt platformfüggetlen definíciókat biztosít a programozáshoz, konverziós és egyéb hasznos makrókat, illetve alapvető programozási feladatokhoz komplett függvénykönyvtárakat, hogy ne kelljen újra feltalálnunk a meleg vizet, ha például egy kétirányú láncolt listát szeretnénk használni. Az elemeiből csak szemelvényeket lehet adni, mivel a teljes Glib referencia csak felsorolás szinten is több tíz oldalt tenne ki.

Fontosabb elemei

- Alaptípusok számokra mutatókra. Ez biztosítja azt, hogy két nagyon különböző rendszeren is egy 32 bites előjel nélküli egész ugyanaz legyen, jelen esetben a `guint32` típus definíciójával.
- Az alaptípusokhoz tartozó számábrázolási határértékek, mint például `G_MAXUINT64` a 64 bites előjel nélküli egészre.
- Alap makrók, ilyenek például a `MIN()`, `MAX()`, `ABS()`, `CLAMP()`, valamint a verziószámot és az operációs rendszer típusát visszaadó makrók.
- Byte sorrend makrók, a big endian, little endian illetve a PDP endian közti konverziókra.

- Numerikus definíciók, ilyenek az `e`, `PI`, valamint a logaritmusokkal kapcsolatos jellemző konstansok.
- Eseményhurok, azaz main event loop. Ezt használja a GTK+ is.
- Szálkezelés, amely egy kényelmes burkoló a POSIX szálkezelő környezethez.
- Memóriakezelés, az `alloc()` függvénycsalád burkolója.
- Dinamikus modulkezelés, "plug-in" jelleggel.
- Hiba és üzenet kezelés.
- String, karakter kódolás konverzió, Unicode kezelés.
- Dátum, idő kezelés.
- XML Parser
- Windows kompatibilitási utilityk.
- Adattípusok listák, fák és egyéb adattároló technikák kezeléséhez.

3. A grafikus programok alapja, az X

Ha a desktop programunknak egy idő után szeretnénk grafikus felületet adni, hogy könnyebben kezelhető legyen, akkor Linux (és a többi UNIX alapú) rendszereken ezt az X Window System (röviden X11 vagy csak X) segítségével tehetjük meg.

Röviden az X-ről

Az X Window rendszert a MIT-n fejlesztették ki az 1980-as évek közepén. Ez egy hálózat alapú, kliens-szerver típusú felhasználói csatolófelület. Az X grafikát használó program futtatásához két dolog kell, egyrészt a program amelyet mi írtunk és az érdemi munkát végzi - ez lesz a kliens, és az X, mely a grafikus megjelenítésért felel - ez lesz a szerver. A két program nem kell, hogy ugyanazon gépen fusson, ezért hálózat alapú az X, a futó program és a megjelenítő X szerver akár a Föld két ellentétes pontján is lehet, csak elég gyors hálózati kapcsolatuk legyen. Itt még egyszer kiemelném az elnevezéseket, ugyanis az első ránézéssel ellentétben az X megjelenítő program a *szerver*! Az X szerver program fogadja azokat a kéréseket, amelyek a bemeneti eszközökről érkeznek, mint az egér, billentyű vagy a kliens program, és a kéréseknek megfelelően a kimeneti eszközökre ad választ, mint videokártya vagy a kliens program. Ezt a folyamatot a platformfüggetlen X protokollon keresztül hajtja végre a rendszer, ami azt jelenti, hogy különböző hardver architektúrájú és különböző operációs rendszerek közt is korrekt együttműködést lehet megvalósítani.

Hogyan programozunk X környezetben?

Az egyszerű válasz erre a kérdésre leginkább az, hogy *közvetlenül* sehogy. A kliensprogram részére az Xlib könyvtár tartalmazza azt a kapcsolódási felületet, mely segítségével az X környezettel kommunikálni lehet. Ez azonban csak egyszerű grafikus megjelenítésekre képes, nagyjából tudunk húzni egy színes vonalat 10-15 sornyi programozással. Készült az X-hez eszközkészlet, az Xt toolkit, mely a grafikus

programozást könnyebbé tette, de azóta sokkal korszerűbb eszközök állnak a rendelkezésünkre, mint például a Qt vagy a GTK+ eszközkészlet. A Gnome/GTK/GDK illetve a KDE/Qt rendszerekkel minden olyan dolgot meg lehet csinálni amit közvetlen X programozással is meg lehetne, azonban összehasonlíthatatlanul hatékonyabban és komfortosabban. Aki valamilyen speciális oknál fogva mégis közvetlenül szeretné programozni az X-et, annak javaslom az `http://xfree86.org/` oldal meglátogatását, illetve az Xgalaga nevű grafikus játékprogram forrásának a böngészését, az ugyanis tisztán Xlib hívásokkal működő program. A kimenete nem véletlenül emlékeztet kísértetiesen a DOS-os korszak játékaira…

4. A GDK

A grafikus programok fejlesztésénél nagyon hamar szembekerülünk olyan feladatokkal, hogy színes vonalat kell húzni egy ablakban egy rajzterületre, vagy netán a kurzor alakját kell átállítani. Ugyan az ilyen feladatokat közvetlenül az X programozásával is meg lehet oldani, azért mi mégsem azzal szeretnénk több okból is. Erről szól ez a fejezet.

Mi az a GDK?

A GDK az a könyvtár, melynek függvényei közvetlenül a grafikus alaprendszerekkel tartják a kapcsolatot, értsük ez alatt az X-et vagy az ablakkezelőt. Megjegyzésként annyit, hogy más rendszerre történő portolásnál a GDK programkönyvtár az, amelyet át kell írni. Ezért lehetséges például a GIMP portolása Windowsra is. Elnevezése a GTK+ Drawing Kit rövidítés alapján történt.

A gyakrabban használt GDK elemek

Grafikus kapcsolat felépítése.
Ne feledjük, hogy X Window környezetben programozunk, ami azzal jár, hogy a futtatott program és a grafikus megjelenítésért felelős Xserver nem feltétlenül ugyanazon a gépen foglal helyet! Ezért bármiféle rajzolási művelet megkezdése előtt a grafikus kapcsolatot fel kell építenünk, majd a munka végeztével le kell bontanunk. Az erre vonatkozó függvényeket a *graphics context* fejezetben találjuk a GDK leírásban. A legfontosabb függvények definíciói:

```
GdkGC* gdk_gc_new (GdkDrawable *drawable);
#define gdk_gc_destroy
```

Rajzoláshoz nyilván szeretnénk színeket használni, melyhez a `gdk_colormap` részben találjuk a megfelelő függvényeket. Ízelítő őül:

```
struct GdkColor;
struct GdkColormap;
GdkColormap* gdk_colormap_new (GdkVisual *visual, gboolean allocate);
gboolean gdk_colormap_alloc_color (GdkColormap *colormap, GdkColor
*color, gboolean writeable, gboolean best_match);
```

Innen már csak egy kis lépés, hogy valamit rajzoljunk a képernyőre:

```
void gdk_draw_point (GdkDrawable *drawable, GdkGC *gc, gint x, gint y);
void gdk_draw_line (GdkDrawable *drawable, GdkGC *gc, gint x1_, gint
y1_, gint x2_, gint y2_);
```

```
void gdk_draw_arc (GdkDrawable *drawable, GdkGC *gc, gboolean filled,
gint x, gint y, gint width, gint height, gint angle1, gint angle2);
```

Válthatunk grafikus kurzort is:

```
struct GdkCursor;
enum GdkCursorType;
GdkCursor* gdk_cursor_new (GdkCursorType cursor_type);
#define gdk_cursor_destroy
```

De a képernyőre íráshoz használt karakterkészleteket is a GDK-ból kezelhetjük:

```
struct GdkFont;
enum GdkFontType;
GdkFont* gdk_font_load (const gchar *font_name);
```

Valamint a szövegek pontos pozicionálását is segítik egyes függvények:

```
gint gdk_string_width (GdkFont *font, const gchar *string);
gint gdk_string_height (GdkFont *font, const gchar *string);
```

A GDK programkönyvtárból csak ízelítőül lehet ebben a terjedelemben megmutatni egyes részleteket, azonban iránymutatásként pont elegendő ahhoz, hogy tudjuk, merre keressük a munkánkhoz szükséges elemeket.

5. A GTK+

A GTK alapjai

A GTK egy olyan eszközkészlet, amellyel grafikus felhasználói felület hozható létre. Eredetileg a GIMP rajzolóprogramhoz lett kifejlesztve, innen is ered a neve: Gimp Tool Kit. Azóta jelentős fejlődésen ment keresztül, komplex, nagyméretű grafikus rendszerek építhetők belőle. A GTK-t használó programot nem csak C nyelven lehet írni, 10 fölött van azon programozási nyelvek száma, melyekhez készítettek kapcsolódó könyvtárat.

A programozás GTK környezetben

Egy alap GTK alkalmazás néhány nagyon egyszerű lépésből áll:

1. A GTK rendszer iniciálása
2. A widgetek létrehozása
3. A callback függvények kapcsolása a widgetekhez
4. A fő eseménykezelő hurokba belépés

Ezt a minimálprogramot az alábbiakban mutatjuk meg, azonban felhívjuk a figyelmet az alábbiakra: A kézzel történő GTK programozás tanulságos, ám fárasztó művelet, a sok egyforma, de könnyen elhibázható utasítás miatt. Egy program néhány (max. 5) ablakig és pár tíz widgetig tartható kézben korrektül, e felett igen könnyen összekuszálódnak a szálak, s nem az érdemi kódolásnál, hanem a felhasználói felület felépítésénél. Természetesen erre van megoldás, ezt a Glade fejezetben mutatjuk meg.

Az alap GTK program

```
gboolean
on_mainwin_delete_event (GtkWidget       *widget,
                         GdkEvent        *event,
                         gpointer         user_data)
{
  /* Saját kezelő rész, jelen esetben a kilépés*/
  gtk_main_quit ();
  return TRUE;
}


int
main (int argc, char **argv)
{
  /* Az i18n támogatás iniciálása */
  gtk_set_locale ();
  /* A GTK iniciálása */
  gtk_init (&argc, &argv);
  /* Nyissunk egy ablakot */
  mainwin = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  /* Callback függvény kapcsolása */
  g_signal_connect ((gpointer) mainwin, "delete_event",
                    G_CALLBACK (on_mainwin_delete_event),
                    NULL);
  /* Egyéb widgetek létrehozása */
  ...
  ...
  ...
  /* Az ablak megjelenítése */
  gtk_widget_show_all (mainwin);
  /* Belépés az eseménykezelőbe */
  gtk_main ();
  /* Kilépés */
  return 0;
}
```

Többszálas GTK programozás

Többszálas programok írásakor felmerül az igénye annak, hogy szálból (nem a főszálból) hívjunk GTK+ függvényeket. Amennyiben ezt az eddig megismert módon tesszük, hamarosan `Segmentation fault` hibaüzenettel találkozunk a képernyőn. Miért?
A GTK+ könyvtári függvények csak *thread-aware* és nem *thread-safe* módon kerültek megírásra. Ez azt jelenti, hogy önmagában nincs többszálas kezelésre biztosítva a könyvtár, azonban ennek figyelembevételével írták meg és így azzá tehető. A GLib és a GDK biztosítja számunkra azokat a függvényeket, melyekkel ezt megoldjuk.
A program legelején, a GTK+ bármely függvényének hívása előtt a `g_thread_init (NULL);` függvénnyel iniciáljuk a szálkezelést. Ezáltal definiálttá válik egy belső makró és iniciálódik egy belső használatú mutex. Ezután a `gdk_threads_init ();` függvénnyel mindezt közöljük a GDK-val is. Ezután a `gtk_init (&argc, &argv);` függvénnyel már iniciálhatjuk a GTK+ programunkat, megnyithatjuk a szálakat, ablakokat hozhatunk létre, mindent amit szeretnénk. Értelemszerűen a szálak aszinkron viselkedéséből adódóan *előbb* hozzuk létre a megfelelő GTK widgeteket és csak utána indítjuk útjára azokat a szálakat, melyek kezelik őket!
A `gtk_main()` főciklust a `gdk_threads_enter()/gdk_threads_leave()` függvénypáros közé zárjuk, melyek valójában az előbb említett belső használ-

latú mutexet zárják illetve nyitják. Ezután már csak annyi dolgunk van, hogy minden kritikus részt (gyakorlatilag GTK+ függvényhívás csoportot) a `gdk_threads_enter()/gdk_threads_leave()` függvénypáros közé zárunk. A főszál kivételével a `gdk_flush()` meghívása javasolt a `gdk_threads_leave()` előtt közvetlenül, hogy a függőben lévő parancsokat az X Window rendszer megkapja.

A minimál program

```
int
main (int argc, char *argv[])
{
  GtkWidget *window;

  g_thread_init (NULL);
  gdk_threads_init ();
  gtk_init (&argc, &argv);

  window = create_window ();
  gtk_widget_show (window);

  gdk_threads_enter ();
  gtk_main ();
  gdk_threads_leave ();

  return 0;
}
```

És mielőtt deadlock-ra gyanakodnánk, a `gtk_main()` magja:

```
if (g_main_loop_is_running (main_loops->data))
{
  GDK_THREADS_LEAVE ();
  g_main_loop_run (loop);
  GDK_THREADS_ENTER ();
  gdk_flush ();
}
```

6. A GNOME

A Gnome-ról első körben a szép felhasználói felület és desktop környezet ugrik be ha halljuk. Programozói szemmel nézve a Gnome ennél több.

A Gnome-ról röviden

A Gnome (többek közt) a következő könyvtárakat biztosítja a programozó számára:

- `libart`, grafikus megjelenítéshez ad plusz támogatást
- `libpanel-applet`, appletek létrehozását könnyíti meg
- `libBonobo` és `libBonoboui` CORBA szerverekhez biztosít átvitelt

- `GNOME-vfs`, virtuális fájlrendszert biztosít Gnome alkalmazások részére
- `libgnomecanvas`, komplex grafikus struktúrákat hozhatunk létre vele
- `libgnome` és `libgnomeui`, hasznos konfigurációs és felhasználói felület kezelő függvényeket biztosít a GTK+ könyvtárakon felül
- `libgnomeprint` és `libgnomeprintui`, részletezésüket lásd később

Ezeknek a könyvtáraknak a részletezése messze meghaladja ennek a cikknek a kereteit, mindenesetre jól mutatja, hogy milyen fejlődésen ment keresztül az elmúlt néhány évben a Gnome/GTK rendszer.

A Gnome könyvtári függvényei

Szemelvény jelleggel néhány Gnome könyvtárat tekintsünk meg közelebbről is.
A Gnome könyvtár egyik függvénycsoportjával nagyon hamar találkozhat a programozó munkája során, amikor konfigurációs fájlokat kell tárolnia a program működtetéséhez. Ezt a `gnome-config` támogatja. Szöveges formátumban, szabványos helyen csoportokba rendezett kulcs-adat párosok alapján tudunk nagyon könnyen konfigurációs fájlokat készíteni és használni. Csak bemutatóul néhány függvénye:
`gnome_config_get_string()` egy szöveges változót nyer ki a fájlból
`gnome_config_set_string()` egy szöveges változót helyez el a fájlban
`gnome_config_get_int()` `int` típusú adatot nyer ki
`gnome_config_get_float()` `float` típusú adatot nyer ki
`gnome_config_get_vector()` tömbbe szervezett adatokat is kezelhetünk
`gnome_config_sync ()` a fájlba írás az nem automatikus, lehetővé téve a változtatások elvetését is
Néhány komplex widget, melyeket a `libgnomeui` biztosít számunkra:
`GnomeApp` menüvel, eszközsorral ellátott komplex ablak
`GnomeAppBar` Státusz-sor és folyamat mutató
`GnomeAbout` Komplett szabványos About (Névjegy) ablak
`GnomeDruid` Telepítést megkönnyítő, több lépésen végigvezető kisalkalmazás
`GnomeColorPicker` Színválasztó ablak
`GnomeDateEdit` Naptár/óra widget
`GnomeIconSelection` Ikon választó kisalkalmazás

7. A gyors és hatékony programfejlesztés: GLADE

A GTK programozásba egy kicsit belemerülve (és ez operációs rendszertől igaz minden ablaktechnikás, grafikus programozásra) hamar ellep bennünket az az irdatlan mennyiségű "favágómunka", amely az egyes eszközök (widgetek) definiálásával, iniciálásával, eseménykezelőkkel való összekötésével foglalkozik. Érdemi kódolást még egy karaktert sem végeztünk, de már 1000 sort írtunk be csak azért, hogy megnyílhasson néhány ablak, melyben beviteli mezők, gombok, menük vannak. Nem lehetne ezt valahogy automatizálni? De igen! A GTK környezetben erről szól a GLADE.

A Glade működése

A Glade egy grafikus felülettel ellátott kódgenerátor. A programunkhoz szükséges grafikus elemeket (ablakok, widgetek, menük, stb. . .) interaktív módon összeállíthatjuk, majd a Glade a hozzátartozó kódot automatikusan generálja. A kód két fő részre osztható, az egyik a grafikus elemeket létrehozó függvények, a másik pedig a `callback` függvények, melyek az egyes grafikus elemekkel történő eseményekhez vannak kötve és a program érdemi részével történő kommunikációt biztosítják. Természetesen generálódik a `main`, illetve egy segédfüggvényeket tartalmazó fájl, továbbá értelemszerűen a megfelelő header-ek. Mindezek mellett a teljes projekt is létrejön a megnyitás alkalmával, azaz a make-fájloktól kezdve a többnyelvű támogatásig minden. A forrásfájl XML formátumú, UTF-8 kódolással támogatva.

A generált program felépítése

A generált program részei:

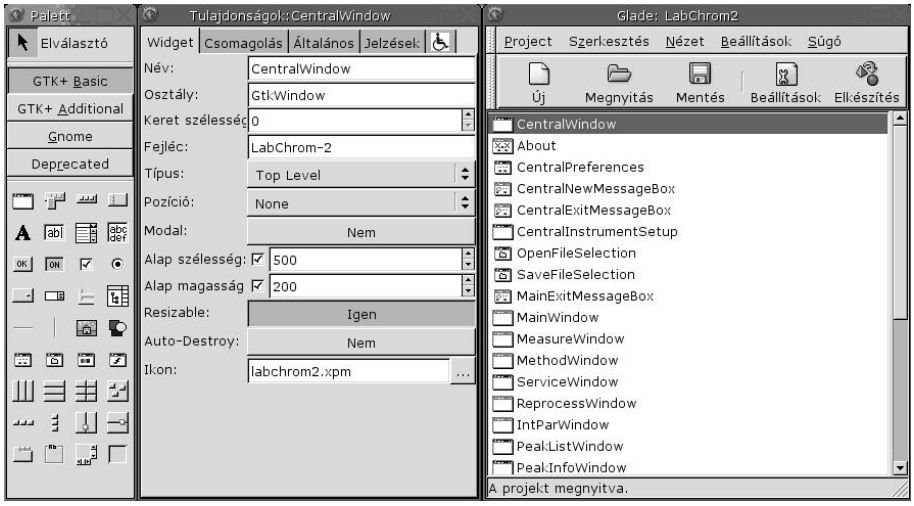
- A `main.c` fájl, melyet a továbbiakban nem módosít a Glade, szabadon átírható. A Gnome iniciálást, a fő hurkot (main loop) és a globális widget változókat tartalmazza.
- A `callbacks.c` és `callbacks.h` fájlok, melyekhez a továbbiakban a Glade csak hozzáfűzi az új függvényeket, a függvénytörzsek szabadon módosíthatók.
- Az `interface.c` és az `interface.h` fájlok, melyek minden esetben újragenerálódnak, átírni őket tilos és értelmetlen. Ezek a függvények hozzák létre a program grafikus vázát.
- A `support.c` és a `support.h` néhány segédfüggvényt biztosít a hatékonyabb kódoláshoz.

Hogyan használjuk?

A 1. ábra mutatja a képernyőt a Glade megnyitása és egy projekt betöltése után. Az éppen munka alatt lévő ablak most nem, de könnyen oda tudunk képzelni egy GTK ablakot, benne widgetekkel. A bal oldali ablakból tudjuk kiválasztani, hogy milyen ablakot szeretnénk, vagy milyen widgetet lerakni az ablakon belülre. A mellette lévő ablakban állíthatjuk be testre szabottan az egyes widgetek egyedi paramétereit, ha változtatni óhajtunk az alapértelmezett beállításokon. A jobboldali ablak a Glade fő ablaka, benne felsorolva a program által használt összes ablak. Ez a (fejlesztés alatt álló) program láthatóan körülbelül 30 ablakot tartalmaz, benne widgetek tömkelege. Jelen pillanatban a Glade által generált kód meghaladja a 20.000 sort, mely kizárólag a grafikus megjelenés felépítéséért felel, az érdemi kódolás ezen felül van. A program által nyújtott segítségről azt hiszem ezek az adatok mindent elmondanak. . .

Egy kódrészlet

```
/*
Egy toolbar és benne egy gomb létrehozása
*/

  CentralToolbar = gtk_toolbar_new ();
```



1. ábra. Glade – munka közben

```
  gtk_widget_show (CentralToolbar);
  gtk_box_pack_start (GTK_BOX (CentralVBox), CentralToolbar, FALSE,
FALSE, 0);
  gtk_toolbar_set_style (GTK_TOOLBAR (CentralToolbar),
GTK_TOOLBAR_BOTH);

  CentralNewButton = gtk_toolbar_insert_stock (GTK_TOOLBAR
(CentralToolbar),
                                "gtk-new",
                                "gtk-new",
                                NULL, NULL, NULL, -1);
  gtk_widget_show (CentralNewButton);

/*
A fenti gombhoz tartozó callback függvény.

A függvény törzse már az "érdemi" kódhoz tartozik,
de jól láthatóan a gomb megnyomása még mindig csak
egy másik ablak megnyitását végzi.
Mindenesetre a create_CentralNewMessageBox (); függvény
- melyet a Glade generált az interface.c fileban -
egy kis ablak 55 sornyi hibátlan kódját hívja!
*/

void
on_CentralNewButton_clicked            (GtkButton       *button,
                                        gpointer         user_data)
{
  CentralNewMessageBox = create_CentralNewMessageBox ();
  gtk_window_set_modal (GTK_WINDOW (CentralNewMessageBox), TRUE);
  gtk_window_set_transient_for (GTK_WINDOW (CentralNewMessageBox),
GTK_WINDOW (CentralWindow));
  gtk_widget_show (CentralNewMessageBox);
}
```

Tippek és trükkök

Néhány hasznos programozási tipp a Glade-del generált programokhoz:
Miután globális változót csak az ablakokhoz és a dialógusokhoz biztosít a Glade, problémát okoz, hogy a widgetek mutatóit megkapjuk, ha valamilyen változtatást óhajtunk végezni rajtuk. Ezen segít a `lookup_widget` függvény. Definíciója:
`GtkWidget* lookup_widget (GtkWidget *widget, const gchar *widget_name);`, ahol a `widget` a keresett widgetet befogadó *toplevel* widget, azaz pl. az ablak mutatója, a `widget_name` pedig a kívánt widget neve, melyet a Glade megfelelő ablakában állíthatunk be. A visszaadott érték a widgetre mutató pointer.
Ha van egy widgetünk, melynek mutatója ismert - például egy callback függvényben megkaptuk -, szükségünk lehet a toplevel widgetjére, azaz az ablakra, melyhez tartozik. Tipikus probléma, amikor egy adott ablakból több független példányt is megnyitunk, majd a `quit` gombbal szeretnénk bezárni, de a callback függvényben nem tudjuk eldönteni, hogy melyik példányt is szeretnénk bezárni az ablakok közül. A bezárást a `gtk_widget_destroy()` függvénnyel intézzük, de mi legyen az argumentum? A megoldás:
`gtk_widget_destroy (gtk_widget_get_toplevel (GTK_WIDGET(button)));` ahol a `button` a callback függvény által visszaadott mutató.
Egy másik gyakori probléma, hogy a megnyitott ablakban privát adatot tároljunk, amely csak ahhoz az ablakhoz tartozik, más, párhuzamosan nyitott ugyanolyanhoz nem. Erre a következő függvények adnak megoldást:
Létrehozunk egy dialógust:
`dialog = create_dialog1 ();`
A `privat_data` mutatót elhelyezzük benne: `gtk_object_set_data (GTK_OBJECT (dialog), "privat_data", privat_data);`
Majd amikor szükségünk van rá elővesszük:
`privat_data = gtk_object_get_data (GTK_OBJECT (dialog), "privat_data");`
Figyeljük meg, hogy ez kulcs-pár alapján történő adattárolás, név alapján azonosítjuk be a tárolt adatot. Itt említenénk meg egy makró párost, melynek segítségével egy 32 bites mutató helyén egy 32 bites `int` típusú számot lehet tárolni érték szerint:
`GINT_TO_POINTER()` és `GPOINTER_TO_INT()`
Felhívjuk rá a figyelmet, hogy ez csak egy tárolási trükk, másra nem használható!

8. A nyomtatás

Egy desktop szoftver, mely grafikus támogatással interaktív mérési adatgyűjtést és feldolgozást végez, elképzelhetetlen nyomtatás nélkül. A nyomtatás a Linux rendszereken egy "picit" másképp néz ki mint amivel más környezetben találkozhattunk. A UNIX alapú rendszereken a nyomtatás a PostScripten alapszik, melyet az alábbiakban részletesebben ismertetünk.
A programból egy PostScript (PS) állományt kell előállítanunk, melyet továbbítunk a nyomtatókezelő rendszernek, s innen a hatáskörünkből ki is került a dokumentum.

A PostScript

Ha megnézünk egy PostScript állományt, láthatjuk hogy valójában egy szöveges állományról van szó, ráadásul a felépítése is jól definiált és publikus. A baj csak a szokásos

vele, roppant körülményes előállítani egy PostScript fájlt ha csak arról van szó, hogy egy keretbe írjunk bele pár sor szöveget és rajzoljunk egy kördiagramot. Valószínű senkit sem lep meg, hogy erre a feladatra is létezik programkönyvtár.

A GnomePrint

A libgnomeprint 1.5 könyvtárral történő nyomtatást egy rövidke példaprogrammal illusztráljuk.
Ékezetes karakterek nyomtatásánál figyeljünk az *UTF-8* konvertálásra, ezt az `iconv()`-al tudjuk megtenni, különben a szöveget nem nyomtatja ki a rendszer.

A mintaprogram

```
extern void
SamplePrint (char *PrintFileName) {
GnomePrintContext *gpc;
GnomeFont *NormalTextFont;
char *PrintCommand;
int FontSetResult;

  PrintCommand = alloca (1000);
  gpc = gnome_print_context_new (gnome_printer_new_generic_ps
(PrintFileName));

  NormalTextFont = gnome_font_new ("Helvetica", 10);
  if (NormalTextFont == NULL) printf ("Font error\n");

  gnome_print_beginpage(gpc,"1");
  gnome_print_gsave (gpc);

  gnome_print_setlinewidth (gpc, 0.2);
  gnome_print_newpath (gpc);
  gnome_print_moveto(gpc, 50, 50);
  gnome_print_lineto(gpc, 60, 60);
  gnome_print_lineto(gpc, 55, 65);
  gnome_print_closepath (gpc);
  gnome_print_stroke (gpc);

  FontSetResult = gnome_print_setfont (gpc, NormalTextFont);
  gnome_print_moveto(gpc, 20, 20);
  gnome_print_show(gpc, "Hello world!");

  gnome_print_grestore (gpc);
  gnome_print_showpage (gpc);
  gnome_print_context_close (gpc);

  sprintf (PrintCommand, "lpr %s", PrintFileName);
  system (PrintCommand);
}
```

A program elemei

A mintaprogramból az látszik már első ránézésre is, hogy a képernyőre rajzoláshoz hasonlít a nyomtatás kezelése, egy (több) virtuális papírra rajzolunk, majd az eredményt átadjuk a nyomtatónak. Az egyes függvények rövid ismertetése:

- `gnome_print_context_new()` és `gnome_print_context_close()`
  Egy nyomtatási munkakapcsolat létrehozása illetve megszüntetése egy `GnomePrintContext` típusú változón keresztül.

- `gnome_font_new()`
  A munkához szükséges fontokhoz hozzuk létre a kapcsolatot.

- `gnome_print_beginpage()` és `gnome_print_showpage()`
  Miután a lapokat nem sorfolytonosan kezeljük mint pl. egy leporellós mátrixnyomtatót ezért minden egyes oldalt nekünk kell megkomponálnunk, ezeket a lapokat ágyazzuk be ezen függvények közé.

- `gnome_print_gsave()` és `gnome_print_grestore()`
  A `GnomePrintContext` aktuális beállításait tároljuk el veremben, illetve hívhatjuk vissza ezzel a függvénypárossal. Amikor egyes transzformációkat végzünk, vagy beállításokat módosítunk, azok visszaállítása néha nehézkes. Ezekkel az utasításokkal az összes beállítást egyszerre menthetjük. Hasonló mint a `PUSH/POP` az assembly nyelvekben.

- `gnome_print_setlinewidth()`
  Az aktuális rajzolási vonalvastagságot állítja be.

- `gnome_print_newpath()`, `gnome_print_closepath()` és `gnome_print_stroke()`
  Vonallal történő rajzoláshoz először meg kell nyitnunk egy új útvonalat, azt le kell a végén zárnunk, majd bevinni az oldalba.

- `gnome_print_moveto()`
  A virtuális tollat leteszi egy adott koordinátára. Rajzolásnál és szövegkiíratásnál egyaránt használjuk.

- `gnome_print_lineto()`
  A virtuális toll utolsó helyéről kiindulva vonalat húz az adott koordinátákra.

- `gnome_print_setfont()`
  Beállítja a kiíratáshoz (az előzőleg `gnome_font_new()`-al becsatolt) fontot.

- `gnome_print_show()`
  Kiíratjuk a virtuális toll pozícióján a szöveget.

A papír méretét, melyre rajzolunk a `gnome_paper_selector_get_height()` és a `gnome_paper_selector_get_width()` függvényekkel kérdezhetjük le. A függvények paraméterezésénél ügyeljünk arra, hogy az X és Y koordináták *nem integer* típusok hanem double, azaz ha szélességnek kapunk például 72-t, akkor lézernyomtatóval körülbelül 3 tizedes jegy pontossággal tudunk ehhez képest rajzolni a lapra, tehát vonalat húzhatunk 50.112 és 50.138 X koordináták közé. Ugyanez vonatkozik a vonalvastagságra is.
A library API a `http://developer.gnome.org/` címen található, a libgnomeprint2 API sajnos jelen pillanatban ennél is „aluldokumentáltabb” és folyamatos fejlesztés alatt van.

GNU/Linuxszal visszük a bankot! Miért és miként tért át a Konzumbank GNU/Linuxra?

Szabó László
Konzumbank RT.

2003.11.08.

Kivonat

A Konzumbank teljes ügyviteli rendszere GNU/Linux környezetben működik. Az előadás során bemutatjuk, hogy milyen folyamat eredményeként értük el ezt az állapotot. Értékeljük az eddigi folyamatot és felvázoljuk a jövő egy lehetséges képét.

1. A Linuxosítás lépései

A bizalom megszerzése (1999-2000)

Kezdetben az Internet környezetben vezettük be a GNU/Linuxot, ahol az Internet működése maga garantálta a sikeres bevezetést. Az itt szerzett kedvező tapasztalatok biztosították a továbblépés lehetőségét.

Kedvező környezet (2001-2002)

A banki ügyviteli rendszer szállítói szintén keresték a továbblépés lehetőségét, rendszerüket átültették GNU/Linuxra. Azóta már Solaris változatát is elkészítették. A menedzsmentet az eddigi kedvező működtetési tapasztalat meggyőzte, így tudott azonosulni a rendszerrel és vállalta a bevezetést. Központosított informatikai környezet kialakítása.

Töretlen lendülettel (2003-)

Fiókok fájl és nyomtató szervereinek cseréje Működési környezet egyszerűsítése miatt a fiókok fájlszervereit Samba fájlszerverre cseréljük.

Jelenleg 40 szervert üzemeltetünk GNU/Linux operációs rendszerrel.

2. A Linuxosítás irányelvei

Lépésről-lépésre

Az átállási kockázat csökkentés érdekében a linuxositást több kisebb szakaszra bontottuk. Az átállásokat alapos tesztelés és próbaüzem előzte meg.

Alkalmazások Linuxosítása

A Linux környezetben nem működő alkalmazások kiváltása, helyettesítése. Egy feladat megoldása előtt megnézzük van-e már megoldás rá az Interneten. Ragaszkodunk a szabványos megoldásokhoz, választásnál előnyben részesülnek a nyílt forrású, szabad licencelésű szoftverek. Intranet rendszert fejlesztettünk a saját megoldások keretbe foglalásához.

A felhasználók és menedzsment projekttel történő azonosítása

Fontos, hogy a menedzsment és a kollégák (650 fő) a változást követően saját területükön pozitív változásokat tapasztaljanak.

Milyen pozitívumokat hozott a banknak a GNU/Linux?

- Gyorsabb, megbízhatóbb működés
- Jelentős költség megtakarítás. Több tíz milliós megtakarítás érhető el a kisebb hardver igény, az operációs rendszer alkalmazói programok szabad licencelése által.
- A rendkívüli események száma harmadára csökkent, az üzletmenet folytonosság javult
- Homogén működési környezet, egyszerűbben – fele személyzettel – menedzselhető rendszer
- Szabványos, átlátható, dokumentált működés
- Gazdag funkcionalitás
- A menedzsment felismerte, hogy ár/teljesítmény, hatékonyság, és megbízhatóság szempontjából jó választás a GNU/Linux!

3. A Linuxosítás következő lépései

Munkaállomások Linuxosítása

- Az átállást akadályozó office alkalmazások kiváltása (excel makrók, stb.)
- 30 gépes környezetben 3-4 hónapos pilot üzem
- a munkaállomások 80%-nak (500) Linuxosítása

A Sun Java System

Cserép János
SUN

2003.11.08.

A vállalati szoftverek világának gyökeresen új megközelítése

A közelmúltban a Sun Microsystems tett néhány olyan bejelentést, melyek következtében mindörökre megváltozik az üzleti szoftver történelme.

Manapság mindenki a költségek és a komplexitás csökkentéséről, a mérhető befektetés visszatérülésről beszél. A Sun Java System az informatikai befektetések és az üzleti prioritások szinkronizálásának forradalmian új megközelítése. Cégünknek megvan a megfelelő tapasztalata és tejes körű portfoliója ahhoz, hogy ezt a stratégiát meg tudja valósítani.

Miért éppen a Java System? A Java technológia továbbra is a megosztott hálózati szolgáltatások forradalmának éllovasa, s felmerült az igény egy olyan optimalizált szoftverrendszer iránt, mely minden szinten maximálisan kihasználná a Java környezet nyújtotta előnyöket. A Java System a nyílt programozói felületek és ipari szabványok melletti elkötelezett és következetes stratégia eredménye, egy olyan új szoftverrendszercsalád, mely minden szinten (szoftver-architektúra, rendszerszintű funkciók, telepítés, felügyelet) maximális integrációt nyújt.

Újonnan meghirdetett szoftverrendszereinkkel egyszer s mindenkorra megváltoztattuk a szoftveripar kereskedelmi modelljét és dinamikáját. Egyszerű, kiszámítható és kedvező árú szoftvertermékeinkkel, valamint üzleti modellünkkel a költségeket és a bonyolultságot egyaránt csökkentjük.

A Java Enterprise System és a Java Desktop System az első termékek a magas fokon integrált, kedvező árú, együttműködő, teljes körű szoftverrendszerek sorában. Ezeket a szoftverrendszereket úgy terveztük, hogy minden környezettípusnak (az adatközpontoktól és asztali számítógépektől a mobil eszközökig és csipkártyákig) és minden felhasználótípusnak (fejlesztők, rendszergazdák, szolgáltatócégek és mobilszolgáltatók) megfeleljenek.

S hogy miért ez a legalkalmasabb időpont a Java Enterprise System bevezetésére? Mert az ügyfelek belefáradtak a különböző gyártóktól származó vállalati infrastruktúra szoftverek időrabló kiértékelésébe és beszerzésébe. Mert torkig vannak a kiszámíthatatlan élettartamú projektekkel és a rejtett megvalósítási költségekkel. Mert le akarják vetni magukról a keserves, büntetéssel is felérő licencek, a véletlenszerű kibocsátási ütemtervek és a végtelen számú verziók nyomon követésének terhét.

A Java Enterprise System lenyűgözően egyszerűvé teszi a vállalati infrastruktúra szoftvereket. A Sun valamennyi iparágvezető hálózati szolgáltatása, mint például a portálszolgáltatások, az azonnali üzenetküldés, az e-mailezés, a Liberty előírásainak

megfelelő hálózati személyazonosítás, a címtár és egyebek most egyetlen, harmonikusan működő szoftverrendszerben hozzáférhetők, melyet szabályos, negyedéves időközökben szállítunk.

S ami még elképesztőbb, mindez csupán 100 dollárba kerül évenként és alkalmazottanként, a cég alkalmazottainak számát véve alapul. Miről is van tehát szó? Arról, hogy a szoftverlicencek vásárlásakor és ellenőrzésekor rengeteg pénzt takarít meg. Nem lesz többé szüksége alkalmazottak népes seregére, hogy az egyes szolgáltatásokat integrálják. Ezt ugyanis mi már megtettük. S a megvalósítás is gördülékeny: egyetlen telepítő és negyedéves automatikus frissítés – ennyi az egész.

A Java Desktop System, kedvező árú asztali környezet, rendkívül stabil alternatíva, sokkal biztonságosabb, hiszen támogatja a nemzetvédelmi hivatalban használt kategóriának megfelelő Java Card technológiát. Egyszerű, könnyen kezelhető kliensfelületével, s az asztaltól elvárható valamennyi szokásos funkciójával tökéletes kiegészítője a Java Enterprise Systemnek – csak a vírusok és a magas árat mutató árcédula hiányzik róla. A Java Desktop System a mobilitást és biztonságot egyesítő átfogóbb stratégiánk részét képezi, biztonságos módon és elérhető áron továbbítva a hálózat erejét mindazokhoz, akiknek erre szükségük van.

A Java System különösebb bonyodalmak nélkül mindörökre megváltoztatja majd a szoftverbeszerzést, -fejlesztést és -kezelést. A Java Systemmel felpörgetheti új hálózati szolgáltatásait és működéskritikus üzleti alkalmazásait, gyorsabbá, egyszerűbbé és minden eddiginél alacsonyabb költségűvé téve őket. Figyelmét és erőforrásait így teljes mértékben az innovációra, a versenyre és a nyereségességre fordíthatja.

Nyílt, új világ: a Linux lehetőségei a kormányzati informatikában

Köntös Zoltán
IBM Hungary

2003.11.08.

A társadalmi, gazdasági formáció változásával az állam szerepe, funkciója is jelentősen változik; fokozatosan erősödik az állam szolgáltató jellege. A közigazgatás a társadalom integráns része, amely funkcióiban tükrözi azokat a célokat, amelyeket az adott társadalom elvár. A közigazgatás különböző szintjein megjelenik az állampolgár részvételi igénye is, hiszen az intézkedések, jogi szabályozások a társadalomért, az emberért történnek. A jelenleg kialakítás alatt lévő közigazgatási modell megvalósítását, az Európai Unióhoz való csatlakozás igényli, hogy a közigazgatás eszközrendszere is igazodjon a megváltozott követelményekhez.

A közigazgatás különböző szintjei, mint a központi államigazgatás vagy az önkormányzati igazgatás eltérő módon hatnak a társadalmi célok megvalósítására: a kormányzás a célok kitűzése, a feltételek megteremtése szintjén, míg az önkormányzatok a megvalósítás szintjén fejtik ki tevékenységüket. Mivel a közigazgatás döntései és intézkedései, valamint annak hatásai átszövik a mindennapi életet, a cél egy olyan gyakorlat kialakítása, amelyben az állampolgárok a közigazgatás működését értük valónak ismerik el.

Többszörösen új kihívás a közigazgatás számára az informatika alkalmazása, amely alapvetően megváltoztatja a hagyományos munkamódszereket. A közigazgatás jellegénél fogva konzervatív, vagyis a stabil, kiszámítható folyamatokat részesíti előnyben, míg az informatika nap, mint nap újítja meg technológiáját és eszközrendszerét. Manapság az állampolgárok ügyeik intézése során egyre gyakrabban találkoznak az informatikai fejlesztések eredményeivel. Az informatizálás azonban eddig többnyire nem jelentett látványos minőségi változást a szolgáltatások színvonalában, a közigazgatás hatékonysága az utca embere számára csak lassan javul. A technológia óriási léptékű fejlődése viszont az egész világon arra készteti a kormányzatokat, hogy csökkenő költségvetési források mellett és a növekvő elvárásoknak megfelelve gyökeresen megváltoztassák a társadalomnak nyújtott szolgáltatásaik módozatait. Az úgynevezett elektronikus kormányzat amellett, hogy az ügyfelek jóval magasabb színvonalú kiszolgálását biztosítja, a kormányzati munka hatékonyságának alapvető javulását is eredményezi. Ez az új kormányzati szolgáltatási modell függetlenül attól, hogy az „egyablakos kormányzat”, a „mindent egy helyen” megközelítés vagy a „megosztott hozzáférés” modelljének nevét viseli, azonos tulajdonságokkal jellemezhető. Cél, hogy az ügyfelek akkor és azon a helyszínen férhessenek hozzá az őket érdeklő szolgáltatásokhoz, ahogy kívánják, s mindezt a legegyszerűbb módon, költségtakarékos megoldásban.

A nyílt forráskódú, magyar nyelvű, az elektronikus kormányzati szférában is felhasználható Linux alapú alkalmazásplatform kidolgozása a következő előnyökkel jár:

- Csökken az általános bekerülési és üzemeltetési költség az informatikai rendszerekbe
- Megelőzhető lehet egyes szoftverszállítók monopolhelyzetbe kerülése
- A kormányzati, állami és költségvetési intézmények hardver parkjának (munkaállomások, szerverek) drasztikus megújítása nélkül is jól használható

Egy nyílt forráskódú, teljesen magyar nyelvű Linux disztribúció kormányzati szintű fejlesztése és támogatása során szóba jöhet egy magyarított külföldi disztribúció, vagy egy alapoktól kezdve a magyar államigazgatás igényeihez fejlesztett magyar változat is. Az előbbiek magasabb fejlettségi szintje, míg utóbbi fejlesztési irányának kézben tarthatósága, illetve teljesen magyar nyelvű támogatása jelent előnyt. A preferált eszközök kiválasztása után szükséges az ehhez kapcsolódó szakkönyvek, oktatási segédanyagok és egységes felhasználói tanfolyam-, valamint vizsgarendszer kidolgozása is. A nyílt forráskódú fejlesztések nagy előnye a nyílt szabványok támogatása, illetve következménye az is, hogy az elkészült alkalmazások jóval rugalmasabban használhatók, mint a zárt forráskódú termékek. A főbb tervezési szempontok a következők:

- Az Internet előnyeinek kihasználása, azaz központosított adatbázisok létrehozása, melyet távolról, mindenki számára elérhetővé tesz a rendszer, nagyfokú integrációt és adatkoncentrációt biztosítva
- Nagy megbízhatóságú alkalmazások alapjául szolgáljon, melyek célja az, hogy minimális időkiesés mellett a rendszerek a nap 24 órájában az állampolgárok rendelkezésére álljanak
- Az üzemben tartási költségek minimalizálása, melynek célja az, hogy a rendszer hardverigénye minimális legyen, magas funkcionalitást biztosítson szerényebb konfigurációjú (PI, PII processzorok) gépeken is, illetve, hogy az egyes fejlettebb verziók mindvégig kompatibilisek legyenek az előző verzióval
- Biztonságos tervezés, azaz a rendszer alkalmas legyen arra, hogy nagy biztonsággal üzemeltethető legyen, adatvédelmi rendszere nehézzé tegye az illetéktelenek hozzáférését, de illetékesek számára ne csak az Internet, hanem mobil eszközökön keresztül is elérhető legyen tartalma.

A kormányzati stratégiához illeszthetően a felhasználóknak, intézményeknek átadott, használatra előkészített személyi számítógépeket a kifejlesztett, előre installált alkalmazásplatformmal értékesítsenek. Ez a lépés nagymértékben csökkentené a hazai, az EU átlagot jóval meghaladó illegális szoftver használatot. Egy informatikai beruházásban a bekerülési ár csak töredéke a hosszú távú üzemeltetési költségnek, mégsem elhanyagolható, hogy a szabad szoftvert nem kell megvásárolni. Nagy mennyiségben a szoftver licenc komoly kiadásnak számít, főleg a géppark bővítésnél és az upgrade esetében. Ehhez hozzájárul az is, hogy a kereskedelmi terméket gyártó cégek gyakran a saját munkájuk megkönnyítése érdekében az új verziók megjelenése után nem sokkal megszüntetik a támogatást a régebbi verziókra, ezzel is kényszerítve a felhasználót a fejlettebb verziók megvásárlására. A kereskedelmi szoftverek árpolitikájára általában jellemző, hogy „ami olcsó azt biztos nagy mennyiségben kell megvásárolni (minden

gépre), amiből pedig nem kell sok (pl. szerveralkalmazások), annak ára a kliensek számától függ". Ezt tovább súlyosbíthatja az is, ha a licencköltséget nem lehet időben elosztani (pl. lízing), hanem egy összegben kell kifizetni.

A fentebb említett rendszer kidolgozása után javasoljuk forráshiányos, informatikailag kritikus helyzetben lévő területeken bevezetését elindítani, azzal a céllal, hogy megfelelő gyakorlati tapasztalatot nyújtsanak a szabad szoftverek felhasználásának előnyeiről és nehézségeiről. A fentieket első körben olyan, izolált intézményekben érdemes megvalósítani, ahol az informatikai rendszerek integráltsági foka viszonylag alacsony. Ennek tipikusan megfelelő területek az oktatási intézmények és önkormányzatok és az egészségügy.

Az IBM Magyarországi Kft. stratégiai tervei között szerepel egy jól használható rendszer kiépítése, mely egyszerre képes a magas fokú biztonságra, és a felhasználóbarát működésre. Fontos, hogy a jelenleg használt felhasználói szoftvereket teljes körűen lehet használni, azaz nincs szükség arra, hogy lecseréljük a már meglévőket. A rendszer elfogadását nagyban segíti, hogy a felhasználói felület kialakítása olyan, hogy nem szükséges betanulás, hiszen a szimbólumrendszere teljesen megegyezik a jelenlegi desktop operációs rendszerekkel.

A javasolt kormányzati Linux stratégia lényege, hogy a Linux operációs rendszer, az azt támogató hardver és szoftver elemekkel és alkalmazásokkal, váljon egyenrangú alternatívává a ma elterjedt Windows és Unix operációs rendszerek mellett. A stratégia rá kell mutasson a Linux által elérhető előnyökre, és támogatnia kell a Linux terjedését annak érdekében, hogy ezek az előnyök valóban minél gyorsabban termőre forduljanak.