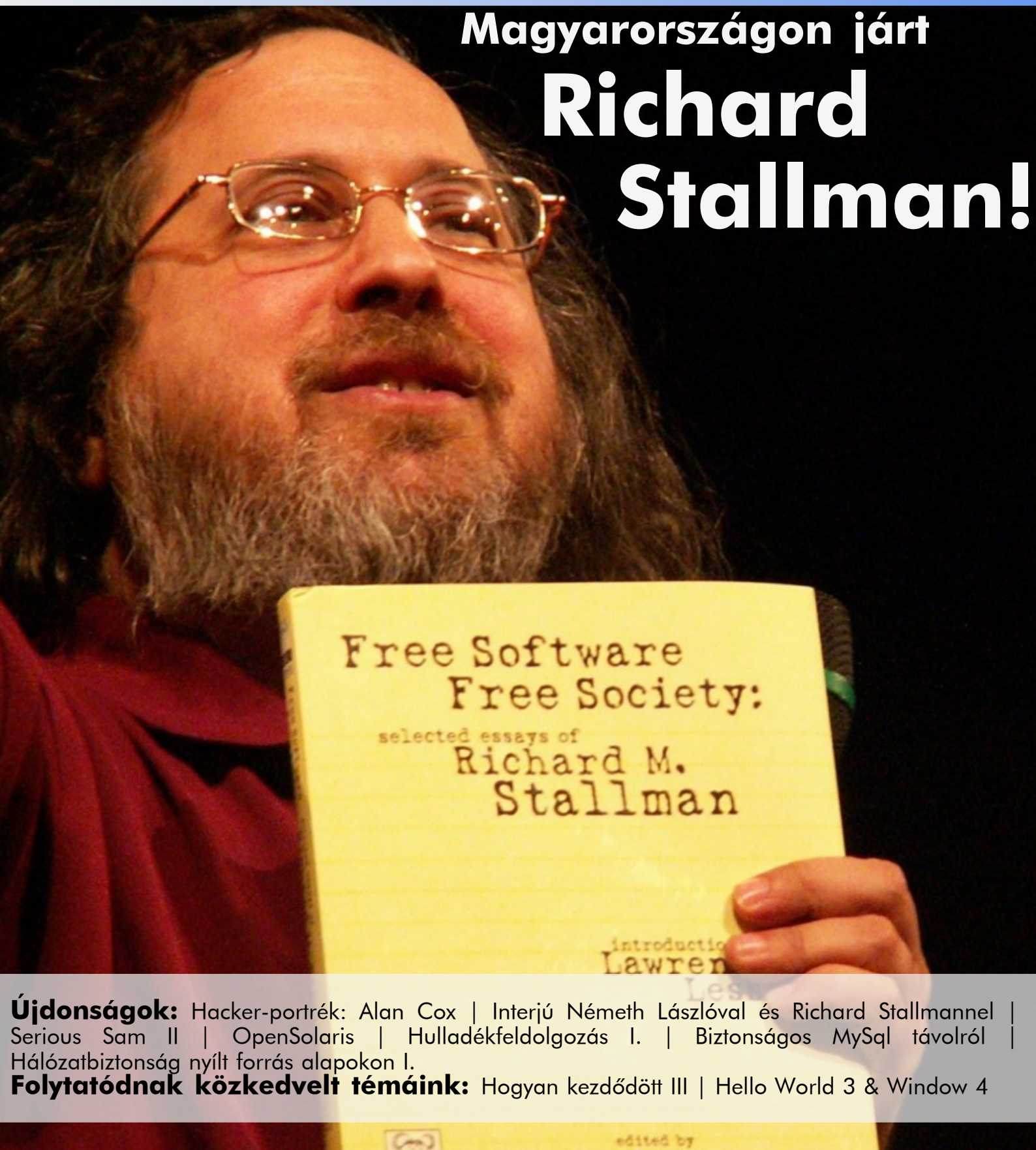
FLOSS ZINE

Free / Libre / Open Source Software fanzine

2009. március

II. évfolyam 1. szám

Magyarországon járt Richard Stallman!



Újdonságok: Hacker-portrék: Alan Cox | Interjú Németh Lászlóval és Richard Stallmannel | Serious Sam II | OpenSolaris | Hulladékfeldolgozás I. | Biztonságos MySql távolról | Hálózatbiztonság nyílt forrás alapokon I.

Folytatódnak közkedvelt témáink: Hogyan kezdődött III | Hello World 3 & Window 4

Tartalom

Lite

Pro

A kalapos kernelkirály - Alan Cox

Hackerportrék 3

2000. január. 1-én csatlakozott a Red Hat-hoz, és 2009. január közepén vált ki a cégből - azért, hogy a tavaly egyre csak csökkenő nyereséget produkáló Intelnél folytassa pályafutását. Alan Cox leginkább a 2.2-es, majd a 2.4-es kernel fejlesztésében és karbantartásában elért eredményei miatt vált a legnevesebb hackerek egyikévé. Korábban egyebek mellett játékokat portolt és fejlesztett, C++-os hálózatot épített, ISDN-kódot írt, és rendszer-adminisztrátorként tevékenykedett.

Alan Cox a FOSS 2007-en (Wikimedia Commons) Creative Commons Attribution ShareAlike 2.5

"Élj gyorsan, halj meg öregen, és közben légy biztos benne, hogy mindenki tudja, itt jártál". - Alan Cox mottója valóban sokat mond életviteléről. A Sonix, a 3COM, az Adventure Soft, az NTL, és a Cymru.Net nevű, kisebb internetszolgáltató aknázta ki képességeit és munkabírását, mielőtt a Wales-i programozó a Red Hat-hoz került. Főállásban ott kezdett el a Linuxszal foglalkozni, ám két évvel később egy interjúban már arra figyelmeztette a nyílt forráskódú alkalmazások fejlesztőit, hogy ebben a szakmában könnyen közömbössé válhat az ember, ha a korábbi hobbija egyszer csak kenyérkereső foglalkozásává válik.

Cox a Red Hattól való távozását 2008. december 23-án tette közzé, azonban már 2007 nyarán úgy nyilatkozott egy cseh honlapnak az Intelről, hogy a pro-cesszorgyártók közül az működik együtt a legjobban a nyílt rendszerek fejlesztésében, úgy, hogy átad jó dokumentációkat, hibainformációkat, sőt, a cég olyan meghajtók fejlesztésében is partner, amelyek a hardvereihez 3D-támogatást, vagy wifi támo-gatást biztosítanak. Cox az interjúban azt is elmondta: az Intel arra törekszik, hogy chipjeinek minden képességét ki lehessen használni, hogy minden bitet proceszszorainak gyorsítására lehessen fordítani Linux alatt is. Ezáltal pedig szerinte az Intel az AMD-től és a Via-tól vehet el piaci szeleteket. Mindehhez képest Cox idei januári munkahelyváltása már nem is annyira meglepő, bár a közösség számára nyilvánosságra hozott, néhány soros indoklásában minderre már nem tér ki:

"Január közepén elhagyom a Red Hat-ot... Nem a családdal, a kertészkedéssel, vagy más csodás dolgokkal fogok több időt tölteni. Elmegyek, a Red Hatnál töltött jó időszakok és a cég határozott támogatása ellenére. Tíz évet töltöttem a Red Hatnál beszállítóként, majd alkalmazottként és most lehetőségem nyílt a szorosabban vett alapszintű fejlesztéseken dolgozni, ami jobban érdekel."

Cox annak ellenére döntött így, hogy az Intel Corporation ez év január elején bejelentette, hogy 2008 negyedik negyedévében 90 százalékkal csökkent a nyeresége az előző

negyedévihez képest, ami 23 százalékos árbevétel-csökkenést jelent 2007 utolsó negyedévi adataival összevetve. A cég erre 200 millió dollárral mérsékelte működési kiadásait, ennek része, hogy öt gyárát bezárta, 5-6 ezer munkahelyet megszüntet, vagy legalább is átszervez. Így az Intel világszerte foglalkoztatott 80 ezer alkalmazottjából a leépítés akár 5-6 százalékot is érinthet. Hogy miért is lehet fontos a legnagyobb chipgyártónak Alan Cox személye, arra a válasz valószínűleg az Intel stratégiaváltásában keresendő. A multi ezután egyértelműen a netbook- és a mobil internet-eszközök architektúrájára kíván koncentrálni, ennek érdekében indította el már 2007-ben a Moblin projektet, amelynek keretein belül nyílt forrású szoftverek fejlesztése folyik mobil eszközök számára. Mindezt összegezve megállapítható, hogy az utolsó lökést a váltáshoz az Intelnek ugyanúgy, ahogy Alan Coxnak is, a világgazdasági válság tavaly őszi eszkalálódása adta.

Cox ugyanis 23 éves kora, vagyis 1991 óta vesz részt a Linux-kernel fejlesztésében. Az elsők közé tartozott, aki éles hálózatra Linux-rendszert telepített, ez a Wales-i Swansea Egyetemen történt, amelynek hallgatója volt akkor. Miután sok hibát talált a hálózati alrendszer kódjaiban, sokat átírt közülük, és hibajavításai hamar az újabb stabil kernel részeivé váltak. A probléma ekkor még az volt, hogy a hálózatok mindenféle védelem nélkül voltak összekötve és az internetre kapcsolt szerverek minden egyes, nekik címzett csomagot megkaptak és feldolgoztak. Ezen előbb a routerek fejlesztése segített, majd Alan Cox, aki az első csomagszűrőt (pf) portolta az 1.3-as kernel sorozatba. Később ehhez kapcsolódott a kernel-rezidens hálózati interfészek konfigurálására való ifconfig kifejlesztése, amelyben Cox is részt vett.

A 2.2-es Linux-rendszermagnak már ő a karbantartója, majd ennek stabil kiadását követően létrehozza a 2.4-es kernelfa saját ágát, a különösen stabil és biztonságos -ac sorozatot (pl. 2.4.13-ac1), amelyen ezért sok szerver disztribúció alapult. Cox 2003-ban átvehette a legnagyobb hackereknek járó FSF Award díjat. Utána egy év szabadságra ment a Red Hattól, hogy újabb diplomát (MBA) szerezzen.

Cox ugyan jó kapcsolatban van Linus Torvalds-szal, ám a kernelfejlesztésről merőben eltérő véleményt vall: "Linus jó fejlesztő, de borzalmas mérnök. Biztos vagyok benne, hogy ebben ő is igazat ad nekem." - mondta Cox 2005-ben, és arra utalt, hogy míg ő maga a stabilitást tartja mindig szem előtt, addig Linus imádja a kódot "hackelni", vagyis számára a kernel könnyű kezelhetősége és a javítás egyszerűsége a fontos.

Jankovich Oszkár

Jegyzet:

1 A Linux-hacker Alan Cox nem tévesztendő össze a hasonló nevű hustoni professzorral, aki egyebek mellett a FreeBSD projektben is dolgozik. Az utóbbi időnként Alan L. Cox néven szerepel a forrásokban, sokszor azonban szintén csak Alan Coxnak írják a nevét.

További felhasznált források:

http://www.redhat.com/advice/ask_alancox.html

http://interviews.slashdot.org/interviews/02/05/20/1314214.shtml?tid=166

http://www.linuxjournal.com/article/5045

http://kerneltrap.org/node/9

http://www.linuxjournal.com/article/217

http://news.cnet.com/8301-13505_3-9803919-16.html

http://www.linux-magazin.de/news/video_alan_cox_ueber_proprietaere_treiber_tainted_kernel_und_gplv3?category=0

ftp://zeniv.linux.org.uk/pub/linux/alan/Talks/OGG/

http://www.linux.org.uk/~telsa/index.html

http://www.linux.org.uk/Articles.cs

http://www.softpanorama.org/People/Cox/index.shtml#Introduction

http://www.factbites.com/topics/Alan-Cox

Hogyan kezdődött III.

**Sikerült jól elnyújtani a rétestésztát, de a múltkori számba, (bár készen volt) nem kerülhetett bele minden, mivel nagyon félrehúzta volna a lapszámot a túlzásba vitt história.
Hol is tartottunk? Valahol itt.**

A BSD története során említésre került a Linux. Mielőtt részletesebben taglalnánk a témakört, meg kell említeni egy korábban kezdődött kezdeményezést, de a végén úgyis minden bekerül a levesbe.

GNU

Mi a GNU? Egy biztos, nem Unix! Ezt még maga Richard Stallman jelentette ki, és egyetlen afrikai névadó sem szólalt fel ez ellen, miszerint bármelyikük is Unix lenne. "GNU's Not Unix."

Ki ez a szimpatikus fiatalember, Richard Matthew Stallman (RMS)? (Nem keverendő össze egy másik RMS-sel, ami a Royal Mail Ship rövidítése, vagyis királyi postahajó. Az egyik leghíresebb Royal Mail Ship egyébként a keveset futott Titanic volt).
A mi RMS-ünk szoftverfejlesztőként kezdte pályafutását, kicsit nem figyelt oda, és a nyílt forrású rendszerek egyik ikonja lett (akár a szó szoros értelmében is).

Ehhez persze kellett az is, hogy tele legyen a szakálla a MIT-ben végzett egyébként remek munkájával. Legjobb lesz, ha erről inkább Ő maga vall:

„Miért KELL megírnom a GNU-t?

Végiggondolva, az aranyszabály azt kívánja, hogyha én szeretek egy programot, akkor azt másokkal is meg kell osztanom, akik

szintén szeretik. A szoftvereladók meg akarják osztani a felhasználókat, majd „leigázni" őket, azt akarják, hogy a felhasználók ne osszák meg a programokat másokkal. Ezzel nem tudok egyetérteni. Nem tudok tiszta lelkiismerettel egy nem nyilvános, vagy szoftver szerződést aláírni.
Évekig dolgoztam az AI laborban, tűrtem ilyen dolgokat, és más „gonoszságokat", de végül túl messzire mentek: nem maradhat-

tam egy olyan intézményben, ahol ilyen dolgokat csináltak értem, az akaratom ellenére.
Azért, hogy a számítógépeket minden szégyenkezés nélkül tovább használhassam, elhatároztam, hogy összegyűjtök egy olyan szabad szoftvercsomagot, ami képes lesz minden nem szabad szoftver nélkül létezni. Elmentem hát az AI lab.-tól, hogy a MIT ne tudja megakadályozni, hogy a GNU-t közreadhassam."

Nos ezért hozta létre a GNU projektet Richard Matthew Stallman. A GNU egy teljes, Unix-szerű (de nem Unix!) rendszer lenne, ha elkészülne, és teljesen szabad, a szó szoftverekre vonatkoztatható minden értelmében (itt még bejátszik a free szó ángliusok által használt jelentése is, az ingyenes).
Azaz a teljes rendszer (forráskóddal együtt) ingyen, szabadon hozzáférhető, módosítható,

terjeszthető, használható bármilyen célra.
A projektet 1983-ban jelentette be RMS. (Másutt 1984-et is írnak).
Szép elképzelés, de még tenni kellett valamit, hogy ez így is maradhasson.
Azért, hogy ne lehessen a kódot bezárni, és fizetőssé változtatni a rendszert, kellett egy licensz, ami mindezt megakadályozza, néhány korlátozás bevezetésével.
Ez lett a GPL, a „GNU General Public License". A GPL a legelső, és egyben a legelterjedtebb valóban szabad licensz. Talán nem tévedünk nagyot, ha azt állítjuk, hogy a szabad szoftver mozgalomhoz szorosan kapcsolódó GPL alapozta meg ezen programok, rendszerek sikerét.

A licensz lényege – más szabad licenszekhez hasonlóan –, hogy a mű szabadon terjeszthető (akár pénzért is), és szabadon módosítható, de a terjesztései, és a módosítások kötelezően szintén GPL licensz alatt kell, hogy megjelenjenek, így biztosítva, hogy a szabad tartalmakból készült bármilyen származékos mű is szabad maradjon.

Stallman 1985-ben létrehozta a Free Software Foundation-t (FSF), a Szabad Szoftver Alapítványt, hogy ezzel is támogassa elképzeléseit, és a GNU rendszer fejlesztését.
Egyébként Ő találta ki a copyleft kifejezést is.
A projekt szépen haladt előre, a GCC, és a GNU Emacs, az elsők között készültek el. 1987-ben már megvolt a Bash shell, 1991-re pedig elkészültek az alapvető fontosságú programok, de még mindig hiányzott valami, és ez ma sincs másképpen.
A GNU rendszerből, a mai napig hiányzik a közepe, a kernel. A rendszer nagyon sok része megvan, és használják is a programokat világszerte, de GNU kernel nem létezik. Talán egyszer elkészül, de nem biztos, hogy nagyot fog változtatni a jelenlegi helyzeten.
A kernel hiányának ellenére a szabad szoftvereket használók legnagyobb része használja a GNU-t.
Hogyan lehetséges ez? Mindjárt meglátjuk, de előbb nézegessük kicsit a ZZ-TOP szökevény fejlesztőket. Kernigham, Ritchie, Stallman (és majd látjuk később, mások is) simán felléphetnének bármelyik rockfesztiválon (ma már ez akkor is lehetséges, ha nem tudnak se zenélni, sem énekelni).
Az említett illetők, és a ZZ-TOP-os suhancok között több hasonlóság is fellelhető. Azon túl, hogy a nemzetközi szépségversenyeken indulva meg kellene elégedniük a vigaszdíjakkal (és ezen még a világbékéről való hosszanti hadoválás sem segíthetne), a

szabadságvágy és a szabadságszeretet minden Open Source fejlesztőben megtalálható valahol, akárcsak a hitelesebb rockbandák tagjaiban. Ha nem így lenne, nem lenne (vagy nem pont ilyen lenne) az open source mozgalom (helyesebben szabad szoftver mozgalom, ahogy RMS kijavítana minket), hiszen tudásukkal nyugodtan reszelgethetnék a kereskedelmi programokat jó pénzért, vagy gründolhattak volna maguknak egy jó kis fejőstehén céget. Ők ehelyett „idealista baromságokkal" töltik az idejüket, mint a kiadói divatoknak ellenálló örömzenészek. (Persze ez sem igaz, hiszen Marconi óta tudjuk, semmi gond sem lett volna, ha a jó öreg állandóan csak pecázással töltötte volna az időt, majdcsak feltalálta volna valaki azt a tetves rádiót. Mint ahogy fel is találta, talán még előtte :)). Volnából egyelőre ennyi elég, menjünk tovább.

Cáfoljuk meg rögtön a bedrogozott rockzenész-elszállt szoftverfejlesztő párhuzamot, jöjjön a simaképű, halszagú, rokongyerek, egy finn egyetemista, (aki persze mégsem olyan közeli rokon, hiszen svéd).

Linux

Már megint egy búbánatos x végű szókreálmány. Mi a Linux? Nos bármi is, de nem egy operációs rendszer. Akkor hogy a jófenében van az, hogy minden csapból mégis ez folyik mostanában, és boldog boldogtalan használja, sőt meg van vele elégedve (persze a szoftverkufárok kivételével).
A dolgok végül a helyükre kerülnek. (Nemsokára meglátjuk hogyan).
A konklúzióval kezdve, adott egy remek, de önmagában használhatatlan (miért is, ohne kernel) programegyüttes (persze a már meglévő nyílt rendszereken Unix, BSD használták őket), és egy valóban kiváló, de önmagában használhatatlan kernel.
Nos mindez, nem utolsósorban a híres ortodox pátriárka, RMS és Linus (igen, az elírt nevű svéd kernel) közreműködésével végül egységes egésszé állt össze. Ezért használunk GNU/Linux-ot, ahogy Stallman rámutatott nemegyszer.

Még egy fárasztó elmeroham, ugye, ha a BSD Unix, de a Unix nem BSD, akkor az is elmondható hogy a Linux Minix, de a Minix nem Linux. Persze ez csak az elején, egy darabig volt igaz.
Na még egyet, a fejlesztő egészen biztosan felhasználó, de a felhasználó egyáltalán nem biztos, hogy fejlesztő (hehh). Kis pihenő után folytatom.

Tehát van egy Linus Torvalds nevű egyetemista valahol Finnországban, aki még a delphoi jósnő írásos jövőtanulmányának sem hinné el, mivé válik néhány év alatt, aprócska kezdeményezése.
A lehetőségek hazája Finnország. (Az öt százalék svéd kisebbségnek mindenképpen, még az ivásra is több lehetőségük van).
Torvalds sem tudja a Linux pontos születésnapját, mivel nem is annak indult, ami lett belőle.
A neve sem volt meg a kezdetektől.

A Linux valahol a Minix-szel kezdődött. A Minix egy oktatási céllal készült operációs rendszer volt a PC kategóriájú számítógépekre. Alkotója Dr. Andrew Stuart "Andy" Tanenbaum professzor, a későbbi atyai jó barát, vagy inkább házastárs (már ami a később lezajlott veszekedéseket illeti), (Kár, hogy nem Tannenbaumnak írja a nevét, hívhattam volna fenyőfa professzornak :(). (Persze lehet így is fenyőfát jelent a neve valamilyen nyelven. Fenyőfa kép nem lesz).

Linus akkoriban a Helsinki egyetem hallga-

tója volt. Szerette volna az Intel 80386-os processzorának lehetőségeit tanulmányozni. Egy Minix-es listára küldött leveléből kiderült, hogy 1991 áprilisától egy ingyenes operációs rendszert farigcsált, és számított arra, hogy néhány éven belül megjelenik a GNU/Hurd.
Mindenesetre Linus a rendszer fejlesztését Minixen kezdte, de hamarosan rájött, hogy még talán akkor is jobban jár, ha saját rendszert ír, és így ismeri meg a 386-os processzor lehetőségeit.
Ekkor még Freax-nak hívta kreálmányát (a freak, free és unix szavakra gondolva a névadáskor).
Felmerült a Linux név is, de nem akart arcoskodásból strigulát, valamelyik kereskedelmi tévécsatorna nagyon élő adásában, így akkor még letett erről. Később, mikor ftp-re került a dolog, hagyta magát meggyőzni a Linux név előnyeiről a másikkal szemben, de azóta is hangoztatja, hogy nem Ő a hibás, csak a rettenetes nyomásnak engedett, mondhatni eleget tett a tömegek követelésének (a tömeget Ari Lemke-nek hívták). (Na jó, szó szerint nem pontosan ezt mondja).
A Linux kernelről 1991 augusztus 25-én érkezett bejelentés a Minix levelező listájára.
A legelső, 0.01 változat szeptember közepén került fel az FTP szerverekre.
Október 5-én megjelent a 0.02, és egy újabb levél a Minix listára.
Mivel az egész történet szépen részletezve nagyon sok helyen megtalálható, így csak nagy vonalakban vesszük át a folyamatot.
Már nem sokat váratott magára az elhíresült vitakör. 1992 január végén írta Tanenbaum Linusnak, hogy a monolitikus kernellel kapcsolatos véleményét fenntartja.
Linus elég lazán válaszolt rá, és nem jutottak közös nevezőre.
A fejlesztés tovább folyt, és a 0.12-es verziónál Torvalds bejelentette, hogy megváltoztatja a rendszer licenszét.
A 0.99-es verzió már teljes egészében GPL alatt jelent meg, 1992 decemberében.
Ekkoriban kezdődött meg a GNU projekt és a Linux fejlesztői közötti együttműködés, és az ún. Linux disztribúciók is kezdtek kibújni a tojásból. 1992-ben az SLS, 1993-ban a ma is elterjedt Slackware és Debian.
A Linux kernelt, és a köré tartozó programokat egy egységes könnyen használható egésszé az úgynevezett disztribúciók teszik. Magyarul ez Linux terjesztést jelent.
Ilyet bárki létrehozhat, persze nem olyan egyszerű, mint egy yukkát feldarabolni, de nem is túl nehéz (azért én egyiket sem tenném, legalábbis egyelőre nem).

Amennyiben a BSD-ről azt írtuk osztódással szaporodik, akkor a Linuxról elmondható a „gombamód szaporodik” kifejezés, de erről később. Most térjünk vissza a hősies kezdetekhez.
Vannak a Unix és a Linux történetében hasonló dolgok, de valahogyan kialakult egy olyan fejlesztői metódus és környezet a Linux fejlesztése során, ami teljesen eltér az addig megszokott dolgoktól, és konkrétan a Unix fejlesztési modelljétől is.
Erről egy igazán szakavatott jó munkásember, Eric S. Raymond (ESR) részletesen ír a Katedrális és a Bazár című sikerkönyvében. (Most akkor az ESR mérő a magasságát vagy a súlyát méri? Ja nem, ESR=electronic series resistance).
Kis kitérő, ezekről a rövidítésekről az jutott eszembe, hogy milyen jó szórakozás, ha a nevek kezdőbetűiből alkotott mozaikokra rákeresünk. RMS-re is nagyon sok jópofaság bejön még, de ki gondolta volna, hogy a HÖA vietnami és svéd nyelven is jelent valamit (az általunk ismert jelentésén túl). (Nhćng Đóm MẠt Höa Châu, vagy that's the swedish way to write how goofy laughs!

höa!höa!höa!).
Vissza ESR-hez, Raymond barátunk szintén nem akárki, a Wikipédia szerint számítógép programozó, író, és a szabad szoftver mozgalom pártfogója.
Persze ezen felül fejlesztő is, és anarcho-kapitalista, valamint feketeöves Moo Do harcos, hoppá, írjunk csak szépeket róla.

Nos, ha nem is rockzenész, de Santiago Segura betegsége esetén simán eljátszhatná a Torrente következő részének főszerepét.

Mit is ír ez a jóember korábban említett könyvében? Többek között:

„A Linux felforgató. Ki gondolta volna még csak öt évvel ezelőtt is (1991), hogy a bolygón szétszórt, csupán az internet finom fonalával összekötött sok ezer fejlesztő részidős bütykölése, mintegy varázsütésre, világszínvonalú operációs rendszerré egyesül? Én biztosan nem."
Egy másik esszéjében a következőket írja:
„Az egyik stílus – amelyet ma zárt kódú („closed source") fejlesztésnek hívunk – hagyományos gyári modell, amikor is a vevő egy lepecsételt bithalmot kap, amelyet nem tanulmányozhat, nem módosíthat vagy fejlesztheti tovább. A zászlóvivő ebben a megközelítésben a Microsoft.
A másik stílus a nyílt forráskód („open source") – így jött létre az internet is –, amikor is a forráskód elérhető, áttanulmányozható, kölcsönös kódvizsgálatnak vethető alá és gyorsan továbbfejlődhet. A zászlóvivő ebben a megközelítésben a Linux operációs rendszer.
A mostanában elhíresült Halloween Dokumentumokban a Microsoft elismeri, azt, ami az utóbbi kilenc hónap folyamán egyre világosabbá vált: a nyílt forráskód („open source") jó úton jár afelé, hogy elavulttá tegye a régi zárt forrású modellt. De, hogy megértsük a miértet, és hogy tisztán lássuk, mit jelent ez a jövő szempontjából, hátra kell lépnünk, és el kell vonatkoztatnunk a Microsoft és a Linux konkrét eseteitől. Három kvalitatív, általános szempontot kell figyelembe vennünk: a megbízhatóságot, a TCO-t (total cost of ownership - a birtoklás teljes költsége) és a szoftverekkel kapcsolatos kockázatokat."
Mindenesetre a Unix katedrálisszerű és a Linux bazárszerű fejlesztését összehasonlítva sok érdekes dologra rávilágít mindkét irományában. Persze az Internet robbanásszerű elterjedése is kellett ehhez, de önmagában nem jelentett volna semmit. Egyelőre hagyjuk a szerű-ket, és nézzük tovább mi történt a Linux háza táján.

Internet

Nemcsak a Linux fejlesztése miatt fontos dolog ebben a témakörben az Internet, de amiatt is, hogy megváltoztak és bővültek a szoftverekkel kapcsolatos igények. Sok ilyen igénynek a kielégítésére, egyedül a nyílt rendszerek voltak képesek, a megbízhatóság, és a gyors fejlesztés révén.
Ismerjük Bill Gates korai véleményét az Internettel kapcsolatban.
Másrészt, ha nincs WWW, vagy valami hasonló, ki akart volna Apache szervereket használni. Persze az is igaz, hogy ha nem ezt, akkor más igényeket elégítettek volna ki a nyílt rendszerek fejlesztői, gyorsan, megbízhatóan. Volt régen webszerver? Nem, nem is volt Web. Most van.
Volt régen adatbázis-kezelő? Volt. Ingyenes? Nem. Most van. Van, nem is egy, és nem is rosszabb, mint a kereskedelmi verziók.
A nyílt forrású rendszerek terjedése, egy korábban elképzelhetetlen folyamatot indított el, és gerjeszt ma is. Az olcsóbb és/vagy egészen használható ingyenes szoftververziók kiadását kényszeríti ki a nagy kereskedelmi szoftvereket fejlesztő cégekből. Ne legyenek illúzióink, enélkül egészen más lenne a helyzet.

A Linux jelentős dominanciára tett szert a szerver rendszerek egyes területein, és az ellenkező vélekedések dacára ez vár rá a desktopon is. Nyilván sokkal lassabban megy végbe ez a folyamat,
mint a szerver fronton, hiszen itt többek között meg kell küzdenie az ASZA szindrómával is, (AlacsonySzintű Agyműködés).

A folyamatos fejlesztés során valamiféle együttműködésre kellett törekedni más rendszerekkel is (mondjuk Unix), ezt a POSIX szabványnak való (valamilyen szintű) megfeleléssel lehetett a legegyszerűbben elérni. (Tudom, vitatott téma, hogy mennyire szabványos, de törekszik rá valamennyire). Így 1994 márciusában jelent meg az 1.0.0 Linux. A verziószámozás innentől lett olyan rendszerű, hogy a középső szám határozta meg, fejlesztői, vagy stabil végfelhasználói rendszerről van-e szó? (A páros jelenti a stabil kernelt). Így a „mezei” felhasználó mindig választhatja a stabil rendszert, ugyanakkor megfelelő gyorsasággal adhatók ki a fejlesztői verziók, a „nem stabil” ágon. Ezután folyamatosan nőtt a Linux programok száma, a rendszerben egyre gyorsabban jelentek meg az új hardvereszközök kezelésének lehetőségei, sőt megjelentek az olyan programok, amiket először Linuxra fejlesztettek ki, majd később írtak át más rendszerekre.

1996. augusztusában jelent meg a 2.0.0 kernel. Itt újra történt valami egészen előremutató dolog, a rendszermag memóriaigénye kisebb lett, míg hatékonysága és megbízhatósága megnőtt. Ez bizony követendő példa lehetne néhány nagy gyártó számára is, de tudjuk, hogy erről szó sincs. Időközben felmerült, hogy kellene a rendszernek egy olyasmi logó vagy embléma, amivel népszerűsíteni lehetne a rendszert („meg hát” a BSD-nek, és a GNU projektnek is van). Az ezzel kapcsolatos vitát hamarosan rövidre zárták, Linus kijelentette, hogy kedveli a pingvineket, és azt is, hogy egy jóllakott, elégedett, vidám pingvint szeretne. Megszületett az embléma, ami a keresztségben a Tux nevet kapta, a (T)orvalds (U)ni(X) szavakból. Ráadásul a tuxedo frakkot jelent, a sarkkutatók szerint ez a pingvin egyetlen ruhája, így ez is stimmel, persze ez már csak hab, azon a bizonyos tortán.

A fejlődés töretlen volt, a disztibúciók szaporodtak. 2001-ben megjelent a Lindows nevezetű, amely egy rövidke per zárásaként 2004-ben kénytelen volt nevét Linspire-re módosítani (Vajon melyik cég perelte be)?

Néhány apróság, amit a nyílt forrású fejlesztőknek és szabadságharcosoknak köszönhetünk:

Szerver oldalon minden alapvető dolog megvolt már, de a desktop perverz felhasználóit grafikus felület nélkül átcsábítani a szabadabb oldalra, igencsak nehézkes lett volna.
A Gnome és a KDE orvosolta ezt a problémát.
1996-ban megjelent a Gimp első kiadása, 1998-ban megnyitották a Netscape forráskódját. Mit csináltak? Nem pereltek, büntettek, tiltottak, megnyitották egy kereskedelmi program forráskódját!
Nem tudom, mit iszik Stallman, de ekkor nyugodtan kinyithatott belőle egy üveggel.
2000-ben adták ki a StarOffice forráskódját (ezt korábban még egy német cég fejlesztette, a Star Division, 1986-ban kezdték fejleszteni, akkor még csak egy szövegszerkesztő volt).

A StarOffice egész jó kis program volt, a Microsoft rendszer árának töredékéért. 1999-ben a Sun megvette a céget, az 5.2-es verziót ingyen letölthetővé tette, majd kiadták a forráskódot, ebből lett az OpenOffice. A StarOffice-ból most is van kereskedelmi verzió, 9.0 verziószámmal. Nincs vele semmi gond, az MSOffice-nál most is jóval olcsóbb, de legtöbben inkább, a ma már OpenOffice.org nevű csomagot használják, ingyen.
Erről jutott az eszembe, sokszor felmerül a kérdés, miből élnek ezek az emberek, miért vesznek valakik programokat, ha hozzáférhetőek az ingyenes megoldások, stb?
Jó kérdésekre jó válaszok jönnek.
Az univerzum és az emberi butaság végtelen. A nyílt forrású rendszerek ingyenessége nem azt jelenti, hogy nem lehet velük pénzt keresni, erre rengeteg példa van. Ezt lehet szépen és durván is csinálni.
Hazai példa a Magyar Office-é. Lehet, hogy van benne hozzáadott érték, de nem hiszem hogy többszázmillió forintot ér. Az OpenOffice-ra épülő Magyar Office-t terjesztő cég bevétele az előbbi szám. Nem emlékszem, melyik évben, és azt sem tudom, hogy ez ismétlődött-e az elkövetkező években, csak azt tudom, hogy abban az adott évben is elérhető volt az ingyenesen letölthető, magyar nyelvű, a felmerülő irodai igények 99,9%-át kielégítő OpenOffice verzió.
Persze az ingyen volt, tehát....
A szebb megoldás pedig, elhelyezkedni fejlesztőként, rendszergazdaként, support területen, stb.
Ma már a kereskedelmi cégek is elég sok pénzt ölnek a nyílt rendszerekbe, nem tennék, ha nem arra számítanának, hogy a többszörösét kivehetik belőle. Tehát van az a pénz, ha nem akar az ember a leggazdagabbak között lenni, hanem megelégszik a megfelelő emberi életkörülmények biztosítására elegendő összegekkel, a maga és családja számára.

Még egy érdekesség, vannak ingyenes Linuxok, és fizetősek. Csakhogy a fizetős megoldásnál sem főként magáért a disztribúcióért kérnek pénzt, hanem a hozzáadott szolgáltatásokért. Az egész fejlesztés a nyílt modellnek megfelelően történik, és hozzáférhetőek a fejlesztés eredményei, hozzáadott szolgáltatások nélkül, ingyenesen.
Korábban néhány cég próbálta összeegyeztetni a nyílt fejlesztést, és a zárt fejlesztéshez tartozó kereskedelmi módszereket, de valahogy mindig csőd, vagy a fejlesztés befejezése lett a vége (Corel, Stormix, Progeny).

Nézzük mi a helyzet 2008-ban?
Van Unix-unk, BSD-nk, Linux-unk.
Mindegyikből több fajta. A Linux-ból annyi

disztribúciót találhatunk, hogy egy élet nem lenne elég a kipróbálásukra sem, nemhogy a megismerésükre.
A főbb Unixokról volt szó, persze nem mindről. A főbb BSD-kről volt szó, persze nem mindről.
A főbb Linuxokról nem sok szó volt eddig, de van egy kiváló segítségünk, a distrowatch.
Mivel semmi sem tökéletes ebben a világban, lehet a distrowatch sem tud minden disztribúcióról, persze ez nem valószínű, de kiindulási pontnak mindenképpen jól megfelel, sőt kiváló.
Egy GNU/Linux terjesztés a kezünkbe ad minden olyan eszközt, amihez a mindennapi munkánkban általában szükségünk lehet. A disztribúciók sok azonos (vagy hasonló) programot tartalmaznak, de nem egyformák. Azért (is) van belőlük ennyi, mert esetenként más-más céllal lettek összeválogatva a komponensek. A distrowatch segít köztük eligazodni.
Ma, 2008.12.1-én, a distrowatch az alábbiakat regélte el nekem.
Az adatbázisában lévő összes disztribúció száma: 585 db.
Ebből aktív: 318 db.
Inaktív vagy nem folytatott: 196 db.
Várólistán: 160 (90 napos várakozási idő van, mielőtt felkerülne egy újdondász a listára).
Nofene. Mi az értelme ennek? Nagyon is sok, itt az egyik legfontosabb emberi tulajdonságról, vágyról és jogról van szó. A választás szabadságáról. (Ja, hogy vannak olyanok is, akik nem akarnak, vagy nem tudnak választani? Ez már legyen az Ő bajuk).

Gondoljunk bele, lehet valaki a régi masinájára egy már inaktív disztribúcióban találja meg a tökéletes választást.
Persze ennél valószínűbb, hogy valamelyik nagy felhasználói táborral rendelkező, igencsak aktív változatot fogunk használni.

Ha már szabadság, nézzünk még egy szabadságharcost.
John „maddog” Hall-t, programozót, a Linux International ügyvezető igazgatóját. Róla előző számunkban megjelent egy bemutató, amit érdemes elolvasni, így itt csak megemlítjuk, mint a "Nagy Öregek" egyikét.
A cikkünkben emlegetett személyeken túl még nagyon sokan tolják a nyílt forrású szoftverek szekerét. Ők a személyes haszonszerzés mindenhatóságát is meghaladva, valóban tesznek valami nagyszerűt az egész emberiség érdekében. A szoftverek szabadsága, az információ szabadságát is jelenti, ez pedig az emberek szabadságának egyik alapvető feltétele.

Munkájukért minden tiszteletet és megbecsülést megérdemelnek, köszönjük!

Nem sokkal a vége előtt érdemes megemlíteni még valamit. Ez az open source kezdeményezés szempontjából egy érdekes mutáns, a félreértések elkerülése végett beszélünk róla egy kicsit.
Az Open VMS-ről van szó. Nem hasonlít a Unixra, sőt más rendszerekre sem az RSX-en kívül (arra is csak azért, mert annak az utóda). A PDP-n futó RSX után jött a VAX-on futó VMS (Digital Equipment Corporation). A VMS 1.0 1978-ban jelent meg. 1992-ben megjelent az Alpha processzorra írt változat. A DEC 1990-től OpenVMS-nek hívta rendszerét (Open Virtual Memory System). Néhány apróság történt ezek után, a DEC-et a VMS-sel együtt felvásárolta a Compaq, ezután a Compaqot vásárolta fel a HP.
Szerencsére a VMS vonal maradt, így vele, vagyis az OpenVMS-sel ma a HP gépein találkozhatunk. Persze közben portolták Itanium gépekre.
Miért is mutáns (a mi szempontunkból)? Mert a forráskód ugyan szabadon elérhető,

de a licensze megtiltja a weben történő elhelyezést, és van még pár korlátozás.
Mellesleg a rendszer talán a világ legmegbízhatóbb operációs rendszere, főleg kritikus alkalmazáskörnyezetben használják. Tavaly töltötte be a harmincadikat.
Mivel elég drága hardvereken fut, nehéz hobby célból futtatni, ezt megkönnyítendő létezik OpenVMS Hobbyist projekt http://www.openvmshobbyist.org/news.php, és PC-n futó emulátor is ES40 http://sourceforge.net/projects/es40/, VAX www.charon-vax.com. Így van esély az alapok elsajátítására és annak az ördögi körnek a megtörésére, hogy a drága hardvereken futó OpenVMS rendszerekhez többnyire csak hozzáértő embereket vesznek fel, a rendszert megtanulni viszont csak OpenVMS környezetben lehet igazán.

Miután ezt tisztáztuk, már csak a jövőbetekintés maradt hátra.

Mértéktartó hírforrások szerint a kiváló svéd acél összekaszabolja a szoftvermogulok által ránkerőltetett kényszerű választások sötét posztóját, így a felfénylő fénysugarakban megláthatjuk az életükért rettegő szoftverkufárok vezetői irodáik homályos sarkaiban remegő sziluettjeit, amint a rájuk eső fénysugarak hatására fortyogva elillannak valami szörnyű, irtózatos helyre, ahol újramaterializálódva, izzó nullákat és egyeseket égetnek egymás petyhüdt bőrébe, így állítva elő teljesen ingyen a nyílt forrású rendszerek kódkészletét, miközben a gagyi vámpírfilmek alkotói sírva dülöngélnek a külvárosi sikátorok éjféli sötétjében, a rideg falak között. Ómen.

Eme kis vidám életképpel be is fejezhetném, de annyiszor emlegettem szakállas alakokat a korábbi oldalakon, hogy mindenképpen ide kell citálnom egy valóban szakállas viccet.

Miért jobb a BSD a Linuxnál?

Ezért.

Szőke József

Tudtad-e?

Pastebin

Igen, beilleszthető bárki a forráskódját erre (http://pastebin.com/) az oldalra. A több mint 60 programnyelv kódjait fogadó portál voltaképpen egy közös hibakereső és kód-megosztó hely, ahol névvel és névtelenül is kicserélhetőek, javíthatóak a kódok. Külön szeparét is lehet nyitni, ha valaki csak az általa kiválasztott felhasználók számára, zárt körben szeretné közzétenni a programjait, vagy programtöredékeit.

Jankovich Oszkár

Gépmagyar

Interjú Németh László Hunspell-fejlesztővel

Tövezés és toldalékolás, szótagszámlálás és elválasztás, betűcserék és elütések javítása, hibás szóismétlés megakadályozása - egyebek mellett erre képes a magyar fejlesztésű Hunspell helyesírás-ellenőrző, nemcsak magyar nyelven. A nyelvhelyességet támogató, nyílt forráskódú alkalmazáscsomag nemrég új szótárral bővült, hogy további képességgel ruházza fel az OpenOffice.org irodai programcsaládot és a Mozilla-típusú böngészőket, illetve levelezőprogramokat.

FLOSSzine: *Február 3-án jelentette be, hogy elkészült az OpenOffice.org-hoz az első használható szinonimaszótár, amely egyben az első nyílt forráskódú, magyar nyelvű szókincstár is egyben, 20 ezer szót, és unicode-os jelkészletet is tartalmaz. Gratulálunk, és a FLOSSzine szerzői és olvasói is köszönik a munkát. Hány év munkája, amit most használhatunk, és hány emberé?*

Németh László: Köszönöm szépen a gratulációt, nemcsak a saját nevemben. Vonyó Attila négy éven át készítette angol szótárát mintegy 6-7 különböző szótár és szakkönyv segítségével, aminek magyar része képezte a szinonimaszótár alapját. Nem számolva a már rendelkezésre álló, részben saját vagy a segítségemmel kifejlesztett eszközöket (pl. Magyar Ispell morfológiai és helyesírási szótár, Hunspell helyesírás-ellenőrző, Szószablya gyakorisági szótár), akkor a szinonimaszótár a szükséges segédprogramokkal 2 hónap alatt készült el. Több időt vett igénybe a tövező-toldalékoló Hunspell programkönyvtár és OpenOffice.org-folt elkészítése.

Fz: *A magyar tezauruszt mennyiben lehet jelen állapotában egyenértékűnek tekinteni az angol, vagy a német tezaurusszal? Mi szükséges, vagy milyen fejlesztői folyamatok szükségesek még ahhoz, hogy magabiztos támaszt adjon a szövegszerkesztéshez?*

N.L: A magyar 20 ezer, a német 50 ezer, a WordNet szemantikai szótárból készült angol nyílt forráskódú tezaurusz 140 ezer kifejezést tartalmaz. A magabiztos támasz már megvan a magyar tezaurusznál is, hiszen a gyakori szavak szinonimái megtalálhatók a 20 ezer szó között. Az angol szinonimaszótár jelentős szakszótári részt tartalmaz (például tízezernyi állat- és növénynevet latinul), ami a felhasználók nagy részét nem érinti, de a jövőben a magyar szótárban is várhatók hasonló fejlesztések. Az újdonság, és a magyar esetében szinte alapkövetelmény, hogy a toldalékolt szavakkal is megbirkózik a szinonimaszótár: a toldalékolt szavakat tövezi, a szinonimákat pedig a keresett szóalaknak megfelelően toldalékolja, így ajánlja például a "nyílvesszőimet" szóra a "nyilaimat" alakot. Érdekesség, hogy a tövezés szinte minden más Hunspell szótárral is működni fog, ahol a szótári toldalékok leírása követi a nyelv szabályait. A kivételek ke-

zeléséhez, illetve az automatikus toldalékoláshoz szótári bővítésre van szükség. Ez a magyar és az angol Hunspell szótárakhoz elkészült, de a német esetében még nincsenek toldalékolt szinonimák, ebből a szempontból tehát az OpenOffice.org magyar szinonimaszótára előbbre jár, mint a német.

***Fz:** Mikor, mely Linux-disztribúciókba kerülhet be a szótár?*

N.L.: A magyar fejlesztésű, illetve magyar fejlesztővel vagy aktív magyar felhasználókkal rendelkező disztribúcióknak talán már része is. A bekerüléstől függetlenül a http://extensions.services.openoffice.org oldalon elérhető az új szinonimaszótár.

***Fz:** A Hunspell esetében mennyi idő telik el általában, ameddig bekerülhet a legfrissebb verzió a nagyobb disztribúciókba?*

N.L.: Legtöbb visszajelzést és javítást a Debiantól (Rene Engelhardt), a Red Hattól (Caolan McNamara) és az openSUSE-tól kapok, de több más disztribúciótól, illetve más operációs rendszerektől (legutóbb a Solaristól) kapok visszajelzést. A megjelenési idő ezeknél a disztribúcióknál rövid.
Megemlíthető még az OpenOffice.org és a Firefox is, ahol kevésbé kiszámítható, hogy mennyi idő alatt frissül a Hunspell. A Sun által karbantartott OpenOffice.org forrásfában már több mint fél éve húzódik a legújabb Hunspell beillesztése (igaz, közben több javításra is sor került a Hunspellben), míg a Firefoxnál Ryan Vander Meulen külsősfejlesztő segítségével pár hét alatt bekerült a Hunspell 1.2.8 a Firefox 3.1 fejlesztői változatába.

***Fz:** A Hunspell jelenleg hány nyelven, hány nyelvhez használható? Van-e ezekben a nyelvekben valami közös, vagy más tényezők miatt érhető el aprogram éppen ezeken a nyelveken?*

N.L.: Több mint száz különböző nyelvű Hunspell szótár tölthető le az OpenOffice.org oldalairól. A nagyobb és fejlettebb országok nyelvei vannak túlsúlyban (sokszor a kisebbség nyelvei is innen, mint a lapp szótár Norvégiából, okcitán Franciaországból, a baszk Baszkföldről).

***Fz:** Úgy olvastuk, a helyesírás-ellenőrző, a szóelválasztó, és a szinonimaszótár után már csak a nyílt forráskódú magyar nyelvhelyesség-ellenőrző hiányzik a szövegszerkesztést támogató anyanyelvi eszközökből. Az a tervek szerint mi mindenre lesz képes (lesznek pl. kontextusfelismerő képességei?), és első verziója mikorra állhat össze?*

N.L.: Létezik már több nyelvhez is nyílt forráskódú nyelvhelyesség-ellenőrző, amit használhatunk az OpenOffice.org-gal, de szükség volna egy kis méretű alapértelmezett nyelvhelyesség-ellenőrzőre is. Ez első lépésben az egyértelmű, különösebb mondatelemzés nélkül is felismerhető hibák javítását tenné lehetővé, és csak a második fejlesztési szakaszban kapna komolyabb mondatelemző képességeket. A magyar nyelvi modul párhuzamosan készülne a nyelvfüggetlen nyelvhelyesség-ellenőrzővel. Alapvető fejlesztési szempont, hogy a nyelvhelyesség-ellenőrző javaslataival elősegítse az igényes dokumentumok készítését, de közben ne gátolja az írás folyamatát felesleges és téves hibajelzésekkel. Első verziója pár hónapon belül elkészülhet, ha sikerül a fejlesztésre támogatást szerezni.

***Fz:** A Szószablyához hasonló projektre lenne-e még szükség a jövőben a további munkájához?*

N.L.: A Szószablya projektben a BME Média és Oktató Kutatóközpontban elkészült a Magyar Webkorpusz, egy 18 millió magyar nyelvű weboldal feldolgozásán alapuló 1 milliárd szavas szöveggyűjtemény. A korpuszból készült gyakorisági szótár mind a Magyar Ispell helyesírási szótár, mind a nyílt forráskódú magyar szinonimaszótár készítésénél nagy szerepet kapott. A Szószablya projekt rendszeres ismétlésével (teljesen automatizálható a folyamat a kifejlesztett eszközökkel) követni lehet a magyar nyelv változásait, ami tíz-húsz év alatt már mérhető a szóhasználat tekintetében, így szükség is lesz rá a jövőben. A nyelvhelyesség-ellenőrző és a szinonimaszótár tervezett folytatásai, a szófaji és mondatelemzést magába foglaló szintaxis-ellenőrző és a WordNet szinonimahalmazoknak megfeleltetett magyar szemantikai szótár kifejlesztése összemérhető a Szószablya projekttel.

***Fz:** Vannak-e - a funkcionalitást, a felhasználhatóságot tekintve - lényeges különbségek az Ön által kifejlesztett eszközök és a világnyelvekhez elérhető, azonos célú eszközök között?*

N.L.: Az Unicode-támogatás, amire a világnyelvek szakszótárainak is szüksége van, mind a mai napig újdonságnak tekinthető. Sőt mint ahogy a napokban a joruba Hunspell szótár fejlesztése rámutatott, komoly igény van a még teljesebb Unicode támogatásra a Hunspellnél is. (A joruba nyelv több Unicode karakterrel leírt ékezetes betűit a Hunspell még nem tekinti egyetlen betűnek, ami a javaslattevés minőségét rontja.) Az összetett szavak kezelése a Hunspell másik erőssége, ami olyan nyelveknél is lehetővé teszi lényegesen jobb helyesírási szótárak fejlesztését, mint a német és a holland. A Hunspell nem egy befejezett eszköz, hanem igény szerint bővül a legkülönbözőbb képességekkel, mint például legutóbb a telugu és más indiai nyelvek összetett szavaiban megfigyelhető sandhi hasonulás támogatásával, vagy a helyesírás-ellenőrzőknél szintén újdonságnak tekinthető szótári kiejtési információk kezelése. Ez utóbbit a Hunspell a szabály alapú kiejtési információkkal együtt képes figyelembe venni a javaslattevésnél.

Fz: *A szövegszerkesztést támogató nyelvi programok kifejlesztése más hozzáállást, technológiát követel-e egy agglutináló nyelv esetében, mint egy flektáló nyelv esetében? Vannak-e lényeges különbségek a mintaillesztési eljárás kidolgozásában?*

N.L.: A különbség jelentős, például az angol nyelv esetében a leggyakoribb pár százezer szóalak egyszerű listája (/usr/share/dict/words) elegendő a helyesírás-ellenőrzéshez, addig a magyarnál a szóalaktan és az összetételi szabályok leírására és kezelésére van szükség. A Magyar Ispell helyesírási szótár első változatai az International Ispell helyesírás-ellenőrző programhoz készültek, ami korántsem agglutináló nyelvhez lett kitalálva. Bonyolult megoldásokkal sikerült csak használható helyesírási szótárat készíteni az Ispellhez, de a Hunspell mellett ma már nincs szükség ilyen trükközésre. Maga a fő Ispell eljárás, azaz a toldalékok leválasztása a szótári alak kereséséhez nem változott, csak kibővült úgy, hogy jóval egyszerűbb és hatékonyabb legyen az agglutináló nyelvekre jellemző sok toldalék kezelése.

Fz: *És vannak-e lényeges különbségek - megint csak felhasználói szemszögből - az Ön eszközei és a kereskedelmi szoftverekbe épített eszközök között (pl. Helyes-e? program)?*

N.L.: A leglényegesebb különbségeket emelném csak ki a helyesírás-ellenőrzés és az elválasztás terén (ez utóbbinál csak az elválasztó program és Nagy Bence magyar elválasztási szótárának bővítése fűződik a nevemhez). A nyílt forráskódú magyar helyesírási szótárral az OpenOffice.org feleannyi téves hibajelzést ad az Index tesztje alapján, mint a Microsoft irodai csomagja a Morphologic helyesírás-ellenőrző programjával, ami az alapszókincs nagyobb lefedettségére vezethető vissza. Meglepő kijelentés, de a magyar kereskedelmi irodai csomagokban és kiadványszerkesztőkben széles körben használt MorphoLogic Helyes-e? nem alkalmas az automatikus elválasztásra, mert az általa nem ismert (gyakran a leghosszabb többszörösen összetett) szavakat nem választja el, ami súlyos szedési hibákat eredményez. A minta alapú Hyphen elválasztóprogram ezzel szemben képes az automatikus elválasztásra, és legújabb változatával további minőségi ugrás várható az összetett szavak elválasztásában.

Fz: *Nem szerepel-e a tervei között, hogy a Hunspellt, és a tezauruszt az OpenOffice.org, és a Mozilla-család mellett a Google Apps-programcsaládnak is a rendelkezésére bocsássa?*

N.L.: Mivel nyílt forráskódú eszközökről van szó, ez már megtörtént. A Google Dokumentumok már régóta használja a Magyar Ispell szótárat, bár a rosszabb eredményt nyújtó Aspell programmal.

Fz: *Egyáltalán van-e - szakmai, vagy közelebbi kapcsolata - a Google magyar honosítóival (hiszen az a rendszer is - nagyjából - nyílt forráskódú programokon nyugszik)?*

N.L.: Nincs, a kapcsolatkeresésre nem válaszoltak még.

Fz: *És van-e szakmai kapcsolata Prószéky Gáborral, a konkurens kereskedelmi program fejlesztésének irányítójával?*

N.L.: Szakmai konferenciákon gyakran találkozunk, de nincs közöttünk szorosabb együttműködés, mivel mással foglalkozunk, a Prószéky Gábor vezette MorphoLogic Kft. most főleg gépi fordítással. Konkurenciáról azért sem beszélhetünk, mert más piacokon versenyzünk, részben azért is, mert nincs pia-

ca a nyelvtechnológiának a kereskedelmi programok körében ma Magyarországon: a Microsoft és így az MS Office állami monopóliumára idén is 25 milliárd forintot szánt a kormány.

***Fz:** Előbb a nyelvészet, vagy előbb a programfejlesztés - melyik területtel hogyan kezdett el foglalkozni, mi az, ami a pályája elején megragadta a számítógépes nyelvészetben?*

***N.L.:** A hobbiprogramozás, amit űztem addig, az ingyenes TeX szövegszedő rendszer és a teljes fejlesztői környezettel rendelkező nyílt forráskódú GNU/Linux felfedezése után fordult komolyabbra. Sokan dolgoztak 1998-ban a GNU/Linux magyarításán. Mivel a linuxos TeX rendszer két legnagyobb hiányossága a magyar tárgymutató-készítő és a magyar helyesírás-ellenőrző program volt, nekiálltam ezek elkészítésének. A Mami névre keresztelt tárgy-mutató-készítő (magyar makeindex pre- és posztprocesszor) hamar elkészült, de a Magyar Ispell helyesírási szótár fejlesztése 2 év kihagyás után indult be igazán. Bár pályáról még nem beszélnék, a helyesírási szótár elkészítésének legemlékezetesebb pillanata az volt, amikor a magyar szóalaktan és szókincs leírása és osztályozása után működés közben láthattam a helyesírás-ellenőrzőt. Ami pedig újra és újra elkápráztat, az a Unix parancssor ereje, ami nélkül sem a helyesírási szótár, sem a szinonimaszótár nem készülhetett volna el.*

***Fz:** Dömölki Bálint dolgozta ki - és állítólag először 1964-ben magyarul publikálta, majd egy orosz szakfolyóiratban is megjelentette - azt az algoritmust, amelyen a Hunspell alapszik. Mennyiben alapszik azon: mennyire azonos és mennyire változott az algoritmus?*

***N.L.:** A magyar számítástechnika hőskorában készült Dömölki Bálint mintafelismerő algoritmusa, amint arról Kovács Győző élvezetes írása, a Válogatott kalandozásaim Informa-tikában c. könyv is beszámol. Az International Ispell és később a MySpell és Hunspell programok is ezt az algoritmust használták a toldalékolásnál a szó eleji, illetve szó végi feltételvizsgálatra. Az Unicode-kódolású szótárakhoz a Hunspell kénytelen volt kiegészítő algoritmust választani, a Hunspell 1.2-es változatától pedig egy teljesen új algoritmus gondoskodik a ragozási szabályok illeszkedésvizsgálatáról a 8 bites karakterkódolású és az unicode-os szótárak esetében is. Ennek oka, hogy a Dömölki-féle algoritmus teljes ábécével indexelt tömböket használ, ami az Unicode karakter-alapkészlet esetén is már nagyságrendileg nagyobb helyigénnyel járna, illetve az agglutináló nyelvek 8 bites szótárai esetén már így is nagy volt a helyigény (vagy más jellegű optimalizáció esetén az időigény) a több tízezernyi ragozási szabály használatakor. Az új algoritmus 5 MB-tal csökkenti a magyar szótár memóriaigényét, miközben kevesebb mint 10 százalékkal növeli az ellenőrzési időt.*

***Fz:** Azt, hogy 2008 végén Pekingben vehetett át kitüntetést a munkájáért, alig írta meg néhány portál. Újságról nem is tudok. A magyar számítógépes nyelvészek, az MTA nyelvtechnológusai mennyire értékelik a munkáit, a fejlesztéseit? Együttműködik-e velük? Vagy kielégíti az, hogy talán inkább csak a szabad szoftver hívei találkoznak a nevével és a munkáival?*

***N.L.:** Szeretném, ha a pályamunka, a tövező-toldalékoló szinonimaszótár megjelenése lenne a széles körben megjelenő hír, amire az OpenOffice.org 3.1 kiadásával is sor kerülhet. A magyar nyelvtechnológusok nagyra értékelik, és aminek különösen örülök, használják is a munkáimat. Számos hibajelzést köszönhetek a magyar felhasználóknak, sőt egy ideig együtt is dolgozhattam magyar nyelvészekkel és nyelvtechnológusokkal a MOKK munkatársaként. Tavaly év végén keresett meg Kornai András nyelvész, hogy vegyek részt a Magyar nyelv- és beszédtechnológiai platform munkájában, amire minden okunk meglesz Godó Ferenccel, jelenlegi munkatársammal, ha sikerre visszük a tervezett magyar nyelvtechnológiával kapcsolatos fejlesztéseinket.*

***Fz:** Köszönöm a válaszokat!*

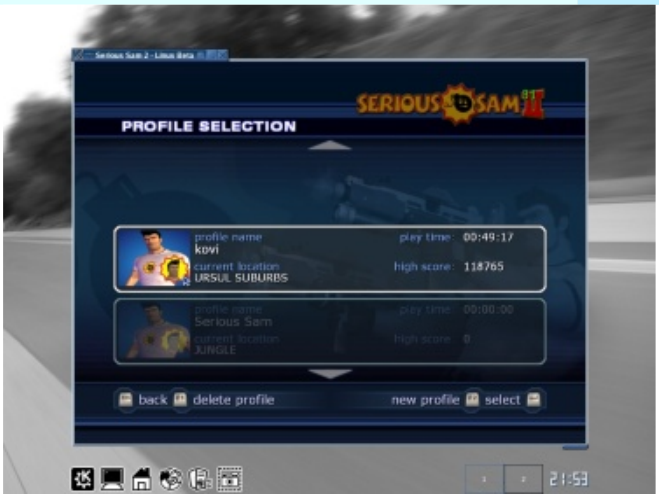
***N.L.:** Köszönöm a kérdéseket! Szeretném megköszönni a magyar szabad szoftveres közösségnek, hogy közvetlenül és az FSF.hu Alapítványon keresztül is hozzájárult az elért eredményekhez és támogatásával lehetővé teszi a fejlesztések folytatását!*

Jankovich Oszkár

Serious Sam 2

Komoly Samu újra tarol

Sam első kalandját talán nem szükséges különösebben bemutatnom az FPS kategóriát kedvelő Olvasóinknak, hiszen ez a szoftver Linux környezetben is jól ismert. Szerencsére nem kell „kegyesen” megfeledkeznem a második epizódról sem, hiszen a béta állapotú natív binárisa stabilan fut Linuxon.



1.ábra A natív játék, KDE asztalon

Bárkiről is legyen szó: aki a Serious Sam The First Encounter linuxos verzióját megpróbálná röviden felvázolni, alapvetően könnyű dolga volna. Az élvezetes, tömegmészárlásra kihegyezett Arcade FPS játékról igazából nem is lehet rosszat írni - negatív kritikát csak attól kaphatna, aki nem rajong a lövöldözős stílusért. A második részről sokkal nehezebb feladat cikkezni, no nem azért, mert rossz programról van szó, sőt! „Komoly Samu” és az ő virtuális világa olyan elképesztő fejlődésen ment keresztül, ami által az előző rész pozitív kritikáit szin-

3.ábra A helyzet Klodovikért kiált

te lehetetlen kellőképpen tovább fokoznom. A horvát fejlesztők hihetetlenül nagyot alkottak!

Miben több az elődjénél?

Mindenben. Ez a megállapítás a Croteam nem mindennapi teljesítményét dicséri: öreg igazság, hogy csak a legnagyobb projektek és a legprofibb csapatok képesek túllépni „előző önmagukon”. Nézzük tehát sorban, milyen újdonságok várják a rajongókat! Először is naprakész projekthez méltó, profilkezeléssel ellátott menürendszert programoztak. A játék lehengerlő látványvilágát új grafikus motor generálja, ebből eredően sem a vizuális képességiben, sem pedig az alkalmazott fizika terén nincs szégyenkezni valója a műfajt húzó programokkal szemben. Rengeteg új, változatos pálya is született, több tucat kemény ellenféllel, és a totális kipusztításokhoz nélkülözhetetlen új fegyverekkel együtt. Ezeken túl sok

4.ábra Ettől óvnak a pszichológusok

új ötlet gyarapítja a listát, hogy csak néhányat említsek: rombolható pályaelemek, mozgatható tárgyak, használható járművek, és egy segítségünkre lévő barátságos faj... Sorolhatnám tovább az újdonságokat, de nem tudnám kellően kifejezni a meglepődésemet. A közkedvelt első epizódot ez a játék annyi téren múlja felül, hogy a bevezető kérdést sokkal inkább így kellene feltennem: „Van-e olyan jellemzője, amiben nem több, mint az elődje?”. A válasz valahogy így hangzana: „Nincs - hacsak az nem, hogy itt is Doom-szerű véres tömegmészárlásról szól a dolog, mint az első részben.

Kissé azért zavaró, hogy pont ez a játék az, amitől a pszichológusok féltve óvják a fiatalságot: részben igazat adva nekik én kiskorú játékost a közelébe sem engednék, mert bizony nem feltétlenül szolgálja a normális jellemének kialakulását...

Linuxon...

Elődjéhez hasonlóan a második rész is alapvetően Win32 alapú szoftver. Ebben a rend-

5.ábra Lehengerlő látványvilág

szerben a főkód megköveteli a DirectX környezetet, abból is a 9-es verziót. A portolási munkálatok épp ezért mély tiszteletet érdemelnek: a piacvezető szoftvergyáros rendszeréről úgy költözött át Linuxra a projekt, hogy alapvetően nem látszik különbség a két változat között. A számunkra fontos bináris a cikk írásakor még béta RC2 állapotú - ennek ellenére megbízhatóan működik. Jelenleg két kirívó hiányossága van: az egyik a pályák közti átvezető videók lejátszására vonatkozik (ezt a feladatot még nem képes hibátlanul megoldani), a másik pedig az „on-line multiplayer” módot érinti: erre még egyáltalán nem képes. Ahhoz, hogy linuxos gépünkön futtathassuk a játékot, szükség lesz a Serious Sam 2 windowsos telepítő lemezére. Ezt akár a Microsoft rendszerének segítségével, akár egy Linuxon futó Wine wrapperen keresztül telepítsük fel a kiszemelt PC-re. A közel 3 GByte méretű installált programot ezek után húzzuk át a személyes mappánkba úgy, hogy az átemelt

6.ábra Céllövölde, horror beütéssel

fájlok módosítási jogával rendelkezzünk. Ha a művelet véget ért, látogassunk el a fejlesztő csapat honlapjára, irány a http://www.croteam.com! A bal oldali „Downloads” linkről indulva töltsük le a natív binárist tartalmazó ss_linuxbeta_public.tar.gz tarballt (~50 MByte), majd struktúrahelyesen bontsuk ki a játék előzőekben átmozgatott útjára, felülírási engedéllyel. Sam kalandjait ezek után a fő könyvtárában, felhasználóként kiadott ./RunSam2 paranccsal hívhatjuk életre. A kompromisszumoktól mentes használathoz nagyjából 2000 MHz órajelű központi egység, 768 MByte központi tár, és egy naprakész (Pixel Shader 2.0 -val bíró) 3D videóhardver szükségeltetik, utóbbi természetesen betöltött GLX illetve DRI meghajtóval. Fontos, hogy a program kódja egyaránt komoly terhet ró minden

2.ábra „Igéző” tekintetek...

hardverre: nagy felbontású textúrákat feszít a poligonokban közepesen gazdag modellekre, melyek mozgáskultúrája eléggé összetett. Ráadásul ezek a modellek tömegével vannak jelen a hatalmas külső (vagy épp a bonyolult belső) tereken, aminek a minősége szintén erős számítógépet feltételez. A teszthez használt 3200+ jelölésű AMD64 CPU, Geforce 6600 TD (256 MByte) videokártya, 1 GByte memória trió nem állt mindig a helyzet magaslatán: 1024x768 felbontásban, részletgazdag és effektdús képen sokszor épp csak 30 képkocka/másodperc sebességre volt képes.

Apróságok és részletek

A kerettörténet még az előző részben megismertnél is langyosabb: arról van szó, hogy a főellenség valamiféle kristály darabjai

után kutat, hogy az általa szerzett újult erővel szabadíthassa a világra minden haragját. Ezeket a darabokat kell begyűjtenünk,

7.ábra Mozsárágyú, helikopter ellen

mielőtt rossz kezekbe kerülnének. Még szerencse, hogy a műfajnak nem lételeme a jó körítés...

A játék hangulatát azonban nem csak a bevezetőben említett, szemet szúró újdonságok biztosítják. Pár óra öldöklés után már érdemes odafigyelni az apróságokra is: az ellenfelek és barátságos népek is nagyon jó hanghatásokkal rendelkeznek, mint ahogyan Sam sem megy a szomszédba egy-két komikus beszólásért (még a gépi narrátor is „laza" stílust képvisel). A fegyverek között található néhány nagyon eredeti is, példaként Klodovik, a „kamikaze-papagáj" - akit gránáttal a nyakában küldhetünk az ellenfél csapatának a közepébe... Kifejezetten jól sikerültek a „járművek": akár egy dinó hátán ülve, akár szegecsekkel kivert gömbben rohangálva, ugyanolyan élménnyel lehetünk gazdagabbak, mint később a plazmaágyús suhanót irányítva. Bár túl nagy dolognak nem mondanám, de a hálózati játékmód is megér egy misét: a kooperatív lehetőséggel élve egyszerre akár tízes nagyságrendben oszthatjuk együtt az ellenséget. Aki szereti a „fejlesztői munka" kihívását, tervezhet saját pályákat is - a natív verzióból ez sem maradt ki (a szoftver fő mappájában kiadott ./RunEditor parancsra betöltődik a pályaszerkesztő).

Konklúzió, játék

A Serious Sam 2 kiadásakor (2005) volt szerencsém kipróbálni a Win32 verzió demóját. Akkor még túlzottan nem erőltettem magam, hogy a teljes programmal üthessem agyon a szabadidőmet. Nem mondanám, hogy meggyőződésből kerültem a windowsos kiadást, de a szívem mélyén azért natív linuxos játéklehetőségben reménykedtem. Így aztán, amikor közkinccsé váltak a szabad rendszerhez kötődő állományok, rögtön rátaláltam az utóbbi évek egyik legjobb, legeredetibb lövöldözős játékára. Itt semmiféle taktikázás, semmilyen mesterséges lassítás nem rontja a légkört: csak a színtiszta reflexmunka, a rutinos mozgáskultúra és céllövöldéket megszégyenítő lőgyakorlat vihet tovább a következő pályára. Furcsa dolog ez, hiszen a műfaj ilyen irányú fejlődése nagyjából tíz éve elhalt... Szerencsére a hor-

8.ábra Effektek effektek hátán

vát srácok képében akadt, akik nem felejtették el a dicső múltat. Manapság persze akad olyan FPS projekt, mely túlmutat Sam második kalandján - ellenben amíg egy játékszoftver élménye olyan erősen merít az Arcade múltból, mint ez, addig nem hirdethető ki egyértelműen „új király". Aki szerette a Doom-ot, vagy éppen a Quake világáért rajongott, az ne keressen tovább: ebben a virtuális környezetben újra rátalálhat a kategória gyökereire, modern csomagolásban. A mellékelt képek önmagukért beszélnek - még úgy is, hogy nem tudják maradéktalanul visszaadni a mozgásban lévő környezet hangulatát. „Samu" kalandjainak második epizódja felnőtt megszállottaknak kötelező darab, kiskorúaknak pedig szigorúan kerülendő.

Kovács Zsolt

Interjú Richard Stallmannel

FLOSSzine: *Mielőtt elmerülnénk a specifikusabb témákban elmondaná mit is jelent a "szabad szoftver" kifejezés?*

RMS: A szabad szoftver azt jelenti, hogy a felhasználója élhet a következő 4 alapvető szabadságjoggal:

- 0. szabadságjog: annak joga, hogy arra használd a programot, amire te akarod
- 1. szabadságjog: annak joga, hogy tanulmányozd és/vagy megváltoztasd a program forráskódját, hogy úgy működjön, ahogy te akarod
- 2. szabadságjog: annak joga, hogy segítsd másokat azáltal, hogy másolatot készítesz a programról és azt közzéteszed, ha ezt akarod
- 3. szabadságjog: annak joga, hogy segítsd közösségedet azáltal, hogy a program módosított változatát közzéteszed, ha ezt akarod

Nos, ha egy program a fenti négy alapvető szabadságjog mindegyikét biztosítja, akkor szabad szoftver, mivel a használatának és terjesztésének rendszere társadalmi szempontból etikus, egy olyan rendszer, mely tiszteli a szabadságod és a közösségeddel való összetartást. Ha azonban egy ezek közül a szabadságjogok közül hiányzik, vagy nem teljes, akkor az társadalmi szempontból nem etikus és nem lehet szabad szoftver. Ahogy látszik, ez nem egy technikai jellegű kérdés, hanem annak a kérdése, hogy a szoftver terjesztési módja tiszteli-e a szabadságodat. A szabad szoftverek fejlesztése tulajdonképpen társadalmi hozzájárulást jelent.

FLOSSzine: *Nem érzi-e úgy, hogy a világ még nem érett meg ennek a filozófiájának befogadására? A közelmúlt egyik példája az OpenSSH projekt, akik egy széles körben ismert és elismert szoftvert fejlesztenek, ám felhasználóik mégis javarészt "megfeledkeznek" arról, hogy mit kaptak a szabad szoftveres közösségtől.*

RMS: Nem gondolom, hogy olyan sokat számítana mennyivel támogatják az emberek az OpenSSH projektet, mivel amennyire én tudom, az általuk fejlesztett szoftverek jelenleg is kielégítik a velük szemben támasztott alapvető igényeket. Természetesen számos olyan dolog lehet, amivel hasznos lehet kibővíteni, talán tervben is vannak ezek, de amennyire én tudom nincsenek komolyabb hiányosságok. Vannak viszont más területek, ahol viszont vannak problémák és szükséges lenne fejlesztésekre, ahol fontos lenne az emberek közreműködése. Ilyen magas prioritással rendelkező projektek találhatóak

például a GNU projekt weblapján is, melyekről valóban el lehet elmondani, hogy szükségük van a támogatókra. Ez a támogatás egyfelől lehet pénzbeni, de állhat abból is, hogy valaki időt energiát szán a projektre.

FLOSSzine: *Ön szerint melyik a jobb megoldás?*

RMS: Azt gondolom helyes dolog olyan embereket pénzzel is támogatni, akik hasznos szabad szoftvereket fejlesztenek. Bizonyos mértékig engem is támogattak olyan emberek, akik méltányolták a munkámat. Hogy az elismerésük a szoftvereknek szólt-e melyeket - javarészt a 1980-es években, kisebb részt az 1990-es években - fejlesztettem, vagy a szabad szoftverek eszméjének terjesztésben elvégzett munkámnak, nem tudom. Mások úgy működnek közre szabad szoftver projektekben, hogy szerződtetnek fejlesztőket. Hogy miért ezt a megoldást választják nem tudom. Vagy valamiféle hálából, vagy egyszerűen csak azért mert szeretnének bizonyos haladást elérni és tudják, hogy ez egy jó módszer lehet erre.

FLOSSzine: *Véleménye szerint érdemes-e törekedniük a szabad szoftvereket fejlesztőknek arra, hogy hosszabb távú megállapodásokat kössenek egyes hardvergyártókkal a teljesebb kompatibilitás érdekében?*

RMS: Szükségünk van szabad driverekre és az olyan esetekben, amikor az eszközt firmware működteti, szükségünk van szabad firmware-ekre is. Ennek okán aztán olyan gyártókra van szükség, akik úgy kooperálnak a szabad szoftverek fejlesztőivel, hogy közreadják az általuk gyártott eszközök specifikációit. Amennyiben ennél tovább akarnak lépni és maguk akarnak szabad szoftvereket fejleszteni hardvereikhez, az igazán remek, nekünk nincs más dolgunk, mint megköszönni ezt. Ez azonban nem feltétlenül szükséges. Ami feltétlenül szükséges, az a specifikációk publikálása. Úgy

vélem, hogy a gyártók morális kötelessége az együttműködés, nekünk pedig ösztönöznünk és kényszerítenünk kell ezt az együttműködést. Azt gondolom azonban, hogy nincs szükség külön megállapodásokra, ha a gyártók így akarnak cselekedni, tegyék azt. Amit én szorgalmaznék az az olyan számítógépek vásárlása, melyek minden részükben támogatottat és egyetlen hardverelem sem teszi szükségessé nem szabad szoftverek telepítését. Nagyszámú vásárló esetén erőteljesen hatást lehet gyakorolni a piacra, hogy ez lehetővé váljon.

FLOSSzine: *Mégis milyen módszerekkel érhetjük el a legszélesebb körű hardvertámogatottságot?*

RMS: Két lehetséges módszer van az olyan hardverek esetén, melyek specifikációi titkosak. Az egyik a reverse engineering, azaz, hogy megfejtjük működésüket. A másik az, hogy a piaci erőnket használjuk fel, vagyis nem vásároljuk ezeket az eszközöket. Ezzel a gondolattal visszajutunk oda, hogy ha van is több tízmillió, vagy akár több mint százmillió felhasználónk a világban ha nem foglalkoztatja őket ez a kérdés, akkor hiába ez potenciálisan óriási piaci erő, nem fogjuk tudni azt kihasználni. Ha a vásárlók foglalkoznának felhasználói szabadságjogaikkal, akkor nem vennék az ilyen termékeket és jóval egyszerűbben meggyőzhetnénk a hardvergyártókat. Ezt elősegítendő állított

fel a Free Software Foundation egy listát , mely azokat az eszközöket tartalmazza - számos kategóriában - melyek támogatottak a szabad szoftverek által. Ahhoz, hogy hatást tudjunk elérni az embereknek el kell kezdeniük ezeket az eszközöket használni és ragaszkodniuk kell hozzájuk.

FLOSSzine: *A laptopok, notebookok esetén különösen gyerekcipőben járunk.*

RMS: Még csak most kezdenek olyan laptopok megjelenni, melyeknek nincs szükségük nem szabad BIOS-ra például. Az én laptopom ilyen. Minden szinten csak szabad szoftvert futtat, kezdve inicializálást végző szoftveren át - ami ez esetben nem BIOS - a firmware-eken át az operációs rendszerig. Azt gondolom ez jó dolog, még ha ez a gép egy prototípus is, ami egyelőre még nem érhető el kereskedelmi forgalomban. Van még néhány hiba, amit javítani kell és remélem, hogy ez meg is történik az elkövetkezendő néhány hónapban, mivel ez komoly előrelépést jelent majd a laptopok kérdésében.

FLOSSzine: *A GNU/Hurd elég régóta húzódik. Van valamilyen tervezett dátum, amikorra használható lesz? , és akkor hogyan tovább, leváltják-e a Linux kernelt, vagy a közösségre bízzák, és mindenki azt használja, amelyiket akarja?*

RMS: Nincs ilyen dátum. Nem tudom, hogy van-e egyáltalán valaki, aki ezt igazán használni akarná, de ez nem is egy kritikus probléma, hiszen létezik egy szabad kernel, nevezetesen a Linux-libre, ami a Linux egy olyan módosított változata, mely nem tartalmaz nem szabad részeket. Egyáltalán nem tartom szükségesnek egy másik szabad kernel létrehozását. Mikor a GNU Hurd fejlesztését megkezdtük még nem létezett szabad kernel. Akkor 1990-et írtunk és a Linux csak 1992-ben vált szabaddá. Az ok ami miatt tehát belekezdtünk a kernel projektbe, hogy legyen egy szabad kernel. Az konstrukció (mikrokernel - a szerk.) amit választottam - talán hibásan - is azt a célt szolgálta, hogy mielőbb elkészülhessünk vele.

FLOSSzine: *Mi a véleménye a manapság elég gyakori dual-licence technikáról?*

RMS: Nincs vele semmi probléma. Ha egy olyan programról beszélünk, amely elérhető több különböző licenc alatt is, akkor amíg az egyik szabad szoftver licenc, addig a szoftver szabad szoftver és a többi licenc ebből a szempontból nem számít. A szabad kód etikus a nem szabad kód nem az. Ilyen egyszerű. Meg kell vizsgálnunk a kód minden egyes részét, hogy milyen módon érhetőek el a felhasználó számára, szabadon, vagy sem. Ez meg fogja adni nekünk a választ.

Pfeiffer Szilard

Tudtad-e?

SCRIBD és társai

Talán az egyik legjobban használható szolgáltatás az interneten a Scribd. Valaki találóan úgy jellemezte, hogy az "írásos dokumentumok You Tube-ja". Ezen az oldalon a legkülönbözőbb írásos dokumentumokat találhatjuk (kézikönyvek, tanulmányok, dokumentációk, stb). Nemcsak böngészni tudunk közöttük, de mi magunk is feltölthetünk, és megoszthatunk anyagokat másokkal. Újszerű kezelőfelülete különbözteti meg a többi hasonló szolgáltatástól, ez lesz, akinek tetszik, másoknak nem. Magyar nyelvű dokumentumokat is találhatunk rajta, akárcsak a hasonló szolgáltatást nyújtó Google Dokumentumokon vagy az Issuun . A Scribd és az Issuu anyagainak egy része regisztráció nélkül is elérhető.

URL: http://www.scribd.com/

Szőke József

Hello Window! - 3. rész

GTK+/gtkmm programozás GNU/Linux alatt

A GTK+ (GIMP Toolkit) egy C nyelven -ám objektum-orientált megközelítéssel- íródott, grafikus felhasználói felületek (GUI) létrehozására használatos alkalmazás-programozási interfész. A gtkmm nem más, mint ennek a függvénykönyvtárnak a C++ változata, pontosabban fogalmazva wrappere. Mindkét terjesztése LGPL licenc alatt történik, így bátran felhasználható mind szabad/ingyenes, mind kereskedelmi szoftverek létrehozására.
A most kezdődő sorozat célja az abszolút kezdetektől indulva bemutatni a GTK+ és a gtkmm hasonlóságait, különbözőségeit, sajátosságait eljutva egy olyan szintre, ahol remélhetőleg a több éves tapasztalattal rendelkező fejlesztők is találnak hasznos, megfontolásra érdemes ötleteket, információkat.

Bevezetés

Ebben a részben a szignálkezelés alapjait ismertetjük és ennek tekintetében vesszük számba a GTK+ és a gtkmm közötti hasonlóságokat és különbözőségeket. Ehhez egy új példprogramot veszünk górcső alá, mely forráskódját tekintve némiképp ugyan bonyolultabb az előző részben tárgyalttól, működésére nézve azonban nem sokban különbözik tőle.

Fogalmak

A korábban már megismert alapfogalmak újra előkerülnek majd, most azonban általános ismertetésük helyett a jelen témánkhoz kapcsolódó specifikumaikat vázoljuk fel.

Main Loop

Ahogy arról az előző részekben szó esett - minden más felületprogramozási nyelvekhez hasonlóan - a GTK is eseményvezérelt (event-driven). De mit is jelent ez? Azt, hogy felhasználói interakciók híján - figyelmen kívül hagyva az ütemezett eseményeket és néhány későbbi részekben részletezendő funkciót - a GTK a main loopban áll és várakozik valamiféle eseményre (event), mint az egér megmozdítása, vagy egy kattintás, esetleg egy billentyűleütés. A main loopba történő belépésre szolgál GTK+ esetén a gtk_main(), míg gtkmm esetén a Gtk::Main::run() függvény. Ha az említett akciók közül valamelyik bekövetkezik, az addig várakozó ciklus úgymond ``felébred", majd a bekövetkezett eseményt propagálja - kvázi üzenetet küld - a megfelelő widget(ek) felé.

Signal

Ez a közvetítő üzenet (jel, jelzés) a szignál. Ilyen üzenetből számos létezik, hisz az egyes widget típusokon különböző események lehetnek értelmezettek, gomb esetén a rá történő kattintás, egy legördülő menünél az egér menüelem fölé történő mozgatása, míg egy widgetnél annak átméretezése. Minden ilyen eseménynek megvan a saját neve, mellyel hivatkozni lehet rá (pl.: ``button-pressed", ``enter-notify-event``, "size-request``). Itt érdemes megjegyezni, hogy a szignálok örklődnek, azaz egy specifikus widget, mint amilyen mondjuk egy RadioButton, vagy egy CheckButton, minden olyan szignállal rendelkezik,

amivel őse a Button, vagy akár annak az őse az a Widget típus. A szignálok egyrészről arra szolgálnak, hogy a GTK rendszerén belül az egyes widgetek egymással kommunikálhassanak. Ha például egy gombot lenyomunk, akkor azt (illetve annak részeit) újra kell rajzolni, ha egy menüelemet kiválasztunk, azt át kell színezni, illetve az esetleges almenüpontokat ki kell rajzolni, míg átméretezésnél az egyes widgetek helyigényét újra kell számolni. Másfelől ha a program írói valamely esemény bekövetkezéséről értesülni szeretnének, megadhatnánk eseménykezelő függvényeket, melyek ezen esetekben meghívódnak.

Callback

Ezen függvények elnevezése a GTK+ terminológiában callback. Az egyes konkrét prototípusok a szignál fajtájától függenek. A C nyelvű változat esetén első paraméterük jellemzően az a Widget - pontosabban szólva Object, hiszen a szignálkezelés ezen a szinten került implementásra a Glib-ben - melyen az esemény kiváltódott. Ezt a paramétert követik a szignálhoz kapcsolódó egyéb jellemzők, az utolsó pedig a szignál bekötésekor megadott, úgynevezett user data, amiről a példaprogram kapcsán részletesebben szólunk. Elöljáróban csak annyit, hogy ez egy meglehetősen kényelmetlen és gyakorta nehézkesen használható megoldás, melyre a C++ nyelvű változat remek alternatívát kínál.
Ennyi bevezető után lássuk az "ehavi`` példaprogramunkat, majd annak értelmezését.

Szignálkezelés dióhéjban

Az előző szám módszertanától eltérve az alábbiak szerint elemezzük a kódokat:
külön-külön vesszük számba ez egyes nyelvi változatok sajátosságait, mivel ezen példaprogramok kódjának hasonlósági foka szerény
először a C nyelvű verziónak fogunk neki, hogy túl lehessünk a nehezén, azután meglátjuk mennyiben más a helyzet, ha gtkmm használatára van módunk GTK+ helyett
ezt követően egy külön rész foglalkozik a két verzió összehasonlításával
a kódot nem sorfolytonosan, hanem a futás logikája szerint követjük, lévén egy kicsit is bonyolultabb esetben - mint amilyennek ez is mondható - már ez a logikusabb
Az alábbi példaprogramok C/C++ nyelvű forrásfájljai, illetve azok eredetijei, a FLOSSzine, valamint a GTK+/gtkmm oldalain az alábbi linkeken érhetőek el:

http://www.flosszine.org/sources/gtk_signal.c
http://www.flosszine.org/sources/gtkmm_signal.cc
http://library.gnome.org/devel/gtk-tutorial/stable/c39.html#SEC-HELLOWORLD
GTK+
A kód a [kód 1] dobozban olvasható!

Részletek sorról sorra

27-28

Annak szükségességéről, hogy a változók egy függvény elején legyenek deklarálva az előző részben esett szó, így erre itt nem térnünk ki. Ellenben fontos, hogy a változók típusa GtkWidget, a specifikusabb GtkWindow, illetve GtkButton helyett. Ennek megyarázata, hogy minden olyan függvény a GTK+-ban mely egy új widgetet hozhatunk létre - azaz a _new végű metódusok - egy GtkWidget típusú objektumra mutató pointerrel tér vissza. Ennek kényelmi okai vannak, jelesül, hogy elkerülhessük a folytonos típuskényszerítéseket, hisz legtöbbször olyan függvényeket használunk, melyek GtkWidgeteket kezelnek, tehát ilyen típusra mutatót vesznek át első paraméterként. Ha egy specifikus - mondjuk GtkButton-t

[Kód 1]

```
1 #include <gtk/gtk.h>
2
3 static void hello( GtkWidget *widget,
4 gpointer data )
5 {
6 g_print ("%s\n", (const char *) data);
7 }
8
9 static gboolean delete_event( GtkWidget *widget,
10 GdkEvent *event,
11 gpointer data )
12 {
13 g_print ("delete event occurred\n");
14
15 return TRUE;
16 }
17
18 static void destroy( GtkWidget *widget,
19 gpointer data )
20 {
21 gtk_main_quit ();
22 }
23
24 int main( int argc,
25 char *argv[] )
26 {
27 GtkWidget *window;
28 GtkWidget *button;
29
30 gtk_init (&argc, &argv);
31
32 window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
33
34 g_signal_connect (G_OBJECT (window), "delete_event",
35 G_CALLBACK (delete_event), NULL);
36
37 g_signal_connect (G_OBJECT (window), "destroy",
38 G_CALLBACK (destroy), NULL);
39
40 gtk_container_set_border_width (GTK_CONTAINER (window), 10);
41
42 button = gtk_button_new_with_label ("Helló Világ");
43
44 g_signal_connect (G_OBJECT (button), "clicked",
45 G_CALLBACK (hello), NULL);
46
47 g_signal_connect_swapped (G_OBJECT (button), "clicked",
48 G_CALLBACK (gtk_widget_destroy),
49 G_OBJECT (window));
50
51 gtk_container_add (GTK_CONTAINER (window), button);
52
53 gtk_widget_show (button);
54 gtk_widget_show (window);
55
56 gtk_main ();
57
58 return 0;
59 }
```

kezelő - függvényt akarunk hívni, akkor vagy fordítási, vagy ami javasol, futás idejű típuskényszerítés alkalmazandó.

32

Főablakunk létrehozása, ahogy eddig is.

34

Az első szignálbekötés. Viszonylagos egyszerűsége ellenére számos apróságra érdemes figyelmet fordítani. Az első maga a g_signal_connect, ami függvénynek tűnhet, pedig ugyanúgy, mint a g_signal_connect_swapped, makró, melyek a g_sinal_connect_data függvényt burkolják. A soron következő érdekesség a G_OBJECT makró, mely futás idejú típusellenőrzést hajt végre a neki megadott paraméteren, majd egy GObject típusra mutatóval tér vissza. A megrögzött C++ programozók joggal kérdezhetik, mi szükség erre, hisz egyfelől majd elvégzi a típusellenőrzést a fordító, meg hát a GtkWindow típus úgy is leszármazottja a GObject "osztálynak``. Ez így is lenne, na de ez itt C, tehát ős-, illetve származtatott osztályokról csak logikailag lehet szó, a típusellenőrzés tehát nem végezhető, sőt minden esetben a hívott függvénynek megfelelő típuskényszerítő makrót kell alkalmazni. A második paraméter a a szignál neve, jelen esetben, mellyel azt adjuk meg, hogy az előző paraméterként megadott object melyik szignáljára is szeretnénk callbacket kötni. Harmadik paraméter azon függvény címe, amelyik meghívódását ki szeretnénk váltani az esemény bekövetkezésekor. A függvénynevet itt is egy makró segítségével adjuk át, ami az előzőekhez hasonlóan C nyelvi hiányosságokra vezethető vissza. Mivel a meghívandó callbackek prototípusai igen sokfélék lehetnek (ami magából a példából is látszik valamelyest és ezek mind külön típusnak minősülnek a C nyelvben ezért ahány féle callback variáció létezik, annyiféle g_signal_connect függvényre lenne szükség. Könnyen belátható, hogy a jogos lustaság más irányba vitte a GTK fejlesztőit. A G_CALLBACK tulajdonképpen egy fordítási idejű típuskényszerítés egy általános függvénytípusra, amivel ugyan egyre megoldottuk, hogy csak egyetlen g_signal_connect_data függvényre legyen szükség, de elvesztettünk minden nemű típusbiztosságot. Ha például egy az adott szignálnak nem megfelelő típusú függvényt adunk meg paraméterként, amit a példabeli függvénynevek felcserélésével könnyen megtehetünk, csúnya meglepetésekben lesz részünk, de csak futásidőben. Az utolsó paraméter az úgynevezett user data, ami arra szolgál, hogy az eseménykezelő függvényünknek, olyan adatokat adjunk át, melyek az esemény megtörténtéből nem következnek. Ilyenek lehetnek például más widgetek címe, ahogy azt látni is fogjuk. Ez esetben az átadott paraméter NULL, ami szintén egy makró mely egy jól nevelt ((void*) 0) kifejezésre fejtődik ki C kód esetén. Zárszóként ehhez a sorhoz csak annyit, hogy a delete_event eseményt az ablakkezelő váltja ki, akkor, amikor az ablakot valamilyen módon (billentyűzet, menü, egér) bezárjuk.

9

Ez a delete_event szignálkezelő függvénye, aminek egy gboolean érékkel kell visszatérnie. Ha ez az érték 0, akkor a GTK folytatja a szignál kezelését, azaz meghívja a gtk_widget_destroy függvényt az ablakunkra. Ha azonban a visszaetérési érék nem 0, ahogy itt sem az, akkor ezzel arra utasítjuk a GTK-t, hogy a szignál további feldolgozásától tekintsen el, aminek révén elérhetjük, hogy hiány nyomogatjuk a jobb felső sarokban a x-et ennek hatására bizony szinte semmi nem történik. Itt érdemes felhívni a figyelmet arra, hogy mivel a C nyelvben, a C++-al ellentétben, nincs bool típus annak analógiájára definiálták a gboolean típust (ami tulajdonképpen egy int) és annak két értékét makróként (TRUE, FALSE).

37

Az előzőhöz teljesen hasonló, annyi eltéréssel, hogy itt a destroy szignálra kötünk be

eseménykezelőt, ami akkor hívodik meg, ha a window változóval meghívódik a gtk_widget_destroy függvényt. Ez több módon is lehetséges. Egyfelől direkt módon egyszerű függvényhívással, amire ebben a kódban nincs példa, másrészről indirekt módon, amire viszont van, erre lesz jó a 47 sor. Illetve lenne még egy út, amit ebben a példakódban szintén elkerülünk.

18

A destroy szignál kezelése általánosságban nézve ritka és itt is csak az a szerepe, hogy a program valamilyen módon ki tudjon lépni. Ha ez a függvény meghívódik, akkor az ablakunk épp megszűnő félben van. Ha ez az eset áll is fenn a programunk futása annak ellenére sem érne véget, hogy az ablakunk bezáródik, hiszen a main loopból nem lépnénk ki. Ezen helyzet elkerülésére ebben az eseménykezelő függvényben meghívjuk a main loopból való kilépésre szolgáló gtk_main_quit-ot.

42

Egy nyomógom létrehozása.

44

Eseménykezelő függvény bekötése a gomb clicked szignáljára, ami a gomb lenyomásakor hívódik meg.

47

A 44. sortól csak a meghívandó eseménykezelő függvényben, illetve az annak átadandó paraméterekben sorrendjében tér el. Ahogy az a függvény nevéből (g_signal_connect_swapped) következik, arról van szó, hogy a gomb lenyomásakor meghívandó callback - jelen esetben a gtk_widget_destroy - paramétereiben a user_data, illetve az az object, amin az esemény kivátótik felcserélésre kerül. Kicsit konkrétabban fogalmazva a user_data lesz a callback első paramétere és a gomb a második. Mivel itt a callback a gtk_widget_destroy függvény, ami paraméterként a destruálandó widgetet várja, a user_data pedig az ablakunk, nem nehéz kitalálni, hogy a gombra való kattintás eredményeként az ablak meg fog szűnni, de csak azután, hogy a Helló Világ üzenet megjelent a konzolban.

3

A fenti állítás csak azért igaz, mert a hello függvény, mint eseménykezelő előbb kerül felkötésre, mint a gtk_widget_destroy, valamint azért, mert az eseménykezelők alapvetően a felkötés sorrendjében kerülnek meghívásra. Fordított esetben előbb hívódna meg a destroy az ablakra, ami -sok egyéb mellett - leköti az eseménykezelő függvényeket.

51

A nyomógomb hozzáadása az ablakhoz.

53,54

A létrehozott widgetek megjelenítése.

56

Belépés az eseményvezérelt szakaszba.

Fentiek ismeretében nagy biztonsággal jósolhatjuk meg példaprogramunk működését. Az elindított alkalmazás egy ablakot jelenít meg, melyben egy Helló Világ feliratú gomb lesz. Az ablak bezárásával hiába próbálkozunk egér, vagy billentyűzet segítségével, ezen kísérletek eredmény csupán egy-egy "delete event occurred`` sor a konzolban. Ha azonban le találnák nyomni gombunkat az ablak hirtelen eltűnik a konzolban egy "Helló Világ`` felirat jelenik meg és a program kilép. Lássuk, hogy érhetünk ehhez teljesen hasonló funkcionalitást C++-ban.

gtkmm

A kód [kód 2] dobozban olvasható!

[Kód 2]

```
1 #include <gtkmm.h>
 2 #include <iostream>
 3
 4 static void hello(const Glib::ustring &hello_msg)
 5 {
 6 std::cout << hello_msg << std::endl;
 7 }
 8
 9 class MyWindow : public Gtk::Window
 10 {
 11 private:
 12 Gtk::Button button;
 13
 14 protected:
 15 virtual bool on_delete_event(GdkEventAny *event)
 16 {
 17 std::cout << "delete event occurred" << std::endl;
 18
 19 return true;
 20 }
 21
 22 void on_button_clicked()
 23 {
 24 Gtk::Main::quit();
 25 }
 26
 27 public:
 28 MyWindow() :
 29 button("Helló Világ")
 30 {
 31 button.signal_clicked().connect(sigc::bind(sigc::ptr_fun(hello), "Helló Világ"));
 32 button.signal_clicked().connect(sigc::mem_fun(*this, &MyWindow::on_button_clicked));
 33
 34 add(button);
 35
 36 button.show();
 37 }
 38 };
 39
 40 int main(int argc,
 41 char *argv[])
 42 {
 43 Gtk::Main kit(argc, argv);
 44
 45 MyWindow window;
 46
 47 Gtk::Main::run(window);
 48
 49 return 0;
 50 }
```

Részletek sorról sorra

43-47

Az korábbi számokban megjelent példaprogramokhoz képest egyetlen eltérést vehet észre a figyelmes olvasó. Mégpedig azt, hogy Gtk::Window helyett MyWindow típust használunk

főablakunk létrehozásához. Mivel azonban a MyWindow publikusan származik a Gtk::Window típusból ez a gtkmm számára nem jelent különbséget. A C változathoz képest a számaztatás itt nem csupán "logikai", vagyis minden a C++-an megszokott előny könnyedén realizálható. Erre példa, hogy a származtatás miatt nincs szükség semmilyen típuskényszerítésre mikor a Gtk::Main::run hívjuk, ami pedig egy Gtk::Window referenciát vesz át paraméterként.

28-37

Saját osztályunk konstruktorában megtehetjük mindazokat a lépéseket, melyeket a C nyelvű változat esetén a main függvényben voltunk kénytelenek implementálni, úgy is mint a szignálok felkötése, a gomb hozzáadása az ablakhoz, a widgetek megjelenítése. Az egységbezárás ezen előnyén túl a származtatásból fakadó örömöket is élvezhetjük, ugyanakkor persze az ebből fakadó kötelességeknek is eleget kell tenni. Ez esetben ez a konstruktor meghívását jelenti, ami rejtett módon megy végbe. Az ősosztály konstruktorának explicit hívása hiányában a Gtk::Window azon konstruktora fut le, mely paraméterek nélkül is hívható. Másrészről viszont az adattagként tárolt buttont is inicializálnunk kell. Itt is lehetne közvetve implicit módon hívni a paraméter nélküli konstruktort, azonban kézenfekvőbb azt a változatot használni, amivel egyszerre a gomb feliratát (label) is megadhatjuk, így egy hívás a későbbiekben megspórolható. Külön szót érdemelnek a szignálok bekötései. Különösebb programozó géniusz nem kell, hogy felfedezzük a szignálok eléréséhez egy signal_szignálnév nevű tagfüggvény meghívását használhatjuk fel. Az ilyen formájú függvények egy, a Glib::SignalProxyBase-ből származó, osztályt adnak vissza, melyek connect nevű metódusai valósítják meg azt, amit a GTK+ esetén a g_signal_connect makró tett meg. Előnye ennek a módszernek, hogy típusbiztos, azaz a connect paraméterként csak olyan függvényt (slot) fogad el, aminek a típusa megfelel az adott szignál kezeléséhez. További előny, hogy a slotokhoz nem csupán egy user data csatolható, hanem tetszés szerinti, s ezek típusa is ellenőrzésre kerül fordításkor. Amennyiben azonban sikerül csupán egy apróságot is félreírnunk a szignál bekötésénél, vagy a slot típusának megadásánál, a C++-is template-ekkel történt megvalósításnak hála akár több oldalas, nehezen kibogarászható hibaüzenetekkel találhatjuk magunkat szemben.

31-32

Lássuk akkor miként is érhető el ugyanaz gtkmm-ben, mint ami korábban GTK+-ban. Első pillantásra is szembeszökő, hogy mindkét sorban találunk olyan hívást, melyek nem a Gtk névtérben definiáltak. Ennek oka, hogy a gtkmm-ben a szignálkezelést libsigc++ nevű függvénykönyvtárral valósították meg, bár az ilyen hívások jellemzően csak a szignálok felkötésekor kerülnek elő. A két tárgyalt sor közötti eltérés könnyen indokolható, ha figyelembe vesszük a megadott eseménykezelő függvények típusát.

31

Ha a megadni kívánt függvény statikus - függetlenül attól, hogy osztályhoz tartozik-e vagy azon kívül definiált, esetleg tisztán C nyelvű kódból származik - slot létrehozásához a sigc::ptr_fun alkalmazandó. Ebben a konkrét esetben a slot létrehozásán túl, paramétereket is hozzákapcsolunk a clicked esemény bekövetkeztekor meghívandó függvényhez. Ennek eszköze a sigc::bind, melynek első paramétere egy slot, a továbbiak pedig a csatolandó paraméterek. Itt csupán egy ilyen van, a gomb lenyomására kiírandó üzenet szövege. Ez persze kissé kényszeredett, hiszen a paraméter értéke soha nem változik, így ennek igazi hasznát ebből a példából nehéz belátni.

4-7

Statikus függvényük a lehető legegyszerűbb csupán azt szemlélteti miként is kell az átadott paraméterekez használni, ezesetben annak értékét a console-ra kiíratni, működését és

funkcióját tekintve a C-s változat azonos nevű függvényével analóg.

32

Ha a megadni kívánt eseménykezelő egy osztály tagfüggvénye, akkor a sigc::mem_fun használható arra, hogy slotot hosszunk létre belőle, azzal a különbséggel, hogy az osztály példányának címe lesz az első paraméter és csak ezt követi a függvény címe. Fontos, hogy a függvényt teljes névtérlistával együtt kell megadni. Természetesen a korábban megismert sigc::bind it is alkalmazható.

22-25

Ez a függvény sem több csupán, mint egy egyszerű példa, ugyanazt a célt szolgálja, mint a GTK+-os verzió azonos nevű függvénye.

15-20

Ezen függvény megértésének kulcsa a virtual kulcsszóban rejlik. Minden szignálhoz tartozik ugyanis egy - az adott widget által implementált - kezelő függvény, mely alkalmasint felülbírálható (ovverride). Ha ezt megtesszük, azzal a szignál kezelésének teljes folyamatát mi irányítjuk, ami mellett komoly érvek szólhatnak, de nem árt körültekintőnek lenni. Ebben az esetben a cél pont annak demonstráljuk, szabad kezet ad a gtkmm abban, hogy egy származtatott widget miként kívánja kezelni az ősosztály eseményeit. A visszatérési érték szerepe ugyan az, mint az előző példában.

A szignálkezelésről összegzésképpen annyit, hogy alapvetően két lehetőség kínálkozik arra,. hogy az egyes widgetek eseményei kezeljük:

Callbackeket kapcsolni azon widgetek azon eseményihez, melyek számunkra érdekesek és ezekben megtenni a megfelelő lépéseket

Felülbírálni a widget saját eseménykezelőjét az öröklődés mechanizmusai útján. Erre mindkét változat esetén van lehetőség, ám a GTK+ megoldása kissé körülményes és nehezebben megérthető, így annak ismertetése valmely későbbi részre marad. A C++ nyelvi eszközeit kihasználva a gtkmm viszont ezt oly könnyedén oldja meg, hogy kár lett volna kihagyni a bemutatást annak ellenére is, hogy a módszerre ritkán van szükség, hiszen többnyire arról van szó, hogy a különböző widgetpéldányok azonos szignáljainak kiváltódásakor más-más irányba szeretnénk terelni a program futását. A felülbírálás révén viszont arra nyílik lehetőség, hogy a szignál kezelésének módját változtassuk meg. Ha nem kívánunk egyebet tenni, mint ami amúgy is történne, hívjuk meg a felülbírált függvény szülőosztálybeli változatát. Ha azonban ez előtt, vagy után még valami mást is tenni szeretnénk, megtehetjük, hogy csak a függvény közepéről hívjuk a szülő metódusát, vagy akár el is hagyhatjuk az ha tudjuk mit és főként hogyan szeretnénk kezelni.

Egy "nyomkodható" ablak

Fordítás és linkelés

A korábbiakhoz hasonlóan az alábbi parancssorok segítségével fordíthatóak elemzett programjaink:

```
gcc gtk_signal.c -o gtk_signal `pkg-config -cflags -libs gtk+-2.0
g++ gtkmm_signal.cc -o gtkmm_signal `pkg-config gtkmm-2.4 -cflags -libs`
```

Futtatás

Próbálkozzunk ezúttal is a ./gtk_window, illetve a ./gtkmm_window paranccsokkal abban a könyvtárban, ahol a fordítást elkövettük.

Eredmény

Bármily hihetetlen ezúttal sem történik semmi egyéb, mint az előző alkalommal. Remélhetőleg azonban a különbség mégis érzékelhető annyiban, hogy legutóbb a meglepetéssel teli borzongást ablakunk váratlan felbukkanása, míg most a bennünk szikraként felvillanó megértés okozza.

Irodalomjegyzék

http://developer.gnome.org/doc/GGAD/ggad.html.
http://library.gnome.org/devel/gtk-tutorial/stable/.
http://gtk.pergamen.hu/.
http://www.gtkmm.org/docs/gtkmm-2.4/docs/tutorial/html/index.html.

Pfeiffer Szilárd

Tudtad-e?

WINPENPACK

Aki sokat utazik, viszont nincs laptopja, és például a munkája során felkeresett különböző intézményekben mindenhol különböző számítógépeken tevékenykedik, annak különösen hasznos a winPenPack projekt megismerése.

A wPP egy olasz kezdeményezés, amely átalakított programokat, illetve programcsomagokat tartalmaz. Ezek néhány könnyű mozdulattal egy pendrive-ra telepíthetőek akár Windows-felhasználóként is, és azon egy szinte teljes operációs rendszernek megfelelő környezetet adnak ki. A 30-60 alkalmazás mindegyike az éppen csatlakoztatott, FAT32-re formázott pendrive-ról futtatható (szinte) bármilyen Windows rendszer felhasználói fiókjában. Se a telepítéséhez, se a működtetéséhez nincs szükség rendszergazdai jogokra, vagy a pendrive-ról történő bootolásra, és így a BIOS átállítására sem. Ez azért fontos, mert egy - hasonló képességekre optimalizált - pendrive-on futó Linux (pl. a Slax - vagy UNetbootin-nal telepítve akármilyen más disztribúció) használatához az eszközről való bebootolásra van szükség, amit a rendszergazda letilthat. Vagy egy másik lehetőség, hogy egy Linuxnak a Windows-os felhasználói fiókon belüli elindításhoz az adott gépen előzőleg ugyancsak egy rendszergazda Wubi programot telepít, amely a csatlakoztatott eszközön helyet foglaló, "live" Linuxot elindítja.

A wPP főként nyílt forráskódú alkalmazásainak és néhány (veszélytelen és reklámmentes) free programjának egyike sem hagy nyomot az adott számítógépen, hanem közvetlenül a csatlakoztatott eszközre menti az összes változást (persze képes a számítógépre is menteni, ha kívánjuk), miközben minden gépen a csatlakoztatást követően azonnal a megszokott környezetet és beállításokat biztosítja. A wPP Essential csomag például akár már egy 2GB-os, 2.0-ás szabványú pendrive-ra is felrakható, kb. 30 alkalmazást tartalmaz az irodai programoktól kezdve a böngészőn és a biztonsági alkalmazásokon keresztül a videovágóig, és a telepítés után még 500MB-ot sem foglal el az eszközön.

Jankovich Oszkár

Hello world! - 4.

A temp file-ok használata

A programoknak időnként szükségük lehet átmeneti állományokra (temporary files), nagyobb ideiglenes adatcsomagok tárolása illetve más programokkal történő adatcsere céljára. A rendszer ehhez biztosítja a /tmp könyvtárat, ahol ezeket a file-okat létrehozhatjuk.

Legyünk figyelemmel az alábbiakra a tmp állományok létrehozásakor:

- A programnak több példánya is futhat egyidejűleg, akár ugyanazon akár más user nevében. A fileneveknek különbözniük kell, nem ütközhetnek.
- A file jogosultságait úgy kell beállítani, hogy más user ne tudja megváltoztatni a filet.
- A filenévnek nem szabad megjósolhatónak lennie.

Az átmeneti állományok kezelésére könyvtári függvények is a rendelkezésre állnak, mind az ANSI standard C mind pedig a UNIX rendszerek részéről.

```
#include <stdio.h>
FILE *tmpfile (void);
char *tmpnam(char *s);
```

A tmpfile és a tmpnam az ANSI C részei, a tmpnam-ot biztonsági okok miatt kerüljük, a tmpfile-t akkor használjuk ha nem akarunk más programmal adatot cserélni az átmeneti állományon keresztül.

```
#include <unistd.h>
int mkstemp(char *template);
char *mktemp(char *template);
```

A mkstemp és a mktemp a UNIX C kiterjesztés részei, az mktemp-et szintén kerüljük a biztonsági megfontolások miatt, az mkstemp használatára az alábbiakban adunk példát.

A file nevét az mkstemp függvénnyel állítjuk elő, mely egyúttal megnyitja a filet, beállítja a szükséges jogosultságokat, flag-eket majd visszaadja a file descriptort. Az mkstemp egy template-ből (sablonból) dolgozik, a sablon egy karakter füzér (string) mely XXXXXX - re azaz 6 darab X betűre végződik. Ezt a 6 darab X-et fogja egy egyedi azonosítóval behelyettesíteni a függvény. Ezen templét alatt egy teljes filenevet (path) értsünk, azaz könyvtár útvonallal együtt, tehát ezen a ponton van lehetőségünk pl. a TMPDIR környezeti változót lekérdezni és használni, ehhez lásd a getenv() függvény használatát.

Az mkstemp-pel létrehozott file-ok nem törlődnek automatikusan, erről nekünk kell gondoskodnunk. Ezt a gondoskodást célszerű nem elfeledni, ugyanis a /tmp könyvtár telítődése a rendszer egy részének a leállásával járhat. Amennyiben az átmeneti állomány csak belső (azaz programon belüli) használatra szolgál, úgy célszerű azonnal az unlink() kiadásával előkészíteni a törlést. Az unlink() a könyvtár bejegyzést egyből törli de a filerendszer a file hivatkozásokat számolja (reference counting) és a close() kiadásáig vagy a program kilépéséig valójában nem távolítja el a rendszerből. Ebből kifolyólag az átmeneti állományt nyugodtan használhatjuk, mert még a program szabálytalan (pl. segfault) kilépése estén is azonnal törlődik a file.

Az alábbi program az alapfunkciókra világít rá, természetesen a file műveletek ennél jóval összetettebbek, bonyolultabbak is lehetnek, ez a része teljesen szabadonválasztott a gyakorlatban. A write_temp_file() visszatérési értéke egy file descriptor lesz, amivel a write(), read() és rokon függvényekkel hivatkozhatunk. A függvény argumentumaként átvesszük a kiíratandó állomány címét és hosszát. Létrehozzuk a filenév template-et, ami az előbb említettek szerint lehetne pl. a TMPDIR környezeti változó alapján is előállítva. Az mkstemp függvénnyel létrehozzuk az átmeneti állományt, majd rögtön végrehajtjuk az unlink-et (ne feledjük, ezt nem csinálja meg helyettünk az mkstemp!) majd kiírjuk az oda tartozó adatokat és visszatérünk a fildescriptor értékével. A beolvasó függvény argumentumként kapja meg a filedescriptor értéket, meg a visszaadandó fileméret helyét, visszatérési érték gyanánt a puffer címét adja vissza a függvény. A file kiolvasása után lezárjuk - ezzel töröljük - az ideiglenes állományt.

Amennyiben a C könyvtári I/O függvényeket használjuk és nem kell átadnunk az átmeneti állományt más programnak, úgy a tmpfile függvényt is használhatjuk, ez létrehozza, megnyitja a file-t, végrehajtja az unlink-et és visszaadja a file pointert. Ahogy az close-zal lezárjuk a filet vagy kilépünk a programból, a file ténylegesen törlődik.

Házi feladat: a program fordításához és futtatásához nem adtunk meg külön parancsot vagy információt, az eddigiek alapján végezzük el a fordítást és futtassuk le a programot!

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int
write_temp_file (char *buffer, size_t length)
{
 char temp_filename[] = "/tmp/my_temp_file.XXXXXX";
 int fd = mkstemp (temp_filename);
 unlink (temp_filename);
 printf ("Generált filenév: %s\n", temp_filename);
 write (fd, &length, sizeof (length));
 write (fd, buffer, length);
 return fd;
}

char *
read_temp_file (int temp_file, size_t * length)
{
```

```
char *buffer;
int fd = temp_file;
 lseek (fd, 0, SEEK_SET);
 read (fd, length, sizeof (*length));
 buffer = (char *) malloc (*length + 1);
 buffer[*length] = '\0';
 read (fd, buffer, *length);
 close (fd);
 return buffer;
}

int
main (void)
{
 char *write_buf, *read_buf;
 int temp_fd;
 size_t read_length;

 write_buf = calloc (100, sizeof (char));

 strcpy (write_buf, "teszt");
 temp_fd = write_temp_file (write_buf, strlen (write_buf));

 read_buf = read_temp_file (temp_fd, &read_length);
 printf ("Visszaolvasva: %s\n", read_buf);

 free (write_buf);
 free (read_buf);

 return (0);
}
```

Tudtad-e?

Magyarubi és Fluxbuntu

A magyar közösség legaktívabb tagjainak jóvoltából egyre-másra jelennek meg az újabb Ubuntu-remixek. Januárban Karmestomi adta ki a Magyarubit, amely a 8.10-es verzió kiegészített és azonnal magyarul tudó változata. A Magyarubi az ígéret szerint havonta frissül majd, ami azt is jelenti, hogy az alkalmazáslista - szavazás és kérések alapján - folyamatosan bővül, így a letölthető és telepíthető, vagy live CD-ként használható iso-összeállítás minden hónapban tartalmilag is többet nyújthat. Februárban pedig - szintén az Intrepid Ibexből - Sevoir elkészítette a takarékos Fluxbox ablakkezelővel és magyarítással ellátott Fluxbuntut, amely jóval fejlettebb a 7.10-es verzióra épülő, hivatalosan kiadott Fluxbuntunál. A letölthető CD-kép nincs 280 MB, a mix lapzártánk idején még nem teljesen stabil.

Jankovich Oszkár

Hulladékfeldolgozás nyílt forrású programmal - 1. rész

Aki levelezőszerver építésére adja a fejét, az tudja, hogy a levelek pontos továbbítása ma már nem elég: az e-mailben érkező vírusok, spamek és más malware-ek áradatát is kezelni kell. Ebben az írásban egy magyar fejlesztésű spamszűrőt mutatok be, amellyel közelebbi ismeretségben vagyok. Jó 3 éve már, hogy a Linuxvilágban megjelent egy cikk a clapf nevű spamszűrőről. Az eltelt idő alatt azonban sokat változott a program, és már nagycsoportos lett, úgyhogy ideje egy újabb ismertetőnek.

Minden programnak kell egy név, és én is szembesültem a keresztelő nehézségeivel. A nehéz szülés eredménye lett a clapf, amely az Ubuntuval ellentétben nem egy afrikai kifejezés, hanem egyszerűen a „clamav for postfix” rövidítése. A spamszűrő ugyanis kezdetben egy egyszerű antivírus interfész volt a clamavhoz, hogy megvédjem az postafiókomat a levélben érkező vírusoktól. A program egy localhost-on futó démon (volt már akkor is), amelynek a postfix átadta a levelet, az megnézte, hogy van-e benne vírus, aztán visszaadta a postfixnek, és így került a tiszta levél az inboxomba. Ha vírusos levél érkezett, akkor azt eldobta, pontosabban 550-es SMTP kóddal elutasította, aminek hatására a postfix visszaküldte (bounce) a problémás levelet.

Ez megfelelően működött egy ideig, azonban a spam probléma kezdett elhatalmasodni, és be kellett látnom, hogy nem működik a kulcsszó alapú spamszűrés, ezért valami jobbat kerestem. A keresés eredménye pedig a bogofilter lett. Ez ugyan jól működött, de fájlaltam, hogy nem illeszthető a postfix-hez, hogy démonként működjön. Ezzel többen is így voltak akkoriban, és volt is ígéret, hogy egyszer majd talán, hátha, de amennyire tudom, a mai napig nem lett belőle semmi. Ezért úgy döntöttem, hogy a saját szórakoztatásomra és megelégedettségemre írok egy saját spamszűrőt.

Amikor a hogyanra került a sor, belebotlottam Paul Graham 'Plan for spam' című esszéjébe, amelyben a bayes-i spamszűrőkről ír, és ahogyan sokan másokat, engem is azonnal meggyőzött. A clapf azonban nem bayes-i spamszűrő – bár sokáig azt hittem – hanem az inverz khi-négyzet algoritmust használja, amit Gary Robinson hozott a szélesebb nyilvánosság elé. Ez egyébként szintén egy statisztikai algoritmus, amely a minimális fals pozitív hibára van kihegyezve.

Miért érdemes bárkinek statisztikai spamszűrőt használni? Azért, mert ezek a programok okosak: megtanulják, hogy milyen leveleket szeretünk, és milyeneket nem. Ebből az információból felépítenek egy ún. token adatbázist, és minden tokenhez (az egyszerűség kedvéért gondoljunk a tokenekre úgy, mint a levélben lévő szavakra) egy valószínűséget rendelnek. Amikor beérkezik egy levél, akkor az adatbázisból lekérdezi az egyes szavak spam valószínűségét, majd oszt és szoroz, a végén pedig kidobja az eredményt, hogy a levél jó (ham), vagy rossz (spam).

A statisztikai spamszűrők legnagyobb előnye az egyéb megoldásokkal szemben az extrém pontosság. Nem találkoztam még olyan emberrel, akit a statisztikai spamszűrője cserben hagyott volna, és az átlagos pontossága 99% alá esett. Egy tetszőleges spamszűrőt pedig végső soron a pontossága igazol, hogy az esetek hány százalékában egyezik a döntése a miénkkel.

A clapf a projekt honlapjáról szerezhető be, a http://clapf.acts.hu/ címről. A program moduláris, ami konkrétan azt jelenti, hogy mi magunk dönthetjük el, hogy milyen funkciókat fordítunk bele. Ha pl. csak vírusok ellen akarjuk védeni a felhasználókat, akkor elég egy támogatott antivírus terméket választani a listából (jelenleg a Dr.Web, a Kaspersky és az avast! támogatott a clamav mellett). Ha spamszűrésre van szükségünk, akkor választhatunk, hogy hol akarjuk tárolni a tokeneket, ill. a felhasználó információkat, továbbá megadhatjuk, hogy akarunk-e fekete- vagy fehérlistát használni, stb. Ezeket részletesen megnézzük később.

A bő lére eresztett bevezető után konkrét konfigurációk kialakításával ismertetem a clapf képességeit. A bemutatott példákban feltételezem, hogy a felhasználó postfix MTA-t használ. Itt szeretném megjegyezni, hogy a clapf nem csak a postfixet támogatja, erről később bővebben szó lesz.

Először nézzük meg azt az esetet, amikor csak vírusszűrést valósítunk meg a nyílt forrású clamav programmal együttműködve!

Első lépésként hozzunk létre egy felhasználót, aki a clapf démont futtatni fogja:

```
groupadd clapf
useradd -g clapf -d /usr/local/var/lib/clapf -s /bin/sh clapf
usermod -L clapf
```

A program kicsomagolása után futtassuk a „`./configure --enable-clamd`” parancsot! Az installálás a szokásos „make install” utasítással történhet. A telepítés után létrejön az `/usr/local/var/lib/clapf` könyvtár is (hacsak a --localstatedir változót másként nem adjuk meg), amely alatt tárolja a clapf a futása közben használt ideiglenes állományokat.

A clapf nem csak arra képes, hogy a clamd-nek átadja a levelet TCP/IP vagy Unix domain socketen keresztül, hanem a libclamav könyvtár linkelésével közvetlenül – tehát a clamd kihagyásával - is használni tudja a clamav-ot. Ha ezt akarjuk, akkor használjuk az `--enable-libclamav configure` opciót.

A telepítés után létrejön a `${prefix}/etc/clapf.conf` fájl (általában a `/usr/local/etc/clapf.conf`), amelyet kedvünkre módosíthatunk, amit bőséges megjegyzések segítenek, azonban az alapértelmezett értékek egy jó kiindulási alapot biztosítanak.

A clapf alapesetben a 127.0.0.1-en figyel a tcp/10025-ös porton, itt várja a postfix felől érkező SMTP forgalmat, amit a 127.0.0.1-en futó tcp/10026-os porton ad vissza. Ezek nem kőbe vésett értékek, a konfigurációs fájlban módosíthatjuk. Itt adhatjuk meg a clamd socket-jének az útvonalát, vagy a TCP port paramétereit is. A clamav könyvtár használata esetén pedig néhány ezzel kapcsolatos paramétert is beállíthatunk, pl. tekintse vírusosnak a titkosított archív mellékleteket, stb.

A postfix nagyon kellemes szolgáltatása, hogy lehetővé teszi a tartalomszűrők számára, hogy SMTP protokoll segítségével kommunikáljanak vele. Ha azonban a clapf szintén beszéli az SMTP protokollt, akkor minek kell elé a postfix? Azért, mert a clapf nem egy általános célú MTA az ehhez szükséges rengeteg komplikált funkcióval, hanem egy speciális tartalomszűrő alkalmazás, amely SMTP ill. LMTP protokoll segítségével kommunikál a külvilággal.

Telepítés után a clapf spamszűrőt a /usr/local/libexec/clapf/check_clapf.sh shell scripttel indíthatjuk el. Utolsó lépésként a postfixnek meg kell mondani, hogy használja a tartalomszűrőnket. Módosítsuk a postfix main.cf ill. master.cf konfigurációs állományait!

/etc/postfix/main.cf:

```
content_filter = smtp:[127.0.0.1]:10025
default_destination_concurrency_limit = 10
smtpd_recipient_limit = 128
lmtp_send_xforward_command = yes
smtp_send_xforward_command = yes
smtpd_authorized_xforward_hosts = 127.0.0.0/8
```

Az első sor azt mondja meg, hogy a postfix a leveleket a 127.0.0.1/10025-ön figyelő tartalomszűrőnek adja át SMTP protokollal (itt egyébként LMTP protokollt is használhatunk). A 2. sorban megadhatjuk a paralel kézbesítések számát. A 3. sor limitálja az egy SMTP session-ben megadható címzettek számát. Ha ezt az értéket 128 fölé akarjuk növelni, akkor hozzá kell igazítanunk a MAX_RCPT_TO változót a config.h állományban. A 4-5. sorok hatására pedig az ún. XFORWARD információkat is átadja a tartalomszűrőnek a postfix. Ez azért jön jól, mert a clapf csak a localhost IP-címét látja, így azonban tudja a levelet nekünk passzoló gép nevét és IP-címét, amit össze lehet vetni feketelistákkal, stb.

/etc/postfix/master.cf:

```
127.0.0.1:10026 inet n - n - 10 smtpd
      -o content_filter=
      -o receive_override_options=no_unknown_recipient_checks
      -o smtpd_helo_restrictions=
      -o smtpd_client_restrictions=
      -o smtpd_sender_restrictions=
      -o smtpd_recipient_restrictions=permit_mynetworks,reject
      -o mynetworks=127.0.0.0/8
      -o smtpd_authorized_xforward_hosts=127.0.0.0/8
```

Ez azt mondja a postfix-nek, hogy a localhost/10026-on is figyeljen. A 'postfix reload' parancsot kiadva hátradőlhetünk, a szerkezet ketyeg. Természetesen meg kell oldani a clamav vírusadatbázisának rendszeres frissítését is. Ha a libclamav-ot használjuk, akkor az adatbázis frissítése után küldjünk egy ALARM szignált a clapf-nak, hogy újratöltse a vírusadatbázist.

```
/usr/local/bin/freshclam --quiet --user clamav --datadir=/usr/local/share/clamav
```

```
if [ $? -eq 0 ]
then
killall -ALRM clapf
fi
```

Ha kíváncsiak vagyunk, hogy mit csinál a program, akkor állítsuk be a verbosity=5 értéket, majd egy HUP szignál küldése után nézzük meg a maillog fájlunkat. Küldjük egy vírusos levelet tesztként, amelynek hatására az alábbihoz hasonló syslog bejegyzéseket láthatunk a maillog fájlban.

```
Feb 8 13:42:18 mail clapf[2706]: 498f51baa5cc364c87fe920c576212: virus scanning
done in 23 [ms]
Feb 8 13:42:18 mail clapf[2706]: 498f51baa5cc364c87fe920c576212: Virus found
Worm.Sober.G
Feb 8 13:42:18 mail postfix/qmgr[1832]: DCE20928CE0: removed
Feb 8 13:42:18 mail postfix/smtp[2705]: DCE20928CE0: to=<sj@xxxx.hu>,
relay=127.0.0.1[127.0.0.1]:10025, delay=0.11, delays=0.04/0.02/0.01/0.05,
dsn=2.0.0, status=sent (250 Ok 498f51baa5cc364c87fe920c576212 <sj@xxxx.hu>)
```

A clapf a levelekhez egy MD5 hash-szerű queue azonosítót társít. A naplórészletből kiderül, hogy felismerte a levélben szereplő Sober.G nevű férget. Mivel SMTP-n beszélünk a postfix-szel, ezért visszaadja neki 498f51baa5cc364c87fe920c576212 queue azonosítót, de a levelet eldobja, és nem adja vissza a postfix-nek. A clapf alapértelmezetten - naplózás után - eldobja a fertőzött leveleket. A clamav kellemes tulajdonsága, hogy több adathalász kísérletet is felismer, és vírusként azonosítja azokat.

Lehetőség van arra is, hogy a fertőzött leveleket karanténba helyezzük. Ehhez csak annyit kell tennünk, hogy értéket adunk a quarantine_dir változónak. Ha a localpostmaster paraméternek is adunk értéket, akkor a clapf minden vírusos levél érkeztéről levelet küld az itt megadott email címre. Ez adott esetben sok (mondjuk ki nyíltan: rengeteg) levelet eredményezhet, ezért ez a funkció alapértelmezésben ki van kapcsolva. Végül pedig a `silently_discard_infected_email=0` beállítással az alapértelmezett eldobás helyett 550-es hibaüzenetet küld vissza a postfix-nek, amely ennek hatására visszapattintja (bounce) a levelet a feladónak. Mivel a vírusos levelek jellemzően hamisított feladóval érkeznek, ezért nem ez a default beállítás.

A következő példában adjunk hozzá spamszűrést a SpamAssassin segítségével! Ehhez a clapf oldalán nem kell módosítani semmit, és feltételezve, hogy az SA már telepítve és konfigurálva van a gépünkön, csak az alábbi módosítást kell tenni a master.cf fájlban:

Módosítsuk az „smtp” nevű szolgáltatást, hogy így nézzen ki:

```
smtp inet n - n - - smtpd
      -o content_filter=spamassassin
```

Majd a master.cf végéhez adjuk hozzá az alábbi sorokat:

```
spamassassin unix - n n - - pipe
      user=spamc argv=/usr/bin/spamc -f -e
      /usr/sbin/sendmail -oi -f ${sender} ${recipient}
```

Ez utóbbi bejegyzéssel azt tudatjuk a posfix-szel, hogyan hívja meg a spamassassint, az előbbivel pedig azt, hogy minden smtp-n érkező levelet futtasson át a spamassassin nevű filteren.

Ha most egy spam levéllel teszteljük a szűrő kombónkat, akkor az alábbihoz hasonló naplórészletet láthatunk:

```
Feb 8 19:18:48 mail spamd[2344]: spamd: connection from localhost.localdomain
[127.0.0.1] at port 50869
Feb 8 19:18:48 mail spamd[2344]: spamd: setuid to spamc succeeded
Feb 8 19:18:49 mail spamd[2344]: spamd: processing message
<f14f01c89af3$8a784f50$3bc40658@xxxx.hu> for spamc:1005
Feb 8 19:18:56 mail spamd[2344]: spamd: identified spam (12.3/5.0) for spamc:1005
in 7.7 seconds, 2156 bytes.
Feb 8 19:18:56 mail spamd[2344]: spamd: result: Y 12 -
BODY_ENHANCEMENT2,DATE_IN_PAST_96_XX,HTML_MESSAGE,MIME_QP_LONG_LINE,URIBL_JP_SURBL
,URIBL_SC_SURBL,URIBL_WS_SURBL
scantime=7.7,size=2156,user=spamc,uid=1005,required_score=5.0,rhost=localhost.loca
ldomain,raddr=127.0.0.1,rport=50869,mid=<f14f01c89af3$8a784f50$3bc40658@xxxx.hu>,a
utolearn=disabled
Feb 8 19:18:56 mail postfix/pipe[2779]: 665D8928CE2: to=<sj@xxxx.hu>,
relay=spamassassin, delay=8.4, delays=0.02/0.05/0/8.3, dsn=2.0.0, status=sent
(delivered via spamassassin service)
Feb 8 19:18:56 mail clapf[2785]: 498fa0a04bc95516003b5ddc41f780: virus scanning
done in 17 [ms]
to=<sj@xxxx.hu>, relay=127.0.0.1[127.0.0.1]:10025, delay=0.32,
delays=0.02/0.04/0.01/0.25, dsn=2.0.0, status=sent (250 Ok
498fa0a04bc95516003b5ddc41f780 <sj@xxxx.hu>)
```

A levél útját követve megállapítható, hogy a levél először spamszűrésen esett át, majd ezután került a clapfhoz antivírus tesztre. A konfiguráció teljesen jól működik, bár szerencsésebb lenne, ha először azt néznénk meg, hogy van-e vírus a levélben. Ha nagyobb forgalmú levelezőkiszolgálót építünk, akkor a rengeteg feldolgozandó levél könnyedén megdobhatja a gép terhelését.

A folytatásban azt mutatom be, hogy a clapf mire képes, ha elemében van. Építeni fogunk egy kisebb ill. egy nagyobb környezetbe szánt konfigurációt, ahol az egyes funkciókat külön gépekre helyezzük. Szó lesz még a 0.4.x sorozatban debütáló házirendszabályokról (policy groups), amelyekkel nagyon rugalmasan, akár felhasználónként testre lehet szabni a clapf működését, ill. megnézzük a clapf PHP-ben írt webes menedzsment felületét. Végül lerántom a leplet néhány félreértésről, FUD-ról vagy egyéb városi legendáról a statisztikai spamszűrőkkel kapcsolatban. Akit a téma mélyebben érdekel, annak pedig ajánlom figyelmébe a SZAK Kiadó gondozásában 2008-ban megjelent SPAMtelenül című könyvemet.

Sütő János

Feel the SUN shine

azaz OpenSolaris testközelből

„Hol volt hol nem volt...”

A SUN Microsystems egy olyan mamutcég, melyről mindenki hallott már, aki csak egy kicsit is otthon van a számítástechnika világában. Azt azonban kevesen tudják, hogyan is kezdődött mindez.

Az első SUN gép a Sun1 (hihetetlen elnevezés) a Stanfordi Egyetemnek készült, és CAD munkaállomás szerepét látta el. Motorola68000-as processzora volt és virtuális memória kezelésére alkalmas Unix alapú OS futott rajta.

1982 február 2-án Vinod Koshla, Andy Bechtolsheim, és Scott McNealy megalapítják a Sun Microsystems-t. Később Bill Joy, a BSD egyik fejlesztője is csatlakozik az alapítókhoz. Maga a SUN egy mozaikszó, Stanford University Network. A cég már az alapítás évében nyereséges lesz. A dotkomlufi egy tipikus példánya. A cég logóját, ami egy ambigramma[1], Vaughan Pratt stanfordi professzor tervezte, erdetileg horizontális, és vertikális szélekkel, csak később állították csúcsaira.
A cég termékei a high-end kategóriába tartozó csúcsteljesítmény igényét hivatott kielégíteni, melyre akkoriban az üzleti oldal számított. Így meglovagolva a kereslet hullámait a cég évek alatt hatalmas vagyonra tett szert. A nagy bevétel, komoly döntéseket is hozott magával. A SUN úgy határozott, hogy felhalmozott vagyonát terjeszkedésre fordítja, ezért van az hogy a világ szinte minden szegletében található képviselete, gyára, irodája, support-hálózata. 2001-re kidurrant a dotkomlufi. A piaci igények és szokások átalakultak, mivel a várva várt high-end típusú teljesítményigény nem lett valós fenyegetés. A nagyobb cégek, mint az E-Trade, vagy a Google, inkább választottak olcsóbb x86-os linuxos rendszereket, mivel ezek szoftver szinten is nagyobb fejlődést mutattak, köszönhetően nyílt forráskódjuknak. A SUNW részvények zuhanórepülésbe kezdtek, 2004-re az 1990-es értéket sem érték el. Első lépésként megpróbálkoztak a költségek csökkentésével, és a meglévő készletek alacsony áron való kiárusításával, de hosszútávon tudták, hogy más megoldást kell keresniük.
A Sun a változó körülményekre sokféleképpen reagált. Egyrészről leállított projekteket és a processzorok optimalizálására fordította erőforrásait. Így születtek meg az UltraSPARC T-sorozat. Valamint összeállt a Fujitsu-val, és a kollaboráció eredménye lett a mai napig népszerű Sun Enterprise szervere család. A Sun Grid projekt keretében egy 3 000 gépből álló szerver farm számítási kapacitását tették kereskedelmileg elérhetővé, míg 2005-re, 3 év után először, sikerült profitot elkönyvelni.

„Open your mind!”

Az első Sun gépeken UnisoftV7 Unix rendszer futott, de ezt már az alapítás első évében felváltotta a SunOS, ami egy BSD alapokra épült (tulajdonképpen a 4.1BSD egy kipofozott változata volt) Unix változat volt. Az AT&T -vel közösen fejlesztették a System V Release 4-et, mely később a Solaris2 alapjául szolgált. Az évek során nem csupán a verziók száma növekedett, hanem a disztribúciók száma is. A Sun mindig is híres volt a stabil működést, és magas teljesítményt nyújtó operációs rendszereiről. A Solaris zárt forráskódú rendszer volt, melyet binárisban lehetett megvásárolni, főleg SPARC architektúrára. 1992 környékén megalkották az INTERACTIVE UNIX -ot, ami már támogatta az x86-os gépeket is. Ezt a kezdeményezést 2001-ben lelállították. Később Trusted Solaris néven készítettek egy disztribúciót, mely több rétegű biztonságot, és fejlett felhasználói szerepkezelést tett lehetővé. A Solaris 10 update 11/06 -ban már gyakorlatilag az összes funkciót átvették a Trusted Solarisból.

A 2001-es visszaesésre válaszul Linux alapú rendszerek szupportját is vállalták SPARC architektúrán, így x64-en SUSE és RedHat disztribúciókat használnak több nagyvállalatnál. 2004-ben nagy visszhangot keltett a hír, hogy a Microsofttal lépett együttműködésre a Sun. Egyebek mellett megegyeztek, hogy a jövőben támogatni fogják egymás rendszereit, és virtualizációs technológiáit, így születhetett meg a Windows x64-es verziója is.

Ugyancsak a változás jegyében a Sun több opensource és nem opensource projektet vett szárnyai alá. Bizonyára ismerős mindenkinek a MySQL, PostgreSQL adatbázis szoftverek, vagy fejlesztőknek a NetBeans, Java (mely önmagában megérne egy külön fejezetet), de a virtualizáció térhódításával a VirtualBox és a Sun xVM is. Természetesen ki ne hallott volna az Open/StarOffice irodai csomagról, amely „gonosz” módon már a java frissítéssel is fel akarja telepíteni magát.

Maga az OpenSolaris projekt 2004-ben indult, és egyfajta opensource konszolidációs törekvés előhírnöke lehetett. Végülis 2005 január 25-én a Sun bejelentette, hogy a DTrace néven ismert fejlesztői segédprogramjának teljes kódját nyílttá teszik a CDDL[2] licensz alatt. Ugyanezen a napon bocsátották útjára az opensolaris.org-ot. Ezt követően sorra tette nyílttá a Sun forráskódjait, így június 14-re eltekintve pár bináristól csaknem a teljes Solaris kód hozzáférhető volt.

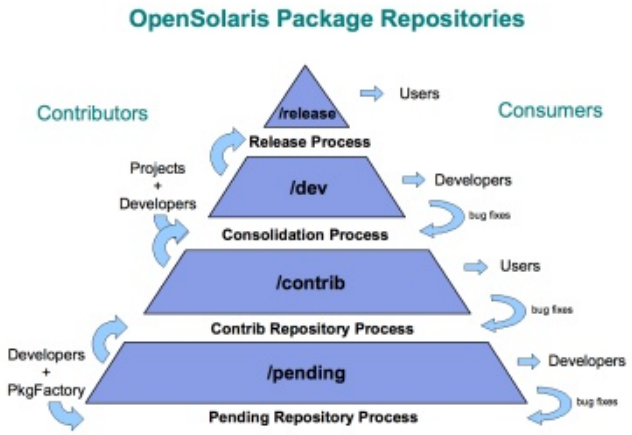
Egy úgynevezett Közösségi Tanácsadói Testület (Community Advisory Board) jött létre, melybe 2 tagot a közösség delegált, másik kettőt pedig a Sun Microsystems alkalmazottai közül választott. A bizottság később más néven újraszerveződött, OpenSolaris Irányító Testület (OpenSolaris Governing Board) néven. Feladatuk a fejlesztői és felhasználói közösségek összetartása, segítése. 2007-ben a Sun felvette Ian Murdock-ot, a Debian projekt egyik alapítótagját (az ő nevéhez fűződik többek között a Debian méltán elismert csomagkezelőjének fejlesztése), hogy vezesse az 'Indiana' kódnévre hallgató, Gnome, és egyéb GNU/linux alkalmazásokat felvonultató OpenSolaris disztribúció fejlesztését. Ezen disztribúció fontos tagja a projektnek, jelenleg a legfrissebb kiadás tavaly december 1-jén jelent meg (OpenSolaris-2008.11). Az OpenSolaris elsődleges célpontja a laptopok, így a legtöbb driver támogatás az integrált, és egyéb mobil technológiákhoz fűződik. Viszonylag friss hír például, hogy a Toshiba idén OpenSolarisszal előtelepített notebookokkal kíván kijönni.

„Mondtam, én csak az ajtót mutatom meg...”

Az OpenSolaris jelenlegi verziója a 101-es build, ami körülbelül annyit tesz, mint a linux eseteben azt mondani hogy az Ubuntu 8.10-et a 2.6.27-es kernellel szállítják. Természetesen vannak ennél magasabb buildek is, de azok a Solaris Express CE (Community Edition) néven tölthetőek le, sőt ha valaki nagyon perverz akkor akár a legújabb kernelt maga is leforgathatja. Természetesen minket az Indiana projekt OpenSolaris disztribúciója érdekel. Ennek telepítése LiveCD image-n keresztül történik. A telepítési eljárásról csak annyit mondanék, hogy ZFS-ről bootol, így csak elsődleges partícióra telepíthető, és felül kell írni a GRUB-ot is, a Sun álltal írt GRUB-bal, mivel a jelenlegi linuxos boot manager-ek még nem támogatják a ZFS boot-ot. Természetesen itt is van lehetőség multi-boot környezet kialakítására, de sajnos a telepítő program ezt nem ajánlja fel. Más nehézséget nem okoz a telepítés, gyakorlatilag minden clickable!

A ZFS-ről érdemes többet mondani, mivel ez a lelke az egésznek. A ZFS-t 2005-ben mutatták be és a Solaris kóddal egy időben csatlakozott az OS-hez, és természetesen része lett az opensource kincsnek. Eredetileg a mozaikszó Zettabyte FileSystem-et jelentett, de mára önálló szó. Mondhatni a jövő fájlrendszere. A legtöbb fájlrendszert önálló diszk eszközökre tervezik, valamilyen céllal, s manager programmal lehet redundanciát elérni. Ellenben a ZFS kernel szinten integrált redundáns fájlrendszer. Ez annyit takar, hogy virtuális diszkrendszer tárolót (Virtual Storage Pool) hoz létre, az úgynevezett Zpool-t, ami egy vagy több fizikai eszköz RAID[0,1,Z1,Z2]-be rendezve. A zpoolból kiemelhetünk, kvótázhatunk területeket, melyeket a ZFS bármely részére fel tudunk mountolni. A kernel szintű redundáns diszk műveletek nagyobb sebességet engednek meg, mintha az alkalmazásrétegből történne a vezérlés, így sok extra funkciót tudtak beültetni. Az egyik ilyen gyorsító funkció például a RAD0+1 esetében a full stripe-ok olvasása és írása, ami röviden annyi, hogy a ZFS figyel arra, hogy az diszk számára egész adatmennyiségek, logikailag is egyben legyenek kezelve, ezáltal megnövelik mind az írás mind az olvasás sebességét, úgy hogy a redundancia is megmarad. Gyakorlati szempontból megközelítve a dolgot, a zpool kezelése rendkívül egyszerű, mivel bármikor be lehet illeszteni új eszközt, vagy dinamikusan futásidőben lehet átszabni a fájlrendszert.

Azért hogy ne csak a rendszergazdák örüljenek, a ZFS Snapshot technológiát implementálták az OpenSolaris userland-ba, így keletkezett a TimeSlider, ami a Mac OS/X rajongóknak ismerős lehet TimeMachine néven. Lényegében az történik, hogy a gép 'megjegyzi' az inod táblát (a fájlrendszer tartalomjegyzékét) és a snapshot után már csak a változást rögzíti. Ez az UFS snapshot-hoz képes annyiban változott, hogy a ZFS okosan addig nem nyúl a régi adathoz (fizikailag), amíg a diszken/diszkeken van fel nem használt szabad hely, így sokáig szinte nulla helyigénye van egy-egy snapshotnak. A TimeSlider segítségével a kívánt mappa állapotát (fájlok, könyvtárak) tudjuk visszaállítani egy gombnyomással.



A másik fontos dolog bármely disztribúciót nézzük a csomagkezelő. Ian Murdocknak köszönhetően ez a része is sokat fejlődött a Solarisnak. Hasonlóképpen a deb, rpm, és egyéb rend-

szerekhez itt is léteznek repository-k. Hasonlásképpen a Linuxhoz, az OpenSolaris közösségnek is van hivatalos, opensoalris.org repó-ja, benne különböző státuszban lévő előfordított alkalmazással, és libekkel, és van nem hivatalos, félhivatalos, és hobbi szinten üzemelő repója is. Ezeket a csomagkezelőket, pkgget/pkgadd/pkgutil paranccsal lehet elérni, vagy packagemanager néven GUI -ből. Felhívnám a kedves Olvasó figyelmét, hogy a közhiedelemmel ellentétben létezik OpenSolaris repó kis hazánkban is! A csomagkezelő helyi adatbázisba rögzíti a függőségeket, leírásokat, verziókat, így a telepítés szinte gyerekjáték. Ha valaki az opensolaris.org a sunfreeware.com és a blaswave.org repóit felveszi közel 28 000 csomagot számolhat meg, így senki nem panaszkodhat, hogy nincs program OpenSolarisra.

Az első bootoláskor ha van lehetőségünk láthatjuk, hogy betölti a szolgáltatásokat. Ez az un. SMF[3] rendszere a Solarisnak. Nem kell megijedni, ez majdnem olyan mint a linux esetében az /etc/init.d/* csupán a Sun továbbgondolta a dolgot, és az elinduló alkalmazásokat, és szolgáltatásokat egy olyan rendszerbe tette, ahol egymás függőségét kell definiálnunk, felhasználói behatásra, kell válaszolniuk, stb. Ebből következik, hogy a futási szintek is átalakultak, Solarisban ezeket milestone-nak (mérföldkőnek) nevezzük, és szerepelnek a szolgáltatási hierarchiában. Ha egy service nem tud elindulni, akkor az maintanance állapotba kerül és az SMF logot készít róla. Egy svc-t úgy tudunk kikapcsolni ha disable statuszba tesszük, ekkor nem fog elindulni a következő induláskor sem. Összességében egész jól használható dologról van szó.

A következő dolog miután bejelentkeztünk, hogy egy kicsit megismerkedjünk a hálózati manager programmal. Ezt a funkciót két service láthatja el. Az újabb, fiatalos lendületű, ám picit bohókás svc:/network/physical:nwam service, vagy az idősebb, bölcsebb, de elég szétszórt svc:/network/physical:default. Az első service-nek még neve is van és GUI épül rá, ő a Network Magic. Lényegében felolvas egy /etc/nwam/llp fájlt, ahool a user definiálhatja (végtelen egyszerű formában), hogy melyik hálózati interfészen mi legyen. Idősebb párjának nincs neve, és annyit tesz, hogy felolvas egy /etc/hostname.<interfésznév> fájlt, ebből megtudja az interfész nevét (plumb,up), majd felolvas egy /etc/hosts fájlt, ahonnan megtudja az interfész nevéhez tartozó IP címet. Mindkettő használható,de én azt mondom, hogy érdemes követni az új hullámot. Ha már van hálózatunk, akkor van mitől félnünk is. Egészséges paranoiánkat az IPFilter nevű kezdetben 3rdparty, de most már Sun készítésű tűzfalprogram enyhíti. Erről elég most csak annyit tudni, hogy a Solaris esetében ez kernelmodulként van jelen így nagyobb teljesítményt tud elérni, mint alkalmazásbeli társai.

Ezzel működőképes rendszerünk van, lehet kattintgatni. Érdekessegképpen megemlíteném a Sun virtualizációs lehetőségeit is. Erre egyrészről van a Solaris Zone. Ebben az esetben a

virtuális gépek tulajdonképpen egy izolált környezetet jelentenek. Nem a mostanság népszerű virtuális technikát használja, hanem egy emulált gépet jelent. Ebben az esetben a létrehozott gép read only-ban mountolja a globális zóna egyes könyvtárait, öröklődnek a felhasznalók, valamint az aktuális telepített csomagok listája is. Valamint a később telepített frissítések is automatikusan átkerülnek a zónákba is. A hálózati interfészeket kívülről egy-egy fizikai interfészre húzott virtuális interfésznek látjuk, így azok csak együtt képesek reagálni (pl.: promiscuous mode). A kernel termé-

szetesen ugyanaz lesz mint a globális zónában, és az architechtúrát sem lehet megváltoztatni. Annyi plusz lehetőség van még, hogy kis ügyeskedéssel lehet linux 2.4 vagy 2.6 -os zonát kreálni egy speciális BrandZ[4] kernel segítségével. Persze az ilyen hibrid megoldásokért a legtöbben nem rajonganak, mert az így létrejött linux nem bír minden linux tulajdonsággal. A másik lehetőség virtualizációra az Sun xVM (Xen) valamint a VirtualBox, ezek mind az AMD-V és az Intel VT-x technológiákat használják és többnyire bírnak HVM és PV lehetőségekkel, hát igen … a virtualizációról regelig lehetne írogatni...
Remélem cikkem elnyerte tetszésed, és szívesen olvasol többet a témában, s ha gondolod most telepíthetsz egy virtuális gépre OpenSolaris-t! Hajrá! ;)

[1]http://en.wikipedia.org/wiki/Ambigram
[2]http://en.wikipedia.org/wiki/Common_Development_and_Distribution_License
[3]http://en.wikipedia.org/wiki/Service_Management_Facility
[4]http://opensolaris.org/os/community/brandz/

Link ajánló:
http://lifewithsolaris.jp/
http://www.blastwave.org/
http://sunfreeware.com/

Csíkos Bálint

Tudtad-e?

Danger from the Deep

"Veszély a mélyből" névre hallgat az a Windows-, Linux-, FreeBSD- és MacOS X -rendszerre is telepíthető játék, amelynek célja, hogy minél realisztikusabban, vagyis a történelmi tényeknek, valamint a valós műszaki adatoknak megfelelően szimulálja a második világháborús német tengeralattjárók működését. Talán a kép- és hanghatásokra is vonatkozó szigorú követelmények indokolják, hogy a nyílt forráskódú, ma is alfa állapotú, 3D-s szimulátorprogram immár több mint öt éves múltra tekinthet vissza. Jelenleg a tervezett funkcióknak mintegy a kétharmada működik. Miközben a projekt folyamatosan C++-programozókat, grafikusokat, történészeket, gépésztechnikusokat, hangmérnököket, csomagkészítőket és tesztelőket is keres, a felhasználók szépen gyarapodnak.

Jankovich Oszkár

Távolról is biztonságos Mysql

Gondolom, nem minden kedves Olvasó tudja, hogy mit jelent a LAMP rövidítés, ezért álljon itt a megfejtés: Linux, Apache, Mysql, Php/Perl/Python. (Leggyakrabban persze a Php-ről van szó.) Ez feltételezi, hogy egy szerveren van az adatbázis és a szkriptünk is. De mi van akkor, ha mégse, és mondjuk a php-ket futtató webszerver kilométerekre, vagy még messzebb van az adatbázisszervertől?
Nem ritka, hogy a programok közti adatforgalom nélkülözi a titkosítást (kisebb gépigény). Ez abban az esetben nem is olyan nagy probléma, amikor a gépet (vagy akár az adott, megbízhatónak tekintett LAN-t) nem hagyja el az érzékeny adat. De mi van akkor, ha a valamiért mégis ilyesmire van szükségünk? Ilyenkor lép be a képbe a jó öreg openssl. Ha követnénk a klasszikus elgondolást, miszerint minden Unixon futó program csak egy valamire való (de arra nagyon), rögtön az stunnel felé fordulnánk. Aki még nem használt stunnelt, annak legyen elég ennyi: olyan alagút, aminek csak a két vége látja titkosítatlanul az adatforgalmat, a köztes állomások nem.
Ha https-t használunk, az nem védelem, elvégre az csak a felhasználó és a webszerver közti utat védi, míg a webszerver és az adatbázisszerver közöttit nem, amennyiben ezek nem egy gépen futnak.

Mysql és az SSL

A mostani Mysql verziók szerencsére rendelkeznek olyan lehetőséggel, amely az ilyen varázslatokat kiküszöböli. Ehhez kell egy megfelelően fordított bináris. Ha a disztribúciónk által kínáltat használjuk, akkor ott szinte semmi gond nincs, forrásból történő fordítás esetén azonban szükséges, hogy az Openssl forrása (vagy a dev csomag) fenn legyen a számítógépünkön. A Mysql 5.1.11-es verziója előtt a configure-nak még meg kellett adnunk, hogy Yassl vagy Openssl van a gépünkön (--with-openssl vagy --with-yasl), azonban a legújabb verzióknál elég csak a --with-ssl is.

A Mysql binárisunk SSL támogatását számos módon oldhatjuk meg. Az első és talán legegyszerűbb:

```
mysqld --ssl --help
```

Ha nincs beépítve SSL támogatás, akkor „ismeretlen opció” hibát kapunk.

A másik, amellyel egy "éles" Mysql szerveren nézhetjük meg:

```
SHOW VARIABLES LIKE 'have_ssl'
```

vagy

```
SHOW VARIABLES LIKE 'have_openssl'
```

Itt háromféle értéket kaphatunk: YES, NO, DISABLED. Az első kettő nem szorul magyarázatra, a harmadik viszont azt jelzi, hogy van ugyan SSL támogatásunk, de nem annak megfelelően lett elindítva a mysql.

Mysql: szerver

A mysql futtatásakor három opciót kell megadnunk:
--ssl-ca (CA tanúsítvány)
--ssl-cert (ez a szerver publikus kulcsa)
--ssl-key (ez a szerver privát kulcsa)

Tehát egy mysqld indítás ehhez hasonlóan fog kinézni:

```
mysqld --ssl-ca=cacert.pem --ssl-cert=server-cert.pem --ssl-key=server-key.pem
```

A kérdéses állományokat az openssl segítségével hozhatjuk létre, de annak részletezése már egy másik cikk témája lenne.

```
Tanúsítványok létrehozása kipróbáláshoz

CA tanúsítvány
openssl genrsa 2048 > ca-key.pem
openssl req -new -x509 -nodes -days 365 -key ca-key.pem > ca-cert.pem

Szerver tanúsítvány
openssl req -newkey rsa:2048 -days 365 -nodes -keyout server-key.pem > server-
req.pem
openssl x509 -req -in server-req.pem -days 365 -CA ca-cert.pem -CAkey ca-key.pem -
set_serial 01 server-cert.pem

Kliens tanúsítvány
openssl req -newkey rsa:2048 -days 365 -nodes -keyout client-key.pem > client-
req.pem
openssl x509 -req -in client-req.pem -days 365 -CA ca-cert.pem -CAkey ca-key.pem -
set_serial 01 client-cert.pem
```

Mysql: kliens

Kliens oldalról kétféle paraméterezés lehetséges. Ha csak az SSL kapcsolat kell minden extra nélkül, úgy az alábbiak szerint indíthatjuk a klienst:

```
mysql --ssl-ca=cacert.pem
```

Esetleg érdemes az adott felhasználóhoz felvenni a REQUIRE SSL-t, hogy még véletlenül se csatlakozzon titkosítás nélkül.

Az SSL kikényszerítése így néz ki:

```
GRANT ALL PRIVILEGES ON teszt.* TO 'teszt'@'localhost' IDENTIFIED BY 'teszt' REQUIRE SSL;
```

A visszavonás pedig így:

```
GRANT ALL PRIVILEGES ON teszt.* TO 'teszt'@'localhost' IDENTIFIED BY 'teszt' REQUIRE NONE;
```

A másik esetben – ha a REQUIRE X509 opciót adjuk meg a felhasználónál – nemcsak az SSL megléte szükséges, de az is, hogy a kulcs és a tanúsítvány is megfelelő legyen. Ekkor így néz ki az indítás:

```
mysql --ssl-ca=cacert.pem --ssl-cert=client-cert.pem --ssl-key=client-key.pem
```

Amint az látszik, fontos, hogy mindkét fél ugyanazzal a CA tanúsítvánnyal készült tanúsítvánnyal rendelkezzen.

Azt, hogy az aktuális kapcsolat használ-e SSL-t, az alábbiak szerint kérdezhetjük le:

```
SHOW STATUS LIKE 'Ssl_cipher'
```

Ha ez nem üres, akkor aktív az SSL.

A Mysql klienséből a STATUS vagy a \s paranccsal kérdezhetjük le. Ekkor vagy „Not in use”-t kapunk, vagy pedig a titkosítás módját, mint például „Cipher in use is DHE-RSA-AES256-SHA”.

Hasznos: PHP példa

Persze a fenti példák mit sem érnek gyakorlati előny nélkül. Lássuk hát, mi a helyzet, ha mondjuk PHP esetében kell alkalmazni a fentieket!

Amennyiben nem óhajtunk tanúsítványt ellenőrizni, úgy nem bonyolódik a dolgunk, hiszen a jól megszokott mysql_connect parancsot egy MYSQL_CLIENT_SSL opcióval kell megtoldani, valahogy így:

```
$conn = mysql_connect('localhost','teszt','teszt',false,MYSQL_CLIENT_SSL);
```

Ha azonban a tanúsítványt is szeretnénk vizsgálni, akkor a mysqli bővítményhez kell fordulnunk. A mysqli_ssl_set paranccsal adhatjuk meg az SSL paramétereket. A részletes paraméterezésért a PHP weboldalán a mysqli_ssl_set oldalt érdemes átnézni. (Illetve ezzel együtt az egész mysql szekciót.)

Világosan látszik tehát, hogy egy már meglévő Mysql-t használó alkalmazást nem nagy ördöngösség átalakítani olyanra, ami titkosított kapcsolatot használ. Szerveroldalon optimális esetben csupán be kell kapcsolni az SSL támogatását, míg kliens oldalon a mysql_connect kibővítésével is már elég sokat nyerünk.

Végül pedig - bár ezt talán hangsúlyoznom sem kell - érdemes gyakran frissíteni a rendszerünket, elvégre az openssl-t is csak emberek fejlesztik, bármikor kiderülhet valami bug.

Hivatkozások:

http://en.php.net/manual/en/mysqli.ssl-set.php

Medve Zoltán

Hálózatbiztonság nyílt forrású eszközökkel I.

A hálózatbiztonság ma már megkerülhetetlen része az egész informatikai biztonság témakörnek, hiszen az informatika – számítástechnika, hálózatok nélkül nehezen képzelhető el. Ebben a témakörben számtalan szakértő van, és folyamatosak a szakmai viták is, de vannak közös pontok. Megpróbálunk ezek mentén haladni.

Míg korábban nem voltak a számítógépek hálózatba kötve, ma már az egyedülálló számítógépek jelentenek kuriózumot. Ennek több oka van, az egyik erősen anyagi, (persze a többi is :)) míg az 1980-as években egy hálózati vezérlőkártya elérhette vagy meghaladhatta az egész konfiguráció árát, (és a többi szükséges eszközről még nem is beszéltünk) ma egy network interface card filléres eszköz, vagy az alapkonfiguráció része.
Arról itt nem fogunk vitát nyitni, hogy mi minősüljön hálózatnak, de érdekességképpen érdemes megemlíteni, hogy George Stibitz már 1940 szeptemberében telexgépet használt arra, hogy a K Model nevű gépével kapcsolatos problémákkal összefüggő utasításokat küldjön a New Hampshire-ben lévő Dartmouth College-ból a New Yorkban üzemelő Complex Number Calculator nevű gépéhez, illetve az eredményeket hasonló módon küldte vissza. A korai időkben a legfőbb számítógépes periféria a konzol volt (nem a mai értelemben vett), innen utasították a számítógépet, és ide is érkeztek a számítógép által küldött üzenetek (telexgép, vagy elektromechanikus írógép-szerű berendezésre kell gondolni), így ez a megoldás a nem túl távoli jövőt is előrevetítette.
A korai számítógép (nagygép) – terminal kapcsolatot sem tekintjük hálózatnak, de a mai intelligens terminálok használata, akár egymás, akár a szerverek elérésére már beletartozik a hálózat fogalmába.

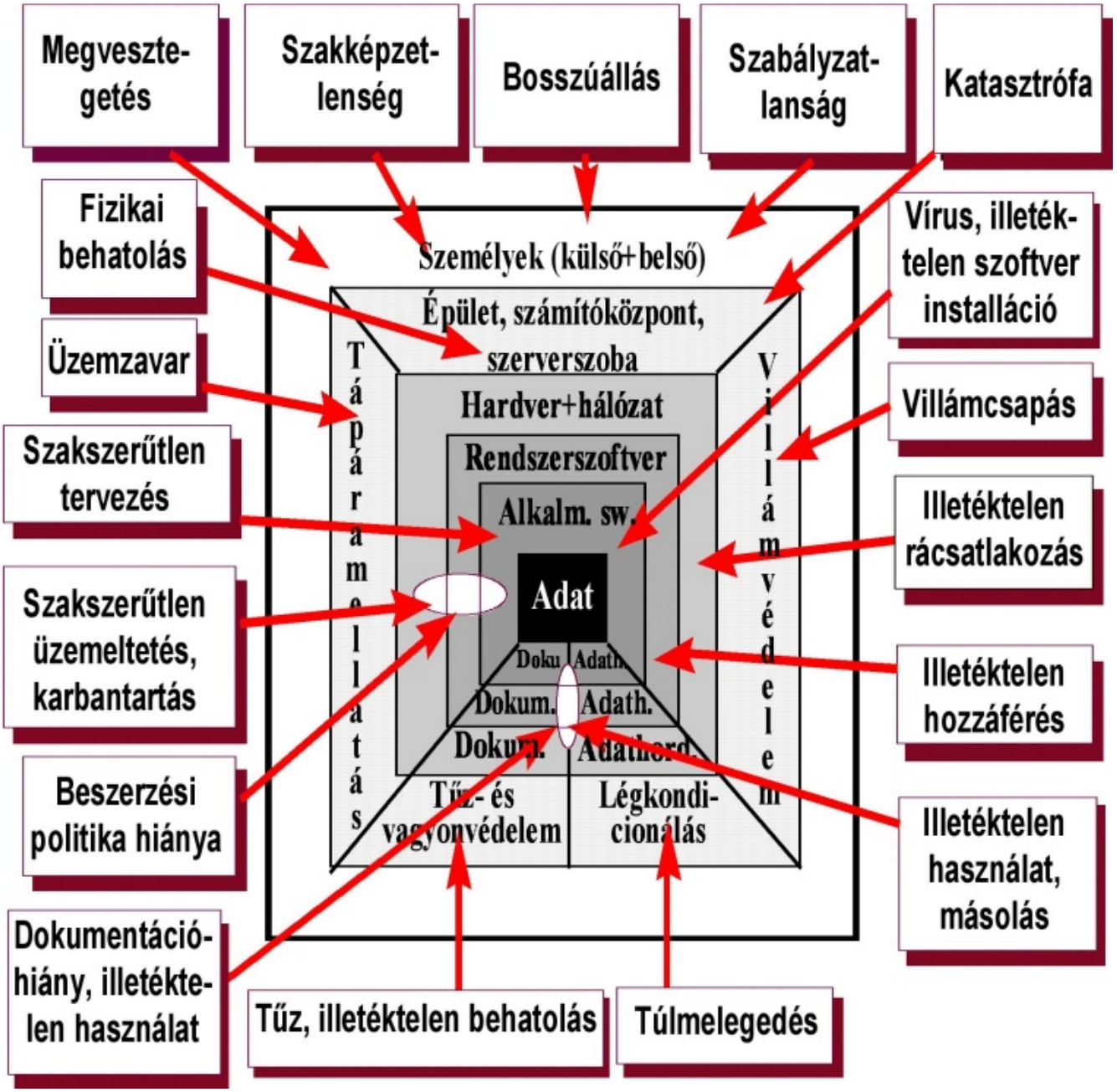
A hálózatok történetét, fajtáit nagyon sok helyen leírták, így ezzel nem igazán foglalkozunk a cikksorozat keretei között (címszavakat viszont felsorolunk néha, hogy akit bővebben érdekel a téma rákereshessen), inkább azzal, hogy a hálózat és annak biztonsági felügyelete, hogyan valósítható meg nyílt forrású eszközökkel.

A mai informatikai környezetben nemcsak a hagyományos számítógépekre kell gondolnunk, hanem a rengeteg beépített, beágyazott rendszerre is, amik könnyen az adott hálózat részévé válhatnak, hosszabb, rövidebb időre. Itt nem kizárólag a hordozható gépekre, (notebook, netbook, PDA), hanem a mobiltelefonokra és sok esetben már a háztartási gépekre is gondolhatunk számítógépként, vagy olyan eszközként, aminek lényeges része a számítógép.
Lassan már nem azon érdemes csodálkozni, ha megjelenik a halas két tonna lazaccal és háromezer doboz fekete kaviárral az ajtónk előtt, hanem azon miért nem történt ez meg sokkal hamarabb, és miért nem tettünk semmit azért, hogy ne törhessék fel az internetre kötött hűtőszekrényünket.

Egy átlagfelhasználó ritkán gondolja végig, mi minden fenyegetheti akár az otthoni rendszeré-

nek biztonságát is, vagy a céges hálózat esetén a munkahelyén mi mindenre kellene még neki is figyelnie.

Ennek végiggondolását segíti az alábbi ábra (egy számítástechnikai védelmi szabályzatból kiragadva).

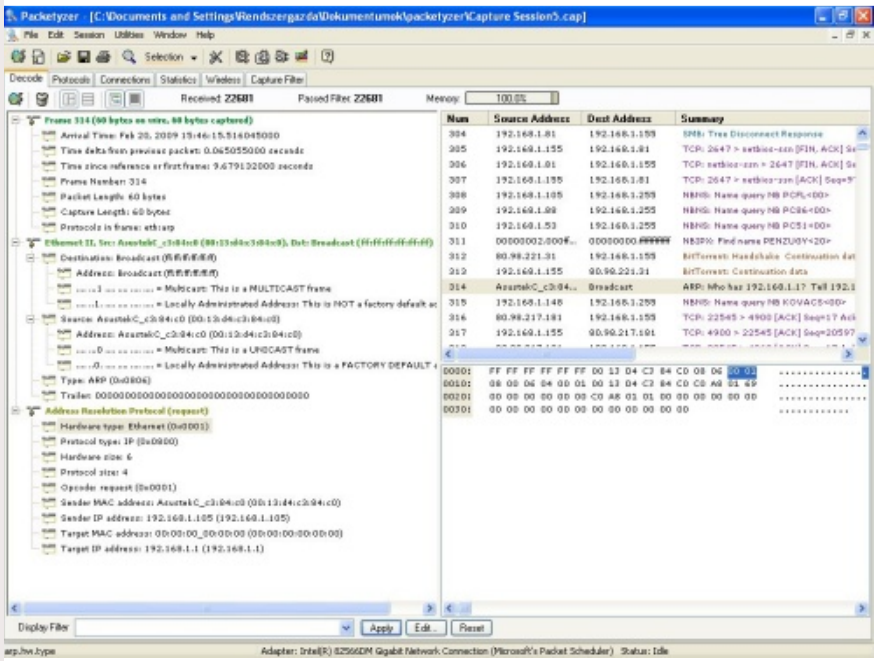


Látható mi minden fenyegeti a hálózatunkat és az adatainkat. Több felmérés szerint is, egy hálózatra leselkedő veszélyek közül a leggyakoribb a belső támadás, ezt érdemes észben tartani.

Másik fontos dolog, az informatikai biztonság növelése és a rendszerek használhatóságának biztosítása nem éppen egy irányba hatnak. Minden biztonsági intézkedés korlátozásokkal jár, a korlátozásokkal pedig a rendszer használhatósága, kényelme csökken.
Továbbá bizonyos költséghatárokon belül maradva, minél biztonságosabbá teszünk egy rendszert, az annál bonyolultabb lesz (persze van az a pénz, amikor ez nem teljesen igaz).

Nézzünk egy nem gyakori példát, tudjuk, hogy a home routerek mindegyikének van valami biztonsági rése (akár több is), legfeljebb még nem derült ki, de ki fog.
Ezért több különböző típust fűzhetünk egymás mögé, hogy a betörőnek nehezebb dolga legyen. Ugyanakkor a meghibásodás lehetősége is ezzel arányosan nő.
Ez a helyzet az üzleti megoldásokkal is. Több vagy bonyolultabb eszköz, nagyobb meghibásodási lehetőség. Ezért itt is meg kell találni az arany középutat, a biztonság és a használhatóság elvárásai között.

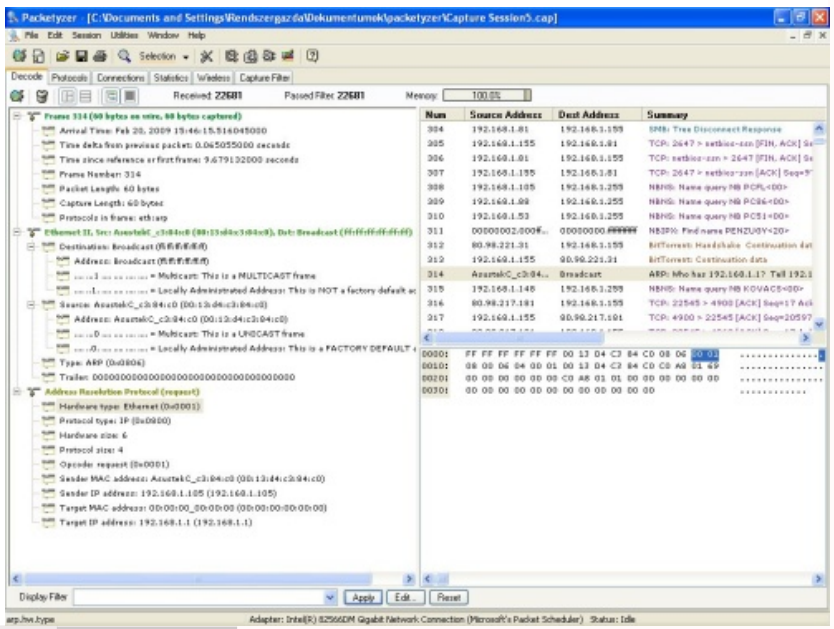
A terjedelmi korlátok miatt nem vesszük át a hálózatok fajtáit, ha valami fontos a témakör szempontjából mindenképpen megemlítjük. Az alkalmazott eszközök közül néhány igazán hasznosat részletesen is be fogunk mutatni, lehet nem a cikk keretein belül, de a fanzinban mindenképpen.
Ha egy hálózat biztonságát akarjuk megteremteni valamilyen szinten, már néhány gép esetén is érdemes megtervezni bizonyos dolgokat, nagyobb gépszám esetén pedig nélkülözhetetlen ez a lépés.



Annyit le kell szögezni (és ebben a szakértők túlnyomó része is egyetért), informatikai biztonság nem létezik, de érdemes rá törekedni.
Ma már nem áltathatjuk magunkat azzal, hogy a mi adataink senkit sem érdekelnek, hiszen erősen automatizálva van a pásztázási és betörési tevékenység, így mindenki potenciális célpont.

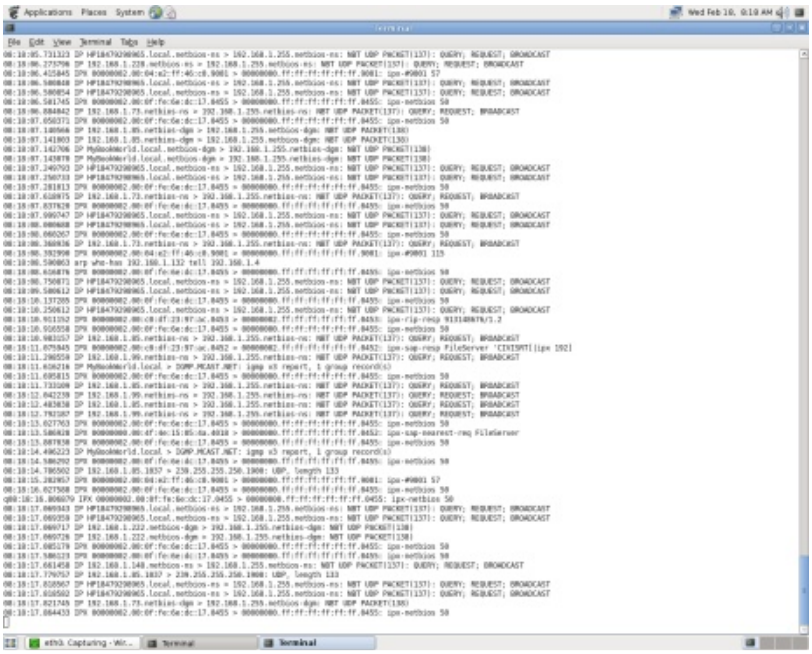
Ahhoz, hogy tudjuk mi történik a hálózatunkon, egy minimális alapozást mégis kénytelen vagyok elkövetni, akinek ez triviális attól elnézést kérek.

A számítógéphálózat valamilyen módon összekötött gépeket jelent. Ennek módja többféle lehet, mi azt a részét fogjuk vizsgálni, amikor ezt egy célészközzel, esetünkben egy erre a célra kifejlesztett hálózati kártyával oldják meg. Ezeknek a megvalósítása többféle lehet, a legelterjedtebb az úgynevezett ethernet kártya.



Mivel nagyon sokféle hálózati eszköz és hálózati protokoll létezik, le kell szűkíteni a kört, a

ma legelterjedtebb és leginkább használt hálózattal fogunk foglalkozni.
Megnézzük mi is kell ma ahhoz, hogy egy átlagos gépről elérhessük az Internetet.
Az átlagos gép egy PC kompatibilis számítógép lesz, ami rendelkezik egy átlagos Ethernet vezérlővel (ez lehet alaplapra integrált vagy külön kártyán megoldott NIC, de akár PCMCIA vagy USB megoldás is), amibe belecsatlakoztatható egy ma már szintén átlagos Cat5 kábel RJ45 csatlakozóval. Ezt beledugványozzuk egy aljzatba, ami lehet ADSL vagy kábelmodem (közvetlen csatlakozás nem ajánlott), vagy ezek mögött lévő tűzfallal ellátott router (ez már sokkal jobb), vagy egy olyan hálózati csatlakozó, mellyel részeivé válhatunk egy kellőképpen védett hálózatnak (ez lenne a legjobb).
A fizikai kapcsolat létrehozása, és a szükséges beállítások megtétele után máris részeivé válhatunk a világ legnagyobb hálózatának, minden ezzel járó jó és rossz következménnyel együtt. (Ha elég ügyesek vagyunk pillanatok alatt szert tehetünk egy úgynevezett zombi gépre, persze ehhez szükséges egy híres szolgáltató supportosa, aki elmagyarázza, hogy a legelterjedtebb op. rendszer alól közvetlenül lássunk ki a netre, a szereld magad modem használatával.)



Jó lenne tudni mi történik, nehogy az animgif kommandó ránktörjön és elvigyen minket, mint a világ egyik legnagyobb spammerét. (Persze a helyzet nem ilyen súlyos, de az egészséges paranoia még senkinek sem ártott).

Mit tehetünk? Az alábbi ábrát mindenképpen érdemes áttanulmányozni, ez a sokat emlegetett OSI modell. Aki nem próbálja meg értelmezni az ábrát, nem nagyon lesz tisztában azzal, hogy mi játszódik le a hálózatán.

(Elkezdtem csinálni egy táblázatot, de a lapzártára való tekintettel feladtam. Inkább idelinkelek pár kiváló ábrát, és talán néhány dolgot hozzáfűzök a végére.)

http://www.cs.brandeis.edu/~rshaull/cs33b/osi-model.png
http://www.teach-ict.com/as_a2/topics/osi_and_standards/osi/images/osi-model.jpg
http://www.automatedbuildings.com/news/oct06/reviews/OSI.gif
http://bfindarto.files.wordpress.com/2008/03/osi3.gif

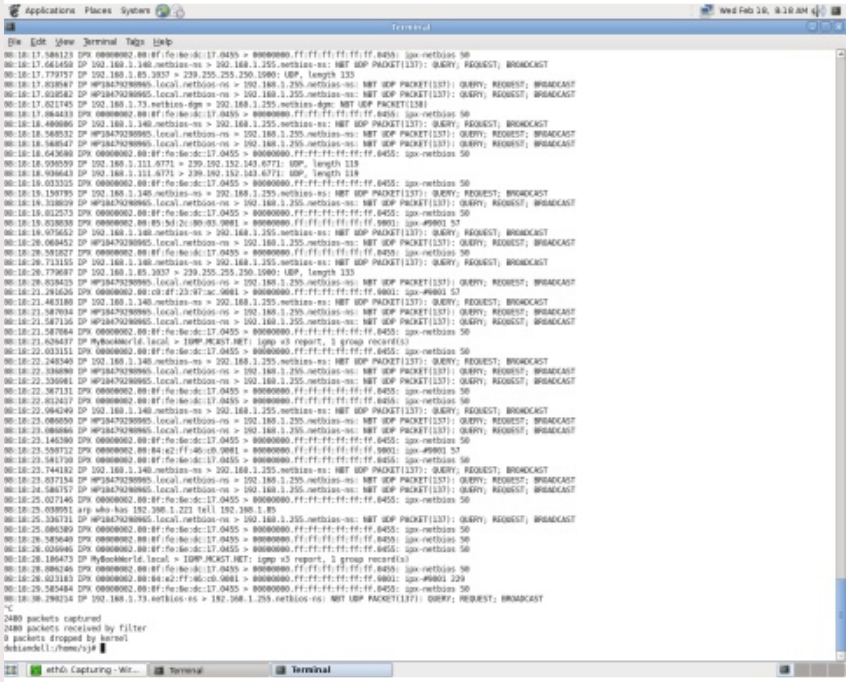
AZ ISO/OSI modell
ISO (International Standards Organization -Nemzetközi Szabványügyi Szervezet)
OSI (Open System Interconnection)
A modellben minden réteg csak a saját feladatával foglalkozik, a szükséges adatokat átveszi és továbbadja a többi rétegtől/rétegnek.

Az OSI referencia modell szerint egy hálózat 7 rétegre van osztva.

A rétegek feladatai:

Az adatátvitellel foglalkozó rétegek:

A fizikai réteg (physical layer)



A kommunikáció legalsó szintje.

A bitek kommunikációs csatornára való kibocsátásáért felelős. Biztosítania kell, hogy az adó által küldött jeleket a vevő is azonosként értelmezze. (Az elküldött 1-es bit, a túloldalon is 1-es legyen.)

Az adatkapcsolati réteg (data link layer)

Alapvető feladata a hibamentes átvitel biztosítása a szomszéd gépek között, vagyis a hibás, zavart, tetszőlegesen kezdetleges átviteli vonalat hibamentessé transzformálja az összeköttetés fennállása alatt. Az adatokat adatkeretekké (data frame) tördeli, továbbítja, a nyugtát fogadja, hibajavítást és forgalomszabályozást végez.

A hálózati réteg (network layer)

A kommunikációs alhálózatok működését vezérli, feladata az útvonalválasztás forrás és célállomás között. Ha az útvonalban eltérő hálózatok vannak, akkor fregmentálást, protokoll átalakítást is végez. Kezeli a torlódásokat, stb. Az utolsó olyan réteg, amely ismeri a hálózat topológiáját.

A szállítási réteg (transport layer)

Feladata a végpontok közötti hibamentes adatátvitel biztosítása. Már nem ismeri a topológiát, csak a két végpontban van rá szükség. Feladata az összeköttetések felépítése, bontása, csomagok sorrenbe állítása.

A logikai összeköttetéssel foglalkozó rétegek:

A viszonyréteg (session layer)

Lehetővé teszi, hogy két számítógép felhasználói kapcsolatot létesítsenek egymással. Jellegzetes feladata a logikai kapcsolat felépítése és bontása, párbeszéd szervezése. Szinkronizációs feladatokat is ellát, ellenőrzési pontok beépítésével.

A megjelenítési réteg (presentation layer)

Az egyetlen olyan réteg, amely megváltoztathatja az üzenet tartalmát. Tömörít, rejtjelez (adatvédelem és adatbiztonság miatt), kódcserét (pl.: ASCII - EBCDIC) végez el.

Az alkalmazási réteg (application layer)

Széles körben igényelt szolgáltatásokat tartalmaz, (a felhasználók által használt protokollokét) Pl.: HTTP, FTP.

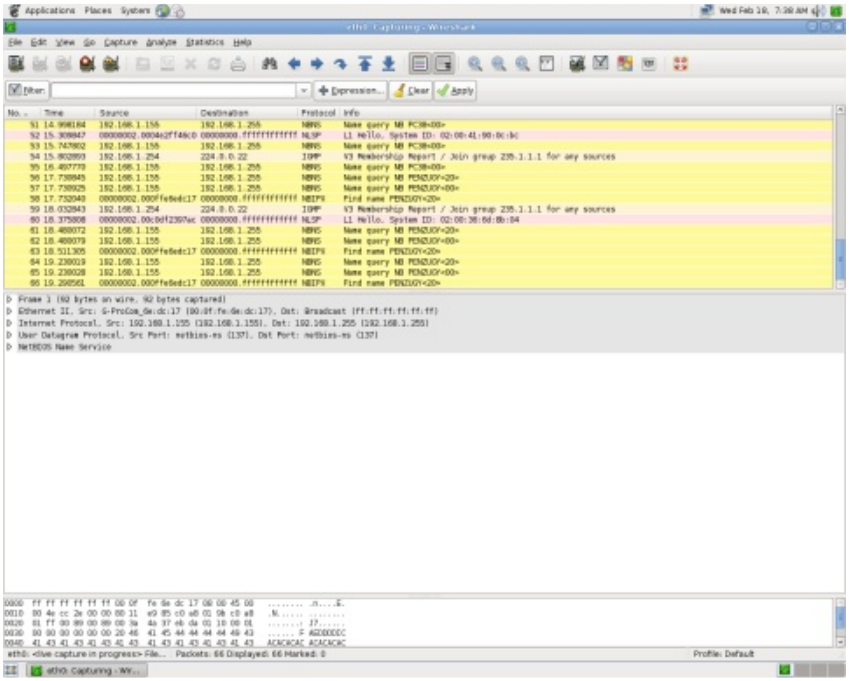
Ha már mindenki megtanulta a fentieket, kötelességemnek tartom közölni, nem jellemző az OSI modell fizikai megvalósítása, ez inkább elméleti megközelítés.

Az OSI referencia modell, hivatkozási alap. A valóban működő megoldások egyre inkább ennek megfelelőek lesznek.

Nézzük ennek fényében hogyan fest egy minket érdeklő megoldás, a TCP/IP az OSI modellel összevetve. Ja, miért nem a referencia modell alapján készítették a TCP/IP-t? Mert ő a korábbi.
Az OSI a TCP/IP után készült.

http://www.hardwaresecrets.com/imageview.php?image=6731
vagyis
http://www.computing.dcu.ie/~humphrys/Notes/Networks/tanenbaum/1-21.jpg
és
http://www.webgobe.ro/konyv/kepek/fig2-17.gif

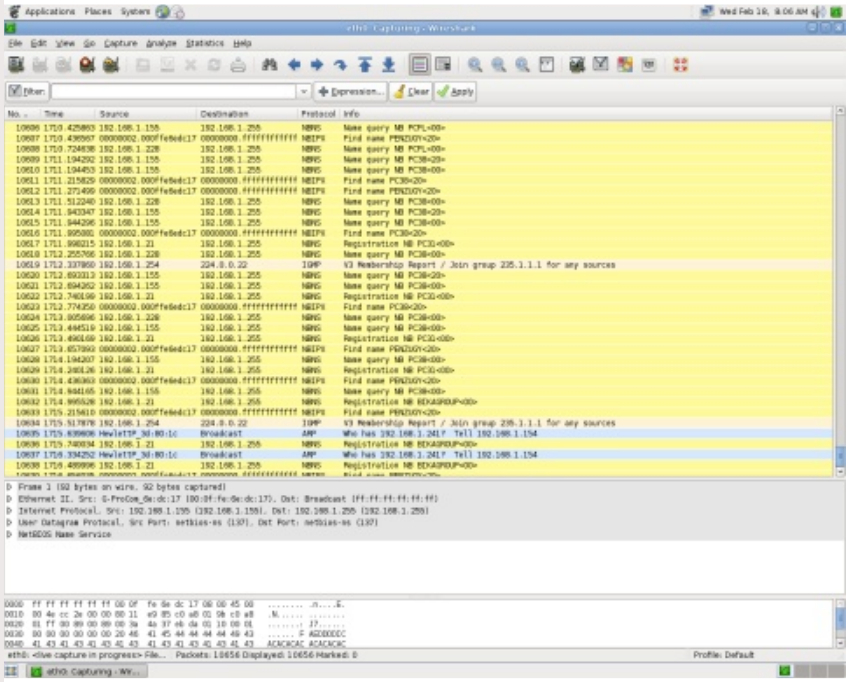
A TCP/IP a két legelterjedtebb protokolljáról kapta a nevét. Története az ARPANET-tel függ össze.
Jelenleg az Internet nevet viselő világméretű hálózattá összekapcsolódott részhálózatok protokollgyűjteménye.
Az OSI modellnél egyszerűbben, négy rétegből épül fel.



A hoszt és a hálózat közötti réteg megkötése, hogy a hosztnak olyan hálózatba kell kapcsolódnia, amely támogatja az IP csomagok kezelését, továbbítását.
Az internet rétegen (hálózatközi réteg) alapul az egész TCP/IP modell. A modellen alapuló hálózatok csomagkapcsolt hálózatok. Az adatok csomagok formájában közlekednek. Ezeknek egyik helyről a másikra akkor is el kell jutniuk, ha a cél másik hálózaton van. Ezt az internet (IP) protokoll intézi.
A szállítási réteg feladata, hogy a kommunikációban résztvevők között meghatározza a kommunikáció mikéntjét. Ehhez két protokollt használ. Az átvitelvezérlő protokoll egy megbízható összeköttetésen alapuló protokoll (transmission control protocol, TCP). A megbízható összeköttetést a tényleges kommunikációt megelőző kapcsolatfelvétellel éri el, a TCP kézfogással (TCP handshake). A kapcsolatfelvétel három lépésből áll, ezután létrejön a kommunikáció, majd a végén a kapcsolat lebontása következik, szintén három lépésben.
A TCP feladata a hibamentes átvitel biztosítása.
Az ugyanitt működő másik protokoll az UDP. Eredeti neve Unreliable Datagram Protocol. Valóban megbízhatatlan, de nem valami bizalomgerjesztő, ha ezt már a nevében is hirdeti, így aztán ma már inkább UDP-nek hívják (User Datagram Protocol). Megbízhatatlan, mert nem kapcsolatorientált, nem tartalmaz nyugtákat, sőt még a fejlécben lévő cheksum mezőt sem kötelező kitölteni, így aztán elveszhet, sérülhet, nem tudható, hogy egyáltalán megérkezett-e?
Na akkor biztos nem is használják semmire, gondolhatnánk. Ámde a valóság ennél érdekesebb, az UDP egy gyakran használt protokoll. Azért mert nem erőforrásigényes, nagyon gyors, és kicsi a sávszélességigénye is. Az előbbiekből adódóan olyan esetekben használják,

amikor a gyorsaság a legfontosabb, ha elvész kis idő után újraküldik. Érdemes megbízható hálózatokon használni, pl.: lokális hálózatokban. Kép vagy hangátvitel esetén is alkalmazzák, amikor nem nagy probléma egy kis pontatlanság.
Az alkalmazási réteg a negyedik ebben a rendszerben. Ez tartalmazza a magasabb szintű protokollokat, és a legtöbb felhasználó az ezeket a protokollokat igénybe vevő programokkal szokott találkozni mindennapi munkája (vagy szórakozása) során. (A pasziánsz nem tartozik ide.)
FTP (FTP programok), HTTP (böngészők), SMTP (egyes levelezők), stb.

Miután elméletben ilyen szépen megalapoztuk a dolgokat, lássuk mi történik a gyakorlatban.



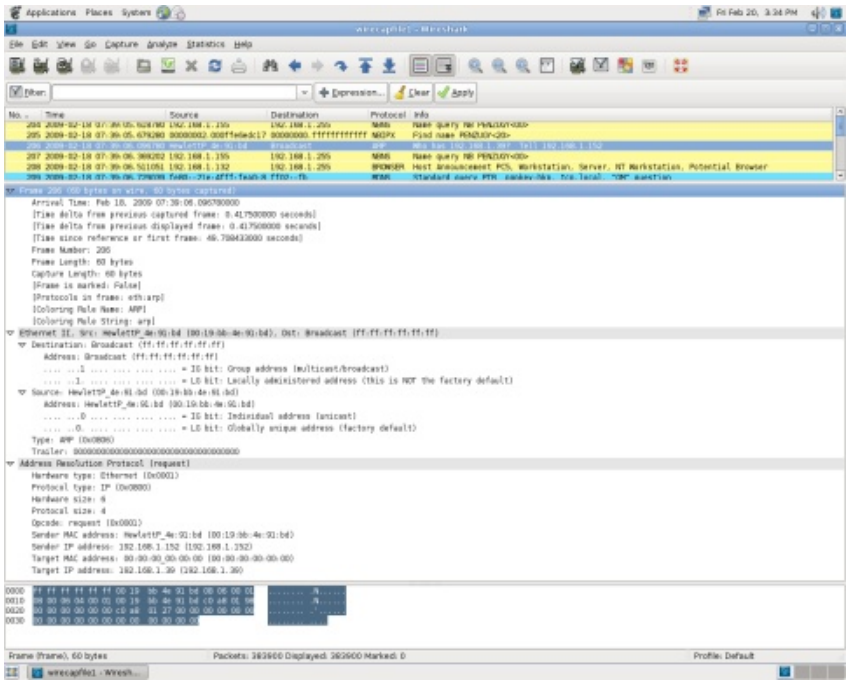
Az internet világméretű elterjedése előtt is voltak hálózatok, ezek kiválóan működtek az adott környezetben, sőt működnek ma is, de az internettől függetlenül.
Az internet használatához szükség van erre a néhány protokollra.
Egy korlátozott hálózat kívánalmainak kiválóan megfelelt az ipx vagy a netbeui protokoll, de ahhoz, hogy a jelenlegi világméretű hálózat részei lehessünk kell egy igen fontos dolog.
Az összes gépet egyedileg meg kell különböztetni.
Ráadásul az Internet hálózati szegmensek összessége, ezért úgynevezett routolható protokollra van szükségünk.
Nézzük a megkülönböztetés megoldását.
Két segítséget kapunk (nem is, hármat), a MAC címet, ami minden létező hálózati vezérlőkártyához egyedi.
Ez a gyakorlatban is így lenne, de majd adódik néhány érdekesség. Ezeket később megnézzük.
A másik a hálózat részét képező gépek egyedi IP címe. Enélkül nem lehetünk az internet része.
Minden az internetre csatlakozó gép egyedi IP címet kap. Nemcsak a szerverek, de az ideiglenes jelleggel kapcsolódó gépek is, amelyekkel böngészünk, levelezünk, stb.
Ez akkor is igaz, ha nem rendelkezünk fix IP címmel, hanem minden csatlakozáskor kiosztanak nekünk egyet.
Sokszor hallani, hogy elfogynak az IP címek, ez igaz lesz az IPV4-re, de már itt az IPV6.
Úgy kell elképzelni az IP címet, mint egy egyedi azonosítót, minél több gép van az interneten, annál több cím kell, mint a gépkocsik rendszámtáblájánál.
A régi rendszám betű betű-szám szám- szám szám formátumú volt, ezek elfogytak volna, ha nem vezetik be a 3 betű – három szám formátumot, ami sokkal több vegyedi variációt enged meg. Sőt egy ideig, a teljes átállásig, egymás mellett működött a kettő. Nem is emlékszem, mikor láttam utoljára régi rendszámot (de szerintem még Mabel sem emlékszik rá).
Az IP címeknél most éljük ezt az időszakot, az átállást, amikor a régi IPV4 és az új IPV6 is jelen van.

Egyelőre nézzük meg az IPV4-et.
Egy mostani IP cím 4 szám egymástól ponttal elválasztva, valahogy így: 192.168.9.142. Az utolsó pont csak a mondat vége miatt van ott.
Ez gyönyörű, de az IP címkiosztás rejtelmeibe mélyen belemenni hosszú lenne, így csak annyit jegyeznék meg, hogy ehhez még tartozik egy úgynevezett alhálózati maszk, pl.:255.255.255.0.
E kettő együtt azonosítja az adott hálózaton lévő adott gépet, ketten együtt alkotják az adott számítógép egyedi rendszámát. Ráadásul ez a rendszám minden csatlakozásnál változhat, hacsak nem rendelkezünk fix IP címmel. (Elnézést a rövidségből adódó pontatlanságokért, de a lényeg megértéséhez talán elég ennyi.)

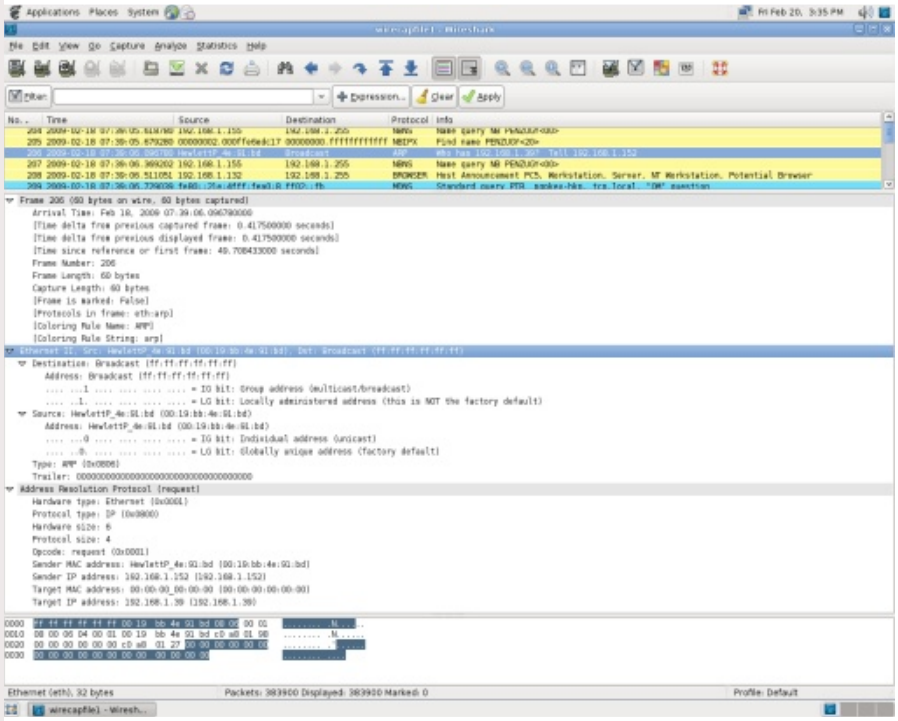
Egyébként meg így néz ki az IP cím, a gép számára:
11000000101010000110010000001111
a 32 bit négy bájtra felosztva
11000000 10101000 01100100 00001111

ugyanez a négy bájt tízes számrendszerben ábrázolva (minden tag 0 és 255 közötti szám lehet).
192.168.100.15.
Belátható, hogy így talán könnyebb megjegyezni (legtöbbünknek).

Minek az alhálózati maszk?
Mivel több (elég sok) összekapcsolódó hálózatról van szó, az IP cím és a hálózati maszk segítségével fogjuk meghatározni a pontos címet.
A cím és a maszk bináris formáját használva végre kell hajtani egy ÉS (AND) műveletet a két szám között. Ez a művelet a maszkolás. Ez két részre osztja az IP címet. A két fő rész közül az első lesz a hálózat címe, a második lesz az adott hálózaton lévő gép címe. Egyik rész sem lehet csupa 0 vagy csupa 1-es. Ha az IP cím gépazonosító része csupa 1-es, az nem egy adott gép címe, hanem a broadcast cím, ami az adott hálózat összes gépét jelenti. Ugyanez csupa 0-val, a hálózat azonosítója. A maszkolás segít annak eldöntésében is, hogy egy adott gép ugyanazon a hálózaton van-e, mint mi, vagy egy másikon. Ha ugyanazon, nem kell az átjáróhoz fordulni, mehet a szórt (broadcast üzenet) a saját hálózatra, ha nem, lehet egyből az átjáróhoz fordulni. Az átjáró segít összekapcsolni a különböző hálózatokat, úgy, hogy más hálózattal (hálózatokkal) is kapcsolatban áll, a miénken kívül. Ahhoz, hogy egy hálózatról kimehessünk az Internetre szükségünk van egy alapértelmezett átjáróra (default gateway). Sajnos ez még mindig nem elég, szükséges foglalkozni a DNS-el. Most mondjam azt, hogy

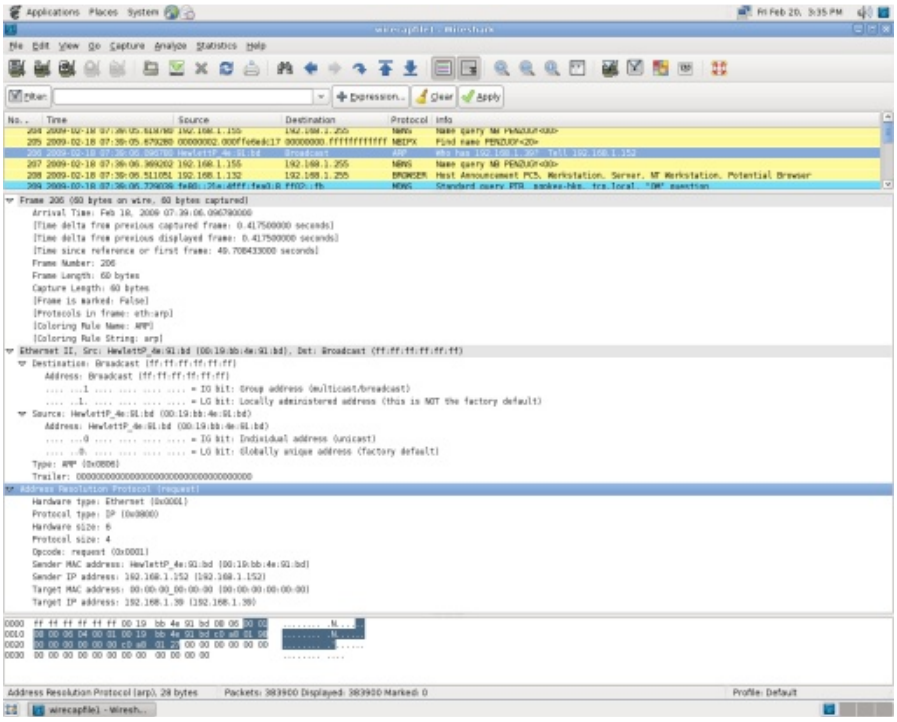


mindenki nézzen utána? Örs levágja a fejemet, ha a bevezető rész több, mint negyven oldal lesz :(. Megpróbálom nagyon röviden, a többi házi feladat :) . Mai, modern világunkban van két igen fontos dolog, a DNS :) . A dezoxiribonukleinsav és a Domain Name System közül a cikk témájához jobban illeszkedő utóbbit választjuk. Mértékadó források állítják, hogy az Interneten előforduló hibák legnagyobb része DNS hibákra vezethető vissza, és valószínűleg igazuk van. Sajnos a legnagyobb szolgáltatók is képesek és hajlamosak ezzel kapcsolatban szarvashibákat elkövetni (vannak személyes tapasztalataim is). Ezután nézzük, miről is van szó. Beszéltünk a könnyen megjegyezhető IP címről, pl.: 11000000101010000110010000001111. De nézhetjük a sokkal barátságosabb formáját is: 192.168.100.15. Ebből néhány százat vagy ezret megjegyezni elég jó teljesítmény. Hogy ne kelljen minden internetezőnek fotografikus memóriával rendelkező matekzseni autistának lennie, létrehozták a Domain Name System-et, a névszerinti címzést, avagy a tartományi névkiszolgálást, stb, lehetne tovább ragozni. Lényeg, hogy ez az egyik legfontosabb szolgáltatás az interneten. A rendszer feladata egy adott webcím



lefordítása a hozzá tartozó IP címre, vagyis egy adott hálózaton lévő gép nevét lefordítja, visszaadja a hozzá tartozó IP címet. Ez jó esetben fordítva is működik. Így lehetségessé válik a domain nevek megjegyzése, vagy valamilyen érthetőbb formában való elképzelése és használata. Például, ha tudjuk, hogy létezik International Business Machine nevű cég, megpróbálhatjuk beírni a böngésző címsorába, hogy ibm.com, vagy ibm.hu. Ezt könnyebb megjegyezni, mint az adott IP címet, persze nem biztos, hogy létezni kell mondjuk kovacsbt.org címnek, de ez az elv elég jó kiindulási alap. Hogy ez működhessen, kellenek a rendszert kiszolgáló szerver gépek. Röviden a a DNS rendszer a domaineket (tartományokat) kezelő, a világon több (sok) szerverre elosztott hierarchikus adatbázis-rendszer. További előny még, hogy ha egy kiszolgáló gép IP címe megváltozik, azt nekünk nem kell tudni, csak a rendszer kezelőinek, akik véghezviszik a szükséges változtatásokat, majd hagyják, hogy a rendszer automatikus része is működjön kicsit. A domain nevek sokkal ritkábban változnak, mint a kiszolgálók IP címei. A rendszer biztosítja a decentralizált kezelést, a névtartományok hierarchikus strukturálását, a nevek egyértelműségét és a bővíthetőséget. DNS szervert lokális hálózatunkban is alkalmazhatunk, sokszor már néhány gép esetén is előnyös lehet saját névkiszolgáló beállítása. Erről egyelőre ennyit. Most már valóban itt az ideje, hogy megnézzük mi történik a hálózaton. Ahhoz, hogy a támadások ellen sikeresen tudjunk védekezni, tudnunk kell mik lehetnek azok, és tudnunk kell hogyan és mi játszódik le a folyamat során. Az egyébként igen kiváló hekkergájd foglalkozik ezzel két WGA kártya ismertető között, vagyis leírja, hogyan áramlanak a nullák és egyesek a kábelen (vezetéken, madzagon), de a valóság sajnos ennél kicsivel bonyolultabb. Nézzük mivel találkozhatunk, ha belenézünk a lengődugóba. Bizony szembejönnek velünk a már tárgyalt protokollok, cso-

magok fizikailag megvalósult, lebitesedett formában. Sokat érünk vele. Honnan tudjuk, hogy az a kábelen lévő kidudorodás (mint az óriáskígyó nyakán lecsúszó dzsungelkutató) ARP, UDP csomag, vagy egy jól fejlett elektroszú? Ehhez kell egy, a folyamatokat megfigyelhetővé, láthatóvá tevő csomagelemző (packet analyzer, vagy network protocol analyzer). Ilyenekből rengeteg van, most csak a nyílt forrású megvalósításokról beszélünk (ezekből is van éppen elég). Az ismerkedéshez hozzáfoghatunk egy rendkívül egyszerű, de hasznos változattal, a tcpdump nevűvel. Már sokszor hallottuk, hogy a nyílt forrású alkalmazások nem eléggé dokumentáltak, ez egészen biztosan így van, hiszen ehhez a rendkívül bonyolult alkalmazáshoz, ami a képernyőre löki a hálózati forgalmat, mindössze alig több mint 1600 soros manual tartozik. Az ismerkedéshez viszont ennél kevesebb is elég. `aptitude install tcpdump` Majd `tcpdump`, és már gyönyörködhetünk is a hálózati forgalomban. Egyes hálózatokban igen egzotikus csomagok is feltűnhetnek, de mi a TCP/IP család tagjait vesszük majd szemügyre. A programot CTRL-C billentyűkombinációval leállítva kapunk egy kis statisztikát a megjelenített csomagokról. Észrevehető, hogy kisebb vizsgálódások, műtétek céljára megfelelő a tcpdump, de komolyabb elemzésekhez érdemes valami fejlettebb megoldást alkalmazni. Több programot is választhatunk, kinek mi lesz a szimpatikus, de nagyon megfelelőnek tűnik a Wireshark (lánykori nevén Ethereal). A Wiresharkon kívül van még egy eszköz ami az Ethereal bázisára épült és mindenképpen meg kell említeni, a Packetyzerről van szó. Ez egy igen kiváló eszköz, aminek csak annyi hibája van, hogy nincs belőle csak Windows verzió (Persze nyílt meg minden, de akkor is :)). Idáig jutottunk léteznek olyan kiváló programok, amik kategóriájuk csúcsát képviselik, de csak a "másik" rendszeren akarnak dolgozni (most csak az alaphelyzetről beszélek, a különböző trükkökről nem).



A Packetyzerrel csomagot is lehet szerkeszteni, de ez nem jelenti azt, hogy mindezt csak Win alatt lehet megtenni, sőt.

Visszatérve a drótcápára, máris működésre bírhatjuk, ha beírjuk a következőket: aptitude install wireshark. Ha minden rendben volt, felkúszik a menübe és onnan indítható. Ez már színes, szagos, csilivili, a csomagokat fájlba menthetjük, visszatölthetjük, szóval matathatunk kedvünkre. Mielőtt jobban belemerülünk meg kell említeni még egy fogalmat, ez pedig a port. (Itt és most tekintsünk el attól, hogy a legfinomabb portok közé tartozik a tawny port és a ruby port, főként az előbbi hosszan érlelt változatai, de egy palack Royal Oporto Tawny Port biztosan nem fog csalódást okozni. A családtagok mind az (újra)erjedésben borpárlattal megállított és erősített borok közé tartoznak, akárcsak a szintén port ugál madeira, vagy a spanyol sherry (ez utóbbi nem tévesztendő össze a hazánkban elhíresült cherry brandyvel). A szerkesztőségi tesztlabor bármilyen beküldött Portot, Madeirát, vagy Sherryt szívesen tanulmányoz, és tapasztalatait közzéteszi :)). Na de térjünk vissza a kevésbé ízletes,

ámde roppant szórakoztató számítógépes portokhoz. A soros, párhuzamos, stb portokról nem beszélünk, sokkal inkább a hálózati kommunikációban használatos portokról. Amikor az internet szolgáltatásait használjuk, a kommunikáció nem közvetlenül számítógépek között zajlik, hanem a számítógépeken futó programok kommunikálnak más számítógépeken futó programokkal. Egy számítógépen egyszerre több kliens- és szerverprogram is működhet. Hogy ezek kommunikációja ne keveredjen össze, kapukat (portokat) használnak, melyeket szám vagy név azonosít. A portok használata nem szabványos, de a legtöbb kiszolgáló úgynevezett "jól ismert kapun" (well known port) várja a kliensek felől érkező kéréseket. A kliensprogramok a használt portok számát általában elrejtik a felhasználó elől, az esetek többségében a felhasználónak erről nem kell tudnia. Minden program, mely egy kommunikációs csatornát indít, egy portot regisztrál magának a rendszerben (az adott számítógépen), mely portra érkező csomagokat ezentúl ez a program fogja kezelni. A portok számozva vannak 0-tól 65535-ig, melyek közül az alsó rész 1024-ig speciális programoknak és kommunikációknak van fenntarva. Akkor lássunk egy konkrét csomagot! A Wiresharkot elindítva máris látszanak a csomagok, ha elegendőt összegyűjtöttünk, ezeket el is menthetjük egy állományba későbbi vizsgálódás céljából. Egy-egy konkrét csomagtípusra hamarabb rátalálhatunk, ha használjuk a szűrés funkciót. Kezdjük egy ARP csomag vizsgálatával. Az ARP (Address Resolution Protocoll) egy cím lekérdező protokoll. Ha két gép "közvetlenül" kommunikál egymással, azaz egy szegmensen találhatóak a kommunikáció alapja a hardver cím (MAC address, hardware address). Az ARP segítségével tudja meg egyik gép a másik gép hardver címét. Amikor A gép kommunikálni szeretne B géppel IP-n keresztül ARP kérést küld, amelyben megadja melyik IP címhez tartozó hardver címre kíváncsi. Ezt a kérést az adott szegmensen minden számítógép veszi (broadcast), és ha birtokában van az adott hardver cím - IP cím páros, a beállításaitól függően válaszol is. A válasz tartalmazza a kívánt hardver címet és a kommunikáció megindul. Ezt a Wireshark segítségével nagyon látványosan ellenőrízhetjük (még egy jól megtervezett jubileumi tűzijáték sincs ilyen látványos). A drótcápa, mint egy háromöves tatu tárja elénk az információkat. A felső harmadban a csomagok sorakoznak szép egymásutánban, ahogy a dróton kolbászolva belefutottak a program lepkehálójába. A középső harmadban a kiválasztott csomagra vonatkozó információk sorakoznak: az egész keret, majd a rétegeknek megfelelően a lényegi információk. Az alsó harmadban láthatjuk mindezt más formában, a második és harmadik harmadban, ha kijelölünk egy részt, az ki lesz jelölve a másik ablakban és viszont. Jól látható, ha keretre kattintunk (frame) ki lesz jelölve az egész csomag. Ha az EthernetII kezdetű sorra a csomag eleje és vége (ebbe van burkolva az ARP), majd, ha az ARP sorra kattintunk a csomag belsejét jelöli ki a program. Így láthatóvá tehetjük az egész csomag felépítését, és a réteginformációk elhelyezkedését. Ez más típusú csomagoknál is így lesz, csak a rétegződés és a tartalom fog eltérni. Láthatjuk az ARP-nél a forrás és cél információkat. Mi lehet a gond? Az, hogy ezeket az információkat nem túl nehéz meghamisítani. Annyira nem, hogy a Packetyzerben van edit funkció, és a meghamisított csomagot egy gombnyomásra vissza lehet küldeni a hálózatba. Persze egy valamire való támadáshoz érdemes alkalmazást készíteni, (hacsak nem vagyunk olyan gyorsak, mint Jesse James), de már így is elkövethetünk érdekes dolgokat. Kezdetnek ennyit, folytatás a következő alkalommal. Támadás!

Szőke József

Impresszum

Főszerkesztő:
Horváth Örs Apor - *Budapest*

Tördelőszerkesztő:
Falusi Ernő - *Budapest*

Logó:
Makay József - SKL Projekt

Szerzők:
Csíkos Bálint - *Budapest*
Jankovich Oszkár - *Budapest*
Kovács Zsolt - *Debrecen*
Medve Zoltán - *Szeged*
Pfeiffer Szilárd - *Mosonmagyaróvár*
Sütő János - *Budapest*
Szőke József - *Mikepércs*
Vomberg István - *Budapest*

A II. évfolyam 1. szám összesített fórumának címe:
http://flosszine.org/forum/II_evfolyam_1_szam

A FLOSSzine elérhetőségei:
E-mail: info@flosszine.org
Web: www.FLOSSzine.org
IRC: #FLOSSzine ; #FLOSSzine.hu ; #FLOSSzine.org (irc.freenode.net)

Köszönet az FSF.hu Alapitványnak a tárhelyért!