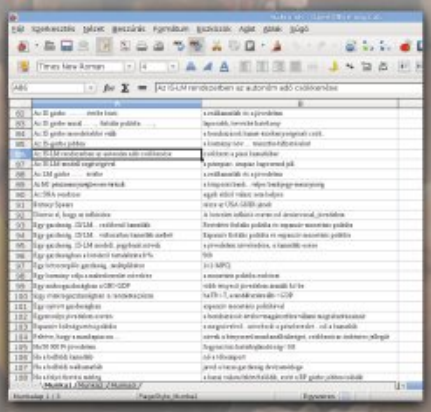
FLOSS ZINE

Free / Libre / Open Source Software fanzine

2009. december

II. évfolyam 3. szám



Táblázatkezelés szabadon

Nagios

Hálózatbiztonság nyílt forrású eszközökkel

Szegedi és budapesti Szabad Szoftver konferencia

Kótyagos pingvintől a Linux Akadémiáig

A telefonos kisasszony már a múlté

Enterprise 128 emuláció Linuxon

Hello Window!

Clapf

http://flosszine.org

Tartalom

Lite

Pro

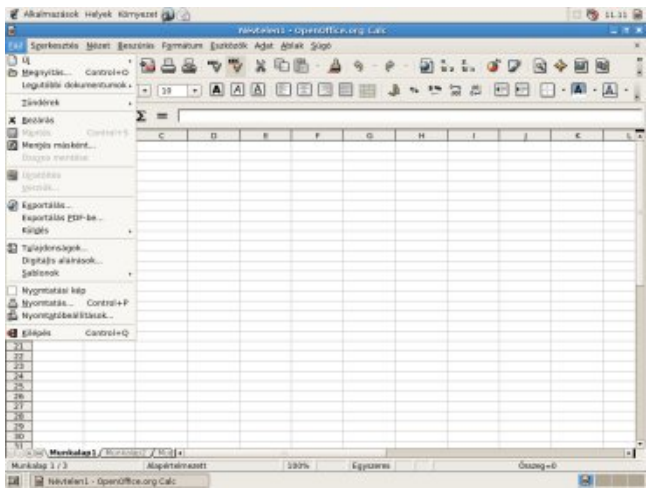
Writer Calc Base Impress Draw Math

Táblázatkezelés szabadon

Az előző számban már taglaltuk a szövegszerkesztés témakörét, az ott leírtak egy része igaz lesz erre a cikkre is, ezért a sima és vonalas papír féle történetet nem ismételjük meg a kockás papírral is.

A táblázatok, számoszlopok, stb kezelése számítógépes eszközökkel olyan régi dolog, mint az elektronikus szövegszerkesztés. A táblázatkezelők a sima és franciakockás füzeteket kiváltani hivatott szoftvertermékek, amik jelentősebb mértékben a mikrogépekkel terjedtek el (persze ez igaz a szövegszerkesztőkre is).
Mint ahogy az is igaz, hogy az egész dolog nem az Excellel vagy a Worddel kezdődött.
Quattro, Lotus (nem a mai IBM-es), Symphony (szintén nem), nem is a legrégebbi megvalósításai a témakörnek, de ha valaki tudja miről van szó, talán tudni fogja azt is, mi az a táblázatkezelő program.
Akiknek ezek a szavak semmit sem mondanak, azoknak ideírjuk, a számítógépes tábklázatkezelő programok, táblázatok és számoszlopok, számsorok kezelésére szolgálnak, ideértve az azokkal végzett különböző műveletek végzését, és az eredmények megjelenítését.
Ebben a témakörben mindenképpen meg kell említeni a Dan Bricklin és Bob Frankston által 1979-ben elkövetett VisiCalc-ot (visible calculator), ha már az előző részben nem említettük meg az 1976-os Electric Pencil-t.
A szövegszerkesztő programok esetén elkövetett hibák itt is előfordulnak. Vétek egy táblázatkezelő programot összekeverni egy könyvelő, vagy adatbázis-kezelő alkalmazással, bár sok helyen megpróbálták a főkönyvi könyvelést is ilyen módon kezelni.
Ahogyan a szövegszerkesztő programok esetén, úgy a táblázatkezelő alkalmazásoknál is, ma már nagyon sok alternatíva létezik a legelterjedtebb platform kiváltására, jó néhány ezek közül teljesen ingyenes és szabad.
Nem nagyon érdemes a függvények számát vagy a kezelt oszlopok és sorok mennyiségét méricskélni, hiszen minden elérhető alternatíva ezekből több, mint elegendőt nyújt az átlagos igényekkel rendelkező felhasználó számára.
A szabad irodai rendszerek közül az OOo idevágó modulja a Calc a leginkább ajánlott, de a különböző disztribúcióknál számos egyéb kiváló programmal találkozhatunk.
A különböző rendszereknél ebben az esetben is megállapíthatunk egy közös elvet, amit alapul véve könnyen eligazodhatunk a különböző megvalósítások kezelése során.
Ez bizony szinte teljesen megegyezik a szövegszerkesztőknél említettekkel, hiszen itt is az elkészítés, eltárolás, újrabetöltés, átszerkesztés, megjelenítés/nyomtatás, stb fázisain kell keresztülmennünk valamilyen sorrendben.
Ez mind jól látható a különböző programok menüit megnézve.
Nézzük mit tudnak a szabad vagy ingyenes táblázatkezelők.

A nagyágyú az OpenOffice.org Calc



Mindent tud, amire a témában egy (csak kevéssé bomlott elméjű) felhasználónak szüksége lehet.
(Jó, a menüpontok nem ott vannak, ahol a piacvezető alkalmazás azokat megjeleníti, de akinek arra van szüksége, pengesse ki annak az árát a zsebpénzéből, vagy a céges költségvetésből a megfelelő példányszámban).
Sok esetben még úgynevezett szakmai fórumokon is olvasható, hogy az OOo magánfelhasználásra ingyenes, de céges alkalmazás esetén már nem.
Óriási tévedés. Az OOo összes modulja bármilyen felhasználás esetén teljesen ingyenes és szabad, tessék nyugodtan használni, akárhány példányban.

Normál felhasználás esetén semmilyen megkötésre nem kell számítani, a vonatkozó licenszfeltételek főleg arra lettek kihegyezve, hogy ez így is maradhasson:
http://www.openoffice.org/license.html
Egyébként az OpenOffice.org programcsomag 2009 október 13-án volt kilenc éves.

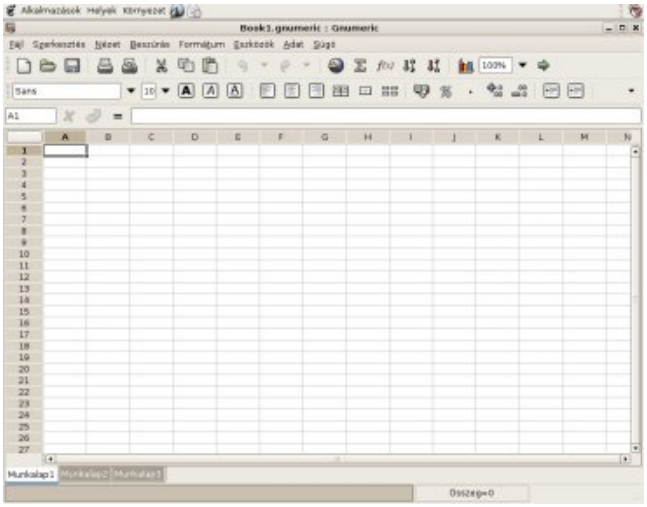
Az OOo Calc semmiben sem maradhat le a nagy vetélytárs mögött, így ebben is található, egyebek mellett, easter egg.
A játék elindításához a Calc program első cellájába kell beírni azt, hogy =game("StarWars").
Ezután megjelenik egy Space Invaders klón. Sajnos elég lassúcska, de igazi retro érzést ad.
Sok más is található a Calc különböző verzióiban, de ez a játék a 3.0-ban még biztosan benne volt.
A programcsomagban a varázslókat „Tündérek” helyettesítik, és a Calc is tud PDF exportot külön alkalmazás használata nélkül, akárcsak az OOo Write.
Összességében elmondható, hogy semmiben sem kell szégyenkeznie fizetős vetélytársaival összehasonlítva (annak ellenére, hogy egy ilyen összehasonlítás mindenképpen furcsa).
http://hu.openoffice.org/calc.html

OxygenOffice Professional Calc

Az OxygenOffice Professional (korábbi nevén OpenOffice.org Premium) kezdetben annyiban tért el az FSF.hu OpenOffice.org-tól, hogy a telepítőkészletbe nagy mennyiségű ingyenes (többségében szabad szoftveres licencű) sablont, clipart képet, betűkészletet stb. helyeztek el. Az OxygenOffice honlapja a SourceForge-on található:
http://sourceforge.net/projects/ooop/
Az OOo Calc (és az OxygenOffice Professional Calc) jelenleg a legprofibb táblázatkezelő alkalmazások a nyílt forrású alternatívák között, de nem kell mindig ágyúval verébre lődözni, ezért nézzük a kistesókat is.

Gnumeric

A Gnumeric egy nyílt forráskódú táblázatkezelő program, amely először a GNOME Office-ban bukkant fel. A Gnumericet a Microsoft Office Excel alternatívájaként készítette Miguel de Icaza. Jelenleg a projektet Jody Goldberg vezeti.
A Gnumeric saját formátuma XML-alapú, de nagyon sok más formátumot is támogat, pl.: Applix, CSV, Data Interchange Format, Microsoft Excel 5.0-XP *.xls (bináris), illetve Microsoft Excel 2007 *.xlsx (xml alapú), HTML, LaTeX, Lotus 1-2-3, MultiPlan, GNU Oleo, OpenDocument, OpenOffice.org 1.x, Plan perfect, Quattro Pro, SpreadsheetML, Xspread és Xbase.
A táblázat méretét szabályozhatjuk a következő módon:
a /src/gnumeric.h fájlban a
define SHEET_MAX_ROWS()
define SHEET_MAX_COLS()
változók kívánt értékeit be kell állítani, majd le kell fordítani és telepíteni kell a programot.



A Gnumeric a GNU General Public License alatt használható.
http://projects.gnome.org/gnumeric/

Kspread

A KSpread egy táblázatkezelő, amely a KOffice-ban található meg. Támogatja a Microsoft Excel, Applix Spreadsheet, Quattro Pro, CSV és az OpenOffice.org Calc fájlainak kezelését is. Mind ez, mind a Gnumeric úgynevezett komoly alkalmazás, amelyekkel az esetek nagy részében kiválthatjuk a „nagyágyúkat”.

http://www.koffice.org/kspread/

Gnu Oleo

Ez egy GNU táblázatkezelő program.
Alaphelyzetben karakteres, de elérhető hozzá grafikus interface is.
Kiválóan használható kisebb feladatokra és X nélküli gépeken, szervereken is nagyon hasznos lehet.
http://www.gnu.org/software/oleo/

Siag

Ez a táblázatkezelő a Siag Office része, a Siag Office (saját bevallása szerint) egy ingyenes és szabad irodai csomag Unix-ra.
A Siag Office a Siag (Scheme In A Grid) táblázatkezelőből, a PW (Pathetic Writer) szövegszerkesztőből, az Egon animációs programból (Egon for president, a PPT mondjon le!), a XedPlus szövegeditorból (nicsak még egy szövegszerkesztő), és a Xfiler fájl menedzserből tevődik össze, valamint még része a Gvu nézegető is.
A honlapon elérhető a forrás, a Mac verzió, az OpenBSD verzió, és linux binárisok rpm és tgz formátumban. A projekt állományai saját oldalukon kívül a Freshmeat oldalain is elérhetőek.

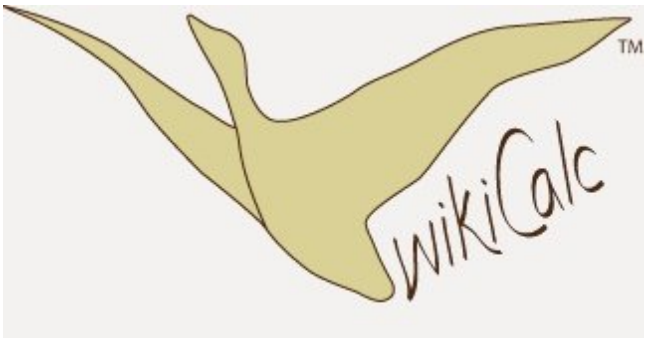
Jelenleg a Siag 3.6.1 Released a legfrissebb verzió.
http://siag.nu/
http://freshmeat.net/projects/siagoffice/

A végére hagytunk két web bázisú megoldást a Simple Spreadsheet-et és a WikiCalc-ot.

Simple Spreadsheet

Ez a táblázatkezelő szintén egy nagyobb csomag része. A Simple Groupware & CMS tartalmazza.
A Simple Spreadsheet egy web-bázisú táblázatkezelő, készítői Javascript, HTML, CSS és PHP felhasználásával alkották. A program free software, a „GNU GPLv2 License” alatt használható.
Ugyan része a Simple GroupWare rendszernek, de különállóan is futtatható.
http://www.simple-groupware.de/cms/Spreadsheet/Home

WikiCalc



Dan Bricklin tovább alkot. Ő és Bob Frankston készítették a VisiCalc-ot. Úgy döntött, hogy az első táblázatkezelő elkészítése után megint kitalál valami újat és elkészítette az első internetes táblázatkezelőt. Ez a WikiCalc.
A számítások a szerveren történnek, a felhasználó böngészőjében a módosítás után azonnal frissülnek az adatok. Szerver és felhasználói oldalról is teljesen platformfüggetlen. A program nyílt forráskódú, GPLv2 alatt használható.
http://www.softwaregarden.com/products/wikicalc/
Ennyit mára a táblázatkezelők világából, ami nem jelenti azt, hogy ne lenne ennél sokkal több említésre méltó táblázatkezelő alkalmazás a szabad szoftverek között, de most ennyire futotta.
Reméljük a legfontosabb szereplők közül sikerült bemutatni néhányat, és talán új információkkal is szolgálhattunk.
Legközelebb a „maradék” irodai alkalmazásokat vesszük elő.

Szőke József

Szeged

Szabad szoftveres konferencia

A Szabad Szoftver Világnapnak több évre visszanyúló hagyománya van Szegeden. Egészen az idei évig csupán egy teremben folytak az előadások és kissé fapadosabb volt az egész. (Ezt szó szerint kell érteni, ugyanis a Kiss Árpád előadóban szó szerint fa padok vannak.) Az idei év azonban több szempontból is más volt, az egész rendezvény átkerült a Szegedi Tudományegyetem József Attila Tanulmányi és Információs Központjába. Ez azontúl, hogy emelte a rendezvény nívóját, azt is lehetővé tette, hogy az előadások több szálon – multithread :) - , több helyen folyjanak.

Az előadások listáját tekintve már rengeteg ismerős név köszönt vissza. Nem sorolom, hogy kik, mert valakit biztosan kifelejtenék, de aki az eddigieken ott volt, az sejti, hogy kikre gondolok. A korábbi évekkel ellentétben az idei nyílt forrású palacsintázás elmaradt és helyette a méltán népszerű pogácsa töltötte be a nasi szerepét. (A szegedi pogácsát még Richard M. Stallman is megdicsérte 2004-ben, amikor a szoftveres szabadalmak ellen érvelt.) Természetesen volt kávé is, de nem azért, mert az előadások unalmasak lettek volna.

Némi technikai fennakadás ellenére sikerült minden látogatót regisztrálni. Egyfelől így mérhetőbb az érdeklődés, másfelől nem kis könnyebbség volt a tombolahúzásnál. Aki persze nem nyert, az vásárolhatott az egyetemi ajándékboltban Szabad Szoftver Világnapos pólót, bögrét vagy lézerpointeres tollat.

A regisztrációnál mindenki kapott ajándékot, ez idén hűtőmágnes volt. Követve a szabadság eszméjét: szabadon választhattak a látogatók, hogy Tux-os, Démonos vagy GNU-s hűtőmágnest kének.

A hangulat szerintem remek volt és jó volt azt is látni, hogy a férfiak kevésbé voltak többségben a korábbi évekhez képest, az előadásokon számos hölgy is részt vett, melyek érdekesek voltak, bár némely témára néhányunk szerint kevés volt a 20 perc. Gyakorlatilag csak kedvcsinálásra volt elég.

Az előadások prezentációi letölthetőek a honlapról:

http://szszk.sed.hu/

A szervezői videók is itt lesznek elérhetőek később. Természetesen a Flosszine csapata is készített videókat, amiket a: http://videos.flosszine.org/ címen találtok.

Hogy lesz-e jövőre is ilyen (színvonalas) rendezvény? Ha rajtunk, szervezőkön múlik, akkor valószínűleg igen. Az idei konferencia lehetőségét pedig szeretnénk megköszönni a szponzorainknak és a természetesen az előadóknak.

Medve Zoltán

Budapest

Szabad Szoftver Konferencia és Kiállítás 2009

Az FSF.hu Alapítvány október végén rendezte meg a Szabad Szoftver Konferenciát és Kiállítást, amelyen mi is jelen voltunk.

Szalai Kálmán az OpenOffice.org újdonságairól, illetve a jövőjéről beszélt. Az előadás végén szóba került a sebesség és megtudtuk, hogy a fejlesztők rajta vannak a témán. Azt tudtam, hogy léteznek kiterjesztések a programhoz, de azt nem gondoltam volna, hogy több százról van szó.

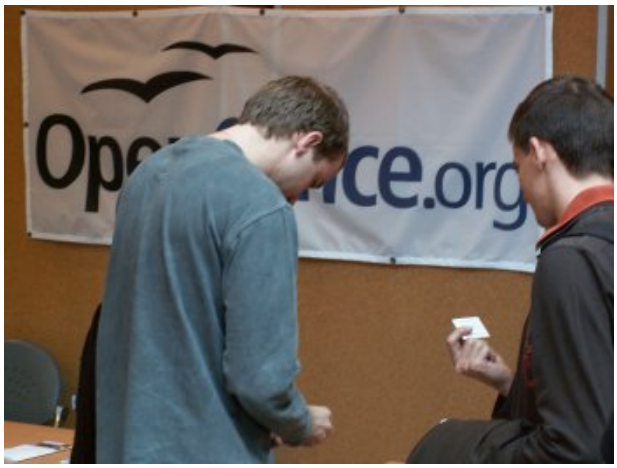
Szó volt egy nagy teljesítményű spamszűrőről is, ezért különösen érdekelt, hogy mire képes a Redis. Bártházi András előadása arról ugyan nem győzött meg, hogy a Redis 'megöli' a memcached-et, de mivel perzisztens tárolást biztosít, ez mindenképpen előny ott, ahol az adatok gyorsítótárazása mellett nem engedhető meg például az újraindítás miatti adatvesztés a cache-ből. Nem is hittem volna, hogy a Redisszel milyen egyszerűen el lehet készíteni egy Twitter klónt. Az egyik kérdésre adott válaszból azonban az is kiderült, hogy a hagyományos relációs adatbázisokat sem kell temetni, mert sok feladatra (pl.: webshop) azok az alkalmasabbak. A Redis erőssége a kulcs-érték alapú tárolás, ami másfajta logikát igényel a fejlesztőktől.

Mivel magam is fejlesztek egy nyílt forrású projektet, és bizonyára van jobb annál, minthogy kézzel vezessek egy TODO fájlt, kíváncsi voltam, hogy mit tud az Ubuntu mögött álló Canonical launchpad.net nevű fejlesztői platformja, amelyet nem régiben tettek elérhetővé. Bár a projekt a Bazaar nevű verziókövető rendszert preferálja, mindenképpen jó hír, hogy használhatom tovább a gitet.

Czakó Krisztián a Xen és a Heartbeat + DRBD segítségével mutatta meg, hogyan lehet összeházasítani napjaink divatos

buzzwordjét a virtualizációt a nagy rendelkezésre állással. A vállalkozás azért pikáns, mert a virtualizáció arról szól, hogy sok gépből egyet csinálunk, míg a HA arról, hogy egy funkciót több (legalább két) gép között osztunk meg. A szerverkonszolidáció és a HA keresztezésének az az eredménye, hogy két géppel nagy megbízhatóságú virtualizációt hozhatunk létre nyílt forrású programok segítségével.

Utolsó előadásként egy Java alkalmazásról volt szó, amely a matektanulást segíti. A GeoGebra eredetileg középiskolai segédletnek készült, de ma már az alap- illetve felsőoktatásban is használható. Máig emlékszem, hogy a paraméteres egyenleteket (vagy függvényeket?) nem igazán sikerült ab-



szolválni a középiskolában, arra pedig még jobban, hogy jó 17 évvel ezelőtt az ábrázoló geometria zh-kat komoly 0 pontra sikerült teljesíteni. Ma már kenném-vágnám ezeket is, annál is inkább, mert mindenféle előképzettség nélkül is sikerült otthon reprodukálni a háromszög köré írható kört. Azt azért hadd tegyem hozzá, hogy úgy, miután Papp-Varga Zsuzsanna megmutatta ezt is a demókkal tűzdelt előadásában.

A szervezők ebédet biztosítottak az előadók számára. Az időzítésem pedig aligha lehetett volna jobb: éppen sikerült elhalászni a végén az utolsó szelet csokitortát.

Szívesen megnéztem volna azt is, hogy eszik-e vagy isszák a Google Androidját. Érdekelt az is, hogyan lehet Linuxszal vékonyklienseket létrehozni. Szerettem volna hallani a KVM-ről, és még sok dologról, amiről a többi előadó beszélt, de egy fenékkel csak egy lovon lehet ülni.

Mivel az előadások párhuzamosan három színhelyen zajlottak (+ a workshopok egy negyedik teremben), ezért bizonyára többek megelégedésére szolgál, hogy a szervezők egy közel 200 oldalas PDF dokumentumot készítettek a konferencia előadásairól, amelyet ugyan az előadók írtak meg, de Zelena Endre és Kiss Gabriella munkája is kellett ahhoz, hogy egy jó minőségű PDF-ben legyen elérhető. A dokumentum a link1 címről tölthető le. Minden teremben videofelvételek is készültek, amelyek a FLOSSzine videomegosztó oldalára (link2) is felkerülnek.

A mintegy 400 regisztrált közül kb. 250-en vettek részt a rendezvényem, amelynek a BME Informatikai Központ adott otthont. A szervezők elégedettnek tűntek az érdeklődők létszámával.

A látogatók szavazhattak a legjobb előadóra. Höltzl Péter és Kis Gergely végeztek holtversenyben az első helyen. Az FSF.hu Alapítvány egy gyors „unplugged” tanácskozás után úgy döntött, hogy mindketten kapnak ajándékot. A FLOSSzine magazin ezúton gratulál nekik.

Szintén az előadások után derült ki, hogy az FSF.hu Alapítvány mely projekteket támogatja. Külön öröm volt nekünk,

hogy a FLOSSzine magazinnak is kedvezett a döntés.

Az aulában kiállítók várták az érdeklődőket, akik találkozhattak velünk is, a FLOSSzine csapatával. A visszajelzések alapján úgy tűnik, hogy akik eljöttek, nem bánták meg. Egy-két apróságot ugyan megemlítettek a HUP egyik blogjában, és úgy tűnik, a szervezők ezekre is odafigyelnek.

link1:
http://konf.fsf.hu/szszkonf2009_konferencia_kiadvany.pdf

link2:
http://videos.flosszine.org/

Sütő János

Filozófia

Szoftverpotlecs

A szabad szoftver, a nyílt forrás, mint fogalmak egyre szélesebb körben ismertek, mára nem csupán az informatika világában, de azon kívül is. Ugyanakkor az is elmondható, hogy az ezen fogalmakhoz kapcsolódó asszociációk számos esetben tévesek, vagy közel sem teljes körű információkra támaszkodnak.

A szabad szoftver, később a nyílt forrás eszméje köré szerveződő fejlesztői csoportosulások az elmúlt huszonöt esztendő alatt egy olyan mozgalmat hoztak létre, mely az internet rohamos terjedésének előnyeit felhasználva mind térbeli mind időbeli korlátait képes volt átlépni. Az eredeti kereteken messze túlnőve nem csupán olyan szoftvereket hozott létre, melyek felveszik a versenyt zárt forrású, kereskedelmi társaikkal, de a résztvevők laza együttműködése révén létrejött közösségek megalkottak egyfajta szokásjogon alapuló - írott, vagy íratlan formában létező - szabályrendszert.

A tulajdonosi (proprietary) szoftverek fejlesztése, vezetése, vagy éppen értékesítése kapcsán megszokott szemléletmódtól való gyökeres eltérés hatásai a szakmai területek túl is tetten érhetőek. Az open soure üzleti modell, a copyleft típusú licencszerzések, a hírnév motivációs erejének gyakorlatban bizonyított működőképessége olyan kihívások elé állította a - szabad szoftver mozgalom releváns - gazdasági és jogi környezetet kialakító szakembereket, melyekre csak részben sikerült megfelelni.
A következőekben a közösség működésének bizonyos jellegzetességi mögött meghúzódó elvekre, sajátosságokra igyekszünk rávilágítani.

Irányzatok

Eric S. Raymond a hackerfilozófia követőit 3 alapvető részre osztja[1] (fanatikusok, mérsékeltek és liberálisok) aszerint, hogy azok miként viszonyulnak magához a nyílt forrású fejlesztéshez, célnak, vagy eszköznek tekintik azt. Ugyanezen 3 kategóriát alkalmazza a tulajdonosi szoftverek, illetve azok ogyártóihoz való viszony szerint is. Az így létrejött 9 kategóriát az együttműködés különböző módszereinek alkalmazása okán tartja fontosnak megkülönböztetni.

Free Software

A Richard M. Stallman (RMS) vezette FSF az első – és sokáig egyetlen – jelentős szervezete volt a szabad szoftver mozgalomnak. Történetileg, főként RMS-nek tulajdonítva, a radikális irányzat képviselőinek gyűjtőhelye, amit azóta is annak tartanak a mozgalmon belül és azon kívül egyaránt. Mindez annak ellenére, hogy RMS maga tagadja, sőt nyilatkozatai alapján sem erősíthető meg a puszta ellenségesség a kereskedelmi szoftverekkel szemben, bár számos esetben a követők ezt mégis így értelmezték és teszik azt ma is.
Ami azonban nem vitatható el, hogy az FSF és a szintén RMS által alapított GNU projekt mind a filozófia megteremtése, mind a szabad szoftverek fejlesztését lehetővé tevő eszközök (licencek és fejlesztői eszközök) megalkotása terén olyan munkát végzett el, mellyel elévülhetetlen érdemeket szerzett. Ma mégis úgy tűnhet, hogy az idő és a körülmények túlhaladták RMS gondolatiságát, melynek megvannak a maga veszélyei.

Open Source

A kezdetek óta létező pragmatikus irányzat képviselői előbb a BSD (Berkeley Software Distribution) fejlesztések körüli csoportokban működtek közre aktívak (ám a számos terjesztés erejüket elemésztette) utóbb pedig a Linux megjelenése révén leltek bázisul szolgáló projektre. Az új operációs rendszer atyja Linus Torvalds, a kereskedelmi szoftverekkel szemben sokkal megengedőbbnek bizonyult RMS-nél és a szabad szoftver helyett is inkább a nyílt kifejezést használta1. Ez talán egy generációváltás is volt egyben a mozgalmon belül, ami máig tart és hatásuk több ponton is tetten érhető.

A különbségek nem elsősorban a vezető egyéniségek véleménykülönbségében keresendőek. Minden irányzat létrehozta a maga licencét (GPL, BSD, PAL, …), mely egyben a fenti kérdésekre adott válaszként is értelmezhető. Bár a nyílt forrású közösségen belül ma is a GPL a leggyakrabban használt licenc, annak egyes részeivel (különös tekintettel a 3-as verzióra) többen nem értenek egyet.

Közösségi jog

Annak ellenére, hogy a nyílt forrás – licencei révén – egyáltalán nem az elért eredmények az egyén, vagy egy csoport által való birtoklását, hanem annak a közösséggel megosztását és egyúttal továbbfejlesztését célozza, mégis létezik a tulajdon fogalma és a tulajdonlás rendszere, melynek megsértése tabu, ám nem a forráskód, hanem a projekt viszonylatában. A kérdés nem úgy merül fel, hogy egy adott

kódrészlet, vagy a teljes kódbázis kinek a tulajdona, hisz ez a kérdés egy közösségi fejlesztésű projektnél teljesen értelmetlen, hanem úgy, hogy ki a projekt tulajdonosa.

A kérdésre a választ Eric S. Raymond a következőképp fogalmazza meg[1]:

„Egy szoftverfejlesztési projekt tulajdonosa az a személy, akinek a közösség által elismerten kizárólagos joga van arra, hogy a program módosított változatait terjessze."

A fenti kérdésre a választ ez a definíció csak részben adja, hisz joggal vetül fel egy újabb kérdés, miszerint hogyan szerezhet egy személy vagy egy csoport a közösség előtt kizárólagos jogot arra, hogy egy adott szoftver módosításait kizárólagosan terjessze, miközben a copyleft biztosította felhasználói szabadságjogok a fejlesztők között is egyenlőséget teremtenek, mind a módosítás, mind pedig a terjesztés tekintetében. A hangsúly arra helyeződik, hogy a közösség milyen módozatokat ismer el ennek megszerzésére:

Ha a projektnek indulása óta egyetlen karbantartója van.
Ha a projektet annak tulajdonosa másra ruházza.
Ha egy elhagyott projekt tulajdonjogát szerezzük meg.

Érdemes megjegyezni, hogy a második forma nem pusztán lehetőség, de kötelezettség is, mivel a közösség erős nyomást gyakorol, hogy egy adott tulajdonosnak nem áll módjában kellő mennyiségű időt energiát áldozni a projektre, ha arra van jelentkező, a tulajdonjogot adja át. Mindemellett azonban hasonlóan erős a nyomás a volt tulajdonos munkájának elismerésére. A harmadik módozat esetén komoly erőfeszítéseket ildomos tenni a
projekt korábbi tulajdonosának felkutatására a közösség rosszallásának elkerülése érdekében.

Angolszász jog

A földbirtoklás angol-amerikai közjogi elve szerint háromféleképpen kerülhet birtokunkba egy földterület[1]:

„A határvidéken, ahol olyan földek találhatók, amelyeknek még soha nem volt tulajdonosuk, a tulajdonjogot kisajátítással szerezhetjük meg, tehát saját munkával kell a földet birtokba vennünk, be kell kerítenünk, és meg kell védenünk a jogcímünket."

„A régi településeken a földtulajdon átadásának szokásos módja a jogcím átruházása, vagyis a tulajdoni iratok átvétele az előző tulajdonostól. Ennél az elvnél lényeges a "jogcímek láncolata". A tulajdonjogot ideális esetben az bizonyítja, hogy az iratok és a jogcím átruházásai addig az időig nyúlnak vissza, amikor a földet eredetileg kisajátították."

„A közjogi elmélet arra is gondol, hogy a földre formált jogcím elveszhet (például ha a tulajdonos örökösök nélkül hal meg, vagy ha a gazdátlan föld jogcímláncolatának megállapításához szükséges iratok nincsenek meg). Az így elhagyottá vált földre passzív kisajátítással formálható igény – vagyis elfoglalja és műveléssel feljavítja a földet, mintha ő lenne az eredeti kisajátító."

A hasonlatosság egy projekt tulajdonlása és az angolszász jogrend földbirtoklási elvei között kézenfekvőek. További érdekesség, hogy a központi hatalom befolyása ezeken a területeken rendkívül gyenge, hisz arra vonatkozóan, hogy ki és milyen határokkal foglalhat területet ebben a virtuális térben nincsenek törvényi előírások2, viszont az erőforrások – mint amilyen a fejlesztői, tesztelői kapacitás, vagy a felhasználói tábor – kellően szűkösek, illetve a projektek értéke kellően nagy ahhoz, hogy a projekttulajdonosokat területeiket „körbekerítésére" és azok megvédésére kényszerítse.

Nooszféra

Az a terület, melyből egy rész elkerítésre kerül, mikor egy új – nyílt, vagy éppen zárt forrású projekt – elindul, illetve ami megművelésre kerül a projekt létezése során, aminek tulajdonjogáról, kisajátításáról, vagy éppen elbirtoklásáról beszélünk, a nooszféra.

A nooszféráról3 – mely fogalmat Edouard Le Roy használta első ízben[2] – Vlagyimir Ivanovics Vernadszkij4 (1863 - 1945) és Pierre Teilhard de Chardin5 (1881 - 1955) munkássága nyomán beszélhetünk, mint minden emberi gondolat teréről:

„Tűzkör húzódik az első felszikrázó gondolkodó tudatok körül. A felizzó pont megnagyobbodik. A tűz mindinkább teret nyer. Végül hatalmas izzás borítja el az egész bolygót. Egyetlen magyarázat, egyetlen szó méltó ehhez a hatalmas jelenséghez. …ez, a „gondolkodó réteg", amely a Harmadkor végén csírázott ki, és azóta végigárad a Növény- és Állatvilág felett: a Bioszférán kívül és afelett, a Nooszféra."

Pierre Teilhard de Chardin[3]

A nooszféra egyes területeinek használati joga, a felettük gyakorolt tulajdonjog kérdése, illetve a művelésük révén "termett" hírnév elosztása az, ami hackerkultúra egyik alapproblémája, melynek – a kultúra képviselői által alkalmazott – gyakorlati feloldása leginkább a John Locke tulajdonjogi

elveinek alapján írható le.

Locke-i tulajdonjog

Locke tulajdonjogi elveinek két alkalmazási területe lehetséges a hackerkultúra viszonylatában, az egyik a korábban említett nooszféra, a másik pedig az Eric S. Raymond által bevezetett ergoszféra, mely az ő megfogalmazásában a „munka szférája”, ahol a szabad szoftver mozgalmának résztvevői tevékenykednek, melyben az általuk alkotott projektek működnek. ESR véleménye szerint[1] gyakorlati jelentőséget csak akkor nyer a két fogalom közti különbség, ha „azt szeretnénk bizonyítani, hogy a gondolatok (a nooszféra elemei) nem birtokolhatóak, projektként való megtestesüléseik azonban igen”. Ezen felvetésnek a nyílt forrású fejlesztők szoftverszabadalmak elleni erőteljes fellépése és a szellemi tulajdonhoz való sajátságos viszonyuláskor van jelentősége.

Tilalmak

Az a tény, hogy a hackerek munkájukat jellemzően önként, anyagi ellenszolgáltatás nélkül végzik, nem csupán a közösség tagjainak munkához, illetve annak eredményéhez való viszonyát határozza meg, hanem a felszínes szemlélő viszonyát a kultúrához, melyben adott esetben nem lát többet, mint egy naiv kommunisztikus törekvést, melyben az „azé a föld, aki megműveli” elv érvényesül.
Ez a belső irányultság tehető felelőssé a kultúra normáinak, következésképp egyes tabuinak kialakulásért is.

Elágaztatás

Annak ellenére, hogy a licencek ezt semmilyen módon nem zárják ki, sőt látszólag támogatják is a forrás nyíltsága és szabad terjeszthetősége által, projekt elágaztatására csak kivételes esetekben kerül sor. Ez annyit jelentene, hogy ha egy fejlesztő – vagy még inkább egy fejlesztői csoport – elképzelései nem egyeznek meg a tulajdonoséval a projekt jövőjét, fejlesztési irányát illetően, akkor a forrás legfrissebb változatát lemásolva, azt egy új projektben hasznosítsák. Ezen módszernek azonban komoly hátulütői vannak.

Egyrészről némi idő elteltével az immáron két projekthez tartozó forráskód annyira eltávolodik egymástól, hogy nem lesz mód a közvetlen együttműködésre, adott problémák megoldásainak kölcsönös cseréire. Másrészről a fentiek nyilvánvalóan megosztják a fejlesztői erőforrásokat, mivel az eddigi egy projektben részt vevő fejlesztők kapacitása, most két projekten oszlik meg, valamint a felhasználói kör is kettéoszlik (rendszerint nem egyenlő arányban), ami az elérhető hírnév mértékét és a kettészakadt projekt darabjainak túlélési esélyeit is radikálisan csökkenti.

Az elágaztatásokat ezen racionális okok és az ezzel összefüggő normarendszer nem, vagy csak ritkán teszi lehetővé (amit a gyakorlat vissza is igazol) és akkor is indoklással kell szolgálni a közösség felé. A projekt – illetve a létrehozott szoftver(ek) – neve az ilyen esetekben (eltekintve jogi kötöttségektől) az eredeti tulajdonost illeti meg, hiszen az ő munkája révén ment végbe az eredeti – hírnév formájában megjelenő – tőke felhalmozása.

Terjesztés

Szintén a forráshoz való szabad hozzáférés, a fejlesztők alapvetően egyenrangú mivolta, illetve a licencelés lehetővé tenné bárki számára, hogy egy adott projekthez változtatásokat készítsen és azokat önállóan terjessze, ez azonban mégsem történik meg.

Ez egyfelől azon praktikus oknál fogva van így, hogy egy önálló javítás karbantartást igényel, mely annak fejlesztőét terheli, és viszonylag kevesek számára hozzáférhető, hiszen a széles felhasználói közösség nem törődik a forrás formájában terjesztett változatokkal, csak ez elkészült termékkel. Ez tehát azt jelenti, hogy bármennyire is hasznos az adott javítás, fejlesztés, annak felhasználása, így az érte megszerezhető elismerés mértéke is csekély lesz.

Másfelől ha a javítás valamilyen hibát okoz, azt a felhasználók már jellemzően nem a változtatást készítőjének, hanem a projekt, illetve annak tulajdonosának rovására írják. Ez a fajta hitelrontás, a hírnév csorbítása a közösség működése szempontjából megengedhetetlen.

Elismerés

A projekt tulajdonosa a projektben felhalmozott elismeréseket – amint arról még szó esik – a közösség szabályai szerint meg kell ossza a közreműködőkkel, ami rendszerint úgy történik, hogy a forrásállományok valamelyikében folyamatosan frissítik a közreműködők listáját.

Ebből a listából bármely hozzájáruló nevét eltávolítani – annak beleegyezése nélkül – kifejezett tabunak minősül, lévén mások hírnevének elorozásával jogtalanul halmozza fel magánál azt. A kapott elismerés tehát örök, ugyanakkor a hozzájárulást elsőként tartalmazó verzió megjelenésekor jellemzően külön is elismerik, növelve annak aktuális értékét.

Hasonlóképpen a befektetett munka elismerésének egy for-

mája a közösség azon elvárása, hogy ha valaki egy elhagyott projekt tulajdonjogára kíván szert tenni, a lehetőségekhez mérten tegyen meg mindent a korábbi tulajdonos felkutatására. Ennek két kézenfekvő magyarázata is adódik. Az egyik, hogy a hathatós próbálkozás növeli annak esélyét, hogy az eredeti tulajdonos elérhetővé válik és maga mond le a tulajdonjogról, ami a projekt átruházásává minősíti át a helyzetet, ami később nem vitatható. A másik, hogy a próbálkozások mértéke egyenes arányban van az új tulajdonos elbirtoklási jogával és fordítottban a az eredeti tulajdonos kései felbukkanása utáni jogával eredeti projektjére.

Pfeiffer Szilárd

Hivatkozások:

[1] Erik S. Raymond. A katedrális és a bazár. Kiskapu Kft., Budapest, 2004.

[2] Edouard Le Roy. Les origines humaines et l'évolution de l'intelligence. 1928.

[3] Pierre Teilhard de Chardin. The Phenomenon of Man. Harper Torchbooks, 1961.

Lábjegyzetek:

1 Ennek magyarázatát az angol free kifejezés félreérthetőségében (szabad/ingyenes), illetve az RMS által képviselt véleménnyel való összefonódásban adta meg Linus Torvald

2A cikk elkészültekor (2009. december 6.) az Európai Unió államaiban ugyan nincs lehetőség számítógéppel megvalósított találmányok szabadalmi védelmére (szoftverszabadalom), azonban más technikai értelemben vezető országok, mint az USA, alkalmazzák ezt a módszert mely rendkívül komoly hatást gyakorolhat a közösség működésére

3 a görög νους (jelentése: elme, tudat, lélegzet) és a σπαιρα (jelentése: gömb, égbolt) összetételéből származik a bioszféra és az atmoszféra mintájára

4 Ukrán geológus, ásványkutató, a bioszféra fogalmának megalkotója.

5 Francia filozófus, jezsuita lelkész, aki tanulmányait a paleontológia és a geológia területén végezte.

Forrás:

A cikk egy dolgozat részét képezi, mely a http://www.pfeifferszilard.hu/ címen érhető el.

Tudtad-e?

Tudtad-e, hogy a http://www.fsdaily.com/ oldalon kitűnő hírportál található. Az oldal free és open source szoftverekkel foglalkozik, a híreket, cikkeket különbözőképpen csoportosítva is megjeleníthetjük.

Tudtad-e, hogy a http://filext.com/ oldalon remek adatbázis található, így az oldal segítségével megtudhatod, hogy melyik fájlkiterjesztés melyik alkalmazáshoz tartozik?

Tudtad-e, hogy a http://www.hogyan.org/ olyan magyar nyelvű oldal, ahol egyaránt találhatsz hasznos leírásokat Linux és Windows témában? Debian, Fedora, Mandriva, openSUSE, PCLinuxOS, Ubuntu, valamint Windows Server2003, 2008, XP, Vista, és Windows 7 témánként csoportosítva. Hasznos leírásokat bárki küldhet, regisztráció után.

emuláció Linuxon

Enterprise 128

A hagyományőrző jelleget szem előtt tartva kevés Linux-felhasználónak lehet oka panaszra: a retro divat mindent falon áthatol, a legtöbb szakmai területen azonnal megtalálja a helyét. Eennek megfelelően a Commodore 64, az Amiga család és a ZX48 gépek is sokadik reneszánszukat élik a modern, PC-s korszakban. Természetesen sok egyéb legendás számítógép és játékkonzol emulátora is bérelt hellyel rendelkezik a téma iránt érdeklődő, rajongó olvasók merevlemezén, én pedig a méltatlanul elhanyagolt Enterprise 128 témáját fogom kissé körberágni.

1.ábra Az Enterprise 128...

A bevezetőben említett masinák fénykoráról nehéz lenne száraz szemmel elmélkednem, hiszen az a szeretet, ami körülfonja ezeket az öreg darabokat, szavakkal szinte leírhatatlan. Emiatt feleslegesnek vélem a nyolcvanas-kilencvenes évek számítástechnikájának hangulatát felidéznem: ha valaki használt éles környezetben bármilyen 8/16 bites „csodát”, akkor csak untatnám a nyilvánvaló és a már oly sokszor körbeírt dolgokkal. Akinek viszont nem adatott meg, hogy saját bőrén tapasztalja meg a számítógépek otthoni szegmensének

2.ábra ...ilyen szívélyesen fogad (Ubuntu asztalon)

felemelkedését, úgysem értékelné a mondanivalómat.
Talán érdemesebb rögtön a lényegre térnem: 1986-ban Magyarországon is forgalomba kerültek az Enterprise Computers International GMBH személyi számítógépei. Saját korának meghatározó hardveréről beszélek: a Z80A CPU köré épült Enterprise 64 ill.128-at Nick grafikai processzorral, Dave hang- és I/O vezérlőcsippel, (jellemzően) 128 Kbyte központi tárral vértezték fel. Sajnos a mértéktelen fejlesztési költségek, valamint a kritikán aluli marketing munkálatok a gép korai bukását eredményezték, azonban a kezdeti, európai kereslet (legfőképpen a hazai forgalom) sok éven át életben tartotta a fejlesztői és végfelhasználói csoportokat. Nem mellékes információ, miszerint az Enterprise a saját szoftvereit gyakran a Clive Sinclair-féle korszakalkotó ZX48 Spectrumtól „örökölte”, ezért a két gépet sokan közeli rokonként tartják számon. Az átültetések természetesen a két platform egyező Zilog központi egységének lehetőségeire támaszkodtak.

A körítés...

Az Enterprise formája rendkívül egyedire sikerült. A szokatlan, több helyen megtört és lekerekített „karosszéria”, a színes billentyűzet, a beépített botkormány nem mindennapi küllemmel ruházza fel a gépet. De a kiszemelt „áldozatunknak” nem csak a fizimiskája kirívó, hiszen buszrendszerének kivezetései sem szokványosak: a szokásos TV/monitor- és hangkimeneten túl gyárilag rendelkezett soros porttal, hálózati előkészítéssel, kazettás magnó vezérléssel, printer csatlakozóval, Expansion és Rombay kiegészítő lehetőséggel. Ezek közül az utóbbinak jutott leginkább kényes feladat: a

Rombay Cartridge hardvermodulok által vált igazi mindenessé az angol remekmű (enélkül csak szövegszerkesztő funkciót tudott ellátni, Enterprise Word Processor néven).
Fontos tudnivaló, miszerint a kilencvenes évek hajnalán a számítógép hazai támogatását a Novotrade Rt. vállalta fel, így az alkatrészek és programok terén e névvel minden felhasználó összefutott előbb-utóbb. Az értékesítési körbe ekkortájt léptek be a néhai a Centrum áruházlánc tagjai, de akárhol is vásárolt a delikvens, az árak mellett a megrendelések átfutása sem volt túl meggyőző. Tisztán emlékszem, ahogyan egy egyszerű Joystick fordító érkezésére nagyjából két hónapot kellett várni a debreceni boltokban, 1988 környékén... Ennek ellenére az a sajátos hangulat, ami egy apró (BASIC) program megírását kísérte, vagy épp a kornak megfelelő játékok előtt ülve jött át a TV képernyőjén keresztül, mindent feledtetett.

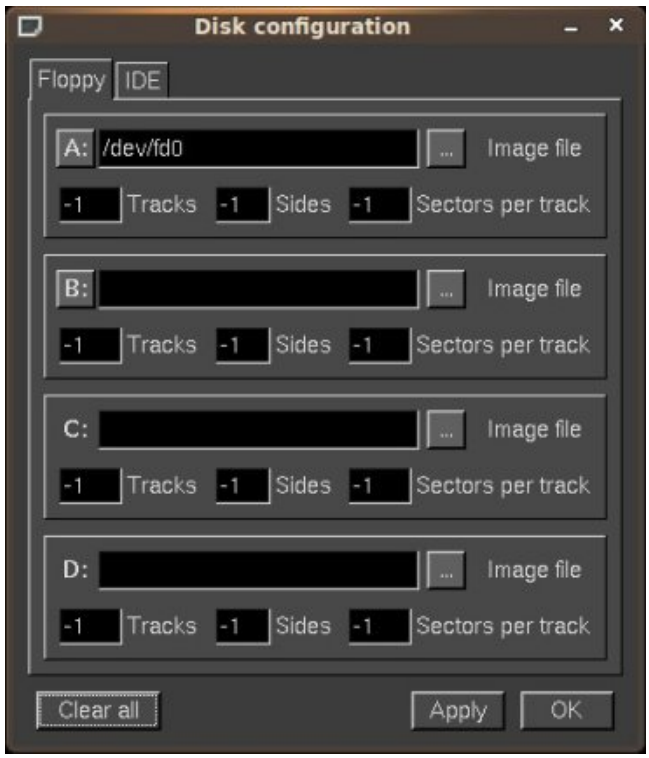
Miért lenne jó emulálni?

Egyrészt azért, mert Magyarországon rengeteg néhai Enterprise felhasználó van. Nekik biztosan sokat jelent újra látni az öreg diagnosztikai képernyőt, vagy éppen egy régi kódot lefuttatni a Basic / Assembler értelmezőn. Másrészről az Enterprise 128 palettájáról elég sok klasszikus játékprogramot csalhatunk át Linux platformra, hiszen a ZX48 szoftvereinek 70%-a ezen a gépen is létezett - néha jobban kidolgozva, mint az eredeti verzió. De van egy harmadik indok is: a világ legjobb emulátorai hazai fejlesztésű szoftverek. Nem meglepő tehát, ahogy a legértékesebb rajongói oldalak is Magyarországhoz kötődnek: az érdeklődőknek barátilag javaslom két URL észben tartását. Egyik a ep128.hu oldalára, másik a enterpriseforever.org-ra mutat: bármelyik lapról indulva letölthetőek a számítógép legnépszerűbb játék- és felhasználói programjai. Érdemes összegyűjteni e muzeális kódsorokat, és struktúrahelyesen kibontani mindet egy írható területre, az emuláció témájánál úgyis szükségünk lesz néhány „tesztprogramra”. Az Enterprise Forever weboldal aktív és figyelemre méltó szakmai fórummal rendelkezik!

3.ábra A virtuális billentyűzet beállítása

Mikor térünk a lényegre?

Rögtön, viszont néhány dolgot még érintenem kell. Az emulációt illetően négy projekt jöhetne szóba, név szerint a Multi-Machine Emulator, az Enter, az ep128emu (Varga István) és az Ep32 (Vincze Béla György). Az első lehetőséget automatikusan ki fogom hagyni, mivel az ide vonatkozó hatásfoka véleményem szerint ritkán ér el 70%-ot. A második szoftver finoman szólva is szakállas, harmadik és negyedik szoftver viszont már egészen használható, ráadásul az ep128emu rendelkezik natív, linuxos kiadással is. Az Ep32 erősen Win32 alapú, így ezt a szoftvert sem fogom érinteni (holott elérhető a forráskódja, csak eddig még senki nem vállalkozott DirectX specifikus sorok linuxos portolására). Marad tehát az ep128emu, ahol nem célom átfogó bemutatást tenni, mivel erre nem lenne elegendő háromszor ennyi nyomtatott oldal sem. Az információkat sokkal inkább kezdő lökésnek, esetleg iránytűnek szánom.



4.ábra Íme, a floppy emulációja

Színpadon a mai vendégünk: ep128emu

Meggyőző multiplatformos munkáról van szó: a GPL licencű emulátor gyors, letisztult és lényegre törő. Hatékony grafikus interfésszel rendelkezik, a futási értékek beállítását megkönnyítendően. Telepítése sem bonyolult: látogassunk el a enterpriseforever.org oldalra, ahol az ep128emu előre fordított általános binárisként, disztribúciókhoz köthető csomagként és forráskódként egyaránt elérhető (a cikk írásakor fellelhető legfrissebb kiadást v2.0.7 azonosítóval jelölik). A disztribúciókhoz köthető csomagokat nem érinteném, ezek

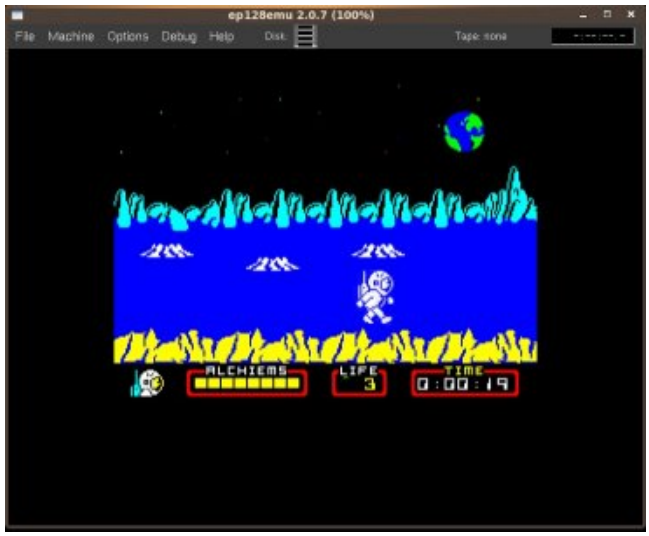
minden felhasználó számára egyértelműen üzembe állíthatóak, a terjesztés csomagkezelőjét használva. Nézzük inkább a maradék két utat!

Forrásból épített emulátor

Mindenek előtt beszéljünk kicsit a teljesítendő függésekről: SCons, FLTK, PortAudio, Python, SDL, Lua, libsndfile, dotconf. Nem nagy lista, végtére is ezeket majd' minden nagyobb terjesztés tartalmazza alapból, de legalább a csomagforrások listáiban biztosan megtalálhatóak. Két fontos dologra azonban ki kell térjek: az ide vonatkozó dokumentáció szerint, függések terén az SDL könyvtárak, valamint az FLTK kitüntetett figyelmet érdemelnek. SDL vonalon a v1.2.10 kiadástól kezdve (beleértve ezt is) mindentől tartózkodnunk kellene, szóval a legmagasabb verziószámú

5.ábra Prince of Persia, Enterprise verzióban

felhasználható SDL könyvárat v1.2.9-nél szabjuk meg. FLTK esetében sok disztribúciónál fellelhető bináris csomag nem megfelelő, mivel ezt a porterek gyakran az --enable-threads opció nélkül készítik, és ez a forrásból épített emulátor esetén nem használható függést eredményez. Szóval az FLTK függést leszünk szívesek mi magunk forrásból felépíteni, az előbb leírt kapcsolóval. Ha minden adott a munkához, adjuk ki a "cvs -z3 -d :pserver:anonymous@ep128emu.cvs.sourceforge.net: /cvsroot/ep128emu checkout -P ep128emu2" parancsot, majd a forrásban a megkívánt fordítót hívjuk segítségül: scons utasításra felépül a program. A létrejött binárisokat tegyük ki valamely elérési útra (pl. /usr/bin), majd a honlapról töltsük le a ep128emu_roms.bin állományt (ha úgy tetszik, az Enterprise BIOS-át), amit másoljunk a személyes mappánkban (általunk) létrehozandó ~./ep128emu/roms útjára. Adjuk ki a makecfg parancsot, majd a feltett kérdésre válaszolva kérjük a konfigurációk használatához a ~/.ep128emu mappát. Készen is vagyunk: ha konzolra gépeljük az ep128emu parancsot, indul az emulátor. De kis türelmet még kérnem kell.



6.ábra Nodes of Yesod

Az általános bináris csomag

Végtelenül egyszerű feladatot kapunk, ha a forráskóddal nem boldogulnánk. Töltsük le a bináris archívot (”Downloads / All releases” menüpont, majd kérjük az ep128emu-verzió-linux.tar.bz2 fájlt), amit bontsunk ki valamilyen írható útra. A benne található ELF binárisok egyeznek a felépített forráskódnál létrejöttekkel. Tegyük a rom állományt a bináris útján lévő /roms mappába, majd visszalépve egy szintet adjuk ki a ./makecfg parancsot, végül jöhet a ./ep128emu. Az emulátor rom fájljai elérhetőek egy zip állományban is, ha ezt szeretnénk használni megtehetjük: a zip archív tartalmát csomagoljuk ki a személyes mappánk ~/.ep128emu/roms útjára. Eredmények a mellékelt képeken... Nézzük inkább a használatot!

A puding próbája

Az indító ep128 (vagy ./ep128) parancsnak akad néhány érdemesebb kapcsolója. Mivel a projekt alapból aktívan használja az OpenGL könyvtárakat, így ennek kikapcsolása szükségessé válhat, ha a megjelenítő vezérlő kompatibilitásával problémánk akadna (pl vibráló programmenü integrált Intel grafikus vezérlővel). Íme a triviális megoldás, használjuk a -no-opengl opciót! Másik fontos kapcsoló pedig a GUI színösszeállítását változtatja meg: -colorscheme <X>, ahol X értéke 0 és 3 között lehetséges. A megfelelően felparaméterezett utasítást kiadva feléled a virtuális Enterprise: diagnosztikai / üdvözlő képernyőjére válaszoljunk a „szóköz” gombbal, mire a rendszerdiagnosztika eredménye megjelenik a munkaterület bal felső sarkában. A Basic értelmező ekkor már működik, így a kiöregedett nyelv szintaktikáját szem előtt tartva bárki elkészítheti (és el is mentheti) saját programját. A gyári játékprogramok használatához viszont a Machine/Tape menüpontban érinteni kell az emulált kazettás magnó vezérlését (állomány betöltése, magnó indítása, megállítása stb.), illetve az Options/Disk/Configure menüben a lemezképek használatát (lemezkép betöltése, illetve HDD

7.ábra Beach Head

kezelés). Bármilyen megfelelő *.tap lenyomatot betöltve mindössze ki kell adni a load -illetve ízlés szerint a start parancsot (utóbbi egyszerűbb: Enterprise esetén ez az F1 funkcióbillentyűre van programozva), majd el kell indítani a virtuális magnót (ha gyorsbillentyűkkel szeretnénk megtenni mindezt: indítás Alt+P, stop Alt+O). Lemezképek esetén hasonlóan egyszerű teendőnk akad, hiszen a lemezképet meghatározva, majd kiadva a start utasítást, megjelenik a lenyomat tartalma. Valódi lemezt is használhatunk, nem csak képállományt: példaként /dev/fd0 eszköznevet gépelve a lenyomat helyére bármilyen, floppy-n lévő Enterprise program életre hívható. Az Options/Keyboard Map szintén fontos lehetőségeket tartalmaz, ahol is beállítható a billentyűkiosztás, valamint a virtuális botkormányok kezelése. A menük további legfontosabb lehetőségei: Options -> prioritás beállítás, képernyő felbontás váltása, megjelenítési paraméterek beállítása (akár fekete-fehér képet is kérhetünk, a'la Junoszty TV), hangszolgáltatás finomhangolása. Machine -> futási sebesség beállítása, reset (gyorsbillentyű F11 illetve CTRL+F11). A géphez mindazonáltal előre definiált konfigurációk is betölthetőek a File/Configuration/Load menüből (EXDOS, TAPE, stb). A képek között reményeim szerint látható lesz, hogy kiváló megjelenés mellett bármelyik régi kedvenc feléleszthető az ep128emu segítségével.

A többletről

Természetesen minden Enterprise emulátor többre hivatott, mint kizárólag a játékokat lefuttatni rajta. Ezen a ponton az ep128emu szintén példaként emelhető ki: kifejezetten olyan alkalmazás, ami minden igényt kielégít. Folyamatos a fejlesztése, kifejezetten sok konfigurációt ismer és részletes beállítási lehetőségeket biztosít (a memória szegmenseire akár egyenként betölthetőek a ROM lenyomatok, az EXDOS működését is kiválóan megoldja, de a Debug funkció is felettébb értékes). Ha valaki többet szeretne megtudni az utánzott gépről, vagy a programozásáról, esetleg a köré épült kultuszról, akkor használja a két ajánlott linket: a hasznos fórumszálak tucatjait böngészve valószínűtlennek tűnik bármilyen megválaszolatlan kérdés...

Megéri a fáradtságot mindez?

Úgy gondolom, igen. Talán van, aki „átlát a szitán”, és rájön, hogy az Enterprise 128 nekem is sokat jelent. Valószínűleg az is sejthető, hogy miért szeretem ennyire: egyrészt én is büszke tulajdonos vagyok, féltve őrzök két felújított példányt. Másrészt pedig régi barátság fűz hozzájuk: éppen csak nyolc éves voltam, amikor a szüleim megleptek egy ilyen csodával (akkortájt értékes ajándék volt, hiszen egyikük közel egy havi keresetébe került). Az eltelt 23 év alatt olyan mélyen szívódott fel bennem a masina élménye, hogy sok mindent képes vagyok megtenni a tizenéves fejjel megírt programom újbóli megtekintéséért. De attól sem idegenkedek, hogy a Beach Head előtt ülve órákig nosztalgiázzak a korabeli háborús játékkal. Aki hozzám hasonlóan érez, ne fogja vissza magát és próbálkozzon bátran! Segítségképpen néhány régi kedvencem memóriatérképét „lefényképeztem” annak betöltött állapotában. Ezeket a lenyomatokat - másokkal együtt - a kovi.uw.hu oldalon elérhetővé tettem egy apró tarballban. Az emulátor Load snapshot menüjét használva azonnal használatba lehet venni a kiválasztott darabot! Végezetül a számítógép nyers, technikai adatai:

A gép technikai adatai
-Z80A CPU, 4MHz üzemi órajelen
-48 KByte ROM, a fordítók Cartridge kazettákon
-64/128 KByte RAM (közel 4 MByte-ig bővíthető)
-Nick grafikus vezérlő áramkör
-256 szín egyidejű használata
-Maximum 672x512 px felbontás „sorváltott” módban
-Dave hang-, memória-, és periféria vezérlő
-3 hangcsatorna, fehér-zaj generátorral
-Szabványos QUERTY billentyűzet, színjelöléssel

Kovács Zsolt

Szerencs 2009

Kótyagos pingvintől a Linux Akadémiáig

A Linux tábor ötlete 2000. nyarán, más hasonló rendezvények láttán merült fel először. Az ötletet 2001-ben Czakó Krisztián (Slapic) valósította meg, és ő volt az, aki éveken át nagyrészt egyedül foglalkozott a szervezési-lebonyolítási feladatokkal is.

Az oktatók a tábor legelső pillanatától vállalták, hogy munkájukat „szerelemből”, avagy elhivatottságból, anyagi ellenszolgáltatás nélkül végzik. Így van ez azóta is: kosztért-kvártélyért dolgozunk.

Mindvégig fontos szempont volt számunkra, hogy használhat

ó és naprakész gyakorlati ismeretekkel gyarapítsuk a résztvevők tudását. Ehhez olyan helyszínt kellett találni, ahol megfelelő számú számítógépterem, illetve előadóterem található, és még szállás is van a közelben. Ehhez társult még az a (kissé rejtett) cél is, hogy Tokaj-hegyalján minőségi borokkal is ismerkedjen meg a közösség. Slapic már ismerte a Szerencsi Középiskolai Kollégiumot, és Szabó Gyula (bácsi) pincészetét, ezért úgy döntött: itt próbálunk tábort szervezni.

Az akkor még alig két hónap alatt megszervezett – egyhetesre tervezett – tábor sikere a vártnál nagyobb volt. Ráadásul az eredetileg maximum 60 főre tervezett tábor férőhelyeit egy szervezési félreértés miatt ennél jóval alacsonyabbra kellett korlátozni. Szerencsére az általános iskola és a szakmunkásképző géptermei, valamint a kollégium gépterme kisegített minket.

Már az első évben is három különböző témával (kezdő-haladó-profi) külön-külön csoport indult a jelentkezők tudásszintjének és kéréseinek figyelembe vételével. A tábori póló és a kihagyhatatlan borkóstolás a kezdetek óta része volt a programnak.. Szintén hagyománnyá vált, hogy a tábor záró vacsoráját szabad téren, bográcsban, a tábor szervezője saját kezűleg, a jelenlévők segítő hozzászólásaitól övezve készíti el. Ez az esemény talán segít közelebb kerülni egymáshoz, emberközelibbé tenni a Linuxot. Az egy hét hamar eltelt, és sok tanulsággal szolgált. Nagyon sok pozitív visszajelzés érkezett, mely megerősítette a hitet, hogy szükség van ilyen jellegű rendezvényre.

A további években több változás is történt a táborban: 2002-től kétszer egy hetes tábort szerveztünk: először a második hetet csak mint lehetőséget hirdettük meg, de miután az első hétre minden hely elfogyott, hamar betelt a második is. Évekig kéthetes volt a tábor, mindkét hétre sikerült kiemelkedő tudású szakembereket meghívni előadónak. Később sor került arra is, hogy nemcsak a közeli linuxos ismerősöket, hanem a szakma által elismert egyéb szakembereit is meghívjuk a táborba.

A tábor fennállásának 5. évfordulójára Gyula bácsi pincéjéből saját bort palackoztunk, és minden résztvevő vihetett belőle emlékbe.

A tábori forma 2007-ben már nem működött elég jól, megcsappant az érdeklődés.

Ennek köszönhető a tábor újragondolása és teljes megújulása. 2008-ban a csapat kemény magja hosszas tervezgetés után elhatározta, hogy professzionális szintre emeli a tábort:

létrehoztuk a Linux Akadémiát. A csapatban azok a régi elkötelezett (elvarázsolt?) linuxosok dolgoznak, akik az évek során a táborban is oktattak, vagy más területen vettek részt a szervezésben.

A Linux Akadémia első évében az LME támogatásának segítségével kedvezményes részvételi díjat tudtunk kigazdálkodni a jelentkezőknek. Talán ennek is köszönhető, hogy 60 fő körül volt a létszám. A résztvevők kb. a fele kezdő csoportba jelentkezett, ahol szintfelmérés után két csoportban folyt az oktatás, 2-2 oktatóval, külön gépteremben. Változás volt az előző évekhez képest, hogy – bár kezdő szinten továbbra is tanfolyamként működtünk – a haladóknak konferencia-előadásokat szerveztünk, melyre vendégelőadókat is meghívtunk a szakma kiválóságai közül. Voltak elméleti előadások másnapi gyakorlati foglalkozással összekötve, illetve délutáni előadások is, melyekre bárki (a kezdők is) beülhetett és meghallgathatta, akit érdekelt.

Újítás volt szállásügyben, hogy a Huszárvár Szállóban is lehetett szobát foglalni. Az azonban látható volt, hogy továbbra is a kollégiumi szállásra volt a legtöbb jelentkező. Ennek valószínűleg nemcsak az olcsóbb árfekvés az oka, hanem a közösségi szellem: ez a rendezvény központja. Itt az aulában jönnek össze az előadások után a résztvevők konzultálni egymással és az előadókkal: sokszor hajnalig folyik az eszmecsere. Lehetőséget kaptunk arra is, hogy esténként használhassuk a gimnázium kiváló minőségű focipályáját, így egy kis testmozgás is alkalmas a résztvevők összecsiszolódására, és persze a testgyakorlásra is.

A Linux Akadémia második évében igyekeztünk tanulni a hibákból, változtatni azokon a dolgokon, amelyek előzőleg kicsit döcögősebben mentek. Sokkal előbb elkezdtük a szervezést (már márciusban), és ez a jelentkezők számában is megmutatkozott: 82 jelentkező volt. A gazdasági válság miatt az is felmerült, hogy az akadémián dolgozó szervezőknek és oktatóknak fizetniük kell a saját részvételi díjukat, ha nem lesz elég jelentkező. Az elhivatottságot mutatja talán, hogy a kérdés feltevése után sorra érkeztek a válaszok: mindenki hajlandó lett volna fizetni azért, hogy taníthat! Szerencsére erre nem került sor, mert rekord számú jelentkezés érkezett (már nemcsak a Kollégiumot, hanem a Diákszállót is megtöltöttük). A részvételi díjat minimális mértékben emeltük csak, hogy a jelentkezőknek ne jelentsen olyan nagy terhet. Örvendetes, hogy nagyon sokan új jelentkezők, ismeretlen arcok voltak, és köztük nagyszámú kezdő. A Kollégium vezetésének segítségével két este is sikerült „csapatépítő” grillvacsorát szervezni a csodálatos parkban. A tapasztalat azt mutatja, hogy ezt a következő években is megpróbáljuk, esetleg már az első este, hogy az ismeretlen emberek hamarabb beilleszkedjenek a közösségbe.

Több résztvevő – és a szervezők egy része is – döntött már úgy, hogy családostól érkezik. A Kollégium parkjában játszótér és homokozó, kosárlabdapálya is van. A környező utcákon több játszótér is található, és ami Budapesten szokatlan, a gyerekek az utcán is békésen biciklizhetnek. Későbbi terveinkben szerepel, hogy a családoknak a családfő délelőtti elfoglaltsága idejére programokat szervezzünk.

Összegzésként: sikerült egy olyan rendezvényt „teremteni”, ahol a linuxos közösség tagjai(nak egy része) rendszeresen találkozhatnak, eszmét cserélhetnek és tudásukat bővíthetik. A „táborlakók” közül sokan vannak, akik évek óta visszajárnak, rendszeres nyári program náluk a részvétel. Az oktatói „mag” is évek óta ugyanaz, de minden évben sikerül új vendégelőadókat meghívni. Elmondhatom, hogy az oktatók is várják évről évre a találkozást.

A Linux Akadémiát jövőre is megrendezzük: 2010. július 4-10-ig. A helyszín, az oktatók, a színvonal ugyanolyan lesz, mint eddig is várunk mindenkit szeretettel.

szalaimicó

4. rész.

Hello Window!

Már a legegyszerűbb felhasználói felület láttán is könnyen belátható, hogy a korábbi példaprogramjaink kissé sántítanak, mégpedig abban a tekintetben, hogy nincs olyan valós életben is használt program amiben egy-egy widget lenne csupán. Ha viszont több widgetet szeretnénk elhelyezni egy ablakban kézenfekvő kérdés, hogy miként tudnák őket a felületen csoportokba rendezni. Erre a kérdésre keressük a választ ebben a részben.

2. Fogalmak

A korábbiakhoz hasonlóan itt is érdemes először tisztázni néhány alapfogalmat és csak utána kezdeni bele az érdemi munkába.

2.1. Konténerek

A widgetek a felületen történő csoportokba csoportokba rendezése konténerek (container) segítségével valósul meg. Ezek olyan láthatatlan widgetek, melyekbe más widgeteket helyezhetünk (pack).

A GtkContainer, avagy Gtk::Container egy önmagában nem használható absztrakt osztály, mely csupán ősül szolgál minden olyan származtatott osztálynak, melyet widgetek tárolására akarunk használni.

Alapvetően két ilyen leszármazott osztálytípussal találkozhatunk a későbbiekben. Ezek a bin és box ősosztályok, melyek

5.ábra Prince of Persia, Enterprise verzióban

maguk is absztraktak és abban különböznek egymástól, hogy hány elem tárolására alkalmasak. Előbbiek összesen egyére, míg az utóbbiak akárhányéra.

2.1.1. Egy elemű konténerek

A GtkBin jelentősége a további származtatásoknál jelentkezik majd, hisz az olyan nélkülözhetetlen típusoknak, mint az ablak (GtkWindow), a gomb (GtkButton), vagy a frame (GtkFrame) mind a GtkBin az ősosztálya.

2.1.2. Több elemű konténerek

A felületi elrendezés kialakításakor játszik fontos szerepet, hisz a benne található widgetek – amit a GTK konténer gyerekeinek (children) nevez – elrendezésén túl azok méretét és konténeren belüli pozícióját is meghatározza. Ilyen típusok például a horizontális, vagy vertikális rendezést biztosító boxok (GtkHBox, GtkVBox), vagy a táblázatos megjelenítést szolgáló GtkTable.

2.2. Méretezés

A GtkContainer osztály legfontosabb funkcionalitása – melyet minden származtatott osztály is felhasznál – az, hogy meg tudja határozni a benne található elemek méretét. Ezt persze nem teljesen önállóan teszi, hanem megkérdezni a benne található widgeteket, hogy mekkora helyre lenne szükségük. Minden egyes widget saját hatáskörben állapíthatja meg, hogy mekkora az a vízszintes, illetve függőleges kiterjedés, ami az igényeinek legjobban megfelelne. Ezt az méretigényt nevezi a GTK size requestnek. Ez a mechanizmus fentről lefelé, azaz a gyökértől a levelek felé terjed abban a fa hierarchiában, melynek gyökere az ablak, közbülső elemeit a konténerek, leveleit pedig a widgetek alkotják.

A legegyszerűbb és legáltalánosabb eset tehát az, hogy a konténerek – amilyen maga az ablak is – összeadják a gyermekeik (a fában közvetlenül alattuk lévő elemek) méretigényét és azt sajátjukként propagálják. A valóság azonban nem ilyen demokratikus. Minden konténernek lehetősége van arra, hogy a benne lévő elemek felé – egy az eredeti igénytől esetlegesen eltérő – méretet (size allocation) adjon vissza mellyel gazdálkodniuk kell és amely rájuk nézve kötelező érvényű.

2.3. Elrendezés

Normális esetben egy ablak rendelkezésre bocsájtja azt a területet, melyet a benne lévő widgetek igényeltek. A kérdés nem is abban áll, hogy me legyen akkor ha pont annyi hely van amennyi kell, hanem sokkal inkább abban, miként jelenítse meg a konténer a saját widgetjeit, ha több a hely amennyire feltétlenül szükség volna. Ilyen eset például akkor állhat elő, ha ez ablak átméretezhető és azt a felhasználó nagyobbra nyújtja, mint amekkora hely a benne lévő widgetek kirajzolásához minimálisan elégséges.

Ezt az esetet szabályozzák azok a paraméterek (pl: fill, expand, pack-type, melyek tulajdonképpen sem a konténerhez, sem pedig a benne tárolt widgethez nem tartoznak, hisz a kettejük viszonyát határozzák meg. Megadásuk akkor történik, amikor egy widgetet szeretnénk egy konténerben – például egy boxban – elhelyezni.

A konténereket és ezen belül a boxokat leginkább úgy képzelhetjük el, mint egy dobozt – vagy ha programozási szakszóval akarunk élni vermet – melynek mindkét végére lehet pakolni. A verem hasonlat már csak azért is helytálló, mert az egyes elemek verembe történő elhelyezése csak egymást követően, csak egymásra lehetséges. Annyiban viszont sántít a példa, hogy a GTK esetén rendkívül ritka – bár egyáltalában nem lehetetlen – hogy elemeket vegyünk ki egy konténerből.

3. Alapműveletek

Nyilvánvaló igény, hogy ha már vannak tárolóink és azokhoz kapcsolódóan widgeteink, akkor azokkal műveleteket lehessen végezni. Az sem túl meglepő, hogy ezt a különböző típusú konténerek esetén másként kell megtenni. Itt csak a legalapvetőbb műveleteket és típusokat vesszük sorra, azokat is csak abban a mértékben mely a koncepció megértéséhez szükséges.

3.1. Létrehozás

A GtkContainer, GtkBin, illetve a GtkBox absztrakt osztályok, így ebben a formájukban nem, csak a származtatott osztályok révén példányosíthatóak. Ezek közül ebben a részben a vízszintes, illetve függőleges elrendezésű boxokat (GtkHBox, GtkVBox), illetve a táblázatot (GtkTable) tárgyaljuk.

3.1.1. GtkVBox és GtkHBox

Függetlenül az iránytól a létrehozáskor két paramétert kell megadunk (homogeneous, spacing), ahol az egyik egyik egy bool, melynek true értéke esetén minden egyes elem azonos helyet foglal majd el a konténerben, míg a másik az egyes elem között üresen hagyandó részt adja meg pixelben.

3.1.2. Table

Létrehozáskor a táblázat sorainak, illetve oszlopainak kezdeti száma, valamint a korábbról ismert homogeneous érték adandó meg. Míg a vízszintes elrendezésű boxoknál a widgetek szélessége, a függőlegeseknél a magassága, addig a táblázatoknál mindkettő azonos ha a homogeneous paraméter true értékként adjuk meg.

3.2. Elem hozzáadása

Ezen függvények közös sajátossága, hogy paraméterként átveszik azt a widgetet, melyet a konténerbe kívánunk helyezni. A korábban említett fa hierarchiából következik, hogy egy elem nem lehet több szülőnek gyermeke (különben erdő szerkezetről beszélnénk), azaz egy widgetet összesen egy konténerben helyezhetünk el. Ha esetleg ezt másodszor is megpróbálnánk – még mielőtt a korábbi konténeréből eltávolítottuk volna – akkor futás idejű hibaüzenetet kapunk.

Ne feledjük, ahogy arról már említést tettünk a GTK rendelkezik referenciaszámlálási metódussal, azaz minden egyes objektum (GtkObject) – esetünkben GtkWidget – rendelkezik egy referenciaszámmal. Ha egy widgetet egy konténerbe helyezünk, annak referenciáját a konténer, annak rendje és módja szerint, növeli eggyel. Ez a referencia mindaddig megmarad, míg a widgetet el nem távolítjuk, vagy a konténer valamilyen oknál fogva meg nem szűnik, ami jellemzően akkor következik be ha az egész ablakot megszüntetjük (destroy).

3.2.1. Container

Az add nevű függvény egyetlen paramétert, az elhelyezni kívánt widgetet veszi át. Ritkán, leginkább csak – ebben a tekintetben – egyszerű konténereknél alkalmazott hívás, lévén olyan alapértelmezett paraméterek használ a widget elhelyezésére, melyek a felhasználó célnak a legtöbb esetben nem felelnek meg.

Használható ugyan a származtatott, bonyolultabb konténerek esetén is (pl: GtkBox, GtkTable), de célszerűbb az azokhoz tartozó, specifikus függvényt alkalmazni, lévén az sokkal rugalmasabban paraméterezhető.

3.2.2. Bin

Ebbe a típusba elemet csak a GtkContainer add függvényével tehetünk. Ha többször hívjuk meg a függvényt anélkül, hogy a GtkBin gyerekét eltávolítanánk futási hibát kapunk, hiszen a GtkBin csak egyetlen elem tárolására képes.

3.2.3. Box

Ahogy arról a bevezetőben szó volt a GtkBox típus olyan, mint egy két bemenetű verem. Ennek megfelelően két függvény van, amivel elemeket (egyet, vagy többet) lehet helyezni a konténerbe. A pack_start függőleges elrendezés (GtkVbox) esetén felülről lefelé haladva tölti meg a konténert úgy, hogy az egymást után elhelyezett elemek egymás alatt jelennek meg, míg a vízszintes elrendezést biztosító változat (GtkHBox) balról jobbra haladva teszi ugyanezt. A pack_end hívás ezekkel ellentétesen működik, tehát alulról, illetve jobbról balra haladva helyez elemeket a tárolóba.

Ahogy a GtkContainer esetén, itt is megadandó a widget, de ezen túl itt a konténeren belüli elhelyezkedést meghatározó értékek is paraméterek. Az expand és fill bool típusú paraméterek, melyek a korábban már említett ”felesleges” hely kitöltésére vonatkoznak. Előbbi azt határozza meg, hogy a widget a konténeren belül rendelkezésre álló helyet kitöltse-e, azaz ha van ilyen akkor az igényelje-e magának (true), vagy lemondjon róla (false) a többi – a konténerben lévő – widget javára. Utóbbi paraméter annak beállítására szolgál,

hogy a rendelkezésre álló – illetve az expand okán elnyert – hellyel mi történjék. Ha a paraméter értéke true, akkor a widget maga tölti ki ezt a helyet, azaz a feltétlenül szükségesnél nagyobb helyen rajzolódik ki, míg ha az érték false, akkor csak a minimálisan szükséges helyre rajzolódik és a maradék részt úgymond üresen hagyja. A pack függvények utolsó paramétere a padding, mely a widget körül (GtkVBox esetén felül és alul, GtkHBox esetén jobbról és balról) hagyandó üres hely értékét adja meg pixelben.
Ha elsőre nem is teljesen világos, mit jelent ez a gyakorlatban, a következő fejezet illusztrációjából minden világossá válik.

3.2.4. Table

Az attach függvény – mely a táblázatok esetén elem elhelyezésére szolgál – kissé összetettebb, mint a korábbiak, lévén egy táblázatnál vízszintesen és függőlegesen egyaránt szükséges megadni a fill, illetve expand paramétereket, melyek kiegészülnek egy shrink opcióval is, ez azonban már túlmutat ennek a résznek a keretein.

3.3. Elem eltávolítása

A származtatott típusok, legalábbis azok, amelyekkel ebben a részben foglalkozunk (GtkTable, GtkBin, GtkBox) nem igényelnek az eltávolítás során semmilyen extra műveletet, így a GtkContainer funkcionalitására támaszkodnak.

3.3.1. Container

A remove függvény értelemszerűen az eltávolítani kívánt widgetet várja paraméterként és ahogy azt említettük az általa tartott referenciát meg is szünteti. Ez jellemzően (a referenciaszámlálás GTK-beli sajátosságairól egy későbbi részben szólunk) azt is jeleni egyben, hogy az eltávolított widgetre az utolsó (hisz többnyire csak a konténere tart referenciát egy widgetre) referencia és ezzel maga a widget is megszűnik.
Ha ezt az esetet el akarjuk kerülni, akkor még az eltávolítás előtt a referenciaszám növeléséről magunknak kell gondoskodnunk. Vagyis ha két lépésben (remove és add) akarunk egy widgetet áthelyezni egyik konténerből a másikba, akkor az eltávolítás előtt növelnünk, a hozzáadás után pedig csökkentenünk kell a referenciát. Utóbbira azért van szükség, mert az új szülőelem maga is növel egyet a referencián, így ha mi a magunk által korábban megnövelt referenciát nem csökkentenénk a widget soha nem szűnne meg.
Hasonlóan a GtkBinhez itt sincs specifikus függvény az eltávolításra, hanem a GtkContainer remove függvényét hívjuk.

4. Pa(c)koljunk

Lássuk mire jó végül is ez a három opció (homogeneous, expand, fill) és mikét függenek össze egymással.
Azoknak, akik már jártasak valamilyen – a GTK+-tól eltérő – felületprogramozási nyelvben bizonyosan találkoztak már olyan eszközökkel (pl: QtDesigner, vagy jelen esetben a Glade), melyek egy konkrét felület elkészítésére alkalmasak. A koncepciók különbözőek ugyan, de mindegyiknél joggal merülhet fel a kérdés, hogy miért is nem lehet egész egyszerűen fix koordinátákkal megadni, hol legyenek az egyes widgetek. Nos lehet, sőt vannak is ilyen eszközök, ugyanakkor az elsőre talán kissé zavaró egyveleg komoly flexibilitást nyújt.

4.1. Homogenitás

Azaz egyenlőség, abban az értelemben, hogy minden egyes elem a konténerben pontosan ugyanakkora helyet foglal el. A következő ábra azt szemlélteti, hogy miként változtatja meg ez az érték – a másik kettő függvényében (expand, fill) – az elemek elhelyezkedését a konténeren belül.
Ahogy korábban a kódsorokat, most a widgetsorokat vesszük sorra a minél jobb megértés kedvéért.

4.1.1. Méretarányos elhelyezés

expand = false, fill = false
A konténerben lévő elemek – ahogy fentiekben fogalmaztunk – nem akarnak egymás rovására helyet szerezni (epxand), így a rendelkezésre álló vízszintes helyet nem is töltik ki, vagyis ezzel a megoldással az egész elemsorra nézve egyfajta balra (pack_end esetén jobbra) zártság alakítható ki.
expand = true, fill = false
Az összkép – az alatta található sor miatt – kissé csalóka, mivel az elemek kissé rendezetlennek tűnnek, ugyanakkor arról van szó, hogy minden egyes elem megszerezte magának – a saját eredeti méretigényének arányában – a rendelkezésre álló plusz helyet és az így allokált (size allocation) térrészen belül középen helyezkedik el.
expand = true, fill = true
Az egyes elemek nem csak hogy kiterjeszkedtek (expand) a korábban fel nem használt területre, de ki is töltik (fill) azt, azaz annak terjedelmében rajzoljék meg magukat.
Ahogy az az ábrából – és talán a magyarázatból is – kitűnik az fill opció állításának semmi teteje anélkül, hogy az expand be ne lenne kapcsolva, hisz e nélkül nincs semmilyen plusz terült, amire a widget magát megnagyobbítva rajzolhatná.

4.1.2. Homogén elhelyezés

expand = true, fill = false
A konténerben lévő elemek – a már használt kifejezéssel élve – nem akarnak egymás rovására helyet szerezni (epxand), így a rendelkezésre álló vízszintes helyet nem is töltik ki, vagyis ezzel a megoldással az egész elemsorra nézve egyfajta balra (pack_end esetén jobbra) zártság alakítható ki.
expand = true, fill = true

Az összkép – az alatta található sor miatt – kissé csalóka, mivel az elemek kissé rendezetlennek tűnnek, ugyanakkor arról van szó, hogy minden egyes elem megszerezte magának – a saját eredeti méretigényének arányában – a rendelkezésre álló plusz helyet és az így allokált (size allocation) térrészen belül középen helyezkedik el.

4.2. Térköz

Térköz megadására két lehetőség is kínálkozik a ha elemeket szeretnénk elhelyezni egy boxban. Az egyik, hogy magának a konténernek állítunk be létrehozáskor – vagy akár később – spacinget, vagy az egyes elemek hozzáadásakor adunk meg paddinget. Hogy mi a különbség a két eset között az alábbi ábra – ahol az fenti blokkban az első, míg az alsó blokkban a második esetre látunk példát – jól illusztrálja.

4.2.1. Tér az elemek között

expand = true, fill = false

Ez a példa nem illusztrálj igazán jól a helyzetet, ami pedig az, hogy ebben az esetben az elemek között jelenik meg az a térköz, amit a konténer létrehozásakor megadtunk.

expand = true, fill = true

Mivel itt mindkét érték true, a widgetek a rendelkezésre álló teret teljes egészében kihasználják maguk megrajzolására, eltekintve természetesen a közöttük megjelenő 10 pixel spacingtől. Érdemes külön megfigyelni a két szélső elemet, azoknak is az ablak széléhez közelebb eső részét a másik megoldással való összehasonlításhoz.

4.2.2. Tér az elemek körül

expand = true, fill = false

A padding megadásával a térköz nem az elemek között, hanem azok körül jelenik meg. Ez azt jelenti, hogy minden elem jobb és bal oldalán (GtkVBox esetén felül és alul) egyaránt jelentkezik a megadott térköz, ennek okán közöttük annak minimum (függően a fill értékétől) a kétszerese.

expand = true, fill = true

Ez az az eset amikor igazán jól látható a widgetek között és az azok mellett megjelenő térköz 2:1 aránya. Az előbb – a szélső widgetek elhelyezkedésénél megfigyelteket – most hasznosíthatjuk, ha észrevesszük itt a szélső widgetek nem tudnak a konténer széléig kiterjeszkedni, lévén két oldalról ki vannak párnázva (pad) 10-10 pixellel.

5. A kód

A fenti példaprogramok forrása, illetve azok eredetijei, a FLOSSzine, valamint a GTK+ oldalain az alábbi linkeken érhetőek el:

http://www.flosszine.org/sources/gtk_packbox.c
http://library.gnome.org/devel/gtk-tutorial/2.17/x387.html

5.1. Fordítás és linkelés

A korábbiakhoz hasonlóan az alábbi parancssorok segítségével fordíthatóak elemzett programjaink:

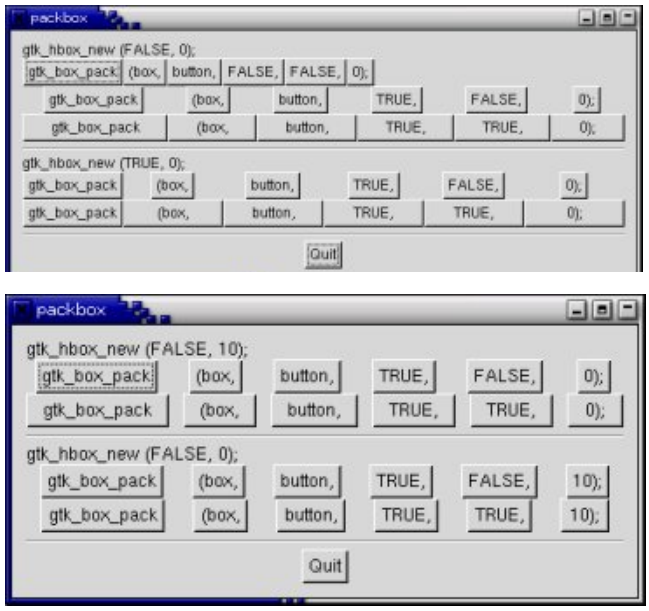
```
gcc gtk_packbox.c -o gtk_packbox `pkg-config --cflags --libs gtk+-2.0'
```

5.2. Futtatás

Próbáljuk ezúttal a ./gtk_packbox 1|2|3, illetve a ./gtkmm_-packbox 1|2|3 parancsokkal abban a könyvtárban, ahol a fordítást elkövettük, ahol a paraméter a teszt sorszáma, abban a sorrendben, ahogy azokat itt is ismertettük (a 3. természetesen ráadás).

5.3. Eredmény

Ha netán úgy érezzük mégsem világos mi is történik, mikor és miért a konténerekbe pakolás kapcsán ne adjuk fel. Elsőre talán az egész mechanizmus jelentősége sem szembetűnő, ugyanakkor érdemes próbálkozni, azaz venni a forrást és játszani a különböző értékekkel (fill, expand, spacing, padding), illetve a létrehozott ablak átméretezésével.



Hivatkozások

[1] Gtk+ / gnome application development
http://developer.gnome.org/doc/GGAD/ggad.html
[2] Gtk+ 2.0 tutorial
http://library.gnome.org/devel/gtk-tutorial/stable/
[3] Gtk tutorial - magyar fordítás.
http://gtk.pergamen.hu/
[4] Programming with gtkmm
http://www.gtkmm.org/docs/gtkmm-2.4/docs/tutorial/html/index.html

Pfeiffer Szilárd

Hello World! 6. rész.

Hobbielektronika

Avagy használjuk is végre valamire a friss tudományunkat!

A „tiszta” programozás területéről most tegyünk egy kis gyakorlatias kanyart a dolgos hétköznapok világába (természetesen vissza fogunk térni az eredeti vonalra is), nézzük meg, hogy mire is használható ez az egész amit eddig csináltunk.

Visszaemlékezve a régi szép DOS-os időkre, egy prototype kártyával ($300-as címtartomány) egy fél atomerőművet el tudott volna vezérelni egy lelkesebb bütykölős PC „hobbista”. Az ISA busz kihalásával ez a lehetőség megszűnt, azonban van még két ki-bemeneti port a gépünkön amit remekül felhasználhatunk fontos dolgok kezelésére, mint például komplett házvezérlő, figyelő- és riasztórendszer, vagy akár egy léptetőmotor meghajtásával a macskát is beengedhetjük vele egy automatizált macskaajtó segítségével.

Itt jegyezzük meg nyomatékosan, hogy a gép I/O portjainak kezelése hiba (pl. rossz címzés) esetén teljes rendszerösszeomláshoz, adatvesztéshez, végleges károsodáshoz vezethet. A külső, fizikai portok – mint például a printer port – áram alatti csatlakoztatása a gépet fizikailag tönkreteheti elsősorban a földelési problémák miatt. Ezek a csatlakozók (szemben például az USB-vel) nem úgy lettek kialakítva, hogy először a földelés a táppal majd az adatvezetékek csatlakoznak összedugáskor hanem a sorrend az egyenlő csatlakozóhosszak miatt véletlenszerű, az időkülönbség nem több mint pár mikroszekundum, ami viszont pontosan elegendő egy CMOS áramkör „megöléséhez”.

Az itt leírtakat mindenki csak a saját felelősségére, a fentiek megértésével és figyelembevételével próbálja ki.

A jogosultságok

Az I/O portok közvetlen elérésével „átnyúlunk” a kernel feje fölött, ami azért valljuk be őszintén illetlen dolog egy igazi operációs rendszer esetén, de a Linux felhasználóbarát és egy jó barát megenged még ilyesmit is, csak kellően kigyúrtak legyünk, azaz root jogosultság szükséges ezekhez a műveletekhez.

A program elején, még bármiféle I/O hozzáférés előtt az ioperm() függvényt kell meghívnunk. Deklarációja:

```
#include <sys/io.h>
int ioperm(unsigned long from, unsigned long num, int turn_on);
```

A from érték az engedélyezni kívánt port tartomány kezdete, például 0x2f8, a num a tartomány mérete, például 2 azaz 0x2f8 és 0x2f9 lesz az engedélyezett tartomány, a turn_on pedig logikai változó, ha értéke 1 akkor az engedélyt kiadjuk, ha 0 akkor megvonjuk. Az ioperm() függvényt többször is meghívhatjuk, ha több, nem összefüggő porttartományt akarunk engedélyezni.

A függvény visszatérési értéke sikeres végrehajtás esetén 0, hiba esetén -1.

Nagyon fontos korlátozás, hogy csak a 0x3ff-ig lehet az ioperm()-et használni.

Az ioperm()-et futtató programnak vagy root-ként kell futnia, vagy pedig a setuid-ot be kell rá állítani. Az ioperm() végrehajtása után már dobhatjuk a root jogosultságot, tovább nem lesz rá szükség illetve a program futása végén nem kell elvennünk, a process befejeztével megszűnik hatása.

A közvetlen port műveletek

A port műveletekhez szükséges függvények, makrók, rutinok a rendszernek és az architektúrának megfelelő könyvtáron belül a sys/io.h helyen találhatók. Régebbi kernelek (2.4 sorozat és az előtt) esetén ez a hely az asm/io.h. Ezek a rutinok általában inline makrók, használatukhoz az #include <sys/io.h> direktíva szükséges, más egyéb library linkelése nem. Jelen pillanatban a gcc-t bekapcsolt optimalizálás mellett kell futtatni (pl.: gcc -O1), ellenkező esetben hibaüzenettel áll le a fordítás, ugyanis a makrók nem kerülnek kifejtésre.

Egy portról byte beolvasását az inb(port) utasítással tehetjük meg, visszatérési értéke a portról beolvasott byte. Egy byte kiíratását az outb(value, port) hívásával érhetjük el, figyeljünk az argumentumok sorrendjére, a sorrend fordított mint amihez régen a DOS-nál szoktunk. Miután itt adatbusz fizikai kezeléséről van szó, egy ilyen I/O művelet körülbelül 1 mikroszekundum ideig tart.

Amennyiben arra van szükségünk, hogy a port hozzáférés után egy kicsi (1 mikroszekundumos) várakozás is legyen, úgy az inb_p(), outb_p() függvényeket (valójában makrókat) használjuk. Ha ez az idő is rövid, akkor az #include <sys/io.h> előtt használjuk a #define REALLY_SLOW_IO direktívát, ez 4 mikroszekundumos várakozási időt iktat be a

port hozzáférés után. Miután ezekben az esetekben a késleltetést a 0x80-as porthoz való hozzáféréssel érik el a makrók, ezért az ioperm()-el engedélyezni kell a 80-as portot. Ezt a módszert egyébiránt mi is alkalmazhatjuk, a 80-as porthoz való hozzáférés semmilyen változást nem okoz a rendszerben.

Az időzítések

Gyakran van szükség a hardveres feladatok során, hogy különféle időzítéseket, késleltetéseket állítsunk be, használjunk. A másodperces nagyságrendű időzítések esete a legkönnyebb:

```
#include <unistd.h>
unsigned int sleep(unsigned int seconds);
```

A függvénnyel másodperces felbontású várakozást iktathatunk be

Ha ennél rövidebb időkre van szükségünk, akkor a

```
#include <unistd.h>
int usleep(useconds_t useconds);
```

függvény lesz a mi barátunk. Argumentuma 0 és 1.000.000 közt lehet.

Használata közben hamar beleütközhetünk egy súlyos korlátba, milliszekundumos felbontás alá nem igazán fogunk tudni lemenni vele. Vajon miért? Azért, mert a kernel feje fölött ugyan átnyúlkálunk, azonban attól ő még ott van, és figyelembe véve a rendszer multitaskos voltát, bizony időszeletekben engedi csak futni a programunkat és ez által 10-20 milliszekundumos idők „kiesnek” a programunkból, ellentétben a DOS-nál megszokott helyzettel, amikor is egyedül uraltuk az egész gépet, cserébe szinte nulla funkcionalitást kaptunk az operációs rendszertől.

Úgyhogy zene lejátszást egy I/O port bitjének PWM-es modulációjával ne ezen a módon próbáljuk eszközölni.:-)

Egy rövid példaprogram:

```
/*
* portio.c: nagyon egyszerű példa a port I/O-hoz
*
* Csak demó a port íráshoz olvasáshoz
* Fordítása: `gcc -O2 -o portio portio.c',
* Futtasd rootként: `./portio'.
*/
#include <stdio.h>
#include <unistd.h>
#include <sys/io.h>
/*
2.4-es vagy az előtti kernelek esetén:
#include <asm/io.h>
*/
#define BASE_PORT 0x378 /* lp1 */
int main()
{
/* Port hozzáférés engedélyezése */
if (ioperm(BASE_PORT, 3, 1))
{
perror("ioperm");
return 1;
}

/* Az adatvonalak beállítása (D0-7) 0xA5 értékre */
outb(0xA5, BASE_PORT);
/* 100 ms várakozás */
usleep(100000);
/* Status olvasása (BASE+1) */
printf("Status: %X\n", inb(BASE_PORT + 1));
/* Kikapcsoljuk a port engedélyezést */
if (ioperm(BASE_PORT, 3, 0))
{
perror("ioperm");
return 1;
}
return 0;
}
```

A hibalehetőségek

* Segmentation fault hibaüzenettel áll le a program, amikor a porthoz hozzá akarunk férni

Megoldás: Nem root jogosultsággal fut a program illetve az ioperm()-mel nem engedélyeztük a porthozzáférést

* Undefined reference üzenettel áll le a fordítás, amikor a portműveletekhez ér.

Megoldás: Nem kapcsoltuk be a -O1 optimalizálási kapcsolót, vagy nem használjuk az #include <asm/io.h> illetve <sys/io.h> direktívát.

* outb() függvény meghívása nem csinál semmit, vagy rosszul csinálja, pedig jó portot és értéket adtunk meg.

Megoldás: Az argumentumok sorrendje rossz, helyesen előbb az érték és utána a port.

Összefoglaló

Ha valaki már otthonosan mozog a printer, vagy a joystick port megcsapolásában, már neki is állhat a PC vezérelt űrhajó építésének (jelzem 69-ben a Holdra szállást ezred akkora gép teljesítménnyel oldották meg, mint amekkora most bármelyikünk íróasztalán ott virít, sokszor csak egy kis szöveg szerkesztésére használva), aki nem, az a következő számban találhat egy kis csemegét ehhez.

Vomberg István

Nagios

Hálózatfelügyelet nyílt forrású programmal

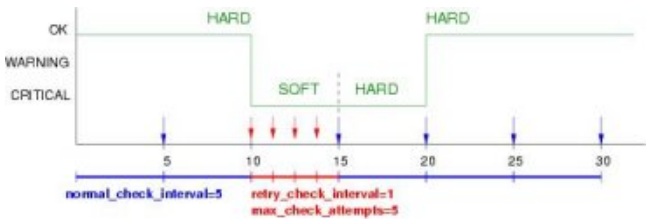
A Nagios egy közismert OpenSource monitorozó rendszer. Tudása talán kisebb, mint fizetős társaié, azonban testre szabhatósága miatt egyike a ma használatos legnépszerűbb felügyeleti rendszernek. Kis- és középvállalatoknak hatékony és költségkímélő megoldás. A Nagios maga egy C/C++ ban írt program, mely egy webes felületen keresztül szabályozható, figyelhető. Telepítése igen egyszerű, azonban konfigurálása sokáig eltarthat, függően az igényektől, és az alaposságtól.


Ethan Galstad

Az alapító Ethan Galstad, jelenleg 33 éves, és Saint Paul-ban, Minnesota(U.S.) államban lakik. Céljai között legkevésbé szerepelt, hogy az OpenSource közösség részévé váljon. Eredetileg űrrepülőket akart tervezni, és építeni, de pár szemeszter fizika, és számtan után úgy döntött, hogy az informatika tudománya mégiscsak közelebb áll hozzá. Miután lediplomázott 10 évig dolgozott rendszergazda, IT támogató munkatárs, web fejlesztő, és egyéb hasonló munkakörökben.

Maga a Nagios projekt, akkor még csak NetSaint, 1999-ben indult. Ethan egy barátjával a saját cég alapítás rögös útjaira tévedett. A cég kis- és középvállakózásoknak, valamint a kerület iskoláinak nyújtott volna hálózati felügyeletet. Hamar szembesültek a ténnyel, hogy nem engedhetik meg maguknak a kereskedelmi forgalomban kapható monitorozó rendszereket, és a nyílt forráskódú hasonló programok, akkoriban, nem voltak teljes értékűek. Ezért és egyéb ambíciók miatt, úgy döntött Ethan, hogy saját program írásába kezd. Két évre rá, 2001-ben kiderült, hogy egy másik cég már birtokolja a NetSaint nevet, így Ethan, hogy spóroljon mind pénzével, mind idejével, megváltoztatta programja nevét Nagios-ra, ami rekurzív anagramma, Nagios Ain't Gonna Insist on Sainthood (Nagios nem fog ragaszkodni a szentségéhez).

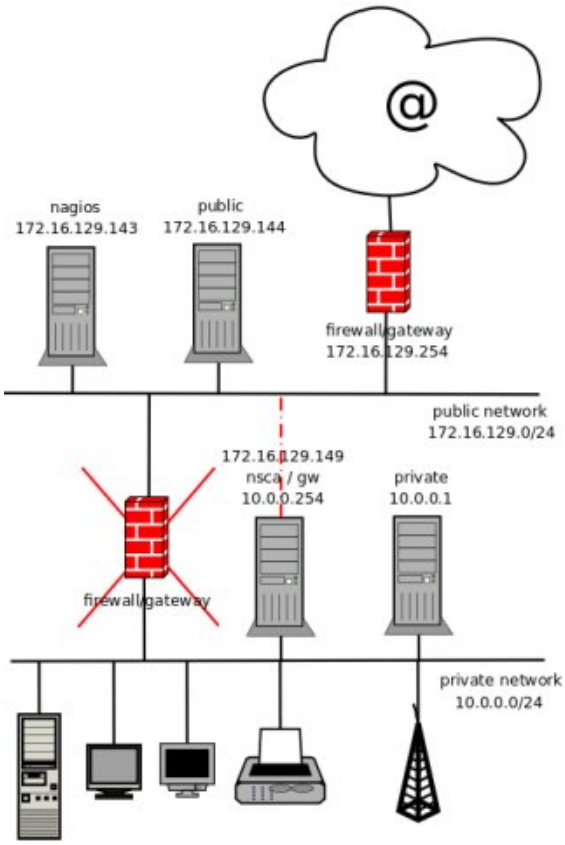
A teljes történethez hozzátartozik, hogy Ethan nem, tervezte a munkáját GPL alatt kiadni, félt hogy ezzel egy részről erősíti a konkurenciát, valamint, hogy sokan fogják kritizálni programozási stílusát , képességét. Azonban mikor megismerkedett a Linux-al, a GPL programokkal (kifejezetten nagy segítség volt számára a GNU/GPL C fordító) és az OpenSource közösséggel, mégis úgy döntött, hogy hozzáférhetővé teszi a kódot GPL licenc alatt. Hamar rájött, hogy nem csupán 20-30 személy vagy cég fogja használni a Nagios-t, és gyorsan kezelhetetlen méretűvé nőtte ki magát a projekt. Mára már számos személy és közösség segíti munkáját, és azóta sok változáson ment keresztül a kód, de saját bevallása szerint a központi rész, eltekintve némi tisztogatástól, közel olyan mint 9 évvel ezelőtt.



A rendszer működése alapesetben nem túl bonyolult. A hálózaton elérhető gép nyitott portját a megfelelő kliensprogrammal, egyszerű hívások alapján ellenőrzi. A beérkezett eredményt logolja. Ha a check_command 0-ás visszatérési értékkel rendelkezik OK állapotnak tudja be és egy ideig nem ellenőrzi újra. Ha a visszaadott érték 1, 2, vagy 3 akkor SOFT állapotba kerül, majd ha továbbra sem kap 0-t akkor a visszatérési kódnak megfelelően WARN, CRIT, vagy UNKNOWN státuszba kerül a szolgáltatás. Ennek megfelelően előre definiált akciót hajt végre, értesít, közbeavatkozik, vagy csak logolja. Természetesen sok egyéb változó szólhat még bele működésébe, de ezeket inkább megpróbálom egy példával szemléltetni.

Hogy a rendszer néhány közismert képességét be tudjam mutatni (NRPE, NSCA) minimum 4 szolgáltató egységet kell megalkotnom. Egy olyan kisvállalati hálózatot képzeljünk el, ahol egy publikus (ez a mi esetünkben 172.16.129.0/24 -es tartomány, ami ugyan nem publikus, de a példánk idejére az lesz) és egy privát (10.0.0.0/24-es

tartomány) hálózat van. A cég publikus szolgáltatásait futtató gépek a public és az ugyancsak kívülről is elérhető nagios gépek. Természetesen ez nem azt jelenti feltétlenül, hogy ezek a gépek kint ülnek a veszélyekkel teli interneten, csupán azt, hogy rendelkeznek külső IP-vel, vagy tűzfalon keresztül szolgáltatásaik elérhetőek „bárhonnan”. Tovább egyszerűsítve a dolgot a belső tűzfalat egy nsca nevű géppel helyettesítette. A belső tartományban helyezkedik el egy private nevű szerver, ami feltehetően a cég belső működéséhez szükséges szolgáltatásokat adja a munkaállomásoknak, és egyéb hálózati eszközöknek.



nagios:
OS-ünk debian, így telepíthetjük a debian repository-ból, ami az alábbi fordításnak felel meg.
https://buildd.debian.org/fetch.cgi?pkg=nagios3;ver=3.0.6-5;arch=i386;stamp=1246312187

```
nagios:~# apt-get install nagios3 apache2
[...]
nagios:~#htpasswd bc /etc/nagios3/htpasswd.users nagiosadmin admin
```

(az apache2 nagios konfig: /etc/apache2/conf.d/nagios3.conf ->/etc/nagios3/apache2.conf)

Ekkor ha belépünk a http://nagios/nagios3 oldalra máris láthatjuk, hogy bőszen ellenőrzi saját magát. Természetesen szép dolog ha SSL-es autentikációt használunk, más porton figyeltetjük, stb., de ezzel most nem foglalkozunk.

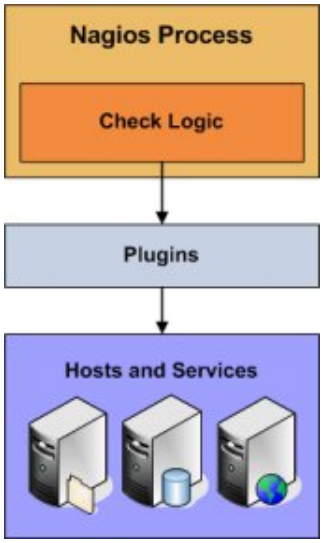
A telepítés egyszerűsége után, kiderül, hogy a végtelenségig lehet bonyolítani az amúgy sem egyszerű konfig fájlati, melyek sok egymásba source-olt fájlba van eltéve. Íme egy kis segítség. (lásd konfigurációs fa képe)

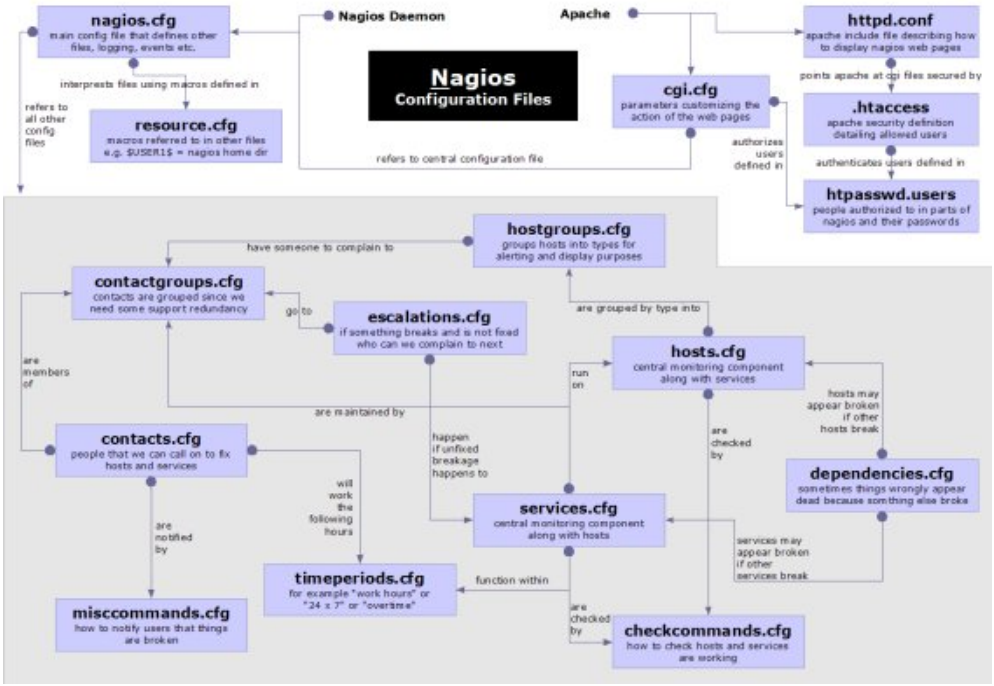
A nagios ezeken az utakon halad végig minden egyes alkalommal. Lássunk is neki a konfigurációnak, hogy a public nevű gépet ellenőrizze egyenlőre active_check metódussal. Az aktív ellenőrzés azt jelenti, hogy a nagios szerver indítja az ellenőrzést, és ő maga hajtja vére a kiértékelést is, vagyis a nagios gépen meghívott bináris, vagy szkript fogja eldönteni, hogy visszatérési értéke mi legyen.

Vegyünk fel még 2 publikus gépet, valamint nevezzük át a szegény localhost-ot nagios-nak. Majd indítsuk újra a nagios-t. A hostgroup-ok sokat tudnak könnyíteni munkánkon, mivel így nem kell felvenni egyenként a szervizeket. Az új gépet mindössze a megfelelő csoportba kell betenni. Természetesen egynél több csoportnak is tagja l

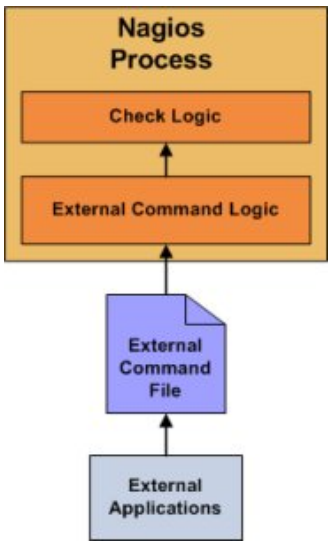
```
define host {
  host_name   public
  alias       public
  address     172.16.129.144
  use         generichost
}
define host {
  host_name   nsca
  alias       public
  address     172.16.129.149
  use         generichost
}

[...]
define hostgroup {
     hostgroup_name  debianservers
alias                Debian GNU/Linux
Servers
members         nagios,public,nsca
}
define hostgroup {
hostgroup_name  httpservers
alias           HTTP servers
members         nagios,nsca
}
define hostgroup {
hostgroup_name  sshservers
alias           SSH servers
members         nagios,public,nsca
    }
define hostgroup {
    hostgroup_name  pingservers
alias           Pingable servers
members         gateway,nagios,nsca
    }
[...]
/etc/nagios3/conf.d/hostgroups.cfg
```

active_checks





passive_checks



ehet egy host.

Most hogy már ismerősek lettünk az aktív monitoring területén, itt az ideje, hogy kombináljuk passzív monitoring-gal. Ezt két módon fogjuk megtenni. Az egyik az NRPE (Nagios Remote plugin Executor) démon, aki a kliens gépeken hallgat az 5666-os porton, valamint a NSCA (Nagios Service Check Acceptor) démonnal, aki pedig a szerver szerepet ellátó gépen fog hallgatni, és várni, hogy a kliens elküldje neki a mögötte lévő gép(ek) adatait.

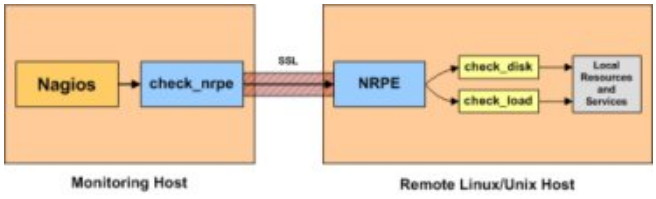
Először a public gépre telepítjük az nagios-nrpe-server csomagot. A nagios gépre pedig az nagios-nrpe-plugin csomagot, ha még nincs fent. A kliensen, ahol démonként indítjuk, az /etc/nagios/nrpe.cfg fájlban tudjuk állítani, hogy milyen szervertől fogadjon el kéréseket, valamint hogyan értelmezze a bejövő kéréseket. Konfiguráljuk fel úgy, hogy a 172.16.129.143-tól elfogadjon kérést (check_load, check_users, check_procs parancsokat). Hallgasson a saját publikus IP-jén az 5666-os porton. Most az argumentum átadásával nem foglalkozunk.

Majd a nagios gépet megkérjük, hogy ellenőrizze az alábbi módon a public gépet (akárhova fel lehet venni, de mi most a host-public.cfg -hez vesszük fel).

```
[...]
define service {
        service_description     load
        use                     genericservice
        host_name               public
        check_command           check_nrpe_1arg!check_load
}
```

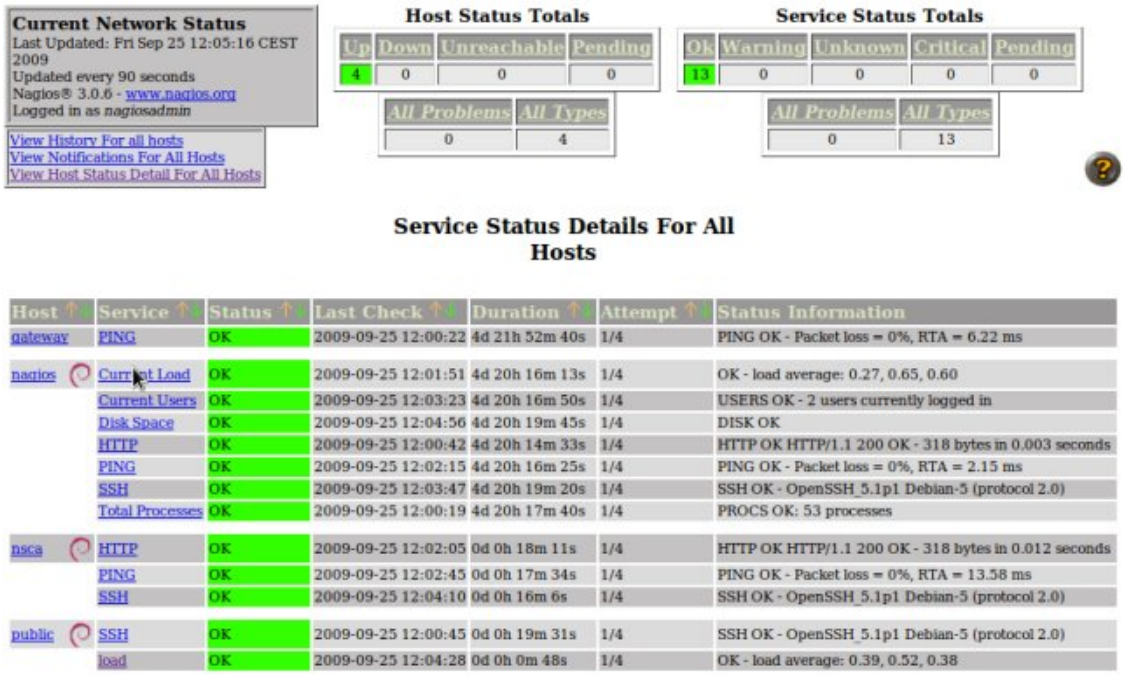
A check_command definícióval már nem kell törődni mivel azt automatikusan megírta a telepítő nekünk a /etc/nagios-plugins/config/check_nrpe.cfg fájlba. Ebben a könyvtárban több .cfg fájl is található, melyeket a nagios felolvas induláskor. Érdemes ezt a logikát követni saját szkriptek írásakor. Ha minden jól ment az alábbi képnek kell fogadnia.

```
public:~#egrep'(^command\[check_|^allowed_hosts|^server_port)'/etc/nagios/nrpe.cfg
server_port=5666
allowed_hosts=127.0.0.1
command[check_users]=/usr/lib/nagios/plugins/check_users w 5 c 10
command[check_load]=/usr/lib/nagios/plugins/check_load w 15,10,5 c 30,25,20
command[check_total_procs]=/usr/lib/nagios/plugins/check_procs w 150 c 200
public:~# netstat na | grep 5666
tcp        0      0 172.16.129.144:5666     0.0.0.0:*               LISTEN
public:~# ls la /usr/lib/nagios/plugins/check_* | egrep '(users|load|procs)'
rwxrxrx 1 root root 18260 20090201 03:15 /usr/lib/nagios/plugins/check_load
rwxrxrx 1 root root 30904 20090201 03:15 /usr/lib/nagios/plugins/check_procs
rwxrxrx 1 root root 17508 20090201 03:15 /usr/lib/nagios/plugins/check_users
```



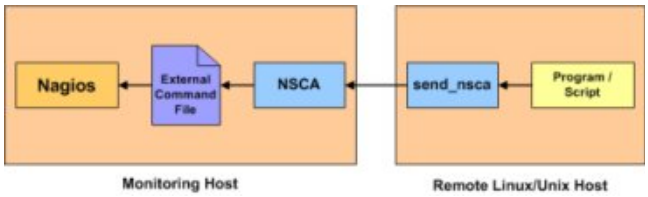
könnyen lehet utasítani (csupán egy templét fájlt kell megváltoztatni), hogy aktívan ellenőrizzen tovább. Ez azért lehetséges, mivel az NSCA esetében mindkét gépen szerepelnie kell az ellenőrzött gépek konfigjai.

Hogy elérjük a kívánt állapotot, az nsca gépen is hajtsuk



http://www.nagios.org/

Ha ezzel készen vagyunk, akkor itt az ideje, hogy belevágjunk az NSCA telepítésébe és konfigurálásába. Véleményem szerint ennek a megértése a nehezebb.



Röviden arról van szó, hogy nem a figyelő gép kezdeményezi az ellenőrzést, hanem csupán csak a beérkező eredményeket, és azok „frissességét” tartja számon. A kliens gép végzi el időnként az ellenőrzéseket, és tolja át a központi gépnek. Az NSCA-t nem elérhető belső hálózatok ellenőrzésére használják, ahogyan mi is a példában. A rendszergazda csinál egy nagios-t a belső hálózatra, majd utasítja azt, hogy küldje ki a központi gépbe a belső hálózat aktuális csekkolási eredményeit. Egy másik dolog, amire még lehet használni, hogy redundáns ellenőrzést alakítsunk ki. Ha mindkét nagios gép eléri a hostokat, mégis felét egyik, felét másik ellenőrzi aktívan (a többit passzívan kapja meg), akkor egy részről terhelésmegosztást értünk el, másrészt egy gép elhalása esetén a megmaradt gépet végre értelemszerűen a nagios gépen végrehajtott lépéseket, hogy végül ezt a képet lássuk. Kivettem a gateway-t, mivel az megegyezett a nagios gép gateway-vel (172.16.129.254), saját magát (nsca) localhost-ra írt csekk szkriptekkel ellenőrzi, a private gépet pedig a belső hálózatba lógatott lábán keresztül.

Tehát mindkét gépre installáljuk az nsca csomagot. Ez a csomag tartalmazza mind a kliens (send_nsca bináris) és a

```
[...]
password=F1o55zine
[...]
command_file=/var/lib/nagios3/rw/nagios.cmd
[...]
[...]
define service{
    use             passive_service
    service_description   TestMessage
    host_name       nsca
    }
define command{
    command_name check_dummy
   command_line $USER1$/check_dummy $ARG1$
    }
[...]
[...]
password=F1o55zine
[...]
```

szerver (nsca bináris) oldali fájlokat. Hogy az nsca gépen ne induljon el a démon vegyük ki az /etc/rcX.d könyvtárakból a szimlinkeket (update-rc.d nsca remove).

Szerver oldalon (nagios):
1)editáljuk meg az /etc/nsca.cfg-t.
2)hozzunk létre egy check_dummy parancsot.
3)vegyünk fel egy passzív szervíz templétet.
4)vegyük fel az nsca hosthoz (host-nsca.cfg) egy TestMessage szolgáltatást a passzív templéttel.

Kliens oldalon (nsca):
1)editáljuk meg az /etc/send_nsca.cfg-t.
2)hozzunk létre egy test text filet.
nsca <tab> TestMessage <tab> 2 <tab> This is a test.
3)majd küldjük el és ellenőrizzük, hogy a túloldalon megérkezett-e.

Remek! Akkot most tegyük automatikussá a host/service

```
nsca:~# /usr/sbin/send_nsca 172.16.129.143 c /etc/send_nsca.cfg < test
1 data packet(s) sent to host successfully.
nsca:~#
```

információk elküldését az nsca gép számára. Ezt úgy

TestMessage	CRITICAL	2009-09-25 13:31:03	0d 0h 20m 8s	1/1	This is a test

tehetjük meg ha a kliens gépen beálltjuk, hogy minden csekk lefutása után futtasson le még egy plusz parancsot is. Erre szolgál a nagios.cfg -ben az ocsp_command option. Az alábbi módosításokat kell végrehajtanunk az nsca gép nagios konfigurációján.
Hozzunk létre egy submit_check_result.cfg-t!
Valamint írjuk is meg a submit_check_result.cfg-t! Ezt

```
[...]
obsess_over_services=1
ocsp_command=submit_check_result
[...]
```

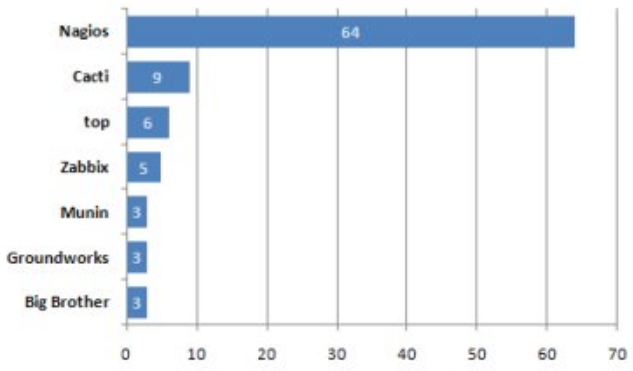
megtaláljuk a

```
define command{
command_name submit_check_result
command_line /usr/lib/nagios3/submit_check_result   $HOSTNAME$
'$SERVICEDESC$' $SERVICESTATE$ '$OUTPUT$'
}
```

http://nagios.sourceforge.net/docs/1_0/distributed.html oldalon, de lényegében egy bash szkript, ami a nagios által sorrendben (HOSTNAME, SERVICEDESC, SERVICESTATE, OUTPUT) adott adatokat elküldi a send_nsca bináris és a send_nsca.cfg konfig használatával. Ahhoz hogy ennek az elküldésnek bármi hasznát is lássuk természetesen fel kell venni a központi gépre a belső hálózaton ellenőrzött gépeket is, csak így értelmezheti a nagios a beérkező ellenőrzési adatokat helyesen.

Természetesen még sok mindenről lehetne beszélni a nagios-szal kapcsolatban. El lehetne elmélkedni biztonság kérdéseken, a performanciát lehetne tuningolni, PNP segítségével diagramokat lehetne rajzoltatni vele, riasztásokat lehetne lekezeltetni, és még sok egyéb finomság képbe jöhet, a kombinációk száma szinte végtelen. Pont ezért olyan népszerű a nagios a rendszergazdák körében. Javaslom hogy, akit jobban érdekel a kérdés húzzon fel pár virtuális gépet, és járja körül a témát, mivel ebben a cikkben ,az idő rövidsége miatt, nem eshetett mindenről szó.

Csíkos Bálint



Hivatkozások:

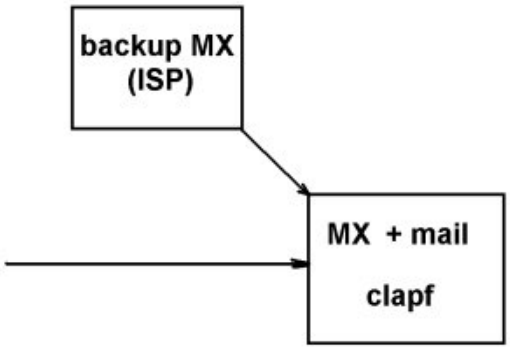
http://archive.fosdem.org/2005/index/interviews/interviews_galstad.html
http://community.nagios.org/2009/09/08/top-5-best-system-monitoring-tools/
http://www.nagios.org/about/propaganda/awards/
http://www.tu-chemnitz.de/urz/kurse/unterlagen/nagios.html
http://nagios.sourceforge.net/docs/1_0/distributed.html
http://i.zdnet.com/blogs/ethan-galstad.jpeg
http://www.tu-chemnitz.de/urz/kurse/unterlagen/rsrc/topcorp.jpg
http://ws.edu.isoc.org/workshops/2008/apricot2008/netmanage/presos/nagios/nagios-config.png
http://www.linuxvilag.hu/content/files/cikk/71/cikk_71_57_61.pdf
http://nagios.org

spamszűrés

Clapf

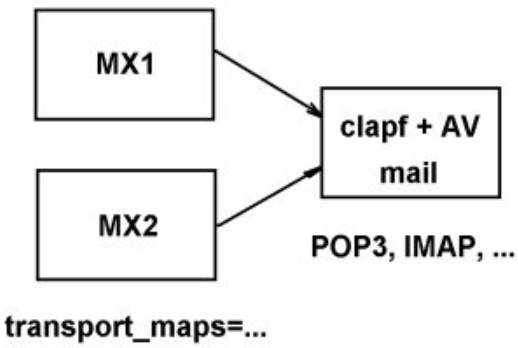
Hulladékfeldolgozás nyílt forrású programmal - 3. rész

Már több, mint fél év eltelt az első Clapf cikk óta. Fél év nagy idő egy szoftver életében, így elhatároztam, ahelyett, hogy újrakezdeném a Clapf bemutatását, inkább befejezem ezt a sorozatot. Ebben a részben gyakorlati példákat, konfigurációkat mutatok be, néhány teljesítményhangolási tippel kiegészítve.



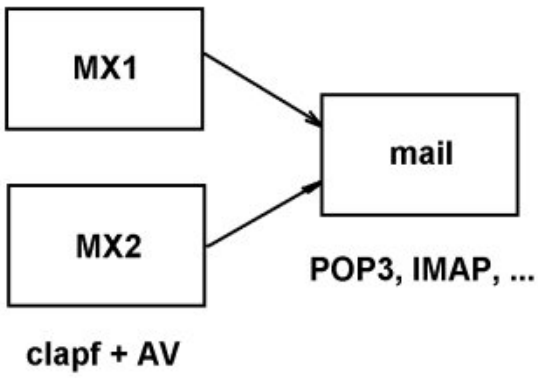
1. ábra: A legegyszerűbb konfiguráció

A legegyszerűbb felállásban egy kkv-nak egyetlen levelezőszervere van. A backup MX-et a szolgáltató nyújtja, amelyről a levelek a cég gépére érkeznek. Mivel a Clapf sok kereskedelmi megoldásnál hatékonyabb, ezért a spamszűrést a cég gépén oldjuk meg.



2. ábra: Egy kkv teljes levelezését is képes elvinni

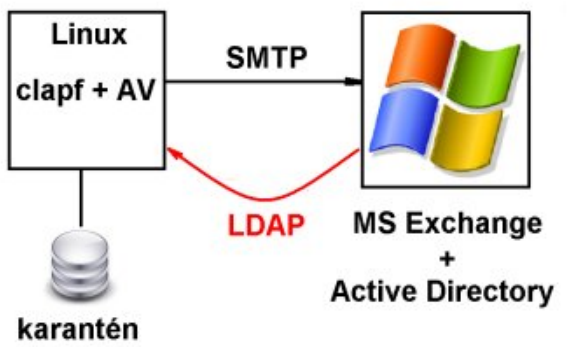
Ha komolyabb levelezést folytat a cég, akkor érdemes 2 MX szervert használni. A 2. ábrán látható konfigurációban az MX1 és MX2 nevű gépek csak fogadják a leveleket, majd továbbítják a mögöttük lévő mail szerverre, ahol a Clapf is fut. Ebben a felállásban az MX-ek (relatíve) olcsóak lehetnek, mert az erőforrás igényes tartalomszűrést nem azok végzik. Ha pedig a belső mail szerver leállna, akkor az



3. ábra: Nagyobb forgalmú kkv sem lehet akadály

MX-ek átmenetileg tárolják a leveleket.

Nagyobb levélforgalomnál érdemes a tartalomszűrést az MX-ekre tenni, és a szűrés I/O intenzív terhelését megosztani közöttük. Ebben az esetben azonban már nagyobb teljesítményű gépeket kell használnunk erre a célra, viszont a

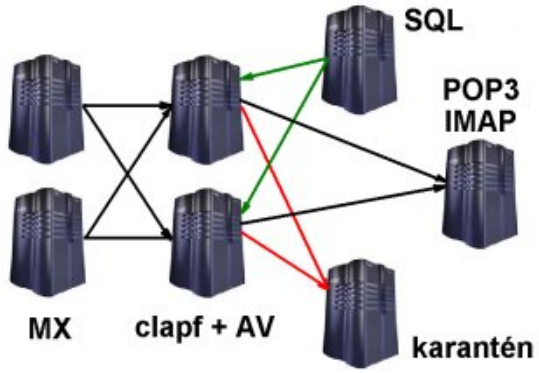


mailszerver terhelése csökken.

4. ábra: A Microsoft környezet sem állhatja útját

Gyakori eset, hogy egy kis, vagy közepes cégnél adott a Microsoft platform, jellemzően az Exchange és Active Directory duóval. Spamet szűrni persze ezeken is lehet, azonban ez nem olcsó mulatság, mivel a nyílt forrású termékek

itt ritkaságszámba mennek. A megoldás azonban egyszerű: telepítsük a Clapf spamszűrőt egy Linuxot futtató gépre (ez akár egy virtuális gép is lehet), majd irányítsuk rá a bejövő leveleket. A Clapf menedzsment felületén mindössze néhány kattintással importálhatjuk LDAP protokollal az érvényes email címeket. Emellett a Linuxos gép a karantén szerepét is elláthatja, így az Exchange-re már csak a tiszta levelek érkeznek meg. Ez a konfiguráció nagy mértékben csökkenti az Exchange szerver terhelését, illetve leveszi róla a nem létező címzettek kezelésének nyűgjét. A felhasználók számára a működés teljesen transzparens, észre sem veszik, hacsak nem



azt, hogy eltűnt a spam.

5. ábra: Clapf egy nagyobb, elosztott környezetben

Nagyobb környezetben, mondjuk egy szolgáltatónál, nem egy, vagy két géppel oldják meg a levelezést, hiszen gondolni kell például a redundanciára, illetve az egyes funkciók, szolgáltatások megfelelő szétválasztására. Az 5. ábrán látható példában 2 MX fogadja a leveleket, majd továbbítják 2 tartalomszűrőnek, amelyeken a Clapf és egy támogatott antivírus szoftver (pl.: Clamav) fut. Ezek a jó leveleket azonnal továbbítják a mailszerverre, ahonnan a felhasználók POP3, IMAP4, webmail, vagy más módon tölthetik le. Végül egy dedikált gépen gyűlnek a spamek - ha úgy állítjuk be - mert a Clapf képes a spameket más SMTP szerverre továbbítani, mint a jó leveleket. Ez természetesen akár felhasználónként is állítható a házirendszabályok (policy group) segítségével. Ha Béla Bácsi úgy akarja, akkor neki csak jelöljük a spameket, és mehet a postafiókjába.

A Clapf preferáltan SQL-ben tárolja az e-mail címeket, doméneket, a felhasználónkénti beállításokat és a tokeneket is. Egy adott méret felett érdemes az SQL szervert is egy külön gépre tenni.

A különféle szabályok segítségével a Clapf viselkedését nagy mértékben testre lehet szabni. Ezt úgy érjük el, hogy Clapf alapértelmezett konfigurációját (= ez a default policy group) felüldefiniáljuk a policy groupban beállított értékekkel. Megadhatjuk például, hogy használjon-e egyáltalán spamszűrést, hol húzzuk meg a spam határt, használjon-e fekete listákat (blacklist), jelölje-e meg a levél tárgy sorát, használjon-e karantént, stb. Nagyobb, például szolgáltatói környezetben ez mindenképpen hasznos lehet.

Természetesen nem kőbevésett szabály, hogy ekkora méret esetén hét gépet kell használni. Az 5. ábra sokkal inkább úgy kezelendő, mint egy szakácskönyv: egy kicsit ebből, ízlés szerint abból, ahogyan az adott környezet, illetve a lehetősé-

gek megkívánják, megengedik.

Menedzsment

6. ábra: A Clapf spamszűrőhöz távirányító is tartozik

A WebUI (=web user interface) a Clapf egy opcionális kiegészítője, ahol egy böngészőből, néhány kattintással lehet elvégezni az adminisztratív műveleteket (pl.: felhasználók, domének, email címek és házirend csoportok kezelése). A felhasználók bejelentkezés után megtekinthetik a karanténjukat (mindenki szigorúan a sajátját, kivéve ha adminisztrátor), módosíthatják a fehér listájukat (white list), elengedhetnek leveleket a karanténból, sőt egy röptében generált képen a ham/spam arányt is nyomon követhetik.

Az adminisztrátorokat érdekelheti az egyes levelek sorsa. A feladat nem nehéz, mindössze 'greppelni' kell néhányat a naplóban. A WebUI azonban egy AJAX-os táblázat segítségével megmutatja egy levél útvonalát a rendszerben. Bizonyos mezőkre húzva az egeret egy ballonban részletes információk jelennek meg (pl.: queue azonosítók, message-id-k, stb.).

Teljesítmény

Ha valaki nagyobb méretű levelezés felett őrködik, bizonyára beleütközött néhányszor a spamassassin teljesítményigényébe. Bár a Clapf túlszárnyalja az SA-t teljesítményben, némi hangolással még többet ki lehet hozni belőle.

Vegyük le 1-re a naplózás szintjét, hacsak nem akarunk valamit debuggolni. Az ideiglenes állományokat tehetjük ramdiszkre, így a Clapf gyorsabban képes beolvasni a levelet.

Kapcsoljuk ki az RBL vagy URIBL lekérdezéseket, mivel a DNS válaszok ideje számottevő lehet, és így visszafogja a Clapf teljesítményét. Egyébként itt szeretném megjegyezni, hogy ha mindenképpen akarunk RBL listákat használni, akkor az egyes gépek közös DNS cache-t használjanak. A legnagyobb teljesítménynövekedést azonban az okos SQL hangolással érhetjük el. Tegyünk elég memóriát a MySQL-t futtató gépbe és tuningoljuk okosan a különféle bufferek értékét. Tegyük az SQL adatokat külön diszkre, vagy külön gépre. Használjunk MySQL replikákat a Clapfot futtató gépeken. A spamszűrőt ugyanis úgy is be lehet állítani (l. lentebb), hogy a replika adatbázist csak olvasás módban használja, amit ebben az esetben viszont durván tuningolni lehet. Megfelelően beállítva az SQL szervert, gyakorlatilag minden kérést memóriából képes kiszolgálni. Végül kapcsoljuk ki a tokenek frissítését vagy irányítsuk azokat memcached démonba, ahonnan késleltetett, illetve kötegelt (batch) feldolgozással lehet a tokenek időbélyegét frissíteni. Fontos, hogy ha kikapcsoljuk ezt a funkciót, akkor ne futtassuk a törlő scripteket (util/purge*).

Tekintsük ismét a 3. ábrát, ahol 2 gépen futtatunk Clapf spamszűrőt. Az előző számban azt írtam, hogy a leveleket (amelyeket a Clapf queue könyvtáraiba mentünk) tehetjük egy közös NFS kiszolgálóra (amely nem szerepel a 3. ábrán). A lehetőség továbbra is adott, viszont NFS-en dolgozni időbe telik, ezért nagyobb teljesítményt érhetünk el, ha a --with-store=fs opciót használjuk továbbra is, és egy cron feladatot használva az rsync paranccsal tesszük át a leveleket arra a gépre (10.1.1.2 a példában), amely a tanítást végzi:

```
*/5 * * * * rsync --chmod=o+rwx --remove-source-files -rza -e "ssh -i /var/lib/clapf/ssh.key" /var/lib/clapf/queue/* 10.1.1.2:/var/lib/clapf/queue
```

Végül néhány számadat a Clapf teljesítményéről. Egy kommersz Core2 Duo-s desktop PC-n 55 levelet képes feldolgozni egy másodperc alatt. Egy Windowsos PC-n futtatott VMWare alatt (512 MB memóriát rendelve a VM-hez) ~54k levél volt az áteresztő képessége óránként, ami nem tűnik soknak (azonban ne felejtsük el, hogy virtuális gépről van szó, ráadásul Windowson). A Freemail 2007-ben napi 18 millió levelet dolgozott fel több, mint 60 darab HP DL 145-ös géppel (legalábbis a http://kepek.origo.hu/galleriesdisplay/upload//0710/Freem2007101215722/img/o08.jpg alapján ilyennek tűntek), amiből 18 csak a spamszűréssel foglalkozott. A 18 millió levél tartalomszűrését akár 14 db Windows-os desktop PC is elvitte volna, ha VMWare alatt Clapfot futtat.

7. ábra: Ennyire pontos

Szörnyen pontos

Befejezésként egy kis kedvcsináló a végére: szeptemberben 1506 spamet kaptam, és csak 1 csúszott át, így a spam felismerés aránya 99.93%. Jó dolog sok spamet elkapni, de még jobb, ha a spamszűrő minél kevesebb jó levélbe harap bele. A 2913 jó levél egyikét sem azonosította spamként, így a fals pozitív hibaarány 0.00% volt, ami 99.97%-os összesített pontosságot jelent. A te spamszűrőd képes erre?

Sütő János

Tudtad-e?

Tudtad-e, hogy a diplodocs.com oldalon megtalálhatod és letöltheted a legkülönfélébb eszközök felhasználói kézikönyveit? Ezek lehetnek háztartási gépek, autók, motorok, vagy akár különböző számítógépek is. Sajnos még sok a hiányzó kézikönyv, de ezen könnyen segíthet bárki, aki akar, hiszen lehetőség van a kézikönyvek feltöltésére. Jelenleg 5600 márka 1870000 felhasználói kézikönyve található meg az oldalon. A név és a logó alapján nem túl nehéz kitalálni, hogy az oldal névadója a diplodocus carnegiei nevű dinoszaurusz volt. Ez a dínó a késő jura korszakban élt Észak-Amerika nyugati részén, és az egyik példány 54 méteres hosszával valószínűleg nem a leghosszabb, de a leghosszabb, teljes csontváz alapján ismert dinoszaurusz. Remélhetőleg egyszer a diplodocs oldal is hasonlóan gigantikus lesz a weblapok között.

Asterisk

VoIP Linux alatt

A telefonos kisasszony már a múlté

Az előző lapszámban szó esett a VoIP kliensekről. Azóta már talán mindenki meg is találta, hogy neki melyik a legszimpatikusabb. Ebben a cikkben kicsit mélyebbre ásunk: megnézzük, hogyan lehet nekünk otthon olyan SIP kompatibilis telefonközpontunk, mint a nagyoknak. Csak első látásra tűnik ördöngösségnek.

Miért Asterisk?

A cikkben az Asterisket szeretném bemutatni. Hogy miért? Talán ehhez van a legtöbb elérhető dokumentáció, fórum, stb. az interneten.

Számos olyan projekt van amely az Asteriskre épül és teljes körű megoldást ad. Az egyik ilyen például a Trixbox, amely CentOS alapú és a webes felületről egész kényelmesen konfigurálható. Ez kezdőknek kellemes lenne, de egy-egy ilyen rendszerhez már-már asztali gép szükséges, amelynek nem elhanyagolható a fogyasztása. Egy ilyen megoldás min. 100 wattal számolva óránként havonta mintegy 3 ezer forinttal dobná meg a villanyszámlát. (Persze csak ha 100 wattból tényleg elég.)

Az egyik korábbi számban bemutattam az otthoni mindenesemet, egy Linksys NSLU2-t, de gyakorlatilag bármilyen vékonykliens (vagy akár Mikrotik Routerboard) használható, amely rendelkezik min. 32, de inkább 64-128 megabájt memóriával és USB csatolóval, valamint olyan processzorral, amelyet a Linux támogat. Esetemben például a Linksys NSLU2 fogyasztása alig 10 watt és nem is melegszik. A benne lévő 266 MHz-es processzor és a 32 megabájt memória mégis elég arra, hogy akár 4 egyidejű hívást fogadhasson a rendszer.

Miért forrás? Miért nem csomag?

Töltsük le a program forrását. Hogy miért a forrást és miért nem jó az adott disztribúció csomagja? Elvileg jó az is, tehát kevésbé gyakorlott felhasználó választhatja azt is. Azonban le kell szögezni, hogy minden újabb verzió számos hibajavítást és újdonságot hoz. A Debian stabil ágában például csak az 1.4.21.2 szerepel, míg a projekt honlapjáról a cikk írásakor már elérhető 1.4.28, illetve az 1.6.1.12 is.

Miért 1.4.x és miért nem 1.6.x?

Röviden: az 1.4.x eléggé kiforrott és elég sok dokumentáció található hozzá a neten. Az O'Reilly is kiadott hozzá egy teljes értékű ingyenesen letölthető elektronikus könyvet. Az 1.6.x nem hozott még annyi újdonságot, hogy megérné váltani, de ha szívesen bütyköl az ember, akkor lehet azt is tanulmányozni.

Fordítás

Aki csomagból telepített, az átugorhatja ezt a részt.

Nézzük, mi is kell az Asterisk forrásának lefordításához a már megszokott gcc, g++, make trión túl:

- openssl és a dev csomagok
- ncurses és a dev csomag (a konzol miatt)
- zlib

Ezeken túl kellhet még a curl és a newt csomag, de ezek hiánya se vészes, pláne otthoni felhasználó esetén. Amire szükségünk lehet még: e-mail küldési lehetőség (sendmail vagy postfix) a hangpostához. Ha esetleg nem fájlba, hanem adatbázisba szeretnénk rögzíteni a hívásrekordokat, akkor még szükség lehet pár csomagra, de ennyire már nem szeretném elbonyolítani.

A fordítás elég egyszerű, asztali gépen csak pár perc, de a korábban említett Linksys NSLU2-n akár több óra is lehet. Összesen négy parancsunk lesz, amiket a kitömörített Asterisk könyvtárában kell kiadnunk root-ként:

```
./configure
make menuselect
make
make install
```

Ezekhez nem fűznék túl sokat, talán csak annyit, hogy a ./configure jelzi, ha valamelyik csomag még hiányzik, illetve a make menuselect lehetővé teszi, hogy finom hangoljuk, mi az amire szükségünk van és mi az amire nincs. (Tájékoztat a függőségekről is.)

Ha a make install is szépen lefutott, akkor gyakorlatilag kezdhetjük is.

SIP és IAX2 eszközök...

Az Asterisk számos protokollt kezel, de a két – Asteriskes központ által leggyakrabban használt – protokoll a SIP és az IAX2. Ebből is a legtöbb hardveres és szoftveres telefon, illetve központ a SIP-et támogatja. Az IAX2-nek talán ott van inkább létjogosultsága, ahol több párhuzamos hívás fut, mert a SIP ilyenkor kicsit nagyobb sávszélességet igényel ugyanannyi, ugyanolyan paraméterű beszélgetéskor. Az

IAX2 mellett szól még talán az is, hogy ha egy nagy cég NAT-olt hálózatot használ, akkor talán könnyebb dolga van a rendszergazdának. (Részletesebben cikk vége fele.)
A SIP-et csak üzenetváltó protokoll, míg az IAX2 multiplexelve küldi a jelzés- és a médiacsomagokat.
Hogy mégis miért használnak inkább SIP-et a szolgáltatók? Talán valamivel kevesebb biztonsági probléma van vele és egy kicsit rugalmasabban bővíthető.

Kodekek...

Számos kodek használható Asteriskkel. Ezek között vannak ingyenesek és licencköteles, vannak kisebb és nagyobb sávszélesség-igényűek, processzor szempontjából bonyolultak és kevésbé bonyolultak. A teljesség igénye nélkül néhány kodek (leggyakrabban használtak/legkönnyebben elérhetőek):
G.711a (alaw) 64 kbit/sec
G.711u (ulaw) 64 kbit/sec
G.729a 8 kbit/sec (licencköteles)
GSM 13 kbit/sec (azért GSM kodek, mert a GSM hálózatok is ezt használják)
A sávszélesség-igényt le- és feltöltés irányba is számolnunk kell, illetve a másodpercenkénti 30-100 adatcsomag IP fejléccel (egyidejű hívásonként) együtt. A processzorigényt azért célszerű szem előtt tartani, mert ha mindkét végpont ugyanazt a kodeket tudja használni, akkor gyakorlatilag nulla gépidőbe kerül a hívás, míg az átkódolás a kodek bonyolultságától függ. Tehát G.711u-G.711u hívásokból többet lekezel a központ, mint G.711u-GSM hívásokból.
Ahhoz, hogy két végpont tudjon beszélgetni egymással, fontos, hogy legyen két olyan kodek, ami között az Asterisk tud közvetíteni. Tehát ha például az egyik eszköz csak GSM kodeket támogat, a másik pedig csak G.711u-t, akkor fontos, hogy az Asterisk mindkettőt ismerje.

Mellékek, kontextusok...

A mellékek gyakorlatilag azok a logikai részek amik egy-egy funkciót megvalósítanak, vagy egy-egy végponti eszköz felé kapcsolatot teremtenek (pl. tárcsázás). A mellékeket általában a leírásuk sorrendjében futtatja az Asterisk, hacsak nem számozott prioritással dolgozunk. További érdekessége még a dolognak, hogy úgynevezett kontextusokat hozhatunk létre. Gyakorlatilag az eszközök esetén is azt adjuk meg, hogy melyik kontextus legyen a belépő (kezdő) kontextus. Egy mellék egy sorának a leírása általában így néz ki:
exten => 1234,1,Dial(SIP/1234,30,r)
Ez például arra utasítja az Asterisket, hogy az 1234 mellék tárcsázásakor az 1234-es SIP eszközt csörgesse 30 másodpercig úgy, hogy a hívó fél csengetés hangot halljon. Az 1-es esetünkben azt mutatja, hogy ehhez a mellékhez ez az első parancs. (A parancsok teljes listája megtalálható a linkek között megadott voip-info oldalon.)
Természetesen nem lenne teljes az élet helyettesítő karakterek nélkül. Ilyenek például:
X – 0-9 számok
Z – 1-9 számok
N – 2-9 számok
[1237-9] – 1,2,3,7,8,9 számok
. - egy vagy több tetszőleges karakter
! - nulla vagy több tetszőleges karakter
s – mindenre illeszkedik (start)
Fontos, hogy ha két minta is illeszkedik egy számra, akkor az Asterisk mindig a lehető legjobban illeszkedőt fogja futtatni, tehát például:
exten => _123X,1,NoOp()
exten => _1234,1,NoOp()
Esetünkben, ha az 1234-et tárcsázzuk, akkor a második fog lefutni, mert az illeszkedik a legjobban.

Nézzük a gyakorlatban...

Ennyi bevezetés után talán vágjunk is bele. A minimális konfiguráció:
asterisk.conf a fő konfiguráció:

```
[global]
astetcdir => /etc/asterisk
astmoddir => /usr/lib/asterisk/modules
astvarlibdir => /var/lib/asterisk
astagidir => /usr/share/asterisk/agi-bin
astspooldir => /var/spool/asterisk
astrundir => /var/run/asterisk
astlogdir => /var/log/asterisk
cdr.conf a hívásrekordok miatt
[general]
cdr_custom.conf hívásrekordok fájlba írása
[mappings]
Master.csv =>
"${CDR(clid)}","${CDR(src)}","${CDR(dst)}","${CDR(dcontext)}","${CDR(channel)}","${CDR(dstchannel)}","${CDR(lastapp)}","${CDR(lastdata)}","${CDR(start)}","${CDR(answer)}","${CDR(end)}","${CDR(duration)}","${CDR(billsec)}","${CDR(disposition)}","${CDR(amaflags)}","${CDR(accountcode)}","${CDR(uniqueid)}","{CDR(userfield)}"
extensions.conf mellékek, az Asterisk egyik alapköve
[general]
static=yes
writeprotect=yes
[default]
[bejovo]
exten => s,1,Dial(SIP/2000,60,)
exten => s,n,Hangup(16)
[kimeno]
```

```
exten => _061XXXXXXX.,1,Dial(SIP/${EX-
TEN}@szolg1,60,)
exten => _0621ZXXXXXX.,1,Dial(SIP/${EX-
TEN}@szolg1,60,)
exten => _06[237]0NXXXXXX.,1,Dial(SIP/${EX-
TEN}@szolg1,60,)
exten => _06NXZXXXXX.,1,Dial(SIP/${EX-
TEN}@szolg1,60,)
exten => 2222,1,Playback(tt-weasels)
exten => 2222,n,Hangup(16)
exten => 2000,1,Dial(SIP/2000,60,)
exten => 2000,n,Hangup(16)

modules.conf modulok kezelése
[modules]
autoload=yes
noload => pbx_gtkconsole.so
noload => pbx_kdeconsole.so
noload => app_intercom.so
noload => chan_modem.so
noload => chan_modem_aopen.so
noload => chan_modem_bestdata.so
noload => chan_modem_i4l.so
noload => chan_capi.so
load => res_musiconhold.so
noload => chan_alsa.so
[global]



rtp.conf média beállítások
[general]
rtpstart=10000  ; az RTP stream kezdő portja
rtpend=10020   ; az RTP stream utolsó portja
                ; (pontosabban jelen esetben a 10021 lesz)
sip.conf SIP beállítások, az Asterisk másik alapköve
[general]
context = default ; ez az alapértelmezett kontextus
port = 5060  ; ezen a porton fog fülelni az asterisk
bindaddr = 0.0.0.0 ; a rendszer összes hálózati csatolóján
                ; (ha ez nem kell, adjunk meg IP-t)
disallow=all  ; az összes kodek tiltása
allow=alaw   ; G711a engedélyezve
allow=ulaw   ; G711u engedélyezve
realm=mysip  ;
domain=mysip  ; a kliensben ezt kell megadni domainnek
registertimeout=50  ;
registerattempts=0  ;
register => 06211234567:titok@sip.szolgaltato.hu
                ; kimenő szolgáltató
[szolg1]
type=peer    ; peer, ha kimenő, user, ha bejövő, friend, ha
mindkettő
username=júzer
secret=titok
host=sip.szolgaltato.hu
canreinvite=no  ; az asterisk a médiaútban marad
                ; NAT-nál ajánlott
insecure=port,invite
qualify=yes
nat=yes    ; NAT-olt végpontnál;
                ; egyébként nem szokott problémát okozni
context=bejovo  ; melyik kontextusba menjen az innen jövő
hívás
disallow=all
allow=ulaw
[2000]
type=friend
username=2000
secret=phone
host=dynamic
context=kimeno
canreinvite=no
qualify=yes
nat=yes
disallow=all
allow=ulaw
callerid=<06211234567>
```

Ha minden jól csináltunk és a VoIP kliensünkre is felkonfiguráltuk a 2000-es melléket, akkor az Asterisk elindítása után tudnunk kell tárcsázni a 2222-t, melyre a klasszikus „Weasels have eaten our phone system” mondatot kell halljuk (magyarul: a menyétek megették a telefonrendszerünket), illetve a VoIP szolgáltatónkon kifele is tudunk telefonálni a 06-al. Az extensions.conf-ban érdemes tanulmányozni a mintákat.

Ugyanígy a bejövő hívás is a 2000-es melléken fog csörögni.

De ha mégse csörög...

Ha mégse csörögne a bejövő hívás, akkor az esetek nagy részében a routerünk nem továbbítja a sip.conf general részében megadott port forgalmát az Asterisk-et futtató gép felé. Hasonló a probléma, ha csak kifele van hang, de befele nincs. Viszont abban az esetben az RTP port tartomány továbbítását kell ellenőriznünk.

Hibakeresés a konzolon...

Az Asterisk konzoljára az asterisk -r paranccsal jutunk. Itt nyomon követhető a központunk működése. Ha valamiért nem túl bőbeszédű, akkor célszerű kiadni a core set verbose 10 parancsot. Ha esetleg módosítunk a konfiguráción, akkor a reload paranccsal olvastathatjuk újra az Asteriskkel anélkül, hogy a már folyamatban lévő hívások megszakadnának. Egyébként a Bash-hoz hasonlóan a TAB gomb lenyomására

elénk tárja az elérhető parancsokat, sőt, még minimális súgóval is segít.

Merre tovább?

A kedves Olvasóra bízom, hogy hogyan bővíti tovább a konfigurációs állományokat. Lehet például mellékekkel bővítgetni, vagy akár be is lehet vele mondatni a pontos időt. Az se megoldhatatlan, hogy este 10 és reggel 6 között ne engedjen be hívást. Vagy kérjen kódot a kimenő hívásokhoz. Ébresztőórának is kitűnő.

A lehetőségek gyakorlatilag korlátlanok. Ez a cikk csupán a jéghegy csúcsára világít rá. Csak az otthoni rendszerem bemutatása meghaladná az újság terjedelmét. Jó kísérletezgetést!

Linkek (angol):

http://www.asterisk.org/downloads
http://www.voip-info.org/
http://en.wikipedia.org/wiki/Session_Initiation_Protocol
http://en.wikipedia.org/wiki/IAX2

Medve Zoltán

Tudtad-e?

Programlelőhelyek

Vannak olyan gyűjtőoldalak, amiket már szinte mindenki ismer,
Ez semmit sem von le nagyszerűségükből, de érdemes néha más gyűjtőprojektre is ránézni az interneten.
A két legismertebb a SourceForge és a Freshmeat. Ezek minden open source keresgélés alapjául szolgálnak, mint a legnagyobb, leggazdagabb gyűjtemények, a szabad szoftverek világában.
A SourceForge jelenleg több, mint 230000 szoftver projektnek ad otthont (ezek között a komplett alkalmazások éppúgy megtalálhatóak, mint a különböző kiegészítők vagy segédprogramok), és regisztrált felhasználóinak száma meghaladja a kétmilliót. A sima letöltéshez nem szükséges regisztrálni.
http://sourceforge.net/
A freshmeat szintén a Sourceforge Inc. (http://web.sourceforge.com/) égisze alatt működik akárcsak a sourceforge.net, de itt a legfrissebb verziókra és frissítésekre helyezik a hangsúlyt.
Mivel szóba került a Sourceforge Inc., talán érdemes megemlíteni, hogy (jelenleg) ők üzemeltetik többek között a Slashdot (ez egy témába vágó híroldal, hírekkel és beszámolókkal, nem csak szoftverekre vonatkozóan) (http://slashdot.org/), a ThinkGeek (Mondhatjuk geekbolt-nak? Persze több annál, beszámolók, tesztek, stb. bőven megtalálhatóak itt, de vásárolni is tudunk). http://www.thinkgeek.com/
Egyébként mindkét előbb említett oldal lényegét sokkal frappánsabban visszaadja, az az angol nyelvű szlogen, ami az oldalnév közelében található, Slashdot – „news for nerds, stuff that matters”, ThinkGeek - „stuff for smart masses”. Ha valaki ezekre tud valami frappáns fordítást, ne kíméljen bennünket, adja közre a fórumban!,
A Fossfor (ez szintén egy gyűjtőoldal, csak itt a Windows felhasználók találhatnak maguknak szabad szoftvereket), és az Ohloh oldalakat is.

http://fossfor.us/
http://www.ohloh.net/

Ez a legutóbbi egyébként nem egy másik szabad szoftveres gyűjtőoldal (persze találhatunk érdekes projekteket itt is), hanem sokkal inkább egy olyan hely, ahol kapcsolatba kerülhetnek egymással a szabad szoftverek fejlesztői és használói.

Hálózat felügyelet

Hálózatbiztonság nyílt forrású eszközökkel III.

Remélhetőleg már mindenki profi a földtani és a hálózati rétegekben, de legalábbis az utóbbiakban.
Ha nem, akkor elég csak annyit elképzelni, hogy az adatcsomag eredeti, az adatokat tartalmazó részére minden réteg először ráteszi (hozzáteszi) a maga kis információit a küldő gépnél, majd a fogadó gépnél ezek lebontása után megkapjuk az eredeti üzenetnek megfelelő információkat.

Ezt el lehet képzelni, mint a hagyma héját (rétegeket), vagy a sarkkutatóknak ajándékba küldött meztelen celebet, aki szépen, rétegesen felöltözködik, mielőtt az egyik jégkunyhóból átmegy a másikba.

Most pedig nézzük a legfontosabb események és protokollok közül néhányat, amiket érdemes lehet ismerni.

Számítógépünk öntudatra ébred (vagy sem), és arra is rádöbben, hogy nincs egyedül az univerzumban. Megpróbálja felhívni magára a figyelmet, és küld egy ARP csomagot a hálózatra (ha szükséges).
Az ARP (Address Resolution Protocol), említésre került már, és majd még fogunk róla beszélni (gondoljunk csak az érettségi vizsgára) :) .
No de viccet félretéve, az ARP felépítése (a működésről már volt szó az előző részben):
A csomag tartalmazza a hardver típusát, a 01 az ethernet.
A protokoll típusát (a hálózati rétegé), itt IP, azaz 0800.
A hardvercím hosszát bájtban, Ethernet MAC cím esetén 6.
A protokoll cím hosszát, ennek tartalma esetünkben 4 (azaz 4 bájt, az IPV4 miatt).
A műveleti kódot, ennek értékei az ARP esetén a következők lehetnek: 1=ARP kérés, 2=ARP válasz, 3=RARP kérés, 4=RARP válasz.
A küldő hardver címét (itt MAC).
A küldő protokoll szerinti címet (itt IP).
A cél hardver címét (itt MAC).
A cél protokoll címet (itt IP).
Broadcast üzenet esetén a megfelelő helyeken “mindenki” címe található, és az válaszol akire “ráillik az ing”. (Vagy Dr. Genya, de erről sokkal később lesz szó).

Mikor kerül elküldésre az ARP csomag? Ha szükséges.

Bővebben: az ARP csomagokkal nem terhelik a gépek feleslegesen a hálózatot, mivel rendelkezésükre áll az úgynevezett ARP cache. Ezt a táblázatot a hálózati forgalmat figyelve is folyamatosan frissítik, és csak akkor küldenek broadcast üzenetet, ha nincs a táblázatban a szükséges információ.
Az ARP-re az IP kommunikáció megkezdése előtt van szükség, így becézhetjük az IP segédprotokolljának is.

A protokoll fordítottja a RARP (Reverse Address Resolution Protocoll), ez egy adott hardver címhez tartozó IP cím megállapítására szolgál.

Az UDP-ről is volt szó korábban, most csak a felépítését nézzük meg.
Az UDP felépítése:
Forrás port száma
Cél port száma
Hossz: fejrésszel együtt
Ellenőrzőösszeg
Alkalmazási adat: az üzenet
Az UDP hasznosságáról már volt szó, de érdemes megemlíteni még azt is, hogy az alkalmazási rétegben tevékenykedő DNS protokoll megbízottja a szállítási rétegben az UDP, vagyis a DNS is az UDP-t használja.

Említettük, hogy az Internet alapja a TCP/IP protokollpáros.
A TCP és az IP protokollokból tevődik össze. Tulajdonképpen két külön protokollról van szó. Kicsit egyszerűsítve a dolgokat az IP az internet rétegben (OSI-ra vetítve a hálózati rétegben), a TCP pedig a szállítási rétegben fejti ki áldásos tevékenységét.
Felépítésük:

IP (IPv4)

Verzió: az IP különböző verziói eltérő datagram-formátumokat használnak, itt van megadva melyikről van szó.

Fejrészhossz: általában 20 bájt, de lehet ettől eltérő, így itt van megadva. Ebből derül ki, hol kezdődnek az adatok.

Szolgáltatástípus (type of service, TOS).

Datagramhossz: az IP datagram teljes hossza (fejléc+adatok) bájtban. Mivel a mező 16 bites, az IP datagram mérete nem lehet nagyobb 65535 bájtnál. A gyakorlatban ritkán nagyobbak 1500 bájtnál.

Azonosítószám, Jelzőbitek, Darabolási eltolás: mindhárom az IP daraboláshoz kapcsolódik (csak az IPv4-ben van ilyen).

Életben maradás ideje (time to live, TTL): Néhány amerikai szerint az Internet egy nagy cső, Európában már bonyolultabb a helyzet, de inkább ne vitatkozzunk ezzel a megállapítással, tehát, hogy a cső ne duguljon el túl hamar, érdemes a csomagoknak (datagram) életben maradási időt (számot) meghatározni. A szám értéke minden egyes útválasztón való áthaladáskor eggyel csökken. Ha a mező értéke 0, az útválasztónak el kell dobnia a csomagot.

Protokoll: ha célba ért a datagram, az itt szereplő azonosító alapján kerül átadásra a szállítási rétegben a megfelelő protokollnak, például a TCP-nek, ha ez az érték 6-os, vagy az UDP-nek, ha 17.

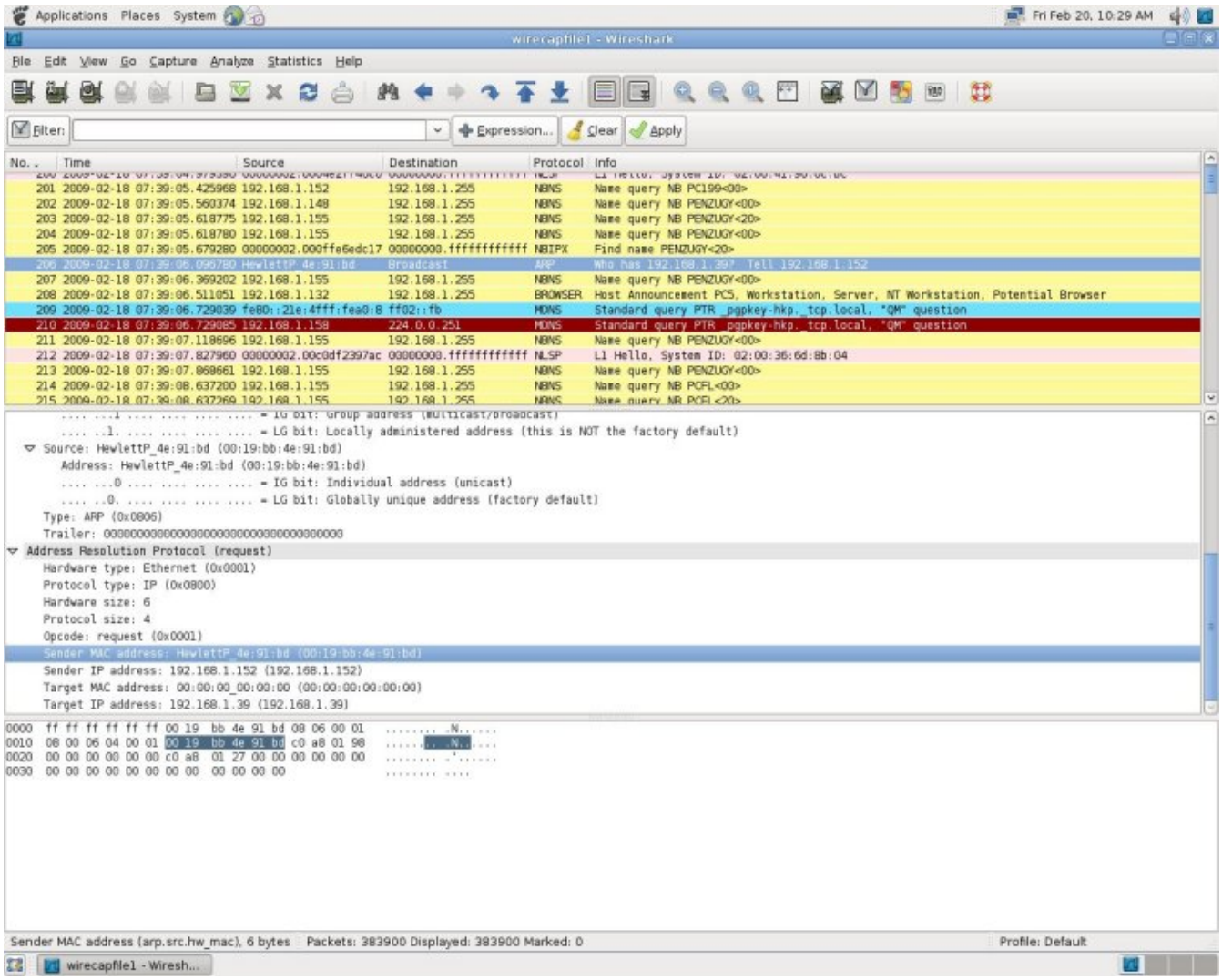
Fejrész ellenőrző összege: ez segít az útválasztónak abban, hogy kiderüljön, ha a fogadott IP csomag bithibát tartalmaz.

Forrás IP cím

Cél IP cím

Opciók.: az opciómezők lehetővé teszik az IP fejrész bővítését, ezzel jelentős bonyodalmakat okozva. Ezért (persze más okok miatt is) az IPv6-ban már nem találkozunk ilyen megoldással.

Adatok: itt az lenne a lényeg, legtöbbször a célba juttatandó szállítási rétegbeli szegmenst tartalmazza (TCP vagy UDP), de tartalmazhat más adatokat is.



1. kép: wire_arp

A különböző rétegbeli protokollok eltérő méretű kereteket tudnak szállítani. Ezért akár az IP datagramok feldarabolására is szükség lehet, sőt.

Lehet olyan keretünk, amely maximum 576-bájtból állhat, ugyanakkor az Ethernet keretek 1500 bájt adatot is tartalmazhatnak.

Itt jön a képbe az MTU. Az a legnagyobb adatmennyiség, amit az adatkapcsolati rétegbeli keretünk szállítani képes, az átvihető leghosszabb adategység névre hallgat, Maximum Transmission Unit, MTU.

Mivel az IP datagram az adatkapcsolati rétegbeli keretbe csomagolva utazgat az útválasztók között, az adatkapcsolati réteg protokolljának MTU-ja korlátozza a méretét, sőt az út teljes hosszán eltérő MTU-jú protokollok ladikjain szeli át az óceánt, így a darabolást többnyire nem kerülheti el.

A darabolás rejtelmeibe most ne menjünk bele, már az is több, mint elég, ha tudunk róla.

(Később még elővesszük).

Az IPv6 majd megszünteti ezt a darabolást, ezáltal egyszerűbbé teszi a csomagok feldolgozását és csökkenti az IP sebezhetőségét.

TCP

Rétegugrás következik (mint azt már megszokhattuk).

A TCP a szállítási rétegben működik. Erről is volt már szó.

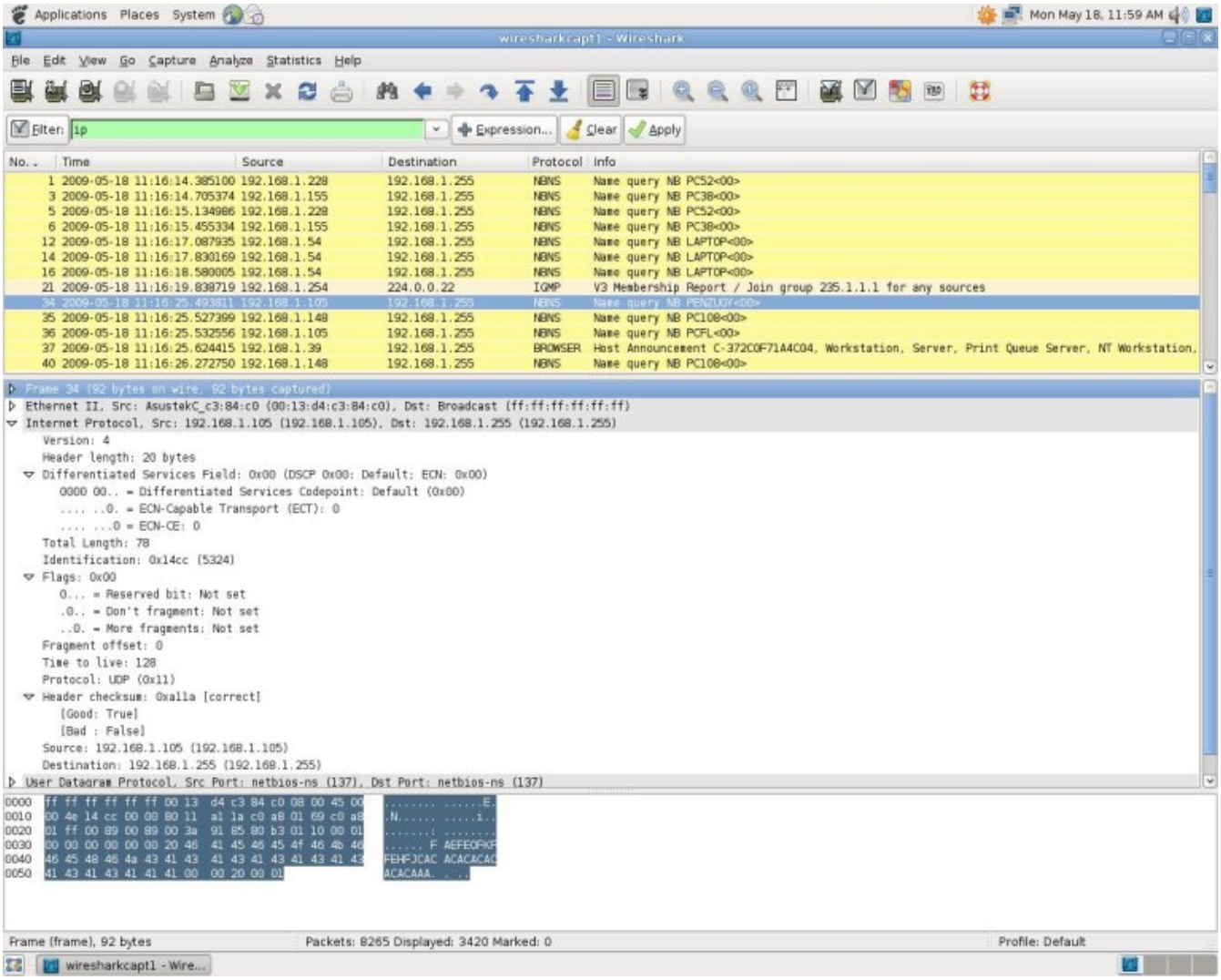
A TCP-ről is csak röviden beszélhetünk a cikksorozat keretein belül, hiszen a megbízható adatátvitelnek csak az elvi taglalása megtölthetne egy kisebb könyvet, a TCP pedig egy ilyen protokoll.

A TCP-t összeköttetés alapúnak szokták nevezni, mert az adatküldés elkezdése előtt az információcserében részt vevő két fél “kezet fog egymással”.

Az adatáramlás megkezdése előtt létrehozzák az adatátvitel paramétereit, ezt hívják háromutas kézfogásnak.

A korábbi heterogén hálózati rendszerek összeköttetése érdekében (szinte minden hálózatnak saját protokollja volt) szükség volt egy hálózatközi protokollra, ami két szimpatikus úriember munkájának köszönhetően meg is valósult. Vinton Cerf és Robert Kahn a becsületes nevük. Kezdetben egy egységnek tekintették a protokollt (Transmission Control Protocol/Internet Protocol), de később kettéválasztották TCP-re és IP-re.

A számítástechnika Nobel díjának tartott ACM vándordíjat 2004-ben kapták meg, többek között a TCP/IP



2. kép: IP

megtervezéséért és megvalósításáért.

A TCP felépítése:

Forrásport száma

Célport száma

Mindkettő a felsőbb rétegbeli alkalmazásokkal való kapcsolattartás céljából van megadva (akár az UDP esetében). Ne feledjük, a szállítási rétegben vagyunk :) .

Sorszám

Nyugtaszám

Ezekre a megbízható adatátviteli szolgáltatás megvalósításához van szükség.

TCP fejrészhossz: tipikusan 20 bájt hosszú, ekkor az opciók mező üres, de az opciók mezőnek köszönhetően változó hosszúságú lehet.

Opciók: ez valóban opcionális és változó hosszúságú, használatára például a szegmensméretről való egyeztetés esetén lehet szükség (adó és vevő között). Több opció van.

Jelzőmező: hat bitet tartalmaz. Az ACK bit jelzi, ha egy nyugtamezőben lévő érték érvényes (a szegmens egy nyugtát tartalmaz). Az RST, SYN, és FIN biteket az összeköttetés létrehozásához és lebontásához használják (az összeköttetés lebontása is háromutas). A PSH bit beállítása esetén a vevőnek az adatot azonnal továbbítania kell a felsőbb rétegnek.

Az URG bit jelzi, ha a szegmensben olyan adat is van, amit a küldő oldali felsőbb réteg “alkalmazása” sürgősként jelölt meg.

Ezzel kapcsolatos a sürgősségi adat mutató is.

Ez utóbbi hármat a gyakorlatban nemigen használják, de a protokoll tartalmazza.

Vételi ablak: “nem hivatalosan” arra használják, hogy a küldőnek fogalma lehessen arról, hogy a vevőnél még mennyi szabad pufferterület érhető el, ennek a forgalomszabályozás (torlódáskezelés) területén is jelentősége van.

Ellenőrzőösszeg

Adatok

Röviden és leegyszerűsítve ennyi a TCP, de ez csak a jéghegy csúcsa.

Sajnos terjedelmi korlátok miatt ebben a részben nem tudjuk végigvenni az összes protokollt, így legközelebb innen folytatjuk.

Szőke József

```
▷ Internet Protocol, Src: 194.109.137.218 (194.109.137.218), Dst: 192.168.1.158 (192.168.1.158)
▽ Transmission Control Protocol, Src Port: http (80), Dst Port: 46660 (46660), Seq: 0, Ack: 1, Len: 0
    Source port: http (80)
    Destination port: 46660 (46660)
    Sequence number: 0    (relative sequence number)
    Acknowledgement number: 1    (relative ack number)
    Header length: 40 bytes
  ▽ Flags: 0x12 (SYN, ACK)
      0... .... = Congestion Window Reduced (CWR): Not set
      .0.. .... = ECN-Echo: Not set
      ..0. .... = Urgent: Not set
      ...1 .... = Acknowledgment: Set
      .... 0... = Push: Not set
      .... .0.. = Reset: Not set
      .... ..1. = Syn: Set
      .... ...0 = Fin: Not set
    Window size: 5792
  ▽ Checksum: 0xf3f3 [correct]
      [Good Checksum: True]
      [Bad Checksum: False]
  ▽ Options: (20 bytes)
      Maximum segment size: 1400 bytes
      SACK permitted
      Timestamps: TSval 41548653, TSecr 864499
      NOP
      Window scale: 7 (multiply by 128)
  ▽ [SEQ/ACK analysis]
      [This is an ACK to the segment in frame: 132]
      [The RTT to ACK the segment was: 0.040780000 seconds]
```

3. kép: tcpreszl

Impresszum

Alapító-főszerkesztő:
Horváth Örs Apor

Felelős szerkesztők:
Pfeiffer Szilárd

Tördelőszerkesztő:
Falusi Ernő

Arculattervező:
Makay József

Szerzők:
Csíkos Bálint
Kovács Zsolt
Medve Zoltán
Sütő János
Szőke József
Vomberg István

Észrevételeket a lapszámmal kapcsolatban az alábbi címen várunk:
http://www.flosszine.org/II_evf_003_szam

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

Köszönet az FSF.hu Alapitványnak a tárhelyért!

Az e-fanzine elkészítéséhez kizárólag nyílt forráskódú, szabad és ingyenes szoftvereket használunk.