FLOSS ZINE

Free / Libre / Open Source Software fanzine

2008. december

I. évfolyam 3. szám

BEMUTATKOZIK A MAGYAR FREEBSD DOKUMENTÁCIÓS PROJEKT!

FreeBSD

The Power To Serve

DIA

OPEN SOURCE FARM

HEXEN 2

MADDOG

HELLO WINDOW 2

HELLO WORLD 3

BASH 2

Tartalom

Lite

Pro

Minden Tisztelt Olvasónknak boldog, eredményes Újévet kívánunk, valamint szabadabb szoftverhasználatot, és több időt az alkalmazásfejlesztésre!

DIA

Talán sokan emlékeznek még a lemezből készített diavetítőre. Itt most nem erről lesz szó. Azért nem kell elkeseredni, most már az internetre is nagyon sok diafilm felkerült, így a lemezes diavetítő helyett ezeket meg lehet nézni a lemezes és lemezfalú számítógépeken is. Egy kis kereséssel rengeteget találhatunk, de az alábbi valóban kihagyhatatlan.
http://www.diafilmmuzeum.hu/

Amiről szó lesz, az a Dia nevű program. A Dia egy GTK+ alapú diagramkészítő program, Linux-ra, Unix-ra és Windows-ra is. A programra GPL licenc vonatkozik, így most karácsony előtt tökéletesen kielégíti a "szok, kici, olcó" követelményeket. Annyit töltünk le belőle, amennyit csak akarunk, amire akarjuk (persze előbb kérdezzük meg az oldalon, mit akarjunk), mindezt teljesen ingyen.

Annak ellenére, hogy még nem érte el az 1-es verzió-számot (Version 0.96.1), teljesen jól használható (persze ez a *X világában megszokott).

Ünnepi, akciós link, *X-re:
http://live.gnome.org/Dia/Download

Windows verzió:
http://sourceforge.net/project/downloading.php?groupname=dia-installer&filename=dia-setup-0.96.1-6.exe&use_mirror=puzzle

Eredetileg nem akartam sokat emlegetni a Visio-t a Dia cikkben, nem szeretném összehasonlítani őket, ez már csak az áruk miatt sem lenne szerencsés, (de már meggondoltam magam), viszont az oldalon közlik, hogy a program tudja a Visio VDX formátumának importját, exportját, és már hibákat is javítottak ezzel kapcsolatban.
A Dia hasonló feladatok elvégzésére alkalmas GNU/Linux és barátai alatt, mint a Visio Windows környezetben, kezeli annak formátumát, sőt ő maga is elérhető Windows-on is.
A Dia azon túl, hogy többféle rendszeren használható, elég jól dokumentált jószág is egyben.
A letöltő oldalról könnyen elérhetjük ezeket az információkat.
A program Linux és Windows verziójának menüi minimális mértékben eltérhetnek, de ez a funkcionalitást nem zavarja.
Legegyszerűbben a Debian package telepíthető, igaz ennek verziószáma csak 0.95, de nincs túl nagy különbség.
A telepítéshez ki kell adni a rendkívül bonyolult „apt-get install dia” parancsot a vírusos képernyőn. (Az ősidőkben egy rettegő hang hívott telefonon, és sikoltozva közölte velem, menjek azonnal, mert vírusos a gépe. Kérdésemre, hogy mégis mi alapján jutott erre a következtetésre, szörnyűlködve mesélte el, hogy egyik pillanatról a másikra fekete alapon fehér betűk jelentek meg a képernyőjén, stb. Ő még ezek alapján az átlagosnál jóval tájékozottabbnak tűnt. Mivel nem tudott tovább dolgozni, kénytelen voltam megnézni az ominózus képernyőt. Az történt, hogy hőn szeretett programjából véletlenül kilépve szembetalálkozott a szörnyű DOS-szal, a kíméletlen karakteres képernyővel. Ma már csak a DEL billentyűre könyöklő delikvensek hívnak néha tanácstalanul, az akaratlanul előállított BIOS setup képernyő előtt szomorkodva. Jobbik eset, ha nem csinálnak semmit, azalatt a pár óra alatt, míg megérkezik valamelyikünk.) Nos ezt a vírusos képernyőt a konzol képes helyettesíteni *X-en, persze nehezebb véletlenül idekerülni, meg nem is nagyon kalapálják az ilyen gépeket a mezei userek, de ez még változhat.
Ha nem sikerült elírni a parancsot, akkor jó esetben minden lezajlik automatikusan, még a program ikonja is bekerül a gnome menübe.

Nézzük a programot részletesebben.

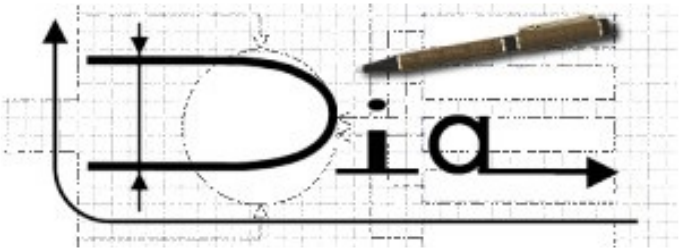
Diagramkészítő alkalmazásként rákényszerül valamiféle fájl menü felkínálására, mint a jólnevelt szerkesztőprogramok általában.

Ezt túlságosan nem részletezném, elég általános, csak a mentés másként, és az export funkciókat taglalnám.

A szokásos, Új, Megnyitás, Mentés menüpontok alatt található a Mentés másként menüpont (persze csak a magyar nyelvű verzióban írják így).

A Megnyitás-ban választható formátumokat azért még ideírom (saját diagramjai .dia, .dxf, a következő formátumok Pixbuf verziója: bmp, gif, jpeg, jpe, jpg, png, svg, svgz, svg.gz, tga, targa, xpm /Pixbuf vége/, svg, Visio XML .vdx, .wpg, .fig).

A víziló ezzel szemben saját formátumain kívül csak



az svg állományokat tudja megnyitni (Visio 2007).

A Mentés másként lehetőséget használva ábránkat a program alapértelmezett formátumában, .dia kiterjesztéssel tudjuk elmenteni. Ezt, mint korábban láttuk, szerencsére vissza is tudja olvasni a program. Itt bejelölhetjük pipával a diagramfájlok tömörítése opciót, ha akarjuk.

Ezalatt találjuk az Exportálás menüpontot.

Itt elég sok formátumban menthetjük el művünket: beágyazott postcript, a következők Cairo változata: png, ugyanez alfával, pdf, ps, svg, /Cairo vége/, .cgm, Dia alakfile .shape, Dia diagram .dia, .dxf, .emf, .plt és hpgl, a következők Pixbuf változata: bmp, ico, cur, jpeg és társai, png, tiff, tif, /Pixbuf vége/ sima ping .png, .svg, .mp, .tex, Visio XML .vdx, .wmf.

A Szerkesztés menü többnyire a szokásos pontokból áll: Visszavonás, Újra, Másolás, Kivágás, Beillesztés, Másolat, Törlés, Szöveg másolása, Szöveg kivágása, Szöveg beillesztése.

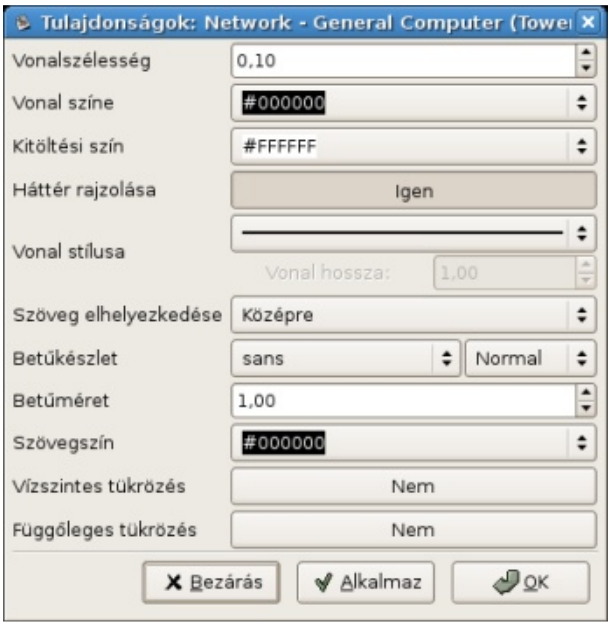
A Diagram menüsor a következő két pontból áll: Tulajdonságok, Rétegek.

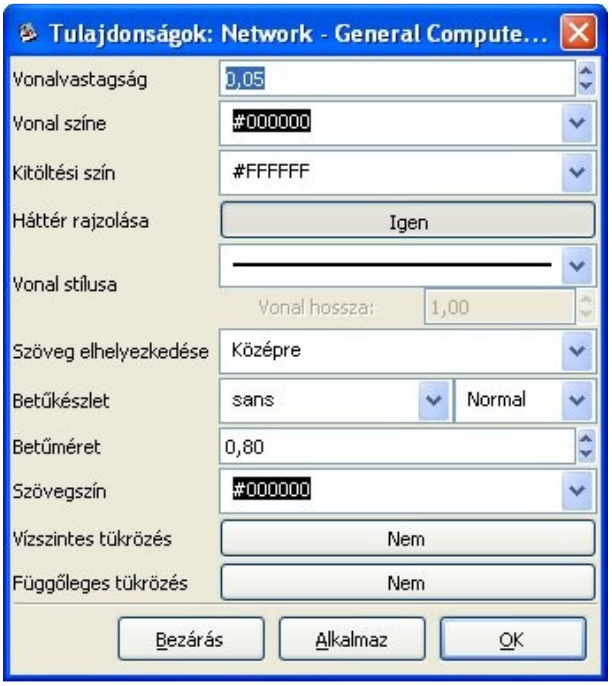
A Tulajdonságok-nál a rács és a színek beállításait módosíthatjuk, a Rétegek pontban rájöhetünk, hogy a Dia kezeli a rétegeket.

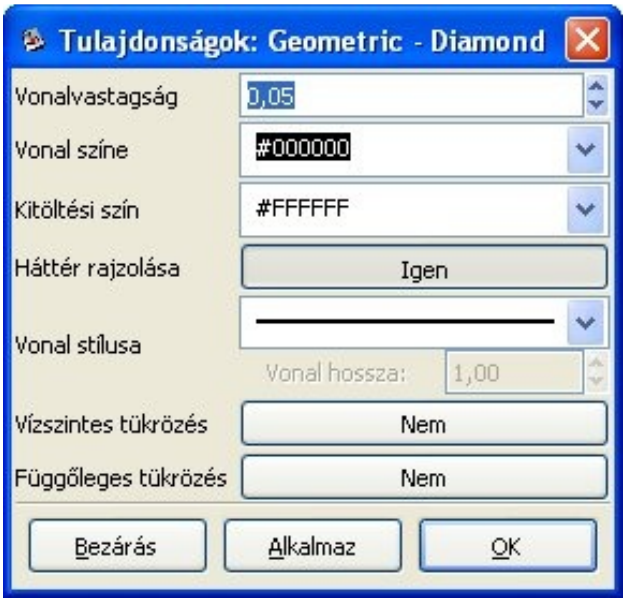
A Nézet menüsorba kerültek a kicsinyítéssel és a nagyítással kapcsolatos tételek, és a megjelenéssel kapcsolatos beállítási lehetőségek.

Objektumok menü. A szokásos tételeken túl (objektumok sorrendje, csoportosítás, stb) három gyám-

A menüpontok tartalma változhat, például egy objektumra vonatkozóan, de kisebb verziószám eltérés nem jelent egetverő különbségeket.







ügyi lehetőség is idekerült, úgymint: Szülőhöz kötés, Szülőtől elválasztás, Gyermekek leválasztása. Alatta az Igazítás menüpont, és a Tulajdonságok menüpont található. Az objektum tulajdonságainál a vonalvastagság és szín, kitöltési szín paramétereket változtathatjuk meg, valamint a háttérrajzolás kapcsolót állíthatjuk be (igen/nem). Beállíthatjuk még a vonal stílusát és hosszát, legvégül pedig még két kapcsológombot nyomkodhatunk, a vízszintes és a függőleges tükrözést kapcsolhatjuk ki és be (igen/nem).

Ezután a Kijelölés menüt találjuk, az alábbi pontokkal: Mindent, Semmit, Megfordítás, Tranzitív, Összekötött, Azonos típus, Csere, Unió, Metszet, Eltávolítás, Megfordítás. Könnyen belátható, hogy ezek egészen kiváló lehetőségeket rejtenek.

Ezután jön az Eszközök menü. A Dia rendelkezik egy később részletezett úszómenüvel, az Eszközök menüpont többek között tartalmaz néhány dolgot, ami itt is, ott is megtalálható.

Magyarul némely tevékenységhez nem kell az úszómenüből kiválasztani az eszközt, az itt, a legördülő részben is megvan.

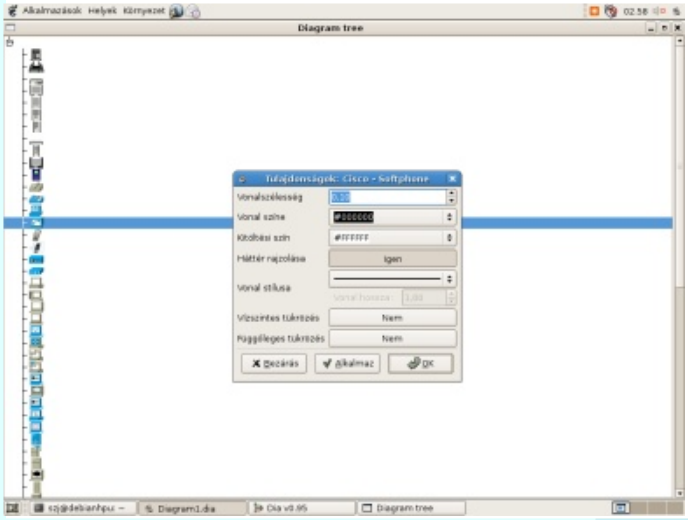
Az itt szereplő lehetőségeket kivétel nélkül elérhetjük billentyűkombinációk segítségével.

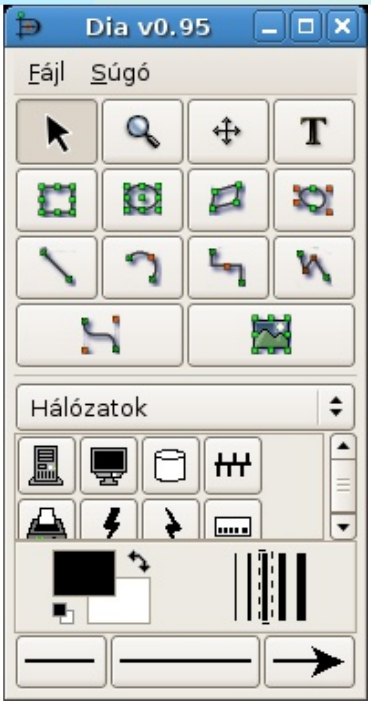
A menüpontok: Módosítás, Nagyítás, Görgetés, Szöveg, Doboz, Ellipszis, Sokszög, Bezier-síkidom, Vonal, Körív, Cikkcakkvonal, Töröttvonal, Bezier-görbe, Kép.

Ezután a Beviteli módok menüpont következik, ebből nekünk kezdetben megfelelő lesz az Alapértelmezett menüpont. A többi speciális karakterek bevitelére szolgál. A Cirilltől kezdve az etióp amharic-on és tigrigna-n át a vietnami-ig rendkívül sokrétűen támogatja a program ezt a tevékenységet. A Windows verzió támogatja a Windows IME-t, és mindegyik az IPA-t (International Phonetic Alphabet).

Végére maradt a Súgó menü, itt ketten laknak, maga a Súgó, és Névjegy barátja.

A magyar verzió súgója is angol, de ezen nem szoktunk meglepődni. Bátor, tetterős, fiatal titánokra vár





a feladat, hogy ezen változtassanak, ha szükségét érzik.

Úszómenü: Ennek is van egy legördülő menürendszere (egyből mindjárt két tételből álló), A Fájl menüben lévő Új és Megnyitás már ismerősek lehetnek, de alattuk akad némi érdekesség.

Diagramfa. Ez egy jópofa dolog, megjelennek az ábrán szereplő objektumok, és ezekre jobb egérgombbal rákattintva, (persze ehhez kell egy jobb egér) különböző műveleteket hajthatunk végre velük.

Alatta kapjuk a Munkalapok és objektumok menüpontot. Itt a munkalapokon elhelyezkedő objektumok között vághatunk rendet, csinálhatunk olyan kedvenc munkalapot (shape), amin minden gyakran használt objektumot elhelyezve gyorsabban dolgozhatunk.

Mellette a Súgó és Névjegy páros már ismerős, ugyanaz, mint a másik menürendszerben.

Ezek alatt, az ikonok a leggyakrabban használt eszközöket jelentik, majd egy munkalap választó lehetőség következik, alatta az objektumok elő és háttérszínét, valamint a vonalvastagságot lehet beállítani. Végül a vonalak és nyilak stílusát tudjuk módosítani.

Már csak a munkálkodás maradt. Ez rendkívül egyszerű, mivel a program könnyen kezelhető.

Dobjunk össze egy kis hálózati diagramot, a következőképpen (persze a munkastílusát mindenki maga határozhatja meg).

Bonyolultabb dolgokat szerencsés lehet előre eltervezni, itt most ezt a lépést kihagyjuk.

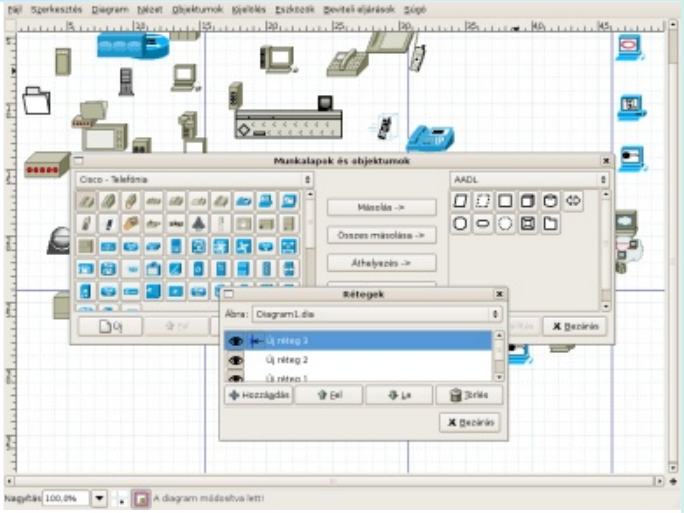
A szükséges objektumokat rádobáljuk a rétegre (rétegekre), előtte persze csinálhatunk egy hátteret is, ha akarunk.

Az objektumkönyvtár nemcsak bőséges, de mi magunk is bővíthetjük, hozhatunk létre újakat, vagy szerezhetünk az internetről.

Csak számítógépekből van vagy háromféle objektumkönyvtár alapból. Innen a kis kézigépektől a takarító néni háromajtós ruhásszekrényéig (ebbe szokta betenni a vizes esernyőjét, olyankor leáll az egész hóbelevanc, vagy kihúz néhány dolgot a konnektorból és bedugja a porszívóját, meg a mobiltöltőjét, az se rossz) sokfélét találhatunk.

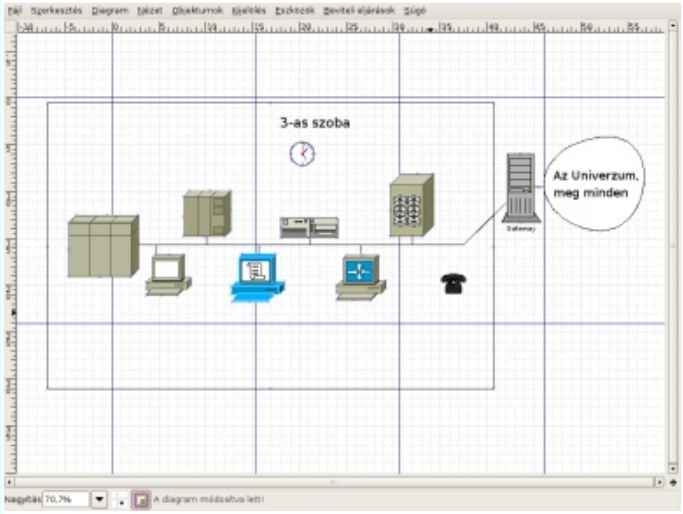
Az elemeket összekötő vonalakat a megfelelő helyre

húzva, azok követni fogják az elmozdított objektumokat, így nem kell mindig bajlódni velük, ha átrendezzük a diagramot.
Egy kisebb hálózatot néhány objektum és vonal használatával percek alatt létrehozhatunk.
Művünket többféle formában elmenthetjük, vagy kinyomtathatjuk.



Végszó.

A Dia elég jól használható program, persze nem mindenben veszi fel a versenyt néhány komolyabb kereskedelmi termékkel, de nem is ez a célja.
Az egyik ismert versenyzővel a Visio-val összevetve



megállapítható, hogy a Visio eladhatóbb külsejével, komolyabb alkalmazásnak tűnik, de nem biztos, hogy ez a helyzet.
A Visio verziói többnyire az objektumkönyvtárban különböznek, a Dia alapból rengeteg objektumot bocsát rendelkezésünkre.
A diagramok kinézete egy-egy vázlat esetében kevésbé lehet fontos, mint az áttekinthetőség.
A könnyű kezelhetőség szintén a Dia sajátja.
Vegyünk két víziemlőst, ha a Visio a víziló, akkor a Dia egy vidra, kisebb, könnyebb, gyorsabb, de nem tud mindent, amit egy víziló (például egy embert agyonnyomni), persze ez visszafelé is igaz.
Rendes leszek, nem teszem ide a víziló és a vidra képét, ehelyett annyit mondok, hogy megfelelő feladatra a megfelelő programot. (Ezt az idióta párhuzamot erősítendő: a Dia Windows alatt, tokkal vonóval, kevesebb mint 50 Mb helyet foglal. A Visio egy Office mentes gépre feltelepítve felrakja magát és néhány Office kiegészítést, ezek után több mint 300 megabájton terpeszkedik).
Amennyiben tisztában vagyunk vele, mit akarunk, és úgy döntünk, a Dia alkalmas erre, bátran vágjunk bele, a program barátságos segítőtársunk lesz ebben.

Szőke József

A cikkhez tartozó fórum címe:
http://www.flosszine.org/dia

Tudtad-e?

A világ egyik legnagyobb, bárki által szabadon használható internetes archívuma itt érhető el: http://www.archive.org. A "gyűjtemények gyűjteménye" nemcsak mozgókép-, élőzene-, audió-, szoftver- és szövegtárakat tartalmaz, illetve rendszerez, hanem a legkülönbözőbb oktatóanyagok, egyetemi kurzusok írásos és video-anyagait is hozzáférhetővé teszi. Igaz, a legtöbb tartalom angol nyelvű, de más nyelvű kollekciókra is könnyedén rá lehet akadni az oldalon, ahonnan legalább 4-5 fájlformátumból lehet választani az egyes tételek letöltésekor. A filmek között jelenleg például Chaplin burleszkjei (http://www.archive.org/search.php?query=Chaplin%20AND%20mediatype%3Amovies) igen népszerűek, az egyik legtöbbször letöltött könyv pedig Szent Ágoston "Az Isten államáról" ("De Civitate Dei") írt művének 1475-ből származó, latin nyelvű kiadásának digitalizált változata (http://www.archive.org/details/augustinidecivitatedei00jensuoft).

VESZETT KUTYA

Jon Maddog Hall 2005-ben megrökönyödést keltett a kevésbé tájékozott Linux-felhasználók körében azzal, hogy 90 ausztrál vállalatot szólított fel jogdíjfizetésre(1) a Linux International(2) ügyvezető igazgatójaként. Hall arra hivatkozott, hogy nem szabad összetéveszteni az innováció lehetőségeit korlátozó szoftverszabadalmak ügyét a márkavédjegyekével, ezért a csalók, és a névvel visszaélő cégek miatt kérnek pénzt a védjegy használatért. Az LI eredetileg olyan számítógépgyártók non-profit szervezete, amelyek segíteni és reklámozni szeretnék a Linux operációs rendszereket, ma azonban már Maddog a végfelhasználók testületének hívja.

Maddog -a szabad szoftver mozgalom egyik vezetője- három évvel ezelőtt már utalt arra, hogy 2000-ben Linus Torvalds lefektette a licenchasználat szabályait. Ezek értelmében mindenkinek, aki Linux névvel forgalmaz egy terméket, vagy szolgáltatást, meg kell előbb vásárolnia a Linux név használatának jogát. Kivételt ez alól csupán a non-profit alkalmazások képeznek, amelyek vagy ingyenes licencet kaphatnak, vagy licenc nélkül forgalmazhatók. Az előzmény az, hogy 1994-ben és 1995-ben több vállalat és magánszemély is a sajátjaként jegyeztette be a Linux márkanevet, és megpróbált jogdíjat beszedni érte(3). Miután azonban a felhasználók és a fejlesztők közösségében ezzel sokan nem értettek egyet, Linus Torvalds a LI segítségével megtámadta ezeket a védjegybejegyzéseket és végül odaítélték neki a jogot - legalábbis az Egyesült Államokban és Németországban(4). Azóta a licenc ügyét a The Linux Foundation(5) nevű világszervezet gondozza.

Maddog már sokkal azelőtt használt és fejlesztett open source alkalmazásokat, mielőtt globális védelmezőjévé vált volna a nyílt forráskódú fejlesztőkultúrának. Igaz, első fejlesztői próbálkozásainak idején, 1969-ben még mindenki egyszerűen csak szoftvernek hívta azokat, hiszen zárt forráskódok akkor még nem léteztek. A hacker különös becenevét a tanítványaitól kapta a Hartford State Technical College-ban, ahol egy ideig a számítástechnika tanszék vezetője volt. Azóta is szereti, ha a Maddognak, vagyis "Veszett kutyának" szólítják, bevallása szerint ez a név még „abból az időből ered, amikor kisebb befolyásom volt a vérmérsékletemre.(6)”

A hackerré válás legendáját Maddog egy gyerekkori történetével is erősíti. Mint többször is elmesélte, négy éves korában beledugta a televíziójuk antennáját egy konnektorba, az áramütés keresztülrepítette a szobán, ő pedig felismerve a technika „ere-

jét", így rögvest aköré kezdte felépíteni a karrierjét, majd az életét. Előbb édesapja játékboltjában szerelte össze a játékokat, majd a középiskolában elektronikára szakosodott. Egy ifjúkori barátjának áramütése után azonban végleg a gyengeáram és a szoftverfejlesztés felé fordult. Vélhetően apja kívánságának megfelelően, először a Drexel Egyetemen szerzett kereskedelmi és mérnöki diplomát 1973-ban, utána a Rensselaer Polytechnic Instituteben MSc fokozatot kapott számítástechnikából 1977-ben. Több, mint 30 éves pályafutása alatt Hall volt programozó, rendszertervező, rendszergazda, termékmenedzser, technikai-kereskedelmi igazgató, író, tanácsadó a világ legkülönbözőbb pontjain önkormányzati és kormányzati szerveknél, valamint főiskolai oktató. Olyan vállalatoknál alkalmazták, mint a Western Electric Corporation, az Aetna Life and Casuality, a Bell Laboratories, a Digital Equipment Corporation, a VA Linux Systems, és a Cisco Graphics.

Jelenleg Maddog a Koolu(7) nevű cégénél tevékenykedik, amelynek társalapítója is egyben. A kanadai vállalkozás a nyílt programokkal (Debian, Android) működő mobiltelefonok -mint az Openmoko- fő elosztója, másfelől azonban a telefonszoftverek fejlesztését és frissítését is végzi, és a fejlesztőközösség tevékenységét irányítja. Részt vesz továbbá a Google Alkalmazások (Google Apps) terjesztésében és egyéb nyílt forráskódú szoftverek fejlesztésében, illetve piacra juttatásában.

Maddog több helyen is hangoztatta azt a meggyőződését, hogy a következő 5-10 évben a Linux, és más nyílt rendszerek elkerülhetetlenül átveszik a globális vezető szerepet. Szerinte erre utal, hogy a szuperszámítógépeken is ezek futnak, és hogy a legnagyobb fejlődő országok is tömegesen vezetnek be Linux-alapú megoldásokat a különböző ágazatokban, miközben az asztali gépek szintén egyre gyakrabban futtatnak Linuxot. Hall érvei között továbbá az is szerepel, hogy a kereskedők is jobban járnak a szabad szoftverek terjesztésével, hiszen nem kell termékdíjat fizetniük utánuk. A "Veszett kutya" másik közkeletű nézete, amelyet szívesen terjeszt, főleg a harmadik világban, hogy a mobilitás, és az informatika együttes korszaka köszönt be nemsokára, vagyis a funkciókban egyre gazdagabb mobiltelefonok, a vékony kliensekkel installált asztali számítógépek, valamint a "felhőkben" futó, központi alkalmazásplatformok fogják jellemezni az egyre inkább közös csoporttevékenységekre specializálódó világhálót(8).

Hivatkozások:
(1) http://www.origo.hu/techbazis/szoftver/20050822torvalds.html
(2) http://www.li.org/
(3) http://en.wikipedia.org/wiki/Linux_Mark_Institute
(4) http://www.linuxmark.org/
(5) http://www.linuxfoundation.org/en/Main_Page
(6) http://www.linuxworld.com.au/index.php/id;527801083;fp;4;fpid;3
(7) http://www.koolu.com/
(8) http://koolu.com/component/option,com_mojo/Itemid,225/

Jankovich Oszkár

A cikkhez tartozó fórum címe:
http://www.flosszine.org/jon_maddog_hall

HOGYAN KEZDŐDÖTT ? II.

Az előző rész utolsó előtti mondatában a GNU és a GPL mozaikszavakat emlegettem, így innen kellene folytatni a történetet. Előtte szeretném megjegyezni (az első részben nem tettem), hogy ez a többrészes cikk nem lehet teljes körű számítástechnikai, vagy FLOSS történet, ebből a szempontból természetesen hiányosnak tekinthető, a bekerült információkat illetően pedig talán esetlegesnek tűnhet, de mindenképpen önkényes. Mivel a nyílt forrású szoftverek előzményeiről és jelenéről szerettem volna írni, a következő részben megpróbálom ezt átfogóbban bemutatni, de biztosan lesz, ami kimarad belőle, hiszen a történések felgyorsultak, az érintett témakör pedig sokkal kiterjedtebbek, szerteágazóbbak lettek, mint az őskorban (számítástechnikai) voltak. Ezért aztán a témakör mindenre kiterjedő bemutatása a cikk keretei között nem lehetséges.

Korábban említettem, hogy a nyílt forrású szoftverek megnevezése olyan kritériumokat határoz meg, amely az azokat nem teljesítő produktumokat egyértelműen kizárják ebből a körből.
A GNU projekt kitételeit és/vagy a GPL és a hasonló licencek előírásait teljesítő szoftverek ebből a szempontból beletartoznak a körbe, de nem ugyanonnan származnak.

1983 és 1984 fontos mérföldkövei ennek a folyamatnak, de ez nem jelenti azt, hogy ezelőtt ne léteztek volna nyílt rendszerek. Néhány párhuzamosan futó dolgot mindenképpen meg kell említeni, kezdjük a korábbiakkal.

Unix

A hardvertől elkülönült, fizetős szoftver nem mindig volt jellemző a számítógépek használata során. Amikor a számítógépek nagyobb számban kezdtek elterjedni, és szélesebb körben elérhető típusok is gyarapodtak, egyre inkább problémát jelentett egy-egy program átírása egy másik rendszerre. Ezek a gondok már elég korán jelentkeztek.

A Unix története a Multics projekttel kezdődött, a Multics indulása pedig a MIT-hez köthető.
A részletek túlzott taglalása helyett elég annyi, hogy a Multics (Multiplexed Information and Computing Service) projektet a MIT és a GE kezdte, 1965-ben csatlakozott hozzájuk a Bell.
A cél, egy mindentudó operációs rendszer kifejlesztése volt. A készítők elég nagy célokat tűztek ki maguk elé, amit aztán nem sikerült teljes mértékben megvalósítaniuk. Ennek ellenére egészen sokáig működtek Multics-ot használó rendszerek, az utolsót 2000-ben állították le. A rendszert PL/I nyelven és assembly-ben fejlesztették.

Pár év alatt kiderült, hogy a készítők kissé túlvállalták magukat, ezért a - Multics-on is sokat dolgozó - Kenneth Lane Thompson, a Multics fejlesztése során elért eredményeket felhasználva, nekikezdett egy új rendszer kialakításának. Az elvárásokat újrafogalmazták, kisebb célokat tűztek maguk elé (Brian Kernighan eunuch Multicsnak nevezte a rendszert). Thompson igen rövid idő alatt megírta az új rendszer lényeges alkotóelemeit. 1969-ben lett kész a Unix első verziója, amit akkoriban nem így neveztek. A Multics nyomdokain haladva, az 1969 szeptemberében elkészült rendszert, Unics-nak (Uniplexed Information and Computing Service, más helyeken Uniplexed Operating and Computing System) hívták eleinte (Kernighan 1970-ben javasolta az elnevezést, ami azért jobb volt, mint az eunuch Multics), majd később a rendszer megkapta a Unix nevet. Az akkor még csak PDP-7-es gépen futtatható rendszert használni szerették volna PDP-11-eseken is, ehhez viszont az egészet újra kellett írni. A hagyományos módszerekkel való munkálkodás folytatása (hardverközeli nyelven való fejlesztés), szörnyűséges jövőt vetített előre – minden egyes gépcsere a rendszer egészének újraírását jelentette volna, valószínűleg mindig assemblyben.

Ettől kissé kirázta a hideg a fejlesztőket, ezért elhatározták, hogy a rendszert egy olyan hatékony, és gépfüggetlen nyelven írják újra, amely alkalmas ezekre a feladatokra.
Ez a nyelv a C nyelv őse volt, amit a fejlesztéshez korábban többedmagával csatlakozó Dennis MacAlistair Ritchie „szállított". A Unixot közösen újraírták ezen a nyelven, így létrejött a világ első hordozható operációs rendszere.

A 70-es évek elejére a Unix egy hatékony, jól átgondolt, és hordozható operációs rendszerré vált. Ennek köszönhetően rohamosan terjedt, egyre többen használták.
Ebben az időben az AT&T -az USA trösztellenes törvényeinek értelmében- elég sok mindentől eltiltották, többek között a számítógépes programok árusításától is. A Unix fejlesztése pedig a Bell Laboratoriesnál -az AT&T egyik leányvállalatánál- történt.
Így kezdetben, mivel nem lehetett eladni, a Unixot odaadták mindenkinek, aki kérte (és mindez a trösztellenes törvénynek volt köszönhető).
Mivel az egyetemeken ekkor már sok PDP-11-es gép volt, sokan szerették volna megismerni, használni a Unixot, (a PDP saját operációs rendszere, finoman fogalmazva kevésbé volt használható, mint a Unix akkori változata).
Az AT&T-t a szoftverekből ekkor nem csinálhatott pénzt, így a rendszer forráskódját is megkaphatták az érdeklődők, ezért az egyetemi hallgatók, tanárok és kutatók kedvükre barkácsolhatták C nyelven az operációs rendszert (persze ez nem ilyen simán és egyszerűen történt).
Innen már csak egy kis lépés minden dolgok kiindulópontja, a káosz. Ennek elkerülésére szolgáltak az ekkor megerősödő szabványosítási törekvések. Ennek egyik megvalósulása a POSIX (Portable Operating System Interface for uniX), amit szoktak Unix szabványként is emlegetni. Meg kell jegyzeni, hogy maga a POSIX sok vitát váltott ki, mivel nincs megegyezéb abban, hogy tulajdonképpen milyen területeken és mit kellene előírnia. A vitatkozást hagyjuk a témában járatosabb egyénekre, itt és most elég annyit tudnunk, hogy a POSIX egy összefoglaló neve azon szabványok családjának, melyeket az IEEE, a Unix operációs rendszerek API-jának meghatározásaként definiált. Formális neve

IEEE 1003, a nemzetközi szabvényé pedig ISO/IEC 9945. A POSIX elnevezést Richard M. Stallman javasolta. (Vajon honnan ismerős ez a név?)
Haladjunk tovább. 1971 elején írták újra a Unixot PDP-11-re. 1972-ben elkészül a C nyelv.
Ezután folyamatosan fejlesztették mindkettőt.

Mivel a részletes történet többféle formában is elérhető, pontos dátumokkal, itt csak a fontosabb eseményeket említeném meg.
A FLOSS közösség szempontjából fontos területeken párhuzamos történéseket kell figyelnünk, ezért néha időugrásokat fogok végrehajtani, remélhetőleg az időparadoxon okozta problémák nélkül.
A Multics, Unics, Unix folyamatnak nem akart vége szakadni, az egyetemi buherálások idejében több száz féle Unix létezett (egyébként a UNIX-ot többféleképpen írhatjuk, de két írásmódra - Unix, UNIX - érvényes védjegy oltalommal rendelkezik).
A Unixok rettenetes elszaporodása azzal járt, hogy egyes cégek teletűzdelték a kódot szabadalmi megkötések hatálya alá tartozó részekkel, később megjelentek a kereskedelmi Unix verziók is.
Ezek egy időben többé-kevésbé sikeresek voltak, de mára jórészt legyűrte őket a szabad világ. (Hurrá!)
Hogy a dolog még ennél is érdekesebb legyen, 1974-ben a kaliforniai Berkeley Egyetem rátette "szőrös mancsát" a Unixra, ugyanis ekkoriban jutottak hozzá egy Unix licenchez. Végtelen kedvességükben a rendszer használata során rengeteg apró hasznos programmal egészítették ki, és módosították az alap rendszert. Így jött létre a BSD, azaz Berkeley Software Distribution. Ami nem volt más, mint az akkori Berkeley Unix (1977). Ez mára az egyik alapváltozata lett a rendszernek.
A másik az AT&T által elindított Unix. Ennek forráskódját 1974-ben tették az oktatási intézmények számára ingyen elérhetővé, de az AT&T feldarabolása után az utód cégek lehetőséget kaptak, hogy a számítástechnikai iparban tevékenykedjenek, így megszületett az első kereskedelmi Unix.
A System III nem volt túl sikeres, de utódja a System V már sokkal inkább beváltotta a hozzá fűzött reményeket. Több kiadása is volt, Release 2, 3, 4 néven.
A System V és a BSD elkülönültek egymástól, de az összes, UNIX-ot továbbfejlesztő cég vagy csoport ezek valamelyikét használta, és használja ma is.
Mielőtt nagyon belemerülnénk a jelentős számú változat taglalásába, érdemes megnézni az alábbi oldalon az utolsó képet, vagy tanulmányozni az alatta elérhető pdf-et.
http://penguin.dcs.bbk.ac.uk/academic/unix/linux/slides/
http://www.levenez.com/unix/unix_a4.pdf
Nos ezért nem fogok minden verzióról hablatyolni, de azért néhányat megemlítek.
Nézzük tovább, mi is lett a Unix-szal.
A System V release 4 kibocsátására 1991-ben került sor. A Unix System Laboratories változata több rendszer vonásait egyesítette magában. Néhány más cég (IBM, HP, stb) ekkor eldöntötte, hogy létrehozzák saját standard Unix verziójukat, ezért létrehozták az OSF-et (Open Software Foundation).
1993. júniusában az AT&T eladta UNIX-át a Novell-nek, így a Unix System Laboratories a Novell UNIX System's Group részévé vált. A Novell elkészítette a UnixWare-t, amely a System V 4-en alapult, de kapcsolatot tudott tartani a Novell NetWare-rel is.
Még ebben az évben a Novell az X/Open-re ruházta át a UNIX névhasználati jogát. Ez a szervezet a POSIX ajánlásokat továbbfejlesztve, a gyártókkal konzultálva megalkotta a Spec 1170 ajánlást,

amely akkor a UNIX minősítés alapját képezte (UNIX 95).
Néhány centiméterrel lejjebb megemlítem a nagyobb cégek saját Unix verzióit, (ezekből van pár, ami ma már nem használatos):
IBM - AIX, HP - HP-UX, Silicon Graphics - Irix, Next - NextStep, a Sun Microsystems először SunOS-nek nevezete a saját verzióját, majd Solarisnak, de mint később látni fogjuk, ezek más alapokon készültek, Novell - UnixWare, SCO - SCO UNIX, Digital - Ultrix, majd DEC OSF/1, amiből később Digital UNIX lett, Microsoft - Xenix, stb.
A Sun először a BSD 4.2-es verzióján alapuló SunOS-nek nevezett rendszerrel próbálkozott, majd váltottak, és a System V Release 4 rendszerre épülve megjelentek a Solaris különböző verzióival. Sok vita folyik a Solaris (és a Sun) jövőjéről, mindenesetre a Solaris ma az egyike a kevés számú, de elterjedt Unix verzióknak.
Az előbbi felsorolásban szereplő UNIX verziók közül néhány ma már nem szerepel a piacon, sőt néhány cég is eltűnt a süllyesztőben, vagy felvásárolták.
Dicsőséges végjátékával a Santa Cruz Operation mindenképpen kiemelkedik a mezőnyből, ez a cég a UNIX jogok letéteményeseként fellépve, fejlesztő és forgalmazó cégből degradálta magát a mindenkivel perre menő szomszéd, és a Unix/Linux felhasználókat gonosz banyaként ijesztgető idióta szintjére. Az eredmény közismert.
A perből a Linux, és a Linux közösség jelentősen megerősödve került ki, így az SCO és a mögötte állók eredeti céljukkal szemben, teljesen ellenkező eredményt értek el.
A UNIX ma is folyamatosan fejlődik, elérhető teljesen nyílt formában, akár PC kategóriájú gépeken és szervereken, valamint nagyobb teljesítményű szerverkörnyezetben, sokszor nem annyira nyíltan.
A Solaris kódjára épülő OpenSolaris talán a legkönnyebben elérhető UNIX verzió jelenleg.
Az IBM AIX, és a HP UX jellemzően inkább a nagyobb teljesítményű kiszolgálók rendszere.

BSD

Talán van aki emlékszik még, minden bogár rovar, de nem minden rovar bogár.
A BSD Unix, de a Unix nem BSD. Vagyis Unix alapokon indult a BSD, de ma már a kettő nem teljesen ugyanaz. Sőt, mint a „jobb" pop együttesek, osztódással szaporodik.
Nézzük ezt a folyamatot kicsit részletesebben.
Korábban már említettem, hogy 1974-ben egy amerikai egyetem, melynek neve University of California, Berkeley, hozzájutott egy Unix licenchez. Ebből lett a Berkeley Software Distribution, azaz BSD. Hogyan? Semmi ördöngösség nincs benne, hacsak a kabala ördögöt nem számítjuk ide, ugyanis az idők során a BSD emblematikus figurája egy kis ördög, vagy ördögfióka lett, a BSD démon (az angolszászok szerint). Megjelenési formájában elég változékony, de legalább mindenki ízlésének megfe-

The original BSD daemon appeared first in 1983 on the cover of the 4.2BSD manuals published by the Usenix Association

lelően választhatja a neki szimpatikusat.

Mint sok más esetben, itt sem egyértelműen meghatározhatóak a dátumok, sok anyag eltérő időpontokat ad meg a történet különböző mozzanataira, a levéltári kutatásokat pedig jelenleg hanyagolnom kell, így lehet vitatni az egyes időpontokat, de ez a történet lényegét, ilyen szinten nem befolyásolja.
Egyes források szerint a Berkeley Egyetem 1974-ben jutott egy Unix licenchez, és telepítőszalaghoz. Ezt a Unixot kezdték fabrikálni, miután sikerült beszerezniük egy PDP-11-est.
Az egyetem két végzős diákja, Bill Joy és Chuck Haley 1975 őszén kezdett érdeklődni a rendszer iránt. Az első terjesztés összeállításáig elég sokan reszelgették a rendszert. 1977-ben Joy létrehozta az első Berkeley Software Distribution-t, az 1BSD-t. Ebben benne volt egy Pascal rendszer, forrással együtt, és az ex szerkesztő is. A következő évben kb.:30 másolatot küldtek szét a rendszerből.
1978-ban jelent meg a 2BSD, ebből már 75 szalagot küldtek ki. A 2BSD végső verziója, a 2.11BSD már komplett rendszer volt, több száz futó példánnyal a PDP-11-es gépeken, a Föld különböző pontjain.
Az egyetemen sem állt meg az élet, az új igényekhez új gép is kellett, így került egy VAX gép a Berkeley-re. A gép saját rendszerének korlátai miatt Bill Joy elkezdte portolni a BSD-t VAX-ra.
A folyamat vége a 3BSD lett, mely a Berkeley első VAX terjesztése lett (1979 végén száz másolatot készítettek belőle).

Ezután történt egy kis változás a UNIX életében, az egyetem és a Bell Laboratories együttműködését új alapokra helyezte az örökké éhes „karvalytőke” :). A 32/V volt az utolsó Bell kiadás, ezután az összes UNIX az AT&T-től származott (System III, majd System V) ezek már stabilan kereskedelmi kiadások voltak (ezek sosem tanulnak :)). Mivel ekkor a Unix teljesen üzleti alapokra került, a Bell kutatói nem adhatták tovább a Unix-szal kapcsolatos eredményeiket.
Ennek ellenére a kutatók folytatták a Unix fejlesztését, ezért kellett egy szervezet a kutatási verziók előállítására és terjesztésére. Így ebben a Berkeley lépett a Bell helyére.
Mindeközben új szereplő lépett a színpadra, a Defense Advanced Research Projects Agency (DARPA). Korábbi tevékenységüknek köszönhetően olyan országos hálózat jött létre (nem a Szovjetunióban, és nem ügynökhálózat), ami összekötötte az összes nagyobb kutatóközpontot, (ezek meg valamiért erősen kötődtek a nagyobb egyetemekhez).
Az amerikai egyetemeken ekkor már sokféle hardver gyűlt össze, némelyikük túl volt fiatal évein is, de a kutatási szempontokat figyelembe véve, nem lett volna szerencsés az együttműködés alapjául egy egységes hardverplatformot kialakítani. Inkább úgy döntöttek, hogy a gépeket az operációs rendszer szintjén fogják együttműködésre bírni.
Mivel a Unix, azaz a BSD már korábban bizonyított, erre esett a választás.
Végső soron a 3BSD alapján kellett kifejleszteni egy bővített verziót, a DARPA közösség számára.
1980 végén jelent meg a 4BSD, amely 150 intézményben 500 gépen futott (a licenc az oktatási intézményekre szólt).
Ezt követte a 4.1BSD, amit az 5BSD követett volna, ha az AT&T nem él ellenvetéssel. (Ez ismerős lesz máshonnan is, a történelem ismétli önmagát).
Az AT&T azt találta mondani, hogy a felhasználók össze fogják keverni a System V elnevezést az 5BSD elnevezéssel. Hmm, ámerika?
Valószínűleg ezek a felhasználók nem a Berkeley-n tömörültek, így az okosabb enged elvnek megfelelően az egyetemen belementek abba, hogy ezután a nagyobb számot nem, csak a kisebbet fogják növelni, tehát marad a 4BSD, mint az elnevezés alapja.
A DARPA az első másfél éves szerződés után, (ez idő alatt Bob Fabry felállított egy szervezetet, a projekt támogatására, amelyet Computer Systems Research Group névre kereszteltek el, CSRG) újabb két éves szerződést kötött az egyetemmel, az ezzel járó támogatását megtöbbszörözte.
A szélesebb körű terjesztésre nem került 4.1a, és 4.1b után kiadták a 4.1c, majd nem sokkal később, 1983 augusztusában a 4.2BSD-t. Időközben Bill Joy csatlakozott a Sun Microsystemhez (1982).
A 4.2BSD lett az addigi legnépszerűbb kiadás. Több mint ezer site licenc másfél év alatt! (Ekkoriban még nem volt minden tyúkólban 3 számítógép.)
Másik érdekessége a kiadásnak, hogy a legtöbb gyártó a BSD-vel szállította rendszereit, az AT&T által árusított kereskedelmi System V helyett, mivel az sem hálózati képességgel, sem a BSD-ben megta-

lálható Berkeley Fast File rendszerrel nem rendelkezett.
A BSD tartotta vezető helyét néhány évig, amíg az összes BSD fejlesztés bele nem került a System V-be. A gyártók csak ezután váltottak vissza a System V-re.
1986-ban jelent meg a 4.3BSD. Ezután Keith Bostic nekiállt portolni a rendszert PDP-11-re, ami végül sikerült neki, így megjelent a 2.11BSD kiadás. Ez olyan jól sikerült, hogy egészen 1998-ig használatban maradt, az utolsó PDP-11-eseken. (Az eredetileg VAX-on 250 kbyte-ra forduló rendszert kellett a 64 Kbyte-os PDP-11-be préselnie. Ma meg :(, (inkább nem is mondok semmit).
Eközben a VAX is kissé öregecske lett, így valamit kellett tenni a BSD-vel, hogy újabb masinákra lehessen portolni. Úgy döntöttek, hogy a kernelt feldarabolják gépfüggő és gépfüggetlen részekre (ez a döntés később nagyon hasznosnak bizonyult).
1988-ban megjelent a Computer Consoles Inc. által gyártott Power 6/32-re készített 4.3BSD-Tahoe, ahol az elnevezés a hardvergyártótól származott. A Power 6/32 gép támogatása nem sokáig tartott, de a portolás során elvégzett munka nem veszett kárba.
Eddig a kiadásig (4.3BSD-Tahoe), a BSD megrendelőknek meg kellett vásárolniuk az AT&T forrás licencét, mivel a terjesztés mindig tartalmazta a forrást is.
Azzal, hogy a felhasználók belenézhettek a forráskódba, passzív használóból aktív közreműködőkké, „fejlesztőkké" válhattak. Ennek az elképzelésnek az ereje az évek során teljesen egyértelművé vált, csakhogy mindeközben folyamatosan nőtt a forrás licenc ára.
Mivel a hálózati kód nem volt része a korai 32/V-ne

k, ezért a gyártók kérték a Berkeley-t, hogy bontsa ketté a BSD-t, és alakítson ki olyan licencfeltételeket, hogy ne legyen szükséges megvenniük a forráslicenct, ha nem akarják.
Így az eredetileg a Berkeley által fejlesztett BSD hálózati kód, és az egyetem által fejlesztett programok 1989 júniusában megjelentek, Networking Release 1 néven. Ez volt a Berkeley első szabadon terjeszthető BSD kiadása.
Érdemes odafigyelni a licenc feltételekre, ezek olyan szabadok voltak, hogy lehetővé tették a kód terjesztését változatlan, de változtatott formában is, sőt csak binárisan is terjeszthető volt a rendszer. A kódért a Berkeley-nek jogdíjat sem kellett fizetni. A forráskódban a szerzői jogi szöveget változatlanul kellett hagyni, és minden, a rendszer kódjára épülő termék dokumentációjában fel kellett tüntetni, hogy a termék kódja a Kaliforniai Egyetem, és a hozzájárulók munkájára épült.
Az adathordozó szalagért 1000 dollárt kértek, de mindenki szabadon másolhatta azt, ha már megvolt neki. Így ftp-re is felkerült. A további fejlesztéseket a megvásárolt szalagok árából tudták biztosítani.
Az alaprendszer fejlesztése tovább folytatódott, virtuális memória alrendszer és NFS (Network File System) került bele. A 4.4 kiadása előtt tesztelni szerették volna az új lehetőségeket is, így kiadtak egy 4.3BSD-Reno nevű verziót 1990 elején. (Nem Jean Renoról kapta a nevét).
A szabadon terjeszthető networking release népszerűsége miatt felmerült, hogy ki kellene adni egy bővített verziót ebből is. Mivel szerették volna, hogy minél több BSD kód legyen benne, ez elég nagy munkának tűnt. Végül kitaláltak egy hálózat alapú fejlesztési metódust, aminek segítségével sikerült elkezdeni a munkát. A Unix programokat újraírták. Másfél év alatt szinte minden fontosabb program és könyvtár esetén sikerült ezt megtenni. Még hátra volt a kernel átírása, ami Mike Karels, McKusick és Keith Bostic munkájának köszönhetően, úgy történt, hogy az egész rendszeren végighaladva, fájlonként ellenőrizve a kódot, eltávolították a régi 32/V kiadás kódjait.
A végére hat olyan fájl maradt, amiben még voltak eredeti 32/V kódok.
Sajnos ezeket már nem írták át, inkább engedélyt kértek a rendszer ilyen formában történő kiadására. Hogy ne kelljen új licencet írni, úgy döntöttek,

hogy az új kiadást Networking Release 2-nek fogják hívni. Ez 1991 júniusában jelent meg. Az ezer dolláros szalagot, (mint korábban is) néhány száz szervezet és magánszemély megvásárolta.
Mivel a teljes értékű szabadon terjeszthető rendszerhez már csak az említett 6 állományt kellett átírni, ez hamarosan meg is történt. A munkát Bill Jolitznak köszönhetjük, aki ezután ki is adott egy teljes értékű, bootolható rendszert, a 386-os PC-kre.
Ez volt a 386/BSD. A rendszert FTP-ről bárki letölthette.
Mivel Jolitznak nem volt túl sok ideje a rendszerrel foglalkozni, egy 386/BSD felhasználókból álló csoport vette át az ügyeket, akik létrehozták a NetBSD kiadást. Ezt ők fejlesztették és tartották karban. Ennek a terjesztésnek a célközönsége, a technikai beállítottságú felhasználók voltak. A csoport célja, hogy annyi platformot támogasson, amennyit csak lehetséges. 1998-ig nem volt disztribúciós média, a rendszert Interneten keresztül lehetett elérni.

A már említett BSD-szaporodás megkezdődött. Ezután néhány hónappal létrejött a FreeBSD csoport. A PC architektúrát támogatták, és a kevésbé hardcore felhasználókat célozták meg.
Ennek megfelelően telepítő scriptekkel látták el a rendszert, és azt CD-ROM-on terjesztették.
A könnyű telepíthetőség, és a széleskörű promóció következményeként, ma a Release 2 alapú rendszerek közül nekik van a legnagyobb felhasználói bázisuk.
Itt már kissé összefonódnak a dolgok (a párhuzamos idősíkokat egy, az IMDB-n 0.78 verziószámot elért amerikai film összezavarta).
Így bejött a képbe egy olyan szereplő, akiről csak később lesz bővebben szó.
A FreeBSD beépített Linux emulátorral rendelkezik, így a rendszeren futtathatóak a nagy számban elérhető Linux binárisok.
A BSD tovább szaporodott. Az 1990-es évek közepén a NetBSD-ből kivált egy csoport, és létrehozta az OpenBSD-t. Céljuk a rendszer biztonságának növelése, valamint a könnyű használhatóság, és a széles körű hozzáférhetőség volt.
Közben az egyetem háza táján sem állt meg az élet, létrejött egy BSDI (Berkeley Software Design Incorporated) nevezetű cég is, abból a célból, hogy támogatást és kereskedelmi fejlesztéseket adjanak a kódhoz. Hozzáadták a rendszerhez a Bill Jolitz-féle 6 állományt, és 1992-ben elkezdték árusítani a rendszert forrással, binárisokkal együtt, kicsivel 1000 dollár alatt.
Nem sokkal később az USL (Unix System Laboratories) beperelte őket (a per lefolyását nem részletezném, megtalálható a HUP-on).
Az egész folyamat 1994 elején egy egyezséggel ért véget. Ez alapján a rendszerből 3 file-t el kellett távolítani, és néhány kisebb változtatást kellett végrehajtani más állományokon.
Az egész következményeképp két BSD jelent meg 1994-ben, a 4.4BSD-Lite nevű, a Networking Release-eknél már lemlített feltételekkel használható, és egy olyan teljes rendszer, amihez meg kellett venni az USL forrás licencet is. Ennek 4.4BSD-Encumbered lett a neve.
A per végi megegyezés kikötötte, hogy az USL senkit sem perelhet be a 4.4 BSD Lite használatáért. Ezért az újabb BSD csoportok is ezen a rendszer alapján írták újra a saját kódjukat, és beleolvasztották saját bővítéseiket, javításaikat. 1995 nyarán kiadásra került a 4.4BSD-Lite, Release 2. Ezután a CSRG feloszlott. A központosított fejlesztési modell helyett azt a módszert támogatták, hogy a forráskód alapján a különböző csoportok, más-más céllal fejlesszék a rendszert, és döntsenek a felhasználók, melyiket is szeretnék valójában. Ez végül be is következett. Ma az elérhető BSD-k száma az itt említettekénél jóval több. A BSD története során említésre került a Linux. Mielőtt részletesebben taglalnánk a témakört, meg kell említeni egy korábbi kezdeményezést, de majd csak a következő részben.

Szőke József

A cikkhez tartozó fórum címe:
http://www.flosszine.org/hogyan_kezdodott2

Tudtad-e?

Az Open Source as Alternative (http://www.osalt.com/) honlap kategóriákba rendezett, jól áttekinthető, párhuzamos kínálatot ad a kommersz és a nyílt forráskódú alkalmazásokból. A webhelyről Unix/Linux, Windows, MacOS és Java platformokon használható nyílt szoftverek tömegeinek letöltése indítható el, míg a pénzért árult szoftverek oldalaira linkek mutatnak.

OPEN SOURCE FARM

2008 októberétől jövő év márciusig tart az a nyílt forráskódú szoftverekről szóló előadássorozat, amely a bárki számára ingyenesen elérhető számítógépes technológia népszerűsítését szolgálja. A budapesti Millenárison a nemzetközi és a hazai fejlesztőcégek legnagyobbjai, illetve a legaktívabb linuxos fejlesztőközösségek képviselői ismertetik megoldásaikat.

A rendezvénysorozat egyik csúcspontja eddig a december 11-ei előadónap volt, amelyen a nyílt forráskód világa felé kacsintgató Microsoft hazai képviselője, Keszei Balázs is részt vett (lásd még részben vele készült interjúnkat is!). Fő mondanivalóját talán ez a mondata foglalja össze a legtömörebben: "A Microsoft nem a nyílt forráskóddal, hanem termékekkel versenyez." Az ő előadását követő, „zárt és nyílt” csapatok közötti pódiumbeszélgetés pedig igazi csemegének ígérkezett, tekintettel arra, hogy nyilvánosan ilyen vitára Magyarországon eddig még nem került sor. A szabad szoftvereket fejlesztő és forgalmazó vállalkozások hazai vezetői Keszei Balázsnak leginkább a MS nyílt forráskódú szabadalmakhoz való viszonyát, és a gazdasági erőfölényével való visszaélését hányták a szemére, amely szerintük a világcég licencelési gyakorlatában mutatkozik meg a leginkább szembetűnően. A sorozat másik csúcspontja az Open Source Farm(1) (OSF), vagy legalább is annak az első évadját lezáró előadónap lesz jövő év márciusában, amikor is a hackerlegenda, Richard M. Stallman(2) személyesen ad majd elő. A két dátum között – ahogyan a korábbi előadásokon is – többek között az openSuse-t támogató Novell, a Fedorát támogató Red Hat, a sok platformra naplózó alkalmazást fejlesztő Balabit, és a számtalan nyílt projektet kezdeményező és támogató Sun Microsystems vezető szakembereitől lehet tanulni.

Az Open Document Format Alliance(3) fővédnökségével megvalósuló projekt célja a közintézményi és a vállalati felhasználókon túl az egyéni számítógép használók meggyőzése a nyílt forráskódú lehetőségek kiaknázásáról. A szövetség és az OSF-szervezők szerint a pénzügyi válság egyre inkább arra ösztönözheti az európai döntéshozókat, hogy egy megtakarítási koncepció keretében akár az egész Európai Unió te-

rületén kötelezővé tegyék a nyílt szabványok alkalmazását. Az ötlet azonban nem új, az elmúlt években a tagországok számos közintézménye - már a válság nélkül is – a nyílt alapokon működő információtechnológiai rendszerek kialakítása mellett tette le voksát. Ha csak a francia postát, a spanyol iskolahálózatot, a német külügyi intézményeket tekintjük, számos nagy hálózattal találkozhatunk, amely Linux-alapú környezetben biztonságosan teljesít.

Az októberi és a novemberi előadónapok ugyancsak ezt szolgálták, amikor olyan példaprojekteket is közelebbről megismerhetett a közönség, mint Szeged, illetve Törökbálint önkormányzatának átállási tapasztalatai a nyílt forráskódú rendszerekre. De az Open Source jogi hátteréről, a vele kapcsolatos uniós tapasztalatokról, a nyílt irodai alkalmazásokról és a böngészőfejlesztésről is hasznos ismeretekkel gazdagodhatott a hallgatóság.

A november 20-i előadássorozat a Green Computingról szólt, vagyis a takarékos és környezettudatos számítógépfelhasználásról, ami a hardver- és szoftverfejlesztőkre éppúgy feladatot ró, mint a felhasználókra, legyen szó akár otthoni gépekről, vagy több száz fős cégekről. Porkoláb Dániel és Fischer Erik energiatakarékossági tesztkísérletek ismertetésén és a Sun Microsystems által nyújtott megoldásokon keresztül engedett betekintést abba, miként jelenthet egyszerre anyagi megtakarítást és felelős gondolkodást a vékony kliensek használata. Ezek után a Balabit-es Höltzl Péter beszélt a kis-, és középvállalatok által is egyszerűen kivitelezhető, olyan, a gyakorlatban már bevált zöld módszerekről, mint a virtualizáció támogatása, vagy a géppark beszerzése során figyelembe vehető takarékossági szempontok. Az ULX/Red Hat-tól érkező Dr. Szántó Iván, majd Dr. Szentiványi Gábor prezentációja révén a jövő környezetbarát rendszereinek technológiai részleteiben, a Cloud Computing(4) rejtelmeiben, illetve a számítógép-használat „közművesítésének” kérdéseiben merülhetett el a hallgatóság.

A magyarországi informatikai környezetváltást az OSF az előadások mellett úgy is támogatja, hogy november végén fejlesztői versenyt írt ki(5). A pályamunkákat két kategóriában várják 2009. február 10-ig.

Jankovich Oszkár – Pfeiffer Szilárd

Hivatkozások:
(1)http://www.osf.hu/
(2)http://hu.wikipedia.org/wiki/Richard_Matthew_Stallman
(3)http://www.odfalliance.org/
(4)http://hu.wikipedia.org/wiki/Sz%C3%A1m%C3%ADt%C3%A1si_-felh%C5%91
(5)http://www.osf.hu/verseny

A cikkhez tartozó fórum címe:
http://www.flosszine.org/osf

Tudtad-e?

Az OpenThesaurus nevű, német nyelvű, szabad szinonimaszótár (http://www.openthesaurus.de/) szócikkeinek száma 2008 november 23-án meghaladta az 50 ezret. A szótár internetes felületen és alkalmazásokon belül egyaránt használható. Állományát a saját bejegyzéseivel bárki gyarapíthatja, illetve javíthatja, aki regisztráltatja magát. Az OT adatbázisa LGPL licenc alatt áll, de a PHP/MySQL-re alapuló webalkalmazás maga GPL alatt használható, és a 2.0.3-as verzió óta már az OpenOffice.org irodai alkalmazáscsomagba is beépíthető.

A MAGYAR FREEBSD DOKUMENTÁCIÓS PROJEKT

A szabad szoftver mozgalom hajnalán csak egy szűk kör használt számítógépeket, amelyek hatalmas, szekrény méretű óriások voltak és hardverelemeik gyakran többe kerültek, mint az akkori menő sportkocsik. Nem csoda hát, hogy ebben a korban nem fektettek nagy hangsúlyt a szoftverek könnyű használatára, hiszen minden felhasználó szakavatott hacker volt. Az idők azonban változtak és nagyobb körnek lett szüksége arra, hogy eligazodjon a számítógépes világban. Ehhez pedig könnyen telepíthető és használható szoftverek kellettek, illetve megjelent a megfelelő dokumentációra való igény is. Mára addig is eljutottunk, hogy a dokumentáció sok szabad szoftverhez több nyelven is elérhető. Magyar viszonylatban is jelentősek a projektekben való részvételek és a honosítási erőfeszítések is. Most a FreeBSD Magyar Dokumentációs Projekt munkájába fogunk röviden betekinteni.

A FreeBSD dokumentáció felépítése

Először tekintsük át, hogyan is épül fel a FreeBSD dokumentációs rendszere. Magát a FreeBSD-t három nagy csoport, három nagy repóban fejleszti. Ezek a területek a következők:

src - Az alaprendszer. Ez a repó nemrég tért át a Subversion verziókezelő rendszerre, a másik két alprojekt továbbra is CVS-ben van.

ports - Az alaprendszerre opcionálisan, igény szerint telepíthető csomagok.

doc/www - A dokumentáció.

A dokumentációt három nagy részre oszthatjuk fel: man oldalak, weblap, könyvek, és cikkek. A man oldalak az alaprendszer részét képezik, tehát az src repóban találhatóak és az alaprendszerrel együtt telepítődnek. Röviden szólva egy-egy parancs, rendszerhívás vagy alrendszer működését dokumentálják. Rengeteg man oldal van, és ezek a rendszer adott részeiről külön kérhetőek le, tehát eléggé elszórtan hordozzák az információkat, ugyanakkor ezek adják a rendszer legrészletesebb és legpontosabb dokumentációját.

A weblap a doc/www repón belül található és általános információkat közöl a FreeBSD fontosabb adottságairól, fejlesztési menetéről, letöltéséről, stb. A weblapról érhetőek el a könyvek és cikkek is. A könyvekre jellemző, hogy nagyobb terjedelműek és egy általánosabb témát fednek le. Például a „FreeBSD Handbook” a rendszer általános használatát mutatja be az egyszerű felhasználói tevékenységektől kezdve a szerverként történő felhasználásig. A könyvek között található továbbá a fent említett három nagy területet leíró három könyv is. Ezek jó kiindulási alapot nyújtanak abban az esetben, ha szeretnénk a projekt valamelyik területének fejlesztésében részt venni. A cikkek rövidebbek és egy-egy specifikusabb témát tárgyalnak, ilyen például a naplózó fájlrendszer beállítása, vagy a Compiz Fusion telepítése és konfigurálása FreeBSD-n.

A technológiai háttér

Most a weblapra, illetve a könyvekre és cikkekre fogunk koncentrálni, hiszen az átlag felhasználó ezekkel kerül először kapcsolatba és jelenleg ezek egy részének létezik csak magyar fordítása. (A man oldalak fordítása több szempontból is összetettebb és körülményesebb, de nem zárkózunk el ezeknek a későbbi lefordításától sem.) A dokumentáció strukturálásához és formázásához az SGML leíró nyelvet használják. Az SGML tulajdonképpen egy olyan nyelv, amellyel feladatokra specializált leíró nyelveket hozhatunk létre. Az újabb XML technológia ismerői számára érdekes lehet, hogy az SGML az XML elődje és annál bővebb lehetőségekkel rendelkezik, de ha úgy gondolunk rá, mint az XML nyelvre, nem állunk

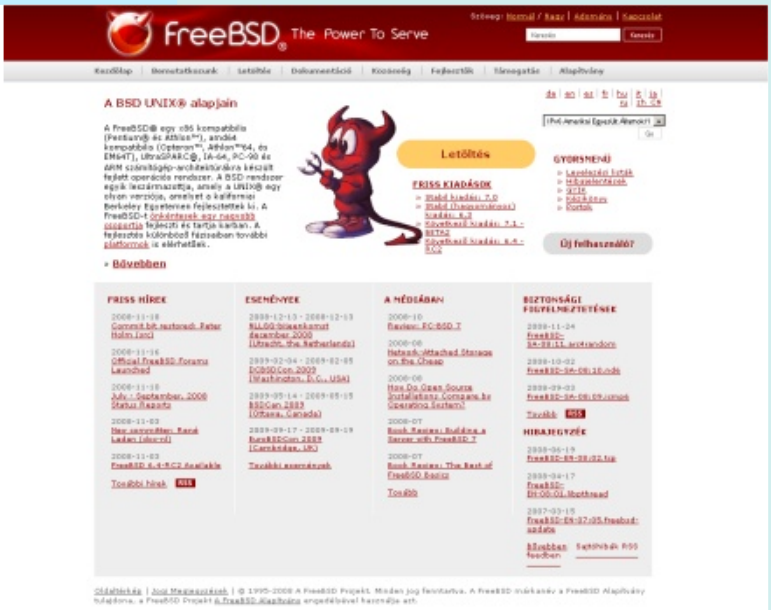
messze az igazságtól. A HTML ismerői számára pedig megjegyezzük, hogy a HTML is egy SGML nyelven definiált jelölőnyelv, így a HTML tartalom feldolgozásához használhatjuk az SGML-hez írt eszközöket.

A weblap esetén éppen ezt a lehetőséget használjuk ki, ugyanis a weblapok HTML-ben vannak strukturálva és az SGML feldolgozóprogramokkal állítjuk elő őket. Így lehetőségünk van például saját entitások definiálására és felhasználására. Például a weblap menürendszere is entitásként definiált, így könnyen beilleszthető minden oldal tetejére, illetve az aktuális FreeBSD verzió száma is entitás formájában tárolt, így egy új kiadás megjelenésekor csak egy helyen kell megváltoztatnunk a verziószámot.

A könyvek és cikkek nem HTML-ben, hanem egy másik SGML-ben leírt nyelven, a DocBook nyelven íródtak. A DocBook egy olyan leíró-nyelv, amelyet kimondottan műszaki dokumentációk írásához hoztak létre és rengeteg hasznos jelölőelemet tartalmaz, például egy fájlnevet a <filename>/etc/rc.conf</filename> formában írhatunk le. Ennek nagy előnye, hogy egy fájlnév leírásakor nem kell elgondolkoznunk azon, hogy milyen formátummal is szedtük a fájlneveket egészen odáig, hanem egyszerűen csak használjuk ezt a jelölőelemet, a kimeneti dokumentumok generálásakor pedig minden fájlnév a megadott stílusnak megfelelően fog generálódni. Ez természetesen a stílusok könnyű megváltoztatására is lehetőséget ad, hiszen csak a stíluslapot kell megváltoztatnunk, és a kimenetben minden ilyen fájlnévként jelölt szöveg az új stílusban fog megjelenni a következő generáláskor.

A magyar fordítások

Ez a struktúra áll tehát rendelkezésünkre, ebből kell kiindulnunk a magyar fordítások elkészítésekor is. A FreeBSD Projekt a már említett három fő repón kívül a fejlesztők rendelkezésére bocsájt egy negyedik, általános célú repót is, ahol a fő repókból elágazások hozhatóak létre egy-egy folyamatban lévő fejlesztés kényelmes menedzselésére. Ez a repó a Perforce verziókezelő rendszer által kezelt, ami első hallásra meglepő lehet, hiszen a Perforce egy kereskedelmi termék, de nyílt forrású fejlesztésekhez a rendszert fejlesztő cég a projektek rendelkezésére bocsájt egy ajándék licencet. A Perforce erőforrás-takarékos branch kezelésének köszönhetően nagyon jól használható az efféle fejlesztésekre. Az src repó SVN-re váltásával az SVN is használható fejlesztői elágazások létrehozására, de a ports és src repók még mindig CVS-ben vannak, amely nem ideális ilyen célokra.



Eleinte a fordításokat elkülönítve végeztük, de később rájöttünk, hogy akár az említett Perforce repóban is létrehozhatnánk egy elágazást a magyar fordításoknak, megkönnyítve ezzel a fordítók közötti munkamegosztást. [1] Eddig a más nyelvekre fordítók teljesen külön dolgoztak, most azonban követte példánkat a spanyol, majd a holland fordítói csapat is, így két újabb elágazás született a Perforce repóban.

Fontos feladat volt számunkra az eredeti dokumentumokkal kapcsolatos állapot számontartása, hogy valamilyen módon le tudjunk kérni egy listát a frissítendő dokumentumok állapotáról és a fordításainkat szinkronban tartsuk az eredeti angol verziókkal. A görög csapattal együttműködve kidolgoztunk egy módszert, amely szerint a fordított dokumentumok eredeti verziójának revíziószámát, illetve az eredeti verzió elérési útvonalát a fordított fájlokban egy fejlécben megjelöljük és utána egy script automatikusan képes egy listát generálni azokról a fájlokról, amelyek időközben aktualitásukat vesztették. Módszerünket a spanyol csapat is átvette.

Ezen kívül időközben létrehoztunk egy wiki oldalt is [3], ahol a projekt állása követhető nyo-

mon, illetve ide a jövőben újabb hasznos információkat is fel fogunk vinni.

Ami a magyar fordítások állását illeti, jelenleg le van fordítva a weblap egy része [2], a „FreeBSD kézikönyv", a „FreeBSD GYIK" és a számos cikk közül tíz olyan, amely különösen fontos, vagy érdekes témát ölel fel [4]. A weblapot illetően rendelkezünk a legfontosabb részek fordításával, ha olyan linkre kattintunk, ami egy lefordítatlan oldalra vezet, akkor az oldal angol változatára jutunk. Később, amennyiben igény merül fel rá, és erőforrásokkal is rendelkezünk, akkor a weblap fordítása ebből az állapotból könnyen kibővíthető újabb részek fordításával. Legfontosabb eredményünk egyértelműen a „FreeBSD Kézikönyv", amelynek fordítása teljes és naprakész. A „FreeBSD GYIK" fordításával egy időben egy nagy aktualizálás is végbement, ugyanis az eredeti könyv tartalmában is szerepelt sok elavult rész. A cikkfordítások közül olyan témákat igyekeztünk kiemelni, amelyek különösen érdekesek, vagy fontosak lehetnek, így rendelkezünk egy bevezető jellegű cikkről a BSD rendszerekről, egy Linux-BSD összehasonlító cikkel, egy Linux felhasználóknak szánt gyors bevezetővel, illetve cikkekkel Compiz Fusion, CUPS, betárcsázós hálózat beállítása, fájlrendszer naplózás használata, laptopokon való használat, más operációs rendszer mellé történő telepítés, illetve a megfelelő verzió kiválasztása témákban.

Végezetül a fordítói csapatról szeretnénk pár szót írni. Jelenleg két taggal rendelkezik a projekt, és mindkettőnk egyben FreeBSD committer is. Jómagam kezdtem el a fordítási projektet a weblap, majd pár cikk fordításával, közben pedig a megfelelő fórumokban sűrűn hirdettem a munkámat. Néha-néha akadt egy-egy segítő, aki kisebb fordításokkal besegített, illetve átnézett fordításokat és javaslatokat küldött, de Páli Gábor <pgj@FreeBSD.org> megjelenéséig nem bővült állandó taggal a csapatunk. Gábor nagy erőkkel dolgozott az új fordítások elkészítésén és a „FreeBSD Kézikönyv" fordítását is egyedül ő vitte véghez, mivel én az utóbbi időben kevés szabadidővel rendelkezem az egyetemi tanulmányaim miatt. Gábor remek munkát végzett és a fent említett script elkészítésében is aktívan részt vett, így jelenleg már ketten vagyunk.

Jövőbeli kilátások

Úgy gondoljuk, hogy a csekély létszám ellenére nagy eredményeket értünk el és a projekt a többi fordítás közt is remekül megállja a helyét, mind naprakészségben, mind minőségben, mind pedig a lefordított dokumentumok számát illetően. Segítség azonban mindig jól jön, ezért itt most egyben fel is szeretnénk hívni a figyelmeteket erre a projektre. Úgy gondoljuk, hogy a fordítások nagyban segítenek abban, hogy egy tágabb réteg ismerje meg ezt a remek operációs rendszert, hiszen sokan vannak, akik kevésbé tudnak angolul, de próbálgattak már Linux rendszereket és így szívesen kipróbálnák a FreeBSD-t is, magyar dokumentáció híján azonban nem mernek belevágni. Másrészt egy fordító rengeteg hasznos tapasztalattal lehet gazdagabb. Megtanulhatja a projektben alkalmazott technológiákat (SGML, XML, HTML, DocBook, DSSSL, XSLT, CSS), fejlesztheti nyelvi képességeit és tagja lehet egy remek közösségnek, ahol saját bőrén élheti meg a szabad szoftverek fejlesztésében való részvétel élményeit.

A kedvet érzők kérem vegyék fel velem a kapcsolatot e-mailen <gabor@FreeBSD.org>, ahol megbeszéljük a továbbiakat, és keresünk egy érdekes dokumentumot, amelynek még nem készült fordítása és jó alap lehet a kezdéshez.

Természetesen nem csak fordításokat várunk, minden visszajelzést, hibajelentést megválaszolunk és nagyra értékeljük ezeket a segítségeket is.

Hasznos linkek:

http://perforce.freebsd.org/depotTreeBrowser.cgi?FSPC=//depot/projects/docproj_-hu&HIDEDEL=NO
A Perforce repó magyar elágazása [1]

http://www.freebsd.org/hu/
– A magyar honlap [2]

http://wiki.freebsd.org/DocTranslationProjects
– Wiki oldal a fordításról

http://wiki.freebsd.org/HungarianDocumentationProject
– Wiki oldal a magyar fordításról [3]

http://www.freebsd.org/doc/hu_HU.ISO8859-2/
– Magyarra lefordított könyvek és cikkek [4]

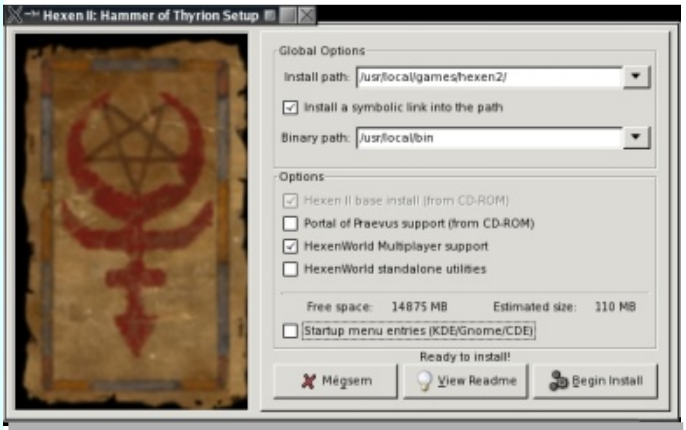
Kövesdán Gábor <gabor@FreeBSD.org>

A cikkhez tartozó fórum címe:
http://www.flosszine.org/magyar_freebsd_dokumentacios_projekt

HEXEN 2 / HAMMER OF THYRION

Ha akad köztetek játékos, aki szereti az ínyenc FPS projekteket, miközben már 10-15 évvel ezelőtt is kereste az önfeledt kikapcsolódást, akkor biztosan emlékszik a Hexen 2-re. Nos, ez a kimagasló alkotás semmiképp sem nevezhető sablonosnak, ezen felül naprakész linuxos támogatással rendelkezik. Amondó vagyok, ne hagyjuk veszni!

A Hexen sorozat második része bő tíz évvel ezelőtt látta meg a napvilágot. Fejlesztői munkája azt a Raven csapatot dícsérte, melynek neve később olyan kasszasikereket fémjelzett, mint a Soldier of Fortune játékok, a Jedi Knight epizódjai, vagy épp a Quake negyedik része. A címben szereplő projekt azonban legkevésbé sem „tömegcikk", erősen kilóg a sorból: a belső nézetű lövöldözős jelleget próbálta ötvözni a szerepjátékok stílusjegyeivel. Emiatt a felhasználói rétegét kezdetben nagyon szerénynek jósolták, a várakozásokkal szemben azonban súlyos tízezrek hódoltak a mesterműnek. Maga a műfaj pedig jóval később, a nagynevű Wheel of Time klasszikusban és kortársaiban forrt ki teljesen, majd eresztett stabil gyökereket.



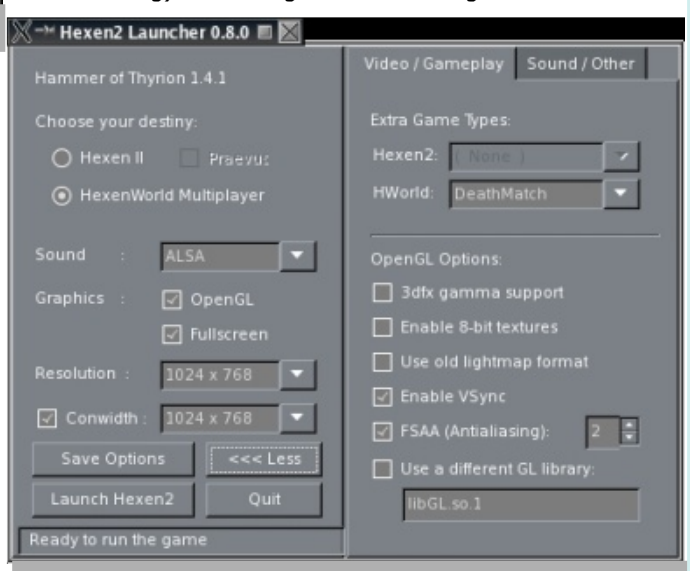
1.ábra Íme a linuxos, Loki alapú telepítő...

Mit tudsz, öreg Hexen 2?

A szoftver világát a Quake finomított motorja generálja, ennek ellenére élvezhető Fantasy környezetet sikerült modellezni vele. Látvány terén a később célegyenesbe ért Quake 2 alapkódjának néhány képességét is implementálták a programsorokba, mai szemmel nézve azonban borzasztóan szögletesre sikerült az egyébként érdekes terep. Természetesen a fizika sem mondható tökéletesnek, (cserébe viszont) sőt, a használatnak nem lételeme az erős központi egység, de még a 3D grafikus hardver sem (a valódi OpenGL 3D mód csupán opció). A hangok és az effektek szintén átlagosak - ellenben a zenei kíséretet semmilyen kritika sem érheti. Nos, ennyit a száraz tényekről. Lássuk inkább, mit rejt a belbecs!

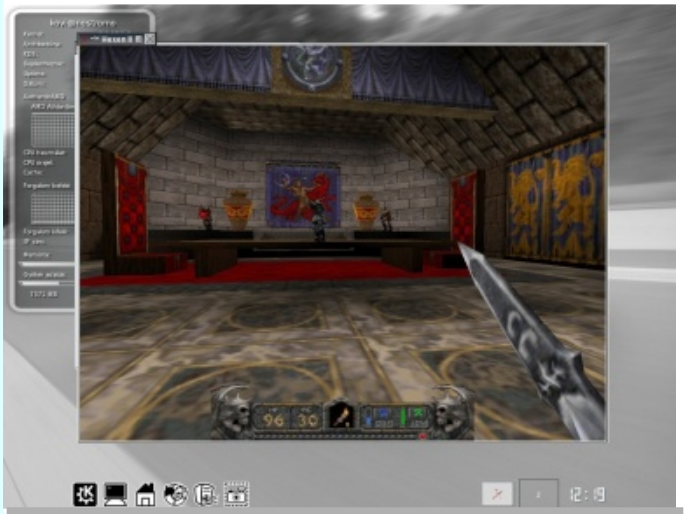
A történet szerint Thyrion földjére ismét rossz idők járnak: négy árnyék-lovas áll a határban, akik háborút, halált, éhezést és járványokat hoznak magukkal. Ez bizony nem hangzik túl jól, de a bajt tetőzve Eidolon, a legfőbb gonosz erő birtokosa is szeretné kivenni részét a pusztításból. Nos, ezt az „ötösfogatot" kellene megállítani valahogyan... (Esélyem sincs olyan mesterien bevezetni a történéseket, mint ahogyan a fejlesztők ezt megtették annak idején. Érdemes megtekinteni a „mesekönyvet": a Hexen 2 telepítő CD-jén található olvasnivalót kísérő ART grafikák sem mindennapiak. Ezt a verziót a Raven hivatalos lapján keresve sem találni.)

A program világa saját stílusához és történetéhez megfelelő kalandorokat kíván: paladdin, lovag, nekromanta vagy bérgyilkos bőrében lehet irtani az ellenséget. Mindegyik kaszt más és más képességekkel bír, különböző fegyverekkel harcol - valamint éltérő módon használja a különféle mannákat (a csupasz fizikai küzdelem mellett szerephez jutnak a varázslatok is). Mindet azért fontos, mert barátságos lényeket nem igazán találni egyik terepen sem: a legelésző bárányokon kívül gyakorlatilag minden mozgó modell ellen-



2.ábra ...és a Launcher alkalmazás.

ségként kezelendő. Velük kapcsolatban az öreg mondás a legtalálóbb, miszerint „a legjobb védekezés a támadás”. Tehát, ha a játékos nem ront rá időben az íjászokra, gólemekre, varázslókra (vagy éppen angyalokra, démonokra, impekre), akkor ők fogják megtenni az első és fájdalmasnak ígérkező lépést... Egyebek tekintetében mindig a klasszikus elgondolás dominál: adott helyről úgy tudunk továbbjutni, hogy egy távoli ponton lévő tárgyat birtokba veszünk. Példaként egy kulcsot, urnát, vagy akár egy varázserőt adó könyvet megkeresve léphetünk be a tárgyhoz kapcsolódó



3.ábra Kaland indul, KDE asztalon.

kastélyba, ahol a következő pályák zálogaként újabb ereklyék után kell kutatnunk. Egy terepről jellemzően három-négy további pálya is nyílik, melyek között mindkét irányban szabad az átjárás. Ebből következik, hogy komoly tájékozódási képességek hiányában az öreg Hexen 2 nem élvezhető. Már pedig kár lenne lemaradni erről a kalandról: a mellékelt néhány kép alapján (remélem) tetten érhető a különc és sajátos belső világ. Igazából csak két dolog lehet zavaró. Először is a mesterséges intelligencia egyáltalán nem áll a helyzete magaslatán (például a háttal álló ellenfelek mögött észrevétlenül el lehet sétálni). Másodszor pedig, az aktuálisan kiemelt tárgy szinte mindig olyan helyen pihen, ami más játékokban megszégyenítené a „rosszindulatúan” eldugott titkos pályarészeket is (kirívó példaként, egy szőnyeg alá rejtett pince lejáró megkeresése akár órákat is elvehet a hasznos időből). Segítségképpen érdemes lehet elolvasni a [[=>]]http://www.loogoos.hu/leirasok/hexenii.html címen található leírást, ahol az oldal készítője részletesen elmeséli a végigjátszás módját.

Linuxon...

A Hexen 2-nek hivatalosan nincs linuxos binárisa. Legalábbis a fejlesztő Raven nem nyújt ilyen támogatást részünkre. Ennek megfelelően külsős porterek munkájában, a Hammer of Thyrion-ban kell keresni a natív megoldást (erre a névre keresztelték a Szabad Rendszer projektjét). Fontos, hogy az átültetett mű „csak” az átírt alapkódot jelenti, így az adatcsomagok miatt rendelkeznünk kell a játék Win32 platformra készített telepítő lemezével is. A bináris beszerzése ügyén látogassunk el a [[=>]] http://uhexen2.sourceforge.net oldalra és töltsük le a Szabadon elérhető, hexen2_installer_verzió.run formájú telepítőt. Fűzzük rendszerünkbe a windowsos Hexen2 lemezt, majd (a Loki alapú megoldásokhoz hűen) futtassuk le az előbb letöltött fájlt, root jogokkal. Miután az adatokkal együtt minden állomány a helyére került (/usr/local/games/hexen2/), felhasználóként kiadott hexen2 illetve glhexen2 paranccsal léphetünk a játékba, kliens módban (előbbi kötés a szoftveres képalkotáshoz tartozik, utóbbi pedig a 3D-s OpenGL leképezéshez). A HexenWorld alapú többjátékos rész (gl)hwcl parancsra érhető el, a dedikált szerver üzemmódot pedig a h2ded indítja.

Ha valaki nem rendelkezik a teljes játékkal, akkor a Hexen 2 demó adatcsomagjait is felhasználhatja: ekkor a *.tgz formába csomagolt játékmotort le kell töltenie a honlapról, majd a próbaverziós műremek pak0.pak és pak1.pak állományait „kézzel” kell bemásolnia a kicsomagolt tarball megfelelő /data1 almappájába.

4.ábra Nem szerencsés egy Nekromantával ellenkezni...

Hibalehetőségek, apróságok

Mivel a mai Linux rendszerekben egyre kevésbé támogatott a hangképzés OSS megoldása, így az ALSA használatáért egy apró trükkhöz kell folyamodni. Esetemben, a (gl)hexen2 -sndalsa -alsadev hw:0,0 parancsot kell kiadnom a hibátlan hangkezelésért (ezen felül az SDL keresztplatform hangkönyvtára is szóba jöhet, mely -sndsdl opcióval kérhető). Az optikai meghajtó kezelése is nyűgös: ha valakinek supermount mechanizmus fűzi rendszerébe a CD lemezeket, akkor a játék zenei sávjai alapból nem hallhatóak. Ilyenkor egy további paramétert kell az in-

dításhoz fűzni, a következők szerint: -cddev /dev/eszköznév.

5.ábra Monumentális helyszínek.

6.ábra A bérgyilkos is veszedelmes.

7.ábra Így készül a mirelit-mágus :)

Aki idegenkedik ezektől a konzolos „munkáktól”, annak fontos lehet: a Hammer of Thyrion némelyik friss verziója rendelkezik egy h2launcher nevű indítóval, amivel a legfőbb képi és hangleképezési módokat grafikusan lehet aktiválni (valamint ennek segítségével az eredeti játék Expansion lemezét is könnyen életre lehet hívni).

Végül a kód igényeit kell említenem. Tapasztalataim szerint ebben a fantázia-világban egy szerény 400MHz órajelű x86 központi egységgel, 128 Mbyte memória társaságában már kompromisszumoktól mentesen lehet kalandozni. Ha a felhasználó ezen felül rendelkezik egy egyszerű, GLX vagy DRI kapoccsal ellátott 3D grafikus kártyával, akkor az OpenGL leképezés által valamivel emészthetőbbé teheti a 10 éves megoldásokat (a mellékelt képek ilyen módban készültek). További jó hír, hogy a teljes telepítés éppen csak meghaladja a 100 Mbyte területet... Az elvárásokat szoftveres oldalról megközelítve csupán a friss GLIBC és SDL könyvtárak jelenlétére kell ügyelni. Gyakorlatilag tehát minden adott a kikapcsolódáshoz: akár egy nyolc-tíz éves masina is megfelel a célnak, naprakész Linux terjesztést futtatva..

Zárásképpen egy figyelmeztetést tolmácsolnék a felhasználók felé! A Hammer of Thyrion készítői óvnak a túlhajtott gépektől, agresszív ROM BIOS beállításoktól: elmondásuk szerint az átirat ilyen környezetben könnyen összeomlik.

Kovács Zsolt

A cikkhez tartozó fórum címe:
http://www.flosszine.org/hexen2_hot

Tudtad-e?

Az Ubuntu elindította a második Szabad Kultúra Szemlét (http://ubuntu.hu/hirek/2008nov/ii-ubuntu-szabad-kultura-szemle). A nép-szerű Linux-disztribúció gazdái 2009. február 6-ig várják azokat a hang-, kép- és filmalkotásokat, amelyek legjobbjait beemelik majd az áprilisban várható, Jaunty Jackalope nevű, új Ubuntu-kiadásba.

AZ OPERÁCIÓS RENDSZER MINT KUTYA

Interjú Kürti Lászlóval, az Open Source Farm főszervezőjével, az Open SKM Agency ügyvezető igazgatójával

Nyílt forráskód, innováció, technológia, üzleti megoldások – ezekkel a szlogenekkel csábítja az Open Sorce Farm nevű kezdeményezés újabban az érdeklődőket havi rendszerességgel a Millenárisba. A jövő tavaszig tartó előadássorozatról a harmadik előadói napon beszélgettünk az Open SKM Agency ügyvezető igazgatójával.

FLOSSzine: Milyenek az eddigi tapasztalataitok?

Kürti László: Nagyon jók. Egyre növekszik az érdeklődés, annak ellenére, hogy mostanában konferencia-boom van. Például hasonló témakörben, mint itt a mai - ez a környezetbarát információtechnológiák, a Green Computing – az elmúlt hónapokban 2-3 hasonló rendezvény is volt. Az Open Source Farm azonban egy hosszabb előadássorozat, amely elméleti és gyakorlati kérdésekkel egyaránt foglalkozik. A korábbi előadói napjaink részben a nyílt forráskóddal foglalkozó aktivisták konferenciái is voltak, azoké, akik túl a technológián egyfajta életmódot vagy fejlesztési módot is látnak a nyílt szoftverekben. A további előadásaink inkább már technológiai jellegűek, jobban koncentrálnak majd bizonyos megoldásokra, megoldás-rendszerekre. Kivétel ez alól, hogy a konferenciasorozat záróakkordjaként az egyik legnagyobb hacker, a világszerte imsert Richard M. Stallmann ad elő a szabad szoftverekről. Ő - szokásához híven - egy amolyan „evangelista" jellegű előadást tart majd, hiszen életműve is azt példázza, hogy a nyílt szoftverek, vagy egyáltalán a tudás megoszthatósága jóval több, mint egy technológiai modell. Ő inkább egy társadalmi rendszerben gondolkodik és magukat a nyílt alkalmazásokat ebbe próbálja belehelyezni.

Fz: Kik képezik az előadássorozat célközönségét?

KL: A kezdeményezés egy elég fura oldalról jött. Az Open SKM a projektnek a szakmai konzultánsa, a megvalósításnak a szervezője, bár a tematika meghatározásában is volt részünk, hiszen az előadókat is mi válogattuk össze. A sorozat ötlete azonban -különös módon- bölcsészektől érkezett, azért is vagyunk itt a Millenárisban. Az alapgondolat az volt, hogy nagyon sok művelt, diplomás ember van, aki mindennapi informatika-felhasználó, de igazán nem lát a dolog mögé. Az elmúlt évek termék-alapú megközelítése miatt pedig a nem-szakmabeliek nagy része úgy képzeli el a számítógépet, mint egy tévét, vagy egy DVD-lejátszót: bekapcsolom, van rajta néhány gomb különböző funkciókkal, azt tudom, és akkor szórakozom vagy éppen dolgozom vele. Miközben kialakult ez a kép az átlagfelhasználókban, ezzel párhuzamosan az informatika egyre jelentősebb helyet foglalt el az életünkben. Nem tudunk úgy megfogni egy telefont, nem tudunk úgy beülni egy autóba, nem tudunk úgy beszállni egy liftbe, nem tudunk úgy venni egy hűtőt, hogy annak ne lenne köze az informatikához. Szóval, az a célunk az egésszel, hogy a gondolkodásmódon változtassunk, hogy a laikusokkal is elhitessük, hogy ma az informatika már sokkal több, mint gombnyomogatás, és ha erről nincsenek ismereteink, akkor felhasználóként is jócskán le fognak maradni. Ezért az egyetemeken túl kifejezetten olyan helyeken hirdet-

jük még az előadásokat, például filmmel, színházzal, vagy irodalommal foglalkozó honlapokon, amelyeket többnyire humán műveltségűek látogatnak. Ők az elsődleges célközönség, mert úgy gondoljuk, hogy számukra sem mindegy, hogy az informatikában bezáródunk-e bizonyos magáncégek kereskedelmi feltételei közé, vagy nyílt szabványokon alapuló, mindenki által megismerhető, nyitott, átlátható keretek között fogjuk az informatikai tudást felhasználni. Azt gondolom, hogy egy nagyon lényeges dolog tisztázni azt, hogy ha valamit megveszek, akkor az az enyém lesz-e, és azt ténylegesen hogyan használhatom. Manapság a bolti szoftverek legtöbbjére igaz, hogy hiába fizetek rengeteg pénzt értük, mégsem rendelkezhetem szabadon felettük. Ha viszont számítógép, hardver lesz az enyém, akkor belerakhatok egy másik memóriát, vagy merevlemezt, egy másik monitort csatlakoztathatok hozzá, azt csinálok vele, amit akarok. Ha akarom, bútortámasztéknak használom. Ezzel szemben, ha egy szoftvert veszek meg, akkor az gyakorlatilag nem is kerül a birtokomba. A zárt forráskódú szoftverekkel ugyanis semmit sem csinálhatok, azon túl, hogy a gyártó által meghatározott jogi keretek között bérelem tőle a felhasználás jogát és fizetek neki a lehetőségért, hogy azokat a sorba rendezett nullákat és egyeseket használhatom.

Fz: A közkeletű bölcsész-aggodalom talán úgy hangzik a szabad operációs rendszerekkel szemben, hogy „a Linux az valamilyen parancssoros program és nem felhasználóbarát”...

KL: Igen, valamikor csupán az volt, ahogy a Dos is az volt. Valamikor én is azzal dolgoztam, Commodor 64-gyel kezdtem, de érdemes tudatosítani, hogy hát már sokkal odébb járunk: a legelterjedtebb szabad rendszerek grafikus felületen is egyszerűen kezelhetőek, úgyhogy az semmilyen nehézséget sem okozhat egy kicsit is intelligens embernek. Ha bárki megfog egy Linux telepítő CD-t, az azon található disztribúció ugyanúgy fölkúszik a gépére, mint egy pénzért árult operációs rendszer, csak igeneket és nemeket kell kattintani közben. Sőt, a szükséges eszközmeghajtók ugyanúgy, akár a telepítés közben, maguktól felkerülnek, a rendszer önállóan felismeri a csatlakozó hardvereszközöket, legyen az videokártya, hálózati kártya, bármi – lényegileg a Linux-változatok nem a megjelenésükben különböznek a legelterjedtebb zárt forráskódú rendszerektől, hanem abban, hogy ez – azon túl, hogy ingyenes – személyesen az enyém. Olyan, mint a kutyám: én megmondom, mit szeretnék csinálni vele, ő megmondja, mit szeretne csinálni velem, kölcsönösen hatunk egymásra, megtanuljuk egymást, együtt élünk. Amilyen szinten akarok hozzá érteni, olyan szinten érthetek is hozzá, bármikor átalakíthatom, nem kell hozzá drága jogokat vásárolnom, ott van a forráskódja, ott van minden lehetőség. De ha nem akarok rajta semmit sem átalakítani, akkor az is működik. Mindez a például a nagyvállalati felhasználók körében már ismert. Ott legtöbbször nem kérdés, hogy milyen szervert működtessenek, egyértelmű, hogy a nyílt forráskódú megoldásoknak van uralkodó szerepük: egyszerűen beállíthatóak, szabadon átalakíthatóak, modulárisak, bármivel bármikor kiegészíthetőek, stabilan működnek, komfortosak és alig képes veszélyeztetni őket bármilyen vírus - vagyis költségkímélőek is egyben. Most az egyéni felhasználókat is meg szeretnénk győzni ugyanerről.

Jankovich Oszkár

A cikkhez tartozó fórum címe:
http://www.flosszine.org/kurti_laszlo_interju

NYIT A MICROSOFT?

E-mailes interjú Keszei Balázs Microsoft Platform tanácsadóval és Kőnig Tiborral, a Microsoft Magyarország főmérnökével

A szeptemberi Ubuntu-konferencia után az Open Source Farm támogatói között is feltűnt a legelterjedtebb zárt forráskódú operációs rendszer gyártója. Sőt, a Microsoft munkatársa előadást is tartott a szabad szoftverek népszerűsítésére hivatott rendezvénysorozaton. A világcég két hazai képviselőjét arról kérdeztük, vállalatuk hogyan viszonyul a nyílt forráskóddal készített programokhoz, illetve a szabad alkalmazások fejlesztőihez.

FLOSSzine: Mely szoftvereinek (alkalmazásainak) a forráskódját tette eddig, és teszi a belátható jövőben nyílttá a Microsoft?

Microsoft: A Microsoft a Shared Source Initiative program keretében már ma is elérhetővé, azaz böngészhetővé, fejlesztéskor vagy hibakereséskor követhetővé teszi a Windows és Windows Embedded operációs rendszerek, valamint az Office alkalmazások forráskódját. További részletek megtalálhatóak a http://www.microsoft.com/resources/sharedsource/productsourceprogram.mspx címen. Elérhető ezeken kívül a Microsoft .NET Framework alkalmazásplatform-technológiák jelentős és egyre növekvő hányada. Erről további információkat itt lehet kapni: http://www.microsoft.com/resources/sharedsource/referencesource.mspx. A dolog jelentőségének megértéséhez érdemes azonban végiggondolni, hogy az operációs rendszerek, az irodai alkalmazások és az alkalmazásplatform a Microsoft szellemi tulajdonának a legfontosabb elemei közé tartoznak.

Slide Keszei Balázs Microsoft Platform stratégia tanácsadó előadásából (sic!)

Fz: A statisztikák szerint az Internet Explorer népszerűsége világszerte csökken, miközben a Mozilla Firefox böngészőé folyamatosan növekszik. Ezen kívül várható, hogy a Google Chrome böngészője is egyre jelentősebb teret hódít. Tervezi-e mindezek tudatában a Microsoft más böngésző (és/vagy fájlkezelő) alapértelmezetté tételét operációs rendszereiben?

MS: Ami a böngészők népszerűségének az alakulását illeti, nem látunk a jövőbe, de abban biztosak vagyunk, hogy az Internet Explorer-használók aránya jelentős marad. Az operációs rendszer felhasználói szolgáltatásai közé tartozik a web böngészése, és az Internet Explorer a következő Windows-verzióban, a Windows 7-ben is rendelkezésre áll majd. A felhasználók természetesen más böngészőt is alapértelmezetté tehetnek a Windowsban.

Fz: A Microsoft Office legújabb, vagy éppen most készülő verziói a jövőben jobban fogják-e támogatni

a nyílt dokumentumformátumokat (ODF: .odt, .ods, .odp)? Illetve a Windows Mediaplayer fogja-e támogatni a nyílt médiaformátumokat (például az Ogg Vorbist)? Ha igen, hogyan?

MS: A Microsoft Office következő, Office „14” kódnevű verziója a nyílt, ISO- és ECMA szabvány Open XML mellett várhatóan támogatni fogja a szintén ISO szabvány Open Document formátumokat is. A Windows Media Player Ogg-támogatása külső kodek használatával megoldható, natív támogatásról egyelőre nem tudunk.

Fz: A hírek szerint a 2010-re elkészülő Windows 7 operációs rendszer már a Cloud Computing jegyében születik és elsődleges cél a fejlesztésnél az internetes alkalmazások igénybevételének, szűkebben a Windows Azure rendszer használatának az ösztönzése. A rajtuk futtatható alkalmazások mennyiben támogatják majd a nyílt fájlformátumok használatát? Hiszen a konkurenciának tekinthető, "többnyire nyílt" Google Apps máris kompatibilis az ODF-el, ahogyan az MS Office Word, illetve az Excel kiterjesztéseivel ugyancsak.

MS: Az Office „14”-ben megjelenő Office Web Applications csomag, ami a felhőben fut majd, képes lesz kezelni az irodai formátumokat. Mindez hasonlít a Google Appshoz, az ugyanis egy irodai programcsomag, ami a Google AppEngine platformon fut.

Fz: Várható-e, hogy a Microsoft újabb, saját, zárt formátumokat hoz forgalomba újabb termékeiben?

MS: Az utóbbi három-négy évben a Microsoft gyakorlatilag valamennyi új fejlesztési projektjében XML alapú formátumot használt a szöveges információk tárolására. Ugyanakkor egy szabványügyi testület gondjaira bíztuk az Open XML irodai dokumentumformátumot, publikáltuk a korábbi Office-verziók bináris formátumainak leírását, a virtuális gépek VHD formátumát és több médiaformátumot.

Fz: Tervezi-e a Microsoft, hogy szoftvereinek a nyílt forráskódú operációs rendszerek alatti futtatását is jobban elősegíti (akár a Wine, vagy a linuxos virtualizációs megoldások támogatásával)?

MS: Ez elsősorban a szoftvergyártók, mint a Xen feladata, a virtualizáció megvalósításához szükséges hardverabsztrakciós réteg (HAL) a Shared Source Initiative használói számára már ma is elérhető. Ami a másik irányt illeti, a virtualizációs környezeteket menedzselő System Center Virtual Machine Manager következő verziójának tervezői most vizsgálják, hogy a Hyper-V és a VMWare mellett a Xen is bekerüljön a kezelt termékek közé.

Fz: A Microsoft milyen okokból kezdte el támogatni a nyílt közösségi projekteket és melyeket támogatja itthon, illetve külföldön?

MS: A Microsoft minden alkalommal az ügyfelek igényeit veszi alapul. Az ügyfelek heterogén hálózatokat használnak, mindenki saját maga dönti el, hogy az adott feladathoz milyen platformot választ. Az iparági együttműködések kialakítása az elmúlt 5 évben lett kiemelten fontos a Microsoft számára, ekkor érett be több olyan kezdeményezés, ami támogatja ezt. A Microsoft globálisan támogatja többek között az Apache, a PHP, és a mySQL fejlesztéseket, hogy minél jobban tudjanak együttműködni a Microsoft rendszerekkel. További információ itt érhető el: http://port25.technet.com/

Fz: Célja-e a Microsoftnak, hogy az önkéntes fejlesztőközösségeket bevonva fejlessze tovább a saját termékeit? Ha igen, melyeket szeretné így fejleszteni?

MS: Ennek a fejlesztési megközelítésének a terepe jelenleg a CodePlex (http://www.codeplex.com), illetve több termék közösségi honosítása a kisebb népek által beszélt nyelvekre.

Fz: Célja-e a Microsoftnak, hogy terméktámogatási rendszerének kialakításába jobban bevonja az önkéntes fejlesztőközösségeket? Ha igen, hogyan?

MS: Ez már ma is így történik, a TechNet (üzemeltetői) és MSDN (fejlesztői) portálokon elérhető tartalom szabadon kiegészíthető közösségi információkkal. Mindkét portál nagy forgalmú fórumokat üzemeltet, ahol szabadon megoszthatók a bevált gyakorlatok, a tapasztalatok.

Fz: Célja-e a Microsoftnak, hogy a szabad forráskódú megoldások közül a termékfejlesztései során ezután többet felhasználjon? Ha igen, melyeket? Lehetséges-e például, hogy kernelfejlesztéshez is felhasznál a jövőben Open Source megoldásokat?

MS: Ez csak akkor képzelhető el, ha a szabad forráskódú

megoldásokban használt licenckonstrukciók harmonizálhatók a Microsoft szellemi tulajdonának védelmére létrehozottakkal. Ezért az eddigi fejlesztések nem a Microsoft-termékekre (például a Windows kernelre), hanem az azokat kiegészítő megoldásokra összpontosítottak (ilyen a Novell ODF-átalakítója).

Fz: Szempont-e a Microsoft-termékek megtervezése és kifejlesztése során, hogy azok ergonomikusan működjenek: minél kevesebb erőforrást használva nyújtsanak minél nagyobb és jobb teljesítményt? (Hiszen a Vista képességei és erőforrásigénye sokak szerint nem ezt bizonyítja.)

MS: Az ergonómia azt jelenti, hogy valami biztonságos, kényelmes, könnyen használható, produktív és esztétikus(*). A Microsoft az egyik élharcos az ergonomikus szoftverek létrehozása területén. A „minél kevesebb erőforrást használva nyújtsanak minél nagyobb és jobb teljesítményt" kitételt inkább az ökonómia, azaz a gazdaságosság írja le. Természetesen ez is nagyon fontos szempont a terméktervezés és –fejlesztés során, és a Vista használata alatt szerzett visszajelzéseket beépítjük a Windows 7 tervezésébe, fejlesztésébe.

Fz: A Microsoft nemrég bejelentette, hogy saját víruskeresőt épít be a rendszereibe (Morro). A Symantec és más cégek is előre kritizálták az új vírusvédő képességeit. Mit lehet tudni róla: valóban a OneCare "lebutított" változata lesz? A rendszerekbe történő integráció mellett csak piaci érvek szólnak, vagy technikai érvek is? Ha igen, melyek?

MS: Nem meglepő, ha egy vírusirtó-gyártó kritizálja versenytársa termékét. A Morro a tervek szerint a Microsoft meglévő, több termékben használt vírusirtó-platformjára épül, de kimaradnak belőle a OneCare nem biztonsági jellegű szolgáltatásai, például a PC időszakos karbantartása a merevlemez tömörítésével. Ez kisebb letöltési méretet és áramvonalasabb működést eredményez.

Általában elmondható, hogy a magán felhasználók nagyon rosszul állnak a vírusirtó használata terén és ezek a számítógépek jelentik az egyik nagy veszélyt. A Microsoft megoldása elsősorban őket fogja támogatni.

Fz: Megállapítható-e, hogy a Web 2.0 kialakulásával és a nyílt forráskódú rendszerek felhasználóbaráttá válásával a Microsoft üzleti modellváltást kényszerül véghezvinni? Ha igen, miben jelentkezik ez a cég üzletpolitikájában?

MS: A Web 2.0 megjelenése szinte minden informatikai szállító üzleti működését befolyásolja. A Microsoft esetében ez azt jelenti, hogy meglévő üzleti modelljei mellé újakat is bevezet. Jó példa erre a Business Productivity Online Suite, ami a hagyományosan az ügyfelek telephelyén (on-premises) futó Exchange Server levelező kiszolgálót, SharePoint Server csoportmunka- és dokumentumkezelő rendszert, és más alkalmazásokat hostolt Microsoft-szolgáltatásként is elérhetővé teszi, méghozzá a dobozos vásárlás vagy a hosszú távú vállalati szerződés helyett, vagy mellett, havi előfizetési díj ellenében. A nyílt forráskódú rendszerek felhasználóbaráttá válásának ehhez nincs sok köze.

Fz: Az otthoni és a kisvállalati felhasználóknak szánt termékek esetében várható-e, hogy jelentősebb változtatások lesznek a Microsoft EULA-iban, vagyis a licencekben és a felhasználás feltételrendszerében?

MS: A licencek változása folyamatosan reagál a vásárlói igényekre. A jövőben egyre több olyan Microsoft termék is elérhető lesz a kisvállalatoknak, amelyeket kedvező módon tudnak majd igénybe venni, ezeket a termékeket a cég az „online" jelzővel árulja.

(*) „ergonómia (görög) a munka gazdaságos megszervezésének elmélete és gyakorlata, az ésszerű erőkifejtés tudománya" (Forrás: Idegen szavak és kifejezések szótára, szerk.: Bakos Ferenc, Akadémiai Kiadó, Bp. 1984, 229. old. – idézi: J.O.)

Jankovich Oszkár

A cikkhez tartozó fórum címe:
http://www.flosszine.org/microsoft_interju

HELLO WINDOW! 2

GTK+/gtkmm programozás GNU/Linux alatt

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

Bevezetés

A cikksorozat második részében az első alkalommal már bemutatott példaprogramokat újra felhasználva a GTK+/gtkmm programozás azon területeit vesszük sorra, melyek nélkül rövid távon el lehet boldogulni, de nélkülözhetetlenek az alapos megértést igénylő feladatok megoldásához.

Alapfogalmak

Még mielőtt ismertetnénk néhány alapfogalmat érdemes kitérni arra, hogy ezen túl a GTK rövidítést akkor használjuk, ha a felületprogramozási nyelvről általánosságban szólunk, míg a GTK+ és gtkmm kifejezések a konkrét C, illetve C++ nyelvű implementációkat jelölik.

Objektum-orientált megközelítés

Öröklődés

Annak ellenére, hogy a mechanizmust nyelvi szinten a C nem, csak a C++ támogatja igen is lehetséges objektum-orientált megközelítéssel élni az első esetben is. Erre kitűnő példa -egyebek mellett- a GTK+. Megoldott a widgetek egymásból történő származtatása, sőt felhasználói widgetek is definiálhatóak a már meglévőekre támaszkodva. Meg kell jegyezni, hogy a gtkmm esetén -lévén ott a nyelv C++- természetesen a származtatás egy nagyságrenddel egyszerűbb, de a letölthető példákat felhasználva némi rutinnal a GTK+ esetén sem igényel különösebb erőfeszítést.

Típusbiztosság

Hasonlóan a korábbiakhoz pusztán nyelvi szinten ez az eszköz sem megvalósítható (C esetén), ugyanakkor a GTK+ minden widgettípushoz -mondhatni osztályhoz- definiál egy-egy makrót, melyek segítségével, fordítási időben (compile time) ugyan nem, de futásidőben (run time) ellenőrizhető egy adott widget valódi típus, hasonlóan ahhoz, mint amire a dynamic_cast használata jelent a C++-ban.

Fentieket figyelembe véve láthatjuk, hogy a GTK+ ugyan C nyelven íródott, de számos, az objektum-orientált nyelvek esetén megszokott terminológiát használ, sőt ezeket a nyelvi eszközök adta mértékben meg is valósítja. Az OOP kifejezéseit ezért tudatosan használom az olyan esetekben is, ahol GTK+ nyelvű fejlesztésről esik szó.

Felületprogramozási kulcsszavak

Widget

A kifejezést, mint gyűjtőfogalmat használjuk a GTK programozás során a felhasználó felület egyes grafikai

elemeinek megnevezésére. Az elnevezés egyébiránt az X hagyományokból eredeztethető. A szó azonban nem csak erre szolgál, hanem annak az ősosztálynak a neve is -még ha ez a fogalom C nyelvben nem is létezik- melyből minden egyes megjeleníthető elem - gomb, menü, csúszka - származik.

GTK Main Loop

Ez a GTK tulajdonképpeni főciklusa, mely a Glib-ben implementál általános main loop-ot felhasználva csatlakozik a X szerverhez. Mindezt a GDK-n keresztül teszi, mely -mint azt az előző részben is említettük- egy burkoló réteg az ablakozó rendszer köré. Ezt azért fontos kiemelni, mert ezzel a módszerrel biztosítható, hogy a GTK működőképes legyen különböző grafikus szerverek (X, framebuffer) és platformokon (GNU/Linux, Windows, Mac OS X) futtatása mellett egyaránt. A main loop tehát az, ami az imént említett kapcsolaton át eljuttatja az ablakozó rendszer alacsony szintű eseményeit a GDK által standardizált formában az egyes widgetekhez.

Signal

A GTK, és egyébiránt minden más felületprogramozási nyelv, egyik kulcsszava. Jelentése (jel, jelzés) jól tükrözi funkcióját. Minden widgethez tartoz(hat)nak különböző események -mint amilyen egy gomb esetén annak lenyomása (vagy éppen felengedése), egy beviteli mezőnél az abba történő írás- melyekről a widgetek tudomást szereznek -egyébként a main loopon keresztül- elvégzik a megfelelő műveleteket -gomb lenyomásnál újrarajzolás, beviteli mezőbe írásnál a karakter megjelenítése- majd értesítést küldenek a program többi része felé. Ezt az értesítést, avagy jelzést nevezzük signal-nek.

Callback

Amennyiben egy adott widget által küldött meghatározott eseményről tudomást akarunk szerezni a program futása során, ezt megtehetjük azáltal, hogy a widget megfelelő signaljéhez egy callbacket, azaz eseménykezelő függvény társítunk. Itt minden olyan esemény elvégezhető, mely nem a widgethez, hanem annak programunkban betöltött szerepéhez kötődik. A korábbi példánál maradva ez egy Szerkesztés gomb lenyomásánál lehet egy dialógus ablak megjelenítése, egy beviteli mezőbe való írásnál -amennyiben ez szükséges- a tartalom ellenőrzése.

Az utóbbi két fogalom, mivel a GUI-k fejlesztése tulajdonképpen eseményvezérelt programozás, kiemelt fontosságú. Ennek okán erről a következő részben részletesebben is kívánunk szólni. Elöljáróban csak annyit, hogy a GTK lehetőséget ad a signalek widgetektől teljesen független használatára is, azaz akár saját osztályainkhoz is rendelhetünk eseményeket. Ez azonban már túlmutat ennek a résznek a keretein...

A megvalósítás

Az előző fejezetekben olvasható metodikák és módszerek közül a GTK GObject, avagy Glib::Object osztálya az alábbiakat valósítja meg:

Öröklődés

Az oject osztály típus minden widget, és egy más a GTK-ban használt nem vizuális elem őse.

Típusbiztosság

Ez az osztály implementálja azt a mechanizmust, melynek segítségével lehetővé válik a GTK+-ban a futás idejű típusellenőrzés.

Signal

Ezen osztályon keresztül valósul meg a szignálkezelés, mely lehetővé teszi adott eseményekhez kezelőfüggvények (callback) kapcsolását. Itt kell megjegyezni, hogy mivel az object nem csupán a widgeteknek őse, így olyan GTK-s, vagy akár saját, elemeknek is lehetnek szignáljai, melyek közvetlenül nem láthatóak, mint az általunk létrehozott GUI részei.

Referencia-számlálás

Minden objectből származó osztály, így a widgetek is, rendelkeznek referencia-számmal, mely tulajdonképpen azt fejezi ki, hogy hányan hivatkoznak az adott elemre. A GTK, pontosabban ez esetben a GLib lebegő referenciát (floating reference) alkalmaz, mely azt jelenti, hogy az objektum létrejöttekor annak referenciája 1 lesz, de ezt a referenciát úgymond nem birtokolja senki, azaz ha a widgetet egy konténer osztályba (melyekről részletesebben a következő rész szól majd) kívánjuk tenni, akkor -az első ilyen alkalommal- a refernciaszám nem nő, annak ellenére sem, hogy ez valójában hivatkozást jelent az adott elemre. A változatlanul hagyott referencia-érték jelenti a lebegő referencia elsüllyesztését (sink). Minden azt követő esetben a konténerből történő eltávolítás csökkenti, ahoz való hozzáadás pedig növeli a referencia értékét. Érdemes felhívni a figyelmet arra, hogy az elmondottak alapján, ha hozzáadtuk widgetünket egy containerhez, majd pedig eltávolítjuk belőle azt, akkor annak referenciája 0-ra csökken, ami maga után vonja a destruktor lefutását. Ezt elkerülhetjük, ha az eltávolítás előtt explicit módon növeljük a referenciát, amit aztán csökkentenünk kell, ha egy másik osztály ``birtokába'' adjuk a widgetet.

Első ablakunk háttere

A kód

```c
#include <gtk/gtk.h>

int main(int argc, char *argv[])
{
  GtkWidget *window;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
   gtk_widget_show  (window);

  gtk_main ();

  return 0;
}
```

```c
#include <gtkmm.h>

int main(int argc, char *argv[])
{


  Gtk::Main kit(argc, argv);

  Gtk::Window window;


  Gtk::Main::run(window);

  return 0;
}
```

Az ismétlés kedvéért -no meg persze, hogy kéznél legyen- lássuk most az előző számban már bemutatott példaprogramokat.

A fenti példaprogramok C/C++ nyelvű forrásfájljai, illetve azok eredetijei, a FLOSSzine, valamint a GTK+/gtkmm oldalain az alábbi linkeken érhetőek el:

http://www.flosszine.org/sources/gtk_window.c

http://www.flosszine.org/sources/gtkmm_window.cc

http://library.gnome.org/devel/gtk-tutorial/stable/c39.html

http://www.gtkmm.org/docs/gtkmm-2.4/docs/tutorial/html/chapter-basics.html

Részletek sorról sorra

Vegyük a példaprogramokat szó szerint sorról sorra górcső alá.

1
A header fájlok beszerkesztésének különbözőségeiről az előző részben esett szó, így erre itt nem térnünk ki.

3
A main függvény a programunk belépési pontja, azaz itt kezdődik meg a futtatás. A függvény paramétereiről részletesebben Vomberg István ``Hello world!" című cikksorozatának második részében lehet olvasni.

5
A C nyelvi verzióban kénytelenek vagyunk kicsit korábban deklarálni azt a változót, ami ebben az esetben widgetünk címét tartalmazza majd, mivel az ISO C90 szabvány még nem, majd csak az ISO C99 támogatja a blokkon belül a kifejezés után elhelyezett változódeklarációkat. Megjegyzendő, hogy 3.0-nál korábbi gcc esetén is problémába ütköznénk a C++ példában használt módszerrel, így a C nyelvű kódokban a biztonság kedvéért maradunk a hagyományoknál, azaz lokális változók deklarációja csak blokkok kezdetén szerepel.

7
Eljutottunk végre az első GTK specifikus híváshoz, mely funkcionalitásában, azonos mégis van köztük árnyalatnyi különbség. A GTK+-os verzióban mind az argc, mind az argv változó címét adjuk át, biztosítandó azt, hogy az init függvény a GTK-nak szóló paramétereket el tudja távolítani a tömbből és azok számával csökkenteni tudja argc értékét. Erre a C++-os változat esetén csak azért nincs szükség, mert -még ha nem is látszik- mindkét változóra referencia adódik át a Gtk::Main konstruktorának. A GTK által értelmezett parancssori paramétereket foglalja össze a http://library.gnome.org/devel/gtk/stable/gtk-running.html oldal.

9
Első widgetünk létrehozása. A C-s, illetve C++-os nevezéktannak megfelelően létezik minden widget-típushoz létezik konstruktor, előbbi esetben gtk_widgetnév_new függvény, míg utóbbi esetben Gtk::WidgetNév::WidgetNév konstruktor formájában. A különbség nem is annyira az nevekben, mintsem a memóriakezelésben rejlik, hiszen egy újjolag allokált objektumot kapunk C esetén, aminek a felszabadí-tásáról magunknak kell gondoskodnunk, míg a C++ a tőle megszokott módon felszabadítja a lokális változókat. Ezen a helyzeten csak a korábban már említett lebegő referencia bonyolít avagy egyszerűsít némiképp.

10
Némi meglepetés érhet bennünket, ezt a sort látván, hiszen első olvasatra nem teljesen nyilvánvaló, hogy miért is van szükség a C nyelvű változat esetén az megjelenítő függvény explicit hívására, és hol marad ez a C++ változatból.

12
A megoldás a két -egyéb tekintetekben egyforma- függvény különbségében rejlik. Ez pedig az, hogy a Gtk::Main::run a paraméterként kapott widget (jelen esetben window) esetén meghívja annak show metódusát. Az azonosságról annyit, hogy a hívások a -korábban már részletezett- GTK main loopot indítják, azaz itt kezdődik meg az az eseményvezérelt szakasz, mely a gtk_main_quit, illetve Gtk::Main::quit hívásokkal ér véget.

14
Visszatérési értékünk mindkét esetben 0, amivel rendszerint azt jelezzük a hívó félnek, hogy a futás rendben lezajlott.

Vágyaink ablaka

Fordítás és linkelés
A korábbiakhoz hasonlóan az alábbi parancssorok segítségével fordíthatóak elemzett programjaink:

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

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

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

Irodalomjegyzék:
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

A cikkhez tartozó fórum címe:
http://www.flosszine.org/hello_window_02

HELLO WORLD! - 3.

Az előző két részben megismerkedtünk a programunk lefordításával, illetve az argumentum lista kezelésével. Amennyiben az argumentum filenevet is tartalmaz, úgy hamar szembekerülünk az úgynevezett wildcard-ok használatával mint pl. lib*.c.Jelen cikkben ezeknek a feloldásáról lesz szó.

Az előző két részben megismerkedtünk a programunk lefordításával, illetve az argumentum lista kezelésével. Amennyiben az argumentum filenevet is tartalmaz, úgy hamar szembekerülünk az úgynevezett wildcard-ok használatával mint pl. lib*.c.Jelen cikkben ezeknek a feloldásáról lesz szó.

Mindenkinek ismerős lehet egy olyan program futtatása, amely fileneve(ke)t kaphat argumentumként, sőt wildcardokkal ellátott fileneveket. Tipikusan ilyen program a grep, mellyel szövegfile-okba kereshetünk szövegrészleteket. Például a .c kiterjesztésű file-okban keressünk rá egy változónévre, mely legyen width_1 és az összes .c kiterjesztésű file-t vizsgáltassuk át:

```
grep -n width_1 *.c
```

A -n opció csak annyit jelent, hogy azt a sorszámot is kiírja, amelyik sorban a megtalált szövegrész szerepel, a keresett szövegrész a width_1 és amire most koncentrálni fogunk az a *.c. Nem is kell külön körülményes rávezetés, természetesen igen, a Linux tartalmaz komplett megoldást, ehhez tartozó függvények formájában, hogy ezeket a fileneveket feloldjuk, azaz egy listában, kompletten kifejtve megkapjuk az összes illeszkedő filenevet ami ezekre az úgynevezett patternekre illeszkedik. Összességében itt három függvénycsaládról van szó, az fnmatch(), a glob() és a wordexp() függvényekről. Ezek közül a napi gyakorlatban a glob()-nak van jelentősége, a másik kettőt ismertető jelleggel tárgyaljuk csak.

Kód 1.

```
#include <stdio.h>
#include <errno.h>
#include <glob.h>
#include <unistd.h>

int main(int argc, char **argv)
{
  int i;
  int flags = 0;
  int ret;
  glob_t results;

  ret = glob(argv[1], flags, NULL, &results);
  if (ret != 0) printf("Error\n");

  for (i = 0; i < results.gl_pathc; i++) {
  printf("%s\n", results.gl_pathv[i]);
  usleep (10000);
  }

  globfree(&results);
  return 0;
}
```

A glob()

```
#include <glob.h>

int glob(const char *restrict pattern, int flags,
int(*errfunc)(const char *epath, int eerrno),
glob_t *restrict pglob);
void globfree(glob_t *pglob);
```

A glob() függvénnyel előállíthatunk egy listát, amennyiben van egy patternünk, majd ha feldolgoztuk ezt a listát, úgy a globfree()-vel felszabadíthatjuk a lista által lefoglalt területet.

Rögtön nézzük is meg egy kis példaprogramon, hogy miről is van szó: [Kód 1]

Nevezzük el a programot prog_03.c-nek és fordítsuk le: gcc -Wall -o prog_03 prog_03.c

A glob_t típusú struktúrában fogjuk megkapni az eredménylistát. A flageket most alaphelyzetben hagyjuk, a hibakezelő függvényt sem használjuk (az a NULL az argumentumlistában), és csak az egy darab parancs argumentumot használjuk, azaz a program futtatása mindössze ./prog_03 *.txt lesz. Arról gondoskodjunk, hogy legyen pár .txt kiterjesztésű file az adott könyvtárban, mert különben hibaüzenettel áll le a program.

Nézzük meg, hogy tudunk ez után egy hibaüzenet kiíratót függvényt csatolni a glob()-hoz:

```
int globerr(const char *path, int eerrno)
{
printf("%s: %s\n", path, strerror(eerrno));
return 0;
}

...
...
ret = glob(argv[1], flags, globerr, &results);
...
...
```

Egyrészt ügyeljünk arra, hogy az #include <errno.h> sort tartalmazza a kód. Amennyiben a globerr() függvény nem 0 értékkel tér vissza, úgy a glob() függvény rögtön visszatér.

Felmerül a kérdés, hogyha a parancssori argumentumlistában több wildcard-os filenév is van, akkor azokat hogyan tudjuk feldolgozni? Itt jönnek a képbe a flag-ek, ugyanis a glob()-ot többször is meg lehet hívni, azonban olyankor már a GLOB_APPEND flaget kell használnunk. Figyelem! **Az első alkalommal viszont ezt tilos!**

```
for (i = 1; i < argc; i++) {
flags |= (i > 1 ? GLOB_APPEND : 0);
ret = glob(argv[i], flags, globerr, & results);
...
}
```

Tételezzük fel, hogy az argumentumlistánk csak ilyen wildcardos fileneveket tartalmaz (azaz például opciókat most nem), tehát a múltkori cikkben említettek alapján az argv[1]-től kezdve a szükséges argumentumokat találjuk. (Emlékeztető kérdés: mit tartalmaz az argv[0]?) A flag beállítása egyszerű, amennyiben a

ciklusváltozó 1, akkor még csak első alkalommal indítottuk a glob()-ot, és a flag értéke marad 0, ha 1-nél nagyobb akkor viszont a flag értékére "rárakódik" (azaz logikai VAGY kapcsolattal - szlenggel mondva - "rá-OR-olódik") a GLOB_APPEND. Azért ez a kacifántos módszer, és nem pedig egy direkt értékadás, mert ezek a flag-ek logikai vagy kapcsolattal használhatók egymás mellett, és a flag változó már tartalmazhat valamilyen értéket, melyre már csak rá kell "ragasztani" ezt a bitet a |= operátorral.

A glob() két olyan lehetséges hibától is függőségben van, amit nem maga a függvény okoz. Amennyiben elfogy a memória a lista feldolgozása során, illetve egy nem olvasható alkönyvtárhoz ér, úgy a megfelelő hibaüzenettel tér vissza, emiatt a függvény visszatérési értékét feltétlenül meg kell vizsgálni. A memóriafoglalás problémáját ne a mai, gigabyte-os korszak szemüvegén keresztül tessenek nézni, ez a függvény már a libc4-ben is jelen volt, s akkoriban még a UNIX is COCOM listás volt, mint operációs rendszer és a memória mennyiségét kilobyte-okban mérték. Ha azonban legalább pár tízezer file van az adott könyvtárban, akkor még ma is tudunk szórakoztató perceket okozni magunknak a függvény meghívásával.:-)

Az fnmatch()

```
#include <fnmatch.h>

int fnmatch(const char *pattern, const char *string, int flags);
```

Az fnmatch() egy egyszerű függvény, ellenőrzi, hogy az adott pattern ráillik-e egy adott filenévre. Értelemszerűen a glob() is ezt használja, azonban az alkönyvtár bejárását is megoldja emiatt ez a függvény (az fnmatch) csak ritkán kerül alkalmazásra egy átlagos programnál. A flagek az egyes karakterek viselkedését írják elő, találat esetén pedig 0-val tér vissza a függvény. A flageket illetően a man fnmatch man page elolvasása javasolt.

A wordexp()

```
#include <wordexp.h>

int wordexp(const char *restrict words,
            wordexp_t *restrict pwordexp,
            int flags);
void wordfree(wordexp_t *pwordexp);
```

Ahogyan a shellben is a parancsértelmező értelmezni tudja például a ~ jelet a felhasználó HOME könyvtáraként, a $XYZ-t ki tudja fejteni mint az XYZ környezeti változó tartalmát, úgy ezt a szolgáltatást a wordexp() függvénnyel programból is megkaphatjuk. Hacsak nem akarunk egy shell jellegű programot írni, úgy a használatába nem fogunk belebotlani. Amennyiben olyan felhasználói inputot kell feldolgoznunk, mely a shellben használatos módszereket is tartalmazza (pl. a tilde ~ karakter mint HOME könyvtár), úgy az alábbi mintaprogram segítségével el tudunk igazodni a használatában. A mintaprogram az ide vonatkozó man page-et idézi. [Kód 2.]

A program ebben a formában azt a listát állítja elő, mintha az "ls [a-c]*.c" parancsot adnánk ki a shellben. Látható, hogy a függvény felülről kompatibilis a glob()-bal, vajon miért van mégis külön ez is, meg a glob() is? A válasz kézenfekvő, a wordexp() jóval több erőforrást, futásidőt igényel, a glob() gyorsabb és hatékonyabb ha nincs szükség a wordexp() plusz funkcióira. A függvény használata egyszerűbb, mint a glob()-é, a wordexp()-et meghívva kapunk egy eredménylistát (ez jelen esetben a p és a w változókon keresztül érhető el), majd felszabadítjuk a lefoglalt erőforrásokat. A visszatérési értékekről, és az egyes flag-ek jelentéséről, használatáról az idevonatkozó man page-et tanulmányozzuk, man wordexp módon.
Összefoglaló

Kód 2.

```
#include <stdio.h>
#include <stdlib.h>
#include <wordexp.h>

int
main(int argc, char **argv)
{
    wordexp_t p;
    char **w;
    int i;

    wordexp("[a-c]*.c", &p, 0);
    w = p.we_wordv;
    for (i=0; i < p.we_wordc; i++)
        printf("%s\n", w[i]);
    wordfree(&p);
    exit(EXIT_SUCCESS);
}
```

Nagyon hasznos függvényeket kaptunk a parancssori argumentumlista feldolgozásához, azonban a cikk keretei közt csak vázlatos ismertetésre van lehetőség. Az idevonatkozó man page-ek tanulmányozásán kívül az alábbi tanácsokat tudjuk adni:
- A megfelelő header file-ok beillesztését (pl. #include <glob.h>) ne felejtsük el.
- A visszatérési értékeket mindig figyeljük és kezeljük le a hibákat.
- Ne feledjük a megfelelő *free() függvénnyel felszabadítani az erőforrásokat.

Gyakorlásként rakjunk össze egy teljes glob()-ot használó kis programot, tehát a külön említett hibakezelő és flag beállító részeivel együtt és teszteljük.

Vomberg István

A cikkhez tartozó fórum címe:
http://www.flosszine.org/hello_world_programozas_linux_kornyezetben_03

Tudtad-e?

Mandriváról magyarul
Novemberben megújult Barta Károly honlapja (http://brtkr.extra.hu), és blogja (http://brtkr.blogspot.com), sőt, elkészítette "A Mandriva Linux 2008 használata" című könyvének harmadik kiadását is, amely teljes egészében letölthető innen: http://brtkr.extra.hu/mandriva/index.html. A szerző egy informatikai Kislexikont is készített, amely egyfelől megtalálható a könyv függelékében, de böngészhető is, itt: http://brtkr.extra.hu/dokuk/kislexikon.html. Barta Károly forrásai nemcsak Mandriva-felhasználók számára hasznosak, hanem bárkinek, aki valamilyen Linuxot, vagy Linuxon is létező alkalmazást futtat.

BASH ALAPOK 2. RÉSZ

Az előző számban bizonygattam, milyen nagyszerű is a Bash, és a parancssor, viszont a múltkori példák nem voltak igazán interaktívak. Ennek okán a legutóbb ígért sed és awk alapok kicsit később következnek, lévén elég nagy téma külön-külön mind a kettő.

Függvények és interaktivitás alapjai

Hogy az alcímbeli témát jobban megértsük, készítettem egy alapfokú számkitalálós játékot.:

```
#!/bin/bash

Valasz()
{
if [ $1 -gt $szam ]
then
echo "A szam kisebb. ($2. proba)"
else if [ $1 -lt $szam ]
then
echo "A szam nagyobb. ($2. proba)"
else
echo "Kitalaltad. ($2. proba)"
talalt=1
fi
fi
}



talalt=0
proba=0
szam=$RANDOM
echo "Szamkitalalos jatek."
echo "Gondoltam egy szamra. (min. 0, max. 32767)"
echo "(Erre gondoltam, de ne less: $szam)"
while [[ $talalt -eq 0 && $REPLY != "x" ]];
do
echo -n "Tipp: "
read
Valasz $REPLY $((++proba))
done
```

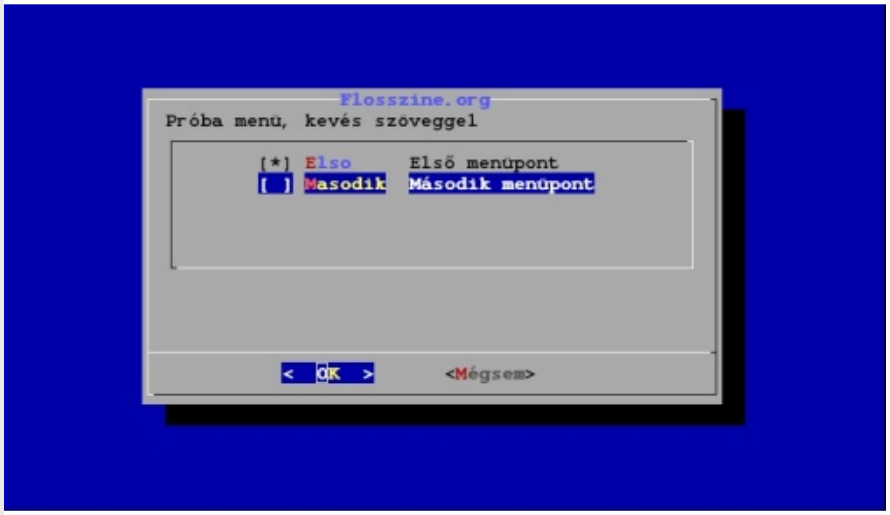
Ami legelőször feltűnhet, az a Valasz() függvény, melynek segítségével moduláris programot építhetünk fel. Az előző cikkben foglaltak elsajátítása után szinte ez már semmi újat nem tartogat, hacsak... Igen, a $1 és a $2 változók! Ha nem függvényben szerepelnének, akkor a

bash szkriptnek átadott változók értékét érhetnénk el bennük. De itt mit? A válasz az utolsó előtti sorban van: a függvény első és második paraméterét.

Ami még érdekes, az a $RANDOM függvény. Ez 0 és 32767 (2^15 – 1) között állít elő álvéletlen számokat. A tipp beolvasása a read utasítással történik, az értéket pedig nem meglepő módon a $REPLY adja. Ugye nem is bonyolult.

De mi a helyzet akkor, ha a bash szkriptünkben valamiért egy billentyűleütést várunk? Erre van egy sokkal egyszerűbb megoldás:

```
Pause()
{
key=""
echo Nyomj egy gombot a folytatashoz!
stty -icanon
key=`dd count=1
2>/dev/null`
stty icanon
}
```



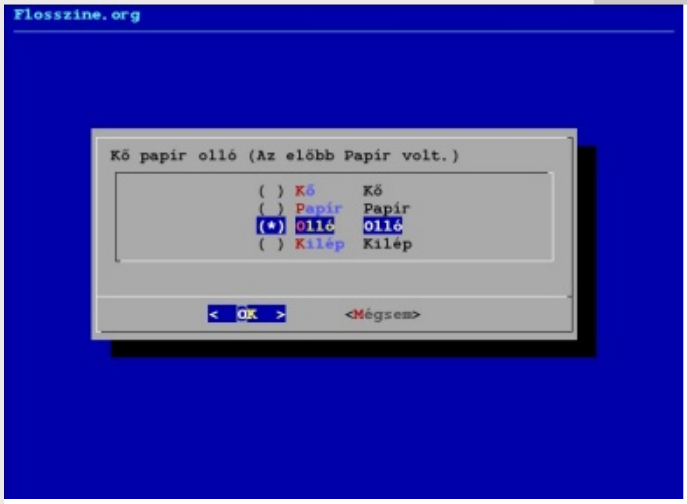
A $key változóban megkapjuk a lenyomott billentyű kódját, de ezt nem használjuk.

Dialog és az Ncurses

Van persze, amikor nem elég az ilyen fokú interaktivitás. Ilyen esetekre készült az ncurses-alapú dialog. Linux rendszereken általában elérhető, és alig 300 kbyte helyet foglal. A dialoggal lehetőség nyílik az alábbi típusú dobozok létrehozására:

Igen/Nem
Menü
Szövegbeviteli mező (egysoros)
Üzenet (jellemzően csak egy OK gombbal)
Szövegbeviteli mező (többsoros)
Választós lista
Rádiógombos lista
Folyamatjelző csík

A pontos paraméterezést a dialog man oldalán érhetjük el. A mellékelt kép az alábbi paranccsal készült:




```
dialog --clear --title
Flosszine.org --checklist "Próba
menü, kevés szöveggel" 15 50 5
Elso "Első menüpont" on Masodik
"Második menüpont" off
```

A kő-papír-olló játék implementációja pedig a következő:

```
#!/bin/bash

valasz=""
while [[ $valaszt != "Kilép" ]];
do
kiir="Kő papír olló"$valasz
dialog --clear --backtitle Flosszine.org --radiolist "$kiir" 12 50 4
Kő "Kő" off Papír "Papír" off Olló "Olló" off Kilép "Kilép" off
2>/tmp/kopapir
valaszt=`cat < /tmp/kopapir`
valasz=" (Az előbb $valaszt volt.)"
done
```

A választott menüpont címkéjét a sztenderd hibakimenetre kapjuk, ezért szerepel a dialog sorának legvégén a 2>/tmp/kopapir, azaz a hibakimenet fájlba történő átirányítása. Gyakorlásképp, akinek van kedve, írja át a számkitalálós játékot úgy, hogy a dialog inputbox vezérlőjét használja bemenetként. (Elárulom, hogy nem nehéz.)

Ha egyszerre például több input típust adunk meg, akkor egymás után kerülnek bekérésre az adatok és a kimenetet pedig tabokkal elválasztva kapjuk meg. Példaképp próbáld ki ezt:

```
dialog --clear \
--backtitle Személyes --inputbox Vezetékneved: 10 50 \
--backtitle Személyes --inputbox Keresztneved: 10 50 \
--backtitle Személyes --radiolist Nemed: 10 50 2 Fiú Fiú off Lány
Lány off \
--backtitle Személyes --checklist Hobbjaid: 12 50 4 Olvasás Olvasás
off Fotózás Fotózás off
Főzés Főzés off Sport Sport off \
2>/tmp/personal
```

Remélem, hogy valamennyire sikerült felkelteni a bash-sal, és úgy általában a parancssoros megoldásokkal kapcsolatosan az érdeklődésedet!

Medve Zoltán

A cikkhez tartozó fórum címe:
http://www.flosszine.org/bash_alapok_02

MAGAS RENDELKEZÉSRE ÁLLÁS EGYSZERŰEN

A szolgáltatások magas rendelkezésre állása ma már mindenhol elvárás. Ehhez természetesen minimum kettő gép kell, hogy ha az egyik kiesik, akkor a másik átvehesse a feladatát. A HA (high availability), azaz magas rendelkezésre állást biztosító megoldások közül egy nagyon egyszerű megvalósítást mutatok be, az mfha projektet.

A működés elve ugyanaz, mint a „nagyoknál”: egy ún. heartbeat (szívverés) program fut minden résztvevő gépen (node). A node-ok közül valamilyen algoritmus segítségével választunk egy preferált gépet, amely a szolgáltatást (pl. webszerver) biztosítja. Az egyik lehetséges mód az, ha minden node azonos súllyal vagy preferenciával rendelkezik, és a leggyorsabb ragadja magához a nyújtani kívánt szolgáltatást. Az mfha számára azonban meg kell adni egy preferenciaértéket is, a DNS zónákban használt MX rekordokéhoz hasonlóan. A kisebb preferencia itt is nagyobb súlyt jelent, és ha minden node elérhető, akkor ő fogja nyújtani a szolgáltatást.

Akármilyen szolgáltatásról is legyen szó, egy (vagy több) IP-cím szükséges hozzá. Akármilyen HA-klaszterről is legyen szó, a szolgáltatás egy (vagy több) virtuális IP-címen érhető el, amelyet mindig az aktív node vesz fel. Ha az éppen aktív node kiesik, akkor a sorban következő veszi fel a címet, és veszi át a meghibásodott node helyét.

Ahogy fentebb utaltam rá, a HA node-ok heartbeat csomagokat küldenek a hálózatra, ezzel tudatják, hogy élnek és virulnak. Az mfha esetében csak az éppen aktív (master) node küld fél másodpercenként „I am alive” csomagot a passzív (slave, stand-by) módban lévőnek. Maga az UDP csomag egyébként szó szerint ezt a szöveget tartalmazza.

A passzív node veszi az üzenetet, és elégedetten dől hátra, hogy minden rendben van, lógathatja tovább a lábát. Ha beüt a baj, és meghal az aktív node, akkor a passzív node-on egy számláló elkezdi figyelni, hány csomagot nem kapott meg, azaz hány fél másodperce nem érhető el az aktív node. Ha a számláló elér egy beállított értéket, akkor arra következtetünk, hogy az aktív node meghalt vagy csak a takarító megbotlott az Ethernet kábelben, és kitépte azt. Igazából nem az érdekel minket, hogy mi van az aktív node-dal, hanem egyedül az, hogy látszik-e a hálózaton vagy sem. Nyújtja-e a szolgáltatást vagy sem? Hiába csak annyi a gond, hogy pl. meghalt a switch hozzá tartozó portja, a gép szolgáltatása akkor is elérhetetlen. Ez a szemlélet egy további eltérést is eredményez az egyéb HA-megoldásokkal szemben. Míg azok jellemzően egy belső, privát hálózaton is össze vannak kötve egymással, és azon történik a heartbeat csomagok továbbítása, addig az mfha a node-ok publikus, pontosabban a szolgáltatással összefüggő interface-en oldja meg ezt a kommunikációt. Ez nem okoz sávszélesség problémát, mert egy csomag kb. 45 byte méretű, azaz durván 90 byte/sec az mfha igénye.

Maga a heartbeat csomag tartalmaz egy azonosítót, amely a 2 node-ra egyedi. Ennek a segítségével tudja eldönteni az mfha passzív node, hogy ugyanabban a csoportban van-e, azaz hogy foglalkoznia kell-e a hálózatról érkező csomaggal vagy sem? Van benne egy preferenciát

jelző szám, további 1 byte az állapotnak (master vagy slave), egy időbélyeg, az üzenet, és végül egy autentikátor mező, hogy az mfha védett legyen a támadások ellen. Ez utóbbi hasonlóan működik, mint a RADIUS esetén az azonos nevű mező, maga az ötlet is onnan van.

Ott hagytuk abba, hogy az aktív node kiesett, és a készenlétben lévő node elszámolt tízig. Ebben az esetben lefut egy erre az alkalomra tartogatott függvény, ami semmi mást nem csinál, mint a system() rendszerhívással meghívja a vip-up.sh nevű shell scriptet, ami nagy vonalakban annyit tesz, hogy felveszi a szolgáltatás virtuális IP-címét, és elindítja a szolgáltatást. Amikor az mfha program leáll, akkor utolsó leheletével hasonló módon lefuttatja a vip-down.sh scriptet, amely megkísérli szabályosan leállítani a szolgáltatást biztosító démont és egy ifconfig paranccsal leteszi a virtuális IP-címet. Ezek a műveletek nem mindig képesek lefutni, pl. ha a korábban említett takarító néni még a tápkábelben is megbotlik – bár ez ellen nincs az a szoftver, amelyik véd. A cél azonban az, hogy az mfha mindent megtegyen a szabályos leállás érdekében, hogy az adatok ne sérüljenek. Ha a gépet pl. kernelfrissítés miatt újra kell indítani, akkor az mfha ezt szépen meg is tudja tenni.

Említettem, hogy egy C-ben megírt program shell scripteket hívogat. Kaptam már ezért, hogy „és akkor ezt most hogy?" - olvasd: ez azért 2008-ban nem túl elegáns. Ez azért van így, mert az mfha egy általános HA-megoldás, és ahány szolgáltatás, annyiféle feladatot kell az elindításukhoz ill. leállításukhoz elvégezni, hogy nem érte volna meg ezt mind C-ben lekódolni. És ha ez a mentegetőzés nem elég, akkor hadd tegyem azt hozzá, hogy a két script végrehajtására csak ritkán van szükség: amikor meghibásodik az aktív node. Ahhoz tehát, hogy minden a helyén legyen, testre kell szabni a vip-up.sh és vip-down.sh scripteket, beléjük írni, hogy mit futtassanak le.

Minden HA megoldás esetén kulcsfontosságú a pontos idő, ill. az, hogy a résztvevő node-ok órája egyformán járjon. Jó ha az mfha node-ok is ntp-vel be vannak állítva. Ez azért is szükséges, hogy a visszajátszásos támadás (replay attack) ellen is rendelkezzen némi védelemmel. Alapesetben 5 másodperc csúszást még elvisel a program, ellenkező esetben eldobja a csomagot. Igény szerint azonban ez a viselkedés kikapcsolható.

A kiesett node-ot ajánlatos hamar rendbe tenni, elhárítani az esetleges hardverhibát, vagy visszaállítani a működőképes kernelt. Ha ezzel megvagyunk, és elindul a preferált node, fut rajta az mfha, és veszi az éppen aktív node heartbeat csomagjait. Ez alapján megállapítja, hogy neki jobb a preferenciája, és átvált aktív módba. Ezzel egy időben (bár nem atomi műveletként) a másik node visszaadja a szolgáltatás erőforrásait. Közös storage használata esetén biztonságosabb az a megoldás, ha a visszatérő és jobb preferenciával rendelkező gép nem veszi vissza azonnal az erőforrásokat, hanem az adminisztrátorok manuálisan teszik ezt meg – ha szükséges.

Ennyi elmélet után nézzünk meg egy példát 2 node segítségével kialakított nagy megbízhatóságú web kiszolgálóra!
Töltsük le az mfha legfrissebb verzióját a http://dev.acts.hu/mfha/mfha-0.3.2.tar.gz címről. Kicsomagolás után a szokásos make paranccsal fordíthatjuk le. A telepítés abból áll, hogy az mfha programot és a vip*sh scripteket bemásoljuk a /usr/local/sbin könyvtárba.
Legyen a 2 node IP-címe 1.2.3.4 és 1.2.3.5, ill. a virtuális IP-cím pedig 1.2.3.6, azaz az egyes VirtualHost bejegyzések ehhez a címhez vannak kötve. Készítsük el először a shell scripeket, amelyek pl. az alábbi módon nézhetnek ki.

```
#!/bin/sh
export
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
ifconfig eth0:0 1.2.3.6 netmask 255.255.255.0 up
apachectl start
```

1. Lista: A vip-up.sh script

```
#!/bin/sh
export
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
apachectl stop
ifconfig eth0:0 down
rm -f /var/run/httpd.pid
```

2. Lista: A vip-down.sh script

Nevezzük ki az 1.2.3.4-et preferált node-nak, amely a szolgáltatást biztosítja, ha elérhető. Legyen az ő preferenciája 10, a passzív node-é pedig 20, ill. állítsuk be az azonosítót 42-re. Válasszunk egy jelszót, pl. hajramfha.

Indítsuk el ezután az mfha programot először az 1.2.3.4 IP-címmel rendelkező gépen az alábbi módon:

```
mfha -s 1.2.3.5 -x hajramfha -i 42 -p 10
```

A másik gépen (1.2.3.5) pedig ezt a parancsot adjuk ki root-ként:

```
mfha -s 1.2.3.4 -x hajramfha -i 42 -p 20
```

Ezek hatására az 1.2.3.4 gépen pár másodperc múlva lefut a vip-up.sh script, ami felveszi az 1.2.3.6 IP-címet, és elindítja az Apache démont. Ha az 1.2.3.4 gépet újraindítjuk, akkor az leállítja az Apache-ot, leteszi az 1.2.3.6 címet. A készenlétben várakozó node pedig kb. 5-10 másodperc múlva felveszi a virtuális címet, és elindítja az Apache-ot. További másodpercekbe telik, mire a hálózat rádöbben, hogy az 1.2.3.6 IP-cím már a másik gépen van. Saját méréseim alapján a böngészők kb. 40 másodperc kiesést tapasztalnak.

Mivel az mfha indítja el ill. állítja le a szolgáltatást, ezért mindig futnia kell. Használhatjuk erre a célra az init konfigurációs állományát (/etc/inittab), vagy írhatunk egy cron job-ot, ami percenként figyeli, hogy fut-e az mfha, és elindítja, ha kell. Azonban akkor járunk a legjobban, és oldjuk meg a legegyszerűbben a problémát, ha D.J. Bernstein daemontools csomagját (http://cr.yp.to/daemontools.html) használjuk. A hozzá szükséges 'run' fájl pl. így nézhet ki:

```
#!/bin/sh
export
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
exec mfha -s 1.2.3.5 -x hajramfha -i 42 -p 10
```

3. Lista: Az mfha használata daemontools-szal együttműködve

Ettől kezdve nem kell aggódnunk, hogy az init nem lőtte-e ki a kedvenc HA programunkat: a daemontools gondoskodik róla, hogy mindig fusson. Sőt, az svc paranccsal igény szerint kényelmesen le is tudjuk állítani.

Az mfha tartalmazza a strike-ha projekt sa.c forrását is, amely a libnet library segítségével ARP csomagokat tud küldeni. Erre azért lehet szükség, hogy a switch-ek és a többi hálózati eszköz megtanulja a virtuális IP-címhez tartozó új MAC-címet. Én azonban azt tapasztaltam, hogy – az én környezetemben - enélkül is működik az átállás.
A program a syslog() rendszerhívással naplózza az állapotváltásokat, így adott esetben nem csak az uptime értékéből tudhatjuk meg, hogy valami történt. Az mfha azt a helyzetet is megpróbálja lekezelni, ha egy switch port fel-le járkál. Ha kiesett az aktív node, akkor nem veszi át azonnal a szerepét a passzív, hanem megvár 10 elveszett heartbeat csomagot, és csak azután. Hasonlóan, ha a nagyobb preferenciájú node magához tér, akkor ő is megvár 10 heartbeat csomagot, mielőtt visszavenné az erőforrásokat: IP-címet, szolgáltatást, stb.

Végül pár szó a biztonságról. Mivel egy HA megoldásnak privilegizált műveleteket kell végrehajtania, pl. IP-cím felvétele, esetleg root jogokkal futó vagy induló programok elindítása, leállítása, ezért az mfha is root jogokkal kell fusson. Jelenleg az mfha a parancssorból vesz minden paramétert, így a jelszóként is tekinthető secret-et is. Ez bizonyos környezetben probléma lehet, pl. ha egy web kiszolgálón valaki egy PHP scripttel lekérdezi – és látja is – az összes processzt, akkor kinyerheti a jelszót. Ennek elkerülésére hamarosan egy konfigurációs állományba kerülnek a paraméterek.

Záró gondolatok

Noha az mfha kiválóan működik, azonban nagyobb környezetben kevés lehet az, hogy csak 2 node-ot támogat. Noha a node-ok rendben átveszik egymás szerepét, azaz megvalósul a HA funkció, bizonyos környezetben ez nem elég. Vegyünk példának egy nagy rendelkezésre állású POP3 kiszolgálót, ahol a postafiókok állapota pillanatról pillanatra változik. Egy ilyen konfigurációban arról is gondoskodni kell, hogy a készenlétben lévő node ugyanazt az állapotot vegye majd át, amit a kieső aktív node maga után hagy. Ez tipikusan egy közös, és általában külső, diszket storage-ot feltételez, amit mindkét (minden) node elér, és a HA szoftver kezeli ezek hozzáférését. Ha ilyennel rendelkezünk, akkor még azokat a parancsokat is bele kell írni a vip*sh scriptekbe, amelyek fel- ill. lecsatolják (mount/umount) a közös diszket. Az mfha ilyen környezetben is megállhatja a helyét. Egy másik lehetőség pl. a Linux-hoz elérhető DRBD használata, amely valós időben szinkronizálja a diszk írásműveleteket a 2 node között, így garantált a konzisztens állapot.

Sütő János

A cikkhez tartozó fórum címe:
http://www.flosszine.org/mfha

Impresszum

Főszerkesztő:
Horváth Örs Apor - *Budapest*

Tördelőszerkesztő:
Falusi Ernő - *Budapest*

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

Közreműködők:
Kövesdán Gábor - FreeBSD
Páli Gábor - FreeBSD

Logó:
Makay József - SKL Projekt

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

Köszönet az FSF.hu Alapitványnak a tárhelyért!