Fejlesztés linuxos környezetben

Javaslatok és ötletek a Linux használatára „Linux-mentes” fejlesztőcsoportoknak.

Meglehetősen viharos változásokat hozott a Linux az utóbbi másfél évben a kiszolgálóoldali operációs rendszerek világában. Ennek tudható be, hogy manapság egyre többen látnak jövőt olyan rendszerekben is, melyek nem tartoznak azon néhány vállalatóriás érdekeltségi körébe, melyek hagyományosan uralkodnak a nagyteljesítményű operációs rendszerek piacán. A Linux fejlődésének következő állomása a kilépés a háttérszolgáltatások világából – ahol a rendszer már bizonyított – az asztali rendszerek színpadára. Ez a lépés azonban nem is olyan egyszerű, hiszen a Linux tervezésekor alapvetően a kiszolgálók kívánalmait tartották szem előtt. Egészen a legutóbbi időkig nem is történt komoly kísérlet arra, hogy olyan alkalmazásokat fejlesszenek ki a végfelhasználó számára, melyek versenybe szállhatnak a jelenlegi piacvezetőkkel.
Jóllehet sokan úgy tartják, hogy a Linux még nem elég felkészült az asztali rendszerek megvalósítására, nincs már messze az idő, amikor a váltás bekövetkezhet. Írásomban megkísérlem felvázolni azokat a gondokat, melyekkel a Linux-felhasználóknak szembe kell nézniük, amikor a rendszer „megszelidítését” tervezik. Hogy ne csak a gondokról és kihívásokról halljunk, igyekszem bemutatni, miként lehet néhányukon felülkerekedni.
A téma esetemben azért időszerű, mivel jelenleg egy kis irodai rendszeren dolgozom, mely eszményi feltételeket teremtett arra, hogy kísérletet tegyek a Linux ilyen célú felhasználására. Nem kétlem, hogy ez a felállás sokak számára nem megfelelő, azonban hogy ne bonyolódjak unalmas részletekbe, megkísérlem a megszokottabb feladatok megoldásának módozatait bemutatni. A projekt valójában egy javás fejlesztés, a központi fejlesztőkörnyezet pedig egy Solaris-rendszer. A számomra kitűzött feladatok: a projekt megtervezése, a kód fejlesztése és ellenőrzése, a hibák felkutatása, valamint a napi ügykezelés – így a levelezés, a kutatási eredmények és a leírások olvasása és készítése – voltak.

Tervezőeszközök

A tervezés során jobbára az UML-t használtuk modellező nyelvként. Ez a nyelv sokféle jelölésmódot tartalmaz az objektumorientált programtervezés különböző feladattípusainak megjelenítésére. Míg a Windows környezet számos folyamatábra-készítő eszközt kínál, a Linux hasonló eszközei kevésbé ismertek, nehezebben beszerezhetőek, és talán nem olyan kiforrottak, mint Microsoft-világbeli társaik. Mivel a fejlesztőkörnyezet alapvetően Java volt, így megfelelő program keresésében nem kellett kizárólag a Linux világára korlátozódnom, hiszen a Java-programok természetüknél fogva rendszerfüggetlenek. Végeredményben, kalandozásaim során két olyan programot találtam, melyek megfeleltek igényeimnek.
Egyikük, az ArgoUML – melynek webhelyét a Kapcsolódó címek között is megtaláljuk – szerzői szerint „gondolkodó” tervezőeszköz. Ez voltaképpen annyit jelent, hogy terveinket áttekintve kísérletet tesz a hiányosságok és a következetlenségek felderítésére. A webhelyen elérhető programváltozat a fejlesztés egy köztes állapotában van – úgy látszik, hogy a szerzők nemrégiben átköltöztek egy másik webhelyre, és ott újult erővel kezdtek a fejlesztésbe.
A Windows alatt a Rational Rose UML eszköz érdemel kitüntetett figyelmet, ezért, ha képesek vagyunk a Rose által is használható projektfájlok írására és olvasására, szerencsésnek mondhatjuk magunkat.
A MagicDrawUML egy kereskedelmi UML tervezőeszköz, mely Javában íródott, így teljesen rendszerfüggetlen, és képes projektjeinket Rational Rose formátumban elmenteni, melyeket így megoszthatunk másokkal is. Mellékesen megjegyzem, akik a Java lassúságára panaszkodnak, itt egy szavuk sem lehet, hiszen a szerzők kitettek magukért e téren is. A felhasználói felület épp olyan gyorsan válaszol, mintha Windows- vagy Linux-rendszert használnánk.
Az egyetlen hibája, hogy pénzbe kerül, bár az összeg töredéke a Rational Rose árának, így egy kereskedelmi fejlesztés esetében ekkora befektetés mindenképpen megéri.

Fejlesztőeszközök

E szakaszban megkísérlem felvázolni, milyen segédeszközök léteznek Linux alatt Java-projektek fejlesztéséhez.
A fordítókörnyezetet a make program szolgáltatta, mely általánosan elterjedt a Linux-rendszereken, emellett a felületek sokaságán elérhető, így jól szolgálhatja rendszerfüggetlen fejlesztési céljainkat.
A legtöbb rendszerfüggetlen makrót és feladatot szabványos fordításvezérlő fájlokban (Makefile) határozhatjuk meg, így a legkisebbre csökkenthetjük a különböző rendszerek használatával kapcsolatban fellépő akadályok mennyiségét.
Mindenekelőtt hozzuk létre az ARCH környezeti változót, értékéül pedig az uname parancs visszatérési értékét adjuk. Ehhez (ha bash héjat használunk) egyszerűen írjuk be a következő parancsot a .bash_profile fájlba:

```
export ARCH='uname'
```

Ezután helyezzük el rendszerfüggő Make eljárásainkat és meghatározásainkat egy fordításvezérlő fájlban, melynek adjuk a Makefile.$ARCH nevet, ahol a $ARCH az uname visszatérési értéke az adott rendszerben. Végezetül, a fő vezérlőfájlban helyezzük el a következő sort, ez lehetővé teszi, hogy a make parancs végrehajtásánál rendszerfüggő értékeink is érvényre juthassanak:

```
include Makefile.$(ARCH)
```

Ha mindent jól végeztünk, a helyes beállítások betöltése futásidőben megtörténik.
A fordítókörnyezet kialakításának következő lépése egy Linux alatt futó Java-fordító felkutatása. Jelenleg számos Java-fordító és -értelmező projekt ismert, azonban a munkánkhoz sikerrel használható Java fejlesztőkörnyezetek közül leginkább csak a Blackdown és az IBM termékei jöhetnek számításba. A helyzettől függően tehát ezek közül kellett választanunk. Mindkét típus lépést tart a Sun legfrissebb fejlesztéseivel.
A Java használatának másik nagy előnye a fordítás után nyert bájtkód rendszerfüggetlensége. Ez azt jelenti, hogy ha további osztálykönyvtárakat, vagy jar fájlokat szeretnénk használni, nem kell a rendszerfüggő megvalósítások után keresgélni. Emellett számos Javaalkalmazás tartalmaz rendszerfüggő könyvtárakat, ezek használata így az adott rendszerre korlátozódik. A velük kapcsolatosan felmerülő nehézségek foka azonban nagymértékben függ alkalmazásuk gyakoriságától. Saját projektem például a JMS-t (Java Message Specification) használja,

melyet a Sun fejlesztett ki, a Java Message Queue (JMQ) pedig ennek megvalósításáért felel. A JMQ-nak jelenleg még nem létezik linuxos változata, következésképpen nem ellenőrizhettem vele a kódomat a Linux alatt. Ehelyett újra kellett fordítanom a kódot, majd (mivel a könyvtárak többnyire jar formátumúak) utána át tudtam vinni a kész anyagot a gépemre, és segítségével összeépíteni a programot.

A kód fejlesztésében segítő alkalmazásokkal Dunát lehetne rekeszteni. Mindegyik szerkesztőnek és felületnek megvan a maga szószólója, így ez ügyben csak félve mer nyilatkozni az ember – mindazonáltal, erőt véve magamon, bemutatom a kedvencemet. Munkaköri feladataim során néhány éve rákényszerültem a vi használatának megtanulására. Miután ezzel végeztem, gyorsan áttértem a vim és a gvim alkalmazására, és soha többé nem használtam a vi-t. Ezután még sok más fejlesztőkörnyezetet kipróbáltam, köztük a JBuildert, sőt, a Microsoft Developer Studióját is, azonban amíg nem használhatom bennük a VI billentyűkombinációit, feléjük se nézek. Nos, való igaz, hogy ezek az összetett fejlesztőkörnyezetek a hibakeresés terén többet tudnak a jó öreg vi-nál, azonban egy-egy megfelelő helyen elhelyezett println() utasítás csodákra képes.

A vim és gvim képes színes szintaktikai kiemelésre, a behúzások automatikus létrehozására, és még számtalan más hasznos dologra. Sőt, megfelelő beállításokkal a programozás-fordítás-hibakeresés ciklus végrehajtására is használható. A vi család leghatékonyabb képessége pedig a makrók alkalmazása, melyek egyszerű ismétlődő feladataink elvégzésével sok időt takaríthatnak meg.

A következőkben még két projektkezelési feladatról kell szót ejtenünk, melyek általában teendőink végére maradnak, ezek a forráskódkezelés és a hibák nyomon követése. A legelterjedtebb forráskód-kezelési programcsomagok az SCCS, az RCS és a CVS. Az SCCS nem érhető el nyílt forrású alkalmazásként, így foglalkozzunk a másik kettővel. Az RCS a kisebb projektek igényeinek felel meg, a CVS-sel való munkára pedig a közelmúltban nyílt lehetőségem. Ez utóbbi jobbára ugyanazt a fájltípust használja, mint az RCS, felhasználói felülete azonban kidolgozottabb, emellett pedig képes több fejlesztő és távoli ügyfél kiszolgálására is. Ez a lehetőség különösen jól jött saját projektemben, CVS-tárolónk ugyanis egy Solaris gépen foglalt helyet. Hogy megmutassam, mi mindenre képes a Linux, egy CVS-ügyfelet telepítettem linuxos gépemre (ez jó eséllyel hamarosan szélesebb körben is elfogadottá válik). A CVSROOT környezeti változó beállításával elértem a távoli Solaris gép CVS tárolóját, így a forráskódkezelést helyben megoldhattam. A CVS az rlogin parancs segítségével képes távoli parancsok futtatására, ezért hát győződjünk meg az elérés biztonságáról. A CVSROOT környezeti változó értékét a következő alakban kell megadnunk a távoli gép használatához:

```
export CVSROOT=:ext:hostname:CVSRepository
```

A jól használható nyílt forrású hibákat nyomon követő programok igen ritkák – mind közül csak kettőt találtam kellően megbízhatónak, és még itt is kénytelen voltam kiejteni az egyiket, mert nem ment át követelményeink szigorú rostáján. Egyetlen alkalmazás maradt tehát: a Mozilla csoport terméke, mely csinos webalapú felülettel, valamint MySQL adatbázisháttérrel bír. Beállításai minden részletre kiterjednek, és néhány apróbb buktató leküzdése után sikeresen és megfelelő sebességgel futott.

Segédprogramok

Projektünk fejlesztésénél hasznát vehetjük néhány közismert linuxos segédprogramnak is. Itt nemcsak a kód megírása és a tesztelés nélkülözhetetlen segédeszközeire gondolok, hanem azokra az egyéb programokra, melyek életünket általánosságban megkönnyíthetik. Így például, a find, a cat, az awk és az egrep parancsokat sokszor alkalmazhatjuk a rendszerfelügyelet, vagy a forrásfájlok közti keresés során, nemegyszer rövid parancsfájlok hatékony részeként. Ha pedig nem telepítünk a Cygwin for Windowshoz hasonló programot, a Windows-felületen dolgozva igencsak hiányát érezzük majd hasonló segédprogramoknak. Létezik a segédprogramoknak egy különálló csoportja, melyek használata erősebben kötődik a kód írásának folyamatához. Ezek segítenek a programlista megfelelő formátumának kialakításában, igazodva valamely szabványhoz – például a Sun Microsystems Java kódolási szabványához. Az Indent segédprogram – mely megtalálható a legtöbb Linux-rendszeren – számos beállítási lehetőségével segít kódunk végső alakjának kialakításában. Jelen pillanatban projektünk megfelel a Sun Microsystems kódolási szabványának. Amikor ennek kialakításán fáradoztam, belebotlottam a Jindent segédprogramba, mely Javában készült, így rendszerfüggetlenül működik. Alapértelmezésben a Sun szabvány formátumát készíti el, de saját beállítási fájl készítésével magunk is meghatározhatjuk a formázás módját.

Rendszerbeállítások

Az adatmentés egyszerűbbé tétele végett fejlesztésünk eredményének nagyobb részét Unix-kiszolgálókon tartjuk, ügyfélgépeink többsége pedig Windows NT. A Unix gépeken NFS és SAMBA működik, így a meghajtók megosztásával helyileg hozzáférhetünk olyan erőforrásokhoz, melyek eléréséhez egyébként be kellene jelentkezni valamely távoli gépen. A linuxos gépeken befűzhetünk mind NFS-, mind SAMBA-megosztásokat, ha pedig ugyanazt a könyvtárszerkezetet alkalmazzuk, mint a távoli gépen, akkor kevesebb esetben lesz szükség gépfüggő fordításvezérlő fájlok készítésére. Így például, az egyik felhasznált jar könyvtár a távoli gépen a /opt/FSUNjmq/lib/jms.jar néven érhető el. Ha a /opt/FSUNjmq könyvtárnak helyi gépünkön ugyanezt a nevet feleltetjük meg, a vezérlőfájlok ugyanúgy használhatóak lesznek a helyi gépen és a távoli Unix-rendszeren. A távoli meghajtók megosztását a következőhöz hasonló paranccsal végezhetjük el (itt az NFS esetét mutatjuk be):

```
share /opt/FSUNjmq
```

A SAMBA esetében másoljuk be a /etc/smb.conf (más rendszereken /etc/samba/smb.conf) fájlba valamely megosztásra adott példát, és a kapott új sorokban szereplő könyvtárakat cseréljük ki az általunk kívántakra (mindeközben ne felejtsünk el hozzáférési engedélyt adni a megfelelő felhasználónak). Néhány esetben előfordulhat, hogy mégiscsak be kell jelentkeznünk a távoli gépre, hogy különböző folyamatokat lefuttassunk – például, ha a távoli gép egy különleges adottságát használja a program, vagy ha nagy mennyiségű adatot kell másolnunk, mozgatnunk, vagy elérnünk. Mindezek mellett, egy távoli gépen bejelentkezve a hálózati működés is hatékonyabbá válik. Itt segít az rlogin, mely egyszerűbb elérést biztosít, mint a telnet. Használatánál előre beállíthatjuk a hozzáférési engedélyeket, így nem kell minden belépéskor beírnunk az azonosítónkat és jelszavunkat. Ehhez mindössze meg kell adnunk a távoli felhasználói könyvtár .rhosts fájljában a helyi gép nevét. Ilyenkor azonban ügyeljünk arra, hogy az írási/olvasási engedélyeket csak a tulajdonos számára adtuk meg, és csoportos vagy általános elérést nem tettünk lehetővé – másként ugyanis automatikus azonosításunk nem fog működni.

A távoli gépekre történő bejelentkezés során az egyik igen zavaró gond az volt, hogy még egyszer meg kellett adnunk a héj működésének beállításait. Ez azonban könnyen kiküszöbölhető, csak egy közös beállításfájlt (profile) kell készíteni, mely egyaránt elérhető a helyi és a távoli gépek számára. Ehhez készítsünk távoli felhasználói könyvtárunkban egy .commonProfile nevű fájlt, és fűzzük be ezt a felhasználói könyvtárat a helyi rendszeren. Így a /home/username könyvtár mellett egy /remote/username nevűt is kaptunk. Ezután már csak annyit kell tennünk, hogy mind a helyi, mind a távoli beállításfájlban a .commonProfile fájlra hivatkozunk, így ha bármit meg szeretnénk változtatni a két környezetben, csak e fájl tartalmát kell módosítanunk.

A Unix-ok távoli elérésével kapcsolatban fontos megjegyeznünk, hogy felhasználói azonosítónkat a helyi gépen nem választhatjuk meg akárhogy – az azonosító, az uid és a gid meg kell, hogy egyezzen azokkal, melyek a távoli gépen hozzánk tartoznak. Ez ugyanis igencsak leegyszerűsíti a dolgunkat az írás, a befűzés és a bejelentkezés engedélyeivel kapcsolatban. A tapasztaltabb felhasználóknak ismerős lehet a NIS fogalma – ez egy olyan módszer, melyben több unixos gép felhasználói azonosítóit és jelszavait egyetlen közös tárolóban helyezik el. Némi kutakodással és megfelelő hálózati beállításokkal magunk is elérhetjük, hogy linuxos gépünk NIS-t használjon más rendszerek felhasználóinak kezelésére. Ezzel feleslegessé válik a felhasználói és csoportazonosítók többszörös használata az új felhasználók beiktatásánál.
Hasonlóképpen nagyszerű lehetőség az X Window ügyfelek futtatása a távoli gépről helyi rendszerünkön, hiszen így távoli elérésünkhöz grafikus felületet is kapunk. Mindazonáltal – jóllehet futtathatunk távoli alkalmazásokat X Window megjelenítőnk ablakaiban – egyes esetekben felmerülhetnek ellentétek a távoli alkalmazás elvárásai és a helyi X Window kiszolgáló lehetőségei között. Klasszikus példaként említhetjük azt az esetet, amikor az X kiszolgáló (a helyi gépen) 16 bites színeket használ, a távoli alkalmazás azonban csak 8 bites színek használatára képes. Ilyenkor nem kell feltétlenül leállítanunk X Window munkafolyamatunkat, és újraindítanunk 8 bites színekkel. Egyszerűbb megoldás lehet még egy X Window munkafolyamatot indítani 8 bites színekkel, vagy a távoli ablakkezelő megjelenítőjét alkalmazni a helyi gépen. Erre példa a következő parancs:

```
X :1 -query remotehost
```

A „:1” azt jelenti, hogy ezt a példányt második megjelenítőként szeretnénk futtatni. Ehhez szükségünk van arra is, hogy a távoli gép DISPLAY változóját szintén a második megjelenítő állítsuk:

```
export DISPLAY=yourhost:1
```

Jó tanácsok a mindennapokra

Zárszóként megemlítem, hogy a Windows mellett a Linux segítségével is elvégezhetők mindazok a feladatok, melyek nap mint nap végigkísérik munkánkat.
A talán a legáltalánosabban előforduló feladat a Microsoft-fájlok írása és olvasása, legyen itt szó a Word, az Excel, vagy a PowerPoint fájljairól. A StarOffice nagyszerű programcsomagot kínál mindezek kezelésére, legfrissebb változata pedig további Microsoft programtípusokat is szárnyai alá vett.
Az elektronikus levelezéshez ügyfelek gazdag választékából csemegézhetünk. Természetesen a választék gazdagságát meghatározza az is, milyen levelezőkiszolgálót használunk. Ha POP vagy MAPI levelezésről van szó, semmi gondunk nem akadhat, hiszen az előbb említett StarOffice, valamint a Netscape is rendelkezik e protokollokat támogató beépített levelezőprogrammal.
A gondok azonban ott kezdődnek, amikor a levelezési szolgáltatónk a Microsoft Exchange-et használja, és nem teszi lehetővé az MAPI vagy a POP elérést. Jelenleg nem létezik MAPI-megfelelő ügyfél a Linuxhoz, így jelen ismereteim szerint csak olyan program jöhet szóba, mely a Windows alatt fut (ha bárkinek tudomása van más megoldásról, kérem, jelezze!). Mindazonáltal ilyenkor is létezik megoldás, erről a következőkben még szót ejtünk.
Sajnálatos módon a Linux jelenleg még nem rendelkezik azokkal a lehetőségekkel, melyekre napjaink irodáiban szükség van, ezért számos esetben a Microsoft termékeinek használatára szorulunk. A WINE lehetővé teszi, hogy Microsoft-alkalmazásokat futtassunk a Linux alatt, a VMWare pedig egy másik operációs rendszer példányát képes futtatni a Linuxon. Ezt mindenképpen érdemes kipróbálni, ha még nem tettük meg! Számomra a VMWare lehetőséget teremtett, hogy a Microsoft Outlook használatával elolvashassam az Exchange kiszolgálón tárolt leveleimet. Nos, ez az ágyúval verébre módszer jellegzetes esete, azonban mit tegyünk – valahogy csak el kell olvasni a leveleket!
A nyomtatás beállításai sok fejfájást okozhatnak a Linux-rendszer beüzemelése során, de ne aggódjunk, hiszen a nyomtatótámogatás és a meghajtók rohamléptekkel fejlődnek. Az alapvető szabályok szem előtt tartásával a linuxos hálózati nyomtatók beállítása nem okozhat különösebb nehézséget. Még akkor sem kell azonnal kétségbeesni, ha éppen nem találjuk a nyomtatóhoz használható meghajtót, hiszen a világhálón számos leírást találunk, melyek egy-egy nyomtatótípus működtetését írják le a megfelelő meghajtó hiányában. Így például, irodánkban működik egy Gestetner PCL nyomtató – saját meghajtó hiányában a HP LaserJet III meghajtójával.
Összefoglalva a leírtakat elmondhatjuk, hogy sok mindent megtehetünk a Linux-rendszerrel, még ha nem is kapunk közvetlen támogatást. Megpróbáltam érzékeltetni, milyen akadályokkal kerültem szembe munkám során, és hogyan oldottam meg őket. A Kapcsolódó címeknél néhány webhelyet találhatunk, melyek hasznos segítséget jelentettek, amikor arra került a sor, hogy valamilyen Windowsban szokásos feladatot Linux alatt oldjak meg. Mindazonáltal, fontos megjegyezni, hogy a Linux egyelőre még nem mindenható – néha vissza kell térnünk a nem linuxos alkalmazásokhoz.

Kapcsolódó címek

VMWare: ➲ http://www.vmware.com-

WINE: ➲ http://www.winehq.org/

MagicDraw: ➲ http://www.nomagic.com/

ArgoUML: ➲ http://argouml.tigris.org/

Segítség a NIS beüzemeléséhez:
➲ http://www.suse.de/kukuk/nis-howto/HOWTO/NIS-HOWTO.html

StarOffice:
➲ http://www.sun.com/products-n-solutions/software/prodapps/

Linux alkalmazások kincsestárai:
➲ http://freshmeat.net/ és
➲ http://slashdot.org/

Hasznos linuxos adatok:
➲ http://linux.com/,
➲ http://www.altavista.com,
➲ http://www.deja.com/usenet,
➲ http://slashdot.org/

Hasznos Java-adatok:
➲ http://java.sun.com/

Bugzilla: ➲ http://www.mozilla.org/bugs/source.html

CVS: ➲ http://www.cyclic.com/CVS/

VIM: ➲ http://www.vim.org/

Mark Stacey jelenleg a dublini székhelyű ICL vállalatnál dolgozik, érdeklődésének középpontjában a Linux és a Java áll. Munkaidején kívül szívesen látogat el messzi országokba. Kérdésekkel a mark.stacey@e-merge.ie címen kereshetjük fel.