Ismerjük meg a Monót!

A Mono haszna például abban áll, hogy segítségével a Linux képessé válik a Microsoft .NET rendszerével való munkára.

Amennyiben írtunk már valaha Linux-munkafelületen futtatható programot, vagy legalább megvizsgáltuk a lehetőségét, akkor ismert előttünk a nyelvi kötéseknek az a sokasága, ami a különböző grafikus felületek eszközkészletéhez rendelkezésre áll. A Linux grafikus felületére való programírásnak ez az egyik erőssége: nem vagyunk egy adott programozási nyelvhez hozzáláncolva. Sajnos hamarosan azt is észre kell vennünk, hogy a különböző nyelvi kötések az API igen eltérő készültségi fokával bírnak. Az egyik nyelven használt eszközkészlet egy másik nyelven esetleg még nincs támogatva – ez a „soknyelvűség” egyik hátulütője. Egy API karbantartásához szükséges munka mennyisége minden egyes nyelv esetén növekszik. Az eredeti API megváltoztatása vagy frissítése esetén a változást minden egyes nyelvi burkolón végig kell vezetni. Most pedig képzeljünk el egyetlen GUI-eszközkészletet, ami bármilyen programozási nyelvből anélkül elérhető, hogy API-burkolót kellene használnunk. Egy olyan eszközkészletről van szó, ami minden őt használó programozási nyelvnek ugyanazt a szolgáltatást nyújtja.

A Mono számos egyéb mellett ezzel a képességgel is rendelkezik, ezáltal teremtve meg felhasználója számára a nyelvfüggetlenséget és a nyelvek közötti párbeszéd lehetőségét.

A Mono története dióhéjban

A Mono története körülbelül két évvel ezelőtt kezdődött a Ximian nevű, Linux-programokra szakosodott cégnél. A Ximian leginkább a Ximian Desktopról, az Evolution naptár- és levelezőprogramjáról, a Red Carpet frissítőrendszerről és a vakbuzgó műszaki igazgatóról, ***Miguel de Icazá***ról ismert. Felismerve néhány, az utóbbi időben napvilágot látott szabványban rejlő lehetőséget, Miguel de Icaza elkészítette azt a programprototípust, amely később a Mono-projektben teljesedett ki.

De mik voltak azok a szabványok, amik szemet szúrtak Miguel de Icazának? Nem titok, hogy ez az ECMA-334 és az ECMA-335 volt, amelyek a Microsoft .NET fejlesztői felület alapját képező technológia leírását tartalmazzák. Ennél a pontnál hangsúlyoznunk kell, hogy lényeges különbség van a .NET fejlesztői felület és az általánosan használt „.NET” kifejezés tartalma között. A Microsoft-termékek és szolgáltatások széles körét – operációs rendszereket, fejlesztői eszközöket, hálózati szolgáltatásokat és alkalmazásokat – illetik a tágabb értelemben használt .NET kifejezéssel. Mi ennek a .NET-nek csak egy bizonyos részével foglalkozunk.

2000 októberében a Microsoft, a Hewlett-Packard és az Intel

Rövid bevezetés a C#-be

A C# magas szintű objektumközpontú programozási nyelv, amelyet a CLI (Common Language Infrastructure, általános nyelvi alap) kiegészítésére terveztek. A tervezői csapat élén *Anders Hejlsberg* (ismert Turbo Pascal- és Delphi-fejlesztő), *Scott Wiltamuth* és *Peter Golde* állt.

Első pillantásra a C# sok mindenben hasonlít a Javára. Bár a parancsformátum szempontjából hasonlóak, a C# számos olyan lehetőséggel is rendelkezik, amivel a Java nem. A szolgáltatások némelyike közvetlenül a C++-ból származik, ilyen a túlterhelés (operator overloading), a felhasználó által meghatározott átalakítások (user-defined conversions), igazi szabályos tömbök és hivatkozás szerinti értékátadás (pass-by-reference). A többi tulajdonság, mint az osztályok, a csatolófelületek, a `struct`, az `enum` és a még különlegesebb `delegate` és függvénymutatók, a C, a C++ és a Java keverékéből származnak.

A korszerű objektumközpontú nyelvek legfontosabb jellemzőinek támogatására a C# szintén rendelkezik a különböző elemszerkezetek – tulajdonságok, eljárások, attribútumok, események és leírás – lehetőségével. Ezzel nagymértékben leegyszerűsödik azoknak a fejlesztőknek a munkája, akik elemeket írnak és RAD-környezetbe akarják foglalni őket.

Azoknak a programozóknak a számára, akik járatosak a Java világában, e kis C#-bevezetésben foglaltak semmilyen gondot nem fognak okozni. Bár az objektumközpontú programozásban gyakorlott programozók érdeklődési területét ez a kis ismertető nem fedi le, a *Kapcsolódó címek* részben a C# nyelv részletesebb leírásaihoz is találhatunk hivatkozásokat.

Bevezetésképpen vessünk egy pillantást a jó öreg Hello World program C#-ben írt változatára:

```
/*
    A hello.cs program kódlistája
*/

using System;

public class HelloWorld
{
    public static void Main()
    {
        for (int x=0; x<3; x++){
            // Console from the System
            // namespace
            Console.WriteLine("Hello World");
        }
    }
}
```

folytatás a következő oldalon

közösen előterjesztette a Common Language Infrastructure (CLI, általános nyelvi alap) nevű futásidejű környezet és egy újonnan kifejlesztett objektumközpontú nyelv, a C# részletes leírását. 2001 második felétől a Ximian hivatalosan is elindította a Mono-projektet, azzal a céllal, hogy létrehozzák a javasolt szabványokon alapuló .NET fejlesztői környezet nyílt forrású megvalósítását. 2001 decemberében az ECMA (European Computer Manufacturing Association, Európai Számítógépgyártók Szövetsége) hivatalosan is jóváhagyta a CLI és C# leírásaira vonatkozó szabványokat.

Áttekintés

A CLI egy alaposztály-eljáráskönyvtárat és egy futtatókörnyezetet jelöl ki, ennek feladata többek közt az azonnali (JIT, Just In Time) fordítást, memóriakezelést, kivételkezelést megvalósító szolgáltatások, valamint a befűzési- és biztonsági szolgáltatások nyújtása. A folyamat jobb szemléltetéséhez segítséget nyújt, ha összehasonlítjuk a forráskód fordításának hagyományos módszerével.

A hagyományos folyamat szerint a forráskódot a fordítóprogram (compiler) alakítja át a gépnek megfelelő utasításokká, amiket közvetlenül a processzor hajt végre. Egy x86-os sorozatú processzor számára lefordított program egy PPC processzoron hibát eredményez, ha a programot előtte a processzornak megfelelően nem fordítjuk újra. Mindez annyiban nehezíti egy program több eszközfelületre való megvalósítását, hogy minden egyes különböző változathoz rendelkeznünk kell a megfelelően fordított kóddal.

A másik lehetőség, hogy egy olyan köztes futtatókörnyezet számára fordítjuk le a forráskódot, amelynek az utasításai függetlenek a háttérben működő eszköztől. A köztes utasítások ezután különböző módszerekkel hajthatók végre. Az egyik módszer, hogy egy értelmezőprogramot (interpreter) használunk. Az értelmezőprogram betölti az utasításokat, majd végrehajtja őket, mintha egy virtuális számítógép működne a háttérben. A második módszer szerint a köztes utasítások a gépnek megfelelő utasításokra való JIT-fordítása futásidőben vagy a telepítéskor történik, s az utasítások ezután közvetlenül hajtódnak végre. Mivel a JIT-fordítás a futtatófelületnek megfelelő utasításokat alkalmaz, a fordítás a célpro-

folytatás az előző oldalról

Minden programnak osztályokból kell felépülnie.

A lista tetején egy megjegyzésrész látható. A C# ugyanazt a megjegyzésformátumot használja, mint a Java vagy a C++. A többsoros magyarázatok használatához a kezdet jelzésére a /* jelet, a végén pedig a */ jelet használjuk. Egysoros megjegyzések esetén a kettős perjel (//) szintén elfogadott.

A megjegyzés alatt nem találunk fejállományt (header), mint a C vagy C++ esetén. Láthatunk viszont egy utasítást, ami a Javából ismert `import` utasításra emlékeztet. A `using` kulcsszó a C#-ben egy névtér megadására alkalmazható, ami lényegében egy osztálygyűjtemény. Ebben az esetben a `System` az az osztálykönyvtár-névtér, amit megadunk. A C# egy kis- és nagybetűket megkülönböztető nyelv, így a `System` nem azonos a `system`-mel. Szintén a Javához hasonló az a tulajdonsága, hogy az utasítások pontosvesszővel zárulnak.

Lefelé haladva a következő sor a legkülső osztály kezdete. A C# fájlonként több osztály megadását is megengedi, így a fájl nevének nem kell a legkülső osztály nevével megegyeznie, mint a Javában; ezzel szemben a fájl nevének *.cs* kiterjesztést kell kapnia.

A `Hello World` osztály nyitó kapcsos zárójel utáni első sora a `Main()` tagfüggvény. Minden programnak rendelkeznie kell `Main()` tagfüggvénnyel, ami a program belépési pontjának a szerepét tölti be. A C++ és Java nyelvektől eltérően a `Main()` nagybetűvel kezdődik. A fő eljárás megadásában szerepel még a `public static void` kifejezés. A `public` beállítja az eljárás láthatóságát és elérhetőségét (lényegében bárki hozzáférhet). A `static` kulcsszó lehetővé teszi, hogy még azelőtt belépjünk az eljárásba, mielőtt egy `Hello World` típusú objektumot hoznánk létre. Végül a `void` kulcsszó azt adja meg, hogy az eljárás nem rendelkezik visszaadott értékkel. A C#-ben a `Main()` tagfüggvény vagy `int` típusú értékkel tér vissza, vagy semmilyen értéket nem ad vissza.

A `Main()` tagfüggvény belsejében az alábbi ciklus helyezkedik el:

```
for (int x=0; x<3; x++){
    // Console from the System namespace
    Console.WriteLine("Hello World");
}
```

A `for` ciklus formátumának a C, C++ vagy Java-programozók számára már ismerősnek kell lennie. A ciklus első része, az `int x=0` egy `x` nevű egész típusú változót ad meg, és a `0` értéket adja neki. Ez a változó tölti be a ciklusszámláló szerepét. A ciklus középső része, az `x<3` kifejezés az a logikai feltétel, ami eldönti, hogy mikor kell a ciklusnak befejeződnie. Esetünkben, ha az `x` értéke `3`-nál nagyobb, a vizsgálat eredménye negatív, kilépünk a ciklusból. A ciklus utolsó része a frissítési szakasz. A C++-hoz hasonlóan a C# is megengedi az utólagos (post) növelést és csökkentést. Az `x++` parancs az `x=x+1` rövidítése. A frissítő szakasz minden ciklusvégrehajtás után lefut. A példában a ciklus 3-szor hajtódik végre.

A C# a `while`, a `do-while`, a `for` és `foreach` típusú ciklusokat támogatja.

A `for` ciklus belsejében a `Console.WriteLine()` tagfüggvény hívása található. A tagfüggvény hivatalos neve `System.Console.WriteLine()`, de mivel kikötöttük, hogy a `System` névteret használjuk, a `System` kezdőrészt elhagyhatjuk. Hosszú programokban ezzel a fogással jelentős mennyiségű gépeléstől kímélhetjük meg magunkat, de ha meggondolatlanul használjuk, bonyolulttá válhat annak a meghatározása, hogy éppen melyik névteret használjuk. A `Console.WriteLine()` tagfüggvény a kimeneti szöveget a rendszerkonzolra irányítja. A példaként vizsgált kód végén lévő kapcsos zárójelek lezárják a ciklust, a tagfüggvényt, majd az osztályt.

A lista például egy *hello.cs* nevű szöveges fájlba menthető, majd a Mono C#-fordítójával, az `mcs`-sel lefordítható. Az eredményként kapott *hello.exe* futtatható fájl az alábbi eredményt szolgáltatja:

```
[jdq@newton mono]$ mcs hello.cs
Compilation succeeded
[jdq@newton mono]$ mono hello.exe
Hello World
Hello World
Hello World
```

Természetesen a C# sokkal többet érdemel annál, mint amit ebben a bevezetőben elmondhattunk róla. A *Kapcsolódó címek* részben néhány jó, a C# nyelvet oktató segédanyagot találhatunk.

cesszornak megfelelően alakítható. A JIT-fordító tovább növelheti a futtás sebességét, oly módon, hogy csak a felhasználásra kerülő utasításokat fordítja le, majd a memóriában tárolja őket a későbbi hívások számára.
A futtatókörnyezet használatának felületfüggetlenségéért a eredményt a futtatási sebesség terén kell köttetnünk. Az eszközfüggő kódra történő fordítás hagyományos módszeréhez képest a végrehajtás lassabb. A sebességkülönbség mértéke függ az adott környezettől és a használt futtatási módszertől. Általában az értelmezőprogram használata igényli a legnagyobb futtatási időt. A JIT-fordítást alkalmazó eljárás sebessége jóval közelebb áll a hagyományos fordítási módszeréhez, mivel mindkettő natív (az adott környezetnek megfelelő) utasításokat eredményez. A futásidejű többletmunka azonban a sebességben mégis eredményez egy kis lemaradást.
Most arra gondolhatunk: egy objektumközpontú nyelv, alapeljáráskönyvtár, futtatókörnyezet – mintha csak a Javáról lenne szó. Ez igaz is – a CLI alkotóelemei nagyon hasonlítanak a Javában lévőkhöz, mégis van egy alapvető különbség. A Java futásidejű része kizárólag a Java nyelv futtatására lett kifejlesztve. Igaz ugyan, hogy néhány más nyelvet is átalakítottak olyan módon, hogy a kimeneti kódja egyezzen a Javáéval, és ezáltal a JVM (Java Virtual Machine, azaz Java virtuális gép) számára futtatható legyen, de ez még mindig nem üti meg a CLI által biztosított nyelvfüggetlenség mértékét. Hiszen a CLI-t éppen arra tervezték, hogy számos programozási nyelv futtatókörnyezete lehessen. A CLI adattípus-rendszere az imperatív nyelvek (mint a C vagy a Pascal) és az objektumközpontú nyelvek támogatására egyaránt képes.
A CLI nemcsak arra képes, hogy többféle nyelven írt programot futtasson (nyelvfüggetlenség), hanem e nyelvek számára az egymás közti adatmegosztáshoz is alapot nyújt (nyelvek együttműködése), ideértve a többnyelvű kivételkezelést is. Az egyik nyelven létrehozott objektumnak a másikban is lehetnek leszármazottai. Ha megvizsgáljuk a CLI alapvető összetevőit, megtudhatjuk, hogy milyen módszerekkel érhető el a nyelvi függetlenségnek ez a magas foka.

A CTS

A CLI lelke a közös típusrendszer, a CTS (Common Type System). A CTS egy osztott típusrendszert határoz meg, amely rögzíti az adatok megadásának, használatának és kezelésének a szabályait. Ennek a szigorúan kötelező típusrendszernek az alkalmazásával a CLI képes biztosítani a típusok épségét, és a nyelvek számára lehetővé teszi a másik nyelv adattípusainak kezelését. A nagyszámú különböző programozási nyelv egyeztethetősége érdekében a CTS két fő adattípust biztosít, amelyek több altípussal, értékkel (értéktípussal) és objektummal (hivatkozástípussal) rendelkeznek. Az értékek az olyan egyszerű adattípusok számára vannak fenntartva, mint az egész és lebegőpontos értékek. Az objektumok a programozási nyelvek számára szükséges összetettebb egyedek leírására használatosak.

A CLS

A CLS (Common Language Specification, közös nyelvleíró) egy olyan keretrendszert ír le, amelyhez a fordítóprogramoknak a nyelvek közti adatcsere érdekében létrehozott programkönyvtárak és bináris állományok előállításakor ragaszkodniuk kell. A CLS valójában a CTS részegysége, ami ésszerű adattípus- és szabályrendszert ad egy nyelv fordítóprogramja számára annak érdekében, hogy az így létrejövő lefordított kódot más nyelvek is képesek legyenek használni, illetve kiegészíteni. Egy nyelv rendelkezhet arról, hogy milyen mértékben támogatja a CLS-t. Azok a nyelvek, amelyek bármely CLS-típus használatát lehetővé teszik, az úgynevezett CLS-fogyasztók. Azok, amelyek képesek CLS-típusok létrehozására illetve kiterjesztésére, a CLS-kiterjesztők. Végül azok a nyelvek, amelyek teljes mértékben magukban foglalják a CLS-t, egyszerre fogyasztók és kiterjesztők.

1. *lista* Részlet egy visszafordított C#-programból a CIL utasításkészlet bemutatására

```
// Eljárás 1 sor
.method public hidebysig specialname
        ➥rtspecialname
        instance default void .ctor() cil
        ➥managed
{
    // Az RVA 0x20ec-nél kezdődő eljárás
    // Kódméret 7 (0x7)
    .maxstack 8
    IL_0000: ldarg.0
    IL_0001: call instance void valuetype /
        ➥[corlib]System.Object::.ctor()
    IL_0006: ret
} // end of method instance default void
  // .ctor()

// Eljárás 2 sor
.method public static
        default void Main()  cil managed
{
    // Az RVA 0x20f4-nél kezdődő eljárás
    .entrypoint
    // Kódrészelet 11 (0xb)
    .maxstack 8
    IL_0000: ldstr "Hello World"
    IL_0005: call void class /

[corlib]System.Console::WriteLine(string)
    IL_000a: ret
} // Vége a void Main() eljárásnak
```

A leíró adat és a CIL

Amikor egy forrásfájlt egy CIL- (Common Intermediate Language, közös közvetítő nyelv) megfelelő fordítóprogrammal fordítunk le, kimenetként egy hordozható futtatható bináris állomány (PE – portable executable) jön létre, amit néha assemblyként is emlegetnek, bár az assembly egynél több fájlból is állhat. A PE két fontos adatrészből tevődik össze. Az első a leíró adat, ami tartalmazza a használt típusok leírását, a CLI által az osztályok előkereséséhez és betöltéséhez használt adatokat, a memóriaelrendezést és egyéb, futásidőben szükséges adatokat. A második rész a közös közvetítő nyelven írt CIL-kód. A CIL a közvetítőutasítások nyelvektől független gyűjteménye. Amikor lefordítunk egy nyelvet a CLI számára, ez a CIL-kód jön létre. A CIL megfelelő teljesítménnyel rendelkezik ahhoz, hogy rengeteg különböző programozási nyelvet kezelhessen, és úgy tervezték, hogy nagy hatékonysággal lehessen a felületnek megfelelő natív utasításokra lefordítani. A CIL-utasítások egy kis része látható az *1. listán*, a Hello World program C#-ben írt változatában.

2. lista A ljlib.cs fájl tartalma

```
using System;

namespace LJlib {

   public class Output
   {
      static public void SayHello ()
      {
         Console.WriteLine("Hello Linux
                        ➥Journal!");
      }
   }
}
```

3. lista A hello.vb fájl tartalma

```
Imports LJlib

Module modmain
   Sub Main()
      Output.SayHello()
   End Sub
End Module
```

A VES

A virtuális futtatórendszer (Virtual Execution System – VES) teremti meg a CLI-re írt programok számára a megfelelő futtatókörnyezetet. Végrehajtja a betöltést, a modulok egyesítését, vezérli a memóriát, megvalósítja a biztonsági szolgáltatások és kivételek kezelését, valamint megfelelő hátteret nyújt a CIL-utasítások végrehajtásához.
A CLI memóriakezelő része tartalmaz egy szemétgyűjtőt (Garbage Collector – GC) is. Más futtatókörnyezetekkel szemben, a CLI megengedi, hogy a forráskódban engedélyezzük vagy tiltsuk a GC használatát. A GC által kezelt adatokat (lefoglalás, felszabadítás) felügyelt (managed) adatoknak nevezzük, ha pedig a GC tiltva van, felügyelet nélküli adatokról beszélünk. A felügyelt kód (a CLI által végrehajtott forráskód) kezelhet felügyelt és felügyelet nélküli adatokat is.

A Mono

A Mono-projekt kettős célt szolgál. Az első, hogy az ECMA CLI- és C#-szabványainak gyakorlati megvalósítását nyújtsa. A második, hogy mindezt a Microsoft .NET fejlesztői felülettel együttműködő módon tegye. Mindkét résznek megvan a maga értéke, és különböző módon szolgálják a Linux javát. Ha például a .NET-megfelelőség megszűnne, a Mono még akkor is a Linux egyik értékes fejlesztői keretrendszere lenne. Mindamellett a Mono azáltal, hogy a linuxos felületet .NET-megfelelővé teszi, futtathatóvá teszi Linuxon a Windows alá fejlesztett programokat. A gondolatmenetet folytatva az is elmondható, hogy a fejlesztők számára immár rendelkezésre fog állni egy megszokott fejlesztői keretrendszer, amely a linuxos programok fejlesztésére történő átállás esetén csökkenti a tanulási kényszerből adódó korlátokat.
A .NET-nek azok a fő részei, amelynek a kiadásán a Mono projekt éppen dolgozik, a Win Forms (System.Windows.Forms), az ADO.NET és az ASP.NET. A Win Forms tartalmaz minden szükséges eljárást, osztályt és eseményt a Microsoft Windows-rendszernek megfelelő grafikus programok kifejlesztéséhez. Mivel a natív Linux grafikus eszközkészlettel szinte lehetetlen a Windows GUI API-hívásokat emulálni, a Mono a WineLib (➲ http://www.winehq.com) segítségével adja a Windows-felületet. Ha már láttunk alkalmazást a Wine alatt futni, akkor tudjuk, hogy a kinézete egyáltalán nem hasonlít a Linux asztali környezeteire. Ennek megoldására a Wine-ban a Mono gondoskodik a témák támogatásáról, hogy ugyanazokat a leképező eljárásokat használja az elemkészletekhez, mint az asztal egyéb részeinél.
Az ADO.NET tartalmazza a Mono számára a .NET adateléréssel kapcsolatos osztályait. Az ADO.NET többre képes egyszerű adatelérésnél: kapcsolat nélküli, méretezhető, XML-en alapuló adatelérési modellt nyújt bármely adatforrásból. Az írás idején körülbelül egy tucatnyi adatbázis működik a Mono ADO.NET adatszolgáltatójaként. A program fejlődéséért, az újabb és újabb gyártó adatbázisának támogatásáért végzett munka folytatódik.
A Monóban az ASPI.NET támogatása két részre lett bontva: a webűrlapokra és webszolgáltatásokra. A webűrlapok alkotják a webalkalmazások felhasználói felületét. A Win Formshoz hasonlóan a webűrlapok is nyújtják a vezérlőeszközökhöz – gombokhoz, szövegdobozokhoz és egyszerűbb vezérlőelemekből álló összetettebb elemekhez – tartozó tulajdonságokat, eljárásokat és eseményeket. Mindez lehetővé teszi, hogy a webes űrlap felülete RAD-eszköz (Rapid Application Development, gyors alkalmazásfejlesztő) segítségével készüljön, a fogd és vidd módszer alkalmazásával, a Gnome Glade programjához hasonlóan. Ezáltal a megjelenítés elválik a program logikájától, csökkentve a szükséges kódolás mennyiségét. A webes szolgáltatások egy SOAP alapú távoli eljáráshívás támogatását kínálják. Az olyan, mindenütt jelen lévő internetes protokollok használatával, mint az XML- és HTTP-szolgáltatások, az adatok és az eredmények hálózatos megosztását teszik lehetővé, még tűzfalakon keresztül is. Az ASP.NET a CLI által támogatott bármelyik nyelven programozható. Ez azt is jelenti, hogy az ASP.NET kódja lefordításra kerül, és nem értelmezőprogram segítségével fut, mint az ASP korábbi változatai vagy egyéb webes parancsnyelvek. Az ASP.NET a Mono számára akár az XSP webkiszolgálón, akár az Apache 2 `mod_mono` összetevőjével elérhető.
A .NET megvalósításában részt vevő Mono-osztálykönyvtárakon túl számos más programkönyvtár és eszköz is érdekes szolgáltatásokat kínál:

- A GTK#, Qt# és Wx.NET a népszerű linuxos grafikus eszközkészletekhez nyújt C#-kapcsolatot. Ezekkel a C#-burkolókkal minden, Monón futtatható nyelv ugyanazokhoz a grafikus eszközökhöz nyer hozzáférést.
- Az OpenGL#, MonoGLo és CsGL a népszerű két- és háromdimenziós API OpenGL-hez ad kapcsolást.
- Az SDL.NET az SDL programkönyvtárhoz történő kötést biztosítja.
- A Gst# Gstreamer multimédiakeretrendszer-kötés.
- Számos kommunikációs programkönyvtár, például a .NET Jabber és a Gnutella.
- NAnt fordítóeszköz (az Anthez hasonló eszköz).

Természetesen ez csak néhány példa, de elég ahhoz, hogy szemléltesse a Monónak azokat a képességeit, amelyek Linux vagy egyéb felületre történő fejlesztés esetén igénybe vehetők.

A Mono használata

Az első lépés a Mono kipróbálásához a ➲ http://www.go-mono.com címen a projekt honlapjának a meglátogatása, ahonnan letölthetjük a forrás tarcsomagjait, vagy a felületünknek megfelelő bináris állományokat. Pillanatnyilag a Monónak Linuxon és Windowson futtatható átirata létezik, de folyamatban van a MacOS X, FreeBSD és más felületekre történő átírás is. Számos Linux-változatra létezik bináris állomány, többek közt Debianra, Red Hatre, SuSE-ra és Mandrake-re. Ha a Ximian Red Carpetet használjuk, a neki megfelelő fájlokat a Mono Chanellen is fellelhetjük. A cikkhez a Mono 0.20-as változatát használtuk. Észrevehetjük majd, hogy a Mono programcsomagokon felül – amelyek a futásidejű, a C# fordító- és osztálykönyvtárakat tartalmazzák – további csemegéket is kapunk. Ezek között találjuk a Mono hibakereső programját, az XSP webkiszolgálót és a Monodoc leírásböngészőt. Amennyiben gondjaink akadnának a Mono telepítésével, forduljunk a honlapon elérhető leírásokhoz.

A Mono jelenleg a következő összetevőkkel érkezik:

- C#- és Basic-fordítóprogramok.
- A VES, ami egy JIT-fordítóból és a hozzá tartozó hulladékgyűjtőből, a biztonsági rendszerből, az osztálybetöltő, ellenőrző és végrehajtó rendszerből áll. Egy értelmezőprogram szintén része ennek az összetevőnek.
- Egy C#-ben írt osztálykönyvtár-gyűjtemény, amely a CLI-szabványban meghatározott osztályokat, a .NET FDL részét képező osztályokat és más Mono által használt osztályokat valósít meg.
- Különböző segédprogramok.

Az `mcs` a Mono C#-fordítóprogramja. Érdekes programozói bravúr, hogy az mcs-t C#-ben írták. A Mono 0.10 óta az `mcs` önmaga fordítására is képes. Ha érdekelnek bennünket az `mcs` parancssori lehetőségei, amelyek megegyeznek a Microsoft C#-fordítója által kínált kapcsolókkal, részletes súgóoldalak állnak rendelkezésünkre.
A Visual Basic.NET Monóban található megfelelőjének, a MonoBasicnek a fordítóprogramja az `mbas`. Bár a fejlesztése még nincs a C#-fordítóéhoz hasonló állapotban, a Basickel való kísérletezgetéshez már megfelelő szolgáltatásokkal rendelkezik.
A Mono két futtatókörnyezete a `mono` és a `mint`. A `mono` a VES CLI meghatározásaival egy JIT-fordítónak megfelelő környezet. Ezzel ellentétben a `mint` egy értelmezőprogram, ami a `mono` számára könnyen hordozható megoldást jelent, de pillanatnyilag csak az x86-os felületet támogatja. A legjobb futási teljesítmény a `mono` használatával érhető el.
A Monóval érkező segédprogramok közül figyelemre érdemes a `monodis` és a `pedump`. A `monodis` egy lefordított assembly visszafejtésére alkalmas, kimenete a megfelelő CIL-kód. Ezt használtuk az *1. listán* látható CIL-kód megjelenítésére. Ha kíváncsiak vagyunk a CIL további részleteire, vagy bepillantást szeretnénk nyerni a hordozható futtatható kód előállításába, játszadozzunk el velük egy kicsit.
Most, hogy már ismerjük a Mono összetevőit, itt az idő, hogy ki is próbáljuk őket. A Mono nyelvi párbeszédének próbájához C#-ben egyetlen eljárással egy egyszerű osztályt írunk, majd egy MonoBasic-programból meghívjuk.
A *2. lista* mutatja a ***ljlib.cs*** C#-könyvtárat, a *3. listán* pedig a ***hello.vb*** MonoBasic-program látható. Az első lépés a ***ljlib.cs*** programkönyvtárrá történő fordítása. A lefordított programkönyvtárak ***.dll*** kiterjesztéssel rendelkeznek, a futtatható állományok kiterjesztése pedig ***.exe***.

- Programkönyvtár fordításához az `mcs -target:library` kapcsolóját használjuk:
  `[jdq@newton]$ mcs -target:library ljlib.cs`
- `Compilation succeeded`
  Ennek eredménye a ***ljlib.dll*** fájl, ami a `Ljlib` névteret és `Output` osztályt tartalmazza. Most a ***hello.vb*** program fordítása következik. Ahhoz, hogy a fordításkor az éppen előállított ***ljlib.dll*** fájl kerüljön felhasználásra, utasítanunk kell a MonoBasicet, hogy hivatkozásként ezt használja. Ezt a `-r` kapcsolóval tehetjük meg:
  `[jdq@newton]$ mbas -r ./ljlib.dll hello.vb`
- `Compilation succeeded`
  Az mbas kimenete a ***hello.exe*** fájl. Ezt a mono segítségével futtathatjuk is:
  `[jdq@newton]$ mono hello.exe`

Hello Linux Journal!

És íme: két nyelv, a C# és a MonoBasic egy futtatható állományban egy időben működik. Bár ez egy végtelenül egyszerű példa, mégis jól mutatja a CLI nyelvfüggetlenségét és együttműködési képességét, és jelzi a Monónak fejlesztőeszközként rendelkezésre álló széles körű lehetőségeit.

Összegzés

Bár a Mono még fejlesztés alatt ál, már így is a Linux programfejlesztése elősegítésének nagy ígérete. Ha az utóbbi két évben mutatott fejlődését vesszük alapul, a Mono jövője nagyon izgalmasnak ígérkezik.

Linux Journal 2003. július, 111. szám

Julio David Quintana (jdq@jdqi.com)
Villamosmérnök. 1997-ben találkozott a Linuxszal, és azóta nem is tud szabadulni tőle. Ha nyelvtani vagy tárgyi hiba kapcsán keressük, akkor nem érhető el, de a dicséreteket és a jókívánságokat szívesen fogadja.

KAPCSOLÓDÓ CÍMEK

A C# összehasonlító áttekintése
➲ http://genamics.com/developer/csharp_comparative.htm

A C# a Java-fejlesztők szemszögéből
➲ http://www.25hoursaday.com/CsharpVsJava.html

A C# Stations C#-oktató
➲ http://www.csharp-station.com/Tutorial.aspx

Az ECMA CLI-szabványa
➲ http://www.ecma-international.org/publications/standards/ECMA-335.HTM

Az ECMA C# nyelvre vonatkozó szabvány
➲ http://www.ecma-international.org/publications/standards/ECMA-334.HTM

A Mono projekt honlapja ➲ http://www.go-mono.com

A Softsteel C#-oktatója és „patchwork” könyve
➲ http://www.softsteel.co.uk/tutorials/cSharp/cIndex.html