# Beágyazott Java GCJ-vel

Nincs feltétlenül szükség Java virtuális gépre ahhoz, hogy beágyazott rendszereken Java programot futtassunk.

Az alábbiakban bemutatjuk, hogyan kell használni a *GCC* fordítóprogramcsomag részét képező *GCJ*-t beágyazott rendszerekhez *Linuxon*. Mint minden eszköznek, a *GCJ*-nek is van olyan előnyös tulajdonsága, mely egyben a hátránya is, nevezetesen, hogy egy magas szintű nyelven, *Java*-ban programozhatunk. A *GCJ* beágyazott rendszereken való futtatásának ötlete először ijesztő lehet, de a cikk végére kiderül, hogy egyszerűbb mint gondolnánk. Reményeim szerint a cikk végére az Olvasó kellően felcsigázott lesz, hogy kipróbálja a tanultakat, és meggyőződjön róla, hogy érdemes a *GCJ*-re gondolnia a legközelebbi munkája során. A *Java* nyelv remek eszközkészlettel rendelkezik ahhoz, hogy gyorsan fejleszthessünk megbízható szoftvereket. Ennek része pl. az automatikus memóriaszemét-gyűjtés, a sokrétű és robusztus futásidejű könyvtár és a rendkívül kifejező objektum-orientált szerkezetek.

## Miért pont a GCJ?

A natív *Java* fordító pontosan azt csinálja, amit a neve sugall: a *Java* forráskódot az adott hardver gépi kódjára alakítja át. Ebből az is következik, hogy az adott gépen nincs szükség *Java* virtuális gépre (*Java Virtual Machine – JVM*), a program futtatásakor nem töltődik be *JVM*, pontosan úgy töltődik be és fut le, ahogy bármely más végrehajtható program. Ez sajnos nem szükségszerűen jelenti azt is, hogy a program gyorsabban fut majd. Előfordulhat, hogy a *GCJ* által fordított kódnál jobb eredményeket kapunk egy natív virtuális gépen futó bájtkóddal.
A *GCJ* egyik előnye az, hogy mivel nincs szükség *JVM*-re, helyet takarítunk meg, továbbá pénzt takaríthatunk meg a jogdíjak miatt is. Mindezen túl, a *GCJ*-vel kizárólag nyílt forráskódú szoftverekkel állíthatunk elő új terméket, ami általában nyerő dolog.

## Buktatók

Mikor a beágyazott rendszerekkel foglalkozó mérnökök gyökérfájlrendszert hoznak létre az adott platformon, az első dolguk az *uClibc*, egy kompakt *glibc* függvénykönyvtár telepítése. Azok kedvéért, akik nem járatosak a *Linux* beágyazott rendszereken való használatában, a standard *C* függvénykönyvtár óriásnak számít egy olyan környezetben, ahol a gyökérfájlrendszer pl. Legfeljebb 8 MB lehet. Helytakarékosság okán, a fejlesztők a standard *C* függvénykönyvtárat egy kisebbel váltják fel, pl. az *uClibc*-vel. Mivel a *GCJ* megköveteli az *Unicode* támogatását, a *glibc* nélkülözhetetlen, hiszen az *uClibc*-ben ilyen nincs.
A standard függvénykönyvtár a *GCJ*-nek 16 MB-ba kerül, így még ha tudnánk is kisebb standard *C* könyvtárra váltani, nagy különbséget nem jelentene. A standard *GCJ* könyvtárból eltávolíthatnánk a *Java* bájtkódok végrehajtásának támogatását, de összességében a *GCJ* hasznossága ettől jelentősen csökkenne.

## A hoszt és a célplatform beállítása

Minthogy e cikk a *GCJ* beágyazott rendszereken való használatával foglalkozik, azt is bemutatja, hogyan készítsünk keresztfordítást és egyszerű gyökérfájlrendszert a célplatformon. Az újoncok kedvéért, a keresztfordító olyan végrehajtható kódot állít elő, amely a fordítást végző géptől különböző célplatformon futni képes.
A fordítót futtató gépet hosztnak, a kódot futtató gépet pedig célplatformnak nevezzük.
Esetünkben a célplatform egy 350 MHz-en ketyegő *PPC 745/755* alapú gép. Ezt az alaplapot áttetsző dobozban árulják, merevlemezzel és kijelzővel, és *iMac* néven is szokás emlegetni. Belátom, ez nem a legjobb példa egy beágyazott rendszerre, de ugyanolyan problémákkal foglalkozik, amelyekkel a valódi beágyazott rendszereken is szembesülünk. Az itt elsajátított ismeretek jól hasznosíthatók lesznek más processzoroknál is.
A hoszt egy teljesen átlagos *IBM ThinkPad* noteszgép lesz, *Pentium III*-as processzorral. A gépen *Yellow Dog Linux* fut, amit majd kicsit módosítunk, hogy a cikkben készített gyökérfájlrendszert használja.

## Helyezzük üzembe a GCJ-t

Először is, szükségünk lesz egy olyan keresztfordítóra, amely egyrészt futtatható a *Pentium III*-as gépen, másrészt képes a *PowerPC 750* processzoron futó kód előállítására. A keresztfordító készítése a célplatformra nagyon macerás lehet; egy működő *GCC* még nem feltétlenül jelenti azt, hogy használható fordítónk van. Néhány további eszközre is szükségünk lesz, például a *binutils*-ra és az adott nyelvhez tartozó standard függvénykönyvtárakra.
Ha gyorsan és fájdalommentesen akarunk keresztfordítóhoz jutni, használjuk a *crosstool* csomagot, *Dan Kegel* keze munkáját. A *crosstool* megcsinálja helyettünk az összes nehéz lépést, ami a keresztfordító készítéséhez szükséges: letölti a forráskódot és a javításokat, végrehajtja a javításokat, elvégzi a szükséges beállításokat a csomagon

és végül elkészíti a fordítót. A *crosstool* beszerzése és kicsomagolása után a következő lépéseket hajtsuk végre a *GCJ* keresztfordító létrehozásához:

```
$ export TARBALLS_DIR=~/
➥crosstool-download
$ export RESULT_TOP=/opt/
➥crosstool
$ export GCC_LANGUAGES=
➥"c,c++,Java"
$ eval `cat powerpc-750.dat
➥gcc-4.0.1-glibc-2.2.2.dat'
➥sh.all -notest
```

Míg a fordító teszi a dolgát, vessünk egy közelebbi pillantást az imént kiadott utasításokra. A `TARBALLS_DIR` változó mondja meg, hová töltse a *crosstool* a fájlokat. Alapértelmezés szerint, a *crosstool* a program készítéséhez szükséges összes fájlt letölti.
A `RESULT_TOP` definiálja a keresztfordító telepítőkönyvtárát. Végül, a `GCC_LANGUAGES` változó állítja be, hogy mely nyelvi front-endek kerüljenek be a fordítóba. A *GCC* rengeteg nyelvet támogat, és minden egyes front-end fordítása jelentősen megnöveli a fordítási időt, ezért a `GCC_LANGUAGES` változó csak azokat tartalmazza, amelyekre szükségünk van.
A *Bash* parancsnyelvében járatlanok kedvéért, az utolsó sor a képernyőre írja a két *.dat* fájl tartalmát, majd végrehajtja az *all.sh* parancsfájlt a `--notest` kapcsolóval. Az egyszerűség érdekében, a *crosstool* eleve rendelkezik azokkal a konfigurációs fájlokkal, amik a célprocesszorhoz és a *gcc*/*glibc* kombinációhoz szükséges környezeti változókat tartalmazzák. Esetünkben a *crosstool gcc 4.0.1*-et készít *glibc 2.2.2*-vel, *PPC 750* processzorra. A *crosstool* az összes ismert processzor és *glib*/*gcc* kombinációhoz rendelkezik konfigurációs fájlokkal. Az összeállítás után a keresztfordítót a *$RESULT_TOP*/*gcc-4.0.1-glibc-2.2.2*/*powerpc-750-linux-gnu*/*bin* könyvtárban találjuk. Legjobb, ha ezt az útvonalat rögtön a `PATH` környezeti változóhoz adjuk, hogy a fordító bárhol könnyen elindítható legyen.

## A crosstool beszerzése és kicsomagolása

A *crosstoll Dan Kegel* munkájának gyümölcse. A ➲ *kegel.com/crosstool* címen mindent megtudhatunk a *crosstool*-ról, amit tudni szeretnénk. A teljes dokumentáció mellett egy kiváló gyorstalpalót is találunk. A cikk írásakor a ➲ *kegel.com/crosstool/crosstool-0.38.tar.gz* címen lévő 0.38-as verziót használtam.
A *crosstool* honlapján feltétlenül ellenőrizzük az összeállítások naplóját (➲ *kegel.com/crosstool/crosstool-0.38/buildlogs*), hogy tudjuk, a *glibc*/*gcc* milyen kombinációi fordulnak sikeresen a célplatformhoz.

## A gyökérfájlrendszer beállítása

Újonnan készített keresztfordítónk első feladata a gyökérfájlrendszer fordítása lesz, amely ebben az esetben a *BusyBox* része. Azért, hogy a beavatatlanok is tudják, a *BusyBox* egy olyan, megdöbbentően kis méretű, végrehajtható bináris fájl, amely magában foglalja a legnépszerűbb *UNIX*-eszközöket, kifejezetten azoknak, akinek minden bájt számít. A *BusyBoxon* gombok százai állnak nyomásra készen, hogy létrehozhassunk kényszerű megkötéseink fájlrendszerét, a szükséges eszközök támogatásával. A példa kedvéért most keresztfordításhoz állítjuk be a *BusyBox* konfigurációját, a méretre történő optimalizálást pedig gyakorlatként az Olvasóra bízom.
A *BusyBox* a beágyazott *Linux* világának egyik fő tartópillére, melynek karbantartását *Erik Anderson* végzi. Letölthető a ➲ *www.busybox.net/downloads/busybox-1.01.tar.bz2* címről. A gyökérfájlrendszer létrehozásához adjuk ki a `make menuconfig` parancsot abban a könyvtárban, ahová a *BusyBoxot* kicsomagoltuk. Ez pontosan úgy működik, ahogy a 2.4/2.6 rendszermagok beállításának felhasználói felülete. Az alábbiakban bemutatom, mit kell tenni a gyökérfájlrendszer fordításához.
Először is, válasszuk ki az összeállításhoz szükséges kapcsolókat. Jelöljük ki a *„Keresztfordítóval szeretné összeállítani a BusyBoxot?”* (*Check the Do you want to build BusyBox with a Cross Compiler?*) feliratú jelölőnégyzetet. A felbukkanó szövegbeviteli mezőbe írjuk be a keresztfordító előtagját, ami esetünkben *powerpc-750-linux-gnu-*. A *BusyBox* parancsfájljai a fordítás közben ezzel fűzik össze a szükséges eszközök nevét (például `gcc` és `ld`). Győződjünk meg róla, hogy a fordító útvonala benne van a `PATH` környezeti változóban, és adjuk ki az alábbi parancsokat:

```
make
make install
```

Az új fájlrendszer a `./_install` könyvtárba kerül. Biztosan feltűnik majd, hogy a *BusyBox* lényegesen gyorsabban fordul, mint a *GCC*.

## A gyökérfájlrendszer benépesítése függvénykönyvtárakkal

Már majdnem kész vagyunk, de az új fájlrendszerünk még teljesen üres, függvénykönyvtárak sehol. A *GCJ* programoknak szüksége van bizonyos könyvtárakra, így a *BusyBoxnak* is, ahogy ez az *1. Táblázatban* látható. Ezek pontosan azok a könyvtárak, amelyeket a keresztfordító is használ. Példánkban a fájlok a *$RESULT_TOP*/*gcc-4.0.1-glibc-2.2.2*/*powerpc-750-linux-gnu*/*powerpc-750-linux-gnu*/*lib* (figyelem, nem elírás!) könyvtárban találhatók, és egyszerűen átmásolhatjuk őket a gyökérfájlrendszerbe:

```
for f in ld.so.1 lib libdl.so.2
➥libgcc_s.so.1libgcj.so.6
 -->libm.so.6 libpthread.so.0 ;
➥do

cp
$RESULT_TOP/gcc-4.0.1-glibc-
➥2.2.2/powerpc-750-linux-gnu/
➥powerpc-750-linux-gnu/lib/$f
<busybox install directory>/lib

$RESULT_TOP/gcc-4.0.1-glibc-
➥2.2.2/powerpc-750-linux-gnu/
➥bin/power
pc-750-linux-gnu-strip <busybox
➥install directory>/lib/$f
done
```

A gyökérfájlrendszerben egy */proc* könyvtárat is kell készítenünk, a *proc* fájlrendszer becsatolási pontjának. Az éles szemű Olvasó bizonyára észrevette, hogy nem őriztem meg a könyvtárak különféle verzióinak kezelésére létrehozott szimbolikus linkeket – ez elterjedt gyakorlat a beágyazott rendszereknél, ahol a konfiguráció az eszköz élettartama során sohasem változik meg, eltérően az asztali gépektől. A `strip` futtatása jelentősen, mintegy 50 %-kal csökkenti a szükséges merevlemez-területet.

1. táblázat *A GCJ-hez és BusyBoxhoz szükséges könyvtárak*

| Könyvtár | Leírás |
|---|---|
| *ld.so.1* | A programok futásához szükséges dinamikusan kapcsolt fájlok betöltéséről és kapcsolásáról gondoskodik. |
| *libdl.so.2* | A dinamikusan kapcsolt fájlok manipulálásához szükséges segédfüggvények gyűjteménye. |
| *libgcc_s.so.1* | A kivételkezeléshez szükséges programozói interfészt definiálja. |
| *libgcj.so.6* | A GCJ futásidejű könyvtára, amely a standard Java könyvtárak megvalósítását tartalmazza. |
| *libm.so.6* | Matematikai függvények könyvtára. |
| *libpthread.so.0* | POSIX-szálak függvénykönyvtára. |

A gyökérfájlrendszert most már átmásolhatjuk a célplatformon lévő ***bbox*** könyvtárba. A rendszert a `chroot` paranccsal vehetjük rá, hogy ezt használja gyökérfájlrendszerként. Rendszergazdaként indított parancssorból adjuk ki a következő utasítást:

```
chroot /tmp/bbox /bin/ash
```

Ennek hatására a / becsatolási pont a */**tmp**/**busybox*** könyvtárra kerül, és elindul a */**bin**/**ash*** parancsértelmező. Működik? Gratulálok! Éppen most hoztunk létre egy gyökérfájlrendszert egy beágyazott rendszerhez, egészen az alapoktól. Nyugodtan veregesse meg a vállát mindenki!
***GCJ***-nek arra is szüksége van, hogy a ***proc*** fájlrendszert becsatoljuk a jelenlegi gyökérfájlrendszerbe, amit a chroot után a következő parancsok végrehajtásával oldhatunk meg:

```
mkdir /proc
mount -t proc none /proc
```

Igaz ugyan, hogy a mi gyökérfájlrendszerünk egy teljesen szokványos merevlemezen van, de nem sok különbséget találnánk, ha mindezt egy beágyazott rendszeren hajtottuk volna végre. Biztosan lenne két nélkülözhetetlen eltérés: egyrészt létre kell hoznunk az ***inittab***-ot, hogy az alaplap a megfelelő parancsfájlokat hajtsa végre induláskor, másrészt a */**dev*** fájlrendszert is létre kell hozni a célplatformnak megfelelő eszközökkel.

## Fejlesztés GCJ-vel

A keresztfordító és a gyökérfájlrendszer létrehozása után, az első ***GCJ***-alkalmazás létrehozása hihetetlenül egyszerű lesz. A tradicionális helló világ programmal kezdünk:

```
Class hello {
 Static public void main(String
➥argc[]) {
   System.out.println("hello
➥from GCJ");
 }
}
```

A ***Java*** konvenciók szerint ezt az osztályt a ***hello.class*** fájlban találjuk meg. A fordítás így történik:

```
powerpc-750-linux-gnu-gcj
➥hello.class --main=hello -o
➥hello-java
```

Mire jó a `--main=hello`? Bármely osztály definiálhat metódust egy alkalmas belépési ponttal. A `--main=hello` azt mondja a fordítónak, hogy a `hello` osztályban a main metódust használja az összeszerkesztésnél. Ha elhagyjuk, hibát kapunk: nem létező hivatkozás a main metódusra (`undefined reference to main`), ami a kezdők számára zavaró lehet, hiszen a `hello` osztályban van main metódus.
Töltsük fel a fájlt a célplatformra, és futtassuk a `chroot`-tal indított parancssori értelmezőből. Ezt fogjuk látni:

```
# ./java-test
Hello from GCJ
```

Ettől a ponttól kezdve, a fejlesztés a megszokott módon halad tovább, azzal a kivétellel, hogy a natív ***javac*** fordító helyett a ***GCJ*** keresztfordítót használjuk.

## Helytakarékosság

A bemutatott példa gyökérfájlrendszere több mint 20 MB. Mivel a beágyazott rendszerek gyakran használnak ***Flash***-memóriát, melynek fajlagos költsége lényegesen nagyobb, mint a merevlemezes rendszereké, a lehető legkisebb méretű gyökérfájlrendszer általában alapvető követelmény.
A gyökérfájlrendszer méretét legegyszerűbben úgy csökkenthetjük, hogy az alkalmazást statikusan szerkesztjük. Bár az ösztöneink azt súgják, ez pont az ellenkezőjét eredményezi, a ***libc***-nek az alkalmazásba kerülő másolata miatt. Azt se felejtsük el azonban, hogy a ***libgcj.so*** a teljes ***Java*** standard függvénykönyvtárat tartalmazza, aminek a legtöbb alkalmazás csupán töredékét használja. A statikus szerkesztés így kiváló módja a sosem használt kód kirostálásának.
No és a `strip`-ről se feledkezzünk meg, különben megnézhetjük, hány bájtnyi hibakeresési információ lesz belegyömöszölve a ***libgcj.so*** fájlba.

## Zárszó

A cikkből remélhetőleg kiderült, hogy a beágyazott rendszerekre történő fejlesztés ***GCJ***-vel megbízhatóan elvégezhető a jelenleg is elérhető, nyílt forráskódú eszközökkel. Bár néhány trükköt nem árt ismerni, az is nyilvánvaló, hogy a gyökérfájlrendszer konfigurálása nem nevezhető éppen heroikus küzdelemnek; a lényeg, hogy azokból a könyvtárakból, amelyekre egyébként is szükségünk lenne, néhány dolog elhagyható. Láthattuk, hogy a gyökérfájlrendszer mérete jelentősen csökkenthető az alkalmazásunk statikus összeszerkesztésével. Röviden, a ***GCJ*** lehet a mi barátunk, ha erőforrások szűkében lévő rendszeren kell ***Java***-ban fejlesztenünk – mindenesetre, érdemes elgondolkodni ezen a következő feladat előtt.

*Linux Journal 2006., 145. szám*

***Gene Sally*** tíz éve dolgozik Linuxszal, különféle munkák keretében. Gene mostanában a beágyazott rendszerekkel foglalkozó mérnökök segítésére összpontosít. Bátran írjon neki a gene.sally@gmail.com címre, ha kérdése van.