© Kiskapu Kft. Minden jog fenntartva

LSB alkalmazásokat készítése

Ne kössük Linux programunkat egyetlen terjesztéshez. Tegyük mindenhol futtathatóvá egy valamennyi nagyobb terjesztés által használt szabvány segítségével.

A Linux Szabványos Alap (***Linux Standard Base, LSB***) az alkalmazás és a futási környezet közötti felületet szabványosítja. Több terjesztés szerzett már futtatási környezetéhez ilyen bizonyítványt. Ebben a cikkben lépésről lépésre megismerkedhetünk az LSB csatolófelülettel.

Az LSB eredete

Az LSB-Projekt 1997-ben alakult, hogy megoldást találjon az egyre komolyabbá váló kompatibilitási problémákra. A különféle terjesztések eltérő élvonalbeli programverziókat használtak és azokat eltérő opciókkal fordították le. Következésképpen gyakran előfordult, hogy az egyik terjesztésen lefordított program nem futott a másik terjesztés alatt. Még ennél is nagyobb gondot jelentett, hogy időnként az alkalmazások akkor sem működtek, ha ugyanazon terjesztés másik verziójáról volt szó.
Az LSB célja eredetileg az volt, hogy egységes megvalósítási gyűjteményt készítsen, amely majd a GNU/Linux rendszer alapjául szolgálhat. A megvalósítási gyűjteményhez lejegyzett specifikációt készítettek. Sok terjesztés azonban nem fogadta kedvezően ezt az ötletet, ugyanis saját alapprogramjuk fejlesztésébe komoly erőforrásokat fektettek, amit versenyelőnyként értékeltek.
Az érdekelt felek aztán megváltoztatták az LSB Projekt alapvető céljait, hogy ezáltal az egész közösségre nézve elfogadható megoldást találjanak. Ezzel elsődleges szerepet kapott a megvalósítással szemben az írott specifikáció, így az LSB irányadó képesség/verzió párok helyett inkább viselkedési útmutatóvá vált. Az új megoldást három fő ág jellemzi: az írott specifikáció, amely a rendszer viselkedését adja meg; a formális tesztkészlet, amely a specifikációnak megfelelő megvalósítást ellenőrzi; és a minta-megvalósítás, amely a specifikációra mutat példát.

Az LSB szerkezete

Az LSB specifikáció jelenleg egy ***gLSB*** nevű általános részből és az ***archLSB*** gépfüggő részből áll. A ***gLSB*** tartalmazza a gépek (architektúrák) közös jellemzőit; nagyon igyekeztünk mindent a ***gLSB***-ben megadni. Az ***archLSB*** tartalmazza mindazon részleteket, amelyek processzortípusonként eltérőek, például a gépi utasításkészletet és a C könyvtár szimbólum verzióit.

Az LSB tartalma

Amennyire csak lehetséges, az LSB már létező szabványokra épül, ideértve a POSIX-ból kifejlődött Single UNIX Specifikációt (SUS), a ***System V Interface Definition*** (SVID) és a ***System V Application Binary Interface*** (ABI) szabványokat. Az LSB az ABI szabvány ELF definícióit és a SUS csatolófelület viselkedési irányelveit használja. A formális listában egyrészt azt határozza meg, hogy mely csatolófelületek mely könyvtáron keresztül érhetőek el, másrészt pedig a hozzájuk tartozó adatszerkezeteket és állandókat is. A jelenleg érvényben lévő könyvtárak listáját a ***„Linux Szabványos Alapkönyvtárak”*** széljegyzetben találjuk.
Az LSB ABI részén kívül az ajánlás felsorolja az alkalmazásokhoz tartozó parancsfájlokban felhasználható parancsokat is. Egyben azt is kimondja, hogy az alkalmazásnak tartania kell magát a ***Fájlrendszer Hierarchia Szabványhoz*** (FHS).
Az LSB egyik kiegészítője a csomagolási formátum. Az LSB kimondja, hogy a csomagformátumnak az RPM fájlformátum csoport tagjának kell lennie, ugyanakkor nem szabja meg, hogy a terjesztésnek kötelező RPM alapúnak lennie. Mindössze azt köti ki, hogy valamilyen módon helyesen fel kell tudni dolgoznia az RPM formátumú állományokat.
Az utolsó említésre méltó dolog a parancsértelmező neve. Az alkalmazás indításakor a parancsértelmező indul elsőként és felelős az a program többi részének valamint a megosztott könyvtáraknak a folyamat címterébe töltéséért. Hagyományosan a ***/lib/ld-linux.so.2 -t*** használják, de az LSB e helyett a ***/lib/ld-lsb.so.1*** nevet határozza meg az IA32 rendszereken. A ***/lib/ld-arch-lsb.so.1*** nevet általában más géptípusokhoz használják. Ezzel az operációs rendszernek lehetőséget adunk, hogy ha a folyamatfeldolgozás elején valami különleges dologra lenne szükség, az alkalmazásnak helyes futási környezetet biztosíthasson. A parancsértelmezőt a következő GCC paraméterrel tudjuk megváltoztatni:

```
-Wl,--dynamic-linker=/lib/ld-lsb.so.1
```

Az itt bemutatandó eszközök mindezt megteszik helyettünk.

Az LSB fordítási környezet

Az emberek rég felfedezték, hogy a kódváltoztatások sokkal egyszerűbbek és olcsóbbak ha a fejlesztési folyamat korábbi szakaszában kerülnek elő. Ezt szem előtt tartva az LSB Pro-

jekt létrehozott egy fordítási környezetet, amely segít az LSB szabványú alkalmazások készítésében. A fordítási környezetben találunk néhány tiszta fejlécállományt, ***stub*** könyvtárakat és fordító csomagolásokat (***wrappers***).
Az LSB e definíciók nagy részét adatbázisban tárolja. Az egyébként kézzel csak nagyon nehezen szerkeszthető specifikációs részleteken felül képesek vagyunk olyan tiszta fejléc fájlokat és ***stub*** könyvtárakat készíteni, amelyek kizárólag az LSB által meghatározott elemeket tartalmazzák. Az adatbázis segítségével biztosíthatjuk, hogy a változtatások és bővítések során az eszközök és a specifikációk folyamatosan szinkronban maradjanak. A telepítendő csomagok listáját a „Linux Szabványos Csomagok" széljegyzetben találjuk.
Az LSB szabványnak megfelelő alkalmazás készítésekor az első lépés, hogy a kódot az LSB fejlécekkel fordítsuk le. Ha a fordítás sikertelen, akkor valószínűleg valamilyen LSB-n kívüli elemet használunk. Ez persze nem feltétlenül végzetes, de mindenképpen érdemes rá különös figyelmet fordítani. Az LSB fejlécek az */opt/lsbdev-base/include* könyvtár alá települnek. Egy gyors teszt kedvéért adjuk meg a GCC-nek a `-I/opt/lsbdev-base/include` kapcsolót és figyeljük meg mi történik. Ezt és néhány további lépést a később ismertetésre kerülő fordító-csomagolás elvégzi helyettünk.
A kód lefordítása után a következő lépés és teszt a kód végleges alkalmazássá szerkesztése (***link***-elése). Általában ez a lépés valahogy így néz ki:

```
gcc -o app1 obj1.o obj2.o -lfoo
```

Az LSB stub könyvtárak a */opt/lsbdev-base/lib* könyvtárban találhatók, melyet a fordítónak a -L kapcsolóval tudunk megadni. Ezeket a ***stub*** könyvtárakat csak fordításhoz használjuk. Futásidőben általában a rendszer saját könyvtárait fogjuk felhasználni. Akárcsak az előbb, a később bemutatásra kerülő fordító csomagolás mindezt elvégzi helyettünk.
Az alkalmazás összeszerkesztése után az `ldd` parancs segítségével meg tudjuk nézni milyen megosztott könyvtárakat használtunk fel. Ezen a ponton az LSB-ben megadott (és a „Linux Szabványos Alap könyvtárak" széljegyzetben felsorolt) könyvtárakon kívül semmilyen más könyvtárnak nem szabad megjelennie. Ha mégis ilyesmi történik, további lépéseket kell tennünk és statikusan kell beszerkesszük őket. Általában a `-Wl,-Bstatic` és `-Wl,-Bdynamic` kapcsolókkal tudjuk megadni, ha az adott könyvtárat statikusan szeretnénk beszerkeszteni. Mostanra már biztosan ráérezttünk: a fordító csomagolás ezt is elvégzi helyettünk.
Példaképpen, nézzük meg hogy néz ki egy tipikus xpdf alkalmazás:

```
# ldd /usr/bin/xpdf
  libXpm.so.4 => /usr/X11R6/lib/libXpm.so.4
  libt1.so.1 => /usr/lib/libt1.so.1
  libfreetype.so.6 => /usr/lib/libfreetype.so.6
  libSM.so.6 => /usr/X11R6/lib/libSM.so.6
  libICE.so.6 => /usr/X11R6/lib/libICE.so.6
  libX11.so.6 => /usr/X11R6/lib/libX11.so.6
  libpaper.so.1 => /usr/lib/libpaper.so.1
  libstdc++-libc6.2-2.so.3 =>
          /usr/lib/libstdc++-libc6.2-2.so.3
  libm.so.6 => /lib/libm.so.6
  libc.so.6 => /lib/libc.so.6
  /lib/ld-linux.so.2 => /lib/ld-linux.so.2
```

És ilyen egy LSB-nek megfelelő xpdf:

```
# ldd /opt/lsb-xpdf/bin/xpdf
    libSM.so.6 => /usr/X11R6/lib/libSM.so.6
    libICE.so.6 => /usr/X11R6/lib/libICE.so.6
    libX11.so.6 => /usr/X11R6/lib/libX11.so.6
    libm.so.6 => /lib/libm.so.6
    libgcc_s.so.1 => /lib/libgcc_s.so.1
    libc.so.6 => /lib/libc.so.6
    /lib/ld-lsb.so.1 => /lib/ld-lsb.so.1
```

A nem-LSB könyvtárak meg sem jelennek az alkalmazás futásához szükséges könyvtárak közt, hiszen statikusan fordítottuk őket az alkalmazásba. Ez persze hátrányokkal is jár: az alkalmazás végrehajtható állománya nagyobb lesz, ugyanakkor kevésbé függ az operációs rendszertől.

Egyszerűsítsünk

Végül elérkeztünk az `lsbcc` és `lsbc++` fordító csomagolóhoz. A kettő valójában ugyanaz a program; egyszerűen csak más néven hívjuk meg őket, jelezve, hogy C vagy C++ módot szeretnénk. Az elképzelés szerint az `lsbcc`-t használjuk ha GCC-vel fordítunk és az `lsbc++` változatot, ha a *g++* fordítót használjuk.

Linux Szabványos Alapkönyvtárak

Az LSB 1.3 szerint a következő megosztott könyvtárak adottak az LSB-ben. Minden más könyvtárat statikusan kell az alkalmazásba szerkeszteni.

Alapkönyvtárak: ***libc***, ***libm***, ***libpthread***, ***libpam***, ***libutil***, ***libdl***, ***libcrypt***, ***libncurses*** és ***libz***.
Grafikus könyvtárak: ***libX11***, ***libXt***, ***libXext***, ***libSM***, ***libICE*** és ***libGL***.
Ahogy az LSB verziók fejlődnek, a könyvtárak listája is egyre bővül majd.

Linux Szabványos Alapcsomagok

Az LSB fejlesztői környezetet letölthetjük a Linux Standard Base oldaláról (lásd az on-line Források szakaszt); letöltéshez egyszerűen kövessük a hivatkozásokat. A következő csomagokat kell telepítenünk:

- ***lsbdev-base:*** ez tartalmazza a fejléceket és könyvtárakat.
- ***lsbdev-cc:*** tartalmazza a fordító-csomagoló eszközt.
- ***lsbdev-chroot:*** tartalmazz az alternatívaként használható, chroot alapú környezetet.
- ***lsbdev-c++:*** tartalmazza a statikus libstdc++ könyvtárat, melynek segítségével bizonyos C++ alkalmazásokat átvihetünk LSB 1.3 alá.

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

A csomagoló eszköz az összes megkapott kapcsolót értelmezi, majd kissé átrendezi őket. Ez után néhány további kapcsolót is beiktat, hogy a normál rendszerkönyvtárak előtt az LSB szerinti fejléceket és könyvtárakat érjük el. Az eszköz felismeri a nem-LSB könyvtárakat is, és kikényszeríti statikus fordításukat.
Minthogy az LSB-által nyújtott fejlécek és könyvtárak a keresési útvonal elejére kerültek, általában biztonságosan használhatunk nem LSB alapú könyvtárakat is. Azért persze nem árt, ha megnézzük, nem függenek-e valamitől, amit szándékosan hagytak ki a LSB fejlécek és könyvtárak közül, illetve vigyázni kell arra is, hogy statikusan szerkesszük be őket. Az `lsbcc` ilyen módon többnyire teljesen átlátszóan tud működni.

Az LSB fejlesztési környezet használata

Az LSB fejlesztési környezett csomagjainak telepítése után a példaalkalmazás átültetése (***porting***) a szokásos három lépéses műveletre egyszerűsödik, egy apró különbséggel:

```
CC=lsbcc ./configure
make
make install
```

(Mivel nem minden .configure figyeli a CC környezeti változót, érdemes elolvasni a help-jét, miképpen adhatjuk meg neki az lsbcc-t (pl: `./configure –cc=lsbcc`) – a ford.)

A ***configure*** parancsfájl az `lsbcc`-t használja a GCC helyett, és ezzel le is vezényli a az LSB környezet különféle tesztjeit, ráadásul beállítja a programhoz szükséges módosításokat és korlátozásokat. Általánosságban a teljes funkcionalitás közel azonos lesz azzal mintha GCC-t használtunk volna. Gyakorlatképpen próbáljuk meg lefuttatni a `configure` parancsfájlt mindkét módon és hasonlítsuk össze a kapott eredményt. Azzal, hogy a `configure` parancsot az `lsbcc` használatára utasítottuk, egy másik előnyhöz is jutunk: a létrehozott `make` fájlokban a CC változó automatikusan az `lsbcc` lesz, nem kell tehát figyelnünk, hogy minden egyes `make` futásakor megadtuk-e (`make CC=lsbcc`).
Az `lsbcc` parancs alapértelmezés szerint a GCC-t hívja meg a módosított paraméterekkel, de környezeti változóban megadhatunk neki más használandó fordítóprogramot is. Bármilyen fordítóval tud dolgozni, a feltétel, hogy parancssori kapcsolói sszeférjenek a GCC kapcsolóival.

Teszteszköz

Az alkalmazás elkészítése után az `lsbappchk` programmal próbálhatjuk ki, hogy valóban megfelel-e az LSB elvárásainak. A program ellenőrzi az alkalmazásban felhasznált megosztott könyvtárak listáját, továbbá megvizsgálja, valóban csak az LSB által jóváhagyott csatolófelületeket használjuk-e. Nézzünk egy példát a használatára:

```
# /opt/lsbappchk/bin/lsbappchk /bin/ls

/opt/lsbappchk/bin/lsbappchk for LSB Specification
➥1.3.3
Checking binary /bin/ls
Incorrect program interpreter: /lib/ld-linux.so.2
Header[ 1] PT_INTERP    Failed
Found wrong interpreter in .interp section:
➥/lib/ld-linux.so.2 instead of: /lib/ld-lsb.so.1
DT_NEEDED: librt.so.1 is used, but not part
➥of the LSB
Symbol clock_gettime used, but not part
➥of LSB
```

Az LSB nem követeli meg, hogy az operációs rendszer által nyújtott eszközök maguk is megfeleljenek az LSB követelményeinek. Ezért nem is igazán várjuk el, hogy a terjesztés saját `/bin/ls` programja átmenjen ezen a teszten. Egyszerűen csak kézenfekvő példa volt.
Az `lsbappchk` kimenetéből megtudhattuk, hogy az `/bin/ls` nem felel meg az LSB elvárásoknak. Az első probléma, hogy nem az LSB által megadott parancsértelmezővel, azaz a `/lib/ld-lsb.so.1`-el fordították le.
A következő gond, hogy az alkalmazás a ***librt.so.1*** könyvtárat igényli, amely nem része LSB által meghatározott könyvtárkészletnek. Végül pedig, hogy felhasználja a `clock_gettime()` függvényt, de nem statikusan szerkesztettük az alkalmazásba (a ***librt.so.1*** könyvtárban találnánk meg egyébként).
Az ilyen alkalmazások javítása általában úgy történik, hogy újrafordítjuk az `lsbcc` segítségével, amely helyesen állítaná be a parancsértelmezőt és a ***librt.a*** könytárat használná a ***librt.so*** helyett. Néha, a statikusan beszerkesztett könyvtár miatt újabb nem-LSB szimbólumok kerülnek az alkalmazásunkba, így aztán az egész folyamatot többször meg kell ismételnünk.
Néhány nagyobb alkalmazás vagy egymáshoz szorosan kapcsolódó alkalmazáscsoport esetében kívánatos lehet olyan megosztott könyvtárak létrehozása, amelyet csak ezek az alkalmazások használnak. Ez LSB alatt is engedélyezett feltéve, hogy a megosztott könyvtár az alkalmazás részeként települ valamint az alkalmazás saját adatterületén és nem a rendszer-programkönyvtár mappák valamelyikében található. Az `lsbappchk -L` kapcsolója segítségével a teszteszköznek megadhatjuk azon megosztott könyvtárak teljes elérési útját, melyek az LSB- helyesség szempontjából az alkalmazás részének tekintünk. Nézzünk meg például az LSB szerint fordított Apache Web kiszolgálót, amely három saját megosztott könyvtárat használ:

```
# /opt/lsbappchk/bin/lsbappchk
  ➥-L /opt/lsb-apache/lib/libaprutil.so.0
  ➥-L /opt/lsb-apache/lib/libexpat.so.0
  ➥-L /opt/lsb-apache/lib/libapr.so.0
  ➥/opt/lsb-apache/sbin/httpd
  /opt/lsbappchk/bin/lsbappchk for LSB
  ➥Specification 1.3.3
  Adding symbols for library /opt/
  ➥lsb-apache/lib/libaprutil.so.0
  Adding symbols for library /opt/
  ➥lsb-apache/lib/libexpat.so.0
  Adding symbols for library /opt/
  ➥lsb-apache/lib/libapr.so.0
  Checking binary /opt/lsb-apache/sbin/httpd
```

© Kiskapu Kft. Minden jog fenntartva

Csomagolás

Amint azt korábban említettem, az LSB kimondja, hogy a csomagnak az RPM fájlformátumot kell használnia. Ez nem jelenti azt, hogy az alkalmazásunk csomagját az RPM programmal kell elkészítenünk, bár általában valóban ez a legpraktikusabb megoldás, főleg akkor, ha már egyébként is használjuk. Másik lehetőség lehet például, hogy a csomagot Debian formátumban készítjük el, majd az `alien` segítségével alakítjuk RPM csomaggá. Akár más eszközöket is használhatunk az RPM formátumú csomag létrehozásához. Van ugyan egy kezdetleges `mkpkg` nevű programunk az RPM fájlformátum készítéséhez, de minden valószínűség szerint a legelszántabb hackereken kívül más számára csak valamilyen segédprogramon keresztül használható. Alkalmazáskészletünkben most elkészítjük az alkalmazást, majd egy ideiglenes ***root*** könyvtárba telepítve meghívjuk az RPM-t a csomag összecsomagolásához és telepítéséhez. Kicsit nehézkesnek tűnhet ugyan, de viszonylag probléma mentesen működik és valamivel egyöntetűbb eredményt biztosít a vadonban fellelhető különféle RPM változatoknál. Nézzük meg, hogy néz ki az `xpaint` alkalmazás példa definíciós állománya:

```
Summary: An X Window System paint program
Summary: XPaint
Name: lsb-xpaint
Version: 2.6.2
Release: 3
Vendor: Free Standards Group
License: MIT
Group: Appbat/graphics
Buildroot: /usr/src/appbat/pkgroot/lsb-xpaint
AutoReqProv: no
PreReq: lsb >= 1.3

%description
LSB szabványnak megfelelő xpaint verzió.
Az XPaint X Window System alapú, színes
képszerkesztő program.
Az Xpaint elsősorban a célból került az LSB
Alkalmazás készletbe, hogy rajta keresztül
bemutathassuk az X11 könyvtárak
használatát.


%pre

%install

%post

%preun

%postun

%clean

%files

%attr ( - bin bin ) /opt/lsb-xpaint
```

Ezen és az alkalmazáskészletben megtalálható alkalmazások fordításához és csomagolásához használt forráskódot az LSB projekt CVS fájában találjuk meg.

És ez tényleg működik?

Igen, tényleg működik, bár az igazat megvallva, időnként még mindig beleszaladunk néhány alkalmazásba amelyek nem mindig követik a tiszta, hordozható kód szabályait. A LSB ellenőrző készlet részeként létrehoztuk egy, az itt leírt eszközökkel lefordított alkalmazás- sorozatot. Eme alkalmazások közt megtaláljuk az Apache, Samba, Lynx, Python, xpdf és groff programokat. Megpróbáltunk olyan valódi alkalmazásokat összeválogatni amelyek a lehető legnagyobb területet lefedik az LSB csatolófelület készletéből.

Mi a helyzet a C++ alkalmazásokkal?

Az LSB 1.3 nem támogatja a C++ nyelvet, így a szükséges könyvtár statikus beszerkesztésének szabálya lép érvénybe. Ennek elkerülésére az LSB 2.0 változatába már beépítettük a C++ támogatást. Közreadjuk az ***lsbdev-c++*** csomagot, amely tartalmazza a `lsbcc` segítségével beállított és lefordított ***libstdc++*** könyvtárat. Ezzel és a GCC 3.2 változatával úgy tűnik jó eredményeket lehet elérni. Más fordítókat és különféle C és C++ könyvtárakat is kipróbáltunk, de az alkalmazás jellegétől függően különféle problémákba ütköztünk.

Célok és elképzelések

Az LSB fejlesztésénél általános szinten további könyvtárakat fogunk felvenni a specifikációba, feltéve, hogy közmegegyezés alakul ki szükségességük tekintetében és elértek egy adott stabilitási szintet. Ezáltal eltüntethetjük a terjesztés által kiadott és az LSB-nek megfelelő alkalmazások készítése közötti rést.

Az LSB fejlesztői környezet esetében tovább javítjuk majd az eszközök használhatóságát és átlátszóságát. A fejlesztői környezet karbantartása igen aktív, és az eszközöket használók visszajelzéseit nagyra értékeljük. Az LSB 2.0 változatban megjelenő C++ segítségével, a fejlesztési környezet lecserélheti a manapság használt ***lsbdev-c++*** csomagot a ***C++ stub*** könyvtárra, amely így az alap LSB fejlesztői csomagba vándorol.

Jelenleg jó néhány kapcsolót be kell állítanunk az ***rpmrc*** vagy az ***rpmmacros*** állományban ha az RPM-et LSB szabványú csomagok előállítására szeretnénk rábírni. Reméljük idővel elő tudunk majd állítani egy LSB módot az ***rpmbuild*** programhoz, amely mindezt automatikusan megoldaná.

Linux Journal 2004. május, 121. szám

Stuart R. Anderson elkövette azt a hibát, hogy meghallották amikor kijelentett, hogy „tudom hogyan kellene ezt megoldani", és így azóta az LSB Written Specification vezetője lett. Amikor nem az LSB-n dolgozik, Stuart szorgosan ügyködik Dél Carolina felvilágosításán a nyílt forrású témákban, egyesével térítve meg a cégeket. Az anderson@freestandards.org címen érhető el.