Building LSB Compliant Applications

Christopher Yeoh (cyeoh@au.ibm.com)
IBM OzLabs Linux Technology Centre
http://www.ibm.com/linux

September 6, 2002

Abstract

The Linux Standard Base Specification version 1.2 was released on the 28th of June 2002 and on the 1st of July the associated certification program for LSB compliant runtime environments and applications was launched. This paper covers the methods that can be used to build an LSB compliant application, the testing procedures available and what is required to have it formally certified.

1 Introduction

The Linux Standard Base (LSB) is a Workgroup of the Free Standards Group which is a non-profit organisation dedicated to accelerating the use and acceptance of open source technologies through the development, application and promotion of standards[1]. The goal of the LSB is to develop and promote a set of standards that will increase compatibility among Linux distributions and enable software applications to run on any compliant Linux system[2].

The LSB specification is comprised of a set of specifications for a binary portability environment. There is a single common specification document and an architecture specific document for each supported architecture platform. The 1.2 version of the specification supports the IA32 and PowerPC™ architectures. The current LSB Certification Program covers the IA32 LSB processor architecture and certification for LSB Runtime Environments and LSB Applications.

2 Specification

In order to fully understand the requirements of an LSB compliant application it is necessary to read the LSB specification. This section highlights some of the more interesting and important points to application programmers.

The LSB specification is a binary, not a source based standard. As a binary standard includes details which are different between different hardware architectures it was decided to split the specification into multiple documents. The generic specification document contains information which is common among all of the supported architectures and there is one additional document for each supported architecture. The set of documents which forms the 1.2 version of the LSB Specification is:

- **gLSB** - generic ABI for all platforms
- **archLSB-IA32** - x86 architecture specific ABI specification
- **archLSB-PPC32** - PowerPC 32-bit architecture specific ABI specification

Instead of duplicating the efforts of other standardisation groups it was decided to instead reference already existing work. The LSB specification document refers to many other published specifications. For example the gLSB refers to the *OpenGL®Application Binary Interface for Linux* document for the OpenGL related portions of the ABI included. All of the architecture specific documents reference the appropriate ELF specification.

The specification covers functionality in the following runtime libraries:

- libX11
- libXt
- libGL
- libXext
- libICE
- libSM
- libdl
- libcrypt

[1] Linux Standard Base Mission Statement `http://www.linuxbase.org`
[2] Free Standards Group Website `http://www.freestandards.org`

- libz
- libncurses
- libutil
- libpthread
- libm
- libc

It is important to note that it is not required that the LSB specification be only implementable on Linux based systems. The intent is to allow for other operating systems to be able to present an LSB compliant Runtime Environment to applications.

2.1 Package and Script Naming

To avoid conflicts between native distribution packages and LSB conformant packages a naming scheme was specified. All LSB compliant packages begin with "lsb-". If the name of the package contains only one hyphen then the name must be assigned by the Linux Assigned Names and Numbers Authority[3] (LANANA). If the name of the package contains more than one hyphen then the portion of the package name between first and second hyphens must either be an LSB provider name assigned by the LANANA, or it may be one of the owners' fully-qualified domain name in lower case.

As all init scripts live in the same directory a similar scheme has been specified to avoid clashes between init scripts for packages. The LSB specification states init scripts can be in one of three namespaces:

- Assigned namespace. This namespace consists of names which only use the character set [a-z0-9]. These names must be reserved through LANANA.
- Hierarchical namespace. This namespace consists of scripts names which look like this: [hier1]-[hier2]-...-[name], where name is again taken the character set [a-z0-9], and where there may be one or more [hier-n] components. [hier1] may either be an LSB provider name assigned by the LANANA, or it may be owners' DNS name in lower case, with at least one '.'
- Reserved namespace. This namespace consists of script names which begin with the character '_', and is reserved for distribution use only. This namespace should be used for core packages only, and in general use of this namespace is highly discouraged.

Several commonly used names have already been reserved and specified in the LSB specification.

2.2 Properties of an LSB compliant application

LSB compliant applications generally have the following properties:

- Only relies on LSB specified interfaces to be provided by the Runtime Environment it executes on
- Linked against the LSB runtime linker (`/lib/ld-lsb.so.1`) and LSB stub libraries or versions of libraries that by default provide LSB versions of interfaces for linking against.
- Either statically links non LSB libraries or bundles the shared libraries with the application
- Passes the *lsbappchk* test program

[3] `http://www.lanana.org`

- Passes the FHS Checklist
- Passes the application's own functional verification testing on the LSB sample implementation and two LSB compliant systems
- Packaged in the RPMv3 format (with the restrictions as outlined in the specification)

3 Build Environment

Build environments are currently being developed to make it easier to generate LSB compliant applications.

Some libraries such as libc use symbol versioning to allow for multiple incompatible versions of a function to exist in the same shared library. This allows for older applications to continue to use old versions of functions and at the same time allow for the library to implement newer incompatible functionality which newer applications can use. In cases where LSB libraries support symbol versioning, a version for each supported interface has been specified.

As the development of these libraries continue, although they contain older versions of functions the ones required by the LSB specification may not be the ones linked to by default.

Shared libraries have been generated for all of the libraries in the specification. These shared libraries only contain stubs for functions and data required by the specification but the symbols are versioned correctly. When linking against these libraries instead of the system shared libraries it is ensured that the produced binary at runtime will request the correct version of the functions. Also if an application uses an interface from that library which is not included in the LSB then a link failure will occur.

Header files for libraries often contain definitions which can affect the runtime behavior of a binary. For example, the content and size of structs can change in size as libraries develop. Its important that the correct representation is used for the version of the function called. Again, the version of a header file on a system may be newer or older than the one required by the LSB and so LSB versions of the header files have been published.

There are currently two packages, *lsbdev-chroot* and *lsbdev-cc*, which take different approaches to the problem of compiling and linking against compliant headers and libraries. Currently using these to compile against is one of the easier ways to build LSB compliant programs. The *lsbdev-base* package provides the LSB header files and stub libraries which both packages utilise. It is planned that there will eventually be LSB certified development environments available.

A version of rpm which makes it easier to build LSB compliant packages is being developed and is available from the Free Standards Group ftp site[4].

3.1 lsbdev-chroot

lsbdev-chroot takes the approach of presenting only the minimum amount of libraries and header files in the environment in which the compilation is occurring. Configure scripts are often fairly clever in looking for libraries and header files to use and so on a normal system it is possible to accidentally include header files which are not LSB compliant or link against non compliant versions of libraries. Building inside the chroot environment greatly reduces the risk of this.

The chroot environment is built up primarily by using bind mounts rather than copying files. This means that the extra disk space requirements of the build environment are fairly minimal,

[4] `ftp://ftp.freestandards.orb/pub/lsb/lsbdev`

though a 2.4.x kernel is required. The bind mounts can cause some problems if system packages are updated whilst the build environment is running.

It is not necessary to have root access to use the build environment. An ssh daemon is run on a high port and it is possible to bind mount user's home directories into the chroot environment. This allows people to login to the restricted environment for compilations. A full editing and debugging environment can be run outside of the chroot, operating on the same files which are being compiled within the restricted environment.

In cases where applications want to link against non LSB libraries these can be specifically imported into the chroot environment. Depending on the sophistication of the build process for the application it may also be necessary to import other tools into the environment.

3.2 lsbdev-cc

lsbcc is a wrapper program which is invoked when compiling instead of the normal compiler. It modifies the arguments passed to the real compiler to ensure that the LSB versions of header files and libraries are used in preference to other ones installed on the system. The full functionality of the native system is available to build, develop and debug the application. Although C++ support has not yet been added to the LSB specification, the lsbdev-cc package also includes *lsbc++* which can be used to compile and link C++ code and ensure the use of LSB header files and interfaces.

Using lsbcc can be as easy as:

```
# CC=lsbcc ./configure
```

or

```
# CC=lsbcc make
```

3.3 Comparing lsbdev-cc and lsbdev-chroot

Whether it is easier to use *lsbdev-cc* or *lsbdev-chroot* to build an application depends on several factors. The default configuration of *lsbdev-chroot* results in a build environment where many shared libraries and some tools from the system are not available. Configure-like scripts can fail in unexpected ways if the methods they use to test for functionality do not work in the same way as normal Linux environments. Configuring lsbdev-chroot to adapt for these situations can take a significant time.

Where an application has hard coded paths to include files and shared libraries it is possible for *lsbdev-cc* to unknowingly build non compliant binaries, though most of these cases would be picked up by the testing described in Section 4. Other situations such as hard coded use of specific compilers in Makefiles can also trick the build environment into doing the wrong thing.

Although there is no guarantee that *lsbdev-chroot* will pick up these problems, because of the restricted environment it is more likely to. Where it is necessary to use a compiler other than gcc *lsbdev-chroot* will probably be the easier method as it is likely source code changes would be necessary for *lsbdev-cc*.

4 Testing

There is currently only one test program in use for verifying compliance of a binary. The *lsbappchk* program checks:

- that the binary and any non LSB shared libraries that it requires only use dynamically linked symbols that are:
  - part of the LSB specification **OR**
  - defined in shared libraries supplied with the binary
- that the object format of the binary is as documented in the generic and architecture specific LSB specifications.

In addition to displaying error messages *lsbappchk* also generates a TET journal file. This file is used as part of the formal certification program and is in a format which makes it easy to be checked automatically.

A program which checks the format of the RPM package is currently in development. The *FHS Checklist* which forms part of Conformance Statement (See Section 5.1) helps confirm that the application has followed the requirements of the FHS document.

lsbappchk and the *FHS Checklist* are unable to detect all of the possible compliance problems. A good way of performing further testing is to install and run the application on different LSB certified Runtime Environments. In order to do this consistently it helps to have a written functional verification test (FVT) procedure. Examples of FVTs which are used for testing Runtime Environments by running LSB compliant applications can be found on the LSB website[5].

The LSB Sample Implementation[6] as far as possible only implements what is required by the LSB specification. This restricted environment can be very useful in detecting non compliant dependencies by an application[7].

4.1 Example

In the example below *lsbappchk* has discovered that */usr/bin/test* is not LSB compliant because:

- It uses the incorrect runtime linker (should use `/lib/ld-lsb.so.1`)
- It uses dynamic symbols `__overflow`, `fputs_unlocked` and `group_member` which are not in the LSB specification.

```
cyeoh@rockhopper:~$ /opt/lsbappchk/bin/lsbappchk /usr/bin/test
/opt/lsbappchk/bin/lsbappchk for LSB Specification 1.2
Checking binary /usr/bin/test
Incorrect program interpreter: /lib/ld-linux.so.2
Header[ 1] PT_INTERP    Failed
Symbol __overflow used, but not part of LSB
Symbol fputs_unlocked used, but not part of LSB
Symbol group_member used, but not part of LSB
```

4.2 Certification Problem Reporting and Interpretations System

When running *lsbappchk* on a binary you may find errors which you don't believe are caused by compliance problems of the application. The Problem Reporting and Interpretations System can be used to find and report problems about:

- Errors or ambiguities in the specifications against which conformance is based, specifically, in the LSB specifications, or the underlying standards referenced by the LSB specifications

[5] `http://www.linuxbase.org/appbat/fvt/lsb-apache_fvt.html`
[6] More information is available from `http://www.linuxbase.org/impl`
[7] This type of testing is required during the certification process

- Errors in the test suites used to assess conformance with the specifications, specifically, in the LSB test suites, or other test suites referenced by the LSB program (if any)
- Errors in the certification system, specifically related to the application process, agreements and completion of Conformance Statements

5 Certification

The LSB certification program currently includes the certification of LSB Runtime Environments and LSB Applications. Although it is a voluntary program, in order to use the LSB trademark it is necessary to complete the certification program for the product. This paper only covers the parts of the program related to the certification of applications. A tutorial on how to certify a LSB Runtime Environment is available at `http://www.opengroup.org/lsb/cert/docs/certprimer/`. Currently the only architecture supported for application and runtime certification is IA32.

Day to day operations of the LSB Certification program are managed by a Certification Authority (CA) appointed by the Free Standards Group. The Open Group is the current Certification Authority and hosts the LSB Certification website at http://www.opengroup.org/lsb/cert/ .

5.1 Certification Process

The Guide to the LSB Certification Program describes in detail the process required to certify a product. That document, *The Free Standards Group LSB Certification policy*, the *The LSB Application for IA32 Version 1.2 Product Standard*, *The LSB Application Version 1.2 Conformance Statement* should be read before attempting to certify a product. In order to have a product certified you must warrant and represent that the product meets all of the conformance requirements from the Product Standard and implements the features as stated in the Conformance Statement.

The following steps outline the process for certification of an LSB Application:

1. Informal Testing. The test suites and the criteria used to measure compliance are freely available so it is possible to know before starting the formal process if the application is likely to have any problems being certified.
2. The company or organisation seeking certification for one their products must register themself at the certification website.
3. The *LSB Trademark License Agreement* (TMLA) is a legal agreement between the applicant and the Free Standards Group regarding the use of the LSB trademark. This needs to be completed before an application can complete the certification process. The TMLA needs to be signed once per company. Once it has been signed, multiple products can then be registered under it.
4. Register the application on the certification website. Each product registration requires entering into an LSB Certification Agreement with the Certification Authority for the service of being certified. The applicant is required to choose one LSB certified Runtime Environment[8] that they will warrant that the application works correctly on. The CA will at this stage choose another LSB certified runtime on which the applicant will need to verify the application also works correctly. The choice of which LSB certified runtime is chosen will be based on language, region hardware profile and usage profile preferences supplied.
5. Warrant that the application passes its own functional verification tests on the two LSB Runtime Environments selected in the previous stage and the LSB Sample Implementation.

[8] selected from the LSB Certification Register `http://www.opengroup.org/lsb/cert/register.html`

6. Complete and upload the Conformance Statement Questionnaire for the product.
7. Upload the test journal output from *lsbappchk*. If there are any test failures then these need to be resolved by references to granted problems reports and entered into the web certification system. Granted problem reports which may be references are classified as either specification interpretations, test suite deficiencies or certification system deficiencies in the problem report database.
8. At the final stage of the process all of the previously supplied information must be confirmed, a second affirmation of a commitment to the TMLA and LSB Certification Agreement must be given along with the payment of certification fees.

Applicants are notified within fifteen calendar days of whether or not the certification submission was successful. If so then the details of the product are entered into the Certification Register. If there is a problem then a resubmission can be done within sixty days. There is an appeals process defined in *The Free Standards Group LSB Certification Policy* document which allows for disputes to be considered first by the the Specification Authority and lastly by the Free Standards Board of Directors.

5.2 Updating Certification

When applications are updated it may be necessary to undergo recertification. Where a maintenance release is made that does not adversely impact the application's conformance to the LSB specification then it is not necessary to retest or recertify. However, if it is a major update to the application then retesting and recertification will be required.

Certification of an application is only valid for limited time period (generally two years), after which recertification is required. If there are no known compatibility reports or problems with versions of test suites newer than were originally used then retesting will not have to be done.

6 Further information

Linux Standard Base Website	`http://www.linuxbase.org`
Free Standards Group Website	`http://www.freestandards.org`
LSB Certification Website	`https://www.opengroup.org/lsb/cert`
LSB Certification Problem Reporting and Interpretations System	`https://www.opengroup.org/lsb/cert/PR`

7 Legal Statement