Linux ライブラリ

[Linux]

・以下のサイトから、メモ
  ・http://www.linux.or.jp/JF/JFdocs/Program-Library-HOWTO/index.html

プログラムライブラリの３つのタイプ

静的ライブラリ (static library)

・通常のオブジェクトファイルの単なる集合体
・慣習として、静的ライブラリは「.a」という拡張子
・この集合体は、ar (archiver) プログラムを使用して作成。
・以前ほどには使われない。これは、共有ライブラリのほうが優れているため

共有ライブラリ (shared library)

・プログラム起動時にロードされるライブラリ
・適切にインストールされると、その後に起動される全てのプログラムは、自動的にその新しいライブラリを使うことになる

次のことが可能

・ライブラリを更新しながらも、古くて後方互換性のないバージョンを使いたいというプログラムを、引き続きサポート
・特定のプログラムを実行するとき、特定のライブラリ、もしくはライブラリ内の特定の関数でさえもオーバーライドすることができる
・既存のライブラリを使用してプログラムが動いている間にも、これら全てをおこなうことができる

これらの望ましい特性すべてを共有ライブラリがサポートするためには、多くの慣習と指針に従わなければならない
特に「soname」と「real name」の違いについて また、それらがファイルシステム内のどの場所に置かれるべきであるかについて

共有ライブラリ名

soname

・全ての共有ライブラリは「soname」と呼ばれる特別な名前を持っている
・soname は、「lib」というプレフィックス、ライブラリの名前、「.so」という語句で構成、さらに後ろに、ピリオドと、インターフェース変更時に必ず増加するバージョン番号、が続く

特別な例外として、最下層の C ライブラリは「lib」では始まりません

・完全記述の soname は、そのライブラリ自身が含まれるディレクトリをプレフィックスとして含む
・実際のシステムでは、完全記述の soname は、共有ライブラリの「real name」への単なるシンボリックリンク

real name

- 全ての共有ライブラリには「real name」(実際のライブラリコードを含むファイルの名前) がある
- real name は、soname に、ピリオド、マイナー番号、もう一つのピリオド、リリース番号、を加えたもの
- 最後のピリオドとリリース番号は任意
- マイナー番号とリリース番号は、ライブラリのどのバージョンがインストールされているかを正確に示し、設定管理に役立つ

linker name

- 加えて、ライブラリ要求時にコンパイラが使用する名前もる
- 単純に一切のバージョン番号を取り除いた soname

管理

- 上記の名前の使い分けが管理の鍵
- 必要とする共有ライブラリの一覧表を内部に持つときには、プログラムは、必要とする soname のみをリストアップするようにする
- 逆に、共有ライブラリを作成するときには、(より詳細なバージョン情報を持つ) 特定のファイル名を持つライブラリのみを作成する
- 新しいバージョンのライブラリをインストールするときには、二、三の特別なディレクトリのうちの一つにそれをインストールし、それから ldconfig(8) プログラムを実行
- ldconfig は、既に存在するファイルを調べ、real name へのシンボリックリンクとして soname 群を作成
- 同様にして、キャッシュファイル /etc/ld.so.cache も作成。

ldconfig は linker name を作成しない

- 通常、この作成作業はライブラリインストール時におこなう
- 単純に、「最新の」soname もしくは最新の real name へのシンボリックリンクとして linker name を作成
- ライブラリを更新したら、リンク時にそれを自動的に利用したい場合、soname へのシンボリックリンクとして linker name を作っておく
- なぜ ldconfig が自動的に linker name を作成しないのか
  - 基本的には、「最新バージョンを使ってコードを実行したいと思われるかも知れないが、そうではなく古いライブラリをリンクしながらの開発作業を希望されることもあるだろう」
  - そのため、ldconfig は、プログラムをどのライブラリにリンクさせたいのかということについては、何の仮定もおこなわない
  - ライブラリとしてリンカに使わせるものを実際に更新するためには、インストーラがシンボリックリンクを個別に変更しなければならない

例

- 完全記述の soname
- ldconfig が /usr/lib/libreadline.so.3.0 というような何らかの real name に対するシンボリックリンクとして作成

```
/usr/lib/libreadline.so.3
```

- linker name も作成するべき

・/usr/lib/libreadline.so.3 を参照するシンボリックリンクとなるでしょう

```
/usr/lib/libreadline.so
```

ファイルシステム配置

・ファイルシステム内のどこかに配置する必要がある
・ほとんどのオープンソースソフトウェアは、GNU 規約に従う傾向がある。
・GNU 規約は、ソースコードを配布するとき、デフォルト設定時にはライブラリを全て /usr/local/lib にインストールするよう推奨
・コマンドを全て /usr/local/bin に配置するようにとも推奨

■ファイルシステム階層規約 (Filesystem Hierarchy Standard; FHS)

・以下にインストールするのがよい

ほとんどのライブラリは

```
/usr/lib
```

起動時に必要とされるライブラリは

```
/lib
```

システムの一部ではないライブラリ

```
/usr/local/lib
```

ライブラリはどのように使われるか

■プログラムローダ

・GNU glibc ベースのシステム (全ての Linux) では、ELF バイナリ実行ファイルを起動すると、自動的にプログラムローダがロードされ実行される
・ローダは /lib/ld-linux.so.X (X にはバージョン番号) という名前
・ローダは、プログラムによって使用されるその他の全ての共有ライブラリを順次探し出し、ロード

■検索対象となるディレクトリのリスト

・/etc/ld.so.conf ファイル内に記述

Red Hat から派生しているディストリビューション

・多くは、通常 /etc/ld.so.conf ファイル内に /usr/local/lib を含めていない。

/usr/local/lib を /etc/ld.so.conf に追加することは、システム上で多くのプログラムを走らせるのに必要な、共通の「修正」

■キャッシュ処理

・プログラム起動時にこれら全てのディレクトリを検索するのは、とても非効率的なので、実際にはキャッシュ処理がおこなわれます。
・ldconfig(8) プログラムはデフォルトで /etc/ld.so.conf ファイルを読み込み、適切なシンボリックリンクを動的リンクディレクトリ内に作成
・キャッシュを /etc/ld.so.cache に書き込み

DLL を追加したとき、または、DLL を削除したり、DLL ディレクトリのセットを変更したりしたときには必ず、ldconfig を実行しなければならない

環境変数

■ LD_LIBRARY_PATH

・特定のプログラムを実行する際、一時的に別のライブラリを代替的に使用することができる。
・標準的なディレクトリ群に先立ってライブラリ検索対象とすべきディレクトリ群を、コロンで区切って並べる
・新しいライブラリをデバッグするときや、特殊な目的のために非標準的なライブラリを使用するときなどに便利

・http://www.visi.com/~barr/ldpath.html

開発やテストには便利ですが、一般ユーザに日常的に使用させようとして、インストール処理で変更すべきではない

・他の方法では回避できない問題に対処するためには、やはり便利
・環境変数を設定したくない場合、次のようにすると、指定の実行ファイルが、与えられた PATH を使用して実行される。

```
/lib/ld-linux.so.2 --library-path PATH EXECUTABLE
```

これらの機能は全てデバッグのためにある

■ LD_DEBUG

・dl*() 関数群が、実行中の処理に関する情報を大量に出力するようになる

例

```
export LD_DEBUG=files
command_to_run
```

LD_DEBUG=files

・ライブラリを扱うときに処理するファイル群とライブラリ群を表示
・検出された依存関係、および、どの so ファイルがどの順番でロードされたか、を教えてくれる

LD_DEBUG=bindings

・シンボルのバインディングに関する情報を表示

LD_DEBUG=libs

・ライブラリ検索パスを表示

LD_DEBUG=versions

・バージョン依存関係を表示

LD_DEBUG=help

・、指定可能なオプションがリストアップ

インストールと使用

動的ライブラリ (dynamically loaded library)

・プログラムの起動時以外のときにロードされるライブラリ
・動的ライブラリは、プラグインやモジュールを実装するときに特に便利
・標準的なオブジェクトファイルや共有ライブラリとしてビルドされてい
・主な違いは、動的ライブラリは、プログラムのリンク時や起動時に自動的にはロードされない、という点
・そのかわりに、ライブラリをオープンし、シンボルを検索し、エラーを処理し、ライブラリを閉じる、という API が存在
・この API を使用するためには、C プログラマはヘッダファイル <dlfcn.h> をインクルードする必要がある

インターフェース

・dlopen()
　・ライブラリをオープンし、使用前の準備をおこなう
・dlerror()
　・エラーを報告できます
・dlsym()
　・与えられた (オープン済みの) ライブラリ内にあるシンボルの値を検索
・dlclose()
　・動的ライブラリをクローズ