

Biarch support and Debian



後藤 正徳

Goto, Masanori

gotom@debian.org

Overview

- Introduction
 - Biarch, Multiarchの紹介
 - Biarchの過去・現状
 - 他システムにおけるBiarch
- Debianにおける各種取り組み
 - amd64, ppc64, toolchain, multiarch, dpkg
- Biarch実現に向けての提案
- まとめとポイント

Debianと64ビットアーキテクチャ

- これまでの64ビットマシンはハイエンド向け
 - alpha, hppa64, ia64, mips64, ppc64, s390x, sparc64
- 最近は個人でも64ビットマシンを普通に使えるようになってきた
 - amd64(x86-64): AMD x86-64, intel em64t
 - ppc64: Macintosh G5
- 当然Debianでも64ビット環境を使ってみたい!
 - amd64の非オフィシャルサポート
<http://debian-amd64.alioth.debian.org/>

32/64ビットとBiarch

- Biarchとは:
 - Bi (2つの) - arch (アーキテクチャ)
 - バイ-アーチ/アーキ/アーク
 - 2つのアーキテクチャ両方を同じマシンで動かすこと
 - Debianでは同じ命令セットの32ビット版と64ビット版を同時に動作させることを指す
 - 例: i386/amd64, powerpc/ppc64, sparc/sparc64, s390/s390x, mips/mips64, hppa/hppa64, sh/sh64
 - Debianでサポートしている大半のCPU命令セットは、実は32/64ビット両方で動作可能
 - 本講演では、片方しか動かせないアーキテクチャ (alpha, arm, m68k) は考慮しない

32ビット→64ビットの利点・欠点

- 64ビットの利点

- メモリ空間が広がる

- 32ビットは4GBの制限

- ビデオ編集・HPC・データベースにとって4GBはもはや狭い

- レジスタ数の増加による性能向上（amd64のみ）

- 64ビットの欠点

- ビット幅が広がることで同時処理可能なデータ・命令数が低下し、CPU性能が下がることもある

- 通常のアプリにはあまりご利益がない

- **64ビットバイナリと32ビットライブラリが混在して利用できない(逆の組み合わせも同様)**

バイナリ起動の流れ

- 1) プログラムから新しいプログラム用のプロセスを fork
- 2) プログラムから `execve(ファイル名, ...)` システムコールを発行
- 3) カーネルは現在のプロセスをファイル名の実行バイナリへ入れ替え
- 4) ファイルの先頭ヘッダを見て、タイプ毎に起動するプログラムを変える
 - `#!/bin/sh` の場合: `/bin/sh` を起動して 7 に飛ぶ
 - 静的バイナリの場合: そのまま実行を開始して 7 に飛ぶ
- 5) ELF動的バイナリの場合 INTERP セクションに書いてある動的ローダを起動
 - i386 の場合: `/lib/ld-linux.so.2`
 - amd64の場合: `/lib64/ld-linux-x86-64.so.2`
- 6) 動的ローダは NEEDED セクションに書いてあるライブラリを解決、同じバイナリ内部にマップする(32か64かを検索するのは動的ローダ)
 - 例: `libc.so.6` 等 (`readelf -a | grep NEEDED`)
- 7) プログラムを実行へ移す

- 5,6でビット幅が違うものを混ぜることができない

32ビット vs 64ビット

- 64ビットマシンでは通常32ビット/64ビットのどちらでも動作可能
 - 32ビットバイナリと64ビットバイナリはプロセス空間が別々で動作できる
 - プロセス間共有 (IPC, メモリ) には課題あり
 - 32ビットライブラリと64ビットライブラリは一緒にメモリへマップできない
- 64ビットマシンで32ビットバイナリを動作させたいなら32ビットのライブラリが必要
- 64ビット環境でも32ビットアプリを使いたい!
 - 商用ソフト(plugin, codec), 過去のソフトウェア資産, ...

32/64ビット混在環境の実現方法

- 動的ライブラリのロードを32/64ビットで切り替える方法: libとlib64ディレクトリ
 - /libと/lib64とにライブラリが入るディレクトリを分けて別々に管理
 - /libにはシステムの基本ビット数のライブラリを格納
 - 32ビット: sparc64, amd64, ppc64, s390x, mips64
 - 64ビット: ia64, (alpha)
 - 現在最も広く実践されている方法
- メリット: /libと/lib64に分けることでパス名から何ビットのライブラリか容易に分類できる
 - いくつかのシステムでは、パスを分けることがABI仕様として決まっている

/libと/lib64

- 利点
 - 分かりやすい
 - 管理しやすい
- 問題点
 - /libと/lib64に分けなければならない
 - パッケージシステムは別々のディレクトリであることを正しく知っておかなければならない
 - もっとカッコイイ方法はないのか?
 - 他のシステムではどうか?

他システムで複数のバイナリ混在を どうしてきたか(1)

- 32/64ビット: Solaris
 - バイナリやライブラリは通常 32ビット (/bin, /usr/lib)
 - 必要に応じて64ビットバイナリを特別な場所におく (/bin/sparcv9, /usr/lib/64)
- rpm
 - multilib transition
 - パッケージビルド時に、/libと/lib64を切り替える
 - specs内部でパス名に%_libdirマクロを使用すると、ビルド時の環境に応じて/usr/libまたは/usr/lib64を切り替えてパス名を埋め込む
 - configureに対しても同様(%configure)

他システムで複数のバイナリ混在を どうしてきたか(2)

- 別ディレクトリにライブラリー式を用意 (altroot)
 - 単純だが両方を管理する必要がある
 - IA64: /emul/i386-linux/
 - FreeBSD: Linuxエミュレータ: /compat/linux
 - Solaris on Linux: /usr/gnemul/sunos/
 - chrootを使用する方法とカーネルサポートによる方法
 - Linuxではaltroot(personality)を使って実現
 - パス解決時、上記personalityの__emul_prefixが設定してあるモードだとlookupのときにそのディレクトリ配下もみる
- その他
 - NeXT: fatバイナリ (2つのバイナリを1つのファイルに組み込む)

他システムで複数のバイナリ混在を どうしてきたか(3)

- 大きくわけて4種類
 - インストール先の切り替え (/libと/lib64)
 - altroot: ルートディレクトリを別にする (/emul/ia32-linux/, ...)
 - chroot: ルートディレクトリを別にする
 - その他システム固有の方法
- 複数CPUとOSをサポートするDebianにおける複数アーキテクチャサポートは過去にない新しいチャレンジ
- 複数アーキテクチャをサポートするとはそもそもどういうことか？

複数アーキテクチャサポート: Multiarch

- Multiarch:
 - Multi (複数の) - arch (アーキテクチャ)
- Biarchの上位概念
- Biarchのように32/64ビットだけにこだわらず、
様々なバイナリ・ライブラリを同時にサポートする
- 様々なパターンが考えられて複雑

Multiarchの分類

- Matt TagertによるMultiarch分類:

<http://www.linuxbase.org/futures/ideas/multiarch/>

- | | |
|--|---|
| 1. 32/64アーキテクチャの混在 (/lib vs /lib64) | ○ |
| 2. 命令セットの混在 (ia64 /emul/ia32-linux) | △ |
| 3. OS環境の混在 (Linux on FreeBSD) | × |
| 4. エンディアンの混在 (mips vs mipsel) | × |
| 5-1. 命令セット拡張の混在 (i386/i686/SSE) | ○ |
| 5-2. ABIの混在 (o32/n32/n64) | △ |
| 6. 複数CPU/OS対応アプリ (gcc: /usr/lib/gcc-lib/arch-os) | ○ |
| (emacs: /usr/lib/emacs/./i386-debian-linux-gnu) | |

Debianで考
慮する必要
があるか?



Multiarch分類とDebian

- 現実的に広く使用されるもの:
 - 1. 32/64アーキテクチャの混在
 - 現在の主な手法: /libと/lib64
 - 5-1: 命令セット拡張の混在
 - アプリ毎に切り替えが現在も可能
 - 特定のディレクトリ(hwcap)に関しては動的ローダが自動的にディレクトリを切り替える仕組みもある
 - 6: 複数アーキ/OSをサポートするアプリ
 - バイナリ毎に見るディレクトリを変える
- Debianでは、既に5-1, 6は対応されている
- Multiarchの中で1には対応できていない
- Debianにおける取り組みをみる

Various efforts to realize Biarch and Multiarch on Debian

DebianにおけるBiarch,
Multiarch化への各種取り組み



Debianにおける取り組み

- ここからはDebianで行われてきた様々な取り組みの一部を紹介
- アーキテクチャ毎の取り組み
 - amd64とlib64
 - ppc64
 - その他 (sparc, s390)
- toolchainのBiarch対応
- Multiarch対応とdpkg拡張

amd64(1)

- 最初のamd64ポート (biarch)
 - 基本的にバイナリは32ビットで動作
 - lib64*: 全ライブラリパッケージを32/64ビット両対応したbiarch化(例: libc6とlib64c6)
 - 難点1: 主要なライブラリパッケージは全てbiarchを意識した対応をしなければならない
 - 難点2: 通常のバイナリは32ビットのまま
 - amd64では命令セットが64ビットに拡張されただけでなくレジスタ数が倍の16個になったがこれを活用できない(性能に大きく効いてくる)
 - 難点3: Debian側に様々な改造が必要になる
 - 主流とならなかった

amd64(2)

- amd64 native port (pure64)
 - Andreas Jochensらによるnativeサポート
 - 現在のDebian/amd64サポートはこのpure64
 - 単純に全パッケージをコンパイルし直しただけ
 - nativeポートなので、全バイナリが64ビット+レジスタ数の増加の恩恵に容易にあずかれる
 - Debian側をいじる必要は一切ない

amd64(3)

- pure64版の欠点

- ライブラリ関連は/lib、/usr/libにインストール
- lib64ディレクトリはlibディレクトリへのsymlink
- Biarchではない!
 - /libが64ビット用に使われてしまっている
 - 32ビットライブラリは/lib32へ? → 他ディストリビューションとの不整合
- i386バイナリを動かしたいときはchroot環境が必要
 - これが現在実用的
 - dchroot
 - × linux32 (altrootツール)を使ってもamd64はルートディレクトリの/libをみてしまうため、Debianではあまり意味がない

ppc64

- ppc64 native port
 - Andreas Jochensによるnativeポート
<http://debian-ppc64.alioth.debian.org/>
 - ppc64では、メモリ空間が広くなるものの通常アプリの性能は低下する
- powerpc biarch
 - 現在考えられている方式: 64ビットバイナリはBiarchサポートで実現
 - native portは行わない(from debian-powerpc lists)
 - 現実にはBiarchが実現できていないので、一部のパッケージ対応に留まる

その他のアーキテクチャへの対応

- sparc64
 - 事情はppc64と同様。Solarisも含め、通常は32ビットを使用。
- s390x
 - ユーザがそもそも少ないため、ほとんど議論されていない
- ia64
 - amd64と同様の問題があるもの、元々ia32バイナリ実行が遅いこと、/emul/が普及していることでそれほど問題ない

toolchainのBiarch対応

- toolchain
 - gcc, glibc, binutils, linux-kernel-headersなどバイナリを生成するために必要なツール群
- メンテナによるBiarch化対応に不断の努力が続けられている
 - glibc: sparc64, s390x, ppc64, amd64
 - linux-kernel-headers: 同様
 - gcc: 上記 + hppa64
- 32ビット環境でも64バイナリがコンパイルできるよう(逆も同様)パッケージ化

toolchain Biarch化による弊害

- 32ビット環境で64ビットバイナリを生成できるよう対応したため、32ビットマシンでパッケージが生成できなくなってしまった
 - ppc64 on ppc32, amd64 on i386
 - sparc64/s390xもこれまで問題があったが、builddが64ビット・使用者数が少ないため無視されていた
- 極めて深刻なFTBFS、現時点で対策なし
 - 私がBiarchを進めないといけないことに気付いたのはこの問題のせい

Multiarch提案

- これまで取り上げたBiarchに関する問題を、Multiarchのレベルで解決する方法
- Tollef Fog Heenによる提案・実装
<http://err.no/ntnu/thesis/report-multiarch/doc/>
- Multiarch分類で見ると
 - 4. エンディアンの混在以外すべてに対応

Tollef's Multiarchの特徴(1)

- ライブラリは(/usr)/lib/arch-osという形でインストールする
 - 全パッケージのインストール先を上記に変更
- Multi-Archフィールド
 - パッケージがこのフィールドを持っていると、別アーキテクチャのパッケージを複数回インストール可能
- ファイルのreference count
 - Biarchパッケージ2つ(例:i386とamd64)を同時に入れる場合/usr/shareも複数回入ってしまう
 - 各ファイル毎にreference countを数え、参照しているパッケージが全てremoveされるまでは実際に削除しない

Tollef's Multiarchの特徴(2)

- dpkg
 - Scottによるdpkg変更のいくつか(filtersなど・後述)を利用
- 実動作するMultiarch環境
- 実運用までには数多くの課題が残る
- Debian scripts (postinstなど)や/binに対する考慮がされていない
- /usr/lib/arch-os/というパス名はやりすぎ?
- 作業は止まり気味?

dpkgへの改造プラン(1)

- Scott James Remnantによる将来構想

<http://www.dpkg.org/FeatureDependencies>

- debian/controlにFeaturesを追加

– 例: Package: system
Features: amd64, i386, linux

Package: foo
Depends: system:i386

Package: bar
Depends: system:amd64

– debian/controlには、次のように書く

Package: foo
Depends: libc6, binutils [amd64], gawk [any]

– 実際には以下のようになる

Package: foo
Depends: system:i386, libc6:i386, binutils:amd64, gawk

dpkgへの改造プラン(2)

- FILTERS

- インストール先ディレクトリの変更

- rpm -relocate相当

- altroot環境に対して有効

- 特定のファイルだけインストールする機能

- CLASSES

- classがインストール/削除methodを指示

結局現状はどうなのか？

- これまでの流れ
 - Biarchに対する様々な取り組み
 - toolchainの一部サポート開始
 - chroot/altroot
- 実際のところちっともBiarch環境になっていない
- 現在残された課題
 - /lib64ディレクトリはどう扱うか？
 - 一般パッケージをどうBiarch対応するか
 - toolchain関連のFTBFS
- 以降では、上記課題を解決するような、講演者が考えるBiarchサポートに関する提案を行う

Biarch Proposal

Biarch化提案



設計

- 少なくともライブラリくらいはBiarchに対応し、amd64で/lib(32ビット)と/lib64(64ビット)に対応
- ユーザはほとんど意識しないで利用できること
- 一般の開発者は難しいことを考える必要なくパッケージが作れること(反例: lib64*)
- 将来のMultiarchの礎、複雑なmultiarchはやりすぎ
 - Multiarch賛成派が多い中の反対意見
- バイナリの実行についてはchroot or altrootで対応
 - amd64はaltrootが / なので、/lib64導入は必須
 - × 32ビットと64ビットのアプリを同時に動かしたい
- これまでのDebianのBiarchサポート延長線上で行う

pathname transition(1)

- Biarch実現の鍵: /libと/lib64を同時にインストールできること
- 課題:
 - 通常ライブラリパス名はlibと決めうちでインストール
 - インストール時にインストール先を変更すると、バイナリ内部に組み込まれたパス名が変更できない
- 解決方法: pathname transition
 - rpmのmultilib対応と同じ方法をとる
 - 全ソースパッケージのソースに対し、パス名にマクロを使う

pathname transition(2)

- 改造前のdebian/rules

build:

```
src/configure
```

```
$(MAKE) -DDATAPATH=/usr/lib/sample
```

install:

```
$(MAKE) install -DDESTDIR=$(CURDIR)/debian/sample
```

```
install -m 644 data.ico `pwd`/debian/sample/usr/lib/sample/data
```

- 改造後のdebian/rules

```
include /usr/include/debian/debian-make
```

build:

```
src/$(configure)
```

```
$(MAKE) -DDATAPATH=$(usrlibdir)/sample
```

install:

```
$(MAKE) install -DDESTDIR=$(CURDIR)/debian/sample
```

```
install -m 644 data.ico `pwd`/debian/sample/$(usrlibdir)/sample/data
```

pathname transition(3)

- 特徴

- 最終的にインストール先に使うパス名は全てマクロを使用
- \$(configure)は-libdir=\$(usrlibdir)付きに展開
- マクロ (debiain-make) をinclude

- 利点

- インストール先パスをアーキテクチャ毎に変更可能
- 将来のmultiarch対応では必須

- 欠点

- 最終的に全パッケージ(1万以上)がこのポリシーに従わなければならない

マクロの対応

- debian-makeにマクロの対応表が入っている

RPMmacro	RPMdir	DebianMacroDebianDir	
%_prefix	/usr	\$(_prefix)	usr
%_exec_prefix	%{ _prefix}	\$(_exec_prefix)	\$(_prefix)
%_bindir	%{ _exec_prefix}/bin	\$(_bindir)	\$(_exec_prefix)/bin
%_sbindir	%{ _exec_prefix}/sbin	\$(_sbin)	\$(_exec_prefix)/sbin
%_libexecdir	%{ _exec_prefix}/libexec	\$(_libexecdir)	\$(_exec_prefix)/lib
%_datadir	%{ _prefix}/share	\$(_datadir)	\$(_prefix)/share
%_sysconfdir	%{ _prefix}/etc	\$(_sysconfdir)	etc
%_sharedstatedir	%{ _prefix}/com	-	-
%_localstatedir	%{ _prefix}/var	-	-
%_lib	lib	\$(_lib)	lib
%_libdir	%{ _exec_prefix}/%{ _lib}	\$(_libdir)	\$(_exec_prefix)/\$(_lib)
%_includedir	%{ _prefix}/include	\$(_includedir)	\$(_prefix)/include
%_oldincludedir	/usr/include	-	-
%_infodir	%{ _prefix}/info	\$(_infodir)	\$(_datadir)/info
%_mandir	%{ _prefix}/man	\$(_mandir)	\$(_datadir)/man
		\$(_var)	var
		\$(_opt)	opt
		\$(_sbin)	sbin

- 各言語用にもdebian-makeと同様なものを提供

代表的なビルドツール側のサポート

- debhelperを使ったパッケージの場合
 - debian/dirs, debian/*.installなどディレクトリ名がかかれるファイルにも同様にマクロを使う
- cdbbs
 - cdbbsのルール内部をbiarch対応するだけで良い
 - ただし内部でパス名やconfigure/makeを直接明示的に使用しているときは変更が必要
- アプリ側が対応していない場合
 - アプリが/usr/libを固定で使用している場合、パッケージ側でBiarch対応する必要がある

依存関係の解決

- あるライブラリパッケージの依存関係が以下の場合:
 - libncurses5 → libc6
- アーキテクチャ指定でインストールすると、依存関係にあるアーキテクチャのパッケージもインストール
 - apt-get install libncurses5:i386 (on amd64)すると libncurses5:i386 と libc6:i386 がインストール
- 別アーキテクチャの異なるバージョンパッケージをインストールしようとするとうエラーになる
 - libc6:i386 2.3.5-3 とlibc6:amd64 2.3.5-3がインストールされている場合、libc6:i386 だけ 2.3.5-4 へのアップグレードは不可

dpkg-shlibdeps

- ビルド時に共有ライブラリの依存関係を検索
- 現在のdpkg-shlibdepsはライブラリのパス名を記録してくれない。Biarch対応していない
- dpkg-shlibdepsを改造し、shlibファイルのエントリにアーキテクチャ名を入れるかパス名を入れる

– 改善前

```
ld-linux 2 libc6 (>= 2.3.5-1)
libanl 1 libc6 (>= 2.3.5-1)
```

– 改善後

```
ld-linux 2 libc6 (>= 2.3.5-1) /lib64
libanl 1 libc6 (>= 2.3.5-1) /lib64
```

debian/control

- Features, Multi-Arch, Subarchitectureなどのフィールドを使用
 - 指定なし: パッケージ側は何も対応しない。32/64ビット両方ある場合、重なったパス名とスクリプトの実行はすべてデフォルトアーキを優先
 - biarch: 32/64ビット両方ある場合、重なったパス名はすべてデフォルトアーキを優先。スクリプトの実行はインストール時にどちらでも実行される
 - multiarch: パッケージ側はインストール先も含めて重ならないよう考慮。/binなども/bin/arch-os形式でなければならない

package scripts

- パッケージスクリプト
 - postinst, preinst, postrm, prerm
 - 特に問題なのがライブラリパッケージのpostinst中で実行するldconfig
- 対応案
 - Biarch非対応パッケージはスクリプトを実行しない
 - Biarch対応パッケージは内部でフラグを見て、複数回実行されても良いものと良くないものを切り分け対応する

dpkgの改良

- Tollef + Scott案を採用
 - ファイルリンクカウントの使用 (/usr/share)
 - Featuresフィールド相当 (Features:i386)
 - アーキテクチャ明示的依存 (libfoo:i386)
 - relocateの使用
 - /emul/ia32-linuxなどaltroot内で使用されるパッケージは、relocateしてインストールできる仕組みがあると嬉しそう

重なったパス名の扱い

- /binに関しては、パッケージがMultiarch対応していない限りインストールしない
 - apt-get install file-utils:i386 file-utils:amd64
 - /bin/lsはamd64環境では64ビットバイナリになる
 - dpkg FILTERSの活用

非native portの64ビットBuild

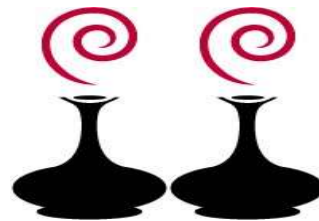
- buildは、biarch 32/64ビットアーキテクチャそれぞれでバラバラに動かす
 - ppc64, sparc64, s390x
 - amd64はnativeなのでこれまで通りbuildを動作させる
- biarch/multiarchという情報が埋め込まれたパッケージのみ、64ビット用にビルド
- 何も対応のないパッケージは、非native portの64ビットbuildからパッケージ生成されない
 - 不要な64ビットバイナリを大量生成することがなくなる
 - toolchainのFTBFS問題を解決できる

その他

- 細かく見ると対応しなければならない周辺ツール・アプリが一杯ある
 - ftp-masters: dak
 - checker: linda, lintian
 - installation tool: apt, aptitude, ...
 - compilation tools: libtool, pkg-config
 - dpkg-provided
- 他の開発者に作業を分担できるように、しっかりとしたポリシーを決めることが重要
- 課題: /etc, /etc/init.d/, /bin, /var や postinst/preinst/postrm/prermの扱い

Summary

まとめとポイント





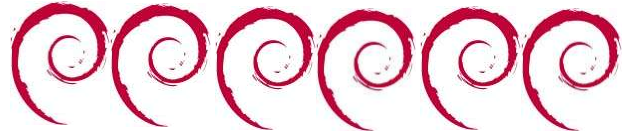
結局Biarch実現で何が嬉しいのか

- ユーザ観点:
 - amd64マシンでi386バイナリを簡単にインストールできる
 - 商業ソフトウェアパッケージ: 大半のユーザにとってはnon-OSSアプリが嬉しい? (codec, flash, ...)
 - 古いバイナリが動かせられる → chrootで十分?
 - amd64をi386マシンとして使っている時に、データベースだけ64ビットで動かす
- 開発者観点:
 - 32/64ビット両方のバイナリコンパイルが簡単にできる
 - lib64*パッケージが必要ない
 - toolchainパッケージがFTBFSにはならない
 - ABI仕様を正しく満せる

何が必要なのか? - Step by step

- Step 1 (32/64ビットライブラリパッケージの強制展開が可能 - /libと/lib64の共存)
 - /lib/, /usr/lib を持つパッケージでマクロを使う
 - debhelper, cdbbsなど最低限のBiarch化
- Step 2 (dpkgで全ての32/64ビット両ライブラリパッケージがインストール可能)
 - dpkg改造
 - 全DebianパッケージのBiarch化
- Step 3 (dpkgで全ての32/64ビットパッケージがインストール可能)
 - ?

誰が何をしなければならないか?

- 一般のユーザ（対応度 )
 - 何も意識しなくてOK (が望ましい)
- 一般の開発者（対応度 )
 - パッケージ内のBiararch対応
 - パス名(debian/dirs等)やconfigureなど
- コアパッケージの開発者（対応度 )
 - ツール対応: debhelper, cdb, dpkg, toolchain, apt, ...
 - debian policyの変更
 - katie (ftp-master)対応

今後

- 完全に動作する実装と提案
- すべてはBiarchに対する需要次第
- 山ほどある課題のうち、どこまで妥協と説得ができるか
- 開発者の協力がどれだけ得られるかが鍵

Question & Discussion

