Multiarch - why it's important

Wookey

The Cross-building victim

4th February 2012
FOSDEM
Brussels, Belgium

Mulitarch

- What is it?
- What does it do?
- How does it work?
- Why does it matter?

Outline

What is Multiarch?

Multiarch is a general solution for installing libraries of more than one architecture on a system

- more general than lib/lib64
- removes all corresponding bodgery (ia32-libs, biarch packages)

Multiarch is very simple

- Put libraries into architecture-specific paths
  - /usr/lib/libfoo (amd64)→/usr/lib/x86_64-linux-gnu/libfoo
  - /usr/lib/libfoo (armel)→/usr/lib/arm-linux-gnueabi/libfoo
  - /usr/lib/libfoo (i386)→/usr/lib/i386-linux-gnu/libfoo

The fundamental thing is that libraries have a canonical path

- Native and non-native locations are the same
- 32/64 special casing goes away (/emul/ia32-linux)
- Cross build and runtime locations are the same
  No more /usr/<triplet>/lib for build-time linking
- Emulated locations are the same (qemu, solaris on linux)

Multiarch Genesis

ia32-libs [is now] the biggest source package in Debian. [. . .]

Tollef Fog Heen

Multiarch Genesis

ia32-libs [is now] the biggest source package in Debian. [...]
Tollef Fog Heen
2005-07-10

ia32-libs was always intended as a temporary solution
Unfortunately the proper replacement took more than 6 years to arrive

Multiarch timeline

- 2004 BOF at Debconf 4
- 2005 Talk at Debconf 5
- 2006 FOSDEM multiarch meeting
- 2008.06 Dpkg multiarch patches uploaded
- 2009.05 apt and dpkg maintainers agree on a package management spec at UDS in Barcelona. Scope restricted to libs.
- 2010.08 Tuple proposal for ABI names drafted
- 2011.02 dpkg multiarch implementation (sponsored by Linaro) lands in Ubuntu. Tuple spec proposed to LSB
- 2011.03 new directory names scrapped, but finalized (and normalized) GNU triplets adopted
- 2011.04: Ubuntu 11.04 released with 83 libraries multiarched, and 14 in a ppa: enough to cross-install flash plugin

Multiarch now

Ubuntu core

110 out of 112 (source) libs in Ubuntu precise main

175 out of 176 (binary) libs in Ubuntu precise main

Ubuntu

425 source out of 2398 source libs, precise (main+universe)

479 source out of 7273 source packages, precise (main+universe)

Debian

360 source out of 2162 source libs, wheezy

401 source out of 7906 source packages, wheezy

Outline

Things Multiarch does

- Cheap emulated environments - emulate only the parts you need to
- Cross-compilation is no longer special - you get it for free!
- Support for cross-grading from one architecture to another
  - (arm→armel, i386→amd64, armel→armhf)
- Better support for binary-only software

Things Multiarch doesn't do

- Install more than one arch of binaries/tools in /bin
- Specify ABI-compatible capabilities (SSE/NEON/ALTIVEC/MMX)
- Different from multilib

Things Multiarch allows

- Partial architectures
- Cross-dependencies (e.g for cross-compilers)
- Cross-built architectures

Outline

How it works - paths

GNU triplets are used for architecture paths, with some adjustment for historical cruft

Debian arch	GNU triplet	Multiarch path
amd64	x86_64-linux-gnu	/usr/lib/x86_64-linux-gnu
i386	i486-linux-gnu	/usr/lib/i386-linux-gnu
i386	i586-linux-gnu	/usr/lib/i386-linux-gnu
armel	arm-linux-gnueabi	/usr/lib/arm-linux-gnueabi
ppc64	powerpc64-linux-gnu	/usr/lib/powerpc64-linux-gnu

These paths matter because the loader path is baked into every binary

- dpkg-architecture -qDEB_HOST_MULTIARCH returns pathname
- an equivalent lsb mechanism is needed for upstream and non-debian distros

How it works - co-installability

Multi-arch-ready packages are given an extra field Multi-Arch

- same: *(libraries)*
  can be co-installed and can only satisfy deps within the arch
- foreign: *(tools)*
  can not be co-installed can satisfy deps for any arch
- allowed: *(both)*
  can be either. Depending packages specify which is wanted

dpkg has support for reference-counting of (doc-)files from co-installable packages that overlap

Multiarch in use

dpkg

```
dpkg --add-architecture i386
```

or

```
echo "foreign-architecture i386" >> /etc/dpkg/dpkg.cfg.d/multiarch
```

apt

apt source entries get an arch field:

```
deb [arch=amd64,i386] http://archive.ubuntu.com/ubuntu precise main
deb [arch=armel] http://ports.ubuntu.com/ precise main
deb-src http://ports.ubuntu.com/ precise main
```

apt has two important config options:

`APT::Architecture` Arch to use when fetching and parsing package lists
`APT::Architectures` All suported arches (native + foreign)

Multiarch in use

```
apt-get update
apt-get install libattr1-dev:armel
```

`dpkg --get-selections` gives:

```
libattr1                    install
libattr1:armel              install
libattr1-dev                install
libattr1-dev:armel          install
```

Crossbuilding

Cross building has 3 major issues

- Installing build dependencies: native tools, cross libs/headers
- Finding/linking libraries
- Running build-time tools

Multiarch helps with all of them.

Crossbuilding - dependencies

1. Generally needed packages are either libs or tools.
2. Some can be both/either.
3. Use Multiarch info instead of annotating each dep.
4. Libs are M-A: same, tools are M-A: foreign
5. The dependency arch is a feature of the *depending* package.
6. The M-A info is a feature of the *depended-upon* package.
7. So we have to annotate the cases that are opposite of what is expected.

`package:native` - when you want a BUILD_ARCH library
`package:same` - when you want a HOST_ARCH tool

Crossbuilding - library paths

Runtime is the same as build-time.

Old system (classic/dpkg-cross)

build-time library path: /usr/arm-linux-gnueabi/lib/libfoo
runtime library path: /usr/lib/libfoo

Multiarch

build-time library path: /usr/lib/arm-linux-gnueabi/libfoo
runtime library path: /usr/lib/arm-linux-gnueabi/libfoo

Much harder for libtool to screw it up

Crossbuilding - build-time tools

Can just run them with qemu
Any dependencies for the tool easily specified

Installing cross build-dependencies

```
# apt-get -aarmel build-dep acl
Reading package lists...
Building dependency tree...
Reading state information...
The following NEW packages will be installed:
  autoconf automake autotools-dev bsdmainutils debhelper
  gcc-4.6-base:armel gettext gettext-base groff-base
  html2text intltool-debian libattr1:armel libattr1-dev:armel
  libc6:armel libc6-dev:armel libcroco3 libgcc1:armel
  libgettextpo0 libpipeline1 libtool libunistring0 libxml2
  linux-libc-dev:armel m4 man-db po-debconf
```

Headers and Dev packages

Dev packages need converting in order to be able to install both native and foreign versions.

- Move include files which differ between arches into /usr/include/<triplet>/
- Moving out all binaries. (usually foo-config - move to pkg-config)
- Get rid of .la files if possible

Changes needed to implement

Obviously many packages are affected (all libs, most -dev, some tools)
But also a number of infrastructure tools (anything that knows about library paths) needed to be fixed before other uploads. Not all of these were expected:

- libc (loader)
- dpkg, apt
- compilers (system library and header paths)
- make (foo: -lbar syntax)
- pkg-config
- pmake
- cmake
- debhelper
- lintian
- libffi
- openjdk (lib-jna)
- dpkg-cross

Multiarch for packagers

- Use dpkg-architecture -qDEB_HOST_MULTIARCH
- Shared libraries must pre-depend on multiarch-support (for M-A libc)
- Move libs into arch-specific paths
- dh_exec exists so you can use $DEB_HOST_MULTIARCH in package install files (/usr/lib/$DEB_HOST_MULTIARCH)
- Fix up pkgconfig files
- Module loaders need attention for the transition
- Get rid of .la files if possible - otherwise empty dependency_libs
- Move /bin and /sbin binaries out into separate packages
- Add Multi-Arch: field
- Move arch-dependent headers into arch-specific paths

Future possibilities

Coinstallable binaries

- Would be useful
- Deliberately left out of initial implementation
- Needs a spec defining

Outline

Things we learned along the way

This is a classic example of a significant distro-wide change. These things are (*very*) hard to get done.

- Use written specs to record shared understanding
- Split your work into bite-sized deliverables
- Make it clear how people can help: http://wiki.debian.org/Multiarch/Implementation
- Design to avoid flag days

Why does Multiarch matter

Significant development of UNIX/FHS/LSB

We've done the hard work and shown that it works
Is this useful beyond Debian and derivatives?

Discuss. . .

Multiarch - Why does it matter

Thanks!

Wookey
wookey@wookware.org
http://wookware.org

about the slides: available at `http://wookware.org/talks/`

SPAM slide #1

Useful URLs:

- http://wiki.debian.org/Multiarch

  Index to specs, instructions for packagers, historical docs

- https://wiki.ubuntu.com/MultiarchSpec

  The main multiarch spec

- ttp://wiki.debian.org/Multiarch/Implementation

  HOWTO for packagers