Multiarch - and Why You Should Care

Wookey

The Cross-building victim

15th February 2012
ELC
Redwood City, California
Multiarch

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

1. Multiarch: What it is
2. Multiarch: What it does
3. Multiarch: How it works
4. Multiarch: Why does it matter?
What is Multiarch?

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

- more general than lib/lib64
- replaces 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

- Change the default linux loader path.

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

Tollef Fog Heen
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. Tuples revised to GNU Triplets.
- 2011.04: Ubuntu 11.04 released with 83 libraries multiarched, and 14 in a ppa: enough to cross-install flash plugin
- 2012.02: Multiarch Dpkg hits Debian Experimental
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

1. Multiarch: What it is
2. Multiarch: What it does
3. Multiarch: How it works
4. Multiarch: Why does it matter?
Things Multiarch does

- Cheap emulated environments - emulate only the parts you need to
- Cross-compilation is no longer special - you get it for free!
- Better support for binary-only software
  - (flash-plugin, windows binaries, ...)

Wookey (Linaro)
Things Multiarch doesn’t do

- Install more than one arch of binaries/tools in /bin
- Specify ABI-compatible capabilities (e.g. NEON/MMX/i386/i586)
  - ABI is calling convention, not instruction set
- Not the same as gcc multilib
Things Multiarch allows

- Support for cross-grading from one architecture to another
  - (arm→armel, i386→amd64, armel→armhf)
- Partial architectures
- Cross-dependencies (e.g. for cross-compilers)
- Cross-built architectures
Outline

1. Multiarch: What it is
2. Multiarch: What it does
3. Multiarch: How it works
4. Multiarch: Why does it matter?
GNU triplets are used for architecture paths, with some adjustment for historical cruft

<table>
<thead>
<tr>
<th>Debian arch</th>
<th>GNU triplet</th>
<th>Multiarch path</th>
</tr>
</thead>
<tbody>
<tr>
<td>amd64</td>
<td>x86_64-linux-gnu</td>
<td>/usr/lib/x86_64-linux-gnu</td>
</tr>
<tr>
<td>i386</td>
<td>i486-linux-gnu</td>
<td>/usr/lib/i386-linux-gnu</td>
</tr>
<tr>
<td>i386</td>
<td>i586-linux-gnu</td>
<td>/usr/lib/i386-linux-gnu</td>
</tr>
<tr>
<td>armeel</td>
<td>arm-linux-gnueabi</td>
<td>/usr/lib/arm-linux-gnueabi</td>
</tr>
<tr>
<td>ppc64</td>
<td>powerpc64-linux-gnu</td>
<td>/usr/lib/powerpc64-linux-gnu</td>
</tr>
</tbody>
</table>

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

- `dpkg-architecture -qDEB_HOST_MULTIARCH` returns pathname
- equivalent `lsb` mechanism (`lsb_architecture`) 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
```

Sets which extra (foreign) architectures are allowed
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 (existing option)
   Arch to use when fetching and parsing package lists
APT::Architectures (new option)
   All supported arches (native + foreign)
Multiarch in use

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

dpkg --get-selections gives:

<table>
<thead>
<tr>
<th>Package</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>libattr1</td>
<td>install</td>
</tr>
<tr>
<td>libattr1:armel</td>
<td>install</td>
</tr>
<tr>
<td>libattr1-dev</td>
<td>install</td>
</tr>
<tr>
<td>libattr1-dev:armel</td>
<td>install</td>
</tr>
<tr>
<td>libc6</td>
<td>install</td>
</tr>
<tr>
<td>libc6:armel</td>
<td>install</td>
</tr>
<tr>
<td>libc6-dev</td>
<td>install</td>
</tr>
<tr>
<td>libc6-dev:armel</td>
<td>install</td>
</tr>
</tbody>
</table>
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
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/lib/libfoo
- runtime library path: /usr/lib/arm-linux-gnueabi/lib/libfoo

Much harder for libtool to screw it up
Can just run them with qemu
Any dependencies for the tool easily specified
All fractions of system-emulation easily accommodated
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

Many tool packages simply need marking Multi-Arch: foreign
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)
- Split out -common packages

Details still subject to debate.
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

Wookey (Linaro)
Multiarch bootstrap ordering - part 1

1. Upload dpkg support for DEB_HOST_MULTIARCH
2. Upload multiarch-aware versions of build tools:
   - pkg-config, make-dfsg, pmake, debhelper, cmake, openjdk-6
3. Upload lintian (preventing ‘unknown control field’ warnings)
4. Upload multiarch aware apt
5. Upload stage1 gcc-4.5 on i386 only
6. Upload libffi (to put i386 headers on correct path)
7. Upload multiarch-enabled eglibc
Multiarch bootstrap ordering - part 2

1. Upload freebsd-buildutils with LIBDIR switched to M-A path
2. Upload libhwloc, liblouis, liblouisxml (libs on new i386 path)
3. Upload gnat-4.4 for full multiarch (before gcc-4.4 due to self build-depends)
4. Upload gcc-4.6 (providing the runtime libraries)
5. Upload ecj for multiarch (before before gcc-4.4/gcj-4.4)
6. Upload gcc-4.5 and gcc-4.4 for full multiarch
7. Rebuild asis with the new gnat-4.4
8. Upload gcj-4.6, gcj-4.4, and any other gcc frontends that don’t self-build-depend, for full multiarch.
9. Upload everything else
Multiarch for packagers - part 1

- Use dpkg-architecture `-qDEB_HOST_MULTIARCH`
- Shared libraries must pre-depend on multiarch-support (for M-A libc)
- Move libs into arch-specific paths
- Add Multi-Arch: field
- `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
Multiarch for packagers - part 2

- Get rid of .la files if possible - otherwise empty dependency.libs
- Deal with any arch-dependent files (e.g. libmagic database)
- Move /bin and /sbin binaries out into separate packages
- 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

1. Multiarch: What it is
2. Multiarch: What it does
3. Multiarch: How it works
4. Multiarch: Why does it matter?
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
Work to include in FHS/LSB is underway
Adoption beyond Debian and derivatives is encouraged

Discuss...
Further info

Useful URLs:

- http://wiki.debian.org/Multiarch
  Index to specs, instructions for packagers, historical docs

- https://wiki.ubuntu.com/MultiarchSpec
  The main multiarch spec

- https://wiki.ubuntu.com/Multiarch/Tuples
  Spec for upstreams

- http://wiki.debian.org/Multiarch/Implementation
  HOWTO for packagers
Multiarch - Why does it matter

Thanks!

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

about the slides:
available at
http://wookware.org/talks/
copyright © 2012 Wookey
license
CC BY-SA 3.0 — Creative Commons Attribution-ShareAlike 3.0