Cross-porting to Windows with Debian

Stephen Kitt <skitt@debian.org>

February 19, 2014

“Kill me now”

— a reaction at DebConf 13

Outline

Using Debian

What?

Using Debian to build software for Windows

What?

Using Debian to build software for Windows involves cross-building

What?

Using Debian to build software for Windows involves cross-building and porting

Outline

Use case 1: Debian

Debian itself includes a number of packages targeting Windows:

- `win32-loader` helps users install Debian from Windows
  - `cpio-win32`
  - `gpgv-win32`
  - `gzip-win32`
- `autorun4linuxcd` starts Debian Live from Windows
- `libreoffice` includes a Windows DLL for its SDK
- `netbeans` includes a Windows platform launcher

There are more candidates (Python for one).

Use case 2: Wine

Wine, the Windows ABI implementation, needs two packages built for Windows:

- `wine-gecko`, xulrunner for Windows used to provide IE functionality
- `wine-mono`, Mono for Windows used to provide .Net

Use case 3: our users

Various projects use (or support using) Debian to build Windows versions:

- Ekiga
- Gpg4win
- VLC
- QEMU
- ...

Outline

Building Windows software

Install `mingw-w64`, and cross-compile.
"Host" is `i686-w64-mingw32` for 32-bit Windows, `x86_64-w64-mingw32` for 64-bit.

- Manually:

```
${HOST}-gcc -g -O2 myprogram.c -o myprogram.exe
${HOST}-strip myprogram.exe
```

- With autoconf:

```
./configure --host=${HOST}
```

- With cmake: use a toolchain file; see `http://www.cmake.org/Wiki/CMake_Cross_Compiling`

Building Windows software for Debian

The usual approach is to build an architecture-independent package containing the generated Windows files.

1. Build-depend (indep) on `mingw-w64` or the specific compiler you need
   - `g++-mingw-w64` will provide 32- and 64-bit C++ compilers
   - `g++-mingw-w64-i686` will provide a 32-bit C++ compiler
   - `g++-mingw-w64-x86-64` will provide a 64-bit C++ compiler

Building Windows software for Debian

The usual approach is to build an architecture-independent package containing the generated Windows files.

1. Build-depend (indep) on `mingw-w64` or the specific compiler you need
   - `g++-mingw-w64` will provide 32- and 64-bit C++ compilers
   - `g++-mingw-w64-i686` will provide a 32-bit C++ compiler
   - `g++-mingw-w64-x86-64` will provide a 64-bit C++ compiler
2. In the `build-indep` target, configure and build with the appropriate options (**not** `-Wl,-z,relro`)

```
dh_auto_configure -Bbuild/windows -- --host=${HOST}
dh_auto_build -Bbuild/windows
```

Building Windows software for Debian

The usual approach is to build an architecture-independent package containing the generated Windows files.

1. Build-depend (indep) on `mingw-w64` or the specific compiler you need
   - `g++-mingw-w64` will provide 32- and 64-bit C++ compilers
   - `g++-mingw-w64-i686` will provide a 32-bit C++ compiler
   - `g++-mingw-w64-x86-64` will provide a 64-bit C++ compiler
2. In the `build-indep` target, configure and build with the appropriate options (**not** `-Wl,-z,relro`)

   ```
   dh_auto_configure -Bbuild/windows -- --host=${HOST}
   dh_auto_build -Bbuild/windows
   ```
3. Install as appropriate; current convention is to ship executables in `/usr/share/win32` or `/usr/share/win64`, DLLs and link libraries in `/usr/${HOST}/lib`.

Outline

Current situation

Compilers

"Traditional" cross-compilers with two toolchains:

- MinGW32, tuple `i586-mingw32msvc`
  - `mingw32` package
  - GCC 4.2.1 (C, C++)
- MinGW-w64, tuples `i686-w64-mingw32` and `x86_64-w64-mingw32`
  - `mingw-w64` package
  - GCC 4.8.2 (C, C++, Fortran, Objective-C, Objective-C++)
  - GNAT 4.8.2
  - GDB 7.6
  - OCaml 4

(mainly using various `...-source` packages)

Current situation

Libraries

- `mingw-w64-dev`, the Windows runtime headers and link libraries
  - includes Pthreads
- Libraries provided by gcc

Current situation

Libraries

- `mingw-w64-dev`, the Windows runtime headers and link libraries
  - includes Pthreads
- Libraries provided by gcc

... and that's it!

There's the rub

Building anything involving libraries is painful:

Building Ekiga

```
--- Getting libregex...
mkdir -p /tmp/user/1000/ekiga/src/regex
cd /tmp/user/1000/ekiga/src/regex; \
for i in regex.c regexec.c regex.h regex_internal.c regex_internal.h regcomp.c alloca_.h alloca.c \
                 localcharset.c localcharset.h; do \
  wget -nv -T 60 -N http://git.savannah.gnu.org/cgit/gnulib.git/plain/lib/$i?id=203e34 -O $i; \
done
[...]
--- Getting libboost ...
wget -nv -T 60 -N -P src http://downloads.sourceforge.net/project/boost/boost/1.52.0/boost_1_52_0.tar.gz
[...]
--- Getting EXPAT ...
wget -nv -T 60 -N -P src https://downloads.sourceforge.net/project/expat/expat/2.1.0/expat-2.1.0.tar.gz
[...]
--- Getting OpenLDAP ...
wget -nv -T 60 -N -P /tmp/user/1000/ekiga/src \
  http://www.openldap.org/software/download/OpenLDAP/openldap-release/openldap-2.4.31.tgz
[...]
```

Outline

Let's fix this!

The Fedora/OpenSUSE/Arch Linux approach

Repackage anything useful in MinGW-w64-specific packages

Let's fix this!

The Fedora/OpenSUSE/Arch Linux approach

Repackage anything useful in MinGW-w64-specific packages

- Fedora has 170 specific packages
- OpenSUSE has 277 specific packages

Let's fix this!

The Fedora/OpenSUSE/Arch Linux approach

Repackage anything useful in MinGW-w64-specific packages

- Fedora has 170 specific packages
- OpenSUSE has 277 specific packages

Specific means all-new source packages, with no direct link to the "main" source package.

Let's fix this!

The Fedora/OpenSUSE/Arch Linux approach

Repackage anything useful in MinGW-w64-specific packages

- Fedora has 170 specific packages
- OpenSUSE has 277 specific packages

Specific means all-new source packages, with no direct link to the "main" source package.
Great for users though!

Outline

Source-package build-dependencies

Let's fix this!

The Debian approach

Source-package build-dependencies

mingw-w64-zlib debian/control

```
Source: mingw-w64-zlib
Section: libs
[...]
Build-Depends: zlib [source]

Package: zlib1g-mingw-w64
Architecture: all
```

No thanks!

No thanks!

(Although source build-dependencies would be nice anyway. . .)

Let's fix this!

The Debian approach

Two new partial architectures

- `mingw64-amd64`
- `mingw64-i386`

(see #606825 for details).

Let's fix this!

The Debian approach

Two new partial architectures

- `mingw64-amd64`
- `mingw64-i386`

(see #606825 for details).
`dpkg-buildpackage` and `sbuild` handle the cross-building, and `dpkg-buildflags` handles the architecture-specific flags.

New architectures

ostable and triplettable

ostable

```
 uclibceabi-uclinux      uclinux-uclibceabi      uclinux[^-]*-uclibceabi
 uclibc-uclinux          uclinux-uclibc          uclinux[^-]*(-uclibc.*)?
 tos-mint                mint                    mint[^-]*
+mingw64-windows         w64-mingw32             w64-mingw32[^-]*
```

triplettable

```
 uclibceabi-uclinux-arm uclinux-armel
 uclibc-uclinux-<cpu>   uclinux-<cpu>
 tos-mint-m68k          mint-m68k
+mingw64-windows-<cpu>  mingw64-<cpu>
```

New architectures

dpkg-architecture

scripts/Dpkg/Vendor/Debian.pm

```
@@ -122,15 +122,17 @@ sub add_hardening_flags {
         #  (#574716).
         $use_feature{pie} = 0;
     }
-    if ($cpu =~ /^(ia64|alpha|mips|mipsel|hppa|arm64)$/ or $arch eq 'arm') {
+    if ($cpu =~ /^(ia64|alpha|mips|mipsel|hppa|arm64)$/ or $arch eq 'arm' or $os eq 'windows') {
         # Stack protector disabled on ia64, alpha, arm64, mips, mipsel, hppa.
         #   "warning: -fstack-protector not supported for this target"
         # Stack protector disabled on arm (ok on armel).
         #   compiler supports it incorrectly (leads to SEGV)
+        # Stack protector disabled on Windows (requires glibc).
         $use_feature{stackprotector} = 0;
     }
-    if ($cpu =~ /^(ia64|hppa|avr32)$/) {
+    if ($cpu =~ /^(ia64|hppa|avr32)$/ or $os eq 'windows') {
         # relro not implemented on ia64, hppa, avr32.
+        # relro not implemented on Windows.
         $use_feature{relro} = 0;
     }
```

New architectures

Advantages

This gives us

- Real binary packages for real binaries

New architectures

Advantages

This gives us

- Real binary packages for real binaries
- Cross-architecture dependencies

New architectures

Advantages

This gives us

- Real binary packages for real binaries
- Cross-architecture dependencies
- No more Windows-specific sub-packages

New architectures

Advantages

This gives us

- Real binary packages for real binaries
- Cross-architecture dependencies
- No more Windows-specific sub-packages
- Cross-building support:
  - `binutils-source` and `gcc-...-source` targets
  - `cross-build-essential`

New architectures

Advantages

This gives us

- Real binary packages for real binaries
- Cross-architecture dependencies
- No more Windows-specific sub-packages
- Cross-building support:
  - `binutils-source` and `gcc-...-source` targets
  - `cross-build-essential`
- and ...

Multi-Arch

Multi-Arch

A blessing and a curse

- A way of installing and running binaries for multiple architectures on a single system

Multi-Arch

A blessing and a curse

- A way of installing and running binaries for multiple architectures on a single system
- The “native” architecture is just a special case

Multi-Arch

A blessing and a curse

- A way of installing and running binaries for multiple architectures on a single system
- The “native” architecture is just a special case
- Many libraries are co-installable

Multi-Arch

A blessing and a curse

- A way of installing and running binaries for multiple architectures on a single system
- The "native" architecture is just a special case
- Many libraries are co-installable
  - Development packages not so much (yet!)

Multi-Arch

A blessing and a curse

- A way of installing and running binaries for multiple architectures on a single system
- The "native" architecture is just a special case
- Many libraries are co-installable
  - Development packages not so much (yet!)
- It's Debian-and-derivative-specific

Multi-Arch

A blessing and a curse

- A way of installing and running binaries for multiple architectures on a single system
- The "native" architecture is just a special case
- Many libraries are co-installable
  - Development packages not so much (yet!)
- It's Debian-and-derivative-specific
  - The toolchains and autoconf support it

Multi-Arch

A blessing and a curse

- A way of installing and running binaries for multiple architectures on a single system
- The "native" architecture is just a special case
- Many libraries are co-installable
  - Development packages not so much (yet!)
- It's Debian-and-derivative-specific
  - The toolchains and autoconf support it
  - cmake and others aren't quite there yet

Multi-Arch

A blessing and a curse

- A way of installing and running binaries for multiple architectures on a single system
- The "native" architecture is just a special case
- Many libraries are co-installable
  - Development packages not so much (yet!)
- It's Debian-and-derivative-specific
  - The toolchains and autoconf support it
  - cmake and others aren't quite there yet
  - Are upstreams interested in patches?

Multi-Arch

A blessing and a curse

- A way of installing and running binaries for multiple architectures on a single system
- The "native" architecture is just a special case
- Many libraries are co-installable
  - Development packages not so much (yet!)
- It's Debian-and-derivative-specific
  - The toolchains and autoconf support it
  - cmake and others aren't quite there yet
  - Are upstreams interested in patches?
  - It involves hacks in various packages

Multi-Arch

Hacking around Python

pyconfig.h

```
#if defined(__linux__)
# if defined(__x86_64__) && defined(__LP64__)
#  include <x86_64-linux-gnu/python2.7/pyconfig.h>
# elif defined(__x86_64__) && defined(__ILP32__)
#  include <x86_64-linux-gnux32/python2.7/pyconfig.h>
# elif defined(__i386__)
#  include <i386-linux-gnu/python2.7/pyconfig.h>
# elif defined(__aarch64__) && defined(__AARCH64EL__)
#  include <aarch64-linux-gnu/python2.7/pyconfig.h>
# elif defined(__alpha__)
#  include <alpha-linux-gnu/python2.7/pyconfig.h>
# elif defined(__ARM_EABI__) && defined(__ARM_PCS_VFP)
#  include <arm-linux-gnueabihf/python2.7/pyconfig.h>
# elif defined(__ARM_EABI__) && !defined(__ARM_PCS_VFP)
#  include <arm-linux-gnueabi/python2.7/pyconfig.h>
[...]
```

Other annoyances

- Usual cross-building issues

Other annoyances

- Usual cross-building issues
- Windows is not POSIX
  - How do we define the contents of `/usr/include`?

Other annoyances

- Usual cross-building issues
- Windows is not POSIX
  - How do we define the contents of `/usr/include`?
- Windows-specific build systems in some packages (*e.g.* `zlib`)

Other annoyances

- Usual cross-building issues
- Windows is not POSIX
  - How do we define the contents of `/usr/include`?
- Windows-specific build systems in some packages (*e.g.* `zlib`)
- Binaries and libraries need special handling:
  - executables end in `.exe`
  - runtime libraries end in `.dll`
  - there are no development libraries for dynamic linking
  - by default some end up in the wrong place

  Where should this be handled? `debhelper`?

Other annoyances

- Usual cross-building issues
- Windows is not POSIX
  - How do we define the contents of `/usr/include`?
- Windows-specific build systems in some packages (*e.g.* `zlib`)
- Binaries and libraries need special handling:
  - executables end in `.exe`
  - runtime libraries end in `.dll`
  - there are no development libraries for dynamic linking
  - by default some end up in the wrong place

  Where should this be handled? `debhelper`?
- Some binary packages don't make sense (`udebs`)

Other annoyances

- Usual cross-building issues
- Windows is not POSIX
  - How do we define the contents of `/usr/include`?
- Windows-specific build systems in some packages (*e.g.* `zlib`)
- Binaries and libraries need special handling:
  - executables end in `.exe`
  - runtime libraries end in `.dll`
  - there are no development libraries for dynamic linking
  - by default some end up in the wrong place

  Where should this be handled? `debhelper`?
- Some binary packages don't make sense (`udebs`)
  - Build profiles to the rescue!

Next steps

1. Get MinGW-w64 in upstream `config.guess` and `config.sub`

Next steps

1. Get MinGW-w64 in upstream `config.guess` and `config.sub`
2. Add the new architectures to `dpkg`

Next steps

1. Get MinGW-w64 in upstream `config.guess` and `config.sub`
2. Add the new architectures to `dpkg`
3. Provide ported packages somewhere
   - (but remember the Windows packages in Debian)

Next steps

1. Get MinGW-w64 in upstream `config.guess` and `config.sub`
2. Add the new architectures to `dpkg`
3. Provide ported packages somewhere
   - (but remember the Windows packages in Debian)
4. Figure out what to do about `/usr/include`

Next steps

1. Get MinGW-w64 in upstream `config.guess` and `config.sub`
2. Add the new architectures to `dpkg`
3. Provide ported packages somewhere
   - (but remember the Windows packages in Debian)
4. Figure out what to do about `/usr/include`
5. Get patches merged

Next steps

1. Get MinGW-w64 in upstream `config.guess` and `config.sub`
2. Add the new architectures to `dpkg`
3. Provide ported packages somewhere
   - (but remember the Windows packages in Debian)
4. Figure out what to do about `/usr/include`
5. Get patches merged
6. Support partial architectures (also useful for `x32` . . .)

Next steps

1. Get MinGW-w64 in upstream `config.guess` and `config.sub`
2. Add the new architectures to `dpkg`
3. Provide ported packages somewhere
   - (but remember the Windows packages in Debian)
4. Figure out what to do about `/usr/include`
5. Get patches merged
6. Support partial architectures (also useful for `x32` ...)
7. Improve cross-architecture binary dependency support

Next steps

1. Get MinGW-w64 in upstream `config.guess` and `config.sub`
2. Add the new architectures to `dpkg`
3. Provide ported packages somewhere
   - (but remember the Windows packages in Debian)
4. Figure out what to do about `/usr/include`
5. Get patches merged
6. Support partial architectures (also useful for `x32` ...)
7. Improve cross-architecture binary dependency support
8. Finish build profile support

Next steps

1. Get MinGW-w64 in upstream `config.guess` and `config.sub`
2. Add the new architectures to `dpkg`
3. Provide ported packages somewhere
   - (but remember the Windows packages in Debian)
4. Figure out what to do about `/usr/include`
5. Get patches merged
6. Support partial architectures (also useful for `x32` ...)
7. Improve cross-architecture binary dependency support
8. Finish build profile support
9. Add support for cross-architecture build-dependencies?

Thanks!

https://wiki.debian.org/Mingw-W64
http://mingw-w64.sf.net

Slides available at http://www.sk2.org/talks/