Cross-compiling Open Source

MOVIAL

Eero Tamminen
<eero.tamminen@movial.fi>

Abstract

What are Make and Autotools? What are the problems with cross-compiling programs that use GNU Autotools to configure themselves? How to solve these problems? What other problems there are?

1 Autotools and Make

GNU Autotools:

- Used to configure software for different compilation environments
- Contains many ready made tests for testing compiler tool-chain, C-library etc. features
- Composed of Autoconf, Automake, Libtool, m4 tools
- Produce '*configure*' script which is shipped with the software

’configure’ scripts:

- Have a standard configuration, build and installation interface
- Produce ’Makefile’ files used by *GNU Make* and ’config.h’ file defining the compilation environment configuration

GNU Make:

- Builds software according to build dependencies and rules specified in Makefiles
- If compilation and target environment are *both* known, Makefiles can be written manually

2 Problems with Autotools

When cross-compiling, ’configure’ scripts either refuse to run, or:

- Compile and run test programs during the program configuration, which fails because cross-compiled programs cannot be run
- Require *manually* exporting environment variables specifying required configuration values

In addition, ’configure’ scripts configure software for the *compilation* environment, **not** to the *target* environment.

2.1 Consequences

Because most of Open Source software uses Autotools, most if it is currently compiled for ARM *natively* e.g. using over-clocked Netwinders and iPAQs...

This is unacceptable if you're going to automate building and testing of a whole embedded distribution.

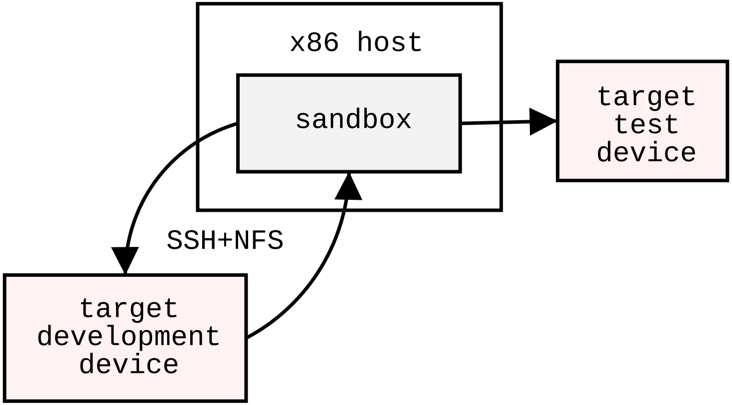
3 Our solution

- Software is cross-compiled on a fast x86 development host
- Building is sandboxed so that ’configure’ scripts find only components which will be available on target device
- *Test programs* created by ’configure’ script are transferred to and run on a networked device that has *the same CPU* as the cross-compilation target and results (output and files) are transfered to the cross-compilation environment. This is automatic and works *transparently* for the ’configure’ scripts

3.1 Requirements

- GNU gcc cross-compilation tool-chains and C-libraries
- A sandbox setup similar to "Linux From Scratch" project
- Networked device with same CPU as the target (for running the configuration tests and install programs) and a target device (to test the resulting software)
- NFS, chroot and Ssh with several auxiliary scripts on both the development host and target device + binfmt_misc trick on the development host for each of the target CPUs

3.2 Build setup



3.3 Advantages

Low level software such as kernel, C-library, Busybox, X11 etc. naturally support cross-compilation "out of the box", but in general configuring software for cross-compilation requires more effort. With our solution (once *that* is configured), it's as easy as on desktop. The target device where the tests are run, can be shared with the whole development team.

If there's enough interest for it, it could be Open Sourced.

3.4 Future

- Multi-user installation
- Replace ssh / sshd with our own remote execution software
- Build system with target software package and root image creation
- Support for distributed cross-compilation
- Test framework
- IDE integration

4 Other cross-compilation problems

While implementing our solution, we encountered some problems:

- Gcc ARM tool-chain is much less stable than x86 one. ARM version of gcc v3.x doesn't produce working binaries for the whole system and vanilla gcc v2.95.x versions need patches[1] to produce good ARM code

[1] Presumably these have been applied to the Skiff tool-chain available from www.handhelds.org

- Gcc 3.x package builds stdlibc++. This means that dynamically linked C++ applications depend from the same C-library as gcc. So you need *separate* tool-chain for each of the kernel major version, C-library and CPU target *combination*

- X11 has it's own Imake based build system and you can't build X11 in separate pieces such as build tools, server & libraries and applications. Instead after a configuration change you use “make World” and rebuild the whole 160MB source tree...

5 Greetings from the developers

“How many billion lines of code you have cross-compiled this week?”

“1GB of RAM makes wonders to your C-compilation times!”

“IDE disks are not designed for this, invest into real hardware and share the machine with others...”

6 References

- Linux From Scratch:
  http://www.linuxfromscratch.org/
- GNU build automation software:
  http://www.gnu.org/directory/devel/build/
- Autotools documentation:
  http://sources.redhat.com/autobook/autobook/autobook_toc.html