MOVIAL

IDEAS IN MOTION

Scratchbox

Cross-compiling a Linux distribution

FOSDEM

27 February 2005

Timo Savola <tsavola@movial.fi>

MOVIAL Corporation

Abstract

Scratchbox is a cross-compilation toolkit that makes the development and building of embedded Linux software faster and easier. It does this by providing a sandbox environment that emulates some characteristics of the target system: dependencies are not mixed with host system's libraries, build scripts that do not support cross-configuration or make it difficult need no tweaking, and compilation is fast on a cheap x86 box.

Scratchbox 1.0 simplifies the especially difficult task of cross-compiling Debian packages. Developers need not be experts of the target systems; installing dependencies, fetching sources, configuring, building, packaging and installing is as easy as doing it natively. You can also run the software inside Scratchbox.

It is not just about Debian and ARM; you can extend Scratchbox to support more architectures, distributions and build tools.

Contents

Project Background

- Research started in 2002
- Eero Tamminen at FOSDEM 2003
  - “Cross-compiling Open Source”
- First public release in June 2003
- Scratchbox 1.0 released in February 2005

- Developed by Movial
- Sponsored by Nokia
- Available under GPL from scratchbox.org

Introduction: Problems

- Building software on embedded devices is slow
- Cross-compiling open source packages is troublesome
- Cross-compiling a whole distribution is even harder

→ Focus of Scratchbox development is on Debian GNU/Linux:
  - Cross-compile it
  - Do it from scratch

Introduction: Scratchbox

- Development is done in a sandbox
  - Software can be built from scratch to an empty filesystem
  - Build scripts find only correct libraries
  - Software can be installed normally to create a root image
- Emulates a non-native system environment
  - “configure” scripts work and produce correct results
  - Tests can be run
  - Built software can be run immediately
- Multiple target configurations
  - Different toolchain (CPU architecture, C-library, ...)
  - Different development tools

Introduction: Results

- Building (configuring, compiling, packaging) is faster
  - Do complete rebuilds frequently
- No need to hack build systems
  - Bring in new upstream packages without delay
  - Development team doesn't need to be full of embedded systems experts

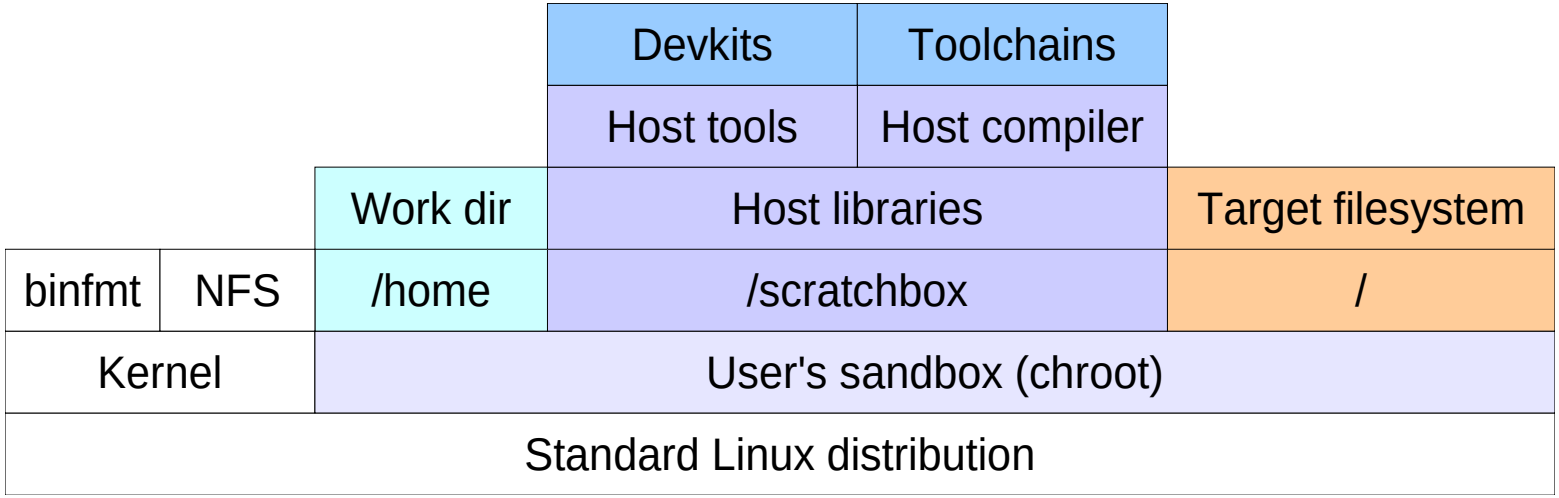
Available Toolchains

- glibc toolchains
  - gcc 3.3.4
  - glibc 2.3.2 + Debian patches
- uClibc toolchains
  - gcc 3.3.2
  - uClibc 20040229

- Prebuilt binaries for x86 and ARM targets
- Custom toolchains can be built from source

- Coming soon
  - gcc 3.4
  - Improved build scripts

How Scratchbox Works



- Build tools are under the /scratchbox directory
  - The sandbox is practically empty
- Host tools take precedence over target binaries
- Host binaries are hardwired to use libraries from /scratchbox
- Kernel executes target binaries using Scratchbox's interpreter

How Scratchbox Works: Targets

- Scratchbox's “targets” consist of:
  - Configuration file (/targets/*name*.config)
  - Filesystem hierarchy (/targets/*name*/ directory)
- Configuration file defines:
  - Toolchain (which effectively defines target architecture)
  - Development kits required for the job
  - Method of handling target binaries
- User's sandbox has one active target at a time
  - /bin is linked to /targets/*name*/bin
  - /usr is linked to /targets/*name*/usr
  - etc.

How Scratchbox Works: Compilers

- Toolchain's binaries are executed via a wrapper
  - “gcc” points to target's cross-compiler
  - “arm-linux-gcc” works if ARM compiler is selected, etc.
- Linker needs additional wrapping to look like a native one

- Build systems think they're compiling natively while they're actually cross-compiling
- ccache and distcc can be used automatically
- Program arguments can be added and removed on the fly

  export SBOX_EXTRA_CC_ARGS=”-pg”

- Toolchains work like normal cross-compilers when used outside Scratchbox

How Scratchbox Works: Binary redirection

Build systems use hardcoded paths...

Scripts have absolute interpreter paths...

Scripts may override PATH...

- “exec” functions are wrapped to redirect target binaries to equivalent host binaries
  - LD_PRELOAD is used to override C-library's functions
- Redirection can be controlled via environment variables
  - Specify source and destination directories
  - Redirect individual binaries explicitly
  - Ignore individual binaries
- Execution can be logged

How Scratchbox Works: CPU-transparency

- Linux kernel detects non-native ELF binaries
  - Standard “binfmt_misc” module
- Handler program runs the binaries using one of Scratchbox's *CPU-transparency methods*

- Scratchbox Remote Shell (sbrsh) executes binaries on real target hardware
  - Typically the device being developed for (or equivalent)
  - Sandbox is recreated at the device using NFS mounts
- QEMU executes binaries by emulating the target CPU
  - General-purpose emulator developed by Fabrice Bellard
  - Supports x86, ARM, SPARC and PowerPC
  - www.qemu.org

How Scratchbox Works: Miscellaneous

- Host C-library reads most config files from /scratchbox/etc
  - Build tools work even when target is broken
- Output of “uname” can be overridden
  export SBOX_UNAME_MACHINE=”arm”
- “automake” and “autoconf” versions are configurable
  export SBOX_DEFAULT_AUTOMAKE=”1.7”
- “autoconf” tries to guess the required version by default

Debian Development Kit

- Provides tools for building, packaging and installing
- Sets up a minimal environment to make the tools work
  - Install base packages as needed

- Packages can be installed with dpkg and apt-get
- Target architecture is used as the installation architecture
  - It can be overridden by hand:

    export SBOX_DPKG_INST_ARCH=”arm”
- Root privileges are not required in the sandbox

- Building works the same way as in normal Debian systems with dpkg-buildpackage, etc.
- Scratchbox's tools are recognised as build-dependencies

Debian Development Kit: Fakeroot

- Debian packages are built using fakeroot
- Fakeroot provides its root-like environment by loading libfakeroot into all executed processes
- CPU-transparency requires libfakeroot compiled for the target
  - Toolchains provide target-specific binaries
  - Installed on the target filesystem
- Sbrsh requires fakeroot with network support
  - TCP-based version was developed for Scratchbox
  - Merged to upstream codebase (“fakeroot-tcp” in Debian)
- Scratchbox's fakeroot uses alternative database format
  - Deleted and renamed files can be detected when loading
  - Does not prevent installing libfakeroot to the target from the Debian package

Demonstration

- Start with a ready Scratchbox 1.0.1 installation
  - User account and NFS exports created
  - sbrshd 6.7 installed on an iPAQ (connected via usbnet)
1. Set up a compilation target
   - Select ARM-glibc toolchain
   - Select Debian development kit
   - Select and configure sbrsh for CPU-transparency
2. Install GTK+ 2 development packages with apt-get
3. Fetch and build the “gqview-1.4.5” source package
4. Install the built ARM binary package
5. Run “gqview” directly from Scratchbox
   - Executed on the iPAQ
   - Displayed in Xnest

Extending Scratchbox

- Additional tools can be built using the HOST target (host-gcc)
  - Install to the /host_usr directory for personal use
  - Install to the /scratchbox/devkits/*name*/ directory to create a development kit
- Create devkits using “sb-devkit-template” source package
  - GAR system manages building of upstream packages

➔ Scratchbox.org can provide web space and version control service for devkit developers

- Stuart Winter has used Scratchbox to port Slackware to ARM
  - www.armedslack.org

Resources

- Downloads, documentation, mailing lists
  - www.scratchbox.org
- Official IRC-channel
  - #scratchbox at irc.freenode.net
- Commercial support available from Movial
  - www.movial.fi