LinuxDevices.com Archive Index (1999-2012) | 2013-current at LinuxGizmos.com | About

- **Search the LinuxDevices.com Archive:**

  Search

- **Explore the LinuxDevices Archive:**
  - February 2012 (11)
  - January 2012 (98)
  - December 2011 (80)
  - November 2011 (99)
  - October 2011 (101)
  - September 2011 (96)
  - August 2011 (115)
  - July 2011 (99)
  - June 2011 (103)
  - May 2011 (107)
  - April 2011 (102)
  - March 2011 (120)
  - February 2011 (98)
  - January 2011 (106)
  - December 2010 (106)
  - November 2010 (106)
  - October 2010 (100)
  - September 2010 (110)
  - August 2010 (105)
  - July 2010 (97)
  - June 2010 (89)
  - May 2010 (80)
  - April 2010 (87)
  - March 2010 (92)
  - February 2010 (83)
  - January 2010 (80)
  - December 2009 (68)
  - November 2009 (76)
  - October 2009 (80)
  - September 2009 (74)
  - August 2009 (69)
  - July 2009 (87)
  - June 2009 (78)
  - May 2009 (89)
  - April 2009 (92)
  - March 2009 (97)
  - February 2009 (105)
  - January 2009 (88)
  - December 2008 (82)
  - November 2008 (75)
  - October 2008 (108)
  - September 2008 (87)
  - August 2008 (96)
  - July 2008 (77)
  - June 2008 (87)
  - May 2008 (97)
  - April 2008 (104)
  - March 2008 (103)
  - February 2008 (90)
  - January 2008 (94)
  - December 2007 (76)
  - November 2007 (97)
  - October 2007 (99)
  - September 2007 (96)
  - August 2007 (109)
  - July 2007 (90)

- June 2007 (92)
- May 2007 (95)
- April 2007 (108)
- March 2007 (121)
- February 2007 (98)
- January 2007 (114)
- December 2006 (80)
- November 2006 (102)
- October 2006 (118)
- September 2006 (104)
- August 2006 (120)
- July 2006 (98)
- June 2006 (108)
- May 2006 (105)
- April 2006 (112)
- March 2006 (112)
- February 2006 (100)
- January 2006 (95)
- December 2005 (74)
- November 2005 (88)
- October 2005 (96)
- September 2005 (101)
- August 2005 (123)
- July 2005 (90)
- June 2005 (102)
- May 2005 (90)
- April 2005 (86)
- March 2005 (103)
- February 2005 (100)
- January 2005 (100)
- December 2004 (79)
- November 2004 (97)
- October 2004 (95)
- September 2004 (102)
- August 2004 (102)
- July 2004 (81)
- June 2004 (89)
- May 2004 (82)
- April 2004 (98)
- March 2004 (114)
- February 2004 (102)
- January 2004 (95)
- December 2003 (83)
- November 2003 (85)
- October 2003 (106)
- September 2003 (116)
- August 2003 (99)
- July 2003 (99)
- June 2003 (93)
- May 2003 (88)
- April 2003 (98)
- March 2003 (109)
- February 2003 (106)
- January 2003 (106)
- December 2002 (79)
- November 2002 (103)
- October 2002 (123)
- September 2002 (111)
- August 2002 (112)
- July 2002 (128)
- June 2002 (121)
- May 2002 (148)
- April 2002 (120)
- March 2002 (185)
- February 2002 (100)
- January 2002 (126)
- December 2001 (76)
- November 2001 (107)
- October 2001 (101)
- September 2001 (104)
- August 2001 (105)
- July 2001 (109)
- June 2001 (104)
- May 2001 (133)
- April 2001 (90)
- March 2001 (145)
- February 2001 (94)
- January 2001 (123)
- December 2000 (82)
- November 2000 (136)

An introduction to the Scratchbox cross-development tool

Apr 1, 2004 — by LinuxDevices Staff — from the LinuxDevices Archive

Foreword: This article by Movial Technical Manager Veli Mankinen introduces a GPL-licensed environment that provides a set of tools to integrate, cross-compile, and test Linux software. Scratchbox provides a sandbox build environment that assures that the intended versions of libraries, headers, and other such files are used during the build.

Mankinen asserts that most of the higher-level software built using GNU Autotools does not cross-compile nicely in their as-is form. Scratchbox solves this problem by allowing the small test programs (used by the configure script to test for availability of features in the environment) to run transparently either using an emulator or through "CPU-Transparency" on an actual target device.

In practice, software configuration and building using Scratchbox is quite identical to how it's done for the desktop, Mankinen says, making it possible to bring regular developers to the embedded project.

Scratchbox – Making Cross-Compilation Faster and Easier

In early years of computing there was no such thing as cross-compiling software. As technology and needs advanced, many different processors and applications were brought to the market. Embedded processors are in many cases powerful enough to run the software. However, development, compiling, and testing of embedded software still requires a lot of effort and time.

As a result, more software has to be run on a variety of hardware platforms and at the same time device manufacturers want to make more use of existing software across platforms to speed up their development. As device manufacturers have identified Linux and Open Source Software as a strategic operating system for embedded devices, there is a need to use the same software across devices. However, when making use of existing software originally developed for desktop or server environments, this becomes very challenging.

One of the reasons for these challenges is because most application development is done on and for desktops and servers. The software compilation takes place natively on those machines. Developers have therefore taken various tools into use that help them compile their software. Unfortunately these tools rarely fully support cross-compiling or have accurate documentation about the scripts and macros used in compiling. Hence, device manufacturers have a need to facilitate and speed up cross-compilation of software for embedded devices.

Device manufacturers' software development and cross-compilation problems

Linux device manufacturers face a number of critical problems when developing and cross-compiling software for devices. The first problem relates to the fact that most Open Source software projects use a 'configure' script to configure their software for compilation. This script is produced by tools called 'autoconf' and 'automake', which processes 'm4' macros written by the application developer. The script will generate the Makefiles that are used for building the software and a 'config.h' header file containing defines for features found on the build system. Configure is meant to ease configuring the software for compilation and its default assumption is that the software will be run in the same environment and processor in which it was compiled in and run from the place where it was installed to.

Autoconf provides application developers certain macros to check out features in the system. The problematic macros for cross-compilation are:

- **AC_TRY_RUN** — Tries to compile, link and run given test code for some feature. Test programs return zero for success.
- **AC_TRY_LINK** — Tries to compile and link test code for library function existence.
- **AC_CHECK_LIB** — Tries to compile and link test code using certain library.
- **AC_SEARCH_LIBS** — This does the same as AC_TRY_LINK, but 'configure' tries to do linking from all of the system library paths, not just from ones given in CPPFLAGS, CFLAGS and CXXFLAGS environment variables.

The second problem device manufacturers face is that most application developers don't take cross-compilation into account when using these macros, so 'configure' ends up cross-compiling test code and trying to run it on the build host, which breaks the configuration. Configure can also find libraries, headers and versions of those that are only present on the build host, not on the target, which will fail either compilation of the program or running it on the target.

A third problem is when the developer does not know whether 'configure' found the correct values or not, unless he manually goes through tens of thousands lines of output it produces (or the binary fails when it's run). This results in a lot of manual work.

For more about the problems of cross-compiling open source programs, see this paper (PDF download) from Movial.

It is of course possible to just take the easy road and use native compiling. However, this is rarely an option as the embedded processors lack the computing power that the desktop computers have, and therefore it takes a lot of time to compile a necessary set of software.

Software development and cross-compilation for Linux device manufacturers can therefore result in a number of problems that slow down the development process significantly, as well as making development and cross-compiling more difficult. This requires more manual work from the development team.

Current cross-compile solutions and limitations

There are commercial embedded solutions available for device manufacturers for solving some of these problems. However, it seems most of those solutions only offer a limited set of programs and libraries that have been made cross-compile aware. This limits the selection of other programs and libraries that the device manufacturer can use in its products. In many cases the libraries and tools offered by the commercial embedded solutions are also quite out-dated.

Device manufacturers therefore face problems when they want to use or experiment with new software libraries and programs that are not supported by any commercial embedded solutions. In this case a device manufacturer then has three options:

1. Decide what software to use and then manually get these to cross-compile, which is often very time consuming and difficult, or . . .
2. Use native compiling which drastically slows down the development process, or . . .
3. Find a whole cross-compile environment with which most of the software and libraries cross-compile straight out of the box.

Scratchbox – speed up development, testing, and time to market

Scratchbox was developed to facilitate making use of any existing open source software in embedded devices. Scratchbox is a configuration and compilation environment for building Linux software and entire Linux distributions. The basic idea in Scratchbox is to offer developers an environment that works and looks like the target environment before the target environment is available. This drastically speeds up development and facilitates cross-compilation, enabling device manufacturers to bring their products to the market faster.

With Scratchbox all the work that needs computing power, like compiling, is done on a desktop machine instead of on the target device. This makes the compilation process much faster, and making small changes and testing cycles a lot shorter. Scratchbox can be run on an x86 machine and supports compiling for x86 and ARM architectures. Adding additional architectures and integration into IDE's can be done rapidly. It is also possible to use other than glibc C-library within Scratchbox.

Scratchbox makes it possible to configure and compile software for other architectures than where Scratchbox itself is running. It is also possible to run the software compiled for other architectures inside Scratchbox using the target device transparently. This means it makes it easier to bring non-embedded developers to the development team since most of the development is similar as development for desktop or servers. Scratchbox makes it possible to share the target device with the entire development team, regardless of their physical location.

When logging into Scratchbox for the first time in ARM-mode it looks like you are logging into a Linux distribution installed on for example an ARM device. However, this distribution seems to be empty (/bin, /usr, /var do not contain anything). This makes it possible to compile and install software so that everything can be installed on the default locations but nothing from the environment will be overwritten. Although the environment looks empty it contains most of the tools needed for normal configuration and compilation process. As it is also possible to run compiled ARM programs in Scratchbox the illusion of being on a real ARM device is almost perfect.

To summarise, Scratchbox offers device manufacturers the following benefits:

- Easy cross-compiling
- Shorter compilation times (Compared to native building)
- Shorter development and test cycles
- Automates and facilitates cross-compilation and development
- Makes it easier to develop your own cross-compiled embedded Linux distribution
- Share target device with the entire development team
- Development does not differ a lot from normal desktop software development
- Bring regular Linux developers to the embedded development team

Conclusion

Embedded software development can result in a number of problems for device manufacturers such as long compile times or many configuration problems when cross-compiling software. Scratchbox is a solution that speeds up and facilitates the development, cross-compile, and test times.

Scratchbox solves most of the traditional problems in cross-configuring and cross-compiling by emulating the target environment. Although you would be using some existing Linux distribution it is in many cases required to be able to compile the whole system for assuring decent testing and quality. In these cases native compiling is rarely an option and Scratchbox can be used to speed up the development.

About the author: Veli Mankinen has been a technical project manager in Movial for over two years, and has been involved in many of Movial's embedded Linux projects. Veli is one of the lead architects and developers behind Scratchbox. The Scratchbox project originated at and is maintained by Movial, a Finnish software design, integration, and development firm specializing embedded Linux and Symbian devices.

This article was originally published on LinuxDevices.com and has been donated to the open source community by QuinStreet Inc. Please visit LinuxToday.com for up-to-date news and articles about Linux and open source.

Related Posts:

- GNU tools gain GPL'd userland cross-build tool
- EclipseCon features two DSDP talks
- Presentation relates Nokia's open source…
- Debian conference videos include embedded
- Embedded Linux app IDE bundles C++ libraries
- Tutorial: Improving an embedded Linux system
- Tiny boards gain Linux cross-tools support
- Using eCos with CMake
- Nokia courts GNOME hackers with tablet discounts
- Getting started with an embedded Linux system emulator
- Targeting virtual hardware
- Linux dev kit targets SODIMM-sized computer module
- Technical webinars address cross-compilation
- Tutorial explains how to build Linux cross compilers
- Linux device software provider supports MIDs/UMPCs

Comments are closed.