PTXdist - A Proposal

Dipl.-Ing. Robert Schwebel

Pengutronix, Braunschweiger Straße 79, 31134 Hildesheim, Germany

r.schwebel@pengutronix.de

Abstract

This paper describes a reliable build system that can be used to construct userland for embedded Linux applications. Building a customized Linux is as easy as configuring a kernel. A single configuration file defines precisely how the system was built and makes it reproducable, even after years.

1. Embedded Linux in Automation

2. Situation

Linux is currently establishing more and more as a reliable operating system for the automation industry. Being a Unix variant Linux has natively most of the features being required in a modern 32 bit operating system: high availablility, good connectivity and a huge bunch of tools for nearly every application in a networked environment are a solid base for productive use. Linux is highly scalable and runs on five dollar processors as well as on multiprocessing systems.

Today all incredients are available to use Linux as an operating system to be used on industrial PCs for the automation industry. There exist several solutions for Hard Realtime extensions, field-bus connectivity and fail safe clustering for FIVE9 reliability. Due to it's portability Linux runs on all modern processor plattforms, such as x86, PowerPC, ARM, MIPS, SuperH and m68k.

3. Development Environments

Contrary to other operating systems there is a standard compiler suite on Linux which can handle all kinds of compilation problems: The GNU Compiler Suite GCC has a very modular frontend/backend concept, so it is possible to use one compiler environment for all kinds of languages (C, C++, Java, Fortran, ADA, ...) and processors. The GCC suite is included in every Linux distribution, so normally a developer has everything he needs to do native and cross development using the same environment. The GCC suite is covered by the GPL License, which ensures that the customer has control over the complete sourcecode. Besides the compiler itself there is a broad range of development tools like editors, debuggers and graphical development environments which cover all ways people are used to program.

4. Building Embedded Systems - Today

Having all the development tools the user needs mostly two components to build up a complete Linux environment for an embedded system:

- The Linux Kernel itself
- A file tree containing the basic libraries and tools (rootfs)

The compilation of a kernel is easy these days: normally one takes just the standard kernel (found on http://www.kernel.org), configures it to the need of the target system (this results in one ASCII configuration file which defines exactly what features the kernel has) and start a kernel build.

Things are more difficult with the root filesystem: there is no standard way to assemble the root tree. Till now people follow three roads when it comes to the root filesystem:

1. Downscale a mainstream distribution (like SuSE, RedHat, Debian)
2. Take a one-disk distribution (AtomicRTAI, MiniRTL, LEM)
3. Compile from scratch

The first two variants have two significant disadvantages when it comes to industrial use: they are not optimized for the special system (which can be a problem on small ressource machines) and – more important – you never know what *exactly* you have on the machine and how it was compiled. This indeed is *very* important for industrial applications: Open Source operating systems like Linux are a rapidly moving target where bugs are fixed on a daily base and versions come out quickly. So if you want to support your customers for 10 to 15 years (this is usual in automation) you definitely need control over the sourcecode.

The third variant is better: if the system is compiled from scratch it is exactly known which versions of the software have been used, which patches have been added and how the packages were configured. The downside is that building a system from scratch can be a lot of work, vulnerable to errors and it is not easy to track which steps have been done in which order.

So what is needed for industrial use of Linux is a combination of a distribution and a compilation from scratch.

5. A Comprehensive Build System for Industrial Linux

There is a better solution to create a root filesystem for industrial embedded systems than using the three methods mentioned above. The idea is to make building a complex root tree as easy as configuring and compiling a Linux kernel.

The system we propose here consists of two parts: a configuration system and a bunch of Makefiles which actually do the work. The configuration system should be well known to anybody who has compiled his own kernel: it is the `make menuconfig` system from the Linux kernel. Like with the kernel the user can easily configure the system by making crosses in a menu structure to enable and disable certain features. The result of the configuration system is also one single ASCII configuration file which describes exactly which components are part of the target system and how they should be configured.

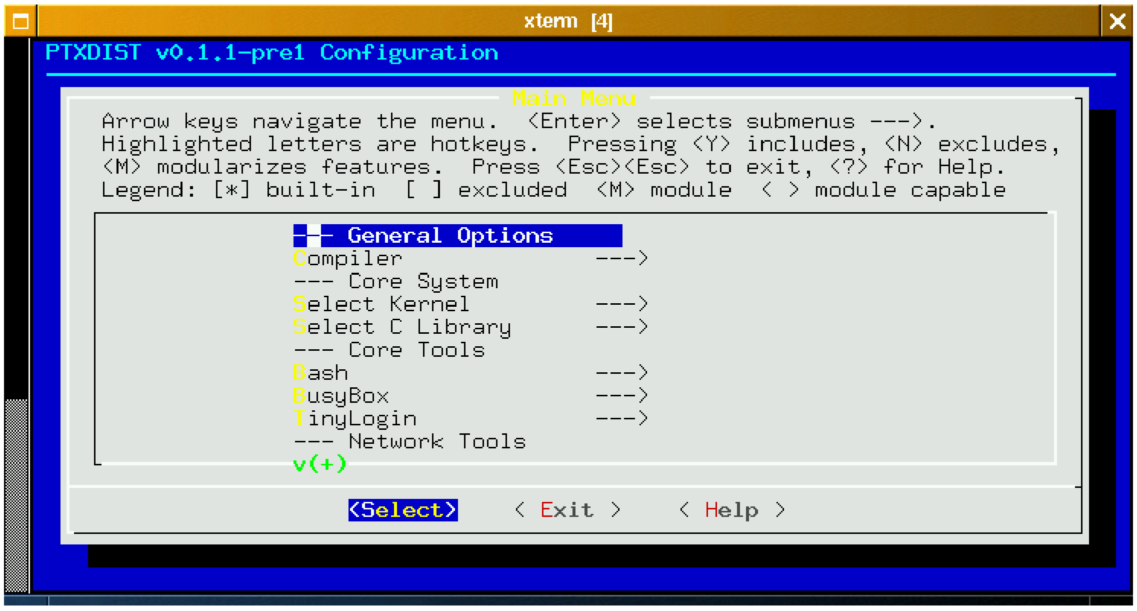
The configuration system itself is pretty small, it fits on one disk. All the software components necessary to construct the final root filesystem are collected from one of two sources: from the

Internet by fetching them from the original websites of the projects or from a local directory. If the configuration was finished and the user does a `make get` it is tested if all sources needed for the build are available, otherwhise they are fetched. After this stage was run it is possible to burn the complete tree to a CD, so the source of the resulting system is completely fixed.

Note that only originally released sources are taken for the target system. This makes it easy to track bugs, because packet maintainers don't like it if you report bugs for versions which have not been released by them. However, in some cases there may be features that are only available with certain patches to the original sources. In this case the patches are put into a separate directory in the installation tree. The patches have well defined version numbers, address only one problem at a time and are tightly integrated into the configuration system. So for example to patch an official Linux kernel 2.4.18 with the `rthal5g` patch for RTAI you simply have to check one cross in the configuration dialog.

The second stage of the Makefiles patches and configures all packages according to what the user has selected in the configuration system. This includes options for packages (such as: should a library be compiled statically or dynamically, which features shall be switched on or off etc).

Finally, `make compile` compiles all packages and `make install` installs the result into a predefined root directory. All stages can be run by entering `make World`.



```
PTXDIST v0.1.1-pre1 Configuration
                              Main Menu
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

              --- General Options
              Compiler              --->
              --- Core System
              Select Kernel         --->
              Select C Library      --->
              --- Core Tools
              Bash                  --->
              BusyBox               --->
              TinyLogin             --->
              --- Network Tools
              v(+)

              <Select>    < Exit >    < Help >
```

6. Document Revision History

2002/08/06, Robert Schwebel: Initial Revision