Embedded Linux Systems

Kitrek Riese
Software Engineering
University Wisconsin Platteville
kitrek@hotmail.com

Abstract

An embedded system is anything with a microprocessor in it that is not considered a general-purpose computer. Often these are things with a microcontroller added to make them 'smarter' such as a coffee maker with a timer in it or a car stereo.

Using Linux in embedded systems is a relatively recent idea. Although now a common practice, the first concepts of embedded Linux arrived on the scene around 1997. Government agencies and the private sector alike are gradually increasing their use of Linux systems and continually looking to Linux to meet their embedded requirements.

This paper discusses all the considerations that must be made in designing an embedded Linux system. It also covers some background information on Linux in general, and some advantages and disadvantages of choosing Linux to meet your embedded needs.

Why Choose Linux

Linux has many properties that give it advantages over traditional embedded operating systems. One of these is the quality and reliability of Linux code. Because separate functionality is found in separate modules and the different modules are split into different files, reflecting their functionality, Linux code has very high cohesion and low coupling. This means the code is easily maintainable. Also because of this design adding functionality to the code simple compared to other designs. Another major advantage of the Linux design is the ability to decide which components to install. This gives Linux the ability to be customized for a particular application.

The main reason for the reliability and ease of use of Linux code is due to the open source development model. This gives many people the ability to address issues in the code, discuss solutions to problems, and implement the best solution. Few other operating systems give you the advantage of such a large community of support and problem solving. The open source development model also gives you the advantage of being able to view and edit the source code. This gives you the opportunity to fix the code without the need to wait for help from outside organizations if you were to find a problem.

Embedded Linux Architecture

An embedded Linux architecture will have a good number of abstraction layers. The The general architecture discussed here contains, from bottom to top, the hardware layer, the Linux kernel (which is broken up into levels of abstraction within itself), libraries that allow applications to communicate with the kernel and the applications themselves.

The hardware element of the general Linux architecture has a number of minimal operating parameters. It must be at least a 32-bit CPU, contain some level of I/O capabilities to aid in debugging, and it must have an appropriate amount of RAM.

The Linux kernel, found above the hardware layer, is the main component of the operating system. Its purpose is to interface the hardware with the applications above using high-level abstractions which are comfortable for the software developers. The idea is that through these abstractions, the applications can be transferred to different hardware with little or no changes. The kernel drives devices, controls I/O devices, manages process scheduling, keeps track of memory sharing, distributes signals, and provides many other administrative tasks.

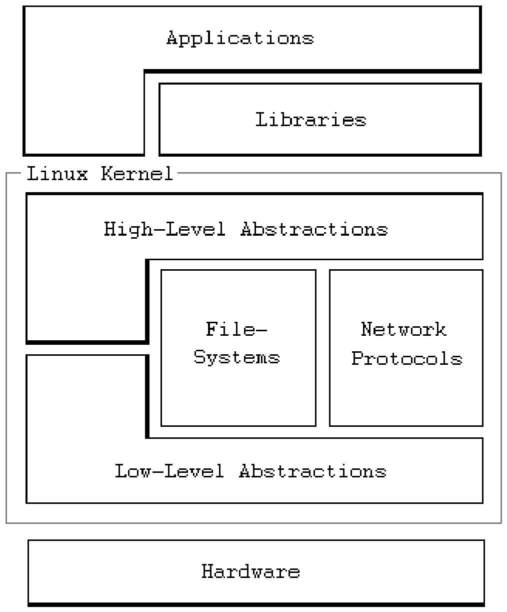


Figure 1: General Linux Architecture

Along with the high level abstractions to the application level, the kernel also contains low level abstractions to the hardware. This provides an interface between the hardware and he rest of the Linux kernel. Configuring the architecture in this way provides the means to change hardware with little changes to the kernel, only the low level abstraction must be reconfigured.

The file system and the network protocols are examples of structured data the Linux kernel must be able to interpret to provide communications between the low and high level abstractions. Because of the large amount of file systems and network protocols which have been developed, in order for the kernel to allow for common high level abstractions, the Kernel must be able to understand a large number of file system formats. This architecture allows the section of the kernel witch deals with the file system to be replaced in the event of a change in file systems. This concept is also used for the network protocol partition.

For the most part, directly above the kernel you will not find a direct interface with user applications. These applications go through different libraries providing yet another layer of abstraction. The most common of these libraries is the GNU C library. Other common libraries include Qt, XML or MD5. Each of these provides a different utility which may be better suited for your particular purpose. These libraries are not part of the application but rather exist separately within the Linux structure. This allows different applications to use the same instance of a library. This saves memory because you need only one copy of the library to be loaded into RAM . (Yaghmour)

Hardware Selection

Linux supports many processors used in embedded systems. These include the x86, ARM, IBM/Motorola PowerPC, MIPS, Hitachi SuperH and the Motorola 68000. They each have their own particular advantages and disadvantages.

The x86 processor family was originally introduced by Intel in 1985, and many other companies including AMD and National Semiconductor now produce compatable processors. This architecture has the great advantage of a large user base and therefore a large amount of documentation. This is mainly because this is the most widely used and tested of all Linux platforms. In fact many applications are developed for the x86 just for this reason, with the plan of porting them over to other platforms for distribution.

The ARM processors have the unique characteristic of not having any particular manufacturer. ARM stands for Advanced RISC Machine and is maintained by ARM Holdings Ltd. They design the CPU cores based on the ARM chip instruction set and allow their customers to manufacture the chip, or find someone to manufacture it for them. This allows the customers various advantages including price competition and greater customizability. But it does provide confusion to new comers to the market. Currently Linux supports ten different ARM CPUs on sixteen different platforms, along with two hundred related boards.

As you can well see, there is no shortage of Linux supported processors. There is more to hardware however than the CPU. Linux also supports a large variety of busses and interfaces. These include popular interfaces like PCI, PCMCIA, Parallel Port, SCSI, USB, FireWire, Serial Port and many others. Along with these, Linux supports a large variety of storage, sound, printer, mouse and display devices.

Embedded Linux systems also have large support for networking devices. Ethernet, 802.11 and Bluetooth are among the most common and well supported.

Chances are whatever hardware you choose you will be able to find a wide variety of Linux support. In addition, if you find yourself breaking new Linux ground you can always fall back on the open source community for help and support.

Development Tools and Environments

The main difference between embedded development tools and the tools of other software developers is that embedded development tools usually run on one platform while their end result is run on another. The software required for this cross platform development includes a bootloader, build-tools, debugging tools, the Linux kernel, and various other software tools.

The GNU cross-platform development tool-chain contains many of this necessary software, and it is available free.   This tool-chain contains software to set up the kernel, a utility to manipulate binary files, the complete C library, and a full compiler. Binutils is the piece of the package that gives utilities to manipulate binary files. This gives you the ability to do different things such as list the symbols in an object file, translate object files, print the strings of printable characters in the object files and much more. The C library can be replaced with alternatives such as the GNU Java compiler, Perl, Microperl, Miniperl, Python, and Ada.

The different parts of this, and other tool-chain packages can be selected and swapped out for your specific application. There are many possibilities for each of these pieces and they must be chosen carefully. Different packages have different quirks in their functionality. For instance, certain packages may build and work fine during simulation but cause problems when installed on the end system.

There are many different integrated development environments (IDE's) available for Linux. Many of these are available as open source. Some popular open source IDE's include Anjuta, Eclipse, Glimmer, KDevelop and SourceNavigator. These support a variety of languages including Ada, Bash, C, C++, Java, Make, Perl, Python and x86 assembly.

Kernel

To have a working kernel for your particular Linux configuration, you need to find the correct source. The main kernel repository (http://www.kernel.org/) will most likely not run on your target system. This is because the development of the embedded Linux systems does not coincide with the development of the kernel. Each architecture will have its own particular kernel configuration suited for it.

Once you have selected the right kernel it must be configured for your needs. Configuration options include loadable module support, networking options, SCSI support, file systems, sound, console drivers and much more.

Typing ‘make config’ at a command line will give a command line interface asking you options one at a time. You may also find a .config configuration file suited for your needs, or one that can be easily modified. This will contain all configuration options for your system speeding up configuration. This can be used with ‘make oldconfig’. If you would rather not work with a command line, ‘make menuconfig’ will give you a text based configuration menu. To go even farther, ‘make xconfig’ displays a X Window configuration menu.

Root File System

The Linux root file system has evolved in such a way that at present it contains many different idiosyncrasies and redundancies. This does not mean, however, that that it is a poor or poorly organized structure. The root file system may appear confusing at first glance, mainly because many of the directories have similar and sometimes cryptic names, but each of the directories has a well-defined purpose.

The root file system contains the directories bin, boot, dev, etc, home, lib, mnt, opt, proc, root, sbin, tmp, usr and var. This basic folder structure is designed to be usable on all Linux machines. This structure can be simplified by removing the home, mnt, opt and root directories. These are designed for use in multi-user systems, therefore are most likely not needed in the embedded system design. The remaining ten directories are most likely necessary to your design. There are others that may be removed under certain conditions but for the most part, it is best to leave them in.

The purpose of these directories is fairly well defined. The bin directory for example contains the essential user command binaries. These are the binaries essential to both users and system administrators. Because this system will most likely not have multiple user accounts, this directory will contain all user binaries.

The boot directory will contain the static files used by the bootloader. Some bootloaders and bootloader configurations do not require this directory. This depends on whether or not your bootloader can get kernel images from your root file system before your kernel is booted.

For the most part, the rest of the directories names correspond fairly well to their purpose. The dev directory contains devices and other special files. The etc directory contains the system configuration and startup files. File directory lib contains essential libraries such as the C library. Directory proc contains a virtual file system for the kernel and process information. The directory tmp contains temporary files, sbin contains system administration binaries, and usr will contain most applications and documents useful to the user. And lastly the var directory contains variable data stored by daemons and utilities.

Bootloader Concepts

The bootloaders job is to load the Linux kernel. This is obviously a very important job, so when choosing a bootloader for an embedded system you should take your time. There are thousands of choices of bootloaders and many different configurations for each.

Your choice in bootloader should depend mainly on your architecture. Some architectures such as the x86 have a small number of well known and proven bootloaders. Choosing one of these will give you the best chance mainly because you will have a large system of support if problems arise. Some architectures don't have a standard bootloader, in this case you must rely on the bootloader supplied by the hardware manufacturer.

For the x86 there are two main bootloaders, GRUB and LILO. LILO (Linux Loader) has been around almost as long as Linux. This bootloader is very well documented and easy to use. GRUB (Grand Unified Bootloader) is the main bootloader for the GNU project. This option is equally well documented and the latest releases are available free for download as well as LILO. Other common options include ROLO, loadlin, EtherBoot, LinuxBIOS, blob, PMON and many more.

Networking Services

Networking is becoming a more significant part of embedded systems. Embedded Linux offers a large range of networking utilities and protocols to satisfy your embedded networking needs. Using Linux networking you can gain access to features like remote administration, network login through telnet, secure communications with SSH, web surfing with HTTP and dynamic configuration with DHCP.

SNMP is the network protocol used to allow remote administration. Almost every device that connects to a TCP/IP network has SNMP capabilities. This gives you the ability to monitor your system both remotely and automatically. There are a lot of different SNMP packages available. The open source world brings us Net-SNMP. This free package is fairly large and contains many different software components. For the purpose of embedded systems, however, the SNMP agent within Net-SNMP is really all that is important. This is the component which allows your system to be managed remotely.

Telnet is a simple lightweight protocol which will allow you to login remotely to your system. Although this service will provide you with all the functionality you need, it is not secure, and therefore not for use on the internet. For that, SSH is the appropriate tool. SSH is a very secure protocol that is widely used and available to anyone who want s to use it via OpenSSH. To use OpenSSH you will also need two libraries OpenSSL and xlib. Apart from OpenSSH there are two other options for open source SSH namely LSH and FreSSH. These tools, however, are not meant for use in production embedded Linux systems.

Network enabled embedded systems are including HTTP web servers more and more. The most common Linux HTTP server in use is Apache HTTP. This, however, is not suited well for embedded systems. Boa and thttpd are much smaller and therefore a better fit for the embedded Linux world. Both are available free, and offer similar support and features.

DHCP (Dynamic Host Configuration Protocol) allows for automatic network configuration of hosts. This means it assigns IP addresses to adjoining systems. An embedded Linux system can very easily be used as a DHCP server. In fact, the standard DHCP package is distributed free with most Linux distributions.

Embedded Examples

Linksys Wi-Fi Router

Linksys, in an effort to create cost effective wireless solutions has been toning a large variety of Linux based routers for a number of years. Linux gives them "the premium OS for inexpensive, feature-packed wireless networking" (Ewing). The Linksys WRT54G has a 125MHz MIPS processor with 16 MB of RAM.

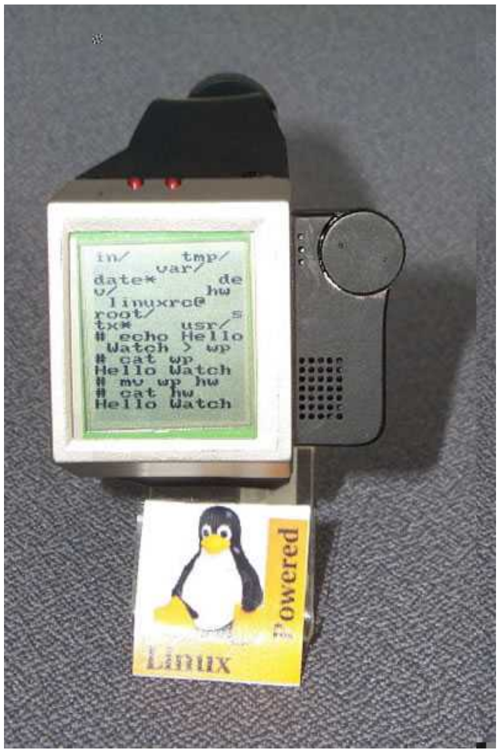
However, the most interesting thing about the WRT54G is that its source code is open source. Available for download, it is just over 145 MB and contains a complete MIPS cross development toolchain. This gives owners the ability to modify their Linksys box and personalize it. Someone has even ported the entire OpenSSH toolchain. The edition of his was hampered by size constraints, so some standard Linksys functions had to be removed.

Figure 3: Linksys WRT54G Wireless Router

The fact is, there are many wireless routers available running Linux firmware including routers by Belkin, Buffalotech, and ASUS.

The Linux Watch

The Linux watch is a classic example of trying to push the envelope. Just 56 by 48 mm, it runs the Linux operating system version 2.2. Its motherboard is a staggering 27.5 mm by 35.3 mm with its 8 MB Flash memory and 8 MB DRAM. The whole thing, including the touch sensitive display weighs only 1.5 oz.



```
in/     tmp/
      var/
date*     de
v/        hw
 linuxrc@
root/      s
tx*     usr/
# echo Hello
 Watch > wp
# cat wp
Hello Watch
# mv wp hw
# cat hw
Hello Watch
```

Figure 2: The IBM Linux Watch

IBM started the development of the Linux watch back in 2000. Now, in a more polished form, the watch includes Bluetooth technology and a high resolution Organic Light Emitting Diode (OLED) sporting a 640 by 480 resolution. In 2001, IBM teamed up with Citizen to get the 'final product' you see today. With its 32-bit RISC processor, the frequency can vary from 74MHz to 18MHz to help save power. While the watch may not be so practical, it is a perfect example of the versatility of Linux.

References

Addison, Darrick. IBM, Embedded Linux applications: An overview. 1 Aug. 2001. ASC Technologies Inc. 1 Apr. 2005 <http://www-106.ibm.com/developerworks/linux/library/l-embl.html>.

Ewing, James. Linux Journal. 31 July 2004. 2 Apr. 2005 <http://www.linuxjournal.com/article/7322>.

Kanellos, Michael. CNET News.com. 11 Oct. 2001. 4 Apr. 2005 <http://news.com.com/Linux+watch+Only+time+will+tell/2100-1040_3-274286.html>.

Yaghmour, Karim. Building Embedded Linux Systems. Sebastopol, CA: O'Reilly and Associates, 2003.