Embedded Linux Operating System

By

Uday Shankar Macha

Advanced Operating Systems

1.Abstract

Embedded systems present unique design challenges such as memory restrictions, booting from flash, and requirements for deterministic operation. This paper discusses some of these problems and describes how Linux answers them. This paper first discusses the requirements of a typical embedded system and then discusses how Linux meets these requirements. An interesting embedded Linux application - Dodge Super8 Hemi concept car is mentioned. Embedded Linux is then evaluated against Windows XP embedded.

2.Introduction

Linux was designed originally as a desktop operating system. Linux is an open source UNIX-like operating system developed by an online community of programmers centered around *Linus Torvalds*, a Finnish graduate student. But it was not a very user-friendly-operating system and lacked the level of polish and support of many commercial operating systems. Many companies like Red Hat were formed to help rectify this issue by providing a highly tested, robust, and commercially supported version of Linux. All of the modifications made are given back to the community as open source code to ensure that it is never branched from the community's development efforts, which are largely responsible for the unprecedented speed with which Linux has grown in deployment.
Because of the popularity of Linux, there are already several software vendors that provide releases of Linux aimed at embedded systems. These vendors typically offer technical support programs for Linux that is geared toward embedded system development.

3. Embedded Linux concepts

3.1 What is an Embedded system?

There is no kernel known as "embedded Linux". It's really "Linux, embedded". That is, a Linux kernel that has been embedded. Embedded Linux is always free, in a certain sense. "Embedded Linux" is probably more accurately described as a Linux kernel, libraries, utilities, and toolkit for building Linux into embedded systems.
It can be decided whether a system is an *embedded system* or a *general-purpose system* by examining its mission in life. An embedded system has a limited mission. It will be

designed to do a particular number of tasks. Those are the tasks it will ever do. But this does not mean that its mission cannot grow. An interesting example of an embedded system's mission growing as circumstances dictate is that of NASA's Voyager 1 and 2 spacecrafts. Both were supposed to explore two planets – Jupiter and Saturn. But the incredible success of those two first encounters and the good condition of the spacecraft prompted NASA to extend Voyager 2's mission to Uranus and Neptune. As the spacecraft flew across the solar system, remote control reprogramming gave the Voyagers greater capabilities than they possessed when they left the Earth [5].

3.2 How to choose a suitable OS for an embedded system?

There are several criteria used to evaluate and choose an operating system for an embedded system or product: availability, resource usage, software availability, feature set, reliability, and performance. Simply whether the operating system is available for the chosen embedded microprocessor can be the major decision factor. Because of the short time-to-market of many of today's embedded products, a port of an operating system to a new CPU or processor board may take too long. Having an operating system that already runs on the board saves the time and the expense of porting it.

3.3 Why Linux for Embedded systems?

The advantages include [3]:

- Linux is royalty-free.
- Linux already includes driver software for a huge number of devices and, because current drivers are well documented and include source code, developing new drivers is easy.
- The wealth of software tools included with Linux can substantially decrease development time.
- Linux's ability to run on generic hardware decreases the costs associated with purchasing development systems.
- Because Linux is being used extensively in universities, the pool of people who understand it- including its internals- is growing every day.

4. What does a typical Embedded system need?

1. Embedded systems rarely have a computer operator and most run as headless systems, meaning that there is no operator interaction required to start and run the system.
2. All operating systems require some resource usage in order to operate. The main resource required is computer memory. Some operating systems have a larger memory footprint than others. In embedded systems where both RAM and ROM is a precious resource, an operating system with a large footprint must not be selected.
3. When developing embedded applications, developers must use software tools such as compilers and debuggers and these tools cannot commonly run on the embedded target due to resource constraints. Embedded development requires running the software development tools such as a compiler, assembler, and linker on a host computer then downloading them to a target computer for execution. The host and target may have different CPU types. In such case a compiler and assembler intended for native use

cannot be used as a cross compiler and assembler. A cross compiler will have to be built or acquired that can run on the host and create machine code for the target.
Embedded operating systems, like desktop and server operating systems, are many times chosen not for their feature set but for the software available to run on them. Embedded systems designers need more software than mere operating system. They require protocol stacks, middleware, device drivers for special hardware, and sometime the actual application software. Whether or not this software is available for the embedded operating system they are considering without the need of porting is a very strong factor in their operating system choice. The feature set of an operating system becomes an important factor if the application software is going to be developed for the embedded system and is not already available. The availability of a certain feature may reduce the software development time. Of course, software that needs to be ported has been written assuming some set of features of the underlying operating system. If those features are not available, it may mean a long and costly port [2].
4. Reliability and performance are two features of an operating system that usually cannot be augmented with application or library level software. Running reliable application software on an unreliable operating system simply makes the application unreliable. The reliability of an operating system is based on how well it was conceived, designed, and coded. Most embedded systems require a higher level of reliability than desktop systems. So choosing a reliable operating system is important. Performance can also be benchmarked. It is usually straightforward to determine if the operating system can meet the embedded systems throughput requirements. What is harder to determine is will the operating system meet the real-time requirements? Real-time performance can be benchmarked, but usually the difficulty lies with determining the real-time requirements of the embedded application. The real-time characteristics of system software can vary greatly as the system is put under load. Often it is discovered that an operating system was the wrong choice because it could not meet real-time requirements that are not discovered until near the end of product development [2].

4.1 How does Embedded Linux provide these features? [2]

1. Linux in Headless and Diskless Operations

Linux was designed originally as a desktop operating system and it follows the UNIX model. A text console is used to interface with the computer operator. The console is used for controlling booting the operating system, giving the system commands, and monitoring the system's activity. Embedded systems rarely have a computer operator. So an embedded Linux must be setup not to rely on one. Linux like all flavors of UNIX uses the file system to store and locate executable programs, and persistent data. Even system devices are referenced by special files on in the file system. A file system on some type of disk device is an integral part of the Linux execution environment.
Making a headless Linux system is not too hard. The console device can simply be a null device. There is no operator interaction required if the startup scripts simply start the application programs instead of login processes. Monitoring can be done remotely if needed by setting a pseudo TTY as the console once a remote login is established. Linux

does support graphical user interfaces with X-windows. But X is strictly optional and not required for the operating system to run.
The problem of running without a disk can be solved in a few ways. There exist Flash memory devices that emulate IDE disk drives. These devices can be physically very small, not much bigger than the IDE connector itself. Linux also includes a RAM disk driver. This is a device driver that uses some amount of available system memory to emulate a disk drive. A RAM disk can be used for data files that do not have to be persistent across a system reboot. It is also possible to use the RAM disk driver to access a file system in ROM. If the embedded system has Flash memory that does not emulate an IDE drive, a flash file system can be used to treat the flash memory like a disk.

2. The Linux Memory Footprint

By default Linux has a big memory footprint, too big for most embedded systems. The Linux distribution is hundreds of megabytes. A typical Linux kernel is 1.5 megabytes uncompressed. RAM use by the kernel for a default configuration is over 4 megabytes. Linux also requires a file system for operation, and if the embedded system does not have a disk this means more memory will be used in the form of a RAM disk.
The solution to the memory footprint issue of Linux is through careful configuration of the Linux kernel. The Linux kernel is modular, and to a certain degree unused kernel facilities can be configured out. Also, kernel resources can be configured such as the maximum number of processes or open files on the system. By configuring the operating system modules and resource, the size of the ROM and RAM footprint can be reduced. It is possible to configure an "off-the-shelf" x86 Linux kernel to be 259K uncompressed. A minimal ROM disk to provide a file system for this kernel can be as small as 102K. The total RAM usage can be under 4 Mbytes. A Linux capable of TCP/IP networking is larger. The kernel with networking configured is about 370K, and the required ROM disk is 740K (both uncompressed). RAM usage in this configuration of Linux is under 8 Mbytes. This is not as small as an embedded RTOS can achieve but small enough for many embedded devices.

3. Software-Development Tool Support for Linux

Most Linux software development is done natively. Programs are written and executed as well as debugged all on the same Linux system. Once the program is working its executable image or a "package" containing its executable image can be copied to other Linux computers for execution. This method of software development can be used for embedded systems as well providing the embedded system has sufficient memory and disk space resources to run the development tools. This typically is not the case, however. So cross development is required.
For cross development, the popular GNU compilers can be used. If the host and target CPU byte order is different such as an x86 host and a PowerPC target, sometimes bugs will surface. When using cross development, some method of downloading a kernel image to the target must be chosen. If the target can access a disk drive (and can boot from disk), the drive can be mounted on the host computer to write the kernel and file system and then moved to the target for booting. If the target can boot an operating system over a network interface, the host computer can simply run a boot server and a file server and the target can do a network boot. However, if the target can only boot

from ROM or Flash memory, a special utility may be required to create a single load image given a Linux kernel and a file tree.

The easiest way to get a workable cross development environment for Linux is to pick up one of the Linux distributions designed for embedded use. These distributions have pre-built cross tools for several embedded processor types. Any byte order related bugs have usually been fixed. These distributions typically include booting code for the target as well as tools to configure and build Linux runtime images that can boot over the network or from Flash.

Cross development also requires cross debug support. In a cross debug environment the debugger runs on the development host. The debugger included on Linux distributions is GDB, the GNU debugger. GDB can run native or cross. In native mode GDB executes or attaches to the executing application to be debugged. GDB then can examine the variables and data structures of the application program or library through the /proc interface. GDB can also trace the program, be notified of signals being sent to the executing program, and send signals itself. GDB reads the symbol information from the executable file of the program and can also display the source from the programs source file. In a cross debug environment the debugger executes on the host development computer, and the application program runs on the target computer. The debugger reads the source and executable file on the host to get source code and symbol table information. The debugger communicates to an agent running on the target in order to examine variables and data structures and to control the process being debugged. The agent for GDB is the GDB Server and it is a lot smaller than GDB itself. GDB communicates to the GDB server either through a TCP/IP connection or through a raw serial connection.

The Linux kernel and device drivers cannot be debugged the same way as a user application. Setting a breakpoint in the kernel or otherwise stopping execution of a process while it is running in the kernel would cause the debugging agent to stop. Complete debug ability for the Linux kernel and device drivers is not supported in most desktop distributions of Linux. Some embedded distributions do support full kernel debugging. In this case GDB attaches to a debugging agent in the kernel usually through a low-level serial connection. This way the kernel or a device driver can be single stepped or breakpoints used even in interrupt routines. The only thing that cannot be debugged this way is the kernel debug agent itself.

Third-Party Software Availability for Linux

There is more and more software available for Linux, not just open-source software, but third-party application programs and protocol stacks as well. The number of device drivers available for Linux is large and growing and the source code for Linux device drivers is almost always free. Linux has a good reputation for reliability on servers, and it has good network and file system performance.

Not only is the source code to Linux free, but also runtimes of Linux are royalty free. For high volume embedded systems with tight per unit cost constraints, having no royalty payments for the embedded operating system is very attractive. It saves money that would go to royalty payments and eliminates the overhead for keeping track of the number of runtimes shipped.

The programming interface for Linux is UNIX. A major reason Linux has so much software available for it is that Linux uses the UNIX application programming interface as well as UNIX object and executable file formats. Much of the public domain and open source software of the last two decades has been written for UNIX and is now available for Linux. Also, third-party software developers have been providing products for flavors of UNIX for years such as AIX from IBM, Solaris from Sun Microsystems, and HP/UX from Hewlett-Packard. It was a simple matter for these software vendors to port their products to Linux. This also means that there are programmers experienced on writing UNIX software that already have the expertise to write Linux software. Because of the popularity of Linux, there are already several software vendors that provide releases of Linux aimed at embedded systems. These vendors typically offer technical support programs for Linux that is geared toward embedded system development. This support includes bug fixes for Linux versions that may be years old and help with ports to embedded devices.

4.Linux Reliability and Performance

Linux is not a real-time operating system. The Linux kernel uses a fair share scheduling algorithm designed for time sharing, not a strict priority preemptive scheduler or some other scheduler suitable for real-time. But that is the least of the Linux real-time problems. The Linux kernel is neither preemptive nor reentrant by user processes. If one process is using a kernel facility no other process can execute until the process finishes or waits on something. This means the worst-case response time of any real-time task running under the Linux kernel is longer than the time it takes to execute the longest stretch of kernel code before a wait. This time is unknown. The worst-case response time is further increased by memory cycle stealing by DMA devices and the CPU cycle stealing for the execution of interrupt handlers. There is no bound on the amount of CPU time used by interrupt handlers under Linux, and these run at a higher priority than any user process. Practically, a user process running on a very fast Pentium computer that is running a heavy load my take hundreds or even thousands of milliseconds to respond.
There are several ways to deal with the problem that Linux has poor real-time performance - ignore the problem, or run real-time applications under an RTOS with Linux itself as a separate task as in the case of RT-Linux, or run a Linux compatible RTOS kernel instead of the Linux kernel. The most popular approach is to ignore the problem. In most cases embedded developers have a problem identifying their real-time requirements at the beginning of a project anyway. This method my work if the application has extremely loose real-time requirements. If it is discovered that Linux is not meeting a real-time requirement then one of the other approaches can always be taken.

5. An Embedded Application

Dodge Super8 Hemi concept car [4]

The vehicle's internal computing architecture consists of four Ethernet-networked PC-compatible computers. Three of the computers serve as passenger terminals -- one in the

front seat, and two in the rear (one for each rear passenger). The fourth functions as a network server and communications gateway.
Each computer contains a miniature (PC/104 based) PC compatible computer board running Red Hat Linux 6.2. At the moment, the concept car prototypes contain large amounts of system RAM (128MB) along with multi-gigabyte disk storage, in order to ease the pain of the software developers.
The front-seat system has a 6.4" LCD display and uses pushbuttons and speech recognition as user input controls, to discourage drivers from taking their hands off the wheel. Voice commands let drivers keep their eyes on the road and hands on the wheel while controlling the in-vehicle audio system, climate control, security system, and phone, or while accessing remote information or devices via the Internet. The two rear-seat systems have 8.4" LCD touch-screen displays, mounted on arms attached to the front seat backrests.
Off-board communication is managed by the server/gateway computer system, and makes use of a combination of three wireless interfaces -CDPD cellular modem, Ricochet wireless, and IEEE-802.11 wireless. 802.11, being limited to short-range distances, provide a convenient means of transferring data to and from the owner's home computing environment when the vehicle is parked at home. Its primary use is to load up the car system with passengers' favorite multimedia files, email, and other data.
In addition to the Linux operating system, a certified standard Java Virtual Machine (JVM) from Sun equipped with Espial's DeviceTop environment provides a software platform for Java-based applications. All of the system's software components are written in Java, for ease of development and maximum portability. Espial's Java-written browser and email client are used in the prototypes, along with a host of DaimlerChrysler-developed in-car applications.

Why Linux?
The project team chose Linux primarily due to the requirement for complete and easy access to all low-level drivers, which is much easier and simpler to do with Linux than with most other operating systems. Also, the stability and reliability of Linux were considered strong pluses, in comparison with other options [6].

6.Comparison of embedded Linux with Windows XP embedded

Windows XP:

Windows XP is not as good an embedded solution as embedded Linux for the following reasons [1]:

Key Point	Summary
Memory Footprint	Windows XP has a memory foot print between 5 and 15 Mb where Embedded Linux has a memory footprint of 259Kb.
Performance	The market has proven that Linux offers performance superior to or equal to Windows for Servers. Given the additional factors against Windows XP for embedded (Size and Complexity), this comparison will be more in favor of Linux for embedded applications.
Maturity	Linux subscribes to an OS model proven through 40 years of innovations. Interoperability issues, performance and general design have had an extremely long time to be tried and improved. Moreover, all these improvements have been done in a very diverse set of platforms.
Networking	All major networking protocols, security features and extensions are available for Linux. In fact, many are implemented on Linux before other platforms.
Configuration	Linux is highly configurable, having been developed and deployed in memory limited environments, in comparison to Windows XP, which has operated in memory-hungry monolithic environments.
Security	Open source nature of Linux allows "many eyes" approach to be used to incredible effect. Security protocols, in particular benefit greatly from this approach because their design and implementation is well documented and understood. A very stunning example of this is the NSA's recent release of a secure Linux.

Cost-Effectiveness	Development in any environment is the greatest expense. Having a diverse community for testing and deployment figures greatly in the success of Linux. Also, because of the highly custom nature of many embedded solutions, the highly configurable nature of Linux makes it particularly cost-effective.
Development Tools	While Windows XP is primarily constrained to development under an IDE environment, Linux provides the powerful UNIX development environment in addition to IDE environments.
Reliability	The deployments speak for themselves. Windows is rarely considered for mission-critical applications where Linux is routinely considered for them.
Support	Microsoft provides a single source of support for their product, limiting competitive offerings. In the Linux world, however, there are many choices among vendors who will provide support, solutions and software.
Innovation	Because of the open nature of Linux source code, it has become a nexus of activity in regards to innovative computing to a far greater degree than any Windows product.

7.Conclusion

Although Linux was not designed for embedded systems, through careful configuration and by using some of the work of the embedded Linux vendors, Linux can be successfully embedded in hardware products. Linux is emerging as more than just a technology but also a platform for embedded applications. Linux does have some problems in the areas of licensing issues, memory footprint, software development, support for embedded processors, and real-time characteristics. Linux is not public-domain software; it is licensed according to the GNU Public License, which has a strict set of rules for use. But because of its popularity, Linux will be considered for more and more embedded applications. Because of its open source nature and the availability of third-party software, Linux will be here for a long time to come.

8.References

[1] LynuxWorks white paper, "*Embedded Linux Towers over Windows XP Embedded*".

[2] LynuxWorks white paper, "*Successfully Using Linux and Open Software in an Embedded System Design*" by Greg Rose.

[3] LynuxWorks white paper, "Why Linux in the Embedded Market?"

[4] *Behind the Wheel with Linux* by Richard Vernon, embedded Linux journal September/October 2001.

[5] Lombardo, John, "*Embedded Linux*", Indianapolis, New Riders, 2001.

[6] The Embedded Linux "Cool Devices" Quick Reference Guide
http://www.linuxdevices.com/articles/AT2478437967.html