An Overview of Embedded Linux

Andrew Tucker
ast@halcyon.com
CSE 585
March 2000
Abstract

In this paper I will discuss the current state of embedded Linux. I'll outline what Linux is and what characteristics make an embedded Linux system different from a desktop system. I will then discuss details in six key areas:

- Configurability
- Real time performance
- Graphical user interfaces
- Development tool support
- Economic considerations
- Technical support options

Following that I will compare the real time performance, OS image size, and RAM requirements of embedded Linux with that of Windows CE. I'll then discuss some retail devices based on embedded Linux and conclude with some general observations on embedded Linux.

Intro

Unless you've been living under a rock for the last few years, you've no doubt heard of Linux. Touted as the open source competitor to Microsoft's Windows NT, it has been much hype and discussion in both the software development and system administration communities, as well as mainstream non-technical news media.

When first introduced, Linux was specifically targeted to desktop PCs running an Intel 80x86 or compatible microprocessor. Linus Torvalds, the inventor of Linux, was quoted as saying “Linux will never run on anything but a PC with an IDE hard disk since that is all that I have”. Looking back from the state of things here in the 21st century, the quote is quite humorous. Today Linux has been ported to many different microprocessors and runs on platforms that don’t include any hard disk at all. These devices are not even general purpose computer systems and include things such as network routers¹, heart monitors², and data collection units for tracking tornadoes². It is these types of systems that have collectively come to be labeled as “Embedded Linux”.

In the next section I’ll look at what distinguishes an embedded Linux system from its desktop cousins. I will then discuss the features of embedded Linux in six areas: configurability, real time performance, GUIs, development tool support, economic considerations, and technical support options. I will then compare the size and real time characteristics of an embedded Linux kernel with that of a similarly configured Windows CE kernel.

I will then discuss three interesting devices based on embedded Linux: an optical telemetry system, the NetTel router, and the IBM handheld computer research project “itsy”. I’ll then wrap up with some general observations on the usefulness and future of embedded Linux.
What makes Linux ‘embedded’?

So far in this paper I’ve used the term “embedded Linux”, but I haven’t really defined what it means. If a system is booted from media other than a hard disk does that make it an embedded system? Are all systems booted from hard disks not considered an embedded system? This section will lay out five distinguishing properties of an embedded system that answers these type of questions. Note that none of these properties are specific to Linux, but are general characteristics of all embedded systems.

Historically, the qualifications of embedded systems have been a large, controversial gray area. Many a heated discussion has involved whether or not a given system is “embedded” – especially at parties with free alcohol and a high number of geek attendees. The properties I will describe shortly will certainly not satisfy everyone. They are general rules of thumb, and are neither necessary nor sufficient to unambiguously distinguish all embedded systems.

The five properties that are generally found in an embedded system are:

- Diskless media for booting and storage
- Lack of BIOS
- Footprint and runtime memory restrictions
- Memory management
- Dedication to a small number of tasks

Bootstrapping is one the first areas that has to be addressed when you are first getting an embedded system up and running. What happens first when the hardware is powered up? Users and developers who have only dealt with desktop computer systems rarely, if ever, have to think about this topic but it is of the utmost importance in an embedded system.

Most embedded systems boot from some type of diskless media. The media can be something like Flash memory, which is read/write, or read only memory such as an EEPROM. Alternatively, the device may not boot from internal media at all, but instead download the operating system image over a network when it is powered up. RAM can be used to store data while the system is running, but a persistent, non-volatile media such as Flash needs to be available to store data while the system is powered off.

The reason that embedded systems typically don’t use disk based media such as a hard drive is for mostly for power consumption and speed reasons. Hard disks consume a lot more watts that are required to drive Flash memory and are significantly slower. Economically, hard disks certainly beat Flash in a cost per byte comparison but embedded systems rarely needs the multi-gigabyte storage of today’s hard drives and a few hundred kilobytes of Flash is a much better solution.

The Basic Input/Output System (BIOS) of a PC is a significant piece of software that goes completely unnoticed by most people. Its job in life is to initialize the built-in hardware in the PC and provide primitive operations for communicating
with that hardware. A typical desktop OS, including Linux, is built on top of the BIOS and takes full advantage of it. In an embedded system you don’t have the luxury of BIOS and you must manually initialize your hardware to a workable state before the OS kernel starts running.

This isn’t as big a problem as it seems. Very few embedded hardware platforms are as complex as the motherboard architecture of a typical PC and they are usually much easier to configure. Many times the manufacturer of the hardware provides sample code that quickly is adapted to get the system up and running.

As we mentioned earlier, embedded systems don’t need the huge amounts of memory and storage that is present on most desktop. Some embedded systems run with only a few hundred bytes of RAM and ROM, and most don’t utilize more than a few megabytes. This is a far cry from the seemingly infinite resources of desktop systems, and is one of the biggest stumbling blocks for desktop programmers transitioning to embedded systems.

Memory management is another big area that distinguishes an embedded system. Since storage requirements are tight, there’s no place for the swap file usually required by a virtual memory system. Additionally, embedded systems run on a variety of microprocessors, some of which do not have the memory management unit necessary to implement virtual memory.

As a side note, it is interesting to note that embedded Linux meets both these requirements. Any Linux system can eliminate the swap file requirement by setting the size to zero when the operating system image is built. Additionally, Linux has been modified so that it does not require an MMU. uCLinux runs on a variety 32-bit microprocessors that do not have an MMU and use only physical memory addressing.

Finally, an embedded system typically is tailored to perform a dedicated series of tasks. They may not have any user interface other than a few indicator lights. The user cannot install and run any custom software other than that which the system was originally designed to run. This is in direct contrast to a desktop computer system where users routinely install and run custom software packages and can accomplish tasks never considered by the system designers.

Conspicuously missing from this list is real time performance. Although it is a hot topic in the world of embedded systems it is not a general requirement. Many embedded systems have no hard or soft real-time requirements and are perfectly satisfied with general, nondeterministic performance of the OS. Embedded Linux does have the option of using a real-time subsystem, as we’ll discuss in a later section.

**Details of embedded Linux**

In the next few sections we will look at the details of embedded Linux in six specific areas. The areas were chosen specifically to try to cover the end-to-end
usefulness of Linux in an embedded system. The goal was to address issues that would come up in all stages of the system lifecycle, from specification to maintenance. Some of the issues, such as economics and technical support, are decidedly non-technical but nonetheless important in the overall development of a system.

**Configurability**

Being able to tailor the OS image to the particular needs of a device is a very important feature in embedded systems. You don’t want to include megabytes of code that for GUI support if your system doesn’t require it. Whittling the kernel down to the smallest possible size that has all the support you need is one of the first tasks a designer faces after the hardware is up and running, and it can be a very difficult one.

Configuring an embedded Linux system is not much different from configuring a desktop Linux system. The OS build process uses a simple text file to select which components to include. The installable module support is then used to load the selected subsystems at boot time. It is also possible to dynamically load and unload modules during runtime, which allows you to tailor the system to use optional components only when necessary.

This configuration ability is very useful in an embedded system. Not only does it allow you to dynamically add and remove support for optional pieces, it also allows you to do partial updates. By splitting your system into several individually loadable modules instead of a single monolithic one, you can upgrade or add a new module without having to change the entire image. This is especially useful if your device is connected to a network that can be used for remote updates.

One drawback of Linux installable modules is the potential for stability problems. Since modules are dynamically integrated into the Linux kernel, they have full access to the OS and do not reap the benefits of memory protection. This is less of a problem on devices whose hardware doesn’t have an MMU. Also, since the system is in a closed loop the user can’t install new modules at their leisure and it’s easier to maintain the integrity of the system.

**Real Time Performance**

The term “real time” is bandied about a lot, so I’ll define what it implies in this paper before I describe the real time features of Linux. By “real time” I mean the ability to put deterministic bounds on the time the elapses between an interrupt occurring and the corresponding interrupt service routine to execute. Other items, such as context switch time and system clock granularity, are related, but not central, issues.

In general, real time performance can be grouped into two broad categories: hard real time and soft real time. Hard real time means that a late answer is a wrong answer. If the system does not respond to an interrupt within a fixed, predictable amount of time disastrous things can happen. Soft real time, on
the other hand, doesn't have any dire consequences associated with a late answer but relies on deterministic timing of interrupt handlers to achieve top performance. It's important to remember that although the actual performance numbers are important, real time characteristics are actually more about deterministic behavior than raw speed.

The bare Linux kernel is not real time. No guarantees are provided for deterministic interrupt handling times and the numbers can vary greatly depending on what the system load is when the interrupt occurs. RTLinux was designed to add hard real time characteristics to the Linux kernel.

First released in February 1997 by Michael Barabanov and Victor Yodaiken of the New Mexico Institute of Technology, RTLinux is not a separate, modified version of the Linux kernel. After considering this possibility, it was deemed to be a maintenance and compatibility nightmare. Instead RTLinux was designed as a small, independent real-time kernel and a set of replacements for the Linux kernel routines that enable and disable interrupts. RTLinux also supplies an API that allows the user to create tasks in the real-time kernel and communicate with processes in the Linux kernel via FIFOs or shared memory.

Figure 1 graphically depicts how the system works. When RTLinux is included and the Linux kernel calls the CLI or STI to disable enable interrupts, respectively, the request gets routed to an RTLinux routine that remembers the current Linux interrupt state. When interrupts occur they are always handled by the RTLinux kernel and are then passed on to the Linux kernel if they were not handled and interrupts are enabled. This allows RTLinux to always react to every interrupt without the Linux kernel getting in the way. This also gets around the problem of the Linux kernel not being able to handle nested interrupts since RTLinux always sees the interrupt first and can ensure that the kernel is never reentered.

RTLinux is not the only player in this arena. Real Time Application Interface (RTAI) takes a similar approach in extending the existing Linux kernel with real time features. They have a few features, such as periodic timers, that are not present in RTLinux but do not have the wide popularity of RTLinux. If you believe the rumors, these two products will merge in some future version.

There are some problems with the features present in both RTLinux and RTAI. First of all it is clear that the features were bolted on as an afterthought instead of being an intrinsic part of the design. Although RTLinux does an admirable job of accomplishing it's goals it tends to be useful in certain classes of applications such as data acquisition rather then being more generally applicable. Another issue is that RTLinux does nothing to address the problem of priority inversion, which can be the cause of serious bugs that are hard to reproduce and track down.
Graphical User Interfaces

As the desktop computer has come to rely on GUIs rather than text based interfaces, many embedded systems have also been adopting them. A GUI in an embedded system may come in many different flavors including small resolution LCDs, an optional VGA output port, or a fully integrated screen with either vertical or horizontal orientation. Although similar in concept to desktop GUIs, embedded systems have some requirements, such as the kernel footprint, that are uniquely their own.

Linux has always suffered from a fragmented market in the area of GUIs. Although the X-Window the closest thing to a standard, there are many popular competitors including emulators for the Win32 API and the look and feel of Windows 95. Embedded systems are certainly free to use any of these systems that are targeted for the desktop, but there are several products that specifically target the unique needs of embedded systems.

Lineo Systems is one provider of a version of embedded Linux with enhancements and commercial support. One of the options that they provide is a custom toolkit for building an embedded GUI. Microwindows and NanoGUI are two public, open source projects that were merged to provide a free library for emulating the Win32 API for use on Linux systems. There are certainly other options – these are just the tip of the iceberg.

There is another, separate route that many embedded devices have chosen to take when providing a GUI. Since network and Internet connectivity are so common these days, many systems choose to include a web server and offer up
their user interface via HTML. This allows the user to access the device remotely from any location with any platform that has a web browser. It also places fewer requirements on the size and shape of the device since it doesn’t have to include a display.

**Development Tool Support**

Embedded Linux developers get the best of both worlds when it comes to development tools. Not only do they get to utilize the same GNU tool chain as their desktop brethren; they can utilize the same hardware debugging tools they have always used. You don’t need to change your oscilloscope or logic analyzer just because you switch over to embedded Linux.

The GNU collection of compilers, debuggers, assemblers and linkers represent a wide array of microprocessor support and one of the best proof of concepts that the open source community has. Not only are they time-tested and freely available, but the availability of source code means that if you want to port to a new processor you have the freedom to implement the compiler support if it is not already available. Very few other embedded operating systems can match this claim, and even fewer do so with little to no up front cash requirements other than development time.

There are a few snags. The GNU debuggers are mostly text based, with varying support for frameworks running on top of them for some platforms. Also, many people have found the GNU kernel debugger to be buggy and hard to use, which is especially frustrating for embedded system development where device drivers are very common. Another difference for users of uCLinux is a special executable file format. Since there is no virtual memory, the executable cannot have relocations and all branches are done with position independent code (PIC). This requires a special linker and modifications to the debugging tools if the processor you are using is not already supported.

**Economic Considerations**

One of the biggest attractions of Linux in general is that it is free. Although you can buy versions from many companies that have more user-friendly installs or additional features, the core Linux code is free for downloading and is subject to the generous GNU redistribution license. This is true for both desktop and embedded Linux, but the embedded space does have some additional issues to consider.

It is possible to incur royalty fees for distributing embedded Linux with your device, depending on your Linux supplier. Lineo’s Embedix has royalties that vary with the OS features and additional packages included such as their GUI toolkit and web browser. Also, device drivers for special hardware that are not included in the regular Linux distribution can be purchased from many different companies and have associated royalties. Although you could certainly write the drivers yourself, available resources and time to market can always influence your “build or buy” decision to go with the faster option.
Considering these factors, your embedded Linux product may not be royalty free after all.

Another concern that has sparked a lot of discussion in the Linux community recently, is the fact that RTLinux is based on a patent\textsuperscript{11}. Although the RTLinux creators have stated that no royalties based on the patent will be required, the details of the written agreement with Linux International is still being worked out.

**Technical Support Options**

One of the big concerns that many people have with an OS that is not supplied by a single company is that of technical support. With no one centralized organization to turn to when bugs are discovered or things go wrong will you be left out in the cold? I think that the situation with embedded Linux is actually better than that with comparable commercial systems.

The number one reason that embedded Linux is in better shape in this area is it’s open source nature. When you are tracking down a nasty bug there is no substitute for being able to step through the source code. Very few commercial operating systems make their source available to developers, and even fewer allow you to modify it to workaround problems or add features. The second area that Linux technical support excels it in availability of qualified experienced people to answer questions. The value of USENET and mailing list discussions has proved invaluable to the legions of programmers using Linux on desktop machines and it is no less so for embedded Linux. As Linux grows in popularity on both desktop machines and in embedded systems, the availability of developers for contract or hire grows as well.

Of course if you do want the comfort of a single name to run to when things go awry, embedded Linux has that as well. Lineo\textsuperscript{7} includes technical support with their custom commercial version of Linux, as do several other companies. Each adaptation has it’s own set of ups and downs, but it does give you a bit more stability standardization in your version of Linux. Of course, none of this comes for free. Custom adaptations of Linux from companies has a higher purchase cost and may or may not have runtime royalties associated with one or more of its components.

**Comparison of CE and embedded Linux**

Since size and performance are key, it’s important to see how embedded Linux stands up to a competitor. Table 1 compares the embedded Linux real time interrupt performance and kernel footprint and RAM usage with that of Windows CE.
<table>
<thead>
<tr>
<th></th>
<th>Worst Case ISR Latency (in microseconds)</th>
<th>Networking Kernel Footprint</th>
<th>Networking Kernel Minimum RAM Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedded Linux</td>
<td>15 Error! Bookmark not defined.</td>
<td>500 KB</td>
<td>4 MB</td>
</tr>
<tr>
<td>Windows CE</td>
<td>5</td>
<td>1.3 MB</td>
<td>4 MB</td>
</tr>
</tbody>
</table>

Table 1: Comparison of Windows CE and embedded Linux

In this comparison, “ISR Latency” refers to the time between an interrupt occurring and the start of the ISR. Also, the kernel used in both operating systems was a the minimal kernel with full networking support. Although I was not surprised by the kernel size differential, I was surprised by the apparent better real time performance of the, as of yet unreleased, Windows CE 3.0.

Real World Uses At a Glance

Of course, the goal of all this is to create some interesting and useful device. In this section we will look at two commercial devices and one research device, all based on embedded Linux.

The Optical Telemetry System\textsuperscript{15} is by far the largest of the three systems and lands squarely in the gray area we discussed in the section on criteria for being classified as embedded. Sporting a 810MB hard disk, lots of RAM and weighing 48 pounds, the OTS has four serial ports, an Ethernet connection, two 65Kb/s Sync interfaces, and 60 input/output ports for control and alarm signals. This system is a good example of the high end capabilities of embedded Linux, as well as its ability to support a wide array of hardware and communication channels.

The NETtel\textsuperscript{16} family of network routers are Linux based devices that have a significantly smaller footprint than that of OTS. Running uCLinux on a90MHz ColdFire processor\textsuperscript{10} the NETtel family of routers supports up 2 serial ports, 2 Ethernet ports, a PCI slot, and 2 USB ports in various configurations and uses LEDs to display status information. The product boots from 1MB of Flash ROM and has only 4MB of SDRAM. The ColdFire CPU does not have an MMU; thus the system does not use virtual memory addressing at all. As an interesting example of using the product for completely unintended purposes, the creators have shows how to use the display LEDs to drive an A/D converter and turn the router into an MP3 player\textsuperscript{17}. Truly ingenious!

The Compaq Itsy\textsuperscript{18} comes in an even smaller package than NETtel routers, but packs a big punch. Measuring only about the size of a credit card and as thick as a 9-volt battery, the Itsy runs a StrongArm SA-1100 at anywhere from 30 to 200MHz and has 32MB of RAM and 32 MB of Flash ROM. It also includes a serial, USB, and IrDA port for communicating with the outside world. One of
the main research goals of Itsy is power consumption and current models can recharge their internal batteries off of the power supplied by the USB connection.

Of course all this is running on embedded Linux. Using v2.0.30 as a base, the researchers added new drivers for their hardware and a specialized Flash file system. Also the kernel was modified to add special memory and power management features, something that could not be done with an OS that was not open source. Itsy is currently only a research project and is not currently available commercially. I would expect that it will be the basis of new products from Compaq in the not so distant future though.

This is just the tip of the iceberg of embedded Linux devices. It seems that a new one pops up at www.linuxdevices.com almost daily, each more interesting and unique than the last.

**Conclusions**

Overall, I think that embedded Linux has a bright future. The ability to run on many different processors, lack of the requirement for an MMU and extremely low cost are huge factors. Also, its popularity seems to be rising rapidly and there is a large installed base of developers rapidly gaining experience.

Of course every rose has its thorns. The embedded Linux is very fragmented and requires more research than other operating systems to find the configuration that works best for your design. It’s requirement of a 32-bit microprocessor and minimal footprint is still too big for some embedded devices. This is true of Windows CE as well and, although the two have a large overlap, embedded Linux seems to be winning the battle in terms of the number of devices that it can be adapted to.

The embedded system market has been exploding in the past few years and Linux is certainly in the heart of it. It will be interesting to watch as the story continues to unfold.
References

2. The RTLinux Manifesto, Victor Yodaiken
4. Inside real-time Linux, Jerry Epplin, DDJ, March 2000
11. http://www.linuxdevices.com, “RTLinux is Patented?”
12. Linux kernel v1.2.12, personal comm with David Bennett
14. based on a release build of the MINCOMM configuration in CE Platform Builder 2.12