

## FEATURE ARTICLE

Greg Ungerer

# Embedded Network Appliances with Linux

The world is full of embedded computing devices. They are used to control all sorts of everyday things, from automobile engines to home video recorders. Traditionally, these types of embedded devices have existed in isolation. They are standalone computing devices that silently do their programmed tasks. More and more of these objects are being connected to networks, and even the Internet. They are actively communicating with each other and the world at large. It is truly becoming a connected world.

This trend has pushed both hardware and software technology in a direction that allows for building low-cost embedded platforms that are networked. These thin servers are perfect for embedding in intelligent devices, devices that are information-aware appliances. They not only perform their own specific control functions but can also interact over a network to allow a whole new set of distributed features.

Embedded devices connected to a network, particularly the Internet, take on a whole new range of responsibilities. They now need to be capable of talking standard protocols, like TCP/IP and its associated application set. They also need to be sensitive to security, authorization, and access issues. These extra responsibilities mean that embedded devices are becoming more complex and require more raw computing power. That makes them slower to build and more difficult to manage and maintain.

There is a revolution occurring within the embedded systems arena that is making low-cost embedded network appliances a reality. A combination of inexpensive, high-performance processors and powerful software platforms that use operating systems like Linux are making it possible to network everything.



The Linux operating system and GNU application set is a popular choice for building powerful, reliable network servers. Given this strong network background, Linux and GNU make a perfect platform for building small, low-cost embedded network appliances. Linux is typically used on interactive, general-purpose workstations and large servers. There are a number of unique challenges involved in getting Linux and the GNU application set operating happily in embedded hardware (i.e., hardware that has only minimal resources).

In this article, I will discuss the advantages of using Linux in small network devices. One example system will be presented in some detail. The key points of the construction process will be examined, and the overall techniques and decision justifications will be provided to show the power of extending Linux into small environments.

The concept of a connected world has created a whole new front of embedded opportunities and Linux is at the forefront for small network devices. The trend is a revolution of low-cost, high-performance appliance possibilities for the things you come in contact with everyday. It's a small world after all as we see the many ways to interact over a network.

## A CLOSER LOOK

An interesting combination of attributes make up low-cost, embedded, network appliances. This unique mix adds new levels of functionality to embedded systems, making new things possible.

Cost can be measured in many aspects of a system, including the cost to purchase, support, maintain, deploy, and more. All of these are important, and for the emerging network appliance, this applies equally to all areas. Any appliance should by its nature be specialized for a task or set of tasks, and do so at the best possible price point.

Many embedded applications are cost-sensitive, especially those destined for mass market consumer appliances. This price pressure will undoubtedly continue as intelligent appliances become more commonplace. The low-cost requirement means that hardware solutions are becoming more integrated—the overall component count is lower and the individual components are capable of much more. Mass market volumes mean that advanced hardware component costs are dropping, and it is now possible to build advanced computing devices with a handful of chips. In minimal forms, these types of devices can be built for less than \$50 and used anywhere for any purpose.

Although you can build powerful appliances at a reasonable cost, it will always be true that personal computers (PC) and workstations will be more powerful. So when compared to the average PC of the day, network appliances may seem under powered, like bare bones computers. They will always be minimal systems designed to carry out a few specific tasks well, not general-purpose computing platforms.

To this end, low-cost embedded network appliances will have a minimal hardware base. They will have smaller capacities of RAM and ROM storage and typically no form of mass storage device (hard disks, CD-ROMs, etc.). The CPU will be slower and more specialized than those used in desktop or server computers. The peripheral device set will be more

limited and often not extensible. These are important characteristics that facilitate constructing embedded appliances at a low cost.

Connectivity to the outside world is a vital aspect of networked appliances. Some type of network interface will be required. What exactly is appropriate will depend greatly on the target purpose of the appliance. Two common interfaces will be asynchronous serial (which will most often be used with analog modems) and Ethernet. Ethernet has become the dominant choice for all types of LAN networking, including demanding factory floor applications. Ethernet is also a popular choice for access to new WAN connectivity options like ADSL and cable modems. There are many other possibilities though, including parallel ports, synchronous serial, fiber optic, wireless, and so forth.

Another important issue with any sort of programmable device is its configuration interface (i.e., how the embedded device can be set up, configured, and used). Many embedded systems have no user-configurable interface, and others may require, or at least support, extensive configuration options. This problem is often made simpler by the fact that the device is networked. The simplest option is to use a small web server inside the embedded network appliance. This has many advantages, most importantly that web browsers are ubiquitous and well-understood by most people. There are, of course, other options ranging from a classic command line configuration method to more advanced network management protocols such as SNMP.

There are many other constraints placed on embedded devices. Often there are size or physical orientation considerations. Some embedded devices must be located inside their host equipment. Often this goes hand in hand with power supply and consumption requirements. These constraints can greatly restrict the choice of components used and ultimately affect the construction cost of any embedded system.

The overriding factor that affects

the cost of any embedded system manufactured in volume is the cost of actually building the hardware. The up-front engineering costs are easily amortized over the lifetime of the product. But the real problem is the time to market, the time from initial product conception to shipping the final product in volume. There is a clear trend with modern, complex, networked appliances that the bulk of development time is spent on software engineering, creating the software system at the heart of the embedded network appliance.

As you can see, there is a mix of features and constraints that make embedded network appliances unique. There is no simple answer as to what the best tradeoffs for performance, cost, and functionality will be. Each product that incorporates an embedded network system will differ.

There are many places where an embedded network appliance may be put to work. Some examples are:

- Personal Digital Assistants (like the Palm Pilot)
- routers
- network printers and print servers
- communications and remote access servers
- set-top boxes
- network web cam
- portable audio player
- remote data logger
- intelligent factory automation devices

Not all of these devices fit into the low-cost category, although in the future they may be classified as such.

## WHY LINUX?

There are many specialized embedded operating systems available today. They range from freely available source code systems, like RTEMS and uC/OS, to vendor supplied and supported proprietary systems, such as VxWORKS and QNX. And now, there is a new player in this arena, Linux.

There are a lot of reasons why Linux is a good choice for embedded network appliances. When trying to build low-cost systems, it helps

greatly when the core software can be used for free. Per unit licensing fees for an embedded system can increase costs for each unit built. There is also the engineering edge gained by having all the source code available while developing. Not only does it make debugging easier, it also means that the kernel and utilities can be tailored to the exact requirements of the system.

Another key advantage of Linux is that it supports many target microprocessors, computing platforms, and a huge range of peripheral devices. It cannot be overstated that time to market can be greatly reduced by this wide range of hardware support. This also translates into giving the hardware designer a wider choice of components to draw from when designing an embedded network appliance. Contrast this with proprietary systems where the operating system will only be available on the processors that the vendor chooses to support.

The network stack and associated utilities are critical to the success of a networked appliance. Linux is strong in this department. The stack is mature and the utility set is second to none. The supported protocol set is huge. There isn't any other operating system that can boast a wide range of network infrastructures like Linux. The network infrastructures make it easy to add powerful network support to any embedded appliance.

Aside from the large base of network support code, there is also an extensive base of other software freely available that will run under Linux. This gives you no shortage of options when constructing a software base for the embedded network appliance. The range of software options is mind-boggling, including simple shell type utilities, scripting languages, scientific and mathematical libraries, and database engines.

Another bonus when using Linux as the core software engine for an embedded appliance is that the support tool set of compilers, linkers, and debuggers is also freely available. The overhead for creating a GNU tool chain is small and can be set up for free. Again, the advantage of having

access to the source code will be self-evident to any embedded software engineer who has struggled with compiler bugs!

The fact that it is a popular operating system illustrates another advantage of using Linux. Development within the Linux community is fast and furious. And, those developments are shared with the community at lightning speed. Bugs are fixed quickly and new hardware is supported as soon as possible. This means that Linux usually has the latest and greatest features.

No one choice will be perfect for every situation, and this is true of Linux as well. For instance, Linux has a larger memory footprint than typical embedded operating systems. This means that the device will require more RAM. Hardware prices are continually falling, especially in the commodity RAM market, so this is actually less of a problem than it may appear.

An important characteristic lacking in Linux is that it is not a real-time operating system. There are no guarantees as to how long a particular event will go unserved, or any strict prioritization of event service order. There are groups that have successfully added real-time support primitives to Linux so this also can be solved. This problem is also alleviated somewhat by the fact that most embedded systems have a small, well-defined set of tasks that can be carefully tailored to minimize problems. Of course, this still does not make up for not being a real-time system.

Linux is a general-purpose interactive operating system, and so it is designed around a file system, at least a root file system. Generally, small, specialized embedded systems do not have built-in file systems. To use Linux in an embedded environment, you will need to provide a file store of some type. You will also need to have file system code within the kernel to manage this, thus increasing your memory footprint. As it turns out, this can be a bonus to the embedded network appliance. With the widespread use of web servers as a means

of communicating to the network and the world, it is now commonplace to have web pages within the embedded system. The natural paradigm is to store these as files just as they would on a normal workstation. In the long run, having to include a file system is an asset rather than a liability.

Although Linux microprocessor support is strong, it is within a similar set of CPU types. Currently, Linux only supports 32- or 64-bit processors, and only processors that have virtual memory management. This restricts your CPU choice for building embedded network appliances, and can drive up the building cost of the device.

However, there is a solution. In theory, it appears that Linux is an excellent choice for a large set of low-cost embedded network appliances.

## EXAMPLE HARDWARE PLATFORM

The Moreton Bay NETtel is a family of low-cost VPN Internet routers designed to provide small Ethernet networks with simple secure access to the Internet and external networks via the Internet. Different NETtel family members provide a range of Internet connectivity options, starting from analog dial-up modems, to higher speed WAN interfaces, such as cable and ADSL modems. Other NETtel family members can be used for custom embedded applications by offering standard peripheral interfaces such as USB host ports.

The NETtel hardware platform is designed for low-cost production with consideration given to the choice of CPU, RAM, ROM, flash memory resources, and peripheral device set. But, the overriding design constraint is the final product price. The NETtel platform is powered from normal household or office electricity, so there are no power supply or consumption restraints. Similarly, there are no physical size limits. However, the usual electronic device emission standards do need to be met.

At its core, the NETtel hardware platform consists of the following components:

- Motorola ColdFire 5307 CPU (90-MHz, 8-KB internal

cache)

- 4-MB SDRAM
- 1-MB flash ROM
- 10-MBps Ethernet
- two RS-232 serial ports

There are a number of options available on NETtel family members, including:

- second 10-MBps Ethernet
- one PCI slot
- two USB host ports

Note the small core component count. By using highly integrated devices, the count and manufacture cost is kept low. Notice also the small amounts of RAM and flash memory. Minimal resources are a typical characteristic of all low-cost embedded systems, and the NETtel is no exception.

It is also worth noting that there is no form of mass storage on the NETtel. It keeps all of its operating code and configuration information in the nonvolatile flash ROM device.

The heart of the NETtel is the Motorola ColdFire CPU. It integrates a number of standard logic parts onto the chip to reduce the overall system part count. It includes such things as cache memory, interrupt controller, DMA engine, timers, memory controller circuits, and device select logic. In traditional workstation and PC computers, these types of functions exist in one or more support chips outside of the CPU.

The ColdFire is a descendant of the popular Motorola 68000 processor family. Those of you familiar with the 68000 will feel right at home with the ColdFire processor. The instruction set and addressing modes are a subset, and the overall CPU core is similar. The processor register set is identical, although the stack pointer implementation differs.

ColdFire processors are 32-bit devices, but one point of concern is that they do not have any virtual memory support. This is going to be a problem with Linux. Fortunately, there is a port of Linux to CPUs that do not have memory management units (i.e., microcontroller Linux, or

µC-Linux).

## RAM AND FLASH MEMORY RESOURCES

The NETtel uses its 1 MB of flash ROM to store the operating system kernel, the root file system, and a configuration file system. It also boots from this flash memory when it is powered on. Flash memory is nonvolatile (it does not lose its contents when power is removed). It can also be reprogrammed in circuit, this means that the NETtel can be loaded with newer revisions of its operating code (firmware) while on the customer's network.

The NETtel contains 4 MB of modern fast SDRAM. This is the main operational memory of the system and is used to hold the running kernel and applications. It is also used to hold a RAM file system. This RAM file system provides a small, temporary file store for applications to use while running. It also provides a convenient place to store device logging and status information.

Generally, the smaller the memory size, the lower the cost. So, considerable effort will be spent to keep the kernel and application set as small as practical. For some applications it will also be necessary to cut down their functionality to conserve memory. Usually this simply means removing features that cannot be used in the embedded system.

## PERIPHERAL SET

The peripheral set was chosen to provide the required network connectivity as simply and inexpensively as possible. The serial ports are integrated devices in the ColdFire CPU. The Ethernet interfaces are provided by a low-cost SMC Ethernet part.

The option to add two USB host ports and the PCI slot provide excellent expandability options for the NETtel. Many off-the-shelf USB devices are useable in a simple networked fashion on the NETtel.

## THE BUILDING PROCESS

Building any type of embedded system is much different from building a typical application. This has made engineering embedded systems

a specialty skill that is both time-consuming and difficult. Traditionally, each new system is hand-crafted, using a mixture of existing and new source code. Normally, operating systems crafted specially for embedded systems are used and their programming interfaces inevitably differ from those of workstations.

Using a Linux-based embedded system alleviates a lot of the pain associated with programming an embedded system. Now the kernel component is almost identical to any workstation-based Linux system. All low-level hardware components that need to be created use the existing well-documented Linux driver model. Similarly, the application environment is also virtually identical. This means a familiar Application Programming Interface (API) and a large base of existing tools and utilities to draw from.

In the case of the Moreton Bay NETtel platform, the µC-Linux kernel provides a Linux interface that is completely compatible with workstation Linux. The primary utility set required has the same networking tools required for workstation Linux, they need only be ported to the ColdFire processor architecture.

A more detailed analysis of the methods used to create the NETtel reveals how the embedded network appliance construction process is simplified by using a standard operating system and utilities.

Embedded systems are generally not capable of hosting their own build environment. The hardware resources available in low-cost embedded systems do not allow for a suite of compilers, linkers, and debuggers to be present. The embedded system software and applications are almost always cross-compiled from a workstation environment. (Note that some embedded systems can and do have full compilation tool chains self-hosted, but the ones I'm examining here do not.)

When using a Linux-based kernel, the choice of tool chain is obvious. The GNU tools offer a complete set of compilers, assemblers, linkers, and debuggers. In fact, they are of a qual-

ity that is arguably as good as tools provided by any commercial vendor of software tools.

For the NETtel product, I used the GNU egcs V1.1.2 compiler. I chose it over the standard GCC compiler because its support for the ColdFire processors is more up to date. The Binutils V2.9.1 package provides the rest of the build tools, supplying the assembler, linkers, and object file manipulators. As a side note, the tool chain is configured to generate ELF format binaries, the same as those generated on a typical Linux workstation.

## MICROCONTROLLER LINUX

As I mentioned earlier, the ColdFire family of processors is tailored for use in embedded systems. This does not make them lesser siblings in the microprocessor family tree, but rather a different branch, specialized for a different type of platform.

The main issue to deal with now is that standard Linux only runs on systems that have virtual memory (VM) support. That is, they have a memory management unit (MMU), whereas the ColdFire does not. Fortunately, a port of Linux exists in Motorola 68k series CPUs that do not have VM support, microcontroller Linux ( $\mu$ C-Linux).

Microcontroller Linux is a port of a standard Linux kernel that doesn't need any form of virtual memory. It offers the same API set as standard workstation Linux. This is important because you want to draw on the large base of software that works with Linux.

There are a few important differences to note. For instance, only the *vfork()* system call is supported for forking new processes. There is no memory protection between user processors or the kernel. There are limits for the allocation sizes of memory regions. And finally, there is a fixed size application stack (cannot be dynamically grown).

As it turns out, these can be worked around easily. In many cases, no special changes are required to port applications to  $\mu$ C-Linux. By and

large, these limitations do not affect the bulk of the API set (i.e., the system calls available to user-level applications).

One of the main side effects of not having virtual memory is that debugging can be more difficult. It is simple for buggy program pointers to overwrite and corrupt other processes, or even the kernel memory space. This can easily lead to system crashes that are hard to diagnose and track down.

## PORTING $\mu$ C-LINUX TO A COLD-FIRE PROCESSOR

It is worth mentioning some of what was involved in porting  $\mu$ C-Linux to a ColdFire processor. There are a number of issues to tackle when porting an existing software system to a new processor architecture, especially with system-level software such as a kernel.

Initially, the effort is to get the kernel compiled into object code for the new processor. This is not as easy as it sounds! The Linux kernel contains copious amounts of in-line assembler. Even though the  $\mu$ C-Linux kernel runs on 68k-based systems, there are a number of ColdFire processor differences, which means much of this assembler needs to be reworked. In particular, the stack pointer differences and lack of addressing modes caused problems.

Next, some attention needs to be paid to deciding on an appropriate memory layout for the system. All peripheral devices are mapped into the physical memory space of the processor. Likewise, all running application processes and the kernel are mapped into the same physical address space. So, a model for what goes where needs to be determined. The kernel needs to be modified with this in mind because it ultimately controls all the memory.

The next consideration is code modifications to support the ColdFire built-in support peripherals. This includes support for the interrupt controller and timer, critical components of any operating system kernel. Similarly, support needs to be coded for the essential external resources associated with the processor. In par-

ticular, the flash memory needs special support. It has several distinct functions, from being the system boot device to storing nonvolatile configuration information.

Support for the wider range of peripheral devices connected to the hardware platform is generally in the form of device drivers. A new driver is required to support the built-in ColdFire asynchronous serial ports. Other devices on common ColdFire-based boards, including the Moreton Bay NETtel, are easier to support. Linux has such a vast range of device drivers that common devices, like Ethernet controllers, are usually already there. Then, it's only a matter of modifying the driver to cater for the memory addressing differences of the specific hardware platform.

Lastly, comes the application binary support. Having no virtual memory means that conventional binary file formats (such as ELF) are inconvenient to use on  $\mu$ C-Linux and the ColdFire processor. The  $\mu$ C-Linux group devised a new flat file format for binaries. Flat binaries contain the usual code and data portions of an application, but they also contain a set of relocation information. When a program is loaded for execution, it is relocated in the memory space allocated for it. This process is transparent to the application.

The  $\mu$ C-Linux kernel porting process was a challenge. But with a solid base, the follow-up efforts to build the application set became easier.

## PORTING THE APPLICATION SET

No kernel is useful on its own. An application set is required to take advantage of the kernel services. The amount of application software available for Linux is staggering. In any embedded system, such as our low-cost embedded network appliances, only a small set of software will be required.

The NETtel application set leans heavily toward network utilities. The core is the set of network configuration and management tools. This includes the commands that configure individual network interfaces (*ifconfig*, *pppd*, *diald*) and set up net-

work addresses and routes (route, ipfwadm, dhcpcd), network services (telnetd, httpd), and Virtual Private Networking (PoPToP).

There is also the usual set of utilities that you would associate with any Linux system—a shell (albeit, a small one), init program, login, ping, and more. There are not too many limits to what can be ported to  $\mu$ C-Linux on the ColdFire processor.

One noteworthy application suite is the web server and associated back-end management code (cgi-bins). It is the core of the user interface for administering the NETtel. Because embedded network appliances generally have no screen and keyboard, the logical method of configuration is through a web browser. A web server and pages can be a surprisingly lightweight and powerful method of configuration.

Given the serious network facilities of the NETtel and  $\mu$ C-Linux, it is no surprise that network file systems can also be used. Both NFS and SMB shares are supported and can be mounted and un-mounted using ports of the standard Linux tools.

## HARDWARE AND SOFTWARE INTEGRATION

The majority of embedded systems are designed for custom hardware platforms. This means that there is almost always a development phase that will entail porting and debugging the system and application software on the target hardware. If there's not a custom hardware platform, then almost certainly there will be some custom peripheral hardware to support.

This specialization of purpose and tasks makes embedded network systems unique. Once again, by using freely available operating software such as Linux, it is an easy job to get done quickly. All source code is available, and you have complete freedom over which parts of the kernel are modified to best support the embedded hardware platform.

Another distinct advantage of using Linux is that any newly created applications can be implemented and tested on a Linux workstation. This

makes application-level debugging simple and less time-consuming. The completed and tested applications can then be integrated into the embedded environment.

## DEBUGGING

Debugging frequently has to be done at arms-length from the hardware. There is often no console, or if there is, it is generally something simple like a serial port. There are none of the luxuries afforded by a cozy workstation environment.

Given the strong system-level software component of any embedded system, there are often complex timing issues at play. These can make debugging tricky. There is no easy answer for these problems, they just need to be worked through methodically. Modern processors like the ColdFire are helping by providing hardware-supported debug mechanisms. The Background Debug Module (BDM) of the ColdFire processors proved an invaluable aid for debugging the NETtel application. It provides the ability to remotely control the ColdFire processor from a workstation, giving access to the internal state of the processor and memory. At the workstation end of the debugging interface is the standard GNU debugger GDB.

One interesting side effect of using a system like Linux is that you can leverage the extensive system statistics and kernel feedback from such facilities as the */proc* file system. A wealth of information—memory usage and allocation information, running process details, network route and configuration information, and device driver statistics and internal state—can be gleaned from the virtual files within this special directory structure. Access to this data can make diagnosing system problems much easier.

By its general-purpose nature, a system such as Linux also means that interactive tools are easy to set up and use as debugging aids. For example, the normal command line shell can be a great tool. It provides you with the ability to interact easily with the system, launching programs as re-

quired or examining the system state at will. Being networked also makes this process easier. The network can provide the interface through which you can run an interactive shell session.

Debugging embedded systems is always a challenge. Using a general-purpose operating system can greatly reduce the time required to debug and stabilize a new system.

## FUTURE DEVELOPMENT

In the big picture, most products have unexpectedly long life spans. Inevitably, all systems need to be maintained and often updated to deal with new features or handle new environments and tasks. By using a Linux kernel and the freely available GNU tools and applications to build a network appliance, you have a head start in extending the product's functionality. As new protocols are devised and applications are developed, they can easily be supported. Adding these pieces becomes a simple porting exercise.

The ability to rely on the mass of Linux and GNU development means great savings in the engineering effort to deliver new features and maintain a reasonable time to market. This cannot be underestimated with today's fast-paced product delivery cycles.


Basing embedded appliances on modern, general-purpose operating environments such as Linux brings a depth of development and guaranteed future that no proprietary or specialized embedded operating system can match.

## FUTURE LOOKS BRIGHT

Linux makes an excellent choice for building embedded devices. It is also well suited for building low-cost appliance-style devices. The quality and reliability of the Linux operating system and ready supply of freely available tools and applications means that the design of powerful appliances can be realized in a short time, and they can be delivered to market at a low cost.

The wide range of available software also means that capabilities can be incorporated into embedded de-

vices that have not existed in them before. This includes everything from simple yet powerful capabilities (such as managing and upgrading them over the Internet), to delivering data such as video and audio, to providing an intelligent processing resource to a network. This can all be done in a safe and secure manner.

There is clearly a big future for Linux in small devices!

Greg Ungerer graduated with First Class Honors from the University of Queensland in 1989 with a Bachelor of Science degree. He has been working in the software industry for 10 years. He has a wide range of experience in the design, coding, testing, and maintenance of Unix (and Unix-like) operating systems, device drivers, networking, and embedded systems. During the last couple of years, he has been building embedded systems using Linux.

## SOURCES

NETtel

Moreton Bay

(408) 441-6492

Fax: (408) 441-6499

<http://www.moretonbay.com>

µC-Linux/ColdFire

<http://www.moretonbay.com/coldfire/linux-coldfire.html>

ColdFire CPU

Motorola, Inc.

(602) 952-4103

Fax: (602) 952-4067

<http://www.motorola.com/>

COLDFIRE

Circuit Cellar, the Magazine for Computer Applications.  
Reprinted by permission. For subscription information,  
call (860) 875-2199, [subscribe@circuitcellar.com](mailto:subscribe@circuitcellar.com) or  
[www.circuitcellar.com/subscribe.htm](http://www.circuitcellar.com/subscribe.htm).