Why Use Linux for Real-Time Embedded Systems

By Bob Japenga

Scope

This white paper is not intended to be the be-all and end-all of the embedded RTOS wars. Rather, it documents the state of where our company (consisting of 9 embedded software engineers) finds itself with respect to choosing an RTOS for the embedded systems we work on.

In this white paper, we will give a brief summary of our experience with various RTOS' over the years (we have been in business for 20 years and some of us cut our teeth on RSX-11M/S over 30 years ago). Then we will provide a few case studies of projects that we have done. Finally we will list what, for us, are the compelling advantages to utilizing embedded Linux in the systems we build.

Summary

Almost all of the embedded systems we build for customers today are networked. Networks require security. And networking is rapidly evolving. We have created networked systems with DOS and other RTOS's but flexibility and security were not hallmarks of these systems. We are convinced that a flexible, open source operating system like Linux will allow our customers to incorporate the latest secure networking solutions better than any of the other options.

Many of our embedded systems have sophisticated user interfaces with graphical displays and, on occasion, touch screens. We have created these on top of other OS's (smx, DOS, Nucleus, OS/2) but strongly believe that the tools and drivers available in Linux make creating such interfaces easier.

We still see the need for the embedded systems without OS's and continue to build embedded systems around many PIC processors. But when networking and/or sophisticated user interfaces are needed, we found, for us, that Linux is the way to go.

Linux and the open source community aren't perfect. But compared to the other options it continues to work for us in the networked embedded systems that we are building.

Embedded RTOS's that we have used at MicroTools

VxWorks (http://www.windriver.com/) – We consulted for a company that attempted to port a control system that we designed using ROM-DOS. The primary reason for the port was to allow for more memory (to break through the DOS 640 K limitation). The biggest challenges were translating code that was

designed for a 16 bit architecture to a 32 bit architecture and restructuring all I/O code to be accessed via drivers. The tools were superb and the reliability of the OS was never a problem. The documentation available was extensive. This implementation never fully went into production primarily because the compelling need for additional memory never manifested itself. Ten years later, 95% of the installations still have our ROM-DOS version.

Integrity (http://www.ghs.com/) – Our first exposure to the Integrity RTOS was on a number of Full Authority Digital Engine Controls (FADEC) used on helicopters. Integrity was chosen because it had already been qualified to FAA standards and did not need to be re-tested. A minimal subset of the RTOS was used. No major problems were uncovered. The biggest disadvantage was the cost per seat and a very restrictive license (we could not access the tool set off-site even if we pulled the software off the machines at the customer's site).

For another customer, we designed a major system around Green Hills' Integrity RTOS that used a wide variety of subsystems. The tools were superb and we found only a few bugs in the extensively tested kernel. Though few, these bugs presented serious problems for us moving forward (one bug forced us to delay implementing a feature until we got to Linux). In addition, its Wear Leveling File System had numerous bugs (it was not robust across power outages and would also go away for minutes when the wear leveling algorithm kicked in). Again without the source code and with the supplier denying a problem, we were stuck creating extensive work-arounds.

Our customer wanted to be able to support Java development for their customers. With the proprietary nature of Integrity, this never was brought to fruition.

Although the documentation was extensive, it often didn't provide the insight into the proprietary software that was needed.

DOS (http://www.datalight.com/products/romdos/) – If you can call DOS a real-time operating system, we have had many years of experience writing real-time systems with DOS. Although memory space and I/O is unprotected in DOS, in 20 years of building real-time controls around it, this never created a safety or integrity problem. We have used ROM-DOS by itself and with smx and others RTOS's on top of it. Do we need to list the strengths and weaknesses of DOS? It doesn't seem to be a real contender any more. In our most recent systems decision, we chose not to go with DOS because of lack of driver and networking protocol support in the future.

Nucleus (http://www.mentor.com/products/embedded_software/nucleus_rtos/) We developed a line of systems around the Nucleus operating system using Cad-UL tools. The tools were adequate and the documentation thorough. But most important, the source code was provided. When the documentation was

not clear and when there were problems (and there will always be problems – even when the code is certified by the FAA), we had the source code to either verify or change.

qnx (http://www.qnx.com/) – For over ten years we supported and designed software around a Unix derivative that was robust and had reasonably advanced tools for the time. The biggest problem we faced with qnx was the lack of support for newer hardware with the older version of software that we were using. With over a thousand systems in the field, each would require a new license fee to upgrade. In addition there were 6-12 man-months of software development to change the interface. Since it was a proprietary system, instead of fixing the source code affected, we were stuck with downgrading the new hardware. Chalk up one significant point for having the source code of the OS.

As an aside – let me say that it is not enough to work with an OS that makes available the source code. You need to have it in hand. On the first Linux project we worked on, after more than 9 months of development we discovered that we had a serious bug in the glibc library. We found that the person who started the project (obviously not from MicroTools!) did not obtain the full sources of all of the libraries. Lesson learned – get all of the source code that is available right up front!

Windows CE (http://msdn.microsoft.com/en-us/embedded/default.aspx) – We have very little experience with Windows CE as an RTOS. We are currently using it on one project where it is the user interface talking to a number of Rabbit controllers.

smx (http://www.smxrtos.com/) – We used smx for over 10 years running a real-time system very successfully. It had a small memory footprint and very low interrupt latency. It was reliable and extensible. Although we didn't find many bugs, since the source code was available and well documented we were able to quickly work around any problems we found.

Case Studies

Company 1 - 2006

A customer of ours who supplies workforce management systems had built a system around GHS/Integrity in 2005. The company sells thousands of these a year. His long range goals included allowing his customers (who programmed these devices in his own proprietary language) to program it in C or in Java. Support for Java on GHS/Integrity just wasn't there. For programming in C, his customers would have to pay upwards of $70,000 to obtain enough seats. So, in 2006 he decided to switch to Linux. With Linux, our customer was able to provide Java as well as C programming and provided his tool chain (because it was open source) at no extra cost.

After the choice was made to go to Linux, a customer with a work force management system written in Python wanted to buy their hardware. With Linux, this was a piece of cake. An open source Python tool chain was found and the customer was off to the races.

As the system expanded, a touch screen was added. Although the exact touch screen controller was not supported in Linux, a very similar one was. In less than a day, we were able to adapt the touch screen driver for the new hardware. We could not have done that with Integrity.

Our customer had 16 MB of RAM and 32 MB of Flash on his system. Our port of Linux easily fit in both RAM and Flash. We even had a backup kernel and backup file system in read-only Flash that was automatically booted in case of file corruption of the main partition.

The following are some of the applications that were included with the version of Linux that we ported:

- telnet
- ftp
- sftp,sc,ssh
- ntp
- vnc
- web server (boa)

Company 2 - 2008

We worked with a company that was using a PC-104 stack on ROM-DOS that had become obsolete. They needed to replace the Single Board Computer (SBC) in the PC-104 stack with another SBC. The unit was designed to send a fax whenever a problem was encountered in the system it was controlling and monitoring. Customers and marketing wanted the unit to send an email rather than a fax whenever a problem was encountered. The company sells about 250 systems a year. The customer wants to sell the same system for the next 10 years.

We looked into replacing the SBC with another ROM-DOS system containing an Ethernet interface and a Send-Mail program. Although they exist, for the most part they are outdated and not currently being supported. We decided that it was too risky to propose this to last ten years from now.

We were able to replace the existing PC-104 SBC with a Linux SBC with Ethernet and save $150 per system.

Company 3 - 2008

A company came to us to propose replacing a proprietary energy monitoring system with a proprietary hardware architecture and operating system. They

were tired of not being able to continuously and simply extend more and more custom features and to easily provide remote updates. They chose Linux and went looking for Linux systems experts who could bail them out. They found us. What took 9 months of proprietary hardware and software, we were able to do in 2 months. In the words of our customer, we were able to heal his pain. I would say that embedded Linux enabled us to help him heal his pain.

Company 4 - 2008

A company came to us to replace their outdated Forth medical instrument. Their marketing team wanted the new device to be able to be securely accessed remotely by doctors using their Blackberrys. Linux made this job simple with its vast supply of open source networking applications and tools with security designed in from the start.

Company 5 - 2008

A heavy industry company came to us because their version of Linux running on one of their industrial controls was not supporting some of the latest protocols, tools and applications. Within two weeks his system was upgraded to the latest version of Linux opening the door to a vast array of tools and applications previously not available to them. Try to do that with Integrity, VxWorks or Windows CE.

Advantages of Using Embedded Linux

Vast array of development options – Not every development tool works for all applications. Try writing a good CGI application with C. With Linux just about every development tool you can imagine is available. (C, C++, Java, Perl, php, Python, even FORTRAN!)

Vast array of applications – One of the major advantages of Linux is the array of open source applications (web servers, ftp, telnet, ntp, ssl, sql, email, and the list goes on). Frankly we want to help our customers concentrate on their specific system needs and not spend time building the infrastructure needed around their applications.

Networking is the name of the game – Linux provides the best networking of all of the possible operating systems. As our systems move deep into the 21st century, IPV6, IP masquerading, network address translation (NAT), and security are going to play an ever more significant role.

New Driver support better than Windows and all others – When it comes to support for the hardware chips that we actually use in embedded systems, Linux is providing these drivers as fast as or faster than they are available in Windows. For example:

- SOC drivers
- I^2C devices

- SPI devices
- Flash File Systems (JFFS2, YAFFS)
- EEPROM
- Custom Embedded Parts
- Networking controllers and PHY's
- Wireless options

Are all available on the latest Linux distributions ahead of the other OS's.

For desktop type drivers Linux is 2nd only to Windows – For drivers for devices available on our desktops, Windows is still the tops. For the following devices, Windows is still better supported:

- WiFi
- Video Chips
- USB devices
- Sound cards
- Capture devices
- Cameras

However, we are finding that Linux is not far behind. More and more manufacturers are releasing devices with both Linux and Windows drivers. But not many are releasing them with VxWorks or Integrity drivers.

All of the Source Code is Provided – Let's just admit it. We are code junkies. We read source code. We're not so compulsive that we have to read everything. But when something doesn't work – we know where to look. And Linux cannot be beaten when it comes to complete openness.

Development is Seamless with a Network File System (NFS) – You can develop all of your code on your PC and the embedded system has full access to all of the files on your PC. This is not helpful in all systems but under certain circumstances it is very helpful.

Has more developers developing ports for their hardware than any other OS - Hardware developers are continuing to port their high and low performance processors to Linux. You can get Linux on:

- Arm 7
- Arm 9
- X86
- Power PC
- ColdFireArm

A micro version of Linux can run on the following processors without memory management:

- Motorola DragonBall (M68EZ328), M68328, M68EN322, ColdFire, QUICC
- ARM7TDMI
- MC68EN302
- Axis ETRAX
- Intel i960
- PRISMA
- Atari 68k

Cost – When we first started down the road of free software, I called it "open sore" software. Although the "sores" are still present, the wallet emptying costs of any comparable OS are also present. And I have changed my mind (no small feat). The total cost is less over the entire development life cycle.

Disadvantages of Using Embedded Linux

Memory footprint – Linux can be utilized best with lots of memory. The last project we bid on was to replicate a previous design with added networking. In the spec to us, the customer asked us to have at least 8 K of RAM and 64 K of flash (that is what the previous design used). Our proposal which added touch screen and networking had 64 MB of RAM and 256 MB of flash! The SBC was cheaper than the board they are currently using. When a 256 MB flash chip costs $4.52 in hundred piece quantities, it is not the driver it once was. However, if power or space limitations require you to go with a single chip solution, a full-blown Linux solution won't work.

Even though we have done full Linux implementation on smaller footprints, it forces you to start making wrong decisions. On a recent project, we built a very simple system with 32 MB RAM and 32 MB of Flash. We were forced to not use Perl in a case where it made the most sense. This cost us significantly in time and development.

Development Tools – We have used NetBeans and Eclipse for development IDE's for Linux. I will say flat out – they do not compare to the quality of the tools for VxWorks, for Integrity, or for Windows! There I said it. The tools are adequate but they have a long way to go. They work with remote debugging over Ethernet which is great – but they are slow and buggy. Please email us if you have a top quality IDE for Linux development. We haven't found it.

Constantly changing interface – As one embedded Linux supplier said: "www.kernel.org Linux is a go-fast technology project, not a finished project." This characterizes our experience. Not everything completed is complete. Things change. Often. We had a problem with a library in 2.16.12 of the kernel that was fixed in 2.6.18. However we could not just migrate the code because the interface to **every** single driver that we wrote was changed between 2.6.12 and 2.6.18! Ouch! Was the interface better? Yes – but… And that is a very big "but."

Debuggers – Sad to say, but the best debugger available is the command line text mode GDB. We have (temporarily we hope) reverted to 1970's style debugging. Making a real-time debugger is a major challenge. We wish that Green Hills would port theirs to Linux. It is superb at debugging multi-threaded applications. We expect this to be remedied in the next few years. We are in this for the long haul.