BSD or Linux: Which Unix is best for embedded applications?

A Wasabi Systems White Paper

Contents:

I. Introduction

With Linux's breakthrough into the server and desktop markets, the OS has emerged as an option for embedded OEMs as well. Several vendors have created their own distributions of Linux intended for the embedded market, and while these have met with only limited success, OEMs are right to look at Unix-like operating systems for their advanced embedded needs. However, as this white paper will discuss, there are several features which make Linux a less than optimal Unix choice for embedded applications.

These features are divided into two general categories: commercial/legal, and technical. On the commercial side, Linux is severely hampered by its license, which requires that all modifications to the Linux kernel (and several other modules of the OS as well) be made open source. While this licensing feature is not important in the desktop or server space, where kernel modifications are rare, it is critical in the embedded space, where modifications to the kernel are part and parcel of OEM's OS investment.

On the technical side, Linux is constrained by its less than robust development tools on the one hand, and its lack of an adequate hardware abstraction layer on the other.

Were Linux the sole Unix-derived OS choice, OEMs might still be justified in selecting it over the increasingly obsolete VxWorks operating system, or the unwieldy Windows offerings. However, concurrent with the expansion of FreeBSD in the server space (the OS was recently chosen by Yahoo! for its Hotjobs site), the

NetBSD operating system has expanded its penetration in the embedded space. NetBSD lacks the user application support of FreeBSD and Linux, so it has not made much of an impact in server or desktop applications. However, such support is not necessary in most embedded applications, and NetBSD offers all of Linux's Unix functionality with none of its licensing or portability encumbrances. Because NetBSD is unlikely to be as widely branded in the consumer space, embedded OEMs may not know of it as familiarly as Linux. However, NetBSD is commercially supported in the embedded market, and may be a more desirable Unix choice for embedded.

II. Commercial Features: The GPL License

The GPL

The GNU General Public License was created by the Free Software Foundation (FSF), the organization headed by Richard Stallman and devoted to the principle that software ought to be freely available.[1] The GPL was, from its inception, intended to be a "copyleft" license: while it allows users to copy, modify, and distribute code it governs, the GPL requires that any derivative work be licensed under the same terms. In other words, any distributed modifications to GPL'd code must themselves be governed by the GPL. This "viral" property of the GPL makes it an exceptionally 'sticky' license – whole libraries of software can become 'contaminated' by the GPL, and proprietary code must be kept entirely separate from GPL'd code in order for it to remain proprietary.

The GPL has achieved what it set out to achieve: it is virtually impossible to make money selling Linux, since any Linux distribution that is sold must also be made open source, enabling users and competitors to obtain it for free. Moreover, since the GPL specifically refers to "work *based on* the [governed] Program,"[2] even new software models – device drivers, some applications, etc. – are automatically covered by the GPL. While the exact scope of the GPL remains uncertain, most experts believe that this provision is much wider than U.S. Copyright law's usual definition of "derivative works," extending to code that merely interacts or links with GPL'd code.[3] As a consequence, a series of increasingly baroque work-arounds have been developed to circumvent these provisions, though none have been tested in court. Moreover, the FSF has stated that it believes they are all invalid,[4] and is aggressively seeking to end some of them (such as the "lighter" external-libraries-only version of the GPL, the LGPL).

The BSD License

[1] While Linux takes its name from Linus Torvalds, Stallman is at least as responsible for the OS's code. Glyn Moody, Rebel Code: Inside Linux and the Open Source Revolution 93 (2001).

[2] Free Software Foundation, GNU General Public License, http://www.gnu.org/copyleft/gpl.html (emphasis added).

[3] Open Software Licenses: Part 2, 5.10 Intellectual Prop. Strategist (July 1999).

[4] Emails by FSF members to Matt Asay, cited in Matt Asay, *A Funny Thing Happened on the Way to the Market: Linux, the General Public License, and a New Model for Software Innovation* (April 2002) (on file with author).

In contrast to the GPL, the BSD License – which governs all the members of the open-source BSD operating system family – places no restrictions on derivative works whatsoever. There is no requirement that modified BSD code be made open source. If a user wishes, she may modify BSD code, keep the entire resulting product proprietary, and attempt to charge licensing fees for its use.

While critics have predicted code fragmentation as a result of the BSD license, it has not come to pass. The four open source BSD projects frequently share code, and instead of fragmentation, there has been increased specialization as the projects focus in different areas of interest.[5] Wind River Systems does offer a proprietary version of BSD, BSD/OS (now in version 4.3), but since the open source communities have more developers than one company, there has not yet been an instance of significant innovation not being shared or duplicated across the open source world. More importantly, the BSD license has allowed projects such as Apache and X11 – as well as Wind River and embedded OEMs -- to modify BSD code and keep the results proprietary, or make them open source, as their business judgment dictates.

Consequences in the Embedded Market

As Linux has begun to threaten Microsoft in the desktop and server space, the software behemoth has spilled much ink decrying the GPL as antithetical to intellectual property, software development, and even American capitalism. This FUD (Fear, Uncertainty, and Doubt) is unfounded. A company running Linux on its servers may still keep proprietary data on those servers. Software companies may write proprietary applications to work atop Linux. And for desktop users, Linux's UNIX-level reliability may hold significant advantages over Microsoft's notoriously crash-prone products. While there may be isolated instances in which GPL'd code must be modified, most ordinary desktop and server users do not modify their OS code.

In the embedded space, however, the GPL matters considerably. Modifications to the kernel are central to many embedded OEM's software development cycle. Porting the kernel to new architectures, device driver development and maintenance, and similar operations are essential tasks in customizing an operating system to work on new or unique hardware. All such software is required by the terms of the GPL to be made open source. Consequently, the GPL may:

1. Cause considerable IP investment to be rendered totally valueless, as software paid for by a customer must be shared with competitors. (Example: A new device driver, which may cost several months and hundreds of thousands of dollars to create, must instantly be shared with competitors as soon as it is distributed.)

2. Expose OEM hardware configuration to competitors, as hardware configuration may easily be 'reverse engineered' from the relevant software code.

[5] Historically, FreeBSD has focused on application support, and is widely used in server applications; NetBSD has focused on wide platform support, and is widely used in embedded; and OpenBSD has focused on security.

3. Create unbounded uncertainty surrounding the legality of application-layer software. With no judicially established limits on the time or scope of the GPL, and with ideologically-motivated free software "zealots" patrolling infringements, companies can never be sure whether their created software is legally theirs.

None of these consequences apply when, as in ordinary desktop or server use, the GPL'd code of Linux and its associated modules is not altered. But in the embedded space, where such alterations are the 'bread and butter' of ordinary business, all three pose significant business risks for potential users.

In sum, the effect of the GPL depends on its application. As a way of capturing and preserving the innovations of hundreds of Linux developers around the world, it has been a success. To desktop and server users of Linux, it is largely irrelevant. For commercial embedded companies, as for any other firms in which kernel modification and driver modifications are important, the GPL can be catastrophic. Of uncertain scope and deliberately-intended durability, the GPL makes it virtually impossible to safely modify Linux for an embedded use and preserve the privacy of such modifications. Particularly with BSD-licensed operating systems commercially supported in the embedded market, embedded OEMs and software companies should proceed with caution.

III. Technical Features

Although the GPL is the clearest single differentiator between Linux and the BSD operating systems, there are several technical features which separate them as well. Because only NetBSD runs on the wide variety of hardware architectures generally required in the embedded space, this section will focus on NetBSD exclusively, though certain points may be applicable to FreeBSD as well.

1. Portability/Hardware Abstraction Layer

NetBSD has had maximum portability as its chief design focus for the last seven years of its open source development. Where FreeBSD has focused on application support, and OpenBSD on security, NetBSD's most distinctive feature is its wide platform support: as of the date of this writing, it runs on a technology-leading fifty one different hardware architectures.

NetBSD's fast portability is due to its unique Modular Portability Layer (MPL). With the MPL, the driver is completely isolated from the hardware platform, I/O instructions or no I/O instructions, interlocking, retry error recovery, bounce buffers, memory type boundaries, scatter/gather maps in host bridges, even peripherals which use pseudo-dma to write a buffer RAM with host CPU copyin and copyout all -- are transparently handled beneath the driver layer. Moreover, several embedded systems using NetBSD have required no additional software development other than toolchain and target rehost.

With Linux, however, device driver code must be reworked for every new architecture. As a consequence, in recent porting efforts by NetBSD and Linux developers, NetBSD has taken as little as 10% of the time to port to new hardware. Engineers ported NetBSD to the SuperH processor core in under six weeks; Linux took three months. NetBSD was ported to the AMD x86-64 in about a month; Linux took six months. As a result, NetBSD supports fifty one supported architectures from the same source tree.

2. Kernel Design/ Development Tools

a. NetBSD's built-in kernel debugger allows comprehensive low-level debugging in real time on the target itself, as well as remotely from a workstation. Linux kernel debugging is only available through third party JTAG tools.

b. The NetBSD kernel provides modular framework for code changes, which facilitates faster, cleaner changes to kernel code than Linux, which does not offer a modular framework for code changes.

c. NetBSD allows for kernel core dumps, which provide a full image of system memory written to disk in the case of a kernel failure. The disk, in turn, can be examined by standard process and kernel manipulation tools. Linux does not offer full kernel core dumps.

d. NetBSD's auto-configuration framework facilitates system and device configuration by simplifying kernel configuration. Under Linux, many devices require explicit hardware information before they can be used.

e. NetBSD's cross-building system (build.sh) allows users to easily target different hardware architectures from a single build machine which need not itself run NetBSD. Linux's cross-development capabilities are more limited; although proprietary embedded Linux companies offer cross-development tools (at significant cost), without such tools cross-building is a complex manual process. (For an example of such a process, see http://www.ltc.com/~brad/mips/mips-cross-toolchain.html.)

3. OS Development Models

NetBSD and Linux are developed according to very different models. NetBSD is developed in a unified way, with the Core Team of the NetBSD Project overseeing additions and modifications to the NetBSD source tree. Adhering to the principle that first is not always best, the NetBSD Core Team tightly controls access to the source tree, and frequently rejects early code contributions in order to wait for better ones. This gatekeeping role is particularly important because NetBSD maintains a unified source tree for all architectures – allowing simultaneous builds on each release -- ensuring high code quality and the most consistent possible environment across platforms.

Linux is, by design, a more 'anarchic' development community. There is no single Linux distribution, but rather, as is well known, a multiplicity of distributions with different featuresets. While the market has coalesced around a select few distributions (RedHat, SuSE, SCO) and developers around a few additional ones (Slackware, Debian), code still diverges among the different distributions. Moreover, since there is no unified source tree for all architectures, Linux for the i386 may be very different from Linux for the x86-64. Each release is built, tested, and distributed separately.

Both communities, naturally, defend their development models. The NetBSD community points to several inferior modules being integrated into Linux (or certain distributions of Linux) simply because they were developed faster than superior pieces of code. For example, Linux had at least two completely independent USB stacks before Linus Torvalds rejected them both and wrote a third one from scratch, after he found both existent Linux stacks unsatisfactory. (When pressed for an explanation as to why he selected the API he did, Torvalds stated: "because I wanted to.") The NetBSD community also claims that the multiplicity of Linux distributions is both confusing and dangerous, because improvements to one distribution may or may not be present in other distributions. Linux developers counter that the multiplicity of distributions makes a wider variety of features available to users, and that there is no single standard of 'best' such as that which the guardians of the NetBSD source tree pretend to impose. Leaders of the Linux community have, further, been able to centralize development around a cadre of their own friends and associates, replicating to some degree the function of NetBSD Core.

The Linux community is almost certainly correct when it comes to popular and widely-supported architectures. There is sufficient intellectual capital in the Linux community to support multiple Linux distributions for i386 and similar platforms, and the market is large enough to ensure code quality. Where Linux falters, however, is in more specialized platforms without such wide market support. For these platforms – whether produced by SuperH, or MIPS licensees, or other vendors – the lack of a single source tree across all architectures, and of unified development, can be critically fatal, because users may be forced to do much of the debugging and other development themselves (or pay consultants to do it for them). For embedded applications requiring or preferring unusual hardware configurations, NetBSD is likely preferable

Finally, the NetBSD development model can often yield better code quality. For example, NetBSD has full integration of kernel and user space code in the source tree. This ensures code changes are debugged in the context of the entire system. With Linux, kernel and user-space code are not tested together until integrated by the distributor. In addition, in NetBSD, regression test-suites are integrated in source tree, which helps isolate unexpected effects when introducing changes. With Linux, no single testing standard exists, so QA depends on the distributor.

4. Maturity of Code Base

NetBSD is the result of twenty five years of open source development. As a consequence, it has a highly mature code base. For example, its networking code ensures better performance and insurance against denial-of-service attacks than Linux's. Linux's TCP/IP suite, while well regarded, is simply less of a known quantity than NetBSD's, which is the reference implementation for Unix. As another example, NetBSD had integrated IPv6 long before Linux. Bugs have had longer to appear, and have been removed for a longer period of time. Although in many applications newer equals better, when dealing with OS code, the more mature a code base is, the more stable and predictable it is likely to be.

5. Memory Management System

The Linux memory management system is designed around the three-level MMU available on Intel x86 processors. For these and similar processors, this works extremely well. However, systems with other MMU designs are forced to suffer the complexity and performance impact of making the underlying hardware appear to function like a three level MMU system. In many cases this requires code to perform specific low level hardware access (for example to flush TLBs) to be scattered throughout the kernel. NetBSD, by comparison, has a cleanly designed pmap abstraction that provides a well-defined interface for the high level routines to perform virtual memory related operations. Each processor's low-level pmap code can then implement the data instructions and algorithms best suited to its MMU.

6. Threading and Scheduling

NetBSD's threading structure optimizes the handling of threads in the kernel, providing performance improvement under thread-intensive conditions. Under Linux, every thread is a high-overhead process, severely impacting performace.

In addition, NetBSD's scheduling algorithms provide prioritized, fast handling of multiple tasks, reducing latency. Linux's scheduling, based on time slices, does not provide equal performance.

7. Application Support

With a high installed user base, Linux offers better application support than NetBSD, though not necessarily FreeBSD or Mac OS X. Most embedded devices do not require a high level of application support, however, so this feature is likely to be less important than development and performance features. Moreover, since NetBSD offers runtime compatibility for binaries compiled for other systems, it allows use of legacy code from Linux, FreeBSD, BSD/OS, Solaris, HP-UX, Digital Unix, SCO, IRIX, and other systems. Linux, in contrast, offers such compatibility for only a small subset of foreign OS binaries.

NetBSD's POSIX-compliant APIs maintain conformity to reference standards, maximizing application portability. Linux is known to have non-conforming APIs and often use platform-specific extensions.

Thus, while application support is likely to be of only peripheral importance to embedded users, even in this area the choice between NetBSD and Linux may be a trade-off depending on the precise functionality required.

IV. Conclusion

There is no one optimal Unix OS choice for all applications. In server and desktop applications, most of the features discussed in this white paper are of only moderate importance, whereas commercial support may be paramount. And while BSD vendors such as Wasabi Systems and Wind River Systems offer a wide range of BSD support (Wind River of BSD/OS and Wasabi of NetBSD), these offerings are primarily targeted at the embedded market. With its plethora of support vendors and application developers, Linux may thus be the better choice for such applications.

In embedded, however, the selection factors are different. The uncertainty of the GPL is a risk factor that must be taken into account by any embedded OEM considering Linux, and while the few remaining embedded Linux companies offer various ways to work around the GPL, none have been tested, and their mere existence testifies to the risks posed by the licensing structure. The GPL alone makes the selection of embedded Linux a questionable choice from a legal and commercial perspective. Were Linux the only Unix available for embedded applications, it might be worth the trade-off. However, given that NetBSD may actually provide better performance – and certainly can more quickly operate on more hardware platforms – there is no need to take such a gamble. NetBSD offers superior Unix functionality on a wider range of platforms than Linux, and yet without any licensing encumbrances. It thus can represent the best of both worlds for embedded OEMs.

Jay Michaelson
David Brownlee