Linux and FOSS: End-to-End (and Top-to-Bottom, Too)

Since its beginnings, Linux has permeated computing systems from enterprise to embedded. While not the only widely used OS, it has potential to achieve end-to-end acceptance with a uni-fied code base and development paradigm.

by Bill Weinberg
LiPS Forum & Linux Pundit

Beginning in the 1990s, Linux and other Free and Open Source Software (FOSS) began an inexorable march from hobbyist-ware into the enterprise and continued out to a range of embedded and ubiquitous computing applications. The penguin's progress started modestly at first, waddling into non-critical utility computing roles, (departmental file and print servers, and intranet servers). Upon proving itself strikingly reliable, Linux then moved into increasingly crucial enterprise application server roles. On the enterprise desktop, Linux displaced legacy UNIX for technical workstation, documentation and data entry terminals.

Free Software actually made its first appearance a decade earlier in embedded applications, with GNU tools gcc and gdb complementing and displacing proprietary compilers and debuggers, followed by the BSD TCP/IP stack, creating a de facto standard for IP networking implementations. Other FOSS components (like BerkeleyDB and Apache httpd) also found their way into larger-scale embedded applications through the mid 1990s. Starting in 1999, Linux began finding its way into a range of edge and access applications. Adoption came from a mix of organic use by developers at TEMs, NEPs and other OEMs familiar with UNIX in management and control plane applications, and from commercial tool kits and services from companies like RedHat / Cygnus, MontaVista and Metrowerks, which is now part of FreeScale.

Today Linux and FOSS experience broad and deep deployment across the entire spectrum of information technology (Figure 1). In the data center, Linux enjoys double-digit market share and rather more modest global desktop deployment in the single digits. Linux and FOSS actually garner an even greater number of embedded applications and constitute the industry's leading platform: Venture Development Corporation reports up to a third of 32- and 64-bit designs were based on the open source OS in 2006.

Progression from Core to Edge, Deployment End-to-End

The incremental progress Linux made in the course of a decade represents a mix of commercial and community investments. On the technical side, key enablers included CPU and board support, architectural advances in scaling, memory access and storage and device drivers. These investments came from across the embedded device ecosystem—semiconductor manufacturers, board vendors, systems suppliers, ISVs and community resources. Business- and tech-savvy semiconductor suppliers like Intel, AMD, FreeScale, Intel and others not only saw the open source OS as an means to "fill sockets" but actually used Linux to bring up their new processors. Board vendors like Advantech, DTI, Kontron Motorola and RadiSys found they could offer richer board support, faster by leveraging community-developed kernels and device drivers. And systems houses like Fujitsu, HP, IBM and NEC saw an opportunity to span and consolidate diverse architectures and product lines while improving margins and expanding services offerings. ISVs saw an opportunity to consolidate and migrate legacy UNIX-hosted products onto a single flexible, interoperable host platform.

As such, Linux quickly accrued broad and deep hardware and software support and today runs on three dozen processor variants, thousands of SBCs and motherboards, across nearly every enterprise vertical and embedded application type (Figure 2).

End-to-End Candidates

Certainly other applications platforms exhibit comparable reach and applicability. Japan's iTRON and µiTRON run on a similarly broad range of CPUs. Sun's Java extends from enter-prise to desktop to embedded, and Microsoft Windows family OSs span a gamut that reaches from the server room to the desktop to in-car and in-hand applications. What makes the GNU/Linux platform different?

The main difference is that the GNU/Linux OS—kernel, libraries and utilities—constitute a single, unified code base. Whether compiled to run on an ARM or an IBM S/390, in an SoC or on a server farm, in an MMU-less microcontroller or a 1000+ CPU supercomputer, the same code implements the same functions, everywhere. The Linux kernel source tree carefully segregates and minimizes architectural idiosyncrasies. CPU-specific code constitutes less than 5% of the total.

Contrast other candidates for end-to-end ubiquity. iTRON, µiTRON (and its stillborn enterprise sibling bTRON) are not OSs—they are *de facto* standard API sets implemented by dozens of different companies with diverse agendas and divergent interpretations of the instruction sets, APIs and protocols. Sun, for pragmatic, application-directed reasons, segmented Java into a range of profiles (J2EE, J2SE, J2ME, mid-p, CLDC, etc.), resulting in fragmentation of class libraries and separate code bases for the major virtual machines (to say nothing about coffee cup clones). Windows operating systems don't pretend to offer continuity with server, desktop and embedded OSs supporting different code bases and API sets.

Open Source vs. Closed Corporate Standards

Standardization is a very good thing. However, standardization and common, community-based implementation, trump standards compliance alone. Andrew Tanenbaum, creator of Minix (on which Linux is loosely based) expressed a key challenge with standardization when he said "The nice thing about standards is that there are so many to choose from." Individual companies producing point products can usually manage to ensure standards compliance for a handful of standards for their products. Most corporate entities, small or large, are in a poor position to comply with, let alone implement, the alphabet soup of standards and protocols, or to build and maintain the tens of millions of lines of code that implement those standards.

Companies boasting the wherewithal to create and implement standards, and presumably compliant products, also have the unfortunate tendency to *improve* the standards they help to define and later implement. They optimize and add value and otherwise ladle on their own secret sauces. Intentionally or not, these enhancements impact interoperability and drive vendor lock-in, in precise opposition to the original goals of open standards regimes.

Open Source looks to standards as a source of requirements to guide implementation and to foster interoperability with other OSs and to support legacy code; GNU/Linux implements (among many others) POSIX, ISO/ANSI C/C++, X11, TCP/IP family protocols, and wireless and wire-line networking. The LAMP stack and Linux desktop applications offer the leading and most compliant implementations of derived protocols like HTTP and

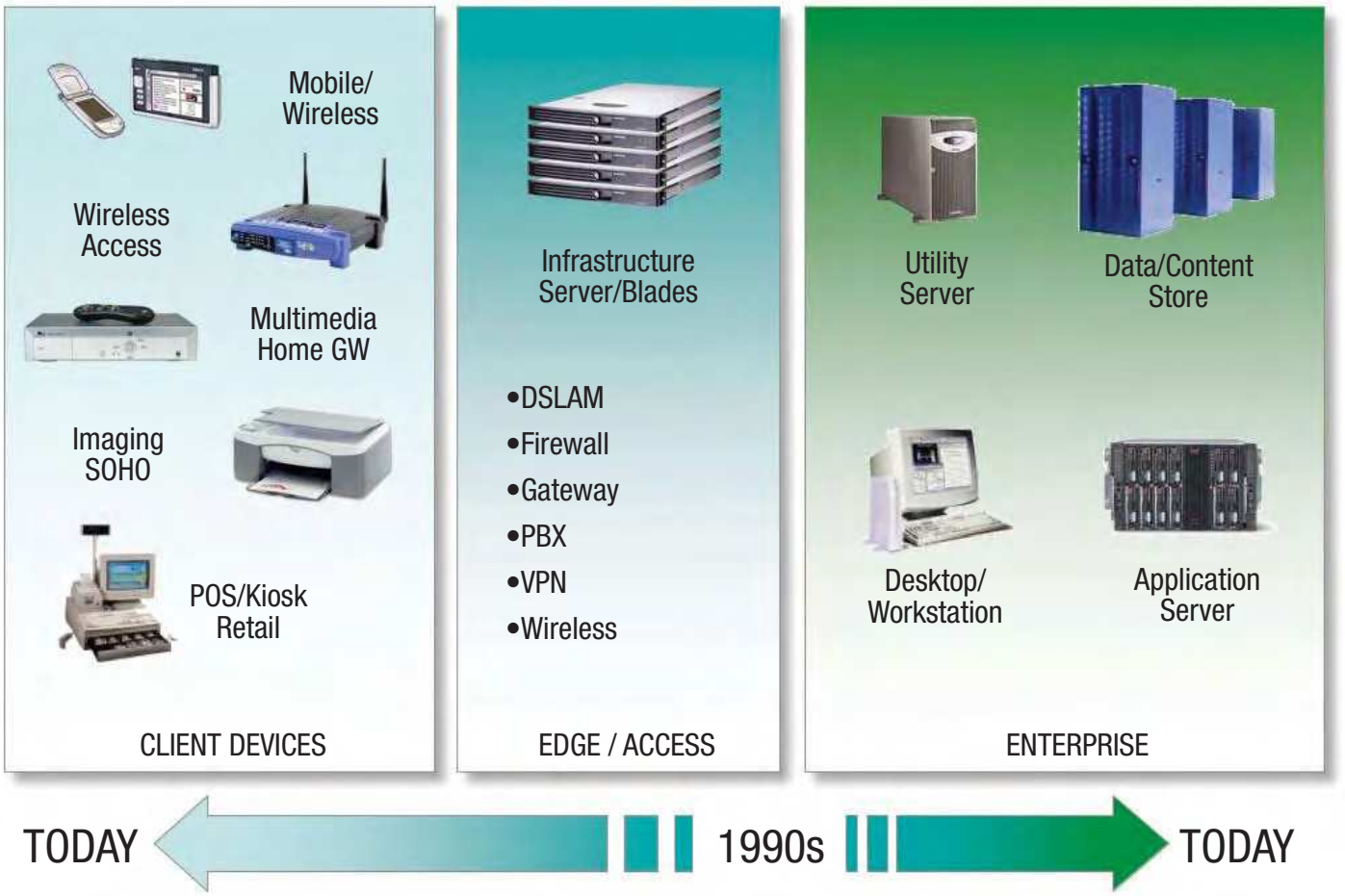

Figure 1 Progression of Linux and FOSS from enterprise/utility computing outward to infrastructure, mobile and other embedded applications.

document formats like HTML, XML, ODF, etc. and myriad other standards and API sets. When code and patches to Linux or other projects omit APIs, re-interpret RFCs or otherwise drift from compliance, a mix of community and corporate interests coalesce to "make things right."

A good example lies in POSIX threads. In the 2.4 kernel timeframe, Linux (and the GNU libraries) supported a *sui generis* threading scheme, and most Linux programs were process-based. As embedded applications for Linux grew in importance, having a pthreads-compliant scheme emerged as a key requirement (e.g., in Carrier Grade Linux). Initially, community figures saw no need for pthreads and lobbied against implementation and integration. In spite of this resistance, IBM offered up Next Generation POSIX Threads (NGPT), a hybrid user-space and kernel implementation. NGPT met with mixed reviews but actually spurred a community effort toward true pthreads APIs and semantics. The result was the development of the highly compliant New POSIX Threads Library (NPTL), which today is the mainline 2.6 Linux threading scheme.

Benefits of a Unified End-to-End Platform

Being able to scale and repurpose a single code base across a continuum of system types and applications yields a range of benefits, some obvious, others less so. In terms of interoperability, the identical implementation of APIs and protocols provides the greatest assurances of interoperability of applications (vs. those based on published standards alone). Complemented by tradi-tional compliance and interoperability testing, developers and users have access to the "best of both worlds"—a standards-based and compliant platform that is also open source, for applications on servers, on the desktop and in embedded applications.

There is also an advantage in unified skill sets. Many organizations run businesses that span horizontally, from enterprise to embedded applications (like telecom carriers and operators, medical services suppliers and governments). Others run vertically integrated businesses (like consumer electronics manufacturers and networking equipment providers). Both types expend huge resources in attempting to level internal technology fragmentation and the training, maintenance and support challenges that fragmentation creates. For decades, these companies have been seeking a strategic end-to-end alternative to a patchwork of legacy platforms.

A more consistent management model is also a boon to organizations. These companies, the eco-systems around them, and the end-users they serve suffer from poor support and quality of service due to disparities in how systems, on and off shared networks, are managed. A single platform with identical system management paradigms and a much smaller range of support issues greatly enhanced organizations ability to provide quality of service at both system and human interaction levels.

Challenges to Linux and FOSS for End-to-End

Linux and FOSS are not a panacea. They constitute a large, dynamic and, some would say messy, code base and technology cloud. A few key areas where Linux and FOSS present challenges to building and maintaining end-to-end applications include the many commercial distributions available. Linux and FOSS are embraced for the freedom of choice and flexibility they offer. Too much choice is not always a good thing, especially when it comes to desktop distributions (Fedora, OpenSUSE, Ubuntu, Xandros et al.), embedded toolkits (MontaVista Linux, Wind River Linux, Open Embedded, etc.), and OEM-derived platforms. Even if the base platform—kernel, libraries, APIs and core functionality—is preserved, differences among distributions, kits and devices can substantially hamper interoperability, especially those dealing with configuration, provisioning and support. At the very least, the multiplicity of Linux editions complicates the life of ISVs, service providers and IT departments trying to deploy applications and services across them.

Linux still lags in terms of application frameworks. The leading proprietary platforms (Windows and Java) offer developers common development environments and application frameworks (even if the platforms do not interoperate as advertised). The Linux desktop boasts two active and fruitful frameworks (GTK and Qt); emerging equivalents also exist for mobile. Open source Eclipse has become the standard for IDEs, but there exists no single widely accepted programming paradigm that can be applied end-to-end. Certainly there exist multiple excellent JVMs, ORBs, rIPCs, databases, web clients/servers, but no single end-to-end capable framework (although mono, the open source answer to .NET, is evolving nicely).

The culture has tended to pay less attention to formal testing regimes. "Many eyes make all bugs shallow," touts open source philosopher Eric Raymond. Indeed, the breadth and depth of the Linux user base exercises, prods and pokes the FOSS code base in ways and means unavailable to most boutique embedded platforms. However, Linux and FOSS have much to learn from the formal testing regimes of proprietary OS suppliers. Today, most formal testing comes from commercial FOSS-based OSVs (Red Hat, MontaVista and others), but centralized community-based testing is catching up, as with the test projects hosted by the Linux Foundation, the home of the Linux Standards Base.

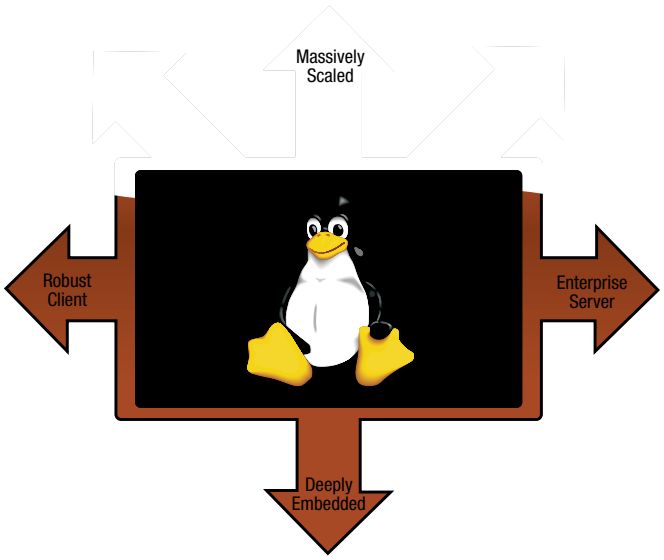

Figure 2 The long reach of Linux scalability.

The intent here has not been to promote Linux and FOSS as a *candidate* platform for end-to-end infrastructure. Rather, it has been to explain why Linux and FOSS are *already* attaining the status of a ubiquitous platform—one that spans the continuum from server to desktop to blades to embedded. Certainly viable alternatives to Linux and FOSS exist at each node; readers need only look to the fragmented embedded OS market for examples of this long tail. End-to-end, a few contenders today pretend to bridge those nodes in a unified fashion, but fall short from a mix of proprietary burdens and fragmented code bases.

This momentum enjoyed by Linux belies the well-known shortcomings of FOSS. Indeed, in many cases, Linux and FOSS are not deployed *because* of their attributes, but *in spite* of them. However, only Linux and FOSS have accrued the unity, critical mass and evolutionary velocity to qualify for this strategic platform role. ◢

Linux Phone Standards (LiPS) Forum
[www.lipsforum.org].

Linux Pundit
[www.linuxpundit.com].