EL/IX: Unifying APIs for Linux and Post-PC Computing

Michael Tiemann – 21 September 1999

Why Embedded Linux?

As Linux continues to gain strength in the traditional PC arena, a number of people are evaluating Linux for use in the revolution beyond the PC desktop: the Post PC revolution. Whereas the PC era created a vast homogenous platform for application development and deployment, the Post PC revolution is creating numerous vertical application opportunities ranging from consumer electronics to digital imaging to office automation to internet appliances to automotive control and beyond. The Post PC era will mean more microprocessors being more pervasive and doing more things than would be possible in a PC-oriented world. This in turn means that new solutions are needed for application development and application deployment.

Intelligent design, good engineering, a large development community, and the GNU software development tools have made it possible to port Linux from its roots as a PC operating system to many of the major processors that are prevalent today, including the Alpha, ARM, IA-64, MIPS, PA-RISC, PowerPC, and SPARC. This portability, a rich set of internet capabilities and the GNU tools (which are used by the majority of all embedded system developers) make Linux appear to be a natural for the embedded systems market, and indeed there are products reaching the market today with Linux inside. But as new companies come to market with their own versions of embedded Linux, the ever-present danger of fragmentation arises.

Unlike proprietary operating systems, Linux is Open Source, and anybody can read, modify, or share it as they wish. Believers claim that this "democratization of technology" is a fundamentally superior approach to centralized technology management, while skeptics fear that if anybody can make a change, the technology will devolve in the ensuing anarchy. In the desktop space, democracy has reigned supreme: the community accepts that "the best code wins", which means it also accepts that lesser code does lose. This model of natural selection has resulted in unprecedented advances in functionality, quality, compatibility, and rate of innovation.

But the deeper one delves into the embedded market, the more one finds that the democratic process that favors superior code gives way to the dictatorial reality of hardware, schedule, power, and cost requirements that are unique to every single application. In this context, code that would be best for one platform is totally out of the question for another, and few embedded developers (even those creating embedded Linux distributions) have the time or skill set necessary to resolve this dilemma. Moreover, Linux is the not the right operating system for *every* job. For deeply embedded applications where every kilobyte counts, the megabyte-sized Linux configurations are not appropriate for the job.

It would be a shame to lose the benefits of Linux as a common platform for software development, but it would also be a shame for Linux to fail to fully connect with the Post PC revolution. An approach is needed that solves the problem in a way that is both Linux-compatible and Linux-independent. This need has led Cygnus Solutions to specify EL/IX, an Embedded Linux API based on POSIX, which is suitable for other embedded operating systems as well. With EL/IX, it will be possible to use desktop Linux systems for application development, and EL/IX to ensure that these applications can be recompiled and deployed across the complete spectrum of Post PC devices.

How should standards be defined for Embedded Linux?

The goal of EL/IX is, first and foremost, to standardize the APIs available in desktop Linux today in such a way that embedded system developers have a clear and consistent idea of what functionality they can expect for a given embedded target:

1. support for developing embedded applications using the Linux desktop environment as both a host and development platform
2. a way to scale that functionality according to the requirements of their embedded target
3. the freedom to use any kernel (Linux or otherwise) that implements the interfaces of EL/IX that their application requires
4. the ability to easily port to any hardware that runs any kernel that supports the subsets of EL/IX they require

Moreover, the first version of EL/IX will adhere to these additional requirements:

5. elements must already be implemented in standard Linux distributions (i.e., don't define what doesn't exist)
6. possible to demonstrate real real-time performance when running on a real real-time OS (prove it works beyond Linux)
7. based on POSIX where possible (build on accepted standards)
8. does not include the kitchen sink (keep it simple)
9. EL/IX will be extensible
10. EL/IX will be Open Source

By meeting the technical requirements of the embedded developer, using internationally accepted standards, with an open-source implementation, Cygnus intends to make the adoption of EL/IX a "no-brainer". Indeed, the impact on today's Linux developer is virtually nil (though in the future it will require the maintainers of desktop Linux to incorporate the implications of EL/IX into their plans). But the impact of adopting EL/IX in the embedded community will be profound: flexibility and freedom will prevail over fragmentation. Moreover, it will provide a normalizing force that will make it easier for software developers to build content that is useful for both desktop and embedded applications, thereby expanding the market opportunity for software without increasing fragmentation.

What should the standard for Embedded Linux look like?

EL/IX is derived from ISO 9945-1 (aka POSIX.1 or IEEE 1003.1). For now, the set of functionality defined by EL/IX will be small (as small as we can make it), but as this open source initiative gains contributors, it could conceivably extend up to and beyond a complete POSIX implementation and into the realm of arbitrary Linux libraries, including graphics, multimedia, and others. We are starting small because we believe that the right approach to the problem is to make enable the API to support configurable implementations, and thus configurability must be considered at the lowest level of the design.

EL/IX will first be implemented as a configurable compatibility layer on eCos, an open source RTOS, much in the same way as Cygnus already provides an optional µITRON compatibility layer for eCos. EL/IX will not include functionality that is not already provided by the standard Linux kernel and glibc -- we do not intend at this time to improve or provide additional functionality for Linux. Indeed, if such functionality were to result in a non-standard version of Linux, our whole effort would be deemed a failure.

The implementation of EL/IX must of course be compatible with the existing Linux implementation of the same functionality. We want to make EL/IX pretty lightweight; it is not our intention to re-implement Linux on eCos, just the minimal parts that make sense in an embedded real-time context.

The essence of the initial implementation is to incorporate a subset of POSIX 1003.1, and incorporate the complete ISO C library, math library, and optional BSD socket interface (POSIX 1003.1g). Later on, we (and hopefully others!) can extend EL/IX to incorporate other interesting parts of POSIX and Linux and additional libraries such as OpenGL, etc.

What's not included in the initial implementation? Everything else! In particular we will not be supporting anything other than a ***single process*** (though multiple threads will be supported…see below). So all of the following is specifically excluded from EL/IX: process support (`fork` and the `exec` and `wait` family of functions), pipes, and fifos. Also, there will be no process group, user or system ID support.

This is intended to be an application level standard and we will not attempt to provide any device driver compatibility, no `/dev`, etc. We will also avoid all terminal I/O support (see the I/O discussion below). Support for file locking is also not sensible in the single process context, so it will not be supported.

POSIX 1003.1

The latest ISO/IEC POSIX 1003.1 standard no longer refers to the old A, B, and C sections, but they are useful to refer to in this discussion. Section A is the basic Unix API, B is realtime extensions, and C POSIX threads.

The initial implementation will support only the following elements of the POSIX standard:

Base Unix API (Section A)

Most file and directory operations:

- `read, write, open, close, lseek, fnctl, mkdir, rmdir, chdir, link, unlink, opendir, readdir, fsync`.
- no `dup, chown,` or `ftruncate`
- debate `stat` and `chmod`

Time operations:

- `time, localtime`.
- No `timezone` support (initially; this is no big loss since the current standardized interface the time handling functions implement is not sophisticated enough for most implementations anyway.)
- (Not in POSIX, but include `gettimeofday`.)

Signals will be included in the standard as an optional element.

The initial implementation may well not include support for signals.

`_exit, abort, exit` will have slightly modified characteristics from the POSIX standard depending under which environment they are executed in (configurable hard stop, system restart, etc. under eCos).

Realtime extensions (Section B)

- Async I/O
- Semaphores (only for semaphores used in a single process)
- Reliable signals (meaning queueable real-time signals, subject to the limit of the maximum number of outstanding real-time signals)
- Message passing (glibc will be enhanced to support `mqueues`, which are missing from the current version of glibc)
- Time related functions: `nanosleep`.
- Other timer-related functions (`timer_gettime, clock_gettime,` etc.) may be included via patches that exist to the Linux kernel today.

We will support `SCHED_RR` and `SCHED_FIFO` (disable timeslice) scheduling policies. The default scheduler will be the eCos MLQ scheduler. We could consider implementing a Linux compatible scheduler for `SCHED_OTHER`, or as a configuration option, but there doesn't seem to be a valid reason to do this in the embedded context. Other schedulers will be configurable as per std eCos functionality - but this is not strictly part of the EL/IX standard as this feature is not available under Linux. Many eCos scheduler implementations should however be compatible with the eCos pthreads implementation.

`mmap` and shared memory are not supported - amongst other reasons they would either require filesystem and MMU support as standard, or are not sensible in the context of a single process. We may elect to support `mlock/munlock` - as NOPs, and potentially add `mprotect` in the future (but would require some basic MMU support being added to the eCos hardware abstraction layer (HAL)).

POSIX threads (Section C)

All of the pthreads functionality is included with the possible exception of `pthread_kill` and `pthread_cancel`. Thread cancellation and signals are optional (configurable) elements of EL/IX, and will probably not be included in the initial implementation.

Per-thread data requirements can be huge. The POSIX standard defines a minimum of 128 keys to be available, which may consume 512 bytes per thread in some implementations. We will provide a `STRICT_POSIX` configuration option that provides this minimum, but the standard default will be 0 - but configurable. We will also follow this type of philosophy for maximum open file limits, etc.

ISO C Library

The full ISO C Library is included with the following additions and modifications:

- `fileno`.
- All the _r reentrant functions (from Section 8 of 1003.1), namely `strtok_r, rand_r, asctime_r, ctime_r, gmtime_r, and localtime_r`.
- Only the C locale will be supported (initially; future versions will be able to support additional locales through configuration options).
- Multiple timezone support will not be included.
- Debatable, but probably include environment, but will not include any variables as standard. Include `putenv`.
- Maths library - C90 for now with the C9x extension of `long double`. Other C9x may be supported in the future

BSD socket library

Now in draft as POSIX 1003.1g, excluding STREAMS (which are now optional). We will probably include most of it, but make DNS operations such as `gethostbyname` optional (most likely to be provided by vendors of TCP/IP stacks that can be configured into non-Linux operating systems).

Not part of EL/IX as such, but infrastructure will be provided to enable third party vendors to plug their stacks and file systems into eCos. This will include support for the `select` functions, `read`, `write`, etc. on sockets and files.

I/O

EL/IX includes BSD sockets (as part of 1003.1g), which should take care of many I/O requirements. We also provide POSIX 1003.1 file I/O facilities which can operate on network drives, flash file systems, RAM and ROM disks, and even real hard disks. But beyond these devices, we really want to support some kind of abstraction layer that can be used to describe a wide range of custom hardware devices that will enable embedded applications to be ported across embedded Linux, eCos, and other operating systems that support EL/IX. For now, the specification of that abstraction layer remains to be defined. In particular, the `/dev` part of the file system hierarchy is currently excluded from the EL/IX proposal. Since I/O is so fundamental to most uses of embedded systems, and since there is so much variety that must be supported, some sort of abstraction layers need to be defined, which is a subject for the next section.

Where will EL/IX go from here?

EL/IX is based on the internationally accepted POSIX standard that is already fundamentally supported in Linux through glibc. EL/IX will grow as components of that POSIX interface are implemented in such a way that they can be easily configured in or out of a given embedded Linux implementation. EL/IX will also grow as other components of Linux are enhanced to support the configurability and scalability that EL/IX enables, and it will also cover a growing number of development platforms.

EL/IX will need to be tested, both in implementation and in concept. The short-term test will be the ability to build simple applications, such as a web server with dynamic page generation, that can use EL/IX to run on desktop and embedded Linux as well as eCos. The medium-term test will be the ability to build complex applications, such as GNOME and some of its utilities, that can use EL/IX to run on an embedded application such as a point-of-sale kiosk. The long-term test will be the ability to use a standard set of configuration tools to unpack any Linux application and port it without modifications to any system that supports the EL/IX APIs that it uses.

EL/IX will also need to take some sort of position on rationalizing the I/O problem. It is too soon to present a solution to this problem, but the scope is at least clear: for each kind of I/O device that we want to support, from an X-terminal to an LED, and from a PCI bus to a serial port, the use of the interfaces should be consistent not just for Linux and embedded Linux, but for deeply embedded systems as well. This means being able to configure I/O devices without necessarily configuring in a whole mess of functionality that is really only needed to support an abstract I/O interface. Similarly, in keeping with the philosophy of a unified development platform, the choices made here should also extend not only to real hardware devices, but to virtual hardware components as well, and the mechanisms to support these devices used should integrate not only with Linux but again also with deeply embedded operating systems as well. The eCos development team has some experience in both these areas which is likely to be valuable in defining an I/O annex for EL/IX.

It is important to be clear that EL/IX is not a Linux distribution. EL/IX is an API that is operating system neutral. We believe that due to the open source nature of both Linux and EL/IX, that EL/IX should be available in a current version of desktop Linux, but not all implementations will be supported by all embedded operating systems (Linux or otherwise). The value of EL/IX is that developers can create applications on the desktop and then use EL/IX to determine where and/or how to migrate them to which embedded platforms. Thus, even while hardware becomes more diverse, it will always be easy to know what software can be ported to what hardware with a simple recompile.

It is also important to be clear that EL/IX is not going to make Linux suitable for deeply embedded targets, nor will it enable very lightweight operating systems to replace Linux as a desktop operating system. In other words, EL/IX will not give operating systems properties or functionality they don't already have, but will instead give developers the ability to target a broader range of application platforms than they can in today's highly fragmented embedded systems market.

Who's behind EL/IX?

Cygnus Solutions is creating the first implementation of EL/IX, but we are not the only ones behind it. Developers from the Free Software Foundation and the GNOME project and industry analysts agree that EL/IX provides the necessary continuity between desktop and embedded Linux to ensure that the Linux market does not fragment as it connects with the Post PC revolution.

Cygnus is uniquely qualified to maintain EL/IX. Cygnus engineers were behind the original GNU `configure` script, the first script used to configure all GNU software in a consistent fashion. Cygnus engineers also help to create, and currently maintain `autoconf`, a next-generation generator of `configure` scripts. Cygnus pioneered the concept of source-level configurabilty for real-time kernels with eCos, implementing a graphical user interface that made it easy for users to use the power of this open source operating system without needing to fully understand the source-level details. And Cygnus employs a majority of the developers that maintain the GNU software development tools, tools that are fundamental to both Linux and embedded systems development alike.

When will EL/IX be available?

As with any new Open Source project, the answer is "Good Question!". We will post updated API drafts, schedule information, and other information to http://sourceware.cygnus.com/elix . By visiting this site and registering your interest (in using and/or contributing to EL/IX), we can keep you up to date on the latest developments.