Reprinted from the

Proceedings of the Linux Symposium

July 23th–26th, 2003
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Review Committee

Alan Cox, *Red Hat, Inc.*
Andi Kleen, *SuSE, GmbH*
Matthew Wilcox, *Hewlett-Packard*
Gerrit Huizenga, *IBM*
Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*
Martin K. Petersen, *Wild Open Source, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*

Developing Mobile Devices based on Linux

Tim Riker
Texas Instruments
Tim@Rikers.org, http://Rikers.org/

Abstract

This presentation will cover available components of embedded solutions that leverage Linux. We discuss bootloaders, Linux kernel configuration, BusyBox, glibc, uClibc, GUI choices such as Qtopia, TinyX, GPE, Konqueror, Dillo, and similar packages that make up a commercial-grade consumer device. This presentation is aimed at those who are just getting into Linux on mobile or other embedded devices.

1 Why Linux?

Linux is a stable, tested platform for production use. The most often-used environment for Linux is as a server for key business services. Is it well suited for embedded use on mobile devices? It is. Linux has a low total cost of ownership as compared to other options. There is a large pool of developers that are familiar with the environment and this pool continues to grow rapidly. The use of an Open Source platform allows for flexible hardware and other product design choices that can save money. There is no dependency on a single vendor for support.

The largest advantage is quick time to market. The wealth of available projects that can be leveraged means that resources can be directed toward the specific value a product has to offer without spending undue resources duplicating what is done on devices that are otherwise similar. This advantage truly comes to light when a community is formed around the product. This community can focus on enhancing the product without direct development cost to the manufacturer. When planning the product lifetime, thought should be given to seeding hardware to key community members so that community support is available early in the product life cycle.

2 Special Needs of Mobile Devices

Mobile devices have features not often seen in desktop or server systems. Most use flash storage instead of traditional hard disk media. NOR flash is a common choice for average-sized storage, but becomes more expensive with larger systems. It is relatively easy to handle from a software perspective, and is commonly available as a direct memory mapped device. NAND flash is cheaper for larger (ie: >32MB) devices, but is not normally mapped to a specific memory location. In addition, flash devices track bad sectors and have error-correction code. Special device drivers and filesystems are required.

Removable storage is also an option. MMC (MultiMediaCard) is one common small-form-factor storage card. These use a 1-bit width serial interface to access the flash. SD (Secure Digital) storage cards can have an enhanced 4-bit interface that allows for faster data transfer and I/O devices, but the licensing from http://SDCard.org/ prohibits releasing

the source to any driver for these cards, so they are not currently recommended for Linux-based solutions.

Compact Flash (CF) storage cards are another popular option. Testing has shown that most, if not all, CF cards are unreliable when power-cycled during a write. Consider this strongly if power-cycles are likely to happen. Batteries often run down on mobile devices.

2.1 Power needs in hardware

Power consumption is critical in a mobile device. If the device has a display, the light is often the single largest power-consuming component. Software should be configured to be aggressive about turning off the lighting. This is commonly user-configurable, and often has a different setting when the device is connected to external power. Wireless interfaces are likely the next largest power consumer. It is wise to spend time during product development tuning the wireless setting for maximum power conservation. Choosing a different wireless chipset can make a large difference in the power needs of the device.

Linux has support for CPU scaling on a number of architectures. Research the devices that are affected when the CPU speeds up or down on different platforms. For example, on typical StrongARM platforms, the LCD display must be turned off during any CPU speed changes. This may mean your product cannot leverage CPU scaling in a useful manner. Other devices that may be affected by CPU speed changes include audio, USB, serial, network, and many other timing-sensitive devices.

CompactFlash and other removable devices may still consume significant power while in a suspended state. It is wise to add support for removing power in software to most system devices before entering a suspend state. It is also wise to avoid polling of any hardware device. As a general rule, interrupt-driven devices will have a lower load on the CPU and therefore consume less power. Physical keys on the device are one common area where this is overlooked. If the device has a power button, it should be on a separate hardware interrupt from the other keys on the device.

3 The bootloader

Choosing a bootloader has a big impact on the development environment. Most embedded systems do not have a traditional BIOS onboard. They do not have APM or ACPI interfaces. Some rely on the bootloader to take part in power-resume states; others just resume execution at the next address after where they entered suspend. Most bootloaders will initialize RAM configuration, and startup other devices in the system. Hardware designs may want to insure that the Linux kernel can be booted with a minimum of hardware setup so that there does not need to be driver code for the remaining hardware in the bootloader as well as in the kernel.

The most common interface to a bootloader is over a serial port. If the device has a keyboard and display, that may be another choice, but there is usually serial port support as well. The bootloader should support flashing new code on the device. New kernel images and filesystem images are loaded over this interface and stored to flash on the device. This requires that the bootloader is able to deal with whatever styles of flash are on the device. Xmodem, Ymodem, and even Zmodem code is available in existing open bootloaders.

If the device will have removable media such as CompactFlash storage cards, the bootloader can be configured to read images from there for flashing. This may be a good option for

upgrading devices in the field. Installing from MMC or other types of storage devices may require significant work implementing device support in the bootloader.

Some embedded systems have built-in ethernet interfaces and can use that for bootp or tftp updates. This is not very common for a mobile device.

For products with a USB client connection, USB serial support can be implemented in the bootloader. This will likely use a different USB device ID than presented by the product in its normal running state. If the product also has USB host connectivity, then it can be imaged off another working device. The kernel and base filesystem should be in a separate area for the device configuration in order to support this.

The bootloader will need to be aware of any partitioning in flash on the device. The kernel will need access to this same information. This information could then be shared with the kernel by dedicating an area of flash to store it. It could be compiled into both the bootloader and the kernel. The cleanest approach is for the bootloader to add it to the kernel command line when booting. There will likely be other kernel command-line options the bootloader will want to store and pass on to the kernel. All of these can be contained in one flash block if desired. Some bootloaders understand the filesystem, as well as the partitions on the device. This allows for one filesystem image that contains the base binaries and libraries as well as the kernel and its modules. This prevents the kernel and modules from ever being out of sync if they are all in the same image, which is a nice feature from a customer support perspective. If the bootloader can read files from the filesystem directly, then one can store the permanent kernel command-line options and other bootloader configuration inside the filesystem. This will likely not include flash partitioning information, as the partitioning would need to be known before the filesystem could be read.

Erasing the bootloader is a Bad Thing. Steps should be taken to insure that it it protected. This might mean putting it in ROM, or protecting the flash block(s) it lives in, or other methods. OS updates in the field should not replace the bootloader as part of the normal procedure.

4 Filesystems

Workstation and server filesystems like ext2 do not scale well for embedded devices. Smaller filesystems include `romfs` and `cramfs`, which are read-only. `cramfs` includes compression. Flash or ROM often is slow to access. The time it takes to read and decompress files is often faster than reading the same file if stored uncompressed. `cramfs` may be both the smallest and fastest choice for a read-only filesystem. In most cases it is also important to conserve memory. This implies that filesystems should be used directly from flash rather than using an initial ramdisk (initrd) whenever possible.

Some applications on the device will create temporary files. If the filesystem is read-only, ramfs support should be added to store these files. One way to handle this is to link /tmp to /var/tmp then mount ramfs on /var and unpack a tar archive into it, or just copy a tree from someplace else on the filesystem. You may want /dev linked in here as well if device files need to have permission or ownership changed at runtime.

An alternative to /dev is to include devfs support in the kernel. You may be able to avoid using devfsd by considering the needs of all the applications that will be included on the device and configuring default devices permissions in the kernel.

Many handheld portable devices use jffs2. This is a compressed journaled filesystem that understands NOR flash directly. It will need a few free blocks to use as workspace in each mounted partition. jffs2 has undergone a great deal of testing to insure that it is always in a readable state even when writes are interrupted by a powerfail or hard reboot. jffs2 support on NAND flash chips is in progress and may be complete by the time you read this. There are other flash filesystems like YAFFS designed just for NAND flash devices. This is a reasonable option with a lot of flash and where compression at the filesystem level is not needed. Removable storage media will still need to use vfat if it is going to be moved to other devices.

Some systems have dedicated partitions in flash for diagnostics or additional code that handles reflashing the device in the case where the normal filesystem is not usable. The impact of requiring this extra flash space should be considered carefully. Reflashing code can be added to the bootloader in much less space. User-space tools can be used for this task as long as the device can get to that point using the existing filesystem. If the device includes removable media access, diagnostics can be shipped on the removable media and the bootloader configured to load a kernel and filesystem from there.

5 Kernel device drivers

Embedded devices often do not include components like PCI, ISA, MCA busses. All devices that will not be present should be turned off in the kernel config to save space. These are not always obvious choices. For example, if the device includes a CompactFlash slot, all of the PCMCIA card drivers would be available as card options. There is no clear list of which drivers are exclusively for pcmcia cards and do not need to be available as they would not be inserted in a CompactFlash slot.

5.1 Connectivity

Many mobile Linux devices include some form of networking. This is probably not a normal ethernet interface but can include WiFi, Bluetooth, IrDA, USB, ppp over serial, cell phone, etc. Some devices will have multiple options available. Very good IPv6 support is available under Linux, but be aware that the IPv6 stack is larger than IPv4 or other options.

6 Libraries

Larger Linux systems are commonly based on the GNU C Library. This is not well suited to small devices. To quote the maintainer:

> . . . glibc is not the right thing for [an embedded OS]. It is designed as a native library (as opposed to embedded). Many functions (e.g., printf) contain functionality which is not wanted in embedded systems.
>
> —Ulrich Drepper
> <drepper@cygnus.com>
> 24 May 1999

Other alternatives will be smaller, but may require some modifications to source that is being ported. This should not be a large task when compared with the other aspects of the port. Alternatives include uClibc, dietlibc, newlib, and using pieces from things such as the Minix C library, libc5, *BSD libraries, etc. uClibc is the best choice today, as it is under the LGPL license to allow for commercial applications, includes shared library support, pthreads, c++, and even most locale features. It is configurable so you can remove things

you don't need. uClibc is tested for compliance with POSIX and ISO standards and passes more tests than glibc.

Other libraries included on the device should be reviewed closely to insure that they do not include components that are not needed. Many of these have compile-time options to exclude large portions of the functionality.

If no applications are going to be added to the system at a later date, library reduction tools can be used to remove the unneeded portions. Some examples of these tools include mklibs, lipo, and libraryopt. The python script `mklibs.py` is used in the Debian project on multiple architectures and is the best place to start.

7 Applications

There are many basic tools that run on top of Linux and make up a basic distribution. Many of the GNU tools could be here. GNU fileutils, textutils, grep, modutils, and many others are found in most every Linux distribution, hence the GNU/Linux naming convention. While this is the norm for a desktop distribution, it is likely that none of the binaries or libraries on an embedded Linux system will be the GNU flavor. Applications like BusyBox and TinyLogin are not GNU packages, nor is uClibc.

BusyBox includes close to 200 different applications in one multicall binary that is under 600k. This allows you to hardlink or symlink to the single binary, and depending on how it is called, it acts as each of those different applications. Each applet is normally less feature-rich than the GNU equivalent, but much smaller in size. If you were to include the normal binaries for all of these from a GNU/Linux distribution, you could have over 5 MB of binaries. Each applet in BusyBox can be enabled or disabled so it's likely that your version will be much smaller when you only include the programs you need.

Your system will likely want to have some services and perhaps a way to get shell access remotely. A small inetd (24k) with a minimal telnetd (32k) could do the trick. The latest version of BusyBox includes both of these. Expansion capabilities might warrant including pcmcia-cs and hotplug code. Use caution in this area as well. Linux supports a lot of PCMCIA hardware but the size of included drivers can add up quickly.

8 Crypto

For secure access OpenSSH is the common choice on Linux desktops and servers. OpenSSH client (215k) and server (254k) normally uses OpenSSL (181k). When OpenSSL is compiled for a smaller set of supported hashes and algorithms, it can be smaller, but even then it is large as compared to most embedded applications. The author is still searching for a small ssh/sshd solution.

FreeSWAN is another secure access method. It is normally even larger, close to 2M in size. FreeSWAN supports Opportunistic Encryption which can be very useful in an enterprise environment. Normal IPSec support is also valuable for many solutions. Adding hardware crypto support as provided by the Texas Instruments OMAP161x chips and converting all crypto systems in the kernel and userland over to use this can save memory as well as boost performance.

9 Package Management

For devices that are expandable, some form of package management is needed. Many desktop systems use RPM or dpkg for this purpose. Both of these are large binaries and more im-

portantly they normally have a large amount of package information stored on the filesystem. BusyBox includes a stripped-down version of both rpm and dpkg which might be a good starting place. The Handhelds.org project has a package manager called ipkg that is small and better suited for embedded systems. The Debian installer uses .udebs for about the same purpose. ipkg packages and udeb's do not normally include man pages or other supporting files, but have just the minimum files included. These packages are both the same format as a .deb file.

Note that Linux Standards Base requires the ability to install rpm files. It also requires many things like libgcc_s and glibc that the embedded device may not provide.

10 Graphical User Interface

There are many choices of GUIs for a mobile device. The Sharp Zaurus models use Qtopia with Qt/Embedded. These are available under the GPL, but using these versions requires that all applications available for the platform are also under the GPL. These components are also available from Trolltech under a different license as royalty-bearing components. This allows applications to use licenses other than the GPL. These components might be the only ones on the device that are not free. The base files needed for a common Qtopia and Qt/E configuration would be about 3 MB in size. This does not include the applications.

The Handhelds.org distribution can use X11 and GTK as the basis for an environment called GPE (GPE Palmtop Environment). This uses TinyX from the XFree86 version of the X Window System. The X server itself will be around 700k and the X11 and GTK libraries add up to about 3 MB. Note that this system retains all the remote display options of a normal X Window System desktop machine.

Smaller systems are available such as Pixel or NanoGUI. These might not include advanced web browsers or other large applications, but they are appropriate for smaller devices. A custom interface could be built on top of systems like DirectFB, GTKfb, Qt/E, SDL, or by writing directly to the Linux framebuffer.

The Qtopia interface has gotten more attention lately and is currently ahead of the GPE work. The OPIE project has enhanced Qtopia and is starting work on a next-generation library that would replace the Qtopia library and clean up the interface in the process. This new project should be available under LGPL, and if it was configured to run on top of TinyX using Qt/X11 instead of Qt/E, could avoid a per-unit royalty. Removing duplicate code between X and Qt/X11 could drop the storage size down very close to that of Qt/E and Qtopia. This retains the remote display options that the X Window System provides, while leaving a path to use much of the available Qt widgets and applications.

11 Web Browser

As time progresses, an embedded web browser will become more and more important. Browsers like Netscape and Mozilla are large. Other browsers might use the same engine but be much more space-constrained. Browsers like Dillo and ViewML are less feature-rich, but even smaller. Text-based browsers like lynx or links could be wrapped with a graphical front end, but this would be time-consuming and not in line with the goals of those projects. There are commercial options like Opera to consider as well. Konqueror Embedded, which uses Qt, is a good mix of size and features. Some minor interface tweaks could make it even better.

12 A Small Example

The TuxScreen project is very tightly storage-constrained. There is no web browser in the Linux-based base filesystem image. This device is a Desktop phone with a StrongARM 1100, a 640x480x8 color touch screen, and only 4 MB of flash storage. In this space we have a 128k bootloader partition (with only 32k used) and a 4MB minus 128k jffs2 root partition. In the root partition is the kernel and modules, uClibc, BusyBox, TinyLogin, pcmcia-cs, lrzsz, inetd, telnetd, XFree86 TinyX, rxvt, matchbox (a window manager), and more, with over 500k left in writable space. For devices that are only remote displays, this is a reasonable target size.

13 Where to Go from Here?

There are many Linux consulting companies that can help with a new product release. Monta Vista and Metroworks are two of the larger players in that space. Red Hat also has a group focused on embedded work. Some of these solutions might include other royalty-bearing components or have the option to spread out the engineering cost over the product lifecycle. There are plenty of individual consultants active in embedded systems work based on Linux. Some are looking for permanent placement and some work on a consulting basis. You should determine the finances of your project and pick a solution that matches with that plan.

The big benefits of going with Linux as the solution is this large available pool of resources to draw from and the ability to have all the source so you retain the option of doing the work internally or through another vendor. Do not ignore the benefit of growing a community around your product. The work that active, qualified developers in the community do to improve your software solution is a valuable benefit. Much of it will be available to you for merely the time it costs to monitor those activities. You can effectively kill off this effort by making it difficult for others to get involved. This will likely have a detrimental effect on your product sales.

When working out the details of contract work, you will likely want to add a requirement that all work be pushed upstream. This implies that it will be peer-reviewed by other Linux developers and may need to be fixed to comply with existing Linux kernel code. This is a very important part of new work done in the kernel. Without it, you and your product will be stuck with an old version of the kernel which the community and other vendors will not be eager to support. When the changes are pushed upstream, they are much more likely to get fixed as the kernel develops. A few free units of fun hardware in the right hands can go a long ways as far as overall software development expenses go.

Conclusion

We have touched on many projects that can be leveraged to offer rapid time to market when developing mobile and other embedded devices based on Linux. I hope this has been useful. There are many useful web sites that offer more information. The author plans to add links to all of the projects mentioned here and others as well in the wiki found on `http://eLinux.org/wiki/` if you would like more details. Good luck with your projects, and welcome to the community.