A Linux Appliance Construction Set

Michael W. Shaffer – Agilent Laboratories

ABSTRACT

Open source UNIX-like operating systems offer unique opportunities for administrators to create appliance style operating system and application packages that meet their own specific needs. This paper provides examples of several commonly useful appliance configurations based on the Linux operating system and documents the motivations, principles, and techniques behind the development of a basic 'Linux Appliance Construction Set' known as LxA which was used to produce them. Experience with LxA so far suggests that its use may help system administrators significantly reduce the amount of time spent deploying, maintaining, upgrading, and documenting certain types of hosts under their control.

Introduction

This paper details the philosophy, tools, techniques, and sample implementations that I have created in the course of developing a Linux Appliance Construction Set which I refer to as LxA. Certainly the philosophy and many of the the general principles detailed herein would apply to the construction of similar systems based on any open source UNIX-like operating system, but I chose Linux because of my long personal experience and familiarity with it. L*x*A expands to: Linux *x* Appliance, for instance: LRA abbreviates Linux Routing Appliance.

The original design parameters for LxA included the requirement that the entire boot and root file system images fit on a single 3.5'' floppy disk. The reader might legitimately wonder why I would embark on the creation of yet another mini Linux when there are already a number of excellent and similar projects active in the open source community [17, 11, 4, 10]. While most mini-Linux distributions aim to provide completely functional systems on low resource platforms, LxA attempts only to provide a single or small number of well defined features per configuration. It is not so much the minimal size as the minimalist philosophy that distinguishes LxA from its peers. In fact, some configurations such as LPA-CD (Linux Printing Appliance on CD-ROM) are quite large and require a CD-ROM for their root file system. Since the initial release of LxA, other community projects have arisen which pursue principles similar to those of LxA with apparently excellent results so far [5].

In general, LxA seems to be well suited to systems such as: Internet connection sharing routers, firewalls, bridges, routers, print servers, certain types of file servers (such as CD-ROM towers), terminal servers, point of sale terminals, browser kiosks, or in general any type of small to medium size single purpose system. LxA would probably not serve well for systems such as desktop 'power user' or developer workstations whose users regularly install, test, and uninstall many applications. LxA is also not intended to serve as a general purpose or standalone Linux distribution although it is hoped that the sample configurations will serve well as a basis for the future development of many different types of Linux appliances.

Motivation

As a system administrator, I frequently work part-time for small businesses, organizations, and departments who don't for one reason or another have in-house system administration staff. When, in mid 1999 I decided to move from South Carolina to Silicon Valley, I was faced with the dilemma of supporting a number of existing installations of Linux machines providing file, print, and network routing services which I would now be able to visit in-person perhaps only once a year. While Linux has inherited both a well deserved reputation for reliability and excellent facilities for remote administration from its UNIX ancestors, there are still situations which are extremely difficult to manage remotely. Furthermore, even with the recent surge in popularity that Linux has enjoyed it can still be difficult to find local companies or individuals who can provide reliable and affordable UNIX system administration to small customers in many regions.

A typical serious problem that I considered facing was the catastrophic failure of a hard disk or power supply in a critical machine such as a print server. While it would be easy to remotely coordinate procurement and installation of replacement hardware for such a failure, the rebuild or restoration of the software load for even a simple Linux server could be quite a dicey undertaking if trusted to inexperienced personnel working three-thousand miles away. I considered the possibility of having replacement equipment shipped to my new location, configuring it myself, and then shipping it on to its destination, but I felt that this approach was undesirable since it would at least double the already significant costs, delays, and risks inherent in shipping computer sized pieces of electronic equipment around the mainland U.S. In addition, there was always the risk that I might make some trivial error in configuration that would only be discovered when the system was powered up in its final destination and failed to work properly (worst of all would be a mistake that also prevented me from remotely accessing the machine).

In response to these considerations I formulated the idea of building Linux systems which would require no installation in the traditional sense but which would simply boot and run directly from the media on which they were delivered. My initial goal was to create fully functional internet connection sharing and print spooling systems contained entirely on a bootable floppy or CD-ROM. As I progressed in this endeavor, I also realized that I was unhappy with the significant amount of time that was required to 'adjust' the typical Linux distribution to my own purposes and tastes after installation, and I decided to enlarge the scope of the LxA project to include development and documentation of a general purpose framework for building small Linux systems tailored precisely to the needs of my typical customers. Such Linux systems would allow me to provide customers with backup copies of the software load for their systems by simply providing extra copies of their media upon delivery and would also allow me to deploy upgrades and fixes to systems much more economically by simply shipping new CDs or floppies to all affected customers. Thus it was that I embarked upon the creation of LxA with the following goals:

- Reduce system setup time
- Reduce system maintenance and upgrade time
- Reduce system complexity
- Reduce probability and impact of hardware failure

These are elaborated below.

Reduce System Setup Time

The typical procedure for setting up a new host using a general purpose operating system involves performing an installation from original media and then applying a long series of patches, packages, and procedures to adjust the system to the particular level of functionality and security desired. Certain operating system projects such as OpenBSD [14] seek specifically to minimize the number of unnecessary components in a default installation and to dramatically reduce the number of steps required to achieve a high level of security after initial setup. Other projects such as Bastille Linux [1] do an excellent job of distilling and codifying community knowledge of the best practices for a given platform into hardening scripts which aid administrators both in learning and applying a consistent security policy to new systems. LxA, on the other hand, attempts to provide a set of procedures and examples for quickly composing new systems based on minimal filesystem trees that include exactly the components required in the proper configuration. Once a prototype system tree is created and tested, it can be deployed to new systems much more rapidly and with greater confidence in its correctness and security.

Reduce System Maintenance And Upgrade Time

Many individuals and organizations seem to forget that most of the time spent on any given computing system will be spent maintaining it, not designing or installing it. For this reason, LxA seeks to reduce to an absolute minimum the time and problems involved in upgrading systems with new software. The typical LxA system will have its root filesystem and configuration files on one or more pieces of removable media, so upgrading the system can be accomplished instantly and reliably by simply halting the system and replacing this easily handled media.

The boot and configuration filesystems can be extensively tested before distribution to remote locations to ensure a smooth upgrade, and in the event of a malfunction it is trivial to just replace the old media and restore the system to a working state until further troubleshooting can be performed. The desire to facilitate this rapid and easy roll-forward and roll-back of system configurations was a primary incentive to the development of LxA.

By avoiding completely the typical hard-disk based installation procedure, LxA systems also encourage rapid deployment of updated software once it has been tested, since mass upgrades and downgrades involve drastically reduced time and risk of service interruption. Additionally, the backup of the system binaries and configuration files of an LxA system becomes almost trivial since it means simply producing and storing one or more copies of the boot and root disk images before distribution. This type of backup allows almost instantaneous recovery to known good states both from system failure and from system compromises or intrusions. Disaster recovery becomes simply a matter of obtaining suitable hardware instead of the time consuming and error-prone process of rebuilding a complete system from whatever notes are available (or even worse, from memory).

Reduce System Complexity

This was one of the primary motivations behind the early development of LxA systems. One of my original targets was to produce a working and useful Linux system composed of the absolute minimum number of pieces and, in the process, to gain an understanding of exactly what each piece provided and required. A second motivation was to reduce dramatically the amount of time and effort needed to thoroughly document the configuration and function of critical systems. All too often documentation is neglected or completely forgotten due to the time and effort required both to research and prepare it.

One can easily see that a Linux based firewall system such as LFA (Linux Firewall Appliance) composed of only around 40 files is far easier both to understand and to document than one based on a typical general purpose distribution which might include hundreds or even thousands of files after even a minimal installation. It is my belief that administrators are most likely to produce reliable and secure systems when they thoroughly understand all the components involved, and LxA seeks to facilitate this understanding by dramatically reducing system complexity.

Reduce Probability And Impact Of Hardware Failure

A typical small LxA system such as LFA will often boot from a floppy and will run entirely from a RAM disk once started. In such minimal configurations, there is no need for either a hard drive or a CD-ROM. This obviously eliminates at least two components which can fail, and the omission of these devices will also aid the longevity of the system power supply by reducing the strain placed on it. For systems such as LPA-CD with larger filesystem space requirements, it is still possible for the system to operate without employing a hard drive, depending on the application.

In some cases a fixed disk is the most efficient solution to certain requirements such as the need for swap or spooling partitions, but even in these situations the system can still be upgraded without the need to manage the internal storage. LxA systems typically feature some simple procedures in their startup scripts to optionally format, mount, and populate any necessary swap or spooling areas at boot time, and LPA-CD includes examples for using RAM disks, network filesystems, or fixed disk partitions for these purposes. If desired, LxA systems can even partition their own fixed disks at boot time through the use of a non-interactive partition editor such as sfdisk in their startup scripts. These features facilitate easy and rapid replacement of failed parts or even whole machines since LxA configurations can be easily tuned for no-install or self-install behavior on new hardware.

Principles

In pursuit of the goals stated above, LxA systems are generally designed to adhere to the following principles:

- Build systems by composition, not reduction
- Run from read-only and/or removable media
- Omit login and run-time configuration
- Use modern and standard software components

The next sections detail these principles.

Build Systems by Composition, Not Reduction

This means developing a set of practices for identifying the minimal set of components required for any given capability and then adding exactly those components to a base or minimal LxA prototype image.

Run From Read-only and/or Removable Media

Although some LxA configurations make extensive use of fixed disk partitions for /var, /tmp, and swap areas, the general rule is to keep all system binaries and configuration files on removable and preferably read-only media. The boot and root partitions may be made effectively read-only by loading them into RAM disks at run time, mounting devices such as floppy disks in read-only mode, or using physically read-only media such as write-once recordable CDs. While floppy and CD-ROM media are readily accessible and familiar to many users, some applications may benefit from the employment of alternative boot and root devices such as flash ram drives or cards.[1]

Omit Login and Run-time Configuration

This principle not only helps to save valuable kilobytes on boot media, but also makes the system much more tamper-proof in potentially hostile environments. Obviously this approach also requires the administrator to perfect the system configuration during staging. This concept perhaps more than any other distinguishes LxA from other small Linux systems such as the Linux Router Project [11]. Certainly this approach will not work well for all applications, but a machine configured in this fashion is a true network appliance, running only exactly what it needs to perform its designated functions. While console login facilities are often omitted in production LxA configurations, the example packages also include simple procedures for installing a minimal set of interactive tools and a console shell on the system for troubleshooting during staging.

Use Modern and Standard Software Components

In contrast to other mini Linux distributions, LxA uses the most modern and standard components available wherever possible. The Linux Router Project (See [11]), for example, employs the Linux 2.0 kernel, libc5, and the BusyBox POSIX tools package [2] to provide a wide range of capabilities while maintaining the smallest possible disk and memory footprint. Certainly there are a number of good reasons for using older components including smaller binary size, greater maturity, and often a more sedate pace of development. In fact, one of the greatest advantages of

[1] Discussions with LxA users during the writing of this paper led to the idea of building LPA systems based on a combination of a 48MB CompactFlash card, a CompactFlash to IDE adapter available from the Tuscon Amateur Packet Radio organization (See [16]), and commonly available hot-swap hard drive carriages for standard size hard drives. This configuration should feature higher reliability than either CD-ROM or floppy based systems while still affording ample capacity for typical configurations. As of this writing, 8-128 MB CompactFlash cards are available at reasonable price points. This option requires little modification to existing LxA configurations since the TAPR device makes a CompactFlash unit appear to a PC system as a standard IDE drive, and the benefits of easy system upgrades will be preserved since the removable hard drive carriage will make it easy to swap flash cards. Finally, this style of system would have the benefit of somewhat higher physical security than CD or floppy based configurations since many swappable drive carriages feature key locks, and the logistics of media distribution may even benefit slightly from the much smaller size and greater ruggedness of CompactFlash compared to floppy disks. Complete details of the outcome of this effort will be documented on the LxA project site [12].

using open source tools in my experience is that stable versions of products tend to remain viable for a very long time due precisely to the availability of source code that can be compiled for older platforms even when binaries are no longer readily available.

Nevertheless, I decided to avoid the use of older and non-standard components wherever I could. When pressed for space on small capacity boot media such as floppy disks, my approach was to save kilobytes by eliminating components entirely rather than recompiling, patching, or down-grading them. Usage of an older kernel and C library requires that all components be compiled specifically for that environment, and I wished to avoid forcing users to compile anything just to get a new LxA system running. Also, while older or customized components would undoubtedly save significant amounts of space on LxA systems, the benefits of using them would become completely negated if I wished to add a tool or service that depended on more modern and bulkier components such as glibc.

This principle represents a significant difference in philosophy between LxA and several of the other mini distributions available. LxA provides not so much a specific distribution of Linux as a framework for assembling this sort of mini system from any current Linux distribution at hand. Using standard components allows more users to modify the system in unforeseen ways and should also allow LxA to track new developments in the Linux kernel, major libraries, and applications much more closely than distributions which rely on heavily patched or custom components.[2]

Issues

Remote Administration

Since LxA configurations consistently use standard binaries and libraries from more conventional Linux distributions, they frequently leave little space on small media such as floppy disks for remote or even local administration tools. The floppy based configurations such as LPA (Linux Printing Appliance) simply have no space for components such as sshd or inetd. In fact, in the case of LPA, there is no room for even a a minimal shell environment. In comparisons with other Linux distributions, even those of the mini variety, this could be considered a serious disadvantage for LxA.

On the one hand, I consciously decided up front to eliminate console and remote login capabilities wherever possible both for increased security and to free up disk space desperately needed for other things. On the other hand, for configurations such as LPA-CD, I have in fact included both a proper login shell and sshd[3] to allow the possibility for remote administration if desired. So, the lack of remote administration facilities can perhaps be seen as either an advantage in disguise or as a non-issue depending on the configuration and applications involved. In some repects addressing this issue may simply require an adjustment of perspective. LxA was intended to address precisely those situations where remote administration capabilities are of little use, so what appears to be an advantage for other systems may in fact not be as significant as it seems in the environments for which LxA was designed.

System Logs and Spool Areas

The options LxA presents regarding boot-time management of fixed disks or the use of RAM disks for writable filesystems obviously raise the question of how to store persistent accounting information such as system logs. The typical LxA system is not one intended for hosting interactive logins, so it is assumed that most information of value would be included in the messages recorded by the system and kernel logging daemons. There is nothing to prevent an LxA system from using startup scripts that re-format the system log partitions only when absolutely necessary, and this would allow persistence of system logs for later analysis and troubleshooting in the event of problems. Alternatively, in certain environments it may be desirable to have LxA based machines send their log messages to one or more remote syslog hosts. Remote logging is often an excellent method for facilitating diagnosis of severe system failures or compromises and is commonly recommended in any event.

Another issue which may arise is that use of the automatic partition and format features would result in the loss of any pending jobs on filesystems such as print spool areas. In general, LxA simply leaves it to the individual administrator to evaluate their application for a system such as LPA-CD and determine what balance of resiliency, persistence, and hardware requirements fits their environment best. For example, the following three scenarios might be considered when deploying print spooling and format conversion servers based on LxA:

1. Booting from a CompactFlash device and using only RAM disks for spooling areas. This option would require careful consideration of the anticipated printing load, especially if a significant number of PostScript jobs would be spooled to non-PostScript printers. A configuration for moderate to heavy loads might benefit from the inclusion of 512-1024MB of RAM.

[2] Even as I was finishing this paper, I was working on upgrading all the LxA example configurations to linux kernel version 2.2.17, glibc 2.1.3, and the latest versions of all the binaries included. The total effort involved was a couple of kernel compiles, some work with find and cp to copy new binaries into the root filesystem trees, and re-making of the distribution tarballs. Other projects such as the Gibraltar firewall [5] have almost completely automated this process by employing scripts to find and assemble all required binary components on the fly from the machine on which the configuration is performed. Such an option would probably be a valuable addition to LxA and may be added in the future.

[3] Sshd will be a standard component of LPA-CD by the time this paper is published.

This would allow an ample 128-256MB for working set and leave 384-768MB for a /var filesystem on a RAM disk.

While this might at first seem like an extravagant amount of memory, consider that the result would be an entirely solid state print server that would likely perform extremely well, limited only by the speed of its CPU. Recent trends in hardware prices would make this only a moderately expensive configuration for most organizations. The obvious disadvantage of this configuration is that it makes no allowance for persistent spool areas, meaning all pending jobs would be lost in the event of a power failure or other serious event. In some cases this disadvantage may well be outweighed by the ruggedness, reliability, and extremely low maintenance requirements that such a configuration would present. This would probably work well for distributed print servers that need to be scattered throughout a building, campus, or region with little on-site system administration attention.

2. Booting from CD-ROM and using local hard drives for spooling, log, and swap area. This option is probably the easiest for the typical UNIX system administrator to evaluate and provision, and would probably work well in the a computer room or departmental environment where the desire is simply to have a standard and easily replicated configuration. Despite its conventional approach, this configuration still presents some options with regard to manipulation of the fixed disks at start up time.

   Simply mounting a pre-formatted and populated /var filesystem provides for a simple startup and complete persistence but requires that spool disks be partitioned, formatted, and populated manually at setup time or in a separate machine. Automatic partitioning and formatting, on the other hand, would provide for extremely quick setup and allow instant replacement of failed spool disks, and this might be a good choice if the cost of loosing spooled jobs and log files is felt to be outweighed by the consequent reduction in maintenance effort.

3. Booting from CD-ROM and using NFS exports for spooling and log areas (with perhaps file based swapping over NFS as well). This option features a mix of advantages and disadvantages from each of the two options above. On the one hand, it relies on having a network infrastructure already in place including at least one machine providing exported filesystems over NFS. On the other hand, if such an infrastructure is available, this option allows the simple startup and complete persistence of the second option, permits a completely solid-state hardware configuration for the print servers themselves as in the first option, and doesn't require either a huge amount of RAM or any local hard disks to be procured for individual print servers. The critical element to consider in evaluating this option is obviously the reliability and speed of the network and NFS file services available. Across a computer room, building, or even a small campus environment with widespread switched 100Mbit ethernet networks, this option could work nicely even for distributed print servers. An obvious disadvantage of this configuration is that a failure of the spool area file server or its network segment could potentially cripple a large number of print services, but the same could also be said of the centralized print spooling servers used in many more conventional arrangements.[4]

The main point the reader should note from these examples is that LxA by design makes choosing or even switching between all three of these alternatives almost trivial. Leaving aside the issue of hardware reconfiguration for a moment, consider the effort that would be required to configure a typical Linux distribution for either the first or third options, and then consider that with LxA choosing between them means changing just a few lines in one startup script. The whole intent of LxA is not to dictate or assume a certain pattern of operation but to make many different options available with as little effort as possible.

Older Hardware

While the low resource requirements of LxA systems may allow the use of hardware that would otherwise be considered obsolete, it is important to note that older hardware is often more prone to failure and, more significant for many organizations, no longer supported by warranties or service agreements. On the other hand, the upgrade cycle for end-user systems in many organizations makes available a significant number of systems each year that are due for retirement. These systems increasingly often pose an expensive and troublesome disposal problem for their owners. Given the right application and software load, some of these machines can be successfully re-purposed as departmental, standby, or load-sharing

[4]Upon reflection, the reader may reasonably ask what advantage there could be in this option over a more conventional, centralized print server. After all, if anything, this option introduces *more* hardware rather than less into the picture when compared to using a single massive machine for the job. One example might be a situation where an organization wishes to make a number of existing non-network and/or non-PostScript capable printers available for network printing throughout a building or campus. If a sufficient number of existing obsolete machines with network cards and CD-ROM drives are available, a configuration like this would allow upgrading these printers with both network direct and PostScript capabilities by connecting them via serial or parallel ports to LPA-CD machines distributed to wherever the printers physically reside.

servers complete with a ready supply of spare parts from any unused machines in the pool to counter the greater potential for failure and lack of support.

Tools and Techniques

I decided when creating LxA to start with an empty system and add features through composition instead of trying to cut down an existing distribution or system through a process of reduction. This approach, I felt, would result in a smaller, easier to understand system with less development time. I wanted to create and document a system in which I could explain the purpose of every single file and process. In addition to learning more about Linux and UNIX systems through this exercise, I hoped to also develop a toolbox of general principles and techniques for building other minimal systems in the future.

Assembling Services

This is a general description of the tools and techniques I use to figure out the necessary components for a given service. I usually begin, of course by locating and examining the binaries for the service I want. For this example I've chosen to use Samba, the NetBIOS file and print service for UNIX-like systems. I always start by installing and configuring the desired service on a staging machine before I try to deconstruct it for a mini system. Of course, any available man pages, web sites, mailing lists, and other available documentation for the service in question should be considered required reading throughout this process.

Distribution Package Managers

Most Linux distributions employ some sort of package management tools for installing and removing software. The two most commonly used package management tools are rpm (the RedHat Package Manager) and dpkg (the Debian PacKaGe manager). Distributions employing rpm include RedHat, SuSE, Mandrake, and many others. Debian, Corel Linux, Stormix, and a few others use dpkg.

The maintainers of package files have usually put a great deal of work into determining the dependencies of each particular package, so I will start there if a distribution package is available for a particular service. The first step is to use the package manager's 'list files' function to see exactly what files are installed on the system by a particular package. To query for the files installed by the rpm package for Samba, I would employ a command such as:

```
# rpm -ql samba
/etc/logrotate.d/samba
/etc/pam.d/samba
/etc/rc.d/init.d/smb
/etc/smbusers
...
/usr/doc/samba
...
/usr/sbin/nmbd
/usr/sbin/samba
/usr/sbin/smbd
/usr/sbin/swat
/usr/share/swat
...
/var/lock/samba
/var/log/samba
/var/spool/samba
```

Newer dpkg based distributions will often split services and their clients up into several packages to minimize dependencies and resource requirements for each package, so I start by querying the package management database to see what installed packages might be related to Samba; see Listing 1. This reveals four packages, and I can tell from the descriptions that I am probably only interested in the first two for the sake of an LxA system. To query for the files installed by each Samba package, I would next invoke dpkg with the -L option for each interesting package:

```
# dpkg -L samba
/.
...
/usr/sbin/smbd
/usr/sbin/nmbd
...
/var/samba
...
/etc/init.d/samba
/etc/cron.daily/samba
/etc/cron.weekly/samba
# dpkg -L samba-common
/.
...
/etc/samba/codepages
...
```

```
# dpkg -l '*samba*'
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: uppercase=bad)
||/ Name          Version   Description
+++-=============-=========-==============================================
ii  samba         2.0.7-3   A LanManager like file and print server for UNIX
ii  samba-common  2.0.7-3   Samba common files used by both clients and server
ii  samba-doc     2.0.7-3   Samba documentation.
pn  task-samba    <none>    (no description available)
```

Listing 1: Package management query: samba.

```
/etc/samba/smb.conf
...
/etc/pam.d/samba
...
```

In general, for the purposes of an LxA system, I ignore any files installed under /usr/doc, /usr/share/doc, /usr/man, /usr/share/man, and /usr/local/man. These files are simply the documentation and man pages and are not necessary for functioning of most services. Other files which are probably not necessary are headers placed in /usr/include or /usr/local/include. Files I give the most attention to are those placed in /etc, /bin, /sbin, /lib, /usr/bin, /usr/sbin, /usr/lib, /libexec, /usr/libexec, and /usr/share (outside of /usr/share/doc or /usr/share/man). Many services will install cron jobs (typically in /etc/cron.daily or /etc/cron.weekly), and these may or may not be necessary for an LxA system. Many times these cron jobs serve only to rotate logfiles, and such actions are not necessary for example when using a remote syslog host. On RedHat the files installed under /etc/logrotate.d may also be safely ignored for similar reasons unless the logrotate utility will be used on the mini system. In the example shown here, the RedHat Samba package includes the swat web based administration tool for Samba, whereas the Debian package seems to leave it to a separate installation. The decision to include or exclude utilities such as these will depend mostly on the size constraints of the boot media for the target system and the anticipated usefulness of the individual tools. Since I hardly ever use swat for managing Samba machines and it presents significant space requirements, I chose to omit it.

The rpm and dpkg package managers, in addition to providing information about specific packages, will usually also provide a list of other packages which a service requires or suggests for proper operation. To query for the description and dependencies of a package using dpkg:

```
# dpkg -p samba
...
Depends: samba-common (= 2.0.7-3),
        libc6 (>= 2.1.2), libncurses5,
...
```

Once again, if the specified dependencies are not already present on the LxA base system image, they must be located and added. The Debian package manager lists dependencies in terms of other packages that must be installed, whereas the RedHat package manager may list either other packages or individual files. Both package managers provide version information for dependencies as well; stating whether each required package must be of some exact version or simply greater than a certain version for compatibility. If a dependency is itself a complex package, it may be necessary to repeat the process of exploration until all levels of dependencies have been satisfied. An example command for listing dependencies using rpm would be:

```
# rpm -q --requires samba
pam >= 0.64
samba-common = 2.0.6
/sbin/chkconfig
/bin/mktemp
/usr/bin/killall
fileutils
sed
/bin/sh
ld-linux.so.2
libc.so.6
libcrypt.so.1
libdl.so.2
libnsl.so.1
libpam.so.0
libreadline.so.3
libtermcap.so.2
/bin/csh
/bin/sh
/usr/bin/awk
libc.so.6(GLIBC_2.0)
libc.so.6(GLIBC_2.1)
```

Both of the package manger listings included a dependency on a package named samba-common, so further exploration of this package would be necessary in this case.

Although the package manager file listing reveals a large number of accessories and configuration tools, I know from experience with this service that the actual file and print service portion of Samba needs only two basic binaries to function, namely smbd and nmbd. Unfortunately, I have found that in making this sort determination, there is little substitute for experience and careful observation, although well documented services may prove easier to analyze than those which are not. As a final step before copying the binaries into my LxA image tree, I use the ls command just to check if they have any unusual permissions or modes set which may need to be preserved:

```
# ls -al /usr/sbin/*mbd
-rwxr-xr-x 1 root root 338092
        Jul 27 03:56 /usr/sbin/nmbd
-rwxr-xr-x 1 root root 738556
        Jul 27 03:56 /usr/sbin/smbd
```

In this case there appears to be nothing out of the ordinary.

Ldd

After an initial survey using the package manager, man pages, and ls, I typically use ldd to see what system libraries the service components are linked with and where they are located:

```
# ldd /usr/sbin/*mbd
/usr/sbin/nmbd:
  libdl.so.2 => /lib/libdl.so.2
                  (0x2aac3000)
  libcrypt.so.1 => /lib/libcrypt.so.1
                  (0x2aac7000)
  libc.so.6 => /lib/libc.so.6
                  (0x2aaf4000)
  /lib/ld-linux.so.2 => /lib/ld-linux.so.2
                  (0x2aaab000)
```

```
/usr/sbin/smbd:
  libdl.so.2 => /lib/libdl.so.2
                    (0x2aac3000)
  libcrypt.so.1 => /lib/libcrypt.so.1
                    (0x2aac7000)
  libc.so.6 => /lib/libc.so.6
                    (0x2aaf4000)
  /lib/ld-linux.so.2 => /lib/ld-linux.so.2
                    (0x2aaab000)
```

If the libraries listed are not already present in the /lib directory of the LxA configuration I am working on, I would use something like:

```
# cd /lib
# ls -al libcrypt*
-rw-r--r-- 1 root root 20340 Nov 1
  13:56 libcrypt-2.1.2.so
lrwxrwxrwx 1 root root    17 Nov 9
  19:03 libcrypt.so.1 -> libcrypt-2.1.2.so
# find libcrypt* | cpio -pvd \
           /usr/local/linutopia/root/lib
```

to add them. I recommend using find and cpio instead of cp since these tools usually do better job of preserving permissions, modes, and linkages.

Lsof

If problems arise or I suspect that the service requires more than just the explicitly linked libraries shown by ldd, I will try executing lsof while the selected service is running to view a comprehensive list of file handles, sockets, and other resources the service has open. This command usually produces rather lengthy output, so I have snipped some irrelevant details from the example in Listing 2. I observe from this listing that both daemons are accessing another library which ldd didn't say anything about, namely libnss_files. They also appear to have some pid files and such open in /var/samba and /var/log, so I would make a note to make these directories available in the /var filesystem of the mini system. Finally, they have some network ports open that they are listening on, namely netbios-ns, netbios-dgm and netbios-ssn. I would certainly take these into account when working out any ipchains rule sets for the mini system.

Once the binaries, all required libraries, and any working directories are in place in the filesystem tree for the mini system, I would copy the /etc/samba directory contents over to the new /etc directory and modify them appropriately. When I really get stuck trying to figure out what obscure file or directory a daemon is missing in the mini configuration, I usually fall back on the old faithful strace utility. While this tool produces even more profuse and cryptic output than lsof, it is often the only means of identifying exactly what resources a process uses. Many utilities will open files or devices only briefly during startup or other phases of their operation, and this can sometimes foil efforts which rely solely on the snapshot view of system handles provided by lsof. The output from strace is a complete 'diary' of all system calls made by a process, and usually looking for calls to open(), stat(), read(), write(), socket(), connect(), send(), recv(), mmap(), and munmap() will reveal all the filesystem and network objects in use by the traced process. A typical invocation of strace would be:

```
# strace -o smbd.txt /usr/sbin/smbd -D
```

```
# lsof | grep mbd
nmbd 405 root txt  REG 3,66 338092  41150 /usr/sbin/nmbd
nmbd 405 root mem  REG 3,66  85238   4116 /lib/ld-2.1.2.so
nmbd 405 root mem  REG 3,66  10224   4167 /lib/libdl-2.1.2.so
nmbd 405 root mem  REG 3,66  20340   4163 /lib/libcrypt-2.1.2.so
nmbd 405 root mem  REG 3,66 936696   4119 /lib/libc-2.1.2.so
nmbd 405 root mem  REG 3,66  32104   4178 /lib/libnss_files-2.1.2.so
nmbd 405 root  3w  REG 3,66    804 923745 /var/log/nmb
nmbd 405 root 4ww  REG 3,66     20 878636 /var/samba/nmbd.pid
nmbd 405 root  5u inet 3016           UDP *:netbios-ns
nmbd 405 root  6u inet 3018           UDP *:netbios-dgm
nmbd 405 root  7u inet 3022           UDP lazarus.hammes-chasn.com:netbios-ns
nmbd 405 root  8u inet 3024           UDP lazarus.hammes-chasn.com:netbios-dgm
...
smbd 407 root txt  REG 3,66 738556  41149 /usr/sbin/smbd
smbd 407 root mem  REG 3,66  85238   4116 /lib/ld-2.1.2.so
smbd 407 root mem  REG 3,66  10224   4167 /lib/libdl-2.1.2.so
smbd 407 root mem  REG 3,66  20340   4163 /lib/libcrypt-2.1.2.so
smbd 407 root mem  REG 3,66 936696   4119 /lib/libc-2.1.2.so
smbd 407 root mem  REG 3,66  32104   4178 /lib/libnss_files-2.1.2.so
smbd 407 root  3w  REG 3,66    500 923748 /var/log/smb
smbd 407 root 4ww  REG 3,66     20 878637 /var/samba/smbd.pid
smbd 407 root  5u inet 3030           TCP *:netbios-ssn (LISTEN)
```

Listing 2: Using lsof to find resources used.

This example would write its output to the smbd.txt file.

Source Code

Of course, one should never forget the one unique resource available to all users of open source software, namely the source code. While reading the source code for system utilities undoubtedly requires the most significant investment in time and effort of any technique discussed here, it also ultimately yields the most significant and longest lasting results in terms of experience, understanding, and confidence when practiced regularly. Based on my own experience, I steadfastly maintain that acquiring and practicing even basic programming and code reading skills will make any system administrator much more capable, flexible, and confident when dealing with new systems, unfamiliar problems, and unusual configurations.

Configuring a Read-only Root File System

Actually, a workable read-only root file system for most Linux applications differs little from the typical hard disk based root file system. LPA-CD uses a slightly different file system layout from many Linux systems, but that's only for the sake of simplicity. The specific things that I found which needed attention included:

The /var file system

Many processes place run time files in /var. Even a completely CD based system that uses no configuration floppy will still require at least a small RAM disk mounted on /var so that processes like smbd have somewhere to save their PID and temp files. Usually I will make /tmp a symlink to /var/tmp to keep all writable areas under one mount point. When an LxA system is configured to format and mount a /var file system automatically at each startup, the necessary files and directories may be populated within it either by using tar to unpack a skeleton image or by running a simple script of mkdir, touch, chown, and chmod commands. LPA uses the latter approach since it has no room for the tar binary whereas LPA-CD uses the former to make modifications easier.

Sockets in /dev

A few processes, notably syslogd and lpd, create sockets in the /dev directory to listen for their clients. Fortunately, syslogd poses little problem since it accepts a -p option specifying an alternative path for the socket that it usually opens at /dev/log. For LPA-CD, I typically start syslogd with a command line such as:

```
# syslogd -p /var/run/syslog.socket
```

I then create a dangling symlink from /dev/log to /var/run/syslog.socket in the read-only root file system to direct syslog clients to the real socket.

BSD lpd, on the other hand, presents considerably more trouble since it offers no command line argument to specify its socket path like syslogd. I was forced in this case to break down and recompile the entire lpr package from source after changing a hard-coded #define in the lpr source code from /dev/printer to /var/run/lpd.socket. Again, I created a symlink from /dev/printer to /var/run/lpd.socket in case any clients outside the lpr package expected this socket to exist. When tracking down issues like this, I usually find lsof and strace to be indispensable.

Various Files in /etc

The only files in /etc I know of that a Linux system tries to write to under normal conditions are /etc/mtab and /etc/ioctl.save. I took the easy way out with mtab and just made it a symlink to /proc/mounts which the kernel maintains automatically. This change allows df and mount to work correctly, and the system will simply discard anything written to /proc/mounts. The ioctl.save file is read and written by init and is used to set console parameters when the system enters single user mode. While init normally creates this file at boot time, it does not actually require it and will function perfectly well in its absence. All the other files typically found in /etc can either be read only or can just be symlinks to something on another file system if preferred. The one command that I haven't found any way to use with a read-only /etc directory is passwd. Because of the way passwd shuffles files around while changing an entry in /etc/passwd, it doesn't seem possible to use it successfully unless it can write to all of the following: /etc, /etc/.pwd.lock, /etc/npasswd, and /etc/passwd.

If the /etc directory will be populated with symlinks, then there are exactly two files which must be physically present in this directory in order for init to start: /etc/inittab and /etc/rc.init. The /etc/inittab file cannot be relocated unless init is recompiled. Additionally, /etc/inittab must specify the path to the system startup scripts, and these scripts must exist on the root file system since nothing else will be mounted at this point. In the case of LPA-CD, I created an /etc/rc.init script which mounts the media for the /local file system so that all the dangling symlinks in /etc become valid. After running this script, LPA-CD's inittab hands off control to /etc/rc.local, which is actually a symlink to /local/rc.local, and this script handles the remainder of the system initialization.

Boot Process and Initrd Image

The Linux kernel contains support for a unique feature known as initrd or the Initial RAM Disk [9]. This capability is of critical importance to space constricted LxA configurations such as LPA since it allows a system to load and mount a small root filesystem image from a device that the kernel may not even contain drivers for. This mechanism requires only that the kernel have support for both RAM disk devices and initrd devices built in and places the burden on the boot loader to get the root filesystem image loaded and unpacked in memory along with the kernel. Thanks to this feature, the standard kernel for floppy based LxA configurations is able to defer support for both floppy and ide devices to loadable kernel modules for a moderate savings in kernel size. The typical boot sequence for Linux systems using the initrd mechanism is:

1. BIOS loads first stage boot loader from a master boot record into memory
2. First stage loads and executes second stage boot loader
3. Second stage boot loader (usually lilo or syslinux) loads the kernel and initrd image into memory and uncompresses both
4. Linux kernel executes the script /linuxrc on the initrd image if present
5. Linux kernel creates the first userland process by executing /sbin/init
6. The init process executes system startup scripts (rc.init and rc.local for LxA)

The original purpose for the initrd mechanism was to allow booting from devices such as SCSI disks using a standard modular kernel without support for these devices built in. The idea was that the /linuxrc script could be exploited to load required driver modules from the initrd root filesystem image and then mount the real root filesystem from another device. LxA actually doesn't use the /linuxrc script at all but instead just runs the whole system from the initrd image. For the floppy based LxA configurations, the standard script for producing an initrd image from a root filesytem tree is:

```
#!/bin/sh
rm -f ./disk/initrd
rm -f ./disk/initrd.gz
dd if=/dev/zero of=./disk/initrd \
          bs=1024 count=$1
losetup /dev/loop0 ./disk/initrd
mkfs.minix -i $2 /dev/loop0
mount /dev/loop0 /mnt
cd ./root
find . | cpio -pud /mnt &> /dev/null
cd ../
umount /mnt
losetup -d /dev/loop0
gzip -9 ./disk/initrd
exit 0
```

Creating a Bootable CD

Like most things PC, the mechanism for bootable CDs is a hack. While most PCs aren't all that good at booting from CD-ROM drives, they are really good at booting from floppies, so that's what they do. To make a CD bootable, I would first produce a floppy disk that boots the way I want it to. For instance, I would use a a script similar to the following:

```
superformat /dev/fd0
syslinux /dev/fd0
mount -t msdos /dev/fd0 /mnt
cp vmlinuz /mnt
echo "prompt 1" > /mnt/syslinux.cfg
echo "timeout 100" >> /mnt/syslinux.cfg
echo "default vmlinuz" \
           >> /mnt/syslinux.cfg
echo "append root=/dev/hdc" \
           >> /mnt/syslinux.cfg
umount /mnt
```

Now, I would reboot my test machine using the floppy to see that it loads the kernel properly and mounts the correct root device (/dev/hdc in this example). If all seems well, then I would make an image of the floppy with dd using a command like:

```
# dd if=/dev/fd0 of=boot.img
```

The file boot.img should then be placed within the directory tree from which the ISO file system image will be created. Finally, the -b option of mkisofs will cause it to place a special entry in the ISO9660 catalog pointing to this floppy image file. What the PC's BIOS actually will do at boot time is map this file on the CD to an 'emulated floppy' and then try to boot from it as if it were a real floppy disk. The rest of the BIOS and any boot loader residing on the floppy image are usually completely fooled. Of course, once the Linux kernel is loaded, the whole illusion evaporates and the CD-ROM goes back to being an ordinary device like /dev/hdc. The following commands would be typical for creating the actual ISO image using mkisofs:

```
# cp boot.img CD/boot/boot.img
# mkisofs -v -R -b boot/boot.img \
   -c boot/boot.catalog \
   -o img/lpacd.iso CD
```

Note that the path given in the -b option should be the path relative to the root of the CD file system. At this point the only remaining task would be using cdrecord to actually write the ISO image to a blank recordable CD:

```
# cdrecord -v speed=4 dev=2,0,0 \
                         img/lpacd.iso
```

This command is typical of what I would use on my own machine, but the speed= and dev= parameters are specific to the particular hardware in use. The -scanbus option to cdrecord will show what devices are available and what value to pass in the dev= parameter for each particular drive. The speed= parameter should be set to a value compatible with the capabilities of both the drive and media used. The mkisofs utility has a number of other options for specifying boot disk and kernel images when creating bootable images. Although the LxA example configurations have not yet been built for Sparc hardware, bootable ISO images for that platform can be produced as well. For more comprehensive details regarding the usage of mkisofs and cdrecord [13, 7].

One important choice that must be made when creating boot and root disks on recordable CD-ROM media is whether to use CD-R (write-once) or CD-RW (re-writable). Despite the obvious penalties for mistakes when using a write-once media, I prefer to use CD-R unless I am completely sure that all target machines have CD-RW compatible drives installed.

Many older CD-ROM drives are not capable of reading or booting from CD-RW disks, but if the drive includes 'Multi-Read' compatibility in its specifications it should read either media with no problem.

Using the Kernel Command Line

In order to allow the maximum amount of flexibility for booting an LPA-CD image without requiring changes to the rc.init script burned onto the CD, I decided to employ a couple of custom kernel command line parameters to pass information to rc.init through the boot loader. The basic idea when using the kernel command line on Linux is to insert whatever parameters are required into the append= parameter of the boot loader and then take advantage of the Linux /proc/cmdline node to extract these parameters from the kernel at runtime. For the sake of LPA-CD, two parameters were required, the block device and the filesystem type for the /local filesystem. The command line parameters are read from the file syslinux.cfg by the syslinux boot loader at startup time and are specified as follows:

```
append root=/dev/hdc lxa_lfs=vfat \
        lxa_ldev=/dev/fd1
```

These parameters are then passed on the the kernel by the boot loader and may be retrieved from the /proc/cmdline node with various utilities such as awk. The actual rc.init script used for LPA-CD and illustrating this technique is shown in Listing 3. Any number of utilities such as cut, sed, or grep, or even perl can be used for this sort of task, but I chose awk since it was the smallest binary which provided a general purpose scripting and text processing tool available on my development system.

Example LxA Configurations

LRA: Linux Routing Appliance

This system provides: a DHCP server, local and caching DNS, dial-on-demand PPP, firewalling, and masquerading for a local network. The entire system boots from a single floppy and runs from a RAM disk. The minimum hardware requirements are: a 486 CPU, 16MB RAM, an ethernet card, a 3.5" floppy, and a modem. LRA was the first configuration developed and seems from analysis of the LxA site access statistics to be the most popular among users.

LFA: Linux Firewall Appliance

LFA requires little more than a floppy disk drive, an ethernet card, a CPU, and some RAM. Users of LFA have reported that it works adequately with as little as 6MB of RAM even with the interactive login package added. All of the work of this system is done by the kernel's own firewalling and routing modules, and user reports suggest that LFA can function well for moderate traffic loads across one or two 10Mbps ethernet segments even with only a 486 CPU. Heavy loads, more segments, or 100Mbps ethernet would definitely require at least 16MB of RAM and a Pentium class CPU for satisfactory operation. LFA is, I believe, almost as minimal a Linux system as one could possibly assemble and still perform useful work. After the startup scripts have run the only processes besides the kernel and init on this configuration are syslogd and klogd. Because of the minimal process load, LFA is extremely stable, runs entirely from a RAM disk, and leaves little opportunity for compromise or failure. This system can provide firewalling, routing, and masquerading between two or more ethernet networks. For many small businesses or isolated departments, a commercial hardware or software firewall package may be excessively costly and present many features that would go unused. In situations such as these, LFA can provide a secure and reliable alternative which supports many of the most useful features of a commercial firewall or router at much lower cost. Excluding the /dev directory, the root filesystem for this configuration contains only around forty files and directories; see Table 1.

```
#!/bin/sh
PATH=/bin:/sbin
export PATH
# Mount /proc filesystem first
mount -n -t proc proc /proc
# Parse custom command line parameters from /proc/cmdline
eval `cat /proc/cmdline | awk '{ match($0, /lxa_lfs=[^ ]+/); \
  printf("LXA_LFS=%s ; export LXA_LFS;\n", \
  substr($0, (RSTART + 8), (RLENGTH - 8))); }'`
eval `cat /proc/cmdline | awk '{ match($0, /lxa_ldev=[^ ]+/); \
  printf("LXA_LDEV=%s ; export LXA_LDEV;\n", \
  substr($0, (RSTART + 9), (RLENGTH - 9))); }'`
# Mount /local filesystem
echo -n 'Mounting '$LXA_LDEV' on /local as type '$LXA_LFS'...'
mount -n -t $LXA_LFS $LXA_LDEV /local
echo 'done.'
exit 0
```

Listing 3: The rc.init script used for LPA-CD.

Path	Comment
/bin	location for user binaries
/bin/sh	ash, an interpreter for startup and other scripts
/bin/hostname	sets and echos the system hostname
/bin/echo	used for writing configuration settings into /proc filesystem nodes and displaying boot messages
/etc	location for system configuration and boot files
/etc/inittab	configuration file for /sbin/init
/etc/fstab	list of active filesystems, their locations, and options
/etc/hostname	contains the system hostname
/etc/syslog.conf	configuration file for /sbin/syslogd
/etc/hosts	local hostname to IP address mappings
/etc/rc.init	first setup script executed by /sbin/init
/etc/rc.local	main system startup script
/etc/networks	local network name to IP subnet mappings
/etc/services	mapping of port numbers to service names
/etc/protocols	mapping of protocol id numbers to protocol names
/etc/passwd	list of local user accounts
/etc/group	list of local user groups
/etc/nsswitch.conf	configuration file for GNU C library map functions such as gethostby-name()
/lib	location for dynamically linked libraries
/lib/ld-linux.so.2	the dynamic library loader, used by all dynamically linked executables
/lib/libc.so.6	the standard GNU C runtime library, huge but essential
/lib/libnss_files.so.2	name service switch module used by C library for lookups from files such as /etc/hosts
/lib/modules	location for kernel modules
/lib/modules/*	any required drivers for devices such as ethernet adapters
/linuxrc	optional pre-init script executed by the kernel at startup (not used by LxA)
/proc	mount point for the kernel's proc pseudo filesystem
/sbin	location for system binaries
/sbin/hwclock	used to sync system time with hardware clock
/sbin/ifconfig	configure network interfaces
/sbin/init	process id 1 started by the kernel at boot time
/sbin/ipchains	used to configure the kernel's IP firewalling rules
/sbin/mount	used to mount and remount /, /proc, and any other filesystems
/sbin/route	used to configure network routing tables
/sbin/insmod	used to load modules such as ethernet drivers
/sbin/syslogd	system message logging daemon
/sbin/klogd	kernel message logging daemon
/tmp	location for working and scratch files
/var	location for spool and log files
/var/run	location for lock and pid files

Table 1: Root filesystem.

Due to its extremely minimalistic nature, I have also found LFA useful as a baseline configuration for creating things like custom rescue and repair diskettes.

LPA-CD: Linux Printing Appliance on CD-ROM

System Requirements

LPA-CD is a complete Linux system which will boot and run directly from the ISO image provided on the LxA project site. The minimal system requirements for an LPA-CD system are: a 386 or 486 CPU, 4MB RAM, an ethernet card, a floppy disk drive, a CD-ROM drive, and obviously at least one printer. Recommended but optional items include a fixed disk or Zip drive for temporary files and swap space and as much RAM as the budget will allow for better performance and higher load capability. While LPA-CD contains many more capabilities than LFA, it benefits slightly in terms of RAM requirements as a result of mounting its root file system from the CD-ROM instead of a RAM disk. The minimal configuration has proven to work well for a light load such as a dozen Windows clients printing small to moderately sized documents through client side PCL drivers to PCL printers. Situations involving large numbers of clients, large print jobs, or conversion of complex, color PostScript jobs will benefit from as much CPU, RAM, swap, and spooling space as the system can hold. Only evaluation of the anticipated system load can dictate what hardware configuration can be considered minimal for a given situation.

Features

An LPA-CD system will spool, convert, and output print jobs from either UNIX lpr or Windows Samba clients. This package allows easy setup and management of print spooling hosts for mixed platform networks. LPA-CD is not based on any particular distribution, but the original sample configuration included components from the Debian 2.2 (Potato) system with the Linux kernel version 2.2.14 and glibc version 2.1.3. In the current release, LPA-CD includes the option for a minimal but complete set of interactive tools and a sulogin shell for use during staging. Since the system includes tools like vi, it is possible to save configuration changes to onto the local configuration floppy while the system is operating (something that is not possible with many of the more resource restricted configurations such as LRA, LFA, and LPA). This allows an LPA-CD box to be started and configured completely standalone if desired. Basic features in the current release of LPA-CD include:

- Linux kernel version 2.2.17
- Glibc 2.1.3
- Linux ipchains kernel firewalling
- PAM (Pluggable Authentication Modules)
- Samba for serving Windows printing clients
- BSD lpr for UNIX printing clients
- Ghostscript for conversion of PostScript/PDF print jobs to printer-native output
- Netpbm for conversion of various graphics formats to printer native output
- Magicfilter for easy and flexible configuration of print job conversion based on file magic numbers

LPA-CD uses the standard BSD lpd for spooling of print jobs, and can support both directly connected serial or parallel printers as well as network attached devices such as HP JetDirect adapters. LPA-CD fills a unique application niche since it provides both print job spooling and automatic conversion of many job formats for UNIX as well as Windows clients. The TODO list for LPA-CD also includes adding support for MacOS clients via the Netatalk package in the near future.

System Configuration

An LPA-CD system is comprised of two disk images, a bootable CD-ROM and a bootable floppy. Either may be used to bootstrap the target machine since many older machines don't have a BIOS capable of booting from a CD-ROM drive. The result is the same in either case; the root filesystem will be the ISO9660 filesystem found on the CD, and all configuration files that would need to be changed locally reside on the floppy image which will be mounted on /local at system startup. LPA-CD offers two choices for a quick start; using disk images or using a tarball. If the environment will require only changing some configuration files, then the disk images are probably easier to start with. If, however, extensive changes or additions to the CD image are anticipated, then the tarball will prove more suitable. The CD image and bootstrap procedures are intended to work well for the majority of typical setups, and the tarball is probably only appropriate if the configuration requires unusual features or extensive changes. In either case, of course, either a CD-R or CD-RW drive is required for actually writing the ISO image to a CD-R disk.

System Startup

The rc.init script included on the CD image is responsible for mounting the appropriate filesystem on /local, and the /local/rc.local script actually does most of the work of starting up an LPA-CD machine. This script should be tuned for the specific machine on which it will be run. Only a summary description of the script is provided here. The phases of startup include:

- Loading modules: Near the top of the file are several invocations of modprobe which load driver modules for devices such as the ethernet adapter.
- Preparing and activating the /var filesystem: After any required modules are loaded, there are several alternative sections in the rc.local script showing examples of how to create and/or mount various types of filesystems for use as temp and spooling areas. Only one of these alternatives should be used, and the others

may be removed or ignored. As mentioned in the Issues section, a careful evaluation of the resources available and the anticipated load should be made before choosing this option. Examples for each alternative are shown here:

```
# RAMDISK
dd if=/dev/zero of=/dev/ram0 \
       bs=1024 count=4096
mkfs.minix -v -i 8192 /dev/ram0
mount -t minix /dev/ram0 /var

# LOOPBACK IMAGE
losetup /dev/loop0 /local/var.img
mkfs.minix -v -i 8192 /dev/loop0
mount -t minix /dev/loop0 /var

# LOCAL DISK
mkfs.minix -v -i 8192 /dev/hdb1
mount -t minix /dev/hdb1 /var

# NFS
modprobe nfs
mount -t nfs \
   spoolhost:/spool/linprinter /var

# Unpacking /var filesystem image
tar -C /var -xvzf \
           /local/var.tar.gz
```

The -v option to mkfs.minix causes a Minix V2 filesystem to be created which is necessary for volumes over 64MB in size. The -i 8192 option causes allocation of 8192 inodes in the new filesystem which is a somewhat larger number than the default. If volumes of more than 256MB will be used, then the ext2 filesystem is probably a better choice.

- Activating network interfaces: This section includes standard ifconfig and route invocations to set up the interfaces and routes as necessary for the particular environment.
- Configuring ipchains rulesets: The default ipchains statements included install a fairly strict set of filters allowing only the traffic necessary for lpr and Samba clients to connect. The rules also allow clients on the local network to ping and be pinged by the LPA-CD box to aid in diagnostics.
- Starting services: This section at the end starts syslogd, klogd, nmbd, smbd, and lpd.

Sulogin Shell

The CD image contains a minimal but mostly complete set of tools for system setup, monitoring, and diagnosis.[5]

- Interactive/miscellaneous commands: awk bash clear cryptwd echo sh vi
- File operations: cat cp chmod chown dd find grep less ln ls mkdir mv rm touch
- Filesystem maintenance: df dosfsck e2fsck fdisk fsck fsck.ext2 fsck.minix fsck.msdos fsck.vfat losetup mkdosfs mke2fs mkfs mkfs.ext2 mkfs.minix mkfs.msdos mkfs.vfat mkswap mount swapoff swapon umount
- File archive maintenance: cpio gzip tar
- System status: date dmesg free hostname lsmod lsof ps strace uname
- System/process Control: halt hwclock insmod kill modprobe reboot rmmod setserial shutdown tunelp
- Networking: ifconfig ipchains netstat ping route
- Samba: mksmbpasswd smbclient smbpasswd smbspool smbstatus testparm
- BSD print spooling: lpc lpq lpr lprm lptest pac
- Print job file format conversion: magicfilter bmptoppm djpeg dvips fig2dev giftopnm gs pngtopnm pnmtops rasttopnm sgitopnm tiff2ps

Conclusions

Open source operating systems present unique opportunities for creation of ultra light systems tuned to specific tasks. By exploiting the flexibility and rich knowledge base of systems like Linux, FreeBSD, OpenBSD, and other open source systems, administrators can greatly reduce the time required to set up, maintain, and upgrade the systems under their control. LxA facilitates this process by providing documentation, tools, and ready-to-run reference implementations for several useful Linux based network appliances. By taking a different approach to composing its Linux based platform, LxA reduces system complexity while increasing security and reliability. In addition, LxA will hopefully be able to quickly incorporate new versions of critical system components when available due to its use of standard components wherever possible. When compared to commercial UNIX systems, Linux presents more options for the average system administrator in building unusual or small system configurations due primarily to the following features:

[5]In production configurations many of these tools may go unused, but they are included in the ISO image for LPA-CD by default since the cost to add them is relatively much higher for a write-once media like CD-R versus a re-writable media such as a floppy. The floppy based configurations such as LRA include in their configuration scripts the option to add a login shell if desired.

The only non-standard tool included with LPA-CD is cryptwd which is a small utility designed to help replace the ordinary passwd tool. As discussed previously, the standard passwd utility will not function correctly with a read-only /etc directory, and this obviously causes problems for a CD based root filesystem. The cryptwd tool (in combination with vi) will allow the changing of user account passwords on a standalone LPA-CD system. To use cryptwd, one would first open the /etc/passwd file in vi and then use an ex command like the following: `:r ! cryptwd <newppassword>`. This will place a string representing the crypted version of the password specified into the current editing session. If it is necessary to include special characters in the password string, they may be escaped with a '\' like this: `:r ! cryptwd \%\&chevy`. Of course, this command can also be used at a shell prompt if the crypted version of some string is needed. This utility chooses a crypt salt for the password randomly, so there is no need to specify one on the command line.

- A standard and flexible RAM disk device
- The initial RAM disk boot capability
- Ease of composing minimal systems suitable for running from removable media
- Generally more sophisticated tracing tools
- Free access to source code for troubleshooting and modifications where necessary
- An overall system ethos focused on flexibility and hackability

The LxA project has definitely succeeded in providing me with a more thorough understanding of the components involved in bootstrapping and running Linux and other UNIX-like systems. In more practical terms, LxA configurations have made it feasible for me to continue supporting customers located at great geographic distances from my new home without excessive travel or expense. As with any open source project, it is difficult to estimate accurately any rate of deployment or success for LxA outside the realm of my own experience. I can say, however, that there have been approximately twenty to thirty thousand visits to the LxA project site since it was first created around December of 1999. Of those who have visited, at least several thousand have taken the time to download one or more of the sample configurations. Of the several dozen users who have sent me email regarding LxA, all have reported positive results and at least several have suggested new features and configurations which they would find useful. I have received at least a dozen replies to a request for usage reports so far, with those responding most often finding a use for the LRA and LPA-CD configurations. Purely subjective observation suggests that a large proportion of active LxA users are from outside the U.S., and I would suppose that this can be attributed to the emphasis the project places on its relatively low resource requirements and the prevalence of older hardware platforms in many areas of the world.

Future Work

TODO List

- More elegant make scripts and better support for RedHat based host distributions.
- Analysis scipts to automate the process of identifying and collecting all the components and dependencies of new services for LxA configurations
- A simple console dialog and/or X based GUI configuration utility for building LxA root file system trees and disk images.
- Enhancement of the LPA-CD root file system image with additional components to form a more general-purpose 'LxA Platform CD'. This will include sshd for remote administration as well as the netatalk package for support of MacOS printing clients and some other services such as dhcpd and pppd.
- Integration of the Linux kernel 2.4 when released.

Alternative Configurations

- LLA (Linux Lynx Appliance): Useful for turning ancient 386 class machines into text based POS terminals, etc.
- LMA (Linux Mozilla Appliance): Another CD based configuration; the classic thin client for creating web and Java based network application systems.
- LCDA (Linux CD-ROM Tower Appliance)
- LTSA (Linux Terminal Server Appliance)
- LVA (Linux VPN Appliance)

Availability

All of the disk images, documentation, and other materials related to LxA are publicly available via the Internet [12]. All software associated with LxA is covered, as are the components on which it is built, by the GNU General Public License [6]. Everything documented in this paper is freely available for any use as long as the distribution requirements of the GPL are followed.

Errata

Any corrections or additions to this paper will be posted on the LxA project site [12] where the full text of this paper will be posted and maintained in HTML format as well.

Author Information

Michael W. Shaffer has worked as a system administrator, software developer, and system engineer for a variety of companies and has been using Linux since he first discovered it in mid-1993. He studied Spanish, English Literature, Philosophy, and most recently Computer Science at the University of South Carolina in Columbia, SC. Although he was born in Connecticut and grew up in South Carolina, he has recently moved to Silicon Valley and currently works as Hostmaster, Postmaster, and Security Officer for the Research Computing Services department of Agilent Laboratories in Palo Alto, CA. Reach him at Agilent Labs RCS; 3500 Deer Creek Road; Palo Alto, CA 94304; USA. His phone number is +1 650-485.2955. His email address is: shaffer@labs.agilent.com .

References

[1] Bastille Linux Project; http://www.bastille-linux.org/ .
[2] BusyBox; http://busybox.lineo.com/ .
[3] Embedded Linux Stargate; http://linux-embedded.com/ .
[4] Floppix; http://floppix.ccai.com/faq.html .
[5] Gibraltar Firewall Project; http://www.gibraltar.at .
[6] GNU Project General Public License; http://www.gnu.org/copyleft/gpl.html .
[7] Linux CD Writing HOWTO; http://www.linux.org/help/ldp/howto/CD-Writing-HOWTO.html .

[8] Linux Kernel HOWTO; http://www.linux.org/help/ldp/howto/Kernel-HOWTO.html .
[9] Linux kernel source tree documentation (including initrd.txt and other files in the Documentation directory); http://www.kernel.org .
[10] LOAF; http://loaf.ecks.org/ .
[11] LRP (Linux Router Project); http://www.linuxrouter.org/ .
[12] LxA Project Homepage; http://equusasinus.com/lxa/index.html .
[13] Mkisofs and cdrecord documentation (including cdrecord(1), mkisofs(8), and README.eltorito); http://www.fokus.gmd.de/research/cc/glone/employees/joerg.schilling/private/mkisofs.html .
[14] OpenBSD Project; http://www.openbsd.org .
[15] Open Directory Tiny Linux Page; http://dmoz.org/Computers/Software/Operating_Systems/Linux/Distributions/Tiny_Linux/ .
[16] TAPR CompactFlash/IDE adapter; http://www.tapr.org/tapr/html/Fcfa.html .
[17] Trinux; http://www.trinux.org/ .
[18] Yahoo! Linux Distributions Page; http://dir.yahoo.com/Computers_and_Internet/Software/Operating_Systems/Unix/Linux/Distributions/ .