Introduction to Stateless Linux

Proposal for the Fedora Project

Last revised September 13, 2004

Introduction

Red Hat has been prototyping an OS-wide collection of changes we call “stateless Linux.” Some of these changes have started to appear in Fedora Core. Based on our experience so far, this is an exciting direction to take the operating system and we want to propose it to the wider Fedora community. Please send us your feedback.

This proposal describes:

- The “stateless Linux” initiative goals and philosophy
- How the project affects Fedora policies and development
- Our initial implementation
- Future plans and possibilities

Stateless Linux

Some ideal properties of an operating system:

1. If you throw a computer out the window, you should be able to recreate its software, configuration, and user data bit-for-bit identically on a new piece of hardware.
2. In any managed deployment from school workstation lab to enterprise server room, single computers should never be modified. Instead, all computers that need the modification should be modified in a single step.

Many people have configured Linux to work without a local hard drive. Imagine that the root filesystem is mounted read-only over NFS, and shared among multiple computers. Imagine that the user home directories for this system are also NFS-mounted.

This diskless deployment is the simplest example of a stateless Linux configuration. No state exists on single computers; all state is centralized.

The LTSP thin client project combines this diskless model with a terminal server, running applications remotely and displaying them on clients.

A traditional Windows-style fat client is at the opposite extreme. Each system can be modified locally. This means that each system must be individually backed up, and can have individual headaches such as updates that fail to apply, a corrupt registry, or whatever. Even enterprise Windows deployments often allow users to install software, mix user data with system components, and so forth.

Over the years, thin clients have gone in and out of fashion. We believe this is because thin clients force a tradeoff; they have important advantages due to their stateless nature, but also important disadvantages.

One goal of the stateless Linux project is to move toward a “best of both worlds” hybrid between thin and fat client. Its properties include:

- Applications run on local systems
  - avoids the need for huge terminal servers with complex load balancing
  - works for laptops
- Software and data are cached on the local disk
  - reduces bandwidth and increases speed
  - the cache can be read-only and thus per-computer state is impossible
  - works for laptops

As part of the stateless Linux project, we are attempting to create this “cached client” technology. However, the stateless Linux plan spans all of thin, fat, and “cached” client: it is first and foremost an architecture and a philosophy for how to design the operating system. The exact deployment details are tunable for the needs of each environment.

We are starting the stateless Linux work in a desktop context, but its advantages are equally applicable to the server. An especially interesting application in the server world is to manage a collection of virtual server instances running on one physical machine. In this case, you would expect the virtual machines to share the same root filesystem.

What about...

Before anyone panics: stateless Linux is an addition to the existing ways to deploy Linux, not a replacement. We do think it will be useful in a wide range of cases, but if it doesn't work for a particular case, the usual Linux deployment styles will still get the job done.

Instantiation

All exciting new software projects have to introduce some new terminology, and stateless Linux is no exception.

We are using the word *updates* in the traditional sense of delivering new RPM packages and installing them to a root filesystem.

We are using the word *instantiation* to refer to getting a root filesystem from a central server to the individual system that will boot the root filesystem. Instantiation is done on a file basis, rather than by installing packages.

The generic term “instantiation” is needed because there are at least two ways to replicate a root filesystem:

- Diskless via NFS, AFS, GFS, or other network filesystems

- Cached using either a userspace solution or a network filesystem that supports local cache

One goal of the stateless Linux initiative is to treat these two cases in the same framework. So for example a single root filesystem could be mounted diskless by some computers, and cached by others.

Stateless Linux doesn't always involve instantiation; many of the stateless principles and goals apply equally to systems updated directly with RPM packages. Such systems can also be managed with a one-to-many, “no local changes” methodology, given the right software.

Implications for Fedora Project Developers

Stateless Linux has a number of implications for how specific packages in the operating system should work, and what capabilities the operating system should have.

Read-only root filesystem

The read-only root enforces statelessness, which tends to break a lot of packages. To address this, we've developed a toolkit of solutions; ways to avoid per-computer state, or store it elsewhere. The right solution varies depending on the situation.

Put the state in memory

Some files are not really meant to be persistent; for example, everything that gets written to /tmp. These files can safely vanish anytime the system reboots. The simplest solution is to keep /tmp in RAM.

The same solution works for dynamically-generated files. For example, when /etc/resolv.conf is written out from DHCP data, it can safely be kept in RAM. Kudzu's /etc/sysconfig/hwconf is a similar case.

Determine the state dynamically

DHCP and hardware probing are two cases where rather than configuring something in advance, it can be recomputed each time.

System administrators can also use this technique to work around issues unique to their deployment. For example, a custom initscript could dynamically determine how to set up a particular machine – but the script is identical on all machines, and the script's results only reside in RAM.

One way to dynamically configure a machine is to query a central server...

Store the state on a central server

Some examples:

- LDAP or NIS for user accounts, rather than /etc/passwd
- Locate printers using LDAP or DNS-SD (ZeroConf)
- Place the state on a file share
- Have syslog send data over the network

In combination with scripts that key off the machine's MAC address, network location, hardware, and other features the possibilities are endless.

Some local state doesn't matter

The only example of this so far is a random seed, which is useful to persist across reboots – but nobody is going to cry if it gets lost.

Store the state locally, but back it up automatically

For laptops, the user's home directory has to be local. However, we can keep it reliably backed up:

- Each time we detect a network link to the corporate network, copy the latest changes to a file server
- Do this automatically and unobtrusively (perhaps with some "notification area" indicator)

Ability to update a file tree

With many stateless Linux setups, a file server exports a root filesystem for instantiation. This means that software installation and update tools, such as yum and up2date, have to operate on the file tree that will be exported, rather than to "/" on the currently running system.

Anaconda also needs the ability to install to a file tree, in addition to a physical system.

Dynamic hardware handling

As with a live CD distribution, manual hardware configuration can't be written to disk. At least for systems with modern, reasonable hardware, dynamic detection and setup of the available devices should be possible. Automatic hardware setup happens to be the Right Thing from a user interface point of view, as well.

For deployments with problematic hardware, admins using stateless Linux will have to write scripts to look at the system and set up the right configuration according to system type, information in the directory, or any other available means.

Users should not need root

One of our user interface goals is that desktop productivity users should never need the root password. This coincides nicely with the stateless Linux model, where even root can only write to /tmp and the home directory, so giving users root doesn't add many new capabilities.

Hardware setup was the largest category of "reasons end users need root today." The Red Hat desktop team has been working hard to eliminate hardware setup, for example:

- Shared printers are automatically detected without configuration
- Local printers are automatically configured when plugged in

- Network interfaces are automatically brought up when a link is present
- Users can choose wireless essid and enter an encryption key from the GUI, without root privileges
- Storage devices and input devices are automatically set up when plugged in

One tricky issue is the date and time; on laptops at least, system-config-date probably should not require a password. It makes people nervous to eliminate the password requirement by default, so it will probably be an additional burden on local sites to set it up for their users.

Generally speaking, we feel that any end user (non-admin) task that requires the root password should be considered a bug.

Initial Implementation Progress

We've been prototyping some aspects of the stateless Linux initiative in Rawhide, leading up to Fedora Core 3. It's by no means fully-baked, but we do have some good progress and perhaps a better understanding of the problem space.

Hardware just works

A major focus has been removing the need to be root to use your hardware, and ensuring that we autodetect and set up hardware on system boot.

As mentioned earlier in this paper, our efforts have covered printing (local and shared), networking (wired and wireless), USB devices such as pen drives, input devices, and more. Red Hat developers are leading the work on HAL, D-BUS, and other components that enable this to happen.

No question that work remains. However, a categorical policy going forward is that we expect an office productivity worker to be able to use all their hardware without the root password, and without touching the command line. We will consider it a bug if desktop-class hardware requires either root or opening a terminal.

Read-only root filesystem support

The package “readonly-root,” if installed, modifies the boot process to mount the root filesystem read-only, and mount certain files and directories read-write in RAM.

Basic tools for managing an OS install tree

We have some basic tools to install an OS to a directory, and take “snapshots” of it. A snapshot freezes the OS install in time so it can be instantiated without changing as it is being copied.

mkinitrd support for diskless clients

The “readonly-root” package also contains a diskless-mkinitrd script which supports creating an initrd to be used in conjunction with pxelinux for network booting. The initrd is created as part of the snapshot process and differs from the non-diskless case in that the network interface is brought up and a root filesystem is mounted over NFS. The diskless-mkinitrd script is intended to be merged into mainline mkinitrd over time.

Rsync-based cached instantiation prototype

To experiment with the “cached client,” we've implemented a simple setup using rsync to copy the OS install to individual clients. This implementation uses two disk partitions, each with a copy of the OS. Updates are performed on the copy that is not in use, for a “double buffered” effect.

There are a number of limitations to this approach, some of them fixable by tweaking the implementation, and some requiring a different approach entirely. However, we wanted to get the overall architecture roughly working before trying to engineer all the details.

That caveat in mind, here are some more details on the current prototype. There are five partitions on a cached client system:

- active root
- active /boot
- reserve root
- reserve /boot
- swap

The “stateless-client” RPM contains a cron job that runs every hour. The hourly cron job asks an LDAP server which OS install the client should be running. It compares this with what's currently cached; if the client is running the wrong OS, it looks up an rsync server in LDAP, and rsyncs the OS to the reserve root partition. When the root partition is fully updated, it updates the reserve /boot partition, and modifies grub to swap the reserve and active partitions.

One possible improvement to the rsync approach is to cache the delta to be synced in some way, rather than rescanning the whole file tree on both client and server in order to sync the cache.

Mobility support

The idea of the “cached client” is that laptops are treated much like a connected system, and fit into the same framework. This extends beyond the OS install. For example, laptops should participate in LDAP and Kerberos infrastructure just as connected workstations do.

To this end, we have to extend all parts of the operating system to support disconnected operation. Some work we've been doing along these lines includes:

- Allowing login to a Kerberos-configured laptop, even when the network is disconnected
- Caching GECOS fields (from NIS or LDAP), so they can be accessed when disconnected
- Support for tracking a connected/disconnected state, using D-BUS notifications, so applications such as the web browser and mail client can automatically enter and leave “offline mode”

Live CD instantiation

With support for a read-only root filesystem and dynamic hardware detection, booting the OS from a CD does not require extensive hacks or modifications to the OS itself.

Given an OS install available for diskless or cached instantiation, we have preliminary support for “write this install to a CD.” This builds an ISO image, mounts it, modifies the image slightly as required, then burns the image to CD.

Future Directions

There are countless ways we can improve the operating system, building around the stateless Linux theme. Here are some examples.

Use a filesystem for cached instantiation

Our first prototype of cached instantiation uses two partitions to support a double-buffered rsync; we rsync the partition that's not in use. This was simple to implement and gets the job done.

A network filesystem that supported cached instantiation would do a better job. It could cache each file “on demand” when the file is used. While connected to the network, the system would never use an outdated file; it could be notified immediately whenever the cache should be expired.

However, to enable disconnection, the entire root filesystem has to be cached in advance. It would not work to have only the OpenOffice.org components you had happened to use prior to disconnection; all files in the OpenOffice.org package that you *might* use while traveling should be cached on the laptop.

Thus, a filesystem tuned for cached instantiation would have hooks to allow userspace to control its behavior (e.g. a “cache everything” command) and hooks to display feedback such as progress bars and connection errors.

User data storage

In an ideal stateless deployment, it's trivial to replace a crashed computer. Imagine this user experience:

- User plugs the replacement computer into the network; it network boots and asks the user to authenticate

- The user is then presented with a list of their computers, e.g. “Joe's Thinkpad,” “Joe's Workstation” and has the option to make the new computer a replacement for one of those or identically configured to one of those; also available might be some stock “templates” such as “standard laptop”
- On choosing “Joe's Workstation” the replacement workstation would instantiate the same OS install as the original workstation, and have the same data in the user's home directory as the original workstation

It would be almost as good even if only the sysadmin can do all this, rather than the user setting up their own new machine.

To get this working, we might imagine implementing:

- Track all the computers belonging to a particular user
- Each computer has some sort of identifiable name, such as “Joe's Workstation”
- Each user can have multiple home directories, if the home directories are local (e.g. for laptops)
- However, it's critical to back up the home directory often – perhaps whenever the laptop connects to the intranet
- These home directory backups have to be associated with a particular computer

Of course in the connected case, this is much simpler and works today: put a single home directory on a network file system and share it between all the user's computers.

Conclusion

Many aspects of the stateless Linux project aren't new. MIT's Athena, live CD projects such as Knoppix, LTSP, InterMezzo, Coda, and several Red Hat customer deployments demonstrate similar ideas.

However, stateless deployment models have always been an exercise in custom development, often requiring difficult-to-maintain package forks from the standard version of the operating system, or complex add-on features.

Some of our goals:

- To support a stateless model “out of the box,” designing the operating system and applications to work with it from both a technical and a user interface standpoint
- To build a uniform architecture where functionally equivalent elements can be mixed-and-matched; for example, diskless, cached, and live CD instantiation
- To extend the architecture to be general-purpose; for example ensuring that laptops fit in
- To work with all the important elements of a production deployment; for example authentication and directory services

Some of the cases we hope to enable:

- Live CDs
- Diskless thin clients
- Virtual servers sharing the same root filesystem
- Diskless blade servers
- “Cached client” laptops

Feedback and contributions are very welcome; please come join us on fedora-devel-list.