SUN BLUEPRINTS
[illegible]
APPROVED

Linux Overview for Solaris™ Users

John Cecere, SunServices

Sun BluePrints™ OnLine—August 2003

http://www.sun.com/blueprints

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
650 960-1300

Part No. 817-3341-10
Revision 1.0, 7/23/03
Edition: August 2003

Linux Overview for Solaris™ Users

This article provides a technical overview of the Linux operating environment and compares and contrasts it with the Solaris™ Operating Environment (Solaris OE). The purpose of this article is to quickly familiarize advanced system administrators with the Linux OE and to provide a reference for Solaris to Linux usage. This article is for readers who are experienced with the Solaris OE and are tasked with deploying, servicing, maintaining, and using Linux-based systems.

The Linux information in this article is based on Redhat Linux 8.0 running on Intel architecture hardware. Because the core of all Linux distributions is essentially the same, most of the material here can be applied to any distribution. It is presented as a comparison to Solaris on SPARC® hardware, although there are some references to Solaris x86.

This article contains the following topics:

- "Linux Origins" on page 2
- "Software" on page 2
- "Hardware" on page 7
- "Kernel" on page 9
- "Networking" on page 15
- "System" on page 18
- "About the Author" on page 25
- "Related Resources" on page 25
- "Ordering Sun Documents" on page 26
- "Accessing Sun Documentation Online" on page 26

Linux Origins

Linux is considered a UNIX® like operating system, primarily because no part of it was derived from the source tree of either AT&T or BSD UNIX. It originated as a project in 1991 by Linus Torvalds, then a college student at the University of Helsinki in Finland.

Linus' name is not pronounced "Ly-nus," but rather "Lee-nus." Consequently, Linux is pronounced "Lee-nucks," but in the U.S., it's more commonly pronounced "Lin-nucks," with a short "i" sound. Note that "Ly-nucks" is generally considered an incorrect pronunciation.

Linus wrote his kernel from scratch, using minix as the initial environment to develop his kernel. Minix is a free operating system used primarily as an educational tool to teach about operating systems. Linus soon expanded the kernel, posting the source code to the Internet and, before long, others began to add support for file systems and hardware.

Today, Linux has the same features as those found in other modern UNIX operating systems features such as priority scheduling, virtual memory, and multithreaded processing. It runs on 16 different architectures, including Intel, SPARC, and PowerPC. It has support for over 30 file system types, and has drivers for over 1800 hardware components. It uses software from the open-source GNU project as its operating environment.

Software

This section describes the Linux OS and GNU OE and compares them to the Solaris OE. Included are reference tables for commands and libraries.

This section contains the following topics:

- "GNU Software" on page 3
- "Distribution Methods" on page 3
- "System Installation" on page 4
- "Commands" on page 4
- "Libraries" on page 6
- "Documentation" on page 7

GNU Software

GNU (pronounced “guh-new”) is the name of a project started in 1984 by Richard Stallman of MIT to develop open-source software and an operating system to run it on. The name itself is a programmer’s joke and recursively stands for GNU’s Not Unix. Most GNU software can be compiled and installed on a variety of operating systems, including virtually all versions of UNIX. The GNU software actually covers a large part of what we call Linux today. Linux itself really only refers to the kernel, whereas GNU refers to most of the software that runs in the environment that the kernel provides. Virtually all of the GNU software can be built and installed in Solaris as well. Since the release of Solaris 8, Sun packages GNU software on the Software Companion CD. In addition, Sun is migrating its windowing environment from OpenWindows and CDE to GNOME, the GNU Network Object Model Environment. For more information on GNU, refer to the GNU Web site at: `http://www.gnu.org`.

Distribution Methods

As mentioned in the previous section, most of the software that runs in Linux environment comes from the GNU project and is open source. Open source means that the software is distributed in the form of source code and that the source code is freely available to everyone. However, for the software to be useful, it must be compiled into machine code. It would be very cumbersome to go about compiling *all* of the software required to build a Linux system. From the `ls` command to the C compiler itself (`gcc`), all the software must be built from the source code. This fact is why there are numerous Linux distributions. A Linux distribution is released in the form of binaries that are easily installed onto a computer.

For Redhat, Mandrake, and Sun Linux, the distribution method is rpm package files. Initially, rpm stood for Redhat Package Manager, but now uses the same pun as the GNU acronym and stands for RPM Package Manager. The rpm package format is similar in functionality to a Solaris package and uses the `rpm` command to install or remove a package. Like `pkgadd` and `pkgrm` in Solaris, the `rpm` command is capable of checking dependencies; running pre-install, post- install, and remove routines; and installing into an alternate root directory. Extensive information about the functionality of rpm is available on the `rpm`(8) manual page. Further information is at `http://www.rpm.org`.

System Installation

The system installation process for most Linux distributions has come a long way since the original kernel was written. My own personal experience of installing Linux for the first time was with Slackware Linux in 1994. It required that I download and create 50 floppy diskettes. Each one of the diskettes contained a single package. In addition, the kernel needed to be compiled from source, because the stock kernel that was included to bootstrap the system only contained basic hardware support. On a machine with an Intel 486DX-33 processor, this compilation process sometimes took several hours.

Today, the installation process for most distributions is very simple and GUI driven. The basic steps for installing Linux are similar to those for installing the Solaris OE: identify the system, select the disk to install to, and choose which locales and software packages to install. Setting the configuration for your keyboard, video card, and mouse for X Windows is required, but these tasks are fairly simple because the installation program is usually capable of probing the hardware to figure out what drivers it needs. All that is usually required from a user is to decide what resolution and color depth to run at. The installation program prompts for how to install the boot loader, LInux LOader (LILO), or GRand Unified Bootloader (GRUB).

Commands

In Solaris OE, `/bin` is symbolically linked to `/usr/bin`. In Linux, these directories are separate and a distinction is made between the system binaries that go in `/bin` and application binaries that go in `/usr/bin`. In Solaris OE, application binaries typically get installed in `/opt` or `/usr/local`. Both operating environments have administrative commands in `/sbin` and `/usr/sbin`.

Most of the standard UNIX commands in Linux share a commonality with Berkeley Standard Distribution (BSD) rather than AT&T System V Release 4 (SVR4). The command-line arguments are taken mostly from BSD. Other commands are completely unique to Linux, but have similar counterparts in Solaris. TABLE 1 lists some commonly used commands in Solaris, with their counterparts in Linux. This list is not comprehensive. For a comprehensive list, consult Bruce Hamilton's UNIX Rosetta Stone Web site at `http://bhami.com/rosetta.html`.

TABLE 1 Common Solaris Commands Mapped to Linux Commands

Solaris Command	Linux Command	Purpose
`df -k`	`df`	List file systems in allocation units of KBytes
`nawk`	`awk/gawk`	Pattern scanning and processing language
`ps -ef`	`ps -aux`	List process status for all processes running
`pkgadd`	`rpm -i`	Add a software package
`pkgrm`	`rpm -e`	Remove a software package
`gzcat file | tar -xvf -`	`tar -xzvf file`	Unbundle a compressed tar file
`cc`	`gcc`	C compiler
`xterm/dtterm`	`konsole/gnome-terminal`	GUI terminal program
`lpsched`	`lpd`	Printer daemon
`tip`	`minicom`	Serial port access program
`snoop`	`tcpdump`	Network packet sniffer
`patchadd`	`rpm -U`	Install a software patch/update
`priocntl -e`	`nice`	Start a process with a given priority
`priocntl -s`	`renice`	Change the priority of a running process
`prstat`	`top`	Actively report process statistics
`useradd`	`adduser`	Add a user account
`/sbin/swapadd`	`/sbin/swapon`	Enable swap devices

TABLE 1 Common Solaris Commands Mapped to Linux Commands *(Continued)*

Solaris Command	Linux Command	Purpose
`format`	`fdisk`	Manipulate disk partition tables
`find / -print | grep` *`xxx`*	`locate` *`xxx`*	Find a file
`mount -F` *`type`*	`mount -t` *`type`*	Mount a file system of type <`type`>
`inetd`	`xinetd`	Internet daemon
`installboot`	`lilo`	Install boot program
`swap -l`	`/sbin/swapon -s`	Display swap information
`who -r`	`runlevel`	Show run level
`truss`	`strace`	Trace a process

Libraries

As with binaries, there is a distinction between system libraries and application libraries in Linux. The system libraries are contained in `/lib` and the application libraries are contained in `/usr/lib`. Solaris does not make this distinction, because `/lib` is just a symbolic link to `/usr/lib`.

Both Solaris and Linux have a runtime linker responsible for linking executables to their shared library dependencies. The mechanism works very much the same way in both operating environments, as shown in TABLE 2.

TABLE 2 Linking Executables to Shared Libraries

Solaris	Linux	Function
`/usr/lib/ld.so.1`	`/lib/ld-linux.so.1`	Runtime linker
`/var/ld/ld.config` (32-bit) `/var/ld/64/ld.config` (64-bit)	`/etc/ld.so.conf`	Linker configuration file
`crle`	`ldconfig`	Configure runtime linker
`ldd`	`ldd`	Show library dependencies

Documentation

In Solaris, virtually all software is documented in manual pages. The same is not true of Linux. While most of the basic UNIX commands, system calls, libraries, and system configuration files are documented in manual pages, other commands and software are documented in HOWTO and README files, GUI-based help programs, and on the Internet. Additionally, many rpm packages install the source code documentation under `/usr/doc` and `/usr/share/doc`. These locations are a good place to find pointers to additional documentation on the Internet, because these locations usually contain references to web pages of the software components.

A repository of documentation called the Linux Documentation Project is available on the Internet. This documentation contains numerous resources as well as most documentation from the sources mentioned previously. The Linux Documentation Project is located at `http://www.tldp.org`.

Hardware

This section describes the Linux hardware support. This section contains the following topics:

- “Support” on page 7
- “Device Nodes” on page 8

Support

A vast number of hardware components are supported in Linux. However, support for newer hardware is not usually provided for some time. Unless the hardware vendor has written a Linux driver for their hardware, someone with programming skills must actually obtain the hardware and write a driver for it. This process usually takes some time after the hardware is released. A few hardware vendors release Linux drivers with their hardware, and the list of vendors that do this is growing as Linux becomes more widely used.

Device Nodes

Most versions of UNIX provide a mechanism to access hardware within the virtual file system structure. This access is normally achieved by creating device nodes on file systems that are linked to their corresponding drivers via major and minor numbers. Both Solaris and Linux employ this mechanism, but they have different approaches to it. Most Solaris device nodes are under a device path hierarchy in the `/devices` directory. These device nodes are then symbolically linked to the `/dev` directory so that somewhat meaningful names can be associated with them. Linux uses only a `/dev` directory and does not employ a device hierarchy. Most of the entries in `/dev` in Linux are actual device nodes. The symbolic links that do exist in `/dev` just point to real device nodes in `/dev`.

When you initially install Solaris, a package named `SUNWcsd` gets installed. This package contains the core Solaris device nodes in `/devices` and symbolic links in `/dev` required for the initial boot of Solaris after installation.

Beyond this, any device nodes and symbolic links that are required get created via `devfsadm`, which is run every time a `boot -r` is done. Device nodes in Solaris are created with the major number mapped to the appropriate driver in the file `/etc/name_to_major`. The driver is initially assigned its major number when `add_drv` is run for it. Redhat Linux has a package that contains device nodes (`dev-3.3.1-2` on my Redhat 8.0 machine), but this package contains the device nodes for all the hardware supported in the latest version of the kernel. The device nodes and their associated major and minor numbers in Linux are maintained as a fixed list at `http://www.lanana.org/docs/device-list/` and any updates to the list require registering them with the list's owner.

This fixed-list mechanism in Linux is in the process of being phased out. A new mechanism for maintaining the `/dev` directory in Linux uses a new virtual file system type called `devfs`. The use of `devfs` allows for hierarchical device trees and dynamically created device nodes, without the need to have a device node for every possible piece of hardware that might exist on a Linux machine. Also, it has the ability to dynamically assign major and minor numbers so that they don't have to be hard coded in the driver (or the list).

For Linux installations prior to kernel version 2.5, support for `devfs` must be compiled into the kernel and a user-space daemon must be downloaded and compiled to take advantage of it. Because this feature was integrated into the 2.5 development kernel, it will be integrated into the next stable kernel release, 2.6.

Kernel

This section describes the kernel and includes reference tables for kernel-related parallels between Linux and Solaris and file system types. This section contains the following topics:

- "Versions" on page 9
- "Configuring and Compiling" on page 10
- "Modules and Drivers" on page 11
- "File Systems" on page 13
- "Swap" on page 15

Versions

Kernel versions in Linux follow a scheme of *x.y.z*, where *x* is the major release number, *y* is the minor release number, and *z* is the feature or bug fix release level. The major release level is currently 2. An even number for *y* represents stable kernel releases where only bug fixes are applied as updates. An odd number for *y* represents a development release, where new features can be added as updates. For example 2.0, 2.2, and 2.4 are stable production kernels, whereas 2.1, 2.3, and 2.5 are development kernels. When a development kernel reaches a critical mass in new features, a new production version is released.

The *z* number is analogous to a Solaris kernel update patch (KUP) revision number. It represents the feature release level for development kernels, and the bug fix level for stable kernels.

The standard way to update your kernel in Redhat Linux is to use `rpm`(8) to install a new binary release of it. If you have a custom kernel, you can update it by applying a source-code patch, in which case you need to recompile the kernel.

The kernel source and patches are available from `http://www.kernel.org`.

Configuring and Compiling

With Solaris 9 OE, just plugging in a new hardware component may be all that is required to install it. More commonly, a `pkgadd` and reboot is done. It all depends on whether the driver is native in Solaris, and whether the hardware can be physically installed without bringing the system down.

The stock kernel installed by default in Redhat Linux can recognize and load drivers for a wide array of hardware, so it may be sufficient for most purposes. If either your hardware requirements go beyond what is in the stock kernel, or you want to remove unneeded drivers built in to the kernel, a kernel rebuild is required. The details of the process are beyond the scope of this article, but the general steps are as follows:

1. Install the kernel source tree in `/usr/src`.
2. Build the configuration in `/usr/src/linux` via `make config`, `make menuconfig`, or `make xconfig`.
3. Build the kernel dependencies via `make dep`.
4. Build the kernel itself via `make`.
5. Build all components you designated to run as modules in step 2 via `make modules`.
6. Make a backup copy of your current kernel, and put a corresponding entry for it in `/etc/lilo.conf` or `/boot/grub/grub.conf`.
7. Install the kernel via `make install` (installs the kernel into `/boot` and runs LILO).
8. Install your modules via `make install_modules` (installs modules into ``/lib/modules/‘uname -r‘``).
9. Reboot using the new kernel.

The configuration phase of building a new kernel (step 2 in the previous list) creates a file called `.config` in the main kernel source directory. It’s a good practice to save a copy of this file elsewhere after you get a good working configuration. If you find yourself upgrading to a new kernel with a new source tree, you can usually just copy this file into the new kernel directory.

More information on building custom kernels is on the Web at: `http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/html_single/Kernel-HOWTO.html`.

Modules and Drivers

The Solaris kernel is entirely modular. The kernel itself performs a core functionality with anything beyond that requiring it to dynamically load a chunk of code to perform the task. These chunks of code are in the form of dynamically linked loadable modules. The Linux kernel can be modular in this fashion, if it is built that way from source. Most drivers have the option of being modules, or being compiled into the kernel itself. However, if a driver is needed as a component required to access the root file system, then compiling it as a module requires you to use an initial RAM disk (`initrd`) as part of the boot process. The `initrd` allows you to have a root file system accessible, complete with `/lib/modules` directory, before the real root file system is mounted. Solaris avoids this situation by using a file-system-aware boot loader (`ufsboot`) that is capable of finding the required modules within a UNIX file system (UFS) before the root file system is actually mounted.

By default, the Solaris kernel searches for modules from the following four directories in this order:

- `/platform/‘uname -i‘/kernel`
- `/platform/‘uname -m‘/kernel`
- `/kernel`
- `/usr/kernel`

If you add a new module, you can load it into the kernel immediately by installing it into one of these directories, running `add_drv` to assign it a major number, and running `devfsadm` to create the device nodes needed to access the new hardware or protocol.

Linux modules are located in `/lib/modules/‘uname -r‘`. To add a module dynamically into a running kernel, compile it and install it into `/lib/modules/‘uname -r‘`. Then use the `depmod -a` command to map out any dependencies for the module. If `depmod` comes up with unresolved dependencies, it’s possible that you need to recompile the kernel to resolve the dependencies.

After running `depmod` successfully, use `modprobe` or `insmod` to load the module. The `modprobe` command checks the dependencies mapped out by `depmod` and automatically loads any modules deemed as dependencies. For example, if you have `sunrpc` and `nfsd` compiled as kernel modules and `modprobe` loads the `nfsd` module, the `sunrpc` module is automatically loaded first, because the networked file system (NFS) needs remote procedure call (RPC) to operate. The `insmod` command is the same as `modprobe`, except that it does not check for dependencies. If there is a dependency that is not satisfied, `insmod` fails and displays an error message about unresolved dependencies.

The module interface in Linux has a configuration file called `/etc/modules.conf`. In this file, you can create module aliases, set driver options, and execute commands to run before or after a module is loaded.

Both the `modprobe` and `depmod` commands reference this file. Driver parameters are set via an options statement for the module in the `/etc/modules.conf` file, similar to the way you set them in `/etc/system` or `kernel/drv/<module>.conf` in Solaris. Possible driver parameters for a module can be listed via `modinfo -p <modulename>`. If the module supports it, you can also set parameters via the `proc` file system interface, or the `sysctl` command, which uses the proc file system interface. Retrieving and setting module parameters using the `proc` interface is generally accomplished by manipulating the contents of files located in the `/proc` directory. TABLE 3 shows the module-related parallels between Solaris and Linux.

TABLE 3 Module-Related Parallels Between Solaris and Linux

Solaris	Linux	Purpose
`modinfo`	`lsmod`	List loaded modules
`modload`	`modprobe/insmod`	Load a module
`modunload`	`rmmod`	Unload a module
`add_drv`	`depmod*`	Install a new module
`ndd -get`	`modinfo -p` or `/proc` *`file system interface`*	Get module parameters
`/etc/system` or `kernel/drv/<module>.conf`	*`options`* `line for module in /etc/modules.conf` or `modprobe <`*`module`*`> symbol=value` or `insmod <module> symbol=value`	Set module parameters at module load time
`ndd -set`	`sysctl / proc` *`file system interface`*	Set module parameters on the fly
`/etc/driver_aliases`	*`alias`* `option in /etc/modules.conf`	Module aliases
`/etc/name_to_major`	Fixed list if not using `devfs`. See "Device Nodes" on page 8.	Module major numbers

* Not an exact comparison. Because major and minor numbers are predefined in Linux, the functionality that Solaris gets from `add_drv` is not needed in Linux. Running `depmod` is just the final step in adding a module to a system.

A STREAMS driver implementation called LiS is available in Linux. The implementation is built outside of the kernel source tree and runs as a loadable module. You can download LiS from `http://www.gcom.com/`.

File Systems

One of the advantages to an open-source operating environment like Linux is that when you want to get something to work, somebody has probably already written a driver for it. This includes file system drivers. For example, in Redhat 8.0, there are no less than 31 file system types supported. However, some of these file system types were ported for read-only access, with an experimental interface for write support. TABLE 4 lists some file system types available in Solaris and their counterparts in Linux.

TABLE 4 File System Types in Solaris and Linux

Solaris	Linux
`autofs`	`autofs`
`nfs`	`nfs`
`udfs`	`udfs`
`hsfs`	`iso9660`
`procfs`	`proc`
`tmpfs`	`tmpfs`
`ufs`	`ufs`
`pcfs`	`msdos/vfat`
`ext2fs*`	`ext2`

* A driver has been written at Sun, but is unsupported.

An installation of Linux initially boots to an environment that has different file system types mounted. TABLE 5 is a list of file systems you might find after installing.

TABLE 5 Linux File System Types

Mounted On	File System Type	Purpose
`/`	`ext2/ext3`	Standard Linux file system.
`/boot`	`ext2/ext3`	Standard Linux file system.
`/proc`	`proc`	Contains process information mapped out so that it can be accessed within a virtual file system framework. In Linux, `/proc` also provides an access mechanism to data contained within the kernel and kernel modules. This access mechanism is analogous to the `ndd` command in Solaris.
`/dev/pts`	`devpts`	Interface used to allocate pseudo `tty` devices.
`/dev/shm`	`tmpfs`	`tmpfs` interface for shared memory. A `tmpfs` in Linux works the same as it does in Solaris. The difference here is that the `tmpfs` is mounted on `/dev/shm`. This requirement is for glibc 2.2 and newer.

The complete list of Linux file system support taken from the `mount`(8) manual page is as follows: `adfs`, `affs`, `autofs`, `coda`, `coherent`, `cramfs`, `devpts`, `efs`, `ext`, `ext2`, `ext3`, `hfs`, `hpfs`, `iso9660`, `jfs`, `minix`, `msdos`, `ncpfs`, `nfs`, `ntfs`, `proc`, `qnx4`, `reiserfs`, `romfs`, `smbfs`, `sysv`, `tmpfs`, `udf`, `ufs`, `umsdos`, `vfat`, `xenix`, `xfs`, `xiafs`.

Swap

Swap space can be allocated as a file or a partition in Linux. Before swap space is used, it must be formatted via `mkswap`.

Caution – Be aware that Solaris x86 and Linux swap have the same partition types, and running `mkswap` on a Solaris x86 partition destroys it. See “Partition Tables” on page 18.

As in Solaris, any swap partitions defined in the file system table (`/etc/fstab` in Linux) are activated at boot time. In Linux, `/sbin/swapon -a` is the mechanism for doing this. You can remove a swap file or partition from use by the kernel with `swapoff`. Like Solaris, Linux users have all kinds of theories as to what size swap should be for a system, and this subject is not within the scope of this article. Like Solaris, Linux does not require the use of swap.

Networking

This section describes Linux networking. It contains the following topics:

- “Interface Configuration” on page 15
- “Startup Scripts” on page 16
- “NFS and the Automounter” on page 17
- “Troubleshooting” on page 17

Interface Configuration

Configuration of an interface from the command line is accomplished the same as in Solaris with the `ifconfig` command. If the network card is compiled as a module, then the module needs to be aliased to a standard interface name in `/etc/modules.conf`. Standard Ethernet interface names in Linux are `eth0`, `eth1`, etc. For example, a 3Com 3C905B 10/100 NIC that uses the 3c59x driver in Linux would have the following entry in the `/etc/modules.conf` file.

```
alias eth0 3c59x
```

This entry assigns this card as interface `eth0` when the 3c59x driver is loaded, then it is possible to up the interface via `ifconfig`.

Note – Because there is no native STREAMS support in Linux, there is no concept of plumbing the interface. Once the driver is loaded and associated to an interface name via an alias in the `/etc/modules.conf` file, an `ifconfig <interface> up` is all that is needed to activate it. By contrast in Solaris, an `ifconfig <interface> plumb` is necessary to allow the IP streams driver to use this interface before an `ifconfig <interface> up` is issued.

As in Solaris, `ifconfig` adds a route to the kernel routing table based on the IP address and subnet mask used on the command line. To add a route manually in Linux, use the `route` command. The command-line options are slightly different than those in Solaris. The following are examples of adding a default route in Solaris versus adding the same default route in Linux.

Solaris example:

```
route add default 192.168.3.254 1
```

Linux example:

```
route add -net default gw 192.168.3.254 metric 1
```

Startup Scripts

The `rc` scripts to bring up networking are entirely different in Solaris and Linux, but they achieve similar results.

- Solaris uses `/etc/init.d/network` and `/etc/init.d/inetinit` to bring up the necessary interfaces, routes, and protocols.
- Linux uses `/etc/init.d/network` to bring up the necessary interfaces, routes, and protocols, but this script calls other scripts from the directory `/etc/sysconfig/network-scripts`. The actual network configuration for a Linux machine is defined in the file `/etc/sysconfig/network` as a series of environmental variables. This file is sourced from all the network configuration scripts. It can be manipulated manually, but the program `netconfig` is normally used to make modifications to it.

NFS and the Automounter

The NFS implementation in Linux has recently started providing NFS version 3 support, allowing for the use of TCP and larger block sizes. Support for NFS version 3 is achieved through the kernel configuration process. The `nfsd` server daemon is implemented similarly to Solaris `nfsd` in that it runs multi-threaded. Eight threads are started by default, but this number can be tweaked by modifying the startup script, `/etc/init.d/nfs`. This script starts both the client and the server processes for NFS. The authentication daemon `rpc.mountd` uses `/etc/exports` as its configuration file rather than `/etc/dfs/dfstab`. The syntax of `/etc/exports` is considerably different than `/etc/dfs/dfstab` in Solaris. Refer to the `exports`(5) manual page for more information.

In Linux, there are two different automounters. The first, `autofs`, was developed in Linux and is the default used for most Linux distributions. The second automounter, `amd`, came from BSD UNIX but has been ported to many operating systems, including Linux. They are both somewhat different in implementation than the Solaris automounter, but serve the same purposes. In addition to NFS, Linux `autofs` is capable of providing access to other types of network file systems and local hardware, such as diskette and CD-ROM drives. In this latter capacity, it serves a purpose similar to that of `vold` in Solaris. For useful configuration examples relevant to those familiar with the Solaris automounter, refer to the Web site at `http://www.linux-consulting.com/Amd_AutoFS/autofs-5.html`.

Troubleshooting

The process of network troubleshooting in Linux is not much different than in Solaris. The `netstat`, `rpcinfo`, and `nfsstat` commands all do essentially the same thing. The major difference is that instead of `snoop`, the tool to use for packet sniffing is called `tcpdump`. The `tcpdump` command gives similar information as `snoop`, but with different command-line options and output formats. For example, the equivalent to `snoop -d <interface>` would be `tcpdump -i <interface>`. The full list of command-line options for `tcpdump`(8) can be found in the manual page. For the latest version and documentation, `tcpdump` is maintained at the following Web site: `http://www.tcpdump.org`.

System

This section describes and compares the Linux system components with Solaris OE. It includes reference tables for standard file system types, boot processes, `init` run levels, and system files.

This section contains the following topics:

- "Partition Tables" on page 18
- "Boot Loader Programs" on page 19
- "Standard File Systems" on page 20
- "Boot Process" on page 21
- "`init`" on page 22
- "User Environment" on page 23
- "System and Log Files" on page 24

Partition Tables

On Intel machines, disks are divided into units called *sectors*. (1 sector = 512 bytes.) Solaris uses the term *block*, but the size is the same. (1 block = 512 bytes.) The first sector of the disk on an Intel machine is called the Master Boot Record (MBR). The first block of a Solaris disk is called the Volume Table of Contents (VTOC). The information contained in the two is similar. They both contain the partition table for the disk. However, the Intel MBR contains a 446-byte area at the beginning for actual boot code, whereas the first block on a Solaris disk is used only for the VTOC. What is referred to as the boot block in Solaris starts immediately after the VTOC and occupies blocks 1-15 of the disk.

The Solaris VTOC has certain tags that you can apply to a partition. These tags refer to a partition's function within Solaris, such as `root`, `swap`, or `var`. A disk with an `fdisk` label on it has tags to identify the partitions, but they usually refer to the type of operating system that resides on that partition. The tag is 1 byte in the partition table and is called the partition type. A 32-bit Windows FAT file system using LBA, an NTFS file system, and a Linux `ext2` file system have partition types of 0x0c, 0x07, and 0x83 respectively. For a full list, run `fdisk` in Linux and type l.

Be aware that there was never a standards body or entity to assign specific values to partition types. This lack of standardization has led to some conflicts. Most notably, the conflict between a Solaris x86 partition and a Linux swap partition. Both of these have a value of 0x82. This situation becomes problematic when you want both of these operating systems to reside on the same machine. If the Linux installation program formats a Solaris x86 partition for use as swap space, the Solaris partition is completely destroyed.

Boot Loader Programs

As previously mentioned, the `fdisk` MBR has a 446-byte area for some boot code. On a normal Windows system, this area contains the DOS boot sector. When Linux is installed, there is an option to load a program called LInux LOader (LILO) in this area. This name is probably a misnomer because LILO can load more than just Linux. It can be configured to boot most other operating systems that you can load onto an Intel machine. There is no real parallel to LILO in Solaris. The need for LILO arose out of the fact that most Intel machines have a BIOS (PROM) that is not capable of recognizing the partition structure of a disk. BIOS cannot boot off of anything other than the MBR at the beginning of the primary disk. By comparison, OBP on SPARC machines has the ability to specify partitions and disks to boot from, as well as set hardware aliases and boot parameters, and save them to NVRAM.

A new boot loader called the GRand Unified Bootloader (GRUB) came out with RedHat Linux 7.3. The GRUB is a file-system-aware bootloader that is more feature rich than LILO. Both LILO and GRUB use configuration files that exist on a file system somewhere to identify bootable partitions and give you a command line or menu to boot them from. For LILO, the configuration file is `/etc/lilo.conf`. For GRUB, it's `/boot/grub/grub.conf`.

Standard File Systems

In Solaris, UFS is used as the predominant file system type. In Linux, it's either ext2 or the journaling version, ext3. UFS grew out of the Berkeley fast-fat file system developed for BSD UNIX by Kirk McCusick. The ext2 file system was the second version of the ext file system, which was created from scratch specifically for Linux. Even though it was created independently from UFS, most of the concepts within ext2 are the same. TABLE 6 provides comparisons between the two.

TABLE 6 Comparison of ufs and ext File System Types

Attribute	Solaris	Linux
File system name	`ufs`	`ext2`
Method of journaling	with logging mount option	Mount as `ext3`
Default block size	8K	4K
Default fragment size	1K	4K
Organization	inodes and data blocks arranged in cylinder groups	inodes and data blocks arranged in cylinder groups
Created with	`mkfs/newfs`	`mke2fs`
Checked with	`fsck`	`e2fsck`
Backed up with	`ufsdump`	`dump`
Restored with	`ufsrestore`	`restore`
Tuned with	`tunefs`	`tune2fs`

Two tunable parameters that you might want to modify after creating an ext2 or ext3 file system are the maximal mount count and the check interval. The maximal mount count is a predetermined number of times an ext2 or ext3 file system can be mounted before a check is forced on it. The check interval is a predetermined time period before a check is forced on an ext2 or ext3 file system. This forced check happens regardless of the state of the clean flag in the superblock; that is, it ignores the fact that the file system was unmounted cleanly the last time it was used. The effect is that clean file systems get checked periodically. Because an fsck can add a lot of time to the boot process, the value of fsck might be negligible. To disable this behavior, use `tune2fs -c 0 -i 0 <partition>` on the partition that contains your file system.

Boot Process

Linux has a `/boot` directory that contains files (including the kernel itself) needed to boot. On pre-1999 Intel machines, it was necessary to have the kernel inside an 8 gigabyte boundary at the beginning of the disk. One of the ways to accomplish this configuration was by making a separate `/boot` file system and physically locating it at the beginning of the disk, or at least before the 8 gigabyte boundary. Because of a feature in LILO called 32-bit logical block addressing (`lba32`) that takes advantage of BIOS functionality found in newer machines, this configuration is no longer necessary. TABLE 7 summarizes the files used by Linux to boot.

TABLE 7 Linux Files Used to Boot a System

File	Purpose
`boot.xxyy`	Copy of the boot sector that LILO makes the first time it runs. Used if LILO is uninstalled via `lilo -u`. Where *xx*=major number of disk and *yy*=minor number of disk.
`boot.b`	Second stage boot loader. Loads the chain loader, kernel, or partition boot sector of a foreign OS.
`chain.b`	Chain loader required to boot a partition that is not on the primary disk.
`map`	File created by LILO that contains the names and physical locations of all the kernels and foreign operating systems that it can boot (as defined in `/etc/lilo.conf`).
`vmlinuz`	Compressed kernel.
`System.map`	The text version of the kernel-symbol table generated by running `/usr/bin/nm` against the uncompressed kernel image during the kernel build process. The Solaris equivalent would be to run `/usr/ccs/bin/nm` against `/dev/ksyms` or `unix.0` from a kernel core dump. `System.map` is used by some programs, such as `klogd` and `ps`, that provide more useful output with symbol names rather than with addresses.
`initrd`	Initial RAM disk. The `initrd` allows you to use a RAM disk as the root file system, usually temporarily. This technique gives the ability to load modules required to access a real root file system and not have to build these modules into the kernel itself. After the real root file system is mounted, the context is switched via the Linux system call `pivot_root`(2).

As in Solaris, the boot process in Linux goes through several stages. TABLE 8 shows parallels between the two processes, without overly generalizing. The process outlined in this table assumes that LILO is installed in the MBR on the boot disk.

TABLE 8 Boot Process in Solaris and Linux

Solaris	Linux
OBP selects defined boot device and reads the boot block (sectors 1-15) of this partition.	BIOS reads MBR and loads first-stage boot loader.
Program read in from boot block loads secondary boot loader (`ufsboot`).	First-stage boot loader finds second-stage boot loader (`/boot/boot.b`) and executes it.
`ufsboot` finds, loads, then executes the kernel.	`/boot/boot.b` reads in descriptor table from `/boot/map`. The map file contains the boot sector of the kernel to be loaded. The second-stage boot loader then loads the kernel.
Kernel mounts root file system, loads modules, and starts `init`.	Kernel mounts root file system, loads modules and starts `init`.*
`init` starts the appropriate `run level` script(s).	`init` starts `/etc/rc.d/rc.sysinit`, then starts the appropriate run level script.

* If `initrd` is required, it is mounted before the real root file system.

`init`

In Linux, `init` works essentially the same as it does in Solaris, although the run levels and scripts are somewhat different. In Solaris, the scripts in `/etc/rcS.d` are always run initially, regardless of what run level the system is booting. Linux does not have a `/etc/rcS.d` directory, but it has a script, `/etc/rc.d/rc.sysinit`, that accomplishes the same purpose. The run level script directories in Linux are in `/etc/rc.d`, but the directories contained therein are symbolically linked to `/etc`, so their location is effectively the same as in Solaris. One other difference is that run level scripts in Linux are symbolically linked from `/etc/init.d` to the run level directories, whereas they are typically hard linked in Solaris.

One thing to note for both Solaris and Linux is the distinction between run level S and run level 1. Both operating environments handle them similarly. Run level S is more of a run level to initially boot to, rather than issue an `init` command to take you to it. Run level 1 is more appropriate to use if you are already at a higher run level and want to bring the machine down to a maintenance state. This is mainly because kill scripts tend to get installed in `/etc/rc1.d` rather than `/etc/rcS.d`.

To boot single-user mode in Solaris, issue a `boot -s` command from the ok prompt in Open Boot PROM. In Linux, boot single-user mode by issuing a `boot single` command from the LILO prompt or add `single` to the list of kernel parameters in GRUB.

If the graphical login option is chosen during the installation process, the default run level is 5. Otherwise, it is set to 3.

TABLE 9 shows the run levels for Solaris and Linux.

TABLE 9 Run Levels for Solaris and Linux

Function	Solaris Run Level	Linux Run Level
Shutdown	0	0*
Single user/Maintenance mode	S/1	S/1
Multi-user mode, no NFS	2	2
Multi-user mode	3	3
Multi-user mode with graphical interface	3	5
Shutdown with power off	5	0*
Reboot	6	6

* `init 0` performs a power off in `runlevel 0` if the kernel is configured for power management.

User Environment

After Linux completes the boot process, either a character terminal (`runlevel 3`) or graphical login screen (`runlevel 5`) is present. In the character-terminal environment, by default there are six virtual consoles that can be used.

The virtual consoles can be accessed by pressing Alt-F1 through Alt-F6 on a keyboard. If a graphical environment is used, the virtual consoles of the character-terminal environment can be accessed by pressing Control-Alt-F1 through Control-Alt-F6. The use of the control key is only necessary to switch from graphical mode to character-terminal mode. To switch back to graphical mode, press Alt-F7.

Two main graphical environments exist in most current versions of Linux: KDE and GNOME. The use of either is a personal preference by the user. Both are similar in functionality.

By default, the root account uses bash as its shell and `/root` as its home directory.

System and Log Files

Many system configuration and log files have the same names and uses in Solaris and Linux. TABLE 10 lists some that are not the same. This list is not all-inclusive. The functionality between the files listed is similar, but the syntax and structure can be different. In some cases, a directory of configuration files is used in Linux where only a single file is used in Solaris. For a more extensive list, refer to the Web site `http://bhami.com/rosetta.html`.

TABLE 10 Differing System and Log Files in Solaris and Linux

Solaris	Linux	Purpose
`/etc/dfs/dfstab`	`/etc/exports`	NFS server configuration
`/etc/auto_master, /etc/auto_home`	`/etc/auto.master, /etc/auto.home`	automounter configuration
`/var/spool/cron/crontabs/root`	`/etc/crontab`	`cron`
`/etc/vfstab`	`/etc/fstab`	File system table
`/etc/mnttab`	`/etc/mtab`	Kernel list of currently mounted file systems
`/etc/inetd.conf`	`/etc/xinetd.conf, \` `/etc/xinetd.d (directory)`	Internet daemon configuration
`/etc/defaultdomain, \` `/var/yp/binding/‘domainname‘\` `/ypservers /`	`/etc/yp.conf`	YP configuration
`/var/adm/messages`	`/var/log/messages`	Sytem log file
`/etc/ftpusers (Solaris 8) \` `/etc/ftpd/ftpusers (Solaris 9)`	`/etc/vsftpd.ftpusers(vsftpd) \` `/etc/ftpusers (wu-ftpd)`	FTP daemon user configuration (`ftpusers`)
`/etc/system`	`/etc/sysctl.conf`	Kernel configuration
`/etc/pam.conf`	`/etc/pam.d (directory)`	Pluggable Authentication Module (PAM) configuration

About the Author

John Cecere is a Regional System Support Engineer for Sun Services and has been with Sun for six years. Prior to that, he spent five years as a system administrator of SunOS 4.x, Solaris, and HP/UX servers for Bellcore. He also spent several years programming C on machines running AT&T SVR3 UNIX.

His experience with Linux dates back to 1994 when he first loaded kernel version 1.1.39 on his PC. He has been happily geeking with Linux ever since.

Related Resources

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other material on or available from such sites or resources. Sun will not be responsible or liable for any damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through any such sites or resources.

- Building custom kernels: `http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/html_single/Kernel-HOWTO.html`
- GNU software: `http://www.gnu.org`
- Kernel source code: `http://www.kernel.org`
- LILO README: `/usr/share/doc/lilo-x.y.z/README`
- Linux consulting: `http://www.linux-consulting.com/Amd_AutoFS/autofs-5.html`
- Linux STREAMS (LiS): `http://www.gcom.com/home/linux/lis/`
- Official device node list: `http://www.lanana.org/docs/device-list/`
- RPM Package Manager: `http://www.rpm.org`
- `tcpdump`: `http://www.tcpdump.org`
- The Linux Documentation Project: `http://www.tldp.org`
- The UNIX Rosetta Stone: `http://bhami.com/rosetta.html`

Ordering Sun Documents

The SunDocs^{SM} program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The docs.sun.com web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is http://docs.sun.com/

To reference Sun BluePrints™ OnLine articles, visit the Sun BluePrints OnLine Web site at: `http://www.sun.com/blueprints/online.html`.