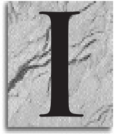
PART I

Oracle on Linux Overview

CHAPTER 1

Linux Architecture



n order to properly set up, configure, and administer Oracle on Linux, you should have some idea of how the Linux operating system works, how it is architected, and how to configure it. Administering the Linux system and configuring options varies, both by vendor and by version of the operating system. In this chapter, you will learn about the Linux kernel, how it works, and how it is configured. Later, in Chapter 4, you will learn more about how to tune specific parameters.

By understanding the architecture of Linux, you will be better equipped to monitor, configure, and tune the operating system. In addition, you will better understand how the operating system and Oracle work together (as well as against each other sometimes). In Chapter 2, you will learn about how Oracle is architected and specifics of what parts of the operating system Oracle depends on. This chapter will begin with the Linux kernel, the I/O subsystem, processes, and threads. Later in this chapter, you will learn about devices and filesystems and how to rebuild the Linux kernel.

Unlike proprietary UNIX operating systems of the past, where you would get your UNIX operating system from your hardware vendor, there are a number of different distributions of Linux. Although Linux source code is freely available on the Internet, most companies that are deploying Linux with Oracle demand a higher level of support than can be achieved by downloading source code off of the Internet and compiling it yourself. Once Linux started becoming more popular, companies such as Red Hat and SuSe began commercializing Linux. In order for Linux to be adopted by commercial users, things like pre-packaged software and support became more of an issue.

Two of the most popular Linux offerings are from Red Hat and SuSe. Red Hat was founded in 1993 and since then has grown to be one of the most dominant Linux vendors. SuSe was founded in 1992 and has recently been acquired by Novell. Both of these companies offer enhancements to the Linux operating system that make it more robust and more suited for Enterprise applications and offer support programs which are a necessity for Enterprise customers.

Both Red Hat Enterprise Linux and SuSe Enterprise Server support features that assist the Enterprise customer with the smooth operation of their system, including the following:

- Kernel upgrade procedures that allow for patching of kernels without having to recompile the entire operating system
- Supported configurations and kernels that are tested and validated
- A guaranteed upgrade path

Obtaining your Linux distribution from a reliable vendor that provides world-class support has many advantages. Many vendors that provide and support Linux are very good. Because of the dominance of the Linux market by Red Hat and SuSe, this book focuses on these two vendors.

Operating System Overview

An operating system exists to allow the user to run programs. The Operating System consists of a number of different types of programs and layers that handle different types of functions. As with most terminology, the initial architects of the early operating systems used analogies in order to illustrate their point. The innermost core of the operating system became known as the kernel, and the outer layers of the operating system were appropriately labeled the shell. The shell is used by the user to interact with the operating system, and between the shell and the kernel lie the system utilities.

The kernel is responsible for operating systems services such as processes management, scheduling time on the CPU, memory management, and system calls. System calls allow the shell to communicate with the kernel and perform functions like accessing disks.

Several different types of kernels are very popular today, some of which may be familiar to you. The first of these is known as a monolithic kernel (shown in Figure 1-1), and is made up of only three layers.

The second type of kernel that is popular uses a microkernel. The microkernel architecture (shown in Figure 1-2) can have many more layers than the monolithic architecture.

The Monolithic Kernel

A monolithic kernel is the simplest kind of kernel, and is created as one single executable, where each part of the kernel accesses the same memory structures. Monolithic kernels have a long tradition of use in operating systems such as UNIX, Windows, and DOS. Though some people believe the trend in more modern operating systems is not to use a monolithic design, Linux does use a monolithic kernel, and integrates its advantages without its traditional disadvantages.

The advantage of the monolithic kernel is that since it is one executable, all of the memory is available everywhere within the kernel. This means that data does not need to be passed between different layers of the operating system, thus improving

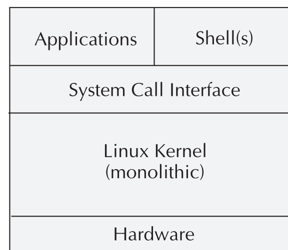


FIGURE 1-1. *The monolithic kernel model*

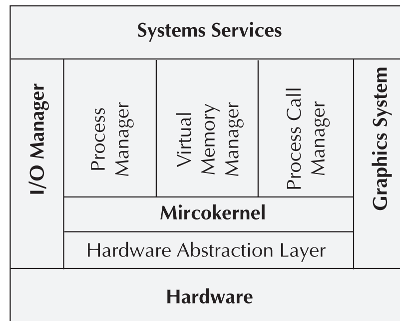


FIGURE 1-2. *The microkernel model*

performance. Traditionally, the disadvantage of monolithic kernels is that whenever you want to add another device driver or kernel addition, the entire kernel must be rebuilt. This can be quite painful, especially for novices.

Unlike traditional monolithic operating systems, Linux uses a hybrid monolithic kernel that supports Kernel Loadable Modules, known as KLMs. Kernel Loadable Modules allow for device drivers and kernel layers to be dynamically loaded and unloaded from the kernel. This offers several advantages. Loadable modules only utilize memory when they are used. Loadable modules are used for several areas of the operating system, including

- **Device drivers** These are programs designed to communicate with and operate a specific hardware device.
- **Filesystem drivers** Filesystem drivers allow many different filesystems to be used in the operating system. Unused filesystems are not loaded into the kernel.
- **System calls** System calls are communication channels between the users and the kernel. By permitting loadable system calls, custom-made system calls can be developed and added into the kernel.
- **Network drivers** Loadable network drivers allow for the addition of new and custom network protocols.
- **Executable interpreters** Various executable interpreters can be loaded and unloaded from the kernel as necessary.

By supporting loadable modules, the kernel has a level of functionality that is available in microkernels while offering the features of a monolithic kernel. There are several commands that can be used to list, view, and manage KLMs. These commands are listed here:

Command	Description
lsmod	Used to list the modules currently loaded in the kernel
insmod	Used to install a KLM (load a module)
modprobe	Used to manage KLMs
rmmod	Used to remove KLMs from the kernel
depmod	Used to handle module dependencies
ksyms	Used to display kernel symbols and which modules they belong to

Loadable modules can be quite useful and convenient, since kernel recompiles are not required to add modules.

The Microkernel

In a microkernel architecture, a very small kernel interacts with the hardware and is surrounded by layers of operating system code that perform different functions. The microkernel performs kernel functions such as thread management and some memory management in interprocess communication, but many other OS services (such as I/O processing) run outside of the kernel. This is also true of device drivers, which can be used even though they haven't been linked into the kernel. These device drivers just need to be able to communicate with the other OS functions as well as the kernel.

Many modern operating systems use a microkernel model, including Microsoft Windows server products like Windows NT, Windows 2000, and Windows 2003. Other operating systems that use the microkernel architecture include AIX and Mach.

Linux Overview

The Linux operating system consists of a hybrid monolithic kernel (as mentioned earlier), system calls and utilities, and shells used to access the OS. As with most UNIX-like operating systems, with the exception of the kernel, you have your choice regarding use of utilities and shells. In fact, most Linux varieties come with at least three shell programs you can choose from. The most popular are sh (the Borne shell), ksh (the Korn shell), and bash (the Borne-Again shell).

In addition, the Linux operating system comes with a huge variety of utilities and applications that can assist you with data manipulation, word processing, computing, and other tasks. The shell, like most Linux programs, can be used in both an interactive mode or in a batch mode. Creating shell programs, or scripts, is a great way to perform maintenance tasks over and over again. A good rule of thumb is that if you are going to do something more than once, make a shell script out of it. They can be very powerful and perform a number of tasks, thus saving you from having to write a compiled program.

In this section, we will explore some of the Linux utilities provided with your Linux distribution (as well as the development environment), the Linux architecture, and how to configure Linux. Later in this chapter, we will explore the Linux kernel configuration, how to rebuild a Linux kernel, and also discuss the boot process.

The Linux Directory Structure

The Linux operating system directory structure uses a tree-like structure. The top level of this tree structure is known as the root directory. From the root, the directory structure branches out into different branches, which in turn branch out into other branches. From the top level, there are a number of core directories that contain configuration information, programs and utilities, user data, and system data. The main directory branches are shown next.

Directory	Usage
bin	The bin directory contains programs or binaries used by the operating system.
boot	The boot directory is used to boot the system.
dev	This directory contains the device nodes or files that link to hardware devices.
etc	The etc directory contains system configuration information.
home	This directory contains user home directories.
lib	The lib directory contains libraries needed by the operating system.
mnt	This directory is used to temporarily mount other filesystems such as CD-ROMs and floppies.
proc	This pseudo filesystem contains Linux configuration and operational information. The pseudo filesystem looks like a filesystem, but is actually a view into the running kernel. The /proc filesystem is used to configure the kernel as well as to get information from the kernel.
sbin	The sbin tree is used for system binaries used during the system startup process.
tmp	The tmp directory tree contains temporary files.
user	The user tree contains user programs and utilities and includes subdirectories such as /usr/bin, /usr/sbin, and /usr/local/bin.
var	The var directory tree contains information that varies as the system runs, such as log files and print spool files.

These directories are crucial to the operation of the Linux operating system. However, they do not necessarily exist within the same filesystem. It is your choice when configuring the system whether directory trees such as /home, /usr, and others are part of the / filesystem or whether they are a mounted filesystem. You will learn more about this later in the chapter when filesystems are covered.

Linux Utilities and Directories

Linux operating systems include a wealth of utilities. Even the most experienced UNIX and Linux guru most likely has not used all of them. These utilities and applications provided with Linux perform a number of different tasks and can be used standalone or with other applications. In this section, you will learn about what types of utilities are provided within the Linux operating system and what they can be used for.

The Linux utilities can be broken down into a number of different areas that perform various general functions. At the highest level, these utilities can be divided into administrative (system) utilities and user utilities. The system utilities can typically be found in the directory /sbin, /usr/sbin, /bin, and /usr/bin. An overview of utilities is described next in terms of this directory structure.

Utilities in /sbin

The utilities in the /sbin directory are reserved for those utilities required for booting, as well as the core functionalities necessary to run Linux itself. There are over 250 utilities in this directory. They provide a wide variety of support functionality relating to the I/O subsystem, filesystem, and network. A short sample of these utilities is given next.

Filesystem and I/O Utilities The following utilities are used in configuring, querying, and starting up the I/O subsystem and filesystems. These utilities exist in the /sbin directory. This list is only a sample and is not all-inclusive.

Utility	Function
badblocks	Used to find bad blocks on a device
bevlabel	Keeps symbolic links consistent even if device names change; used in conjunction with Oracle RAC and RAW devices
blvtune	Used for elevator sorting; an important filesystem tuning tool
bdisk	Formats and partitions a disk drive
bsck	Filesystem checker
bdparm	Gets hard-disk parameters
bwapon/swapoff	Enables and disables swapping on a device

Network Utilities The following utilities are used in configuring, querying, and starting up the network. These utilities exist in the /sbin directory. Again, this list is only a sample and is not all-inclusive.

Utility	Function
arp	Address Resolution Protocol
ethtool	Used to manage and monitor the network adapters at the hardware layer
ifconfig	Used to configure and manage network adapters
route	Manipulates IP routing tables

General OS Utilities The following utilities are used in running the operating system. These utilities exist in the /sbin directory. This list is only a sample and is not all-inclusive.

Utility	Function
date	Used to print or set the system time
clock	Determines processor time
chkconfig	Query and update system services runlevel information
grub	The bootloader
init	Starts and controls system processes
lsmod	List loaded modules
poweroff	Shuts down and powers off the system
service	Starts a service

As you can see, there are a number of very critical services used during the system startup process that exist in the /sbin directory. This directory is critical during the boot process and is part of the / filesystem. This is used as part of the boot process before the /usr filesystem is mounted (if /usr is a separate filesystem).

Utilities and Daemons in /usr/sbin

The utilities in /usr/sbin, though system utilities, are not necessarily needed to boot the system. They contain administrative tools used for the system to function once it is

up and running. There are over 300 utilities in the /usr/sbin directory. A sample of some of these tools is given here.

Utility	Function
useradd	Used to add new user accounts to the system
groupadd	Used to add groups to the system
lpadmin	Used to administer the line printer service

In addition, the /usr/sbin directory is where many of the daemon executables are found. Daemons are programs that perform services for the system. Some daemons run in the background all the time, while other daemons are run only when needed. Daemons are very important to the operation of the system. Some of the most important ones that reside in /usr/sbin are listed next.

Daemon	Function
atd	The atd runs jobs that have been submitted to run at a later date (a scheduled time).
crond	The crond runs regularly scheduled jobs submitted by cron. These might be regular system or user programs that run every hour, every day, every week, and so on.
smbd	The Samba daemon serves SMB/CIFS services to clients.
smtpd	The smtpd is the simple mail transfer protocol daemon.
sshd	This is the secure shell server. SSH clients connect through this daemon.
tcpd	This is a network server for TCP traffic.

As you can see, the /usr/sbin directory contains many utilities used by the system and by the system administrator. This directory is very important to the operation of the system, in conjunction with the system utilities in the /sbin directory.

Utilities in /bin

The utilities in the /bin directory are similar to /sbin in that they are mostly administrative in nature; however, the /bin utilities might be used by regular users as well, hence they are separated from /sbin. The reason these files are in /bin rather than /usr/bin is that it is possible that some of these utilities might be needed by the system administrator

during boot, thus they will be available before /usr is mounted. Some of the utilities in /bin include the Linux shell programs, as listed next.

Shell	Description
bash	The GNU Borne-Again shell. This is the newest shell, incorporating features of both sh and ksh. Bash is the default shell program.
csh	csh was the first shell program that provided the ability to create shell programs resembling their C language cousins. It therefore allowed shell programmers to create sophisticated shell programs.
ksh	The Korn shell is a more sophisticated version of the basic Borne shell. You can create complicated shell programs using ksh.
sh	sh is the original Borne shell. This shell provides basic shell functionality as well as basic shell programming abilities.
tcsh	The tcsh is an enhanced csh that allows for command-line editing and completion.
zsh	zsh is a new shell that provides many of the features of ksh but has been enhanced to include spell checking, history, and so on.

As you have heard before, Linux and UNIX are all about choices. You are free to use whichever shell you prefer.

Utilities in /usr/bin

The utilities in /usr/bin are the utilities most likely to be used by the end user. These utilities are used for browsing directories, editing files, sending e-mail, and performing other tasks. The /usr/bin directory contains almost 1500 utilities that can be used by both the administrator and end user. These utilities also include many of the development environment utilities mentioned in the next section. A sample of some of the files in /usr/bin are shown next.

Utility	Function
at	Used to run programs at a time in the future
awk	A pattern scanning and processing language
compress	Used to compress and expand files
cut	Prints selected parts of a file
diff	Compares files and prints differences
df	Displays available disk space
ftp	File Transport Protocol tool
ghostscript	Postscript and PDF interpreter

Utility	Function
gzip	Used to compress and decompress files
lLess	Used to view the contents of files
lpr	Lineprinter interface; used to print files
mysql	The optional mysql database
perl	A sophisticated programming language interpreter
rsh	The remote shell processor
ssh	The secure shell processor
sudo	Used to run a program as root
tail	Reads from the end of a file
top	Displays top CPU processes
who	Displays a list of users on the system
zip	Used to compress and uncompress files

I have not covered all of the utilities in detail because there are so many of them. As you become more familiar with Linux, you will become comfortable with a subset of these utilities that suit your needs and experience. You will find that you can use different utilities and applications to perform the same tasks with different levels of complexity and functionality. For example, you can write shell programs with any of the shells mentioned in the previous section, but you can also create sophisticated scripted programs using PERL. PERL is a sophisticated language that can easily call other applications, such as SQL*Plus.

The Development Environment

The Linux development environment is quite sophisticated and full featured. It's mostly made up of the GNU compilers and libraries, as well as a wealth of system routines and libraries. The development environment supports C++, C, and assembly language programs, and thus it's heavily focused on C++ and C. Because Linux source code is written in C and is readily available for developers to modify and recompile, it is crucial that a robust C development environment be available within the Linux operating system.

The Linux development environment consists of the GNU compiler collection, the debugging tools, and the associated utilities, including

- The GNU compiler collection (gcc)
  - cpp, the C preprocessor
  - as, the assembler
  - ld, the linker

- The GNU Debugger (gdb)
- The binutils, a set of binary utilities used by developers:
  - ar, the archiver—creates, modifies and extracts files from code archives
  - nm—used to list symbols within object files
  - objcopy—used to copy and convert object files
  - objdump—used to display the contents of a binary file
  - ranlib—generates index to archive contents
  - size—lists section and total sizes of object files
  - strings—lists printable strings within files
  - strip—discards symbols from files
  - readelf—displays information about ELF format files

The development environment is very important to the Linux system, just as it is to UNIX systems. Linux is a dynamically configurable system which does not require the kernel to be relinked; however, Oracle does require a relink as part of its installation and configuration. It is necessary for the linker to be installed in order for Oracle to be installed.

The development environment available within Linux is similar to that offered on many UNIX platforms. In fact, the GNU compiler collection is available on many UNIX platforms. However, this development environment is very different from the development environment you would find under Windows. It is also unlike the graphical development environment you would get from Oracle, such as with JDeveloper.

Let's look at some of the components of the Linux development environment.

GCC, the GNU Compiler Collection

The GNU compiler collection consists of the C preprocessor, the assembler, and the linker. These components are all needed in order to process programs. Although the GNU compiler collection is typically invoked by using the command **gcc**, the individual components can be called separately.

You might wonder about the difference between C programs and C++ programs. The GNU compiler collection contains only one compiler, but will compile both C and C++ programs. The GNU compiler collection also compiles Objective C programs as well. The **gcc** command contains optimizations for different processors and systems and is considered one of the best performing compilers available today. In fact, many benchmarks use the GNU compiler collection.

cpp, the GNU preprocessor, performs functions such as macro expansion, conditional compiler directives, inclusion of header files, and line control. The preprocessor is the first step in creating binary executables from source code.

as, the GNU assembler, is used to create object files from assembly code files. The object files it creates are appended the .o suffix and are binary object files. Once you have created object files from C source files, they must be linked together with system object files.

ld, the GNU linker takes object files and links them together with system libraries and object files in order to create binary executable files or programs. Since the linker takes object files and combines them together to make executable files, it is not always necessary to have source code to all of your programs. Oracle ships with a set of object files that are taken and linked with system libraries and object files in order to create the Oracle executable. You don't have access to Oracle source code, but at times you might need to create assembly language programs to replace some of Oracle's functions. For example, in order to change the base address where Oracle runs, it is necessary to create an assembly language file called ksms.s and relink the Oracle binary. This is all done without having access to Oracle source code.

The GNU Debugger

The GNU debugger allows you to create programs with special debug information included. This allows you to set breakpoints within the code and step through the code as it is executing. This is valuable in that it allows you to examine the contents of variables while the program is running. This might help you to determine where the problem is in the program itself—for instance, where a variable is being overloaded or where memory is not properly being released.

In addition, the debugger allows you to examine a program after it has terminated. The debugger may be able to help you determine where the program has terminated so you can fix it. For those of you who have done any type of system development, you will appreciate the debugger utility.

The Binutils

The binutils are a collection of utilities used by developers to assist with the development, deployment, and debugging of Linux applications. They are simply a set of various GNU utilities that are packaged together, and consist of the following applications:

- **ar, the archiver** Creates, modifies, and extracts files from code archives. Code archives are libraries of object files used to organize these object files. Although an archive can be made up of any type of file, we are mostly concerned with object libraries. The ar utility is used to maintain object files within these libraries.
- **nm** Used to list symbols within object files. This is a very useful debug tool that can help you find symbols within object files.

- **objcopy** Used to copy and convert object files. objcopy is used to convert object files from one type to another, such as from bdf to elf.
- **objdump** Used to display the contents of a binary file. This is useful for developers who are concerned about the actual contents and format of their object files, as well as the function of their code.
- **ranlib** Generates an index to archive contents. The ranlib utility will take an object library and create an index of symbols so you can find which object in the library has the symbol you are looking for.
- **size** Lists the section and total sizes of object files. The output displays the various sizes of different parts of your code, such as text and data segments, as well as the size of the bss section of the code.
- **strings** Lists printable strings within files. This is very helpful if you are looking for a particular object file that has displayed an error message.
- **strip** Discards symbols from files. It can be used both for size considerations as well as for security.
- **readelf** Displays information about ELF format files. This is similar to objdump except that it works with ELF files.

Though the binutils may be of no use to many of you, they serve an important purpose when it comes to software development and debugging. This is especially true if you are only planning on running Oracle.

The make Utility

The make utility is used to create binary files from source files and object files based on a template, or makefile. The makefile is used to define dependencies between source files, object files, and executables. These relationships, as well as methods of creating various dependent files, are used to create binaries. For example, the following relationships can be set up within a makefile:

```
test relies on test.o
test is created by linking test.o with lib.a
test.o relies on test.c
test.o is created by running gcc on test.c
```

This is just a description and does not represent the actual syntax of the makefile and the make utility.

Once these relationships are set up, all that is necessary is to run make against the makefile. If test.o is newer than test.c, make understands and will not recompile test.c. If test is newer than test.o and lib.a, it also will not relink test.

Because of the sophistication of make and makefiles, you can set up complicated relationships and when a file or files have been changed, it is not necessary to recompile and relink everything. Make will understand the relationships and will only compile and link what is necessary. This is important for programs such as the Linux kernel, which contain hundreds of programs. If everything needed to be recompiled and relinked every time a single change was made, it would take hours.

The Linux User Interface

Linux has a number of different user interfaces available for use. Among these user interfaces are the command-line interface using the shells mentioned earlier in this chapter and the graphical user interface that uses the X Windows user interface. In many systems I see, the system administrators prefer to only run the graphical environment when needed, since it uses more memory and CPU resources than the character interface. However, some tasks, such as installing Oracle, require the GUI interface.

X Windows

In 1984, MIT formed Project Athena in order to develop a compatible graphical user interface (GUI) that would work on their many different brands of workstations, thus allowing for continuity among those systems. The X Windows system was designed as a graphical user interface that could run both remotely and locally equally well, while relying on either local or remote resources. X Windows has been a part of the Linux operating system since the first release.

The X Windows system is available on almost every operating system and supports a large number of different computer architectures. X Windows is the standard graphics protocol for Linux, but is also supported on Microsoft Windows as well.

The X Windows architecture is divided into two components: the server, which serves graphical images, and the client, which serves the application. To many, the terminology seems backwards. If I were to run an application on a server with the graphics being displayed on my workstation, the workstation is actually the X client, and the server where the application is running is considered the client. This might take a little getting used to for those of us who work with other client/server type applications.

The X Windows system is designed to display things on a screen. It is not designed to provide a look and feel. That's the job of the window manager. The window manager (such as Motif) is designed to provide the look and feel of components such as titlebars, buttons, sliders, and so on.

GUI Environments

Like most things in Linux, you have a choice. Linux comes with many different graphical environments. The two most popular ones are Gnome and KDE. There are other environments available today, but these are currently the ones most people choose. This is partly due to the fact that these two environments ship with the two largest Linux distribution versions. If you want to use another graphical environment for yourself, feel free to. Since Oracle's GUI utilities are written in Java, they run directly using X Windows, irrespective of window manager.

Overview

The graphical user interface (GUI) desktop is the graphical environment that you can log into in order to use the Linux system. For those of you familiar with the Microsoft Windows operating system, it is a similar concept. The GUI desktop provides you easy access to system tools and applications without having to traverse the operating system in a command-line mode. GNOME and KDE, the major desktop environments, both serve the same function, but have a different look and feel. The GNOME desktop environment comes out of the GNU Project and has been around since 1997. GNOME stands for the GNU Network Object Model Environment. KDE, which appeared in 1996, stands for the K Desktop Environment.

The Oracle Java utilities will run equally well regardless of whether you are using the GNOME or KDE desktop environments, or even if you are not running them. In order for these utilities to work, you must have an X Windows server that supports Java.

Remember that the graphical desktop environment can be a useful tool, but it is not a necessity. These days it doesn't matter whether OEM is completely web-based. Web-based administration is simply an option; it's not the only way to do it.

In fact, most users who utilize the system as an Oracle Database server will never see this environment since both system and database administrators will most likely be administering the OS and database remotely over the network. However, for tasks such as installing software, and using tools like the Oracle Enterprise Manager, GNOME and KDE can be very convenient.

By having the ability to use this tool when necessary, you might find some tasks much easier. Others, however, can be done just as simply through the command-line interface. Which interface you use, or whether you actually need one is really up to you.

GNOME

The GNOME desktop began as a free alternative to KDE, which started out mired in licensing issues. GNOME was seen as a free and open way to enable the non-technical user to utilize the Linux operating system.

The GNOME desktop is made up of a number of different development tools and APIs that allow developers to both enhance the GNOME environment as well as develop native applications. While you can develop applications specifically for GNOME, it is not necessary. X Windows and Java applications work as well.

KDE

K Desktop Environment (KDE) predates GNOME and was originally fashioned after the Common Desktop Environment (CDE), a commercial product developed by several major UNIX vendors. While KDE and GNOME are similar in nature, there is some licensing controversy surrounding KDE. If you want to commercially distribute a KDE application, you must purchase a KDE development license before you start coding your application.

KDE is also made up of a number of development tools and applications specifically developed for the KDE environment. Both GNOME and KDE have been ported to other operating systems besides Linux.

Linux Web Server Utilities

Linux first became popular as a web server, even before administrators considered it an excellent Oracle server.

Linux includes one of the best web servers ever designed, the Apache web server.

The Apache Web Server

The Apache web server is an open source HTTP web server that runs on many platforms including Linux, UNIX, and the Windows operating system. The Apache web server is available with the Linux operating system as well as from Oracle as part of Oracle's Internet Application Server (IAS) product and IBM's WebSphere product. The Apache web server is known for its configurability, the ability to interface into multiple database products including Oracle, DB/2, and others. The Apache web server's performance is comparable to any other commercial web server and as of 2003, 62 percent of all web servers were running on the Apache HTTP server.

The Apache HTTP server contains a number of modules, including a PERL module, an authentication module, and a proxy module. In addition, many utilities are available to analyze Apache log files.

There are a number of ways to develop applications that use both the Apache web server and the Linux operating system. These include languages such as Java and PHP. PHP has become a very popular way of creating web applications that include database content.

CGI Programming in Linux

In order for web users to access the Oracle Database, it is necessary to use a CGI or common gateway interface. The Apache HTTP server supports several CGI languages including mod_perl and PHP. mod_perl is an optional module for the Apache web server that embeds the PERL interpreter into the Apache server itself in order to avoid restarting the PERL interpreter every time it is called. By having this as a module within the HTTP server itself, significant overhead can be avoided.

More recently, PHP (a recursive acronym for PHP Hypertext Processor) has become a very popular CGI for Linux and Apache. PHP is one of the preferred tools for creating web applications using Oracle Internet Application Server. Applications for using PHP to access Oracle will be discussed in more detail later in this book.

The Linux Boot Process

The boot loader is responsible for starting up and booting the Linux operating system. Linux has available to it several different boot loaders. Among these are Lilo and Grub. Grub is the newest boot loader and is most often used. It's a state-of-the-art boot loader and supports features such as the use of configuration files. The boot process is a multiphase process

Stage 1—The Primary Boot Loader

In the first stage, the primary boot loader is called by the hardware's BIOS. This primary boot loader is a very small piece of code that is only responsible for invoking the secondary boot loader. The system BIOS simply starts running code from a specific place on the disk. This is the primary boot loader. Once the primary boot loader has been invoked, it will begin running the much larger, secondary boot loader.

Stage 2—The Secondary Boot Loader

The secondary boot loader is where more operating-system specific code exists. This code exists on the /boot partition. The GRUB boot loader will display a menu of available operating systems that can be booted. Here you can select the operating system or kernel you want to boot and GRUB will continue with the next stage, which is loading the OS.

Stage 3—Loading the OS

In this stage, the OS loads. In some cases, a RAM disk is created in order to load drivers needed during the installation process. This RAM disk, known as the initrd or initialization RAM disk, is necessary to load SCSI device drivers. Because of this, whenever you create a new Linux kernel, you must also create a new initrd image.

Once enough of the OS has been started, control is passed to the /sbin/init program. The /sbin/init program in turn starts services and tools. The /sbin/init program also mounts filesystems needed for normal operation of the OS. Once this stage has completed, the Linux system is ready to be used.

Linux Source Code

One advantage of Linux is that the source code is available freely on the Internet, as well as with the Linux distribution. If you choose to install the kernel source "package," it will be installed in the /usr/src directory. Under this directory, you will see a directory

named for the version of the Linux source code stored there, such as /usr/src/linux-2.4. Under this directory, you will find the source code for the Linux kernel as well as device drivers. With the Linux source code installed, you not only can rebuild the kernel, you can add additional device drivers as well.

In many cases, it is not necessary for you to rebuild the Linux kernel by hand since the rpm or installation package will rebuild the kernel for you. However, under certain special circumstances you may want to rebuild the Linux kernel. This is covered in the next section.

Rebuilding the Linux Kernel

Rebuilding the Linux kernel is typically unnecessary; however, you might find that you want to make some changes that require a kernel rebuild. In this case, follow these instructions in order to build your own kernel.

1. Make sure you have the Linux source code package installed. Alternatively, you can download this off http://www.kernel.org/. You would not do this for Enterprise Linux.
2. Run the command **make mrproper**. This command cleans up the source tree and removes any remnants of previous kernel builds, thus ensuring you are creating the kernel from scratch.
3. Create the .config file. A good starting point for the .config file are the files in the /usr/src/linux-2.4/configs directory. You will find a number of files in this directory with names like kernel-2.4.21-i686.config, kernel-2.4.21-i686-smp.config, and kernel-2.4.21-i686-hugemem.config. Pick the kernel config template that best represents your system. If you have more than one processor, pick the kernel config file with smp in the name. If you have more than 4GB of RAM, pick the one with hugemem in the name.
4. Customize the configuration file. This can be done by editing the file, or by using the command **make menuconfig** in order to run the Linux kernel configuration utility (shown in Figure 1-3). You could also use **make config**, **make xconfig**, or **make oldconfig** to create the kernel configuration file.
5. There are three ways to modify the .config file before building your custom kernel: **make config**, **make menuconfig**, and **make xconfig**.
6. Once you have created the configuration file, run **make dep**. This will create the kernel dependencies.
7. Run **make clean** in order to clean the kernel tree.
8. Build the kernel by running **make bzImage**. This will create the Linux kernel. It's not a bad idea to modify the makefile to make sure the default kernel

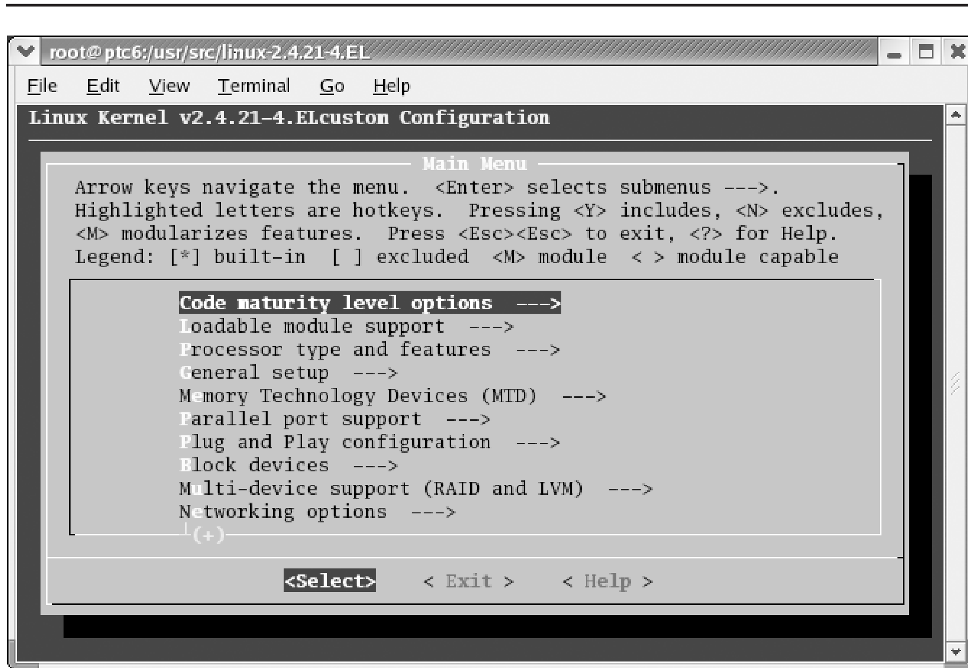


FIGURE 1-3. *The kernel configuration utility*

isn't overwritten. With Red Hat, the line beginning with EXTRAVERSION tags custom on the end of the kernel name.

9. In order to create the Linux modules, run the command **make modules**.
10. Once you have created the Linux modules, you will install them by running the command **make modules_install**. This will install the modules into the proper directories. The directory where the modules are installed includes the kernel version number. For a custom kernel, the directory might be something like /lib/modules/2.4.21-4ELcustom/kernel/drivers.
11. In order to install the Linux kernel, run the command **make install**. This command performs a number of functions including copying the kernel to the boot partition, building a new initrd image, and adding the entry to the bootloader configuration file.

Once you have concluded these steps, the kernel is ready to be booted.

Processes and Threads

Within the Linux operating system, there are many programs simultaneously running on the system. In the operating system, these "programs" are commonly known as processes. Each process has its own virtual memory space and runs independently of other processes. As you will see in the next chapter, Oracle depends on many different processes to perform different functions. Threads are concurrent "threads" of instructions which run within a single process, like having concurrent programs within a single larger application. The way that Linux is architected there is little benefit from running with threads instead of processes, as they take the same types of kernel accounting structures, and most kernel programmers strongly recommend against using threads. See the following section for more details.

What Is a Process?

A process has memory and CPU cycles associated with it since it's a program that performs a function and runs on the processor.

Processes are used throughout the operating system. The Linux tasks scheduler is responsible for scheduling all of the processes running on behalf of the users. Each process runs for a few milliseconds and is then taken off of the CPU before another program is loaded in . This operation, known as a task switch or context switch, ensures that all running programs get some amount of processor time. Within the kernel, a process is made up of a number of different components; code ("text"), data, stack variables, file I/O information, and signal tables. These components must all be copied in and out of memory as the process switches in and out.

The context switch operation occurs tens of thousands of times per second on your system. Although this is an integral part of the Linux operating system, it is an expensive operation. In order to minimize the expense of the context switch, the Light Weight Process or Thread was developed.

Oracle's Use of Processes

Oracle's architecture takes advantage of the use of many processes. As you will learn in the next chapter, the Oracle instance is made up of a number of different processes that perform different functions. By using a number of different processes on different CPUs, Oracle can achieve very good scalability.

What Is a Thread?

A thread minimizes the cost of the context switch by allowing several processes to share some of the same information. If you have many users utilizing the same program, the program can be loaded into memory and the same memory and stack variables

can be used by different incarnations of the same program by just moving the stack pointer to where that instance of the program happens to be running. This cuts down on context switches, since only the stack pointer is changing.

These threaded operations only work within the larger context of a process, thus different processes that are running different programs still require a context switch. This is known as a user-space thread.

In addition to user-space threads, there are kernel-space threads that are used to run the operating system itself. This allows the kernel itself to be threaded and thus very efficient.

Devices

The Linux operating system is made up of many devices. A device can be seen as a file that is used to send input and output through. Some files represent a piece of hardware and some are software components. Regardless of whether the file is a hardware or software component, I/O can be directed to it in essentially the same manner.

Many hardware devices exist on the system, such as the floppy drive, the fixed disk, network devices, and so on. Devices are divided into two categories: character or raw devices and block devices.

Character Devices

A character device is a device where the user process transfers data directly to the device in a sequential manner, rather than block devices which can be accessed in a more random fashion. Some examples of raw or character devices are

- Tape devices
- Floppy disk drives
- Terminal devices
- Parallel print devices
- Sound devices

In addition, character devices can be accessed by programs that would normally use block devices. Oracle can actually use raw devices for its data storage. However, Oracle must use a block interface to this character device. This will be explained in more detail in subsequent chapters.

Block Devices

Block devices allow buffered I/O and can handle more specific requests for data. Block devices can be accessed randomly. Unlike a character device that can receive any size request, block devices will access the underlying data only in certain block size requests. This is why they are known as block devices. Block devices include the following:

- RAM disks
- SCSI disk devices
- CD-ROM devices

The devices we normally use for Oracle utilize block device–based filesystems.

Logical Volume Managers

An Oracle database can use raw devices or filesystems, but it can also use an LVM for its storage. A Logical Volume Manager, or LVM, is a software product that manages storage by combining physical disk blocks from different physical disks into a logical disk volume. This allows you to manage your storage in an orderly manner for both performance and ease of use. Features of an LVM include

- **Storage aggregation** Many individual disk drives can be combined together into a single logical disk drive.
- **Storage virtualization** The storage can be reconfigured, added to, and resized even while that storage is being used. You can add additional storage to a volume while you are using it.
- **Snapshots** With an LVM, you can take snapshots of the data, thus creating faster and more efficient backups.

LVMs have been used with Oracle for storage for many years. Linux LVMs are becoming much more common, with several types available for use.

Automatic Storage Management

Automatic Storage Management (ASM) is Oracle's answer to LVMs. ASM is an LVM which has been developed specifically for Oracle, working with it to provide an Oracle storage virtualization layer. In essence, ASM simplifies Oracle storage management

by allowing disk groups to be formed that can be used to hold the Oracle data and logs. By using ASM, the actual physical placement of the data within the file group is done automatically as well as tablespace management. ASM is designed to both simplify and optimize data storage within the Oracle database and is another option that you might choose.

Filesystems

Most storage within the Linux system is done via a filesystem. A filesystem is the layer above the block devices that allows user access to the disk devices. It permits structures such as directories and files. The filesystem is created on top of the disk device, has its own cache, and is used to manage access to the disk drive.

Filesystems provide features such as directories, which are abstractions that allow for a hierarchical storage structure. A directory is a structure that can hold other directories or files. These files, in turn, can hold data.

Another feature of a filesystem is its ability to place properties on directories or files in order to provide security by allowing permissions to be put on files and directories.

Filesystems may have other features, such as the ability to work within a cluster and the ability to journal the data being written to the filesystem. Some specific filesystems and their features are discussed in the next sections.

ext3

The primary filesystem used within Linux is ext3, which is based on ext2 with filesystem journaling. Journaling reduces the amount of time needed to recover changes to the filesystem if the system is uncleanly rebooted, which can take hours on a multiple-gigabyte storage device. In a journal led filesystem, changes that are constantly being made to the disk are written to a special file called the "journal." This journal is similar to the redo log on Oracle and is used to fix inconsistencies that might have occurred during a power failure or some other loss of processing. Journal led filesystems have been in use for many years and are extremely fast and stable.

OCFS

OCFS is the Oracle Cluster Filesystem. The Oracle Cluster Filesystem is an Oracle filesystem that allows multiple systems to use the same filesystem. This overcomes the problem of accessing shared storage for Oracle RAC (Real Application Cluster) systems.

The advantage of OCFS is that you can build a cluster using a filesystem. This makes the management tasks involved in handling data files used in an RAC system much easier. With OCFS, you can create data files as you would under any other filesystem. Files can be set with auto growth on and additional files can be added as needed.

RAW Devices

RAW devices are also supported by Oracle. With RAW devices, Oracle does not use a filesystem. The Oracle RDBMS manages the storage entirely itself. RAW devices have been very popular for Oracle because of performance. By bypassing the filesystem code, CPU usage can be reduced since there is no filesystem processing being done. However, managing Oracle on RAW devices can be complex and difficult.

Until the introduction of OCFS, Oracle RAC databases were required to be built on RAW devices.

By using RAW devices, Oracle is responsible for the data storage, and since the Oracle RAC cluster manages the storage as well as the cluster, the system coordination problems are solved. RAW devices are limited and difficult to manage. With Oracle 10*g* we now have several options for a RAC cluster: RAW devices, OCFS, or ASM.

Summary

This chapter has introduced you to the Linux operating system. You have learned a bit about the directory structure, the tools available to you, and how to configure some of the key components. It is important to understand the OS in order to properly manage the Oracle database on this OS. Of course, this chapter is not a complete administration guide for Linux. There are plenty of good books on the market for that.

In this chapter, we attempted to give you a flavor of the Linux architecture, so that in the next chapter when you learn about the Oracle architecture, you will begin to understand how the two work together. There, you will find out about the Oracle Database architecture and afterward begin to learn more of the specifics of administering the Oracle Database on Linux.