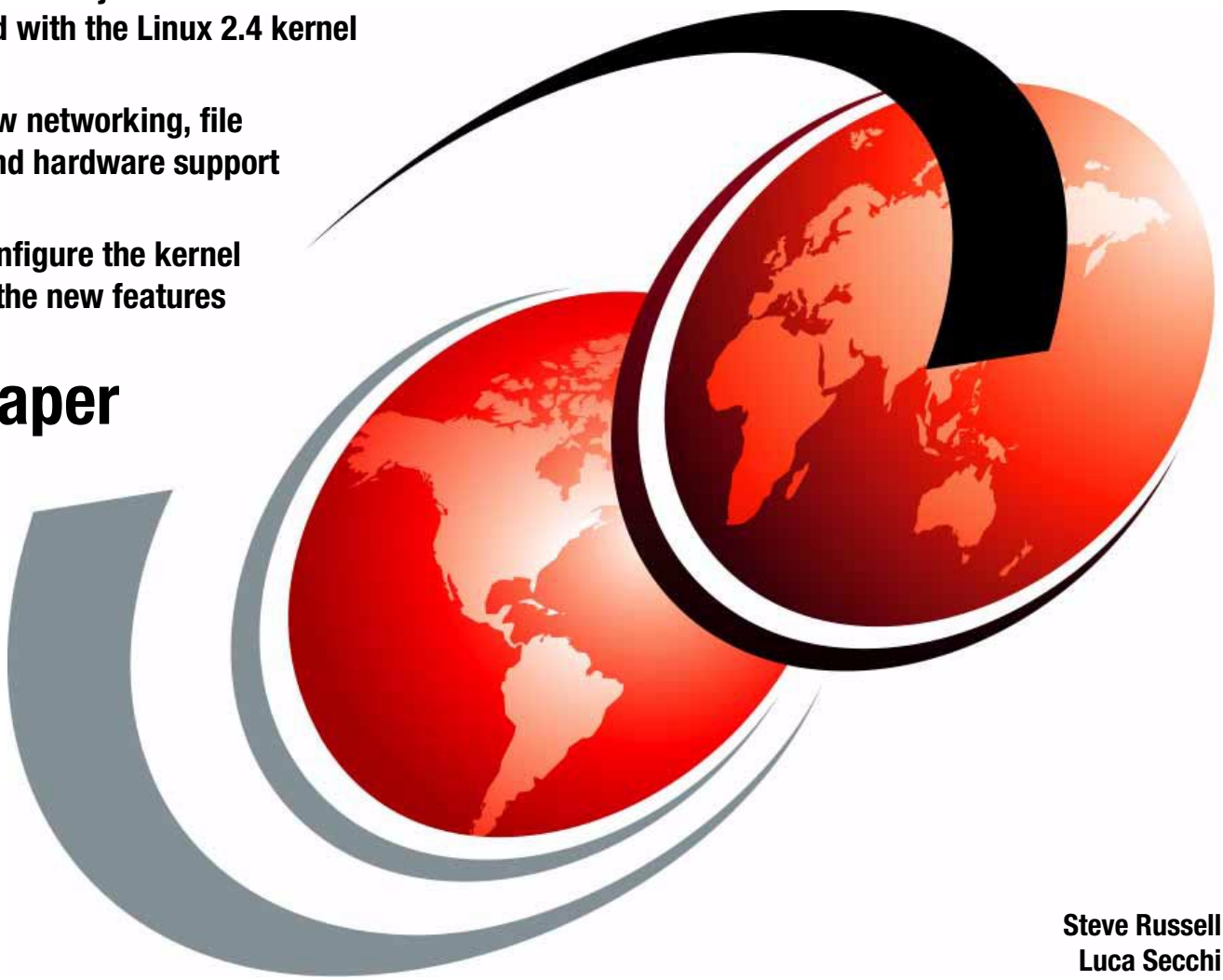
IBM

Running the Linux 2.4 Kernel on IBM @server xSeries Servers

- Describes the major new features introduced with the Linux 2.4 kernel
- Covers new networking, file system, and hardware support
- How to configure the kernel to enable the new features

Redpaper



Steve Russell
Luca Secchi

ibm.com/redbooks

Redbooks

International Technical Support Organization

Running the Linux 2.4 Kernel on IBM @server xSeries Servers

August 2001

Take Note! Before using this information and the product it supports, be sure to read the general information in “Special notices” on page 87.

First Edition (August 2001)

This edition applies to Linux distributions using the Linux 2.4 kernel

This document created or updated on August 8, 2001.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 662
P.O. Box 12195
Research Triangle Park, NC 27709-2195

Contents

Preface

This redpaper is geared toward technical staff within IBM, our customers, and our Business Partners who intend to make use of the latest version of the Linux kernel (V.2.4). The new kernel offers a number of features and enhancements not previously available, making Linux even more suitable as an enterprise-class operating system. Features such as new file systems that support logical volumes and journaling, improvements to Samba, support for Asynchronous Transfer Mode (ATM) networking, and support for a wider variety of hardware all combine to enhance the appeal of Linux for many applications.

IBM has a strong commitment to Linux across all of its server platforms and nowhere more so than for the IBM @server xSeries class of servers. xSeries systems provide an excellent platform for Linux deployment, with support for four distributions of Linux, namely, Red Hat Linux, Caldera OpenLinux, TurboLinux, and SuSE Linux.

Discussions of the Linux kernel are somewhat independent of the specific distribution being used. Because of this, we have used the Red Hat distribution for many of the examples, but these can easily be used to extrapolate to other distributions.

The team that wrote this redpaper

This redpaper was produced by a team of specialists from around the world, working at the International Technical Support Organization, Raleigh Center.

Steve Russell is a Certified Consulting IT Specialist at the International Technical Support Organization, Raleigh Center, where he manages residencies and produces Redbooks related to IBM xSeries servers. Before joining the ITSO, Steve had a technical marketing role, working in the UK as a member of the IBM server organization in EMEA. Prior to that, he spent nearly 15 years managing and developing PC-based hardware and software projects at the IBM Hursley laboratory in the UK. He holds a degree in Electrical and Electronic Engineering, is a member of the Institution of Electrical Engineers and a Chartered Engineer.

Luca Secchi is an IT Specialist working for IBM Global Services in Italy. He has two years of experience with the Linux operating system as system administrator and consultant. His areas of expertise include security and network management. Luca holds a degree in Electronic Engineering from the University of Cagliari in Italy.

Information from a wide variety of sources was drawn upon in the preparation of this paper. Our thanks go to the many contributors who created the vast number of Linux Web sites, HOW-TO documents, man pages, and other repositories that are such a valuable resource for Linux enthusiasts all over the world.

Special notice

This publication is intended to help technical staff implement Linux systems based on the Linux 2.4 kernel. The information in this publication is not intended as the specification of any programming interfaces.

IBM trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

e (logo)® 
IBM ®

Redbooks
Redbooks Logo 

Comments welcome

Your comments are important to us!

We want our redpapers to be as helpful as possible. Send us your comments about this redpaper or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  `ibm.com/redbooks`

- Send your comments in an Internet note to:

  `redbook@us.ibm.com`

- Mail your comments to the address on page ii.

1

New file system features

We first explore the topic of support for a number of new file systems, one of the major areas to be enhanced with the release of kernel 2.4. In particular, we focus on support for Logical Volume Manager, and for the new journaled file systems now available for Linux.

We also briefly introduce new SAMBA performance enhancements, Network File System Version 3 (NFSv3) support, RAW I/O device implementation and the new partition tables and file system support introduced in Linux 2.4.

We start with a general introduction to the basic concepts and mechanisms fundamental to the new file systems' operation, examining these from a somewhat technical perspective. Chapter 4, "Compiling the Linux 2.4 kernel" on page 49 gives practical guidance for compiling the features necessary to enable support for the file systems discussed.

1.1 Kernel and file system overview

In this paper, we explain the basic concepts of the Linux kernel structure and organization as they become useful in understanding the new features and abilities offered by version 2.4 of the kernel. We have strived to maintain a balance between clarity and completeness, so as to speak to both Linux experts and less experienced users at the same time.

An overview of the file system organization used within Linux will be useful, primarily because many new important features have been introduced in this area, and because of the way that Linux manages files and file systems.

Just as in the UNIX operating system to which Linux is closely related, many types of objects are treated as a file in Linux systems. For example, all of the following objects appear to the system as different file types:

- Regular files, such as text files, databases, programs, etc.
- Directories
- Symbolic links
- FIFOs (named pipes)
- Sockets (used for networking and interprocess communication)
- Block and character devices (see 1.1.3, “Block, character and network devices” on page 6)

When dealing with files, it is useful to understand the procedures and software structures that allow the operating system (OS) to map files onto hardware devices.

1.1.1 Linux kernel and devices

In simple terms, Linux has a layered structure in which each layer communicates only with those adjacent to it. This is illustrated in Figure 1-1:

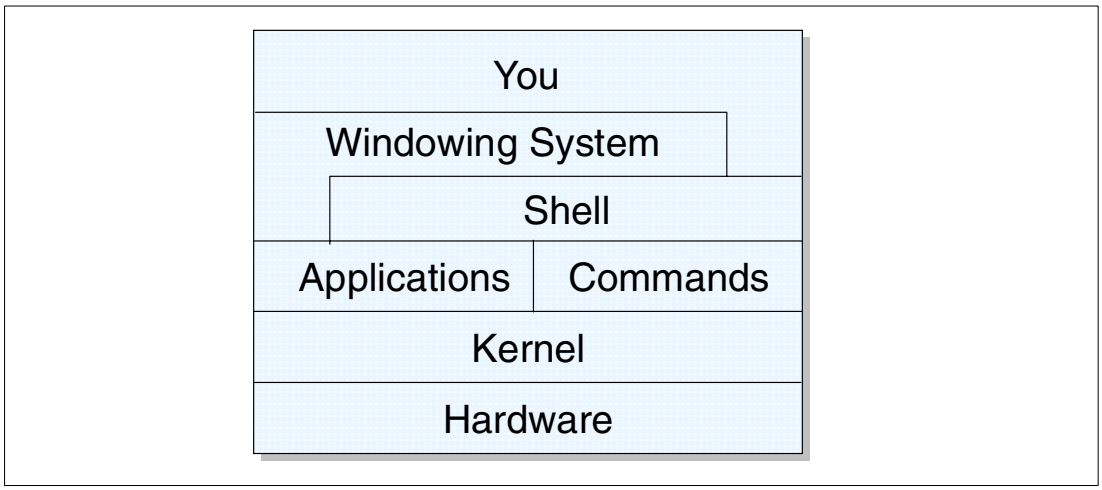


Figure 1-1 Simplified structure of Linux

Linux is very flexible and configurable, and offers a number of different shells and windowing systems. For our purposes, however, we can disregard both of these layers, as they are simply ways to interface the user to the underlying applications and commands layers.

User applications and commands work in the so-called user space; this means that they have only mediated access to system resources through the kernel. This mediation includes file access rights control, CPU time scheduling (multitasking), memory and device access control, etc.

All requests for system resources must go through the kernel to get to the hardware level. Here, each request, including those from the kernel itself, must be translated into precise manipulation of hardware device *controllers* and *registers*.

Controller, or adapter, is a generic term used to refer to the electronic logic present in the hardware interface to almost every physical device. Examples include a SuperIO chip used to control mouse and serial ports, or an integrated device electronics (IDE) or small computer system interface (SCSI) controller used to manage a system's hard disks.

It is important to understand that each of these controllers has a different and unique set of control and status registers, which are used by the system's software to control the attached device. For the kernel to support a specific device, it has to use hardware-specific code that manipulates the device's registers correctly in order to get the expected behavior from the device.

1.1.2 Linux Extended-2 (ext2) file system basics

When ext2, devised by Rémy Card, was introduced in 1993, it quickly became the most utilized file system on Linux systems. The organization of Ext2 is similar to that of the Unix File System (UFS). Each logical partition is divided into blocks which store two types of information:

- Data belonging to a file

We talk about each kind of file in general terms in 1.1, "Kernel and file system overview" on page 2.

- Metadata information

  This is, as you would expect, information about a file's data, such as how the actual data is stored (in which blocks), ownership of the data, what access permission privileges are granted to users and groups, etc.

Blocks

Storage on devices is divided into blocks. It is important to note that a block represents the minimum allocation unit for storage. If a file is smaller than the block size, an entire block is still allocated to store it, thereby wasting space.

Blocks are organized into block groups. Each block group duplicates information regarding the entire file system, for recovery purposes, and contains file data and information structures (see Figure 1-2 on page 4):

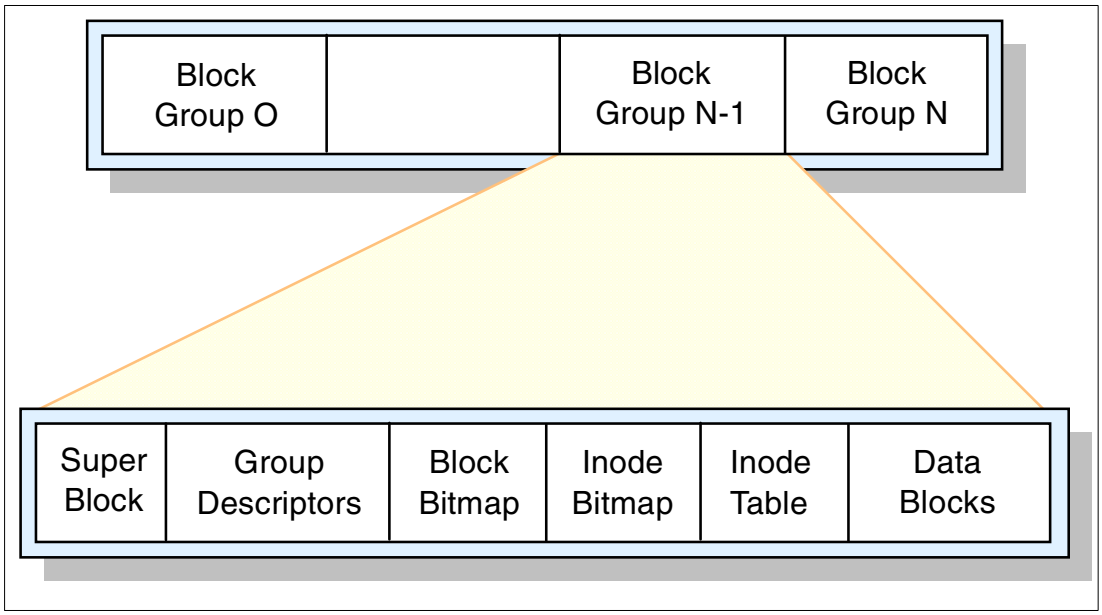


Figure 1-2 Block group structure

Examining the figure, we can see how block groups are organized:

1. The superblock of each block group contains information about the basic size and structure of the file system, such as the block size in bytes, the number of blocks in a group, the number of free blocks in the file system, and so forth.
2. Group descriptors contain pointers to the block bitmap, the inode bitmap, and the inode table, as well as information about the number of information nodes (inodes), directory and free blocks in the group.

   Note: When the file system is created (using the mke2fs utility), the block size is specified (it ranges from 1024 to 4096 bytes), as is the number of inodes. The number of inodes specified when the file system is created is directly related to the maximum number of files that can be stored in the file system, and cannot be changed at a later date.

   If you need to store more files than the file system can support due to the number of inodes specified, one way to overcome the limit is to create a new file system, then to copy the old file system to the new one. Another way to avoid this limitation is to use dynamic inode allocation techniques (see 1.3, “Journaled file systems” on page 11).

3. The inode table contains inode structures. The ex2 file system describes each file with a inode data structure. Each inode contains the description of the file: file type, access rights, owners, timestamps, size, and pointers to data blocks.

Each inode structure is associated with a single file in the system and has pointers to data blocks containing a file’s actual data. Sufficiently large files are accommodated by using indirect pointers and, if necessary, double-indirect pointers, as shown inFigure 1-3:

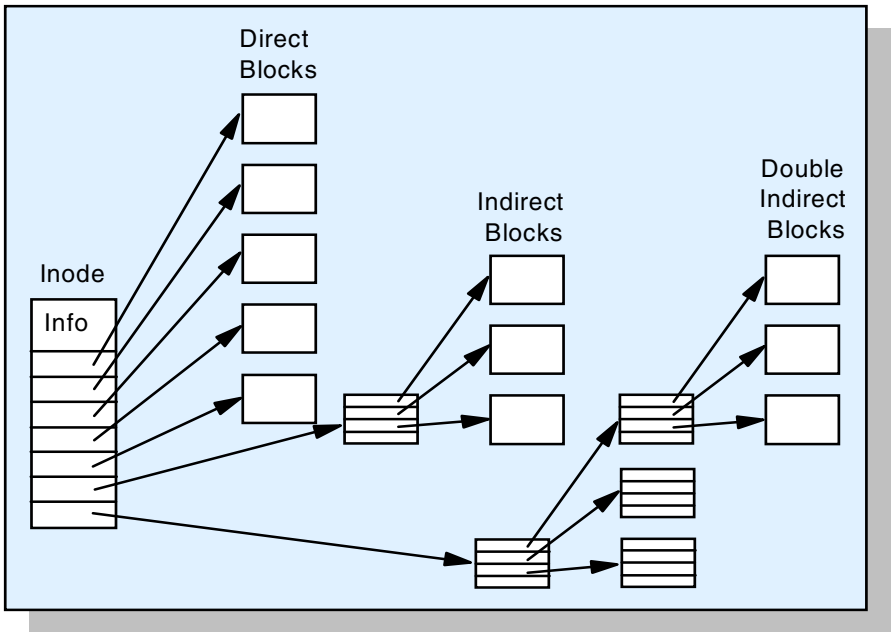


Figure 1-3 Simplified inode structure

Directories are just files related to a specific inode that each contain a list of files, referred to by inode numbers and associated with file names. When a process refers to a file by using its pathname, the kernel scans the directory tree structure, starting from the root directory, to find the specific inode associated with the file. Figure 1-4 illustrates the linkage between inodes and files:

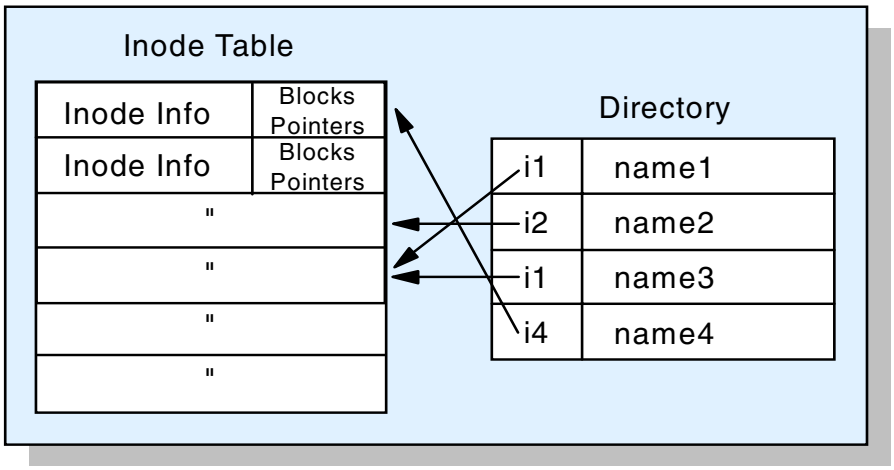


Figure 1-4 Directory files and inodes

The ext2 file system uses bitmaps to track free blocks. The block bitmap consists of an array of bits, where each bit corresponds to a block in the group and is set to 1 if the block is in use or to 0 if the block is free. A similar mechanism is used to track inodes belonging to the group, using the inode bitmap.

This section has provided only a brief introduction to the ext2 file system. More detailed information can be found at:

http://www.redhat.com/mirrors/LDP/LDP/tlk/tlk.html

Note: Even prior to the availability of ext2, the real file system in Linux was separated from operating system services by an intermediate layer called the Virtual File System (VFS). We discuss VFS in 1.1.4, “Virtual File System” on page 6.

1.1.3 Block, character and network devices

The most interesting feature of kernel device management is *abstraction*. This is the term used to describe how all hardware devices are made to look like regular files and, just like files, can be opened, closed, read and written to, using file manipulation system calls. Each device on the system is represented by a special file in the /dev directory.

We can distinguish among block, character and network devices. Block devices are represented as linear arrays of bytes that can be randomly accessed; this includes floppy disks, hard disks or CD-ROMs. Character devices may only be accessed in a sequential manner, and include mice, keyboards, and serial or parallel ports, as well as multimedia devices such as sound cards or TV tuners. Network devices, naturally, deal with networking connections based on different hardware standards such as Ethernet or Token ring.

Block and character devices are created as *device special files* by the **mknod** command, or indexed by the devices file system (devfs) mechanism (see 3.1.1, “Linux 2.4 devfs” on page 44). Each device special file has a major and minor number associated with it.

Major numbers identify the driver to be used with a particular device controller, for example an IDE controller; minor numbers distinguish between different devices connected to the same controller, for example a particular IDE hard disk.

Network devices are built dynamically in memory as the system finds and initializes them.

We discuss the new hardware support for block, character and network devices found in the 2.4 kernel in Chapter 3, “Linux 2.4 kernel hardware support and performance” on page 43.

1.1.4 Virtual File System

The Linux file system structure is hierarchical, similar to that found in UNIX systems. All files are organized in a tree with a single root. This is a big difference from operating systems such as Windows or MSDOS, in which different devices are associated with unrelated file systems.

In Linux and UNIX systems, we have access to devices through file manipulation, as previously stated, and it is also common to have access to different file system types (such as FAT or ext2 in Linux) mapped on different directories of the same tree structure.

To achieve this great flexibility when dealing with different file systems, Linux implements a Virtual File System (VFS) switch software layer that provides a uniform file system interface to user space applications. As the name suggests, the VFS represents a virtual map of all the files in the system environment, and offers a way to index and access those files in a transparent way.

As we can see in Figure 1-5 on page 7, the VFS operates on a different abstraction level than the real file systems, giving the whole system great flexibility:

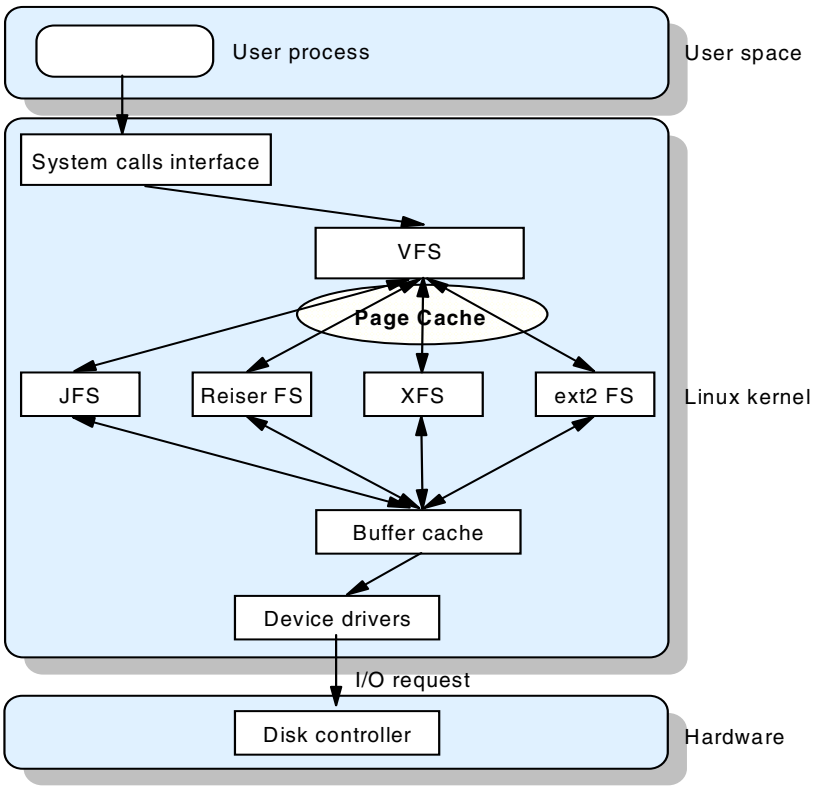


Figure 1-5 Virtual File System layer

VFS describes files in terms of inodes and file system structure in terms of superblocks, in much the same way that the ext2 file system does, but take care not to confuse the two levels of operation. VFS inodes exist only in the kernel's memory and are cached as long as they are useful to the system. Among other information, they contain an identifier for the device holding file's data and an inode number that is unique within a specific file system. These two elements together identify a file that is unique throughout the entire Virtual File System.

Another important field of the VFS inode structure is the *inode operation* field. This is a pointer to a set of routine addresses. These routines are specific to the file system that actually holds a file's data; the kernel uses the pointer to access device-specific routines in order to perform operations on the file or device

It is not our intent to provide a complete explanation of VFS software structures and functions. Rather, we discuss those aspects that are useful for the topics covered in this document. For a detailed discussion of VFS and the Linux kernel, go to:

http://www.redhat.com/mirrors/LDP/LDP/tlk/tlk.html

VFS operation

To briefly understand how the VFS operates, we provide an example, highlighting only the points significant to our discussion.

Suppose we have to mount the second partition of a SCSI disk on the /mnt directory. This is achieved with the following command:

```
$ mount -t vfat /dev/sda2 /mnt
```

1. To begin with, the VFS has to find out if it can handle the file system specified with the `-t` flag, in this case `vfat`, and must look for it through the list of file systems. If it does not find the file system in its internal list, and the kernel is built to support modules, the kernel daemon will load the appropriate module.
2. Next, VFS has to find the VFS inode of the /mnt directory that is to be the vfat file system mount point.
3. If the associated inode tells VFS that /mnt is a directory and that there is no other file system already mounted on it, VFS reads the information on the vfat superblock and builds a corresponding VFS superblock structure. In the VFS superblock structure, the system can find all the indispensable information about how the data is organized on the device, such as the first inode in the file system, the block size or a pointer to a set of superblock routines for the specific file system, and so on.

At this point, each time a user application has to access the mounted file system, the mount point directory is accessed. VFS translates each read and write operation to the appropriate operations for the specific file system being accessed.

This entire process is completely transparent to the application layer.

Note: The VFS layer has been improved and had some of its bugs fixed. In older versions of Linux, file caching was dependent on a dual-buffer system that simplified some coding but caused problems for kernel developers. Code had to be written to make it impossible for these buffers to be out of synchronization. Buffer synchronization slowed down the system and the redundant buffer increased memory use. File caching is no longer dependent on a dual-buffer system with Linux 2.4.

1.2 Logical Volume Manager

One of the most important new features in the Linux 2.4 kernel is the Logical Volume Manager implementation by Heinz Mauelshagen. This kernel module and its related utilities are still in beta release (version 0.9.x) at the time of writing this redpaper, but are a very important feature for Linux in enterprise applications.

Volume managers are a de facto standard among UNIX operating systems, such as the IBM AIX system, and it, along with other features we discuss in this paper, represents a fundamental milestone on the road to Linux enterprise support. Another, related feature, implemented in an earlier Linux kernel release, is the Multiple Devices (MD) driver. This used to manage software redundant array of independent disks (RAID), and can be utilized in conjunction with LVM to provide more flexibility to the system.

That said, it is always preferable to use hardware RAID technology rather than software implementations, because hardware solutions offer a more robust, higher performance solution, which is important in critical applications. IBM @server xSeries systems, in conjunction with the IBM ServeRAID adapter, are particularly well suited to the high availability environments.

1.2.1 LVM basics

In modern enterprise systems, applications sometime need to have access to a data store that exceeds the capacity of one or more physical disks. The ability to manage this great amount of data as a whole is also a necessity. In addition, it is often impossible to install a specific file system across different devices. Hardware RAID controllers can offer a solution to this problem by making multiple physical drives appear as a single device to the operating system.

Volume managers, such as LVM, offer another, integrated solution to this problem. LVM makes it easier to deal with large amounts of data by implementing a volume driver that makes a collection of disks appear as a single logical disk. A volume driver is a mid-level block driver that operates between buffer cache and physical peripherals (see Figure 1-5 on page 7) through manipulation buffer_head basic I/O descriptors (see 1.2.4, “LVM internals” on page 11).

Logical Volume Manager is a powerful instrument for the system administrator, allowing organization of online storage in volumes that can be resized or moved without any denial of service to your users. Other benefits come from the use of meaningful names such *marketing* or *customers* instead of standard device names such as sdb or hdc, and from the ability to create a snapshot of any file system under control of LVM.

1.2.2 LVM structure and terminology

The LVM storage system is organized in layers, as shown in Figure 1-6:

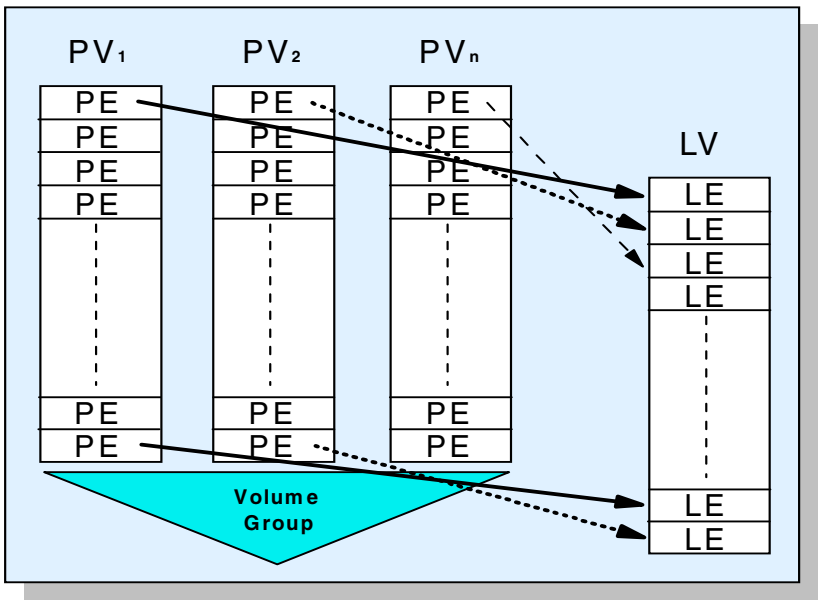


Figure 1-6 Volume groups, physical volumes and logical volumes relationship (striped mapping)

At the bottom of the LVM structure, we find the physical volume (PV). A physical volume is actually a single partition or a device (for example, a hard disk, or a hardware or software RAID array). Each physical volume is divided into physical extents (PE) that can range from a few megabytes to several gigabytes in size to allow allocation flexibility. Physical volumes are combined into volume groups that form pools of storage for administrative purposes.

To make the available storage space usable, we have to specify how the physical volumes are to be used as block devices. This is done by creating a logical volume (LV), which is analogous to a partition in a system without volume management. Physical volume space is allocated to logical volumes in units called logical extents (LE). Each logical extent is mapped to a physical extent or, in other words, a logical extent is a physical extent that has been allocated to a specific logical volume address space.

As we allocate space from our volume groups to our logical volumes, we can specify whether we want a striped or a linear mapping policy.

- Linear mapping implies that physical extents from a physical volume will be assigned in sequence to a logical volume, one after the other, adding physical extents from another physical volume only when the first volume's extents are exhausted.
- Striped mapping implies that physical extents are allocated to a logical volume from different physical volumes in a cyclical fashion (as shown in Figure 1-6 on page 9). This could enhance performance if the physical volumes are distinct devices (for example, two separate hard disk drives), but may reduce performance if we use different partitions from the same physical disk, because of the added seek time (the disk's head assembly has to move from one partition to another many times).

1.2.3 LVM snapshots

Backing up your data is a necessity understood by anyone that has worked with computers for any length of time. Finding a suitable time to perform the backup process can be a problem, due to growing volumes of data and the relatively slow performance of backup devices such as tape drives. This problem is exacerbated in business critical applications, where the amount of time available for performing backups while your server is offline may be limited. Similarly, it is possible to unmount a file system before doing a backup, but this is often impossible in production environments.

To help solve this dilemma, LVM provides the ability to take a snapshot of a logical volume. This is done by creating a new block device which is an exact copy of the logical volume at a certain point in time, making it possible to back up data from the snapshot without taking the data offline.

Backing up data from real applications is not quite that simple, however. A problem with this technique is that the data in the logical volume has to be in a consistent state when the snapshot is taken. If a snapshot is taken of a logical volume that holds a database while that database is active, it is possible, even probable, that the database will be in an inconsistent state at the time the snapshot is taken, and the resulting snapshot will not be usable. To circumvent this problem, database applications often have software hooks which can be used to force the application to temporarily suspend operations, leaving the database in a consistent state while the snapshot is taken.

The same issue can apply for basic file system snapshots, but there is an initial kernel patch that supports software hooks to allow consistent snapshots to be taken. User mode applications that do not provide hooks are likely to be caught in inconsistent state despite file system coherence. You need to be aware of this danger and perform extensive testing before you consider implementing the snapshot function.

At present, the snapshot capability is still an unsafe and only partially developed feature, but it is likely to become an important element of LVM in future releases.

1.2.4 LVM internals

To understand how LVM works, we must go back to the VFS structure. In Figure 1-5 on page 7, we can see another important element in the kernel management of file systems: the buffer cache interface to the device drivers.

All read and write requests are subdivided into atomic operations described by data structures inside the kernel. Read and write operations are subdivided this way to allow for better use of system resources. Each atomic element contains information about which mid-level driver (for example, the SCSI disk driver) should be used to handle it.

Atomic operations are queued to the specific drivers to which they refer. Drivers are, in turn, organized as a combination of mid-level and low-level interacting entities. The mid-level element interacts with the rest of the kernel, while the low-level element is strictly hardware-specific.

The mid-level driver periodically dispatches the atomic requests in its queue to its low-level driver using special function calls (namely make_request_fn). These function calls handle the data specifically for each low-level driver involved in the operation; for example, they would perform different operations on IDE drivers than on SCSI drivers.

So, when a volume manager is used as a mid-level driver (imitating a real disk device), the information contained in the request data structures refers to a location at the logical level and not to hardware physical devices. It refers in fact to some logical extent that the LVM knows how to map to an actual physical extent.

What the volume manager's make_request_fn does is to remap the logical volume onto a physical one, including the specific physical device and the cylinder and sector numbers to be used. At this point, a substituted data structure has to be reprocessed by the kernel, that would now invoke a different make_request_fn to drive a real device.

In this way, volume managers offer much flexibility, and it is even possible to specify another volume manager as the new mid-level driver, causing a stacking of multiple volume managers. This was not possible in Linux kernel 2.2 without special patches, but it is a structural characteristic of the 2.4 kernel.

We will not delve deeper in this topic, although it is important regarding system performance and efficiency, since block devices are often a performance bottleneck for modern systems and their optimization is crucial. If you do want to know more about LVM, visit the Sistina Software Web site at:

http://www.sistina.com/lvm

For LVM installation and utilization examples, refer to 4.4, "Logical Volume Manager" on page 55.

1.3 Journaled file systems

Thanks to open source community momentum and industry support, Linux is gaining most of the primary enterprise server features. Especially important is the ability to manage large storage volumes in an efficient manner, focusing on scalability and availability. For example, it is not uncommon to see large arrays of storage devices, even on Intel CPU-based machines such as xSeries servers. An xSeries server using a single ServeRAID-4H adapter can support up to 60 physical drives, offering storage capacities of up to nearly 5 TB. Systems with Fibre Channel subsystems (such as the Netfinity FAStT500 RAID Controller and associated disks) can achieve even higher capacities.

Such systems are therefore dealing with huge numbers of files, both large and small in size, and must deliver high throughput and fast recovery from system crashes, while minimizing fragmentation and optimizing efficiency in space usage.

In addition, new applications demand new capabilities and the ext2 file system is not necessarily best suited to all new tasks:

- Large amounts of system storage demand fast and efficient recovery from a system crash. Ideally, the time to recover has to be independent of storage size. This need is addressed by journaling techniques.
- In modern enterprise applications, we have to be prepared to work with even petabytes of data (2^{50} bytes or thousands of terabytes). This means that file system data structure has to be more scalable than that found in traditional file systems.
- It is also important to optimize space, supporting sparse files. This implies allocation of physical storage only to those parts of a large file that are actually used. Again, this implies a more flexible approach to file mapping.
- On large servers, it could be valuable to have the ability to allocate large, contiguous files in a scalable way, thus efficiently tracking unused and allocated disk space in the system to reduce fragmentation and enhance performance.
- Directories with a large number of entries can cause inefficiencies when files are being located. An efficient lookup mechanism is needed to minimize scan time.
- Finally, ext2 and similar file systems have a maximum number of inodes (see 1.1.2, “Linux Extended-2 (ext2) file system basics” on page 3), which is determined at time of creation. Choosing an excessively large number of inodes can mean wasted space, but if the number originally selected is too small, you may have to create a new file system and copy data to it. This issue is resolved by using a dynamic inode allocation mechanism.

At present, there are four journaled file systems for Linux that, in different ways, attempt to solve these issues. Most of them are not yet in the mainstream kernel 2.4 release, but are available as patches to the kernel code. The four file systems are:

- Reiser File System (ReiserFS), included in the kernel since version 2.4.1-pre4
- IBM Journaled File System (JFS), now at version 1.0.00
- SGI’s XFS (eXtended File System), now at version 1.0
- Extended-3 file system (ext3); this is an enhancement of the ext2 file system with journaling abilities added. There is no patch yet available for ext3 against 2.4.x versions of the kernel. For this reason, we do not discuss the ext3 file system any further. For more information, we refer you to:

  http://olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.html

1.3.1 Journaling file system basics

As previously stated, an important aspect of large storage capacity is integrity of data. Modern file systems, such as ext2, implement diagnostic analysis and repair to prevent data loss in case of system failure. This procedure is performed each time a device is not correctly unmounted prior to system shut down, and is often capable of rebuilding a correct file system with little or no data loss. The problem with this approach is that it is time consuming. Recovering from a system crash on systems with large amounts of storage demands a lot of time, during which the server is not available.

Using Linux in enterprise applications requires few downtime periods (perhaps none), thus a new approach to data integrity is required. Journaling file systems are an established answer to this requirement in traditional UNIX environments.

File systems update their internal structure information by writing it to disk. As we discussed in 1.1.2, “Linux Extended-2 (ext2) file system basics” on page 3, this kind of information is called metadata and is stored in different ways, depending on the file system in use. Each metadata update may require write operations that could be interrupted by a system crash. At the next power up cycle, the loss of metadata would demand a check and, if possible, a fix to overcome any inconsistencies.

A journaling file system records all metadata changes in a special area called the log, or journal. The journal is updated with details about each file operation that the system is about to perform. The actual operation is performed only after the journal is updated. If the system crashes before the operation is completed, there is enough information in the journal to either complete the operation or to reverse it and therefore return the system to a consistent state.

Thanks to this technique, it is possible to avoid a full system scan and to complete the system recovery in a fraction of the time typical for a traditional file system. Moreover, there are no files lost during the recovery process because the information about pending operations is available in the journal.

The price of this added robustness is a general lack of performance by journaled file systems compared to more traditional ones. This performance loss can be minimized using other optimization mechanisms, as we will see.

1.3.2 Scalability and performance

The growth in storage media capacity and the increasing demand for larger archives of data, with greater directory and file sizes, presents new problems that modern file systems must face. The number of bits used to store file information in inodes (such as file size) or general file system related information (such as partition size) has to grow in order to deal with new media capacity. Table 1-1 shows a summary of the important parameters for the journaled file systems under discussion:

Table 1-1 Comparison of journaled file systems

File system	Maximum file system size	Block size	Maximum file size
SGI XFS	18 thousand petabytes	512 bytes to 64 KB	Nine thousand petabytes
ReiserFS	4 giga blocks, 16 TB (4 KB blocks)	Up to 64 KB	4 giga blocks, one thousand petabytes (cache limit 16 TB on 32-bit architectures)
IBM JFS	From 4 to 32 petabytes (depending on block size)	512, 1024, 2048, 4096 bytes	512 TB (512 byte blocks) Four petabytes (4 KB blocks)

The new 2.4 kernel extends the 2 GB VFS limit to support files up to 16 TB in size, but the maximum device size limit is still only 2 TB. This means that at present it is not possible to fully exploit the features of these file systems. These limits will certainly be extended in future releases of the kernel, and the file system scalability will be ready for it.

Dealing with huge media stores efficiently implies new file system organization and is no longer a mere question of the number of bits needed to handle the increased sizes. Implementing structures and methods such as extents and balanced trees (B+trees) helps to provide good performance in these situations.

Extents (cousins of those introduced in LVM paragraph) are groups of continuous logical blocks. They are defined by three parameters:

- First block
- Extent size
- Offset

The first block indicates the address where the extent begins, the extent size tells us the number of blocks in the extent, and the offset specifies the sequence of the extent within the file. Extents are very useful in dealing with large files. They enhance spacial locality because of block contiguity and are also efficient in providing a way to organize and track free contiguous space on disks. An example of the way extents relate to a block device is shown in Figure 1-7:

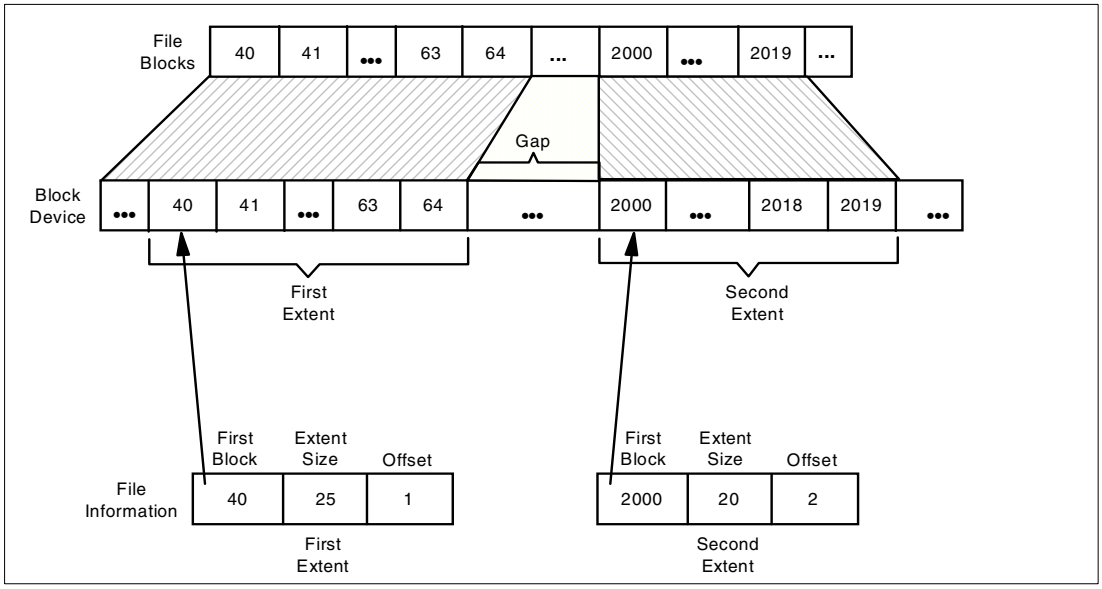


Figure 1-7 File extents

Extents make it possible to locate several free blocks at the same time, reducing the overhead associated with data structures. It is not necessary to use a bit for each block, so the amount of data needed to track free space is no longer proportional to the file system size. Space can be wasted, however, if many extents are equal in size to data blocks, which may be the case if there is excessive file fragmentation.

B+tree data structures organize and index data in a hierarchy which includes two types of node, called internal node and leaf node. All nodes are indexed by a key-pointer pair to allow the tree to be traversed. Pointers are also provided to allow sequential access from one leaf node to the next. B+trees are balanced; this means that the length from the root of the tree to each leaf is always the same. An example is shown in Figure 1-8:

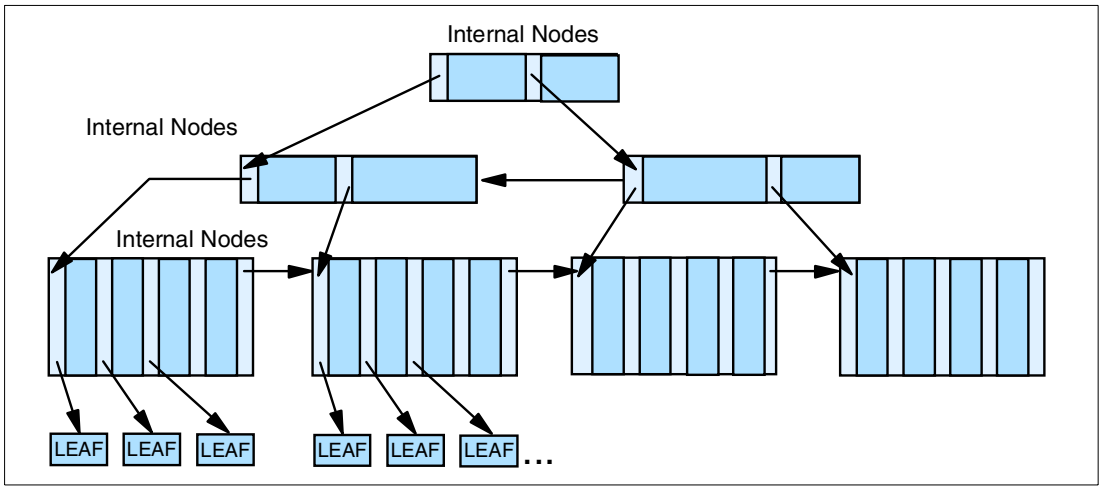


Figure 1-8 B+tree generic structure

The organization of the tree allows for fast indexing of resources (such as files in a directory structure) and preserves sequential access.

1.3.3 XFS

The 1.0 version of this SGI file system was released in May 2001. It is available as a patch against the linux-2.4.2 and linux-2.4.3 versions of the kernel. An installer for Red Hat Linux v7.1 is available from the SGI Web site, and other installers are under development. SGI supplies an ISO CD-ROM image to create a modified version of the Red Hat installer, allowing easy installation of XFS as the root file system.

XFS is a journaled file system that provides fast recovery after a system crash. It allows direct input/output (I/O), bypassing system caching by using direct memory access (DMA). This can mean better performance in some circumstances, by avoiding the overheads of CPU-managed data transfer.

Access control lists (ACLs) and extended attributes are supported and the file system is ready for Data Storage Management (XDSM) support.

Note: ACLs provide more precise control of the permissions than those present in standard UNIX systems. Arbitrary users and groups can be granted or denied access in addition to the three traditional classes of users (user, group, others).

XDSM provides a file system and its host operating system with an appropriate set of functions and semantics to support data management applications.

XFS comes with a toolset that includes a dump and restore utility (with support for all file system features, such as ACL and quotas), a repair utility, a file system editor, tools to grow the file system, and an ACL editing utility. LVM snapshots of XFS file systems are not currently supported.

XFS: scalability and performance

Block address and inode numbers in XFS are 64-bit numbers, as are all the global counters used. The file system is subdivided into allocation groups, which are used to improve the scalability and parallelism of the system, especially in multiprocessor environments.

To avoid performance degradation with increasing file system dimensions, XFS replaces the use of block bitmaps to allocate free space (seen in 1.1.2, “Linux Extended-2 (ext2) file system basics” on page 3) with a combination of extents allocation and B+tree indexing, as discussed in 1.3.2, “Scalability and performance” on page 13.

By replacing a simple directory list technique with the more efficient B+tree indexing algorithm, XFS is able to support a large number of directory entries without incurring unacceptable delays. Good search performance is achieved even for directories containing thousands of files.

XFS’s use of extents and B+trees offers significant performance advantages when handling large files. In comparison, the indirect and double indirect addressing technique that is used by ext2fs (see 1.1.2, “Linux Extended-2 (ext2) file system basics” on page 3) is prone to performance problems as files grow in size. This is primarily a result of the indirection mechanism, which forces multiple disk accesses to locate all file blocks.

XFS also supports sparse files. This support is closely related to extent addressing within the file, using the offset parameter. If a very large file is sparsely populated, unused areas within the file are not allocated until they are used. This reduces the physical storage occupied by the file, making it available for use by other files, and minimizing waste of system resources. If an application tries to read data from the middle of the file, where disk space has not been allocated, the system return a null value since no data is present.

XFS overcomes another problem inherent in the ext2 file system, that of having a fixed number of inodes, determined when the file system is created (see “Linux Extended-2 (ext2) file system basics” on page 3). Your initial choice effectively imposes a limit on the maximum number of files that can be supported in a specific ext2 file system. To avoid this problem, XFS implements dynamic inode allocation, and uses new data structures based on B+trees to keep track of inode locations within each allocation group. This approach adds some internal complexity to the system and means that utilities such as those used for backup and recovery have to understand the new structures.

For XFS installation and utilization examples, refer to 4.5.2, “XFS” on page 65.

1.3.4 ReiserFS

The Reiser file system (ReiserFS) from Namesys has been available in kernel 2.4.x releases since version 2.4.1-pre4. Patches must be applied to some kernel 2.4.x versions for correct use of the file system.

ReiserFS is a journaling file system based on fast balanced trees (B*trees), an enhancement of earlier B+tree algorithms. It minimizes wasted space allocation by using dynamic inode allocation and by packing small files together.

Namesys has a commitment to introduce plug-ins to allow customization of file system objects and methods in the next major release of ReiserFS and it has, quoting from the Namesys Web site, the “*technical foundation [...] for systematically introducing the best features of database and keyword systems into the file system, where all applications in the OS can use them*.”

ReiserFS comes with a toolset including repair and resize utilities.

ReiserFS: scalability and performances

ReiserFS is based on B*trees and organizes every file system object within a single tree structure. Directories, for example, are treated as subtrees of the main tree. This tree is illustrated in Figure 1-9:

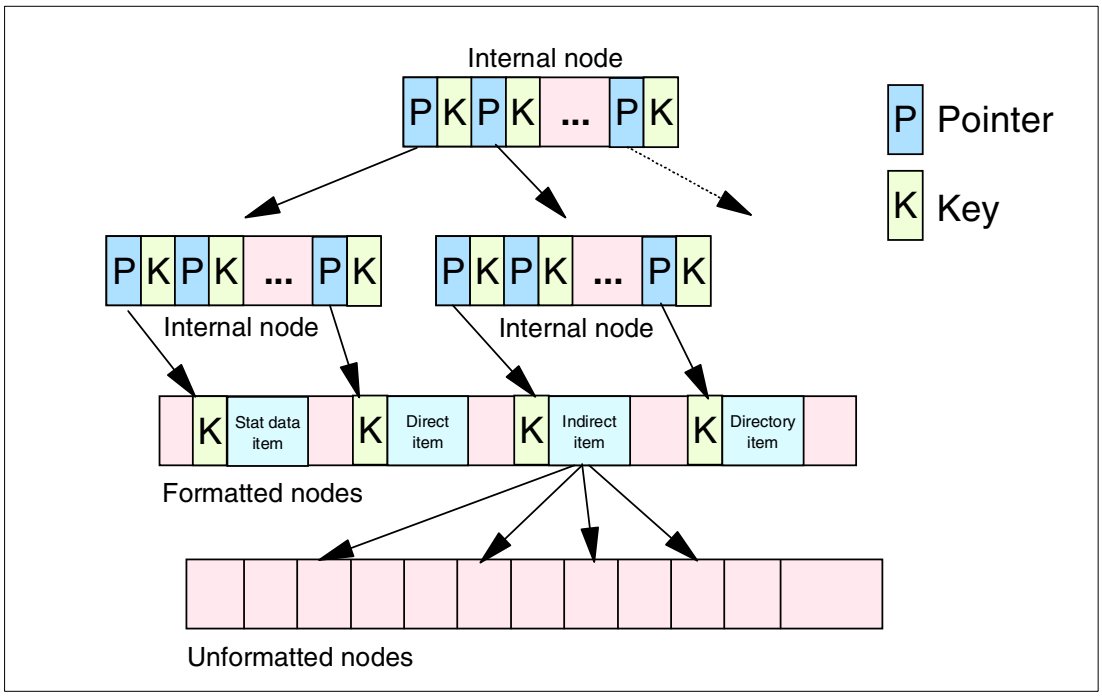


Figure 1-9 ReiserFS tree structure

In the preceding figure, we can see that in the ReiserFS B*tree structure there are three node types:

- Internal nodes

  These consist of sets of keys-pointer pairs, that provide links to other internal nodes or to formatted nodes. The links subdivide the tree into subtrees, and allow the file system to locate the appropriate formatted node for which a request has been made.

- Formatted nodes

  These nodes contain a key which uniquely identifies each one, and data items, which can be of one of four types:

  - Stat data items, which contain most of the information found in an ext2 inode, apart from the address of file blocks.
  - Direct items, which contain the tails of files.
  - Indirect items, which contain pointers to unformatted nodes.
  - Directory items, which contain directory entries.

- Unformatted nodes

  These are logical blocks with no particular format; they are not assigned a key. All file data is stored in unformatted nodes except for the last part of a file, known as the file tail, which is stored in a direct item as part of a formatted node.

Examining the structure of directories and files in ReiserFS a little more closely, we see that:

- Files consist of:
  - a stat data item
  - zero, one or more indirect items pointing to data in unformatted nodes
  - a direct item
- Directories consist of:
  - a stat data item
  - one or more directory items

The file system organization is such that keys, used to identify tree nodes, are used in place of inode numbers. This gives the file system great flexibility, allowing different possible ordering strategies. While the ReiserFS default is to optimize locality of reference and packing efficiency, the next release of the file system is planned to allow overriding of the default packing strategy by using system calls.

The main intent of the ReiserFS developers was to handle large numbers of small files while optimizing space allocation and performance. There has been no particular emphasis on optimizing for large files, beyond increasing the block size. Free space management is accomplished through bitmap usage and there is no use of extents, although extents are also planned for future releases. There is some support for sparse files, but speed issues will have to be addressed in future releases.

ReiserFS distinguishes itself, however, in its handling of small files. Being able to manage blocks up to 64 KB in size produces a significant waste of space when storing small files if no appropriate allocation technique is implemented. With a minimum allocation block of 64 KB, a file of 64 KB plus 1 byte would waste almost 50% of the allocated space. To overcome this problem, ReiserFS stores small files and file tails inside formatted node direct items, as we have already seen.

Using this approach, both files and filenames are stored in the balanced tree, efficiently packing small files, directory entries, stat data and the tails of large files inside formatted node direct items. ReiserFS can arrange the aggregation of files to achieve maximal locality of reference, thanks to appropriate tree ordering methods.

There are some architectural weaknesses of ReiserFS that are a result of this organization, however:

- A tail that grows large enough to occupy an entire unformatted node must be removed from its formatted node and converted into an unformatted node, with implications for system performance.
- A tail can be spread across two formatted nodes, which can create a performance issue if there is poor locality of reference.
- Separation of a file's body from its tail can have performance implications.
- Adding bytes to a file or tail that is in the middle of a formatted node implies data shifts in memory, causing potential performance issues.

Finally, dynamic inode allocation is inherent in the B*tree structure we have described previously, giving ReiserFS great flexibility dealing with huge number of files.

For ReiserFS installation and utilization examples, refer to 4.5.1, "ReiserFS" on page 61.

1.3.5 JFS

After a number of beta releases, v1.0.01 of the IBM JFS is now available as a patch against Linux kernel versions 2.4.0 to 2.4.5.

JFS is a journaled file system that provides fast recovery after a system crash. It provides asynchronous logging, which increases the performance of the file system, and is tailored primarily for the high throughput and reliability requirements of servers, from single processor systems to advanced multi-processor and clustered systems. JFS can also be used in client configurations where performance and reliability needs are high.

JFS supports access control lists (ACLs) and extended attributes, the latter providing a generic storage and access mechanism for data attached to JFS objects.

Tools available for JFS include a repair utility and a defragmentation utility. A resize utility is under test and will be released in the near future.

JFS: scalability and performances

All of the file system data structure fields necessary to provide full 64-bit support are 64-bits in size, allowing JFS to support both large files and partitions. The basic parameters of the file system are given in Table 1-2:

Table 1-2 File system parameters for JFS

Parameters	Size
Minimum file system size	16 MB
Maximum file system size	A function of file system block size: • 512 terabytes (block size 512 bytes) • 4 petabytes (block size 4 KB) In practice, the maximum is the largest file size that virtual file system framework supports.

A JFS file system is built on top of a disk partition, and has a slightly different structure compared to previous file systems introduced. First of all, to support Distributed Computing Environment Distributed File System (DCE DFS), JFS separates the notion of a disk space allocation pool, called an aggregate, from the notion of a mountable file system subtree, called a fileset. There is exactly one aggregate per partition, but there may be multiple filesets per aggregate. In the first release, JFS only supports one fileset per aggregate; however, all of the metadata has been designed for the fully general case.

We can define a fileset as set of files and directories that form an independently mountable subtree. A fileset is completely contained within a single aggregate. When multiple filesets exist within a single aggregate, all of the filesets share a common pool of free aggregate disk blocks as defined by the aggregate control structures.

As in XFS, we have allocation groups that divide the space in an aggregate into chunks, and allow JFS resource allocation policies to use well known methods for achieving good JFS I/O performance.

JFS uses a Block Allocation Map to track the allocated or freed disk blocks for an entire aggregate. Since all the filesets within an aggregate share the same pool of disk blocks, this allocation map is used by all of the filesets within a single aggregate when allocating or freeing disk blocks. When the aggregate is initially created, the data blocks for the map to cover the aggregate space are allocated. The map may grow or shrink dynamically as the aggregate is expanded or shrunk.

The Block Allocation Map is organized as a tree in which the leaf nodes are special bitmaps that keep track of free blocks. JFS uses a technique called Binary Buddy to collect together and track free aggregate blocks.

Two different directory organizations are provided. The first organization is used for small directories and stores the directory contents within the directory's inode. This eliminates the need for separate directory block I/O and the need to allocate separate storage. Up to eight entries may be stored in-line within the inode, excluding the self (.) and parent (..) directory entries, which are stored in separate areas of the inode.

The second organization is used for larger directories and represents each directory as a B+tree keyed-on name. It provides faster directory lookup, insertion and deletion capabilities when compared to traditional unsorted directory organizations.

JFS uses extents organized in B+tree structures to record allocated space for a file, in a way analogous to that described for XFS in 1.3.3, “XFS” on page 15.

In general, the allocation policy for JFS tries to maximize contiguous allocation by allocating a minimum number of extents, with each extent as large and contiguous as possible. This allows for large sequential I/O transfers, resulting in improved performance.

JFS supports sparse files and dynamic inode allocation. In contrast with XFS, inodes are assigned dynamically by allocating inode extents that are simply a contiguous chunk of inodes on the disk.

For JFS installation and utilization examples, refer to 4.5.3, “JFS” on page 68.

More information about JFS can be found at:

http://oss.software.ibm.com/developerworks/opensource/jfs/

1.4 Miscellaneous

Apart from the journaled file systems we have already discussed, several additional file systems are supported like those systems already supported by Linux 2.2.x kernels. In addition, support for some of those previously supported file systems has been enhanced.

In this section, we outline these changes along with other new or improved features related to Linux file systems that have been incorporated into the 2.4 kernel.

1.4.1 File systems and partition tables

The Linux attitude toward openness is reflected in the variety of non-native file systems and partition tables supported. With the 2.4.x kernels, this breadth of support has been developed and improved by the introduction of new file systems and enhanced support for others.

Principal file systems supported by the Linux 2.4 kernel are:

- FAT (File Allocation Table), used on DOS
- VFAT (Virtual File Allocation Table) and FAT32, used on Windows 95/98/Me
- HFS (Hierarchical File System), used on MacOS
- HPFS (High Performance File System), used on OS/2, now including write operations
- NTFS (New Technology File System), used on Windows NT/2000 (this system is still experimental and cannot safely perform write operations)
- SGI’s EFS (Extent File System), in read-only
- UDF file system, used on DVD disks

In addition, the list of partition tables supported by the kernel has been expanded. These now include, among others:

- SGI partition tables
- BSD (Berkeley Software Distribution) disklabel (free BSD partition table)
- Sun partition tables
- Macintosh partition map

1.4.2 Network file systems

In addition to including improvements to local file systems, the 2.4 kernel has introduced improvements for network attached file systems.

Distributed services are widely used in global corporations or small businesses. File sharing in Linux is supported using Network File System (NFS), Server Message Block (SMB), Andrew File System (AFS) from IBM, and others. Key requirements for network file systems are scalability from small to large networks, reliability and speed accessing geographically distributed file resources in a transparent way, security and ease of administration.

NFS

The Network File System (NFS) is a component of Open Network Computing Plus (ONC+) by SunSoft. The Linux implementation has been updated to support version 3 of this protocol, as defined by Sun Microsystems. NFS has evolved to meet the requirements of global enterprises, and now offers a rich set of important features, including:

- Scalability

  Small to large networks are easily supported and configured.

- Transparent and uninterrupted access to global files through automatic mounting tools

  Users and applications do not need to know whether files reside on a local disk or a remote system.

- Local disk caching and improved performance

  Performance differences when accessing remote files versus local files are minimized.

- Information protection is provided by security architecture

  Multiple security mechanisms are available, allowing you to choose the one best suited to the task at hand.

- Support for diskless client systems
- Network protocol independence

  NFS runs on multiple transport protocol layers.

- Portability

  NFS is machine- and operating system-independent.

Improvements to NFS that have been incorporated into the 2.4 kernel include better integration with other operating systems' implementations of NFS and the use of reliable file locking of NFS partitions. This is one of the most important features of NFS for enabling heterogeneous systems to share their resources.

With support for NFS Version 3, the 2.4 kernel offers better compatibility and many advantages over NFS Version 2 (supported by Linux kernel version 1.2). The NFS Version 3 protocol was defined in 1992 by a group of vendors that included IBM, Digital, SunSoft, and others. Enhancements include:

- Improved client write throughput through safe asynchronous writes.

  The NFS version 3 client collects multiple requests and writes the entire data at once through the server's cache, reducing the total number of write operations.
- Reduced server load, resulting in increased scalability and performance.
- Support for Access Control Lists (ACLs).
- Support for large files on NFS servers.

1.4.3 Samba 2.2

The Samba suite is a piece of free software that implements the Server Message Block (SMB) protocol, also known as Common Internet File System (CIFS). SMB is in common usage, in conjunction with NetBIOS, in Windows environments, to share files and resources through the network.

The current version (Samba 2.2.x) contains many improvements and additions in comparison to its predecessor. Samba is available for many UNIX popular platforms (including AIX, BSD, HP Unix, Novell and many others). For Linux systems in particular, some new features that take advantage of the 2.4 kernel functionality, either directly or through patches.

First, Linux 2.4 is able to take advantage of the new Samba oplock support. Oplocks are a locking mechanism in the SMB protocol that can improve file sharing performance. The previous release of Samba did not work correctly with oplocks, since Linux and Samba caching were independent mechanisms; this resulted in file system inconsistencies. To achieve system consistency, it was necessary to disable oplocks, causing performance loss.

Using Linux 2.4 kernel with Samba 2.2 solves the problem, allowing optimum Samba performance whether files are accessed locally, using an SMB share, or in other ways.

In addition, Samba 2.2 can handle Access Control Lists (ACLs) that Windows NT and 2000 use to set permissions on files and directories. ACLs offer much closer control over permissions than found in standard UNIX systems (see 1.3.3, "XFS" on page 15). ACLs are supported in the 2.4 kernel by applying a patch available from:

http://acl.bestbits.at/

Applying this patch enables Samba ACL support under Linux. Samba will then preserve NTFS ACLs rather than mapping the permissions to the less flexible standard UNIX permission scheme, thereby achieving better Linux integration in Windows networks.

File systems such as XFS and JFS also offer support for ACLs, as we saw in 1.3, "Journaled file systems" on page 11).

1.4.4 Raw I/O devices

The 2.4 kernel introduces a new feature, raw I/O devices, to the Linux operating system. As we have seen in 1.1.3, "Block, character and network devices" on page 6, the system interacts with devices through a complex mechanism mediated by the kernel. This mechanism includes a caching layer.

Disk I/O caching results in better performance for frequently accessed data, but requires system resources to transfer data from kernel buffers to application memory. This can throttle I/O throughput, however, for database operations or for large data movements that do not benefit from caching. Another potential drawback with the standard access methods is that applications do not have complete control over data caching algorithms or the transactional reliability of data written to disk.

Previous versions of raw device support were not suitable for inclusion in the Linux kernel, because they required doubling the number of device nodes (every block device also had to have a device node). Many commercial UNIXs still use this implementation. The current raw device implementation uses a pool of device nodes, which can be associated with any arbitrary block device, reducing the number of device nodes required.

For RAW I/O installation and utilization examples refer to 4.6.3, “Raw I/O” on page 74.

2

New network features

In this chapter, we introduce some of the most interesting features related to networking that have been introduced in the Linux 2.4 kernel.

The Linux kernel has gone through a complete rewrite of the networking layer, involving major improvements to the overall performance and stability of the system. We describe the new embedded firewalling and NAT abilities, the wake one network-socket model, and the completely new kernel web daemon: khttpd. Other features, such as support for Asynchronous Transfer Mode (ATM) and DECNet protocols, and new hardware support, are briefly introduced.

2.1 Linux networking architecture overview

We can say that Linux was born thanks to the Internet, and that its ongoing development has always been deeply related to the Internet in particular and networking in general. Since its early days, the Linux kernel has included a BSD Sockets Interface, and support for more and more classes of sockets and protocols have been added in the ensuing years.

The 2.4 kernel network stack has been improved to support better parallelism in multiprocessor (SMP) systems with the introduction of the new softnet architecture. As we saw earlier (see 1.4.2, “Network file systems” on page 21), Network File System Version 3 (NFSv3) is now supported and there is also better support for the Simple Network Management Protocol (SNMP).

To help in our discussion of the 2.4 kernel’s enhanced network abilities, we begin by briefly outlining the TCP/IP suite implementation found in the kernel code.

2.1.1 Sockets

Linux bases its networking abilities around a general software interface called the BSD Socket Interface. This name comes from the first networking version of UNIX, 4.3 BSD (Berkeley System Distribution), upon which Linux’s networking implementation was modelled. It supports an extended version of BSD sockets and various network protocols including ATM, DECnet and TCP/IP (being compliant with the IPv4 specification).

Note: The network subsystem in the kernel code has been totally revised. Linux networking is now multithreaded. This means that the code has been modified to be deserialized as much as possible to allow better scaling on symmetrical multiprocessing (SMP) systems.

The BSD Socket Interface gives Linux an abstraction layer over the different underlying socket classes, known as address families. This abstraction level allows any application in userspace to operate on socket communication channels in a standard way. The exact operations needed to manage a specific BSD socket depend on its address family, but the kernel performs those operations in a completely transparent way. This approach is very similar to what we have seen with regard to file systems (see 1.1, “Kernel and file system overview” on page 2).

The relationship between the various components is illustrated in Figure 2-1:

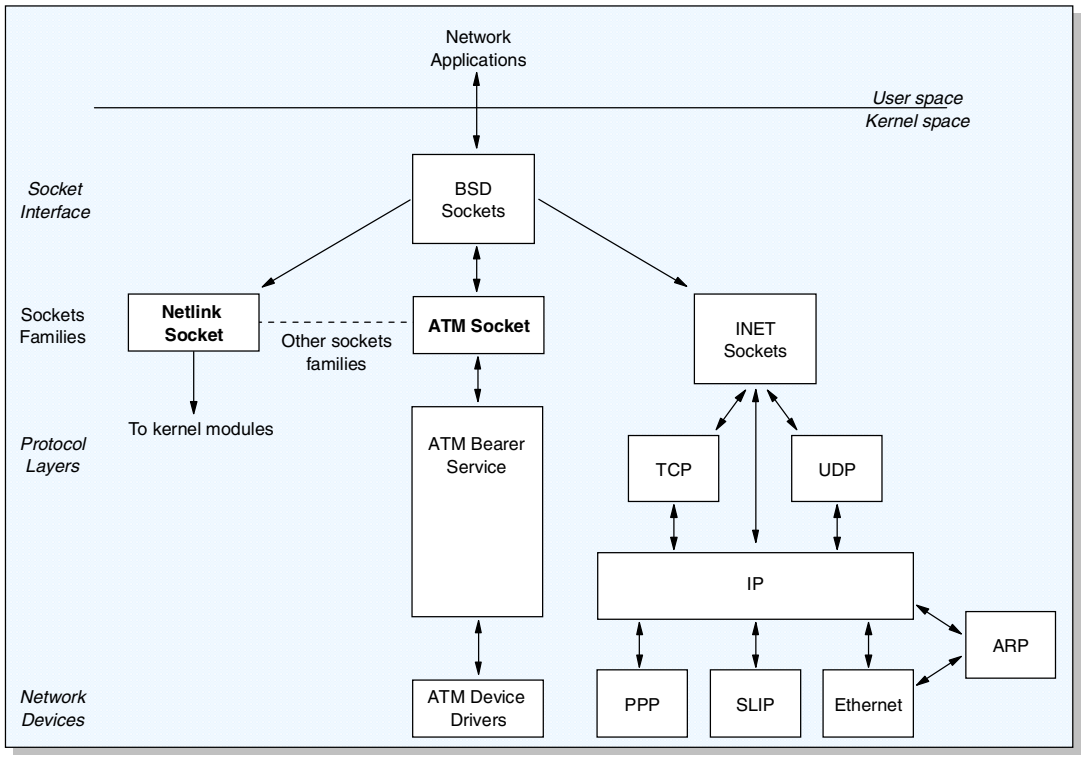


Figure 2-1 Network layer structure

Sockets are used for both networking and interprocess communications. A socket is treated by the Virtual File System (VFS) as a particular kind of file (see 1.1, “Kernel and file system overview” on page 2) and is actually one end of a connection between processes. Each communicating process must bind to a socket describing the link to its counterpart.

The inode associated with a socket is described by an inode data structure that has a particular extension embedded in it; this is the socket data structure, shown in Figure 2-2:

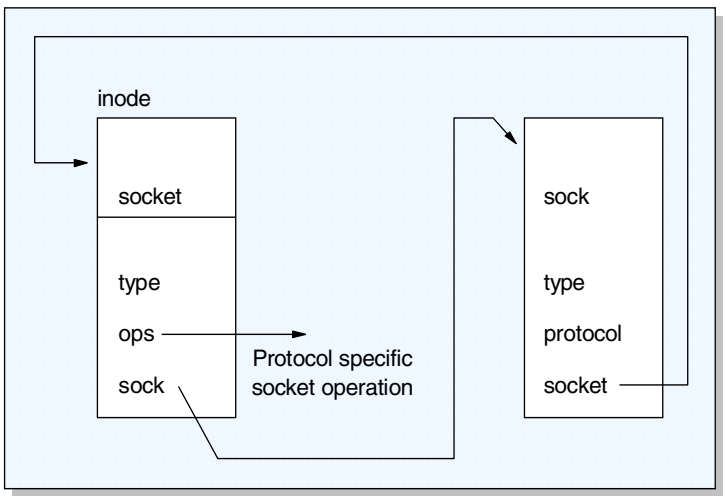


Figure 2-2 Socket data structure and inode

Among other fields, this socket structure specifies the type of socket involved, the routines that should be used to manage the socket, which depend on the socket address family, and a link to a different structure (sock) that specifies protocol information and the routines the kernel has to call to deal with the particular socket at hand. When a socket is created by a process, the socket domain, type and protocol must be specified.

The socket domain indicates one of the address families supported by Linux. INET is the most used protocol family, and is used to implement ARPA Internet Protocols (such as TCP, UDP and IP). The NETLINK address family is used by Linux to transfer information between user space applications and kernel modules. A sample taken from the kernel source shows a complete list of socket domains:

```
#define AF_UNIX         1       /* Unix domain sockets          */
#define AF_INET         2       /* Internet IP Protocol         */
#define AF_AX25         3       /* Amateur Radio AX.25          */
#define AF_IPX          4       /* Novell IPX                   */
#define AF_APPLETALK    5       /* AppleTalk DDP                */
#define AF_NETROM       6       /* Amateur Radio NET/ROM        */
#define AF_BRIDGE       7       /* Multiprotocol bridge         */
#define AF_ATMPVC       8       /* ATM PVCs                     */
#define AF_X25          9       /* Reserved for X.25 project    */
#define AF_INET6        10      /* IP version 6                 */
#define AF_ROSE         11      /* Amateur Radio X.25 PLP       */
#define AF_DECnet       12      /* Reserved for DECnet project  */
#define AF_NETBEUI      13      /* Reserved for 802.2LLC project*/
#define AF_NETLINK      16
#define AF_PACKET       17      /* Packet family                */
#define AF_ASH          18      /* Ash                          */
#define AF_ECONET       19      /* Acorn Econet                 */
#define AF_ATMSVC       20      /* ATM SVCs                     */
#define AF_SNA          22      /* Linux SNA Project (nutters!) */
#define AF_IRDA         23      /* IRDA sockets                 */
#define AF_PPPOX        24      /* PPPoX sockets                */
#define AF_WANPIPE      25      /* Wanpipe API Sockets */
#define AF_MAX          32      /* For now.. */
```

The sample is taken from the Linux/include/linux/socket.h file of the 2.4.2 kernel source distribution (some lines have been deleted for sake of clarity). Some of the new networking abilities introduced with the 2.4 kernel are indicated in these lines of code. We can recognize the support for ATM and DECnet among the families, for example.

Returning to the data structure in Figure 2-2 on page 27, the type field specifies the type of socket requested:

- A stream connection socket to provide a reliable two way data stream, with no packet loss and no corruption or duplication of packets. For the INET family, this is supported by the TCP protocol.
- A datagram connectionless socket to provide two way data transfer, with no guarantees that packets will be successfully transferred. For the INET family, this is supported by the UDP protocol.
- A raw socket that provides direct access to underlying protocols.
- Reliable Delivered Messages. This provides two way data transfer of datagrams with guaranteed arrival of data.
- Sequenced Packets, providing a stream connection with fixed packet sizes.
- A packet type, which is a Linux-specific extension of BSD sockets for accessing packets directly at the device level.

The kind of protocol chosen depends on the family and type selected for the socket. For the INET family, a process can, for example, choose TCP for a stream socket and UDP for a datagram socket. All of the supported protocols are listed in the file /etc/protocols.

INET protocol stack

The Internet address family contains the TCP/IP protocol suite. This suite was introduced during the first ARPANET project and is the de facto standard in today's Internet communications.

The TCP/IP protocol suite can be considered structured in layers. TCP/IP's structure does not correspond directly to that of the Open System Interconnect (OSI) Reference Model designed by the International Standard Organization (ISO). The OSI Reference Model is designed to define network activities in conceptual terms by defining a seven layer structure, where one or more protocols are associated with each layer. TCP/IP joins together some OSI layers in a single layer, and simply ignores other OSI layers.

We have summarized the TCP/IP layered structure in Table 2-1.

Table 2-1 TCP/IP layers and functions

TCP/IP layer	Examples	Description	OSI equivalent
Application	Telnet, ftp, NFS, DNS, and so on.	Standard Internet services and applications.	Application Session Presentation
Transport	TCP, UDP.	Ensures end-to-end, connection-oriented communications.	Transport
Internet	IP, ARP, ICMP	Accepts and delivers packets for the network (unique addressing, routing, fragmentation, error reporting, etc.).	Network
Network access	PPP, SLIP, MAC	Logical interface with network hardware (encapsulates IP datagrams into frames, maps IP addresses to physical addresses used by the network).	Data link
Physical	Ethernet, Token Ring, RS232	Hardware standards specifications (pin connectors, cables, and so on).	Physical network

The Linux TCP/IP protocol stack reflects this layering in its code and data structures. BSD socket layer routines call the INET layer routines that were registered during network initialization. These allow the BSD socket layer to create a socket for the INET address family, through which it can receive and transmit data, or listen, waiting for connection requests.

Suppose we have a client-server application such as an Internet browser connected to a Web server. To make this connection possible, the server must create an INET BSD socket and bind it to an address. This address consists of an IP address (usually assigned to a network interface present in the system) and a port number. The address is added to the sock data structure linked to the socket, and is used by the TCP layer to route each incoming packet to the appropriate INET and BSD socket. The server can now use the socket to listen for inbound connection requests.

Once the client has created its own socket, it can be used to make outbound connection requests. When a connection is established, the client sends and receives data to the server through an INET socket connection using TCP/IP protocols. The transmitted data passes through the BSD socket layer, and goes on to the underlying INET socket interface, with received data traveling in the opposite direction.

Each protocol layer in the INET address family needs to add its own header and tail to the data packets, and removes them when processing received data. For this reason, Linux implements a particular kind of buffer called socket buffer (skb), and that is where it stores packet data and header and tailer information.

2.2 Linux 2.4 firewalling and Network Address Translation

Linux 2.4 introduces improved packet filtering, mangling and Network Address Translation (NAT) through a new framework called netfilter. It adds stateful inspection of packets and a completely revised packet selection subsystem called IP Tables.

The complete rewriting of the packet filtering and routing infrastructure within the kernel allows for increased security and flexibility when dealing with network firewalling and NAT. With the iptables application, Linux has improved the ability to manage the new features provided by the netfilter infrastructure.

2.2.1 Netfilter framework

The Linux 2.4 kernel introduces the netfilter framework for packet mangling, filtering and Network Address Translation (NAT). Netfilter operates by mean of hooks placed in the network protocol stacks.

As we have mentioned before, each packet traverses the protocol stack to pass through a socket connection. Netfilter hooks are positioned at well-defined points along the path a packet takes through specific protocol stacks. Currently, support is implemented for IPv4, IPv6 and DECnet protocols. When one of these points is reached, the protocol invokes the netfilter framework, passing it information about the packet and the hook number.

Kernel modules can register to listen to the different hooks. When a particular netfilter hook is activated, all the modules registered for it are called, following a priority order. Each module, when called, can examine the packet and ask netfilter to do one of the following:

- Allow the packet to traverse with no intervention
- Drop the packet, stopping its traversal
- Forget about the packet, stopping its traversal
- Queue the packet, which is used for user space handling
- Call this hook again

On top of netfilter, there are currently four major subsystems. These are:

- The iptables packet classification system
- The connection tracking system
- The Network Address Translation system
- The ipchains and ipforward classification system (for backward compatibility)

Netfilter hooks

We now examine the IPv4 protocol to understand how netfilter works.

In the IPv4 protocol stack, we have 5 hook points (see Figure 2-3):

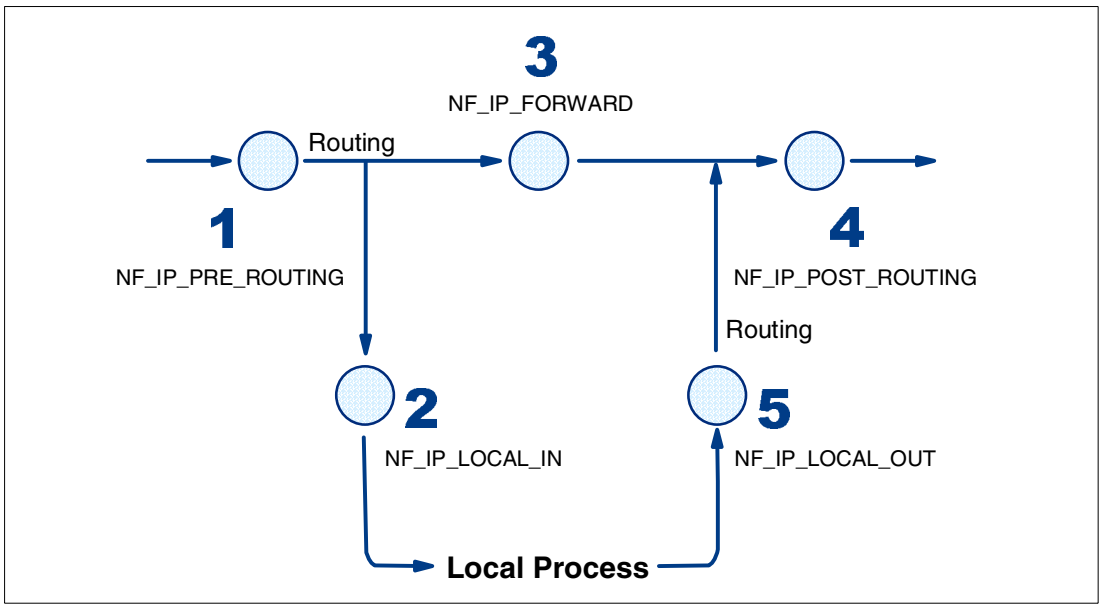


Figure 2-3 Hook points in IPv4 protocol stack

These hook points are named as follows:

1. IP pre routing (NF_IP_PRE_ROUTING)
2. IP local in (NF_IP_LOCAL_IN)
3. IP forward (NF_IP_FORWARD)
4. IP post routing (NF_IP_POST_ROUTING)
5. IP local out (NF_IP_LOCAL_OUT)

In Figure 2-3, we see that packets come from the network (from the left), passing through hook 1. Upon reaching the routing code (IP layer), they are dispatched to the local host through hook 2 or to another network destination (leaving from the top right) through hooks 3 and 4. Packets created locally (from the bottom) go through hook 5 and are routed to other networks after going through hook 4.

We will follow the path of a packet forwarded by the local host through the INET address family protocol stack. We can follow the path in Figure 2-1 on page 27, and the layers it traverses in Table 2-1 on page 29.

The network card receives an Ethernet frame, assuming we are connected to an Ethernet network. The frame MAC address matches the Ethernet card MAC address, making the card issue an interrupt. The kernel network driver transfers the frame to system memory and allocates a socket buffer (skb) to store frame data. We recognize the TCP/IP Network Access layer in this operation, and we are executing at the kernel network device driver level.

The socket buffer is queued on the receive queue and a soft interrupt (softirq) is issued.

Note: This is one of the most important changes in the overall network stack implementation. The Bottom Half mechanism has been replaced with softirq, enhancing parallelism on SMP systems.

When a system call is finishing or when a hardware interrupt handler exits, the CPU runs all software execution requests that are queued to it. The original mechanism implemented by the Linux kernel to handle these queued processes was called Bottom Halves. The problem with this method is that it does not take advantage of multiple CPUs. No more than one Bottom Half can run simultaneously, no matter how many CPUs the system can support.

Starting with the development kernel version 2.3.43, and now part of the 2.4 kernel, a new mechanism has been introduced, called softirqs. Softirqs are fully-SMP versions of Bottom Halves that can run on as many CPUs at once as required.

The network receive softirq is registered and, when detected, causes the kernel to dequeue the skb to be handled by the IPv4 packet handler. The packet has now been transferred to the TCP/IP *Internet layer* corresponding to the IP layer of the INET address family stack protocol. The IPv4 handling function performs some initial checks on the packet. If any of the checks fail, the packet is dropped.

If the checks are successful, the first netfilter hook is called. At this point, the registered kernel modules are invoked. They may then determine whether the packet will continue on its path, be dropped, or be dispatched to a user-level application. Registered kernel modules at each hook point can make this decision, as they are invoked through the rest of the process that follows.

After netfilter hook traversal, the packet route is determined by the IP routing function, and all the IP options are processed.

The packet can now be directed to its destination:

- For a local destination:
  - The second netfilter hook is called
  - The transport layer protocol is processed by the appropriate INET address family stack layer (TCP or UDP) functions
  - The skb is then passed on to a user space process
- For a remote destination, the packet has to be forwarded to another network; in this case, the packet does not reach the local transport layer.
  - The packet's Time To Live (TTL) is checked and, if it is less than one, the packet is dropped and a `time exceeded` message is returned to the sender.
  - TTL is decremented by one.
  - Fragmentation code is executed if the Maximum Transfer Unit (MTU) parameter of the destination network is exceeded.
  - The third netfilter hook is called.
  - Final checking on the packet is done.
  - The packet is sent to the appropriate device driver after the fourth netfilter hook is called.
- If an appropriate routing table entry was not found, an error message is issued.
- If the packet is a multicast one, the appropriate routing will be undertaken.

The fifth netfilter hook is called only if the packet is generated locally and moves from the user space or kernel space toward the bottom of the stack. It is called before routing is applied by the IP layer. After the routing decision has been made, the packet passes through the fourth netfilter hook before being sent.

2.2.2 IP Tables

IP Tables is a packet selection subsystem based upon the netfilter framework. It follows from ipchains (from kernel 2.2), ipfwadm (kernel 2.0) and from the original BSD's ipfw. IP Tables is a flexible and extensible tool that interacts with netfilter to apply tables of rules through which the packets have to pass.

As we saw before, each kernel module can register for a particular hook point in the netfilter framework, and can inspect and alter the packets traversing that point. At the same time, a module can register a new table and match each packet it receives from netfilter against the registered table.

IP Tables provides an array of rules in memory with information for each type of packet, defining which rules to go through. The rules are arranged in chains and can be modified or extended by user space applications, using specific function calls. The main application to manage tables and chains of rules is named iptables.

In the following paragraphs, we give an overview of the characteristics of IP Tables. This is not intended to be an iptables tutorial, but it emphasizes the improved packet filtering abilities available through iptables.

Actually, three instances of IP tables are implemented in netfilter itself; these are:

- filter
- NAT
- mangle

Filter table

This table only filters packets, never changing them in any way.

In Figure 2-4 on page 34, we see that the filter table hooks into three netfilter points: NF_IP_LOCAL_IN (point 2), NF_IP_LOCAL_FORWARD (point 3) and NF_IP_LOCAL_OUT (point 5).

Each packet passes only one of these hooks. Specifically:

- Locally created packets traverse NF_IP_LOCAL_OUT (point 5)
- Packets whose destination is local traverse NF_IP_LOCAL_IN (point 2)
- Forwarded packets traverse NF_IP_LOCAL_FORWARD (point 3)

There is a chain of rules connected to each hook:

- The INPUT chain is connected to NF_IP_LOCAL_IN (point 2)
- The OUTPUT chain is connected to NF_IP_LOCAL_OUT (point 5)
- The FORWARD chain is connected to NF_IP_LOCAL_FORWARD (point 3)

The names used are the same as those used in the 2.2 kernel ipchains facility, but the overall logic of packet traversing has changed.

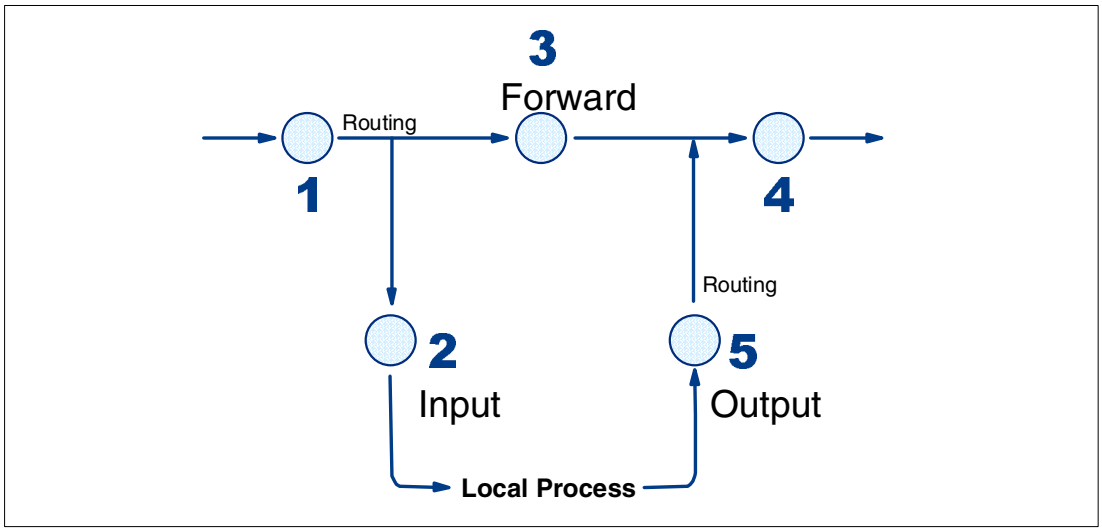


Figure 2-4 Filter table hooks

Each packet goes through a chain until it matches a rule. At this point it can be dropped, accepted or processed somehow, depending on the matched rule's target (discussed further on). If no match is found, the default policy for the chain, which is usually set to drop the packet, is executed.

Setting rules is a simple task, using the **iptables** command and associated flags. Some examples are:

```
iptables -t filter -A INPUT -s 127.0.0.1 -j DROP #Adds a rule at the to INPUT chains
iptables -t filter -D FORWARD 2 #Deletes the second rule from the FORWARD chain
iptables -t filter -P OUTPUT DROP #Sets the default policy for OUTPUT chain to drop packts
```

For a full explanation of the **iptables** command and its options, refer to the man pages.

As we have already mentioned, IP Tables relies on other modules to feed it the packets as appropriate, and deals exclusively with chains of rules. Each rule in a chain is composed of one or more matches and a target.

Filter targets

The Targets field tells netfilter how to manage the matched packet. IP Tables has four standard built-in targets (DROP, ACCEPT,RETURN and QUEUE), but new target extensions can easily be added, or a user-defined chain can be specified as a target. Here are some example where we create a new chain (user_chain) and jump to it from the INPUT chain:

```
iptables -N user_chain #create a new chain called user_chain
iptables -A user_chain -d 9.87.36.1 -j DROP #adds a rule to the new user_chain
iptables -A INPUT -p tcp -j user_chain #jumps from INPUT to user_chain for TCP packets
```

User-defined chains can jump to other user-defined chains, but you should take care to avoid loops that will cause packets to be dropped.

The possible targets are:

- Built-in targets:
  - DROP: drop the packet without any communication to the sender
  - ACCEPT: accept the packet
  - RETURN: go to the end of the chain (for built-in chains, the default policy is executed; for user-defined chains, the packet continues from the calling chain's next position)
  - QUEUE: queue the packet for user space processing (this sends the packet to a user space application)
  - USER_CHAIN: jump to a user-defined chain
- Standard extension targets:
  - LOG: log the packet using syslog and continue traversal of the chain
  - ULOG: log the packet with a user space logging application
  - REJECT: drop the packet and send a `port unreachable` ICMP error packet to the sender
  - MIRROR: exchange source and destination, and resend the packet (this is actually a mangling action and is for testing purposes)

Filter targets can be extended. Target extensions consist of a kernel module and a iptable expansion to deal with new targets provided.

Matches

A match specifies the characteristics that determine whether or packet should be selected for handling by IP Tables. The basic types of matches are based on:

- Source and destination IP address (`-s` and `-d` flags of the **`iptables`** command)
- Protocol used (`-p` flag):
  - Physical interface on which the packet came in or is going out (for example, `-i eth0` specifies the first Ethernet adapter).
  - Whether or not the packet is a fragment (`-f` flag). Fragmentation occurs when a packet is too large to fit in a particular network frame size.

The IP Tables packet selection system is extensible.

State extension (stateful inspection)

State extension bases its packet matching upon the connections to which packets belong. This allows stateful inspection of packets, also referred to as stateful filtering or dynamic filtering, and it is one of the most useful filtering tools available.

Previously (in the 2.2 kernel), it was necessary to read each packet 's header in order to know if it was part of an existing connection (for example, a connection started by a client inside the internal network). Thus the packet itself had to be trusted, allowing an attacker to use malicious packets, with appropriate header flags set, to overcome the protection.

With the stateful inspection tracking mechanism, it is now possible to associate all packets that come into the system with a particular connection. This is possible because the connection tracking code maintains a list of current connections in memory. This memory usage is also present when using NAT, and implies no additional memory costs beyond NAT's.

Although stateful inspection is based on connection tracking, it is useful even for a connectionless protocol such as UDP. If, for example, a Domain Name Service (DNS) request has to be performed outside the protected network, UDP packets are used. The stateful firewall can keep track of all outgoing DNS requests, allowing only DNS packets from queried hosts to pass through.

Another important advantage of stateful firewalling is that it is easier to administer, because fewer rules have to be specified to achieve the desired results.

NAT table

Network Address Translation allows a router to modify packets' source and/or destination information when they pass through it. NAT functionality is divided between the connection tracking framework and the NAT code itself. We have talked about connection tracking in the previous section.

In Figure 2-5, we see that the NAT table hooks into three netfilter points: NF_IP_PRE_ROUTING (point 1), NF_IP_POST_ROUTING (point 4) and NF_IP_LOCAL_OUT (point 5).

There is a chain of rules connected to each hook:

- the PREROUTING chain is connected to NF_IP_PRE_ROUTING
- the OUTPUT chain is connected to NF_IP_LOCAL_OUT
- the POSTROUTING chain is connected to NF_IP_POST_ROUTING

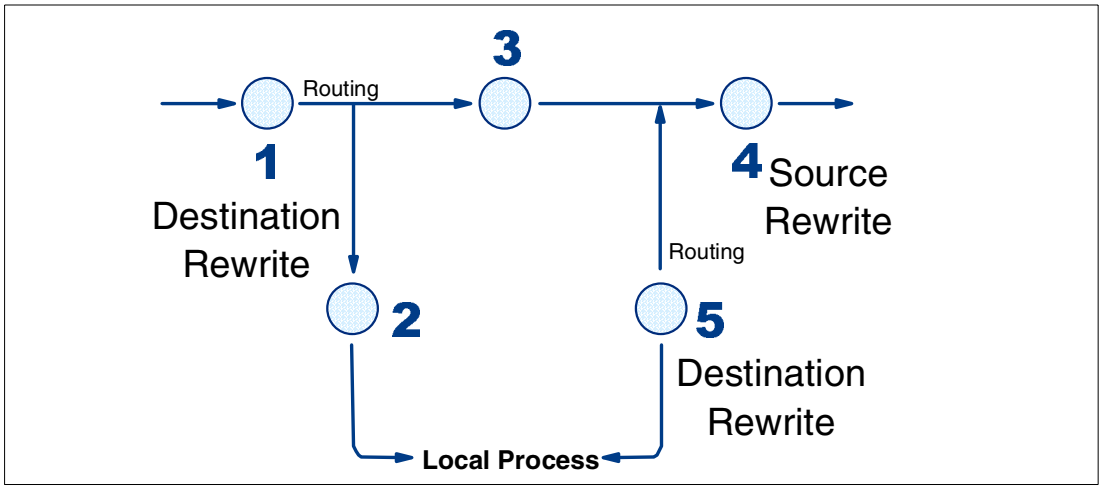


Figure 2-5 NAT process

NAT could be useful for several purposes. We can see some examples of its use:

If our network has a unique connection point to the Internet, that is, a single public IP address, all response packets have to be directed through that IP address. If a host in the local network tries to connect to an Internet host by sending packets which have the local host's own internal (private) IP address, the connection will fail because the private address would be considered invalid by Internet routers.

In this case, source NAT (SNAT) can be used. Using this technique, all outgoing packets that pass through the firewall/routing host connected to the Internet are modified. The firewall will change the source address for each packet, replacing it with its public IP address.

The destination host believes that the connection is requested by the firewall/router. From that moment on, the firewall will track each incoming packet belonging to the translated connection and will forward the packet, modifying its destination IP address to match that of the appropriate internal machine.

Another application of NAT is transparent proxying. If we have a proxy application on our router, we can make destination NAT (DNAT) redirect all incoming packets to the local proxy application. This way, the proxy would be completely transparent.

NAT targets

For the filter table, we have different types of targets we can specify using NAT.

The targets presently available are:

- SNAT (source NAT)
  - Performed by the NF_IP_POST_ROUTING hook, it alters the source address of outgoing packets.
  - Masquerading is a special case of SNAT; it operates on dynamically assigned IP addresses, such as for dial-up connections. Connection information is discarded when the interface goes down.
- DNAT (destination NAT)
  - Performed by the NF_IP_PRE_ROUTING hook and, rarely, by the NF_IP_OUTPUT hook, it and alters the destination address of incoming packets.
  - Redirection is a specialized case of destination NAT; it is the same as applying DNAT to the address of the incoming interface.

NAT targets can be extended in a similar way to filtering targets. Connection tracking and NAT have been designed to be extensible to accommodate any number of protocols and mapping types. It would be possible, for example, to create a specific mapping to implement load balancing techniques or failover algorithms.

NAT and packet filtering

It is worth noting that NAT and packet filtering mechanisms do not interfere with each other. In fact, DNAT occurs before the actual routing and packet filtering take place, and SNAT occurs after routing and packet filtering have been performed.

Mangle table

Packet mangling is the ability to modify a packet's information in various ways.

In Figure 2-6, we see that the mangle table hooks into two netfilter points: NF_IP_PRE_ROUTING (point 1), and NF_IP_LOCAL_OUT (point 5):

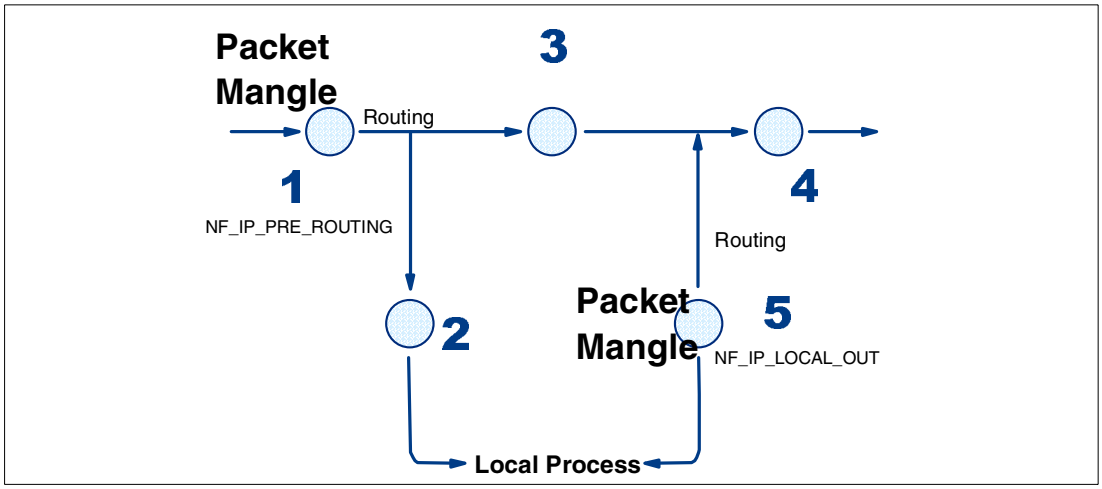


Figure 2-6 Packet mangling

Packet mangling could, for example, modify the priority of certain connections, changing the Type of Service value in the packets.

Mangle targets

There are three type of targets currently implemented:

- TOS (Type of Service): allows the TOS field to be changed in the packet header; this could be Minimize-Delay, Maximize-Throughput, Maximize-Reliability, Minimize-Cost, Normal service.
- TTL (Time to Live): allows for incrementing, decrementing or setting to a precise value the TTL field in the packet header.
- MARK: allows modification of a packet's skb associated buffer, setting the nfmark field to a particular value.

In the other tables, target extension is possible by writing new modules to define actions to be taken on packets and new shared libraries, so as to provide command line options for iptables.

Queuing and user space logging

As we saw previously (see "Filter targets" on page 34), one of the possible targets available in the filter table is QUEUE. When a rule has this target, the packets that match the rule are inserted into a special queue inside netfilter. The queued packets are transmitted to a user space process, using a netlink socket (see 2.1.1, "Sockets" on page 26).

The user space process can examine, modify, drop or accept the packet. It can also insert the packet back at the exact point it came from. This is an important extensibility feature that allows you to execute firewalling code in user space, and also helps to keep some complex code outside the kernel.

Very similar to the queuing ability is user space logging. It is implemented through the ULOG filter target and is unidirectional, that is, the packet matching the appropriate rule is copied and sent to user space through a netlink multicast socket. One or more user space applications can be listening to this multicast socket and can receive a copy of the packet.

Each application can therefore collect all of the information it needs from the packets and log them to an appropriate logfile or database.

2.3 Miscellaneous

Other improvements in Linux networking abilities include the integration of a Web server within the kernel space (khttpd), the wake-one strategy for network sockets management, and support for new network devices and protocols, including Asynchronous Transfer Mode (ATM), ARCNET and DECnet.

2.3.1 Kernel Web server and wake-one

The 2.4 release of the Linux kernel has largely focused on improving overall system performance. Different parts of the kernel have been rewritten in order to overcome possible bottlenecks. Examples of improvements to areas affected include enhancements to scheduling and threading, better SMP support, and the deserialized network stack implementation. The integration of a Web server within the kernel space and the new wake-one policy are also intended to offer performance improvements.

khttpd

One of the interesting new features in the Linux 2.4 kernel is the implementation of a kernel Web daemon called khttpd, which runs in the kernel as a module. The daemon can handle only static pages, that is, pages that contain only HTTP and picture files held on the server, and must pass all non-static requests, such as CGI scripts, Java server pages, etc., to a Web server in user space, like Apache or ZEUS.

Therefore, Linux can now process static HTTP requests more efficiently. khttpd can take advantage of its kernel space implementation, speeding up the serving of static pages and freeing the user space Web server from trivial file transfer operations. The result of this is, in many cases, an overall performance increase.

The present khttpd architecture is based on a single thread per CPU handling all requests. Each request is placed in a request queue. Let us assume a particular request is waiting for a connection. When the connection is established, the request's header is decoded and a decision is made. If the request is for a static page, the corresponding data is sent and logging operations are performed. If a dynamic page is requested, the request is passed to a user space Web server. This process is illustrated in Figure 2-7:

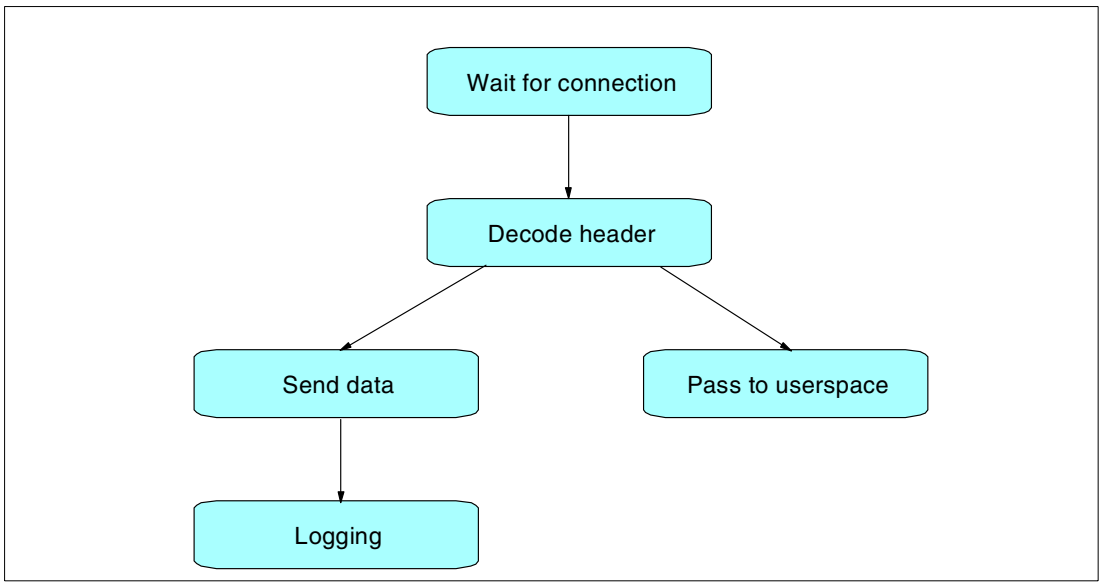


Figure 2-7 Deciding whether khttpd or another Web server should handle a request

All requests are grouped together by state. Each group is serviced one at a time, starting with requests that are pending connections through to those waiting for logging operations. This method is used to enhance efficiency and to realize a modular design. Each of the states in which a request can exist is served by a different procedure that can be changed without interfering with the other parts of khttpd.

We have to point out that khttpd is one of the more controversial features of the 2.4 kernel. It is not yet clear what real performance improvements are achieved, even though many benchmarks have been performed. For examples of the type of questions surrounding this feature, you may want to visit these two URLs:

`http://lwn.net/2001/0118/a/cl-khttpd.php3`
`http://lwn.net/2001/0118/a/cl-khttpd-persistent.php3`

Another kernel daemon Web server has been introduced by Red Hat: Threaded linUX http layer (Tux). It has functionality similar to khttpd and can be found at:

`http://www.redhat.com/products/software/webservers/tux/`

Wake-one

The UNIX network-sockets model has been changed in the Linux 2.4 kernel. Using the 2.2 kernel, processes waiting for an event from a network socket were all woken up at event detection. Only one succeeded in getting the event, and the rest had to put themselves back to sleep. Linux is an efficient system and does this quickly, but eliminating this stampede effect increases network speed significantly.

The new model is called wake-one. As you might suspect, this new code wakes up only one process when an event occurs and allows server applications (such as Apache) to be more efficient, making Linux an even better choice of Internet server.

2.3.2 DECnet

DECnet is a protocol stack that was originally designed by Digital Equipment Corporation (now subsumed within Compaq) to interconnect its products. It has had different implementations, and the specification for DECnet Phase IV has been made freely available by Compaq. This has allowed open source development of a DECnet implementation for Linux systems.

DECnet is an old standard that can support a limited address space and lacks modern networking advanced abilities. Nonetheless, it is still used in many legacy systems and has strategic relevance for Linux integration with those systems.

Although DECnet could be implemented over many different link layers, ranging from PPP to X.25, it is currently implemented only on Ethernet and Generic Routing Encapsulation (GRE) IP tunnels; the latter is a tunneling protocol developed by Cisco.

DECnet is organized in different layers and is represented within the kernel as a particular address family (see 2.1.1, “Sockets” on page 26). The layers are broadly characterized as follows:

- The lowest level layer is the kernel device driver, corresponding to the Data Link layer of the OSI model.
- The DECnet routing layer sits above the device driver. It performs the same function as, for example, the IP and ARP protocols, and corresponds to the OSI network layer.
- Above the routing layer, we find the DECnet Network Service Protocol (NSP) layer that is very similar to TCP and corresponds to the OSI transport layer.
- The BSD socket layer interface lies between the NSP layer and applications. Together, the BSD socket layer and associated system libraries realize the DECnet session control layer.

As we have already mentioned, DECnet routing is implemented in the netfilter infrastructure (see 2.2.1, “Netfilter framework” on page 30).

The front end of DECnet is a set of utilities that use the kernel DECnet address family protocols to transfer files, connect to a remote machine, and so on. These utilities include:

- `dncopy`, which copies files between Linux and VMS systems
- `dntype`, which displays the contents of files on VMS systems
- `dndir`, which displays the contents of VMS directories
- `dndel`, which deletes VMS files
- `dntask`, which executes a remote command procedure
- `dnprint`, which prints a VMS file
- `dnsubmit`, which submits a batch job to a VMS queue
- `fal`, which serves Linux files to VMS clients
- `dnetd`, the DECnet super-server (daemon)

Most of these applications use the Data Access Protocol (DAP), which is similar to ftp from the TCP/IP address family. For more information about these utilities, see the SourceForge Web site, found at:

http://linux-decnet.sourceforge.net/fileutils.html

2.3.3 ARCNET (Attached Resource Computer network)

The Linux 2.4 kernel also adds support for ARCNET network interface cards. ARCNET, developed by Datapoint Corporation in 1977, is a hardware and data link layer specification (from the OSI network model) with some transport layer function (including flow control and hardware packet acknowledgment), for a local area network that is geared towards embedded and real-time applications.

ARCNET is embedded in many companies' products, a list of which can be found in the ARCNET Trade Association (ATA) Web site at:

http://www.arcnet.com/

Major features of ARCNET include deterministic performance (allowing you to calculate worst case messaging times), noise immunity, low-cost hardware design, and the ability to mix various types of transmission media (twisted-pair wire, coaxial cable, and fiber optic cable) on the same network.

ARCNET uses a token-passing access method, similar to that used by Token ring, in a star, tree or bus network topology. Different transfer speeds are supported, ranging from 19 Kbps to 10 Mbps, with 2.5 Mbps being the standard speed. A new specification, called ARCNET Plus, supports data rates of 20 Mbps.

In the ARCNET token-passing architecture, a node must possess the token in order to transmit data. The token is passed from one computer to the next, on a numerical order basis that is independent of a computer's position in the network. The participant that owns the token can transmit one data packet of variable length (1 to 508 bytes). After transmitting the packet, the token has to be passed to the next node, even if the transmission fails.

ARCNET guarantees that a packet will arrive at its destination or, if this is not possible, that the failure will be reported, making the basic network more reliable than Ethernet.

2.3.4 ATM support

Another networking enhancement of the Linux 2.4 kernel is the addition of Asynchronous Transfer Mode (ATM) support. This support has been developed by the Institute for Computer Communications and Applications (ICA) at the École Polytechnique Fédérale de Lausanne.

ATM is a network technology for high speed integrated service networks. It is widely used for high speed backbone connections and offers Quality of Service (QoS) guarantees, which facilitate new classes of applications such as multimedia. It is the network technology chosen for Broadband Integrated Services Digital Network (B-ISDN).

ATM is a very complex technology, perhaps the most complex ever developed by the networking industry, and it is not possible to summarize its architecture here. We provide a list of Linux 2.4 supported protocols and drivers, directing the reader to more detailed reviews of the subject.

Some general documents that provide general ATM information can be found in:

- *An Overview of ATM Technology*, G. Kessler, January 1995

  http://www.vtac.org/Tech/TDatm_overview.htm

- *ATM Internetworking*, A. Alles, Cisco Systems Inc. white paper, May 1995

  http://www.cisco.com/warp/public/614/12.html

- *ATM on Linux*, from the ICA

  http://lrcwww.epfl.ch/linux-atm/

Using information from ICA's Web site, we can summarize the current status of the ATM implementation for Linux systems:

- Device drivers
  - ATM over TCP pseudo device for experiments without real ATM hardware
  - Efficient Networks ENI155p-MF and ENI155p-U5 155 Mbps ATM adapter
  - SMC ATM Power155 9741D/F and 9746D/F 155 Mbps ATM adapter
  - TI TNETA1570-based 155 Mbps ATM adapter by TU Chemnitz
  - ZeitNet ZN1221 and ZN1225 155 Mbps ATM adapter
  - IDT NICStAR 77901/77903 155 and 25 Mbps ATM adapter (77201/77211 SAR)
  - FORE ForeRunnerLE (25 and 155 Mbps; uses the IDT driver)
  - Madge (Collage 25 and 155 Client/Server)
  - All Interphase PCI Chip ATM NICs (x575, x525, and x531)
  - FORE PCA-200E
- Drivers currently under development
  - IBM TurboWays 25
  - FORE Systems ForeRunnerHE (155 and 622 Mbps)
- Supported connection types
  - Constant bit rate (CBR) permanent connections
  - Best-effort (UBR) permanent connections
  - SVCs with UNI 3.0/UNI 3.1/UNI 4.0 unicast signaling
  - Q.2963.1 peak cell rate re-negotiation
- Protocols
  - "Raw" unreliable ATM transport without AAL ("AAL0")
  - "Raw" unreliable ATM transport over AAL5
  - IP over ATM (NULL and SNAP encapsulation)
  - ATMARP (RFC1577; client and server) for PVCs and SVCs
  - Arequipa (Application REQuested IP over ATM; currently for 2.0 kernels only)
  - LAN Emulation (client and server) for SVCs
  - LANE Version 2 (client) Multi-Protocol Over ATM (MPOA, client)
  - ATM Name Service (ANS, client and server)

3

Linux 2.4 kernel hardware support and performance

This chapter provides an overview of the new Linux resource management subsystem and device file system (devfs), and includes a summary of the new hardware supported by the Linux 2.4 kernel.

We also briefly summarize the performance benefits of the Linux 2.4 kernel, with emphasis on an enterprise perspective. We do not provide benchmark test results, nor do we discuss real-life case studies. Instead, we examine performance in general terms, providing pointers to existing information on the subject.

3.1 Hardware support

Great effort has been put into the reorganization and optimization of the Linux kernel. Many improvements are related to hardware support and management. These enhancements include the new devfs device management system, a major update of the SCSI subsystem, improved enterprise support (thanks to the extended capacity of the system), and new hardware support.

IBM xSeries servers are well positioned to capitalize on these improvements, offering a range of systems from entry-level to enterprise-class. The high-end systems offer support for gigabytes of memory, large RAID configurations of SCSI and Fibre Channel disks, and options for high-speed networking.

3.1.1 Linux 2.4 devfs

We have seen that, in Linux, all hardware devices look like files and, just like files, can be opened, closed, read and written to using file manipulation system calls (see 1.1.3, “Block, character and network devices” on page 6). Each device in the system is represented as a special file in the /dev directory.

In addition, each device is allocated major and minor numbers. These numbers uniquely identify a particular device. Modern Linux distributions have very large /dev directories with over 1200 entries, because all conceivable devices must have an entry in order to be available if present. These are usually created by a program called MAKEDEV.

This brings up a number of points:

1. To maintain uniqueness of device numbers, a central coordinating authority is required.
2. There are two distinct databases of minor and major numbers: one inside the kernel code, the second in the /dev directory.
3. The size of the /dev directory can grow to huge proportions, increasing the time needed to access devices and the amount of disk space needed, which could be a issue for embedded systems.
4. Major and minor numbers are 8-bit each, limiting some types of drivers. For example, a maximum of 16 SCSI disks are supported, each with 15 partitions.
5. There are only cumbersome ways to operate on a read-only root file system, such as a CD-ROM, a ROM file system, or a shared NFS root, because it is not possible to change tty ownership and protections without write access.
6. It is not easy to use non-UNIX file systems (such as NTFS) as the root file system, because they do not directly support features such as character and block special files or symbolic links.

Some of these problems can be solved using specific user space applications, or by implementing complex scripts. Devfs tries to give a coherent and complete answer to these problems and others, and to provide additional benefits.

Devfs offers a dynamic management system for device drivers on Linux. It organizes all devices into subtrees of the /dev directory, using meaningful names, such as /dev/scsi/host0/bus0/target0/lun0/part6 for a specific partition on a SCSI disk connected to adapter 0, on channel 0, with ID 0, and logical unit number 0. These devices appear in devfs automatically, building up a map of device drivers installed in the system.

Note: Devfs is selected by the kernel build option CONFIG_DEVFS_FS. Whether or not the file system is mounted at boot time is determined by the kernel build option CONFIG_DEVFS_MOUNT. The default can be overridden by using the appropriate boot time option:

```
devfs=mount
devfs=nomount
```

Devfs works in kernel space and interacts with a device management daemon, called devfsd, in user space. The daemon allows you to modify device driver configuration concerning ownership and protection, to dynamically add devices from the devfs lists at boot time, to support absolute device drivers, and provides compatibility by maintaining support for the old style device names, using information in /etc/devfsd.conf.

The devfs approach resolves the issues we discussed earlier, among others, in the following way:

1. It reduces potential conflicts between device drivers by allowing you to design device drivers that do not allocate major and minor numbers directly. The driver can ask devfs to allocate available numbers and to link them to the driver name. Developers can decide on the name of their device driver autonomously (preferably related to the driver functionality), reducing possible conflicts with other drivers.
2. There is no more need for duplicated information about devices, because all information is managed dynamically by devfs.
3. Devfs creates its entries dynamically, in fact it creates and organizes a tree structure without allocating disk space. Thus it reduces both the time needed to access devices and disk space usage.
4. Devfs can allocate a 32-bit device identifier to a device (like having a 32-bit minor number), without causing compatibility problems with existing system libraries, since the number is internal to the kernel.
5. Devfs does not need to allocate real inodes on the root file system, hence it offers a clean way to build read-only root file systems.
6. For the same reason, devfs offers a means for using non-UNIX file systems as root.

Devfs brings other benefits:

- It makes locating particular devices on the system much simpler. The old system requires speculative device scanning, which is inefficient.
- It allows you to implement intelligent device access protection and permission management, because a script may be invoked each time a particular device is opened or closed.
- It facilitates automatic mounting of a file system when new media is inserted into a drive.

3.1.2 Devfs summary

In spite of the enhancements we have discussed, and the many other benefits it offers, devfs is a controversial addition to the Linux 2.4 kernel. It has been described as kernel bloatware, that is, a superfluous piece of software in the kernel. In the devfs FAQ, the author, Richard Gooch, addresses this. The FAQ and other useful and more detailed documentation about devfs are available from:

(http://www.atnf.csiro.au/~rgooch/linux/docs/devfs.html

3.1.3 Hardware support: IBM xSeries servers

IBM is active in the Linux community, and versions of Linux are available that run across the entire range of IBM @server systems. The general Web site for Linux at IBM is:

http://www.ibm.com/linux

In particular, the Intel CPU-based xSeries servers offer a reliable foundation for Linux-based computing, making available the performance, reliability and manageability required for core business applications. IBM is working closely with four leading Linux distributors (Caldera International, Red Hat Inc., SuSE AG, and TurboLinux), offering tested and validated configurations for the full line of xSeries servers.

xSeries systems can be configured with large amounts of memory, RAID disk subsystems, and high-speed networking, and are available in up to 8-way SMP configurations. All these hardware resources must be supported by the operating system in order to be effective. Information about running Linux on xSeries servers is available from:

http://www.pc.ibm.com/ww/eserver/xseries/linux/

Linux capacity has been greatly increased in several areas:

- 4 GB of RAM is supported on Intel hardware.
- 16 Ethernet cards can be installed in one system.
- 10 IDE controllers can be installed in one system with enhanced UDMA mode support.
- The 2 GB file size restriction has been removed and 2 TB file systems are supported (see 1.3.2, “Scalability and performance” on page 13).
- Improved storage reliability and manageability are now available with journaled file systems (see 1.3, “Journaled file systems” on page 11) and LVM tools (see 1.2, “Logical Volume Manager” on page 8).
- Enhanced symmetrical multiprocessing (SMP) support is available for up to 32 processors on 32-bit architectures and 64 processors on 64-bit architectures.
- Support for multiple I/O Advanced Programmable Interrupt Controller (I/O-APICs) is available.
- There is added support for Intel IA-64 (Itanium/Merced) processors.
- More than 16 SCSI disks are supported (see “Linux 2.4 devfs” on page 44) and a rewritten SCSI subsystem is offered. For more information, see “*The Linux 2.4 SCSI subsystem HOWTO*” at

  http://gear.torque.net/scsi/SCSI-2.4-HOWTO/)
- An updated Adaptec AIC7XXX SCSI driver is offered.

3.1.4 General hardware support

The Linux 2.4 kernel extends and enhances support for different CPUs and system buses, multimedia devices, communication ports, and video adapters. In addition, new types of network cards are now supported, as we have seen in 2.3, “Miscellaneous” on page 38.

CPU and system bus support

New CPU architectures are supported, including:

- Intel IA-64 (Itanium/Merced) processors
- IBM S/390
- Hitachi SuperH

Memory Type Range Registers (MTRRs) are supported for AMD K7 and Cyrix processors, offering improved performance for some high-bandwidth devices.

Support for bus architectures is extended with major improvements in ISA bus support and the introduction of I_2O (Intelligent Input/Output) support. ISA plug and play devices can be used during the Linux boot process, and are supported directly by the kernel instead of requiring userspace utilities. I_2O tries to realize OS independent drivers, and is a superset of PCI bus.

Support for established and some relatively new technologies has been added to the kernel, including:

- PCMCIA
- USB (Universal Serial Bus)
- IEEE 1394 (FireWire)

Ports, multimedia and video cards

The multimedia support introduced with the new kernel is essentially targeted at desktop applications, but we mention it for completeness. Peripherals discussed in this section are all classified as character devices (see 1.1.3, “Block, character and network devices” on page 6). Some are not particularly relevant for server applications, but help to promote Linux in the marketplace.

External port support has been improved in several ways:

- There is a new generic parallel port driver. This driver can be used to perform plug and play polling of system ports.
- DMA transfer mode is now supported for parallel ports.
- It is now possible to redirect console messages to the printer for a paper log of system operations; this is also useful for debugging.
- Linux 2.2 Infrared port support is now standard in the 2.4 kernel.

Also included are driver updates for sound cards and video4linux devices, and improvements for older display adapters and a number of new ones. Direct kernel support is focused on standard VGA cards, whether extended card support is provided by the XFree project.

The most interesting new feature regarding video cards is the addiction of the Direct Rendering Manager (DRM) to the Linux kernel. This solution offers increased security and stability to the system when performing intense graphic work, by regulating applications’ access to the graphic hardware.

3.2 Performance

For an operating system to be suitable for enterprise-class application, one of the most important factors is a commitment to scalability, performance and capacity. As we have seen in this paper, the overall kernel structure has been revised and enhanced with these goals in mind.

As discussed in 3.1.3, “Hardware support: IBM xSeries servers” on page 46, the Linux operating system now has improved capacity in many areas, from memory usage to data storage. Scalability has also been addressed in several ways, such as support for new scalable file systems, enhanced SMP support (including a rewritten network layer that can exploit SMP), and support for high-speed networking technologies.

One major part of the kernel rewrite has focused on the scheduling process and the system threading capacity. This important optimization increases the ability of the new Linux kernel to take full advantage of high-end system platforms such as xSeries servers.

The scheduler is a process that arbitrates access to the current CPU for multiple processes. It is implemented in the kernel and handles different types of processes. These can be broadly classified as:

- The Idle thread, which is always executed when the CPU is otherwise idle.
- Kernel threads, which are created in kernel mode. These can access the kernel address space and usually have no user address space. They cannot be preempted (interrupted) by the scheduler process.
- User tasks, which work in user mode and are subject to preemption by the multitasking scheduler.

Linux 2.4 can now handle many more simultaneous processes or threads because of its improved scalability on multiprocessor systems; it also provides a configurable process limit. The threading model used in Linux 2.4 has been changed to a scalable (or soft) version. Previously, the thread limit was 1024, which led to poor performance with large numbers of users. The limit is now set at run time, so the only real limit is the amount of memory available.

4

Compiling the Linux 2.4 kernel

Many features we introduced earlier in this paper may require the user to compile the Linux kernel. Features that are included in a basic installation are dependent upon which Linux distribution you are using. You will need to determine whether or not the features you require are available by default and, if not, configure and compile the kernel to include them.

The examples given in this chapter are based on Red Hat Linux 7.1, but the principles are broadly transferable to other distributions. One of the reasons for selecting Red Hat is that SGI has released an interesting bootable CD-ROM image, which works in conjunction with the Red Hat installation disks and allows you to install XFS as the root file system. It also provides you with LVM, ReiserFS, devfs, and other features, all ready to use.

The standard Red Hat installer does not include LVM, devfs and XFS in the kernel image, so, unless you use the SGI disk, the user has to compile the kernel to apply the appropriate patches. We therefore provide some guidance for kernel configuration and installation of user space applications.

Detailed, distribution-independent information about upgrading a Linux 2.2 kernel machine to version 2.4 can be found at the following URL:

http://cyberbuzz.gatech.edu/kaboom/linux/Changes-2.4/changes24.html

4.1 Hardware

We used a variety of IBM @server xSeries and Netfinity servers to test the 2.4 kernel installation and configuration during the writing of this paper. In particular, we used xSeries 330/Netfinity 4000R class servers, which make excellent high-density systems for Linux applications, as they are only 1U in height and many of them can be configured in a single rack, with external storage enclosures if required.

Business-critical applications are likely to need large capacity RAID storage subsystems and IBM xSeries servers support a number of such options, including Fibre Channel and SCSI controllers. We used a Netfinity 5600 server equipped with a ServeRAID SCSI disk controller. For more information about RAID configuration and utilization under Linux, we recommend the following Redbooks:

- *Red Hat Linux Integration Guide for* IBM @server *xSeries and Netfinity*, SG24-5853
- *TurboLinux Integration Guide for* IBM @server *xSeries and Netfinity*, SG24-5862
- *Caldera OpenLinux Integration Guide for* @server *xSeries and Netfinity*, IBM SG24-5861
- *SuSE Linux Integration Guide for* @server xSeries *and Netfinity*, SG24-5863

4.2

As we mentioned in the introduction to this chapter, SGI has released a CD-ROM image, which replaces the Red Hat installer and allows you to install XFS as the root file system, among other features. You can download the SGI Installer from their Web site at:

http://linux-xfs.sgi.com/projects/xfs/1.0_release.html

as an ISO image. After downloading the file and creating the CD-ROM, you are ready to perform a new installation or a system upgrade.

4.2.1 Installing a new system

To
support booting from the CD-ROM, you need to create a boot floppy disk and perform a CD-ROM or network installation. Information about creating a boot floppy disk can be found in the Red Hat documentation, available at:

http://www.redhat.com/apps/support/documentation.html

After booting, the SGI Installation utility goes through different configuration screens that are essentially the same as those of the original Red Hat installation utility. The only real difference is a new configuration screen which asks the user to choose which partitions should use the XFS file system.

A
7.1 disks as each one is required, and installation proceeds normally.

4.2.2 Upgrading an existing installation

Before upgrading a previously installed Red Hat system, we recommend that you back up all of your existing data, should problems arise during the process.

There is another step you should take before performing the upgrade, which is to install the devfs utilities on the existing system (3.1.1, “Linux 2.4 devfs” on page 44). This is necessary, since the SGI installer enables devfs on the updated system. During the Red Hat update process, only currently installed packages are affected and no new package installation is performed. Unless you install the devfs utilities, the updated system will fail to load the devfsd daemon at the next reboot, leaving the system in an unusable state.

As we explained in 3.1.1, “Linux 2.4 devfs” on page 44, devfs changes the way devices are mapped and reorganizes the /dev directory. The devfsd daemon preserves compatibility with your existing applications by creating the appropriate links in the /dev directory, so as to allow references to old device names to be maintained.

The problem is overcome by installing the devfsd-1.3.11-sgi.i386.rpm package prior to the upgrade. This package can be found on the SGI CD-ROM in the RedHat/Rpms/ directory.

4.3 Compiling and patching

For those who wish to tailor the kernel to their own needs, either to remove unwanted features or to add some that are not included as standard, we provide a brief outline of the process. This is not intended as a complete guide to compiling the kernel; you are highly encouraged to read the kernel documentation (found in the Documentation directory under kernel source directory) and the kernel Howto, which can be found at:

http://www.linuxdoc.org/HOWTO/Kernel-HOWTO.html

We assume that you have Red Hat 7.1 Linux installed on your system, and you that you have verified that the appropriate software listed at:

http://cyberbuzz.gatech.edu/kaboom/linux/Changes-2.4/changes24.html

is installed.

Tip: If you used SGI's installation disk, you may find that your swap partition fails at boot time. This is due to the new names scheme of devfs. To overcome the problem, edit your /etc/fstab file and change the swap line, substituting /dev/sda6 (if sda6 is your swap partition) with the devfs equivalent (/dev/disks/disc0/part6 for example; see 3.1.1, “Linux 2.4 devfs” on page 44). After this modification, execute:

```
[root@localhost /root]#  swapon -a
```

to enable the swap space.

4.3.1 Preparing to compile the kernel

Log in as root to perform the following steps:

1. Mount the first Red Hat CD-ROM.
2. Install the Linux kernel header rpms:

```
[root@localhost /root]# cd /mnt/cdrom/RedHat/RPMS
[root@localhost /root]# rpm -i kernel-headers*.rpm
[root@localhost /root]# rpm -i kcc*.rpm
```

   Note that kcc compiler is necessary to correctly compile the kernel under Red Hat systems.

3. Unmount the first CD-ROM and mount the second CD-ROM to install the Linux kernel source and documents:

```
[root@localhost /root]# cd /mnt/cdrom/RedHat/RPMS
[root@localhost /root]# rpm -i kernel-source*.rpm
[root@localhost /root]# rpm -i kernel-doc*.rpm
```

Note: The following items should be considered:

1. If you are using the SGI kernel, perform steps 1 through 3 using the SGI CD-ROM.
2. Red Hat 7.1 uses kernel version 2.4.2. If you want to compile a more recent version, you should download the kernel source from http://www.kernel.org
3. If you choose to use the standard vanilla kernel source, remember to install the appropriate modutils utilities that correspond to your kernel version. These utilities are available from: http://www.kernel.org/pub/linux/utils/kernel/modutils/v2.4

   You should choose the highest patch level that is equal to or below that of your current kernel.

4. If you want to patch the kernel, you should do it at this point. First of all, you may want to make a backup of your source directory tree by using these commands:

```
[root@localhost /root]# cd /usr/src/linux-2.4
[root@localhost /root]# make clean
[root@localhost /root]# cd /usr/src
[root@localhost /root]# tar zcvf old-tree-source.tar.gz linux-2.4
```

5. Download the appropriate patch from the Internet. Patch files are usually in tar.gz format, so you need to copy the archive to /usr/src/linux-2.4 and untar it with a command such as:

```
tar -xvzf patch_name.tar.gz
```

   This command creates a new file, named patch_name.

6. We now need to patch the kernel by issuing these commands:

```
[root@localhost /root]# cd /usr/src/linux-2.4
[root@localhost /root]# patch -p1 < /usr/src/linux-2.4/patch_name
```

7. After the patches are applied, you can configure the kernel. This can be done with three distinct utilities:

   – make config (textual interface)
   – make menuconfig (semi-graphic interface)
   – make xconfig (X Windows-based interface)

   The first option requires more experience and a good understanding of all configuration options. The other two choices are almost identical and equally simple to use. We used make xconfig throughout this chapter to simplify the process.

4.3.2 Kernel configuration

We now take you through the steps of configuring the kernel using xconfig:

1. Start Xserver and the window manager:

   ```
   [root@localhost /root]# startx &
   ```

2. Open a terminal window and execute:

   ```
   [root@localhost /root]# cd /usr/src/linux-2.4
   [root@localhost /root]# make xconfig
   ```

   The xconfig window is displayed (see Figure 4-1):

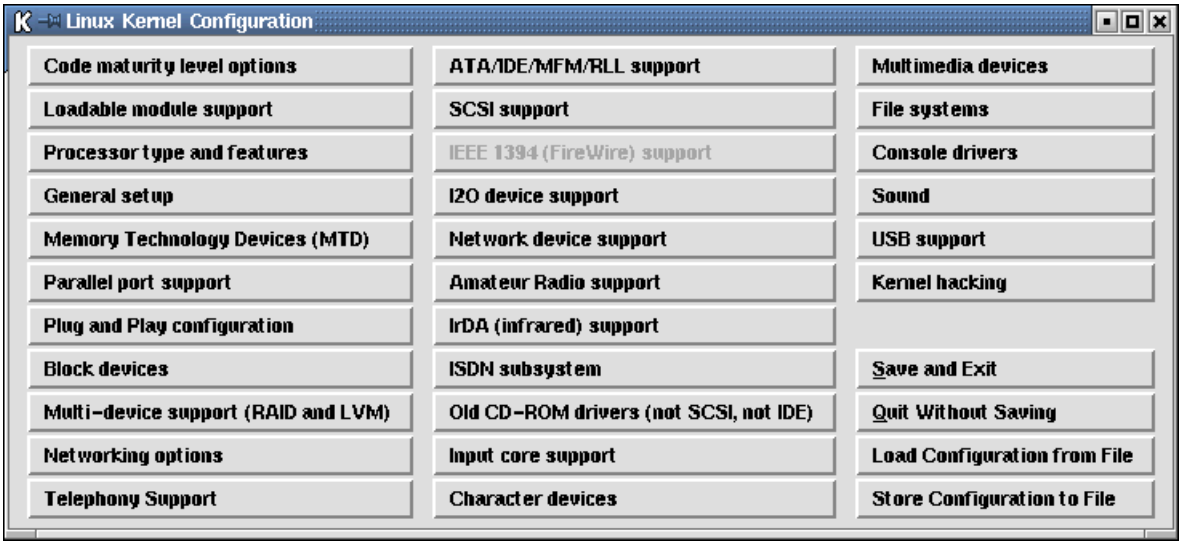


Figure 4-1 Linux kernel configuration screen

3. To configure a specific item, click its button. We suggest you start from **`Code maturity level options`** and work through all configuration steps by clicking **Next** on each screen. This method ensures that you configure all kernel features without forgetting any. Some options are unavailable if previous options have not been set.

   You can see an explanation for each option by selecting **Help** for the chosen option.

4. Most new features require that the CONFIG_EXPERIMENTAL option be set. This option is set in the GUI under **`Code maturity level options`**, as you can see in Figure 4-2:

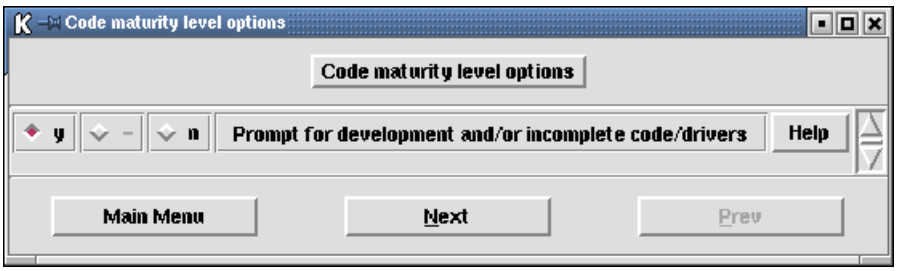


Figure 4-2 Setting the code maturity options

5. The second option you should set is CONFIG_MODULES. This is set in **`Loadable module support`** by selecting **Enable loadable module support**. It is also a good idea to disable CONFIG_MODVERSIONS by deselecting **Set version information on all module symbols**.

6. Reboot the machine and select the appropriate entry in the boot loader screen.
7. If the system boots properly, you now compile any modules you need, using the following commands:

```
[root@localhost /root]# cd /usr/src/linux-2.4.2
[root@localhost /root]# make modules
[root@localhost /root]# make modules_install
```

4.4 Logical Volume Manager

As we saw in 1.2, “Logical Volume Manager” on page 8, the LVM feature is already present in the Linux 2.4 kernel. We recommend, however, that you check the Sistina Web site:

http://www.sistina.com

for updates to patch your kernel. As an example, if you are using kernel 4.3 or earlier, you must apply the latest LVM patch. If you are using the SGI kernel, the LVM module is ready to use.

To include LVM support when recompiling the kernel, you should download any necessary patches and apply them to the source code. Complete details are available at:

http://www.sistina.com/lvm/doc/lvm_howto/

During kernel configuration, you should set the CONFIG_BLK_DEV_LVM option. This is done with xconfigin the Multi-device support (RAID and LVM) window by selecting **Logical volume manager (LVM) support**, as shown in Figure 4-4:

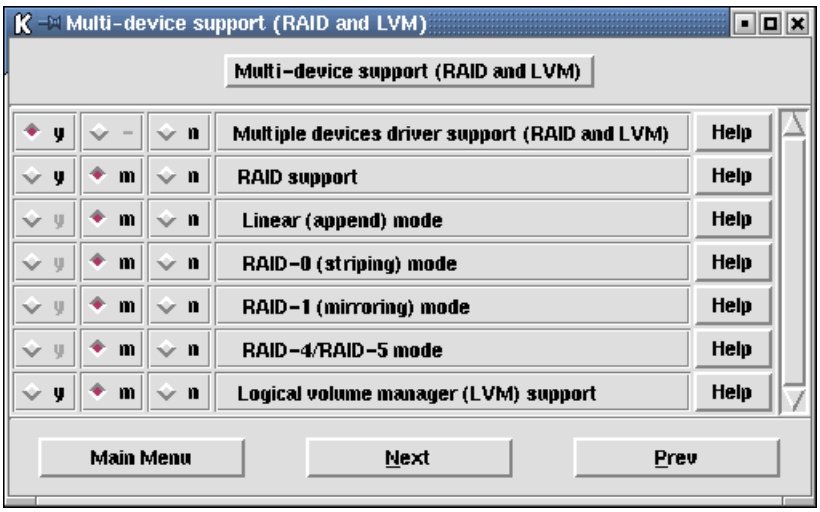


Figure 4-4 Selecting LVM

LVM can be compiled into the kernel or, as we did, as a module.

If you compile LVM as a module, you should be able to check if everything went well using modprobe:

```
[root@localhost /root]# modprobe lvm-mod
```

4.4.1 LVM utilities

Kernel configuration alone is not enough to actually use LVM. You also need the LVM utility package so that you can enable and configure LVM functions. This is true even if you are using the SGI installation. The latest version tarballs can be found at:

ftp://linux.msede.com/lvm/current

and rpm binary packages can be found here:

ftp://linux.msede.com/lvm/current/binaries/redhat

Optionally, the Mandrake rpm package can be installed from:

ftp://speakeasy.rpmfind.net/linux/Mandrake-devel/cooker/i586/Mandrake/RPMS/lvm-0.9-1mdk.i586.rpm

Suppose you have downloaded the Mandrake rpm package into the /root directory; to install and verify LVM, you should go through the following steps:

1. First of all, use the rpm utility to install the package:

```
[root@localhost /root]# ls
lvm-0.9-1mdk.i586.rpm
[root@localhost /root]# rpm -Uvh lvm-0.9-1mdk.i586.rpm
Preparing...                ########################################### [100%]
   1:lvm                    ########################################### [100%]
```

2. Use modprobe to load the LVM kernel module into memory

```
[root@localhost /root]# modprobe lvm-mod
```

3. Finally, use vgscan to check which volume groups are installed at the moment.

```
[root@localhost /root]# vgscan
vgscan -- reading all physical volumes (this may take a while...)
vgscan -- no volume groups found
```

To enable LVM at boot time, you should modify /etc/rc.d/rc.sysinit on Red Hat system (for other distributions, check the LVM-HOWTO). Look for the line `Mount all other filesystems` and insert these two commands

```
vgscan
vgchange -ay
```

This is illustrated in Figure 4-5:

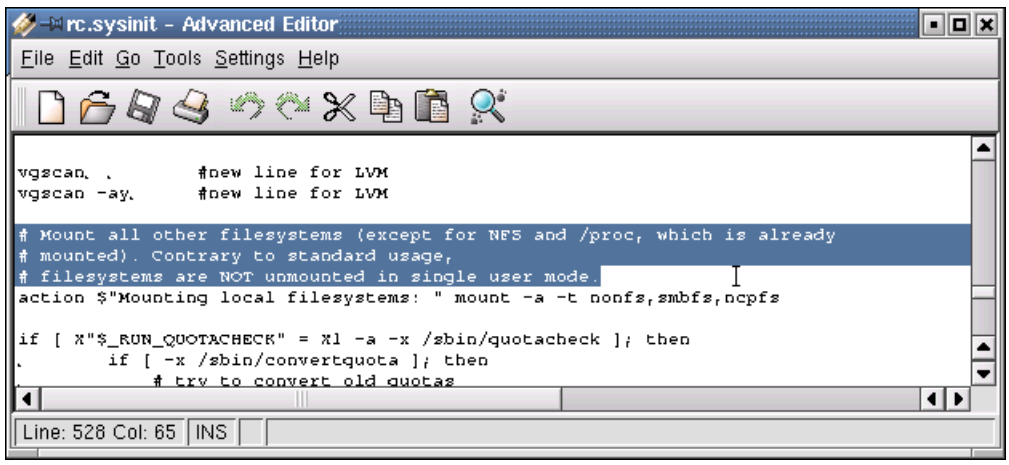


Figure 4-5 Updating rc.sysinit

In the same way, you should modify `/etc/rc.d/init.d/halt` to deactivate volume groups at start time by inserting

```
vgchange -an
```

near the end of the file, after the line `# Remount read only anything that's left mounted`, and before `# Now halt or reboot`.

4.4.2 Configuring LVM

Now that the LVM utilities are installed and the system boot and halt sequences are ready, we can configure our volume groups and logical volumes. The following is an example based on our configuration; be sure to determine which partitions you intend to configure as LVM on your system, and modify your own input accordingly.

1. First, we need to set a special flag for each partition we want to use as a physical volume. More precisely, we have to set the partition type to 8e, signifying that it is the Linux LVM partition type. To do this, we use the latest version of the fdisk utility:

```
[root@localhost /root]# fdisk /dev/discs/disc0/disc
The number of cylinders for this disk is set to 1282.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): p
```

Note: We used the new devfs name to specify the appropriate disk.

2. The **p** command prints out the partition table for the disk we are examining. The output of the command should look something like this:

```
Disk /dev/discs/disc0/disc: 254 heads, 63 sectors, 1282 cylinders
Units = cylinders of 16002 * 512 bytes

                Device Boot    Start       End    Blocks   Id  System
/dev/discs/disc0/part1   *         1         3     23971+  83  Linux
/dev/discs/disc0/part2             4      1282  10233279    5  Extended
/dev/discs/disc0/part5             4       643   5120608+  83  Linux
/dev/discs/disc0/part6           644       797   1232122+  82  Linux swap
/dev/discs/disc0/part7           798      1282   3880453+  83  Linux
```

3. We then use the **t** command to change the type of, in our case, partition 7:

```
Command (m for help): t
Partition number (1-7): 7
Hex code (type L to list codes): 8e
Changed system type of partition 7 to 8e (Linux LVM)
```

4. Using the **p** command once again, we can view the results:

```
Command (m for help): p

Disk /dev/discs/disc0/disc: 254 heads, 63 sectors, 1282 cylinders
Units = cylinders of 16002 * 512 bytes

                Device Boot    Start       End    Blocks   Id  System
/dev/discs/disc0/part1   *         1         3     23971+  83  Linux
/dev/discs/disc0/part2             4      1282  10233279    5  Extended
```

```
/dev/discs/disc0/part5              4       643   5120608+  83  Linux
/dev/discs/disc0/part6            644       797   1232122+  82  Linux swap
/dev/discs/disc0/part7            798      1282   3880453+  8e  Linux LVM
```

5. We write the change to the partition table using the **w** command:

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Re-read table failed with error 16: Device or resource busy.
Reboot your system to ensure the partition table is updated.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
```

We also changed the type of the first partition of the second SCSI disk to Linux LVM, so that we have two partitions to allocate to the new volume group we are about to create. The two LVM partitions are:

```
/dev/scsi/host2/bus0/target0/lun0/part7
/dev/scsi/host2/bus0/target1/lun0/part1
```

These names specify all of the SCSI parameters for the selected partitions and are synonyms for /dev/discs/disc0/part7 and /dev/discs/disc1/part7, respectively (see 3.1.1, “Linux 2.4 devfs” on page 44).

6. The next step is to create the physical volumes on the partitions we have selected:

```
[root@localhost /root]# pvcreate /dev/scsi/host2/bus0/target1/lun0/part1
pvcreate -- physical volume "/dev/scsi/host2/bus0/target1/lun0/part1" successfully
created

[root@localhost /root]# pvcreate /dev/scsi/host2/bus0/target0/lun0/part7
pvcreate -- physical volume "/dev/scsi/host2/bus0/target0/lun0/part7" successfully
created
```

7. We can examine a physical volume using the pvdisplay utility:

```
[root@localhost /root]# pvdisplay /dev/scsi/host2/bus0/target0/lun0/part7
--- Physical volume ---
PV Name               /dev/scsi/host2/bus0/target0/lun0/part7
VG Name               newvg
PV Size               3.7 GB / NOT usable 1.5 MB [LVM: 124 KB]
PV#                   1
PV Status             available
Allocatable           yes
Cur LV                0
PE Size (KByte)       4096
Total PE              947
Free PE               947
Allocated PE          0
PV UUID               93cATO-bUIb-SD8W-6bLS-lmMz-Ednr-SFQTYD
```

8. Now we allocate the physical volumes we created to a new volume group:

```
[root@localhost /root]# vgcreate newvg /dev/scsi/host2/bus0/target0/lun0/part7
/dev/scsi/host2/bus0/target1/lun0/part1
vgcreate -- INFO: using default physical extent size 4 MB
vgcreate -- INFO: maximum logical volume size is 255.99 Gigabyte
vgcreate -- doing automatic backup of volume group "newvg"
vgcreate -- volume group "newvg" successfully created and activated
```

9. Logical volumes can now be created inside the volume group. we create two such logical volumes:

```
[root@localhost /root]# lvcreate -L 100M -n logical1 newvg
lvcreate -- doing automatic backup of "newvg"
lvcreate -- logical volume "/dev/newvg/logical1" successfully created

[root@localhost /root]# lvcreate -L 100M -n logical2 newvg
lvcreate -- doing automatic backup of "newvg"
lvcreate -- logical volume "/dev/newvg/logical2" successfully created
```

10. We can examine the volume group, newvg, using the vgdisplay utility:

```
[root@localhost linux]# vgdisplay -v newvg
--- Volume group ---
VG Name               newvg
VG Access             read/write
VG Status             available/resizable
VG #                  0
MAX LV                256
Cur LV                2
Open LV               2
MAX LV Size           255.99 GB
Max PV                256
Cur PV                2
Act PV                2
VG Size               7.59 GB
PE Size               4 MB
Total PE              1943
Alloc PE / Size       50 / 200 MB
Free  PE / Size       1893 / 7.39 GB
VG UUID               UuNgOP-PUla-xCTw-Sv83-mCWS-SvSF-GwTkoF

--- Logical volume ---
LV Name                /dev/newvg/logical1
VG Name                newvg
LV Write Access        read/write
LV Status              available
LV #                   1
# open                 1
LV Size                100 MB
Current LE             25
Allocated LE           25
Allocation             next free
Read ahead sectors     120
Block device           58:0

--- Logical volume ---
LV Name                /dev/newvg/logical2
VG Name                newvg
LV Write Access        read/write
LV Status              available
LV #                   2
# open                 1
LV Size                100 MB
Current LE             25
Allocated LE           25
Allocation             next free
Read ahead sectors     120
Block device           58:1
```

```
--- Physical volumes ---
PV Name (#)           /dev/scsi/host2/bus0/target0/lun0/part7 (1)
PV Status             available / allocatable
Total PE / Free PE    947 / 897

PV Name (#)           /dev/scsi/host2/bus0/target1/lun0/part1 (2)
PV Status             available / allocatable
Total PE / Free PE    996 / 996
```

This summary is quite detailed and offers a complete view of the volume group, logical volume, and physical volume status.

LVM utilities

A number of utilities are available to help you manage your logical volumes. To explore the usage of these utilities in depth is beyond the scope of this paper. However, to see the utilities that are available, use the rpm utility to query the package:

```
[root@localhost linux]# rpm -ql lvm | grep sbin
/sbin/e2fsadm
/sbin/lvchange
/sbin/lvcreate
/sbin/lvdisplay
/sbin/lvextend
...
```

For each application, use the man program to access a detailed description of the functions available. For example:

```
[root@localhost linux]# man lvcreate
```

We want to highlight the fact that, even though LVM is still in beta release, it has several promising strengths and advantages of which the approaching final release should take full advantage.

As a last comment, all of the installation procedures were performed on a Netfinity system using the IBM ServeRAID adapter in a perfectly transparent way.

4.5 Journaled file systems

The three file systems we are going to explore (ReiserFS, XFS and JFS) have different installation procedures.

- ReiserFS comes with the Red Hat 7.1 basic installation, although it is not possible to use it in the root partition during the standard installation process.
- XFS is available as a patch against the standard kernel, but is also present in the base system using SGI's installation disk. It is possible to use XFS as a root file system during SGI's installation procedure.
- JFS is only available as a kernel patch. This means that you have to recompile the kernel in order to use it in your system.

  It is important to note that JFS is still in beta release.

Each new journaled file system comes with a full set of utilities, which are used to create and administer the file system.

4.5.1 ReiserFS

The ReiserFS module is installed by default during the Red Hat 7.1 installation process, making it immediately usable on a new or updated system. If you are configuring the kernel to include this file system, you should set CONFIG_REISERFS_FS by selecting **Reiserfs support** in the File systems window of xconfig (see Figure 4-6).

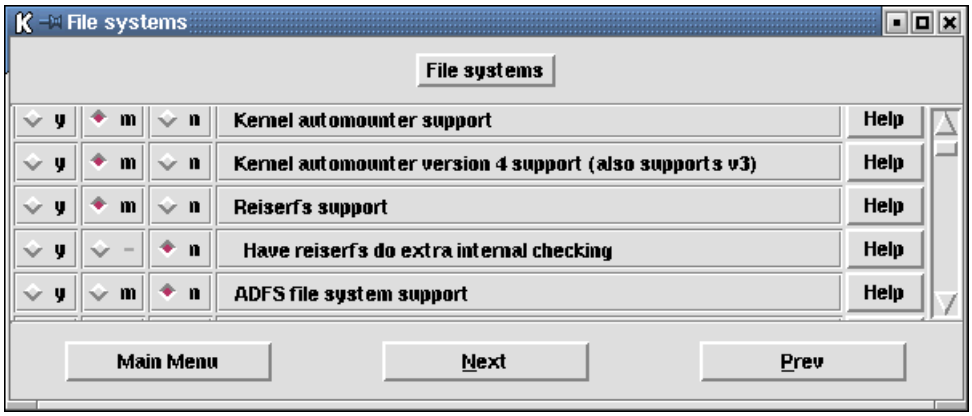


Figure 4-6 Selecting ReiserFS support

You can compile ReiserFS as a module or inside the kernel. Note that the **Have reiserfs do extra internal checking** option is for debugging purposes and should not normally be set.

Since ReiserFS is included in the main kernel, you should not have to patch against any recent Linux version, but you may need to install the ReiserFS utilities (reiserfs-utils).

To verify if your system has the appropriate utilities installed, use the rpm utility:

```
[root@localhost linux]# rpm -ql reiserfs-utils
/sbin/debugreiserfs
/sbin/fsck.reiserfs
/sbin/mkfs.reiserfs
/sbin/mkreiserfs
/sbin/reiserfsck
/sbin/resize_reiserfs
/sbin/unpack
/usr/share/doc/reiserfs-utils-3.x.0f
/usr/share/doc/reiserfs-utils-3.x.0f/README
/usr/share/man/man8/debugreiserfs.8.gz
/usr/share/man/man8/mkreiserfs.8.gz
/usr/share/man/man8/reiserfsck.8.gz
```

If the utility package is not installed, you can find it on the second Red Hat 7.1 disk.

To create a new ReiserFS file system on a logical volume, such as one of those created in 4.4.2, “Configuring LVM” on page 57, we follow these steps:

1. First, we use the **mkfs** command to create the file system:

```
[root@localhost /root]# mkfs -t reiserfs /dev/newvg/logical2

<-------------mkreiserfs, 2000------------>
reiserfsprogs 3.x.0f
Creating reiserfs of 3.6 format
Block size 4096 bytes
Block count 25600
Used blocks 8212
```

```
Free blocks count 17388
First 16 blocks skipped
Super block is in 16
Bitmap blocks (1) are :
        17
Journal size 8192 (blocks 18-8210 of file /dev/newvg/logical2)
Root block 8211
Hash function "r5"
ATTENTION: YOU SHOULD REBOOT AFTER FDISK!
            ALL DATA WILL BE LOST ON '/dev/newvg/logical2'! (y/n)y
Initializing journal - 0%...20%...40%...60%...80%...100%
left 0, 2731 /sec
Syncing..

ReiserFS core development sponsored by SuSE Labs (suse.com)

Journaling sponsored by MP3.com.

To learn about the programmers and ReiserFS, please go to
http://www.devlinux.com/namesys

Have fun.
```

2. After the file system has been created, we mount it to make it accessible:

```
[root@localhost /root]# mkdir /mnt/vol2
[root@localhost /root]# mount -t reiserfs /dev/newvg/logical2 /mnt/vol2/
```

3. Just to show the file system is now available, we create a simple text file in it:

```
[root@localhost /root]# echo "First file in ReiserFS filesystem" > /mnt/vol2/firstfile
[root@localhost /root]# ls -l /mnt/vol2/
total 4
-rw-r--r--    1 root     root           32 Jun  7 19:10 firstfile
```

ReiserFS utilities

A number of utilities are available to help you use the ReiserFS file system.

If you have problems with the file system, the debugreiserfs utility can be used to help you to resolve them. If called without arguments, it prints out information about the super block found on the device:

```
[root@localhost linux]# umount /mnt/vol2
[root@localhost linux]# /sbin/debugreiserfs /dev/newvg/logical2

<-------------debugreiserfs, 2000------------>
reiserfsprogs 3.x.0f
Super block of format 3.6 found on the 0x3 in block 16
Block count 25600
Blocksize 4096
Free blocks 17388
Busy blocks (skipped 16, bitmaps - 1, journal blocks - 8193
1 super blocks, 1 data blocks
Root block 8211
Journal block (first) 18
Journal dev 0
Journal orig size 8192
Filesystem state VALID
Tree height 2
Hash function used to sort names: "r5"
Objectid map size 2, max 972
```

Note: You have to unmount the file system to use the debugreiserfs application.

By using different flags with the **debugreiserfs** command, we can obtain more information about the file system status. For example, the -j flag prints the journal contents. Additional information be found by using the man utility.

A very important utility is reiserfsck. It *l*ooks for the reiserfs filesystem on a device, replays transactions which are to be replayed and either checks or repairs the filesystem (from man pages).

We ran the reiserfsck tool on our previously created file system, after unmounting it:

```
[root@localhost linux]# umount /dev/newvg/logical2
[root@localhost linux]# reiserfsck /dev/newvg/logical2

<-------------reiserfsck, 2000------------>
reiserfsprogs 3.x.0f
Will read-only check consistency of the partition
Will put log info to stderr
Do you want to run this program?[N/Yes] (note need to type Yes):Yes
Analyzing journal..nothing to replay (no transactions older than last flushed on
e found)
Checking S+tree..ok
Comparing bitmaps..ok
Checking Semantic tree...ok
```

Resizing the file system

The last utilities we examine are used to resize a ReiserFS file system.

In this example, we show you how to increase the space available.

1. First we enlarge the existing logical volume, using the **lvextend** command:

   ```
   [root@localhost linux]# lvextend -L+1G /dev/newvg/logical2
   lvextend -- extending logical volume "/dev/newvg/logical2" to 1.1 GB
   lvextend -- doing automatic backup of volume group "newvg"
   lvextend -- logical volume "/dev/newvg/logical2" successfully extended
   ```

2. Now we need to extend the file system in the logical volume:

   ```
   [root@localhost linux]# umount /dev/newvg/logical2
   [root@localhost linux]# resize_reiserfs /dev/newvg/logical2

   <-------------resize_reiserfs, 2000------------>
   reiserfsprogs 3.x.0f
   reiserfs_open: bitmap 3 was marked free
   reiserfs_open: bitmap 4 was marked free
   reiserfs_open: bitmap 5 was marked free
   reiserfs_open: bitmap 6 was marked free
   reiserfs_open: bitmap 7 was marked free
   reiserfs_open: bitmap 8 was marked free
   ReiserFS report:
   blocksize             4096
   block count           287744 (25600)
   free blocks           279524 (17388)
   bitmap block count    9 (1)

   Syncing..done
   ```

Note: We could have extended the file system without unmounting it. In that case, the correct instructions would have been:

```
[root@localhost linux]# resize_reiserfs -f /dev/newvg/logical2
```

3. To verify the new size, we use the **df** command:

```
[root@localhost linux]# mount -t reiserfs /dev/newvg/logical2 /mnt/vol2
[root@localhost linux]# df /dev/newvg/logical2
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/newvg/logical2    1150936     32840   1118096   3% /mnt/vol2
```

If, instead, you wish to reduce the logical volume size:

1. Issue the **resize_reiserfs** command to reduce the ReiserFS file system size:

```
[root@localhost linux]# umount /dev/newvg/logical2
[root@localhost linux]# resize_reiserfs -s-500M /dev/newvg/logical2

<-------------resize_reiserfs, 2000------------>
reiserfsprogs 3.x.0f
You are running BETA version of reiserfs shrinker.
This version is only for testing or VERY CAREFUL use.
Backup of you data is recommended.

Do you want to continue? [y/N]:y
Processing the tree: 0%
     left 0, 0 /sec

nodes processed (moved):
int        0 (0),
leaves     1 (0),
unfm       0 (0),
total      1 (0).

ReiserFS report:
blocksize             4096
block count           159744 (287744)
free blocks           151528 (279524)
bitmap block count    5 (9)

Syncing..done
```

2. Now you can reduce the LV accordingly:

```
[root@localhost linux]# lvreduce -L-500M /dev/newvg/logical2
lvreduce -- WARNING: reducing active logical volume to 624 MB
lvreduce -- THIS MAY DESTROY YOUR DATA (filesystem etc.)
lvreduce -- do you really want to reduce "/dev/newvg/logical2"? [y/n]: y
lvreduce -- doing automatic backup of volume group "newvg"
lvreduce -- logical volume "/dev/newvg/logical2" successfully reduced
```

3. Mount the LV to check the new size:

```
[root@localhost linux]# mount -t reiserfs /dev/newvg/logical2 /mnt/vol2
[root@localhost linux]# df /dev/newvg/logical2
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/newvg/logical2     638952     32840    606112   6% /mnt/vol2

[root@localhost linux]# mount -t reiserfs /dev/newvg/logical2 /mnt/vol2
[root@localhost linux]# df /dev/newvg/logical2
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/newvg/logical2     638952     32840    606112   6% /mnt/vol2
```

For more information on ReiserFS, go to:

http://www.namesys.com/

4.5.2 XFS

If you use the SGI installation disk, XFS is ready to use and an automatic procedure helps to create an XFS file system on each mount point. Alternatively, SGI XFS Linux Release 1.0 is available as patches against linux-2.4.2 and linux-2.4.3 from:

http://oss.sgi.com/projects/xfs/

If you are configuring a kernel from scratch, you have to apply patches from:

ftp://oss.sgi.com/projects/xfs/download/Release-1.0/patches

Choose the appropriate version and type, and refer to:

http://linux-xfs.sgi.com/projects/xfs/1.0_source.html

for detailed information about how to apply the patches.

After patching the kernel source (see 4.3, “Compiling and patching” on page 51), you have to set **Page Buffer support** (CONFIG_PAGE_BUF) and **SGI XFS filesystem support** (CONFIG_XFS_FS) in the File systems options window of xconfig, as shown in Figure 4-7:

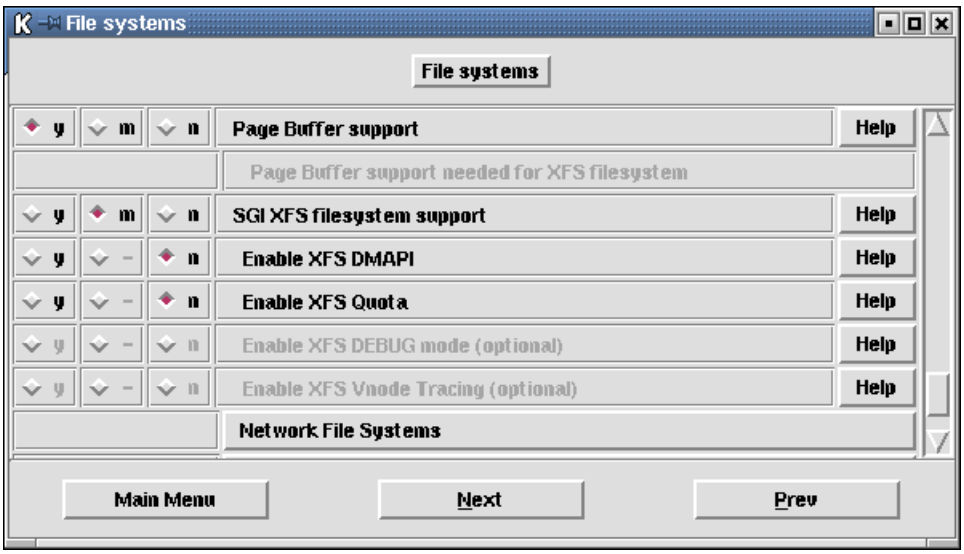


Figure 4-7 Setting kernel options for XFS

XFS support is based on an abstract buffer cache model built on top of the Linux page cache, which is why it is necessary to enable page buffer support. You can compile the necessary code either into the kernel or as a module. The other options shown in the figure are useful for debugging purposes.

If you want to use XFS as a root file system, it is simpler to have it compiled into the kernel, otherwise you need to create a initial ramdisk (initrd), as explained on the SGI Web site at:

http://linux-xfs.sgi.com/projects/xfs/1.0_rpm.html

Additionally, SGI IRIX disks can be migrated to Linux by setting two more configuration flags: **Advanced partition selection** (CONFIG_PARTITION_ADVANCED) and **SGI partition support** (CONFIG_SGI_PARTITION). These can be found in the Partition Types window within File systems, as shown in Figure 4-8:

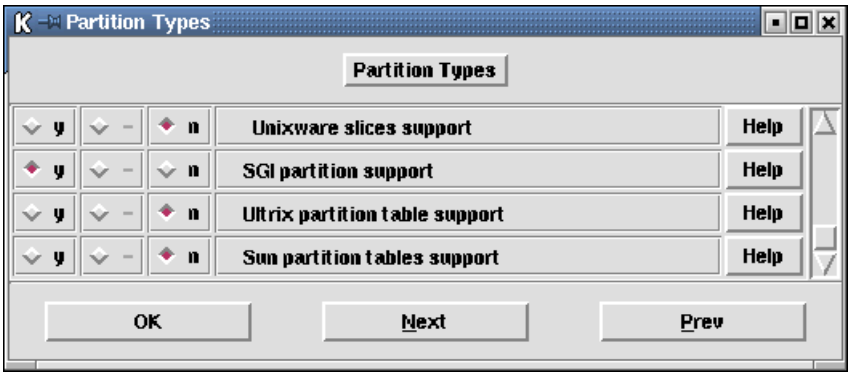


Figure 4-8 Settings to allow migration of SGI IRIX disks

XFS utilities

Before compiling the kernel, it is necessary to upgrade the following utilities to the indicated version or later:

- For modutils-2.4.0, go to:

  http://www.kernel.org/pub/linux/utils/kernel/modutils/v2.4

- For autoconf-2.13, go to:

  http://www.gnu.org/software/autoconf/autoconf.html

- For e2fsprogs-devel-1.18, go to:

  http://e2fsprogs.sourceforge.net

To verify that your system has the appropriate utilities installed, use the rpm utility:

```
[root@localhost linux-2.4]# rpm -ql xfsprogs
/sbin/fsck.xfs
sbin/mkfs.xfs
/sbin/xfs_repair
...
/usr/sbin/xfs_admin
/usr/sbin/xfs_bmap
/usr/sbin/xfs_check
/usr/sbin/xfs_db
/usr/sbin/xfs_growfs
/usr/sbin/xfs_info
/usr/sbin/xfs_logprint
/usr/sbin/xfs_mkfile
/usr/sbin/xfs_ncheck
```

Other utilities can be found as part of the xfsdump package:

```
[root@localhost linux]# rpm -ql xfsdump
…
/usr/sbin/xfs_estimate
/usr/sbin/xfs_fsr
/usr/sbin/xfsdq
/usr/sbin/xfsdump
/usr/sbin/xfsinvutil
/usr/sbin/xfsrestore
/usr/sbin/xfsrq
…
```

For detailed information on each utility's operation, view the respective man pages.

If the utilities are not present in your system, they can be found at the SGI ftp site as rpms:

ftp://oss.sgi.com/projects/xfs/download/Release-1.0/RH7.1-SGI-XFS-1.0/RedHat/RPMS/

or as tarballs:

ftp://oss.sgi.com/projects/xfs/download/Release-1.0/cmd_tars/

These URLs also host other applications to manage quotas, access control lists (ACLs), Data Management API (DMAPI), and so on.

Creating an XFS file system

To create a new XFS on a logical volume, follow these steps, using your own partition and volume information where appropriate:

1. Use the **mkfs** command to create the new file system, using the -t flag to specify XFS:

```
[root@localhost /root]# mkfs -t xfs /dev/newvg/logical1
meta-data=/dev/newvg/logical1    isize=256    agcount=6, agsize=4267 blks
data     =                       bsize=4096   blocks=25600, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=0
naming   =version 2              bsize=4096
log      =internal log           bsize=4096   blocks=1200
realtime =none                   extsz=65536  blocks=0, rtextents=0
```

2. Just as we did for the ReiserFS, we can mount the XFS file system in a directory and create a text file as a simple test that we can access it:

```
[root@localhost /root]# mkdir /mnt/vol1
[root@localhost /root]# mount -t xfs /dev/newvg/logical1 /mnt/vol1/
[root@localhost /root]# echo "First file in xfs filesystem" > /mnt/vol2/firstfile
[root@localhost linux]# ls -l /mnt/vol1
total 4
-rw-r--r--    1 root     root           29 Jun  7 18:50 firstfile
```

Using the utilities

To give an idea of how some of the more important utilities operate, some examples may be useful:

- We can use xfs_admin to define a filesystem label:

```
[root@localhost linux]# xfs_admin -f -L xfs_lv /dev/newvg/logical1
writing all SBs
new label = "xfs_lv"
```

- xfs_info is used to retrieve file system information:

```
[root@localhost linux]# xfs_info /mnt/vol1
meta-data=/mnt/vol1              isize=256    agcount=68, agsize=4267 blks
data     =                       bsize=4096   blocks=287744, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=0
naming   =version 2              bsize=4096
log      =internal               bsize=4096   blocks=1200
realtime =none                   extsz=65536  blocks=0, rtextents=0
```

- xfs_db (debug) provides an interactive debug shell to overcome particular problems:

```
[root@localhost linux]# xfs_db -f /dev/newvg/logical1
xfs_db:
```

- A particularly important utility is xfs_check. This command verifies whether an XFS file system is consistent. Because XFS is a journaled file system, xfs_check is normally run only when you believe the file system has a consistency problem:

```
[root@localhost linux]# xfs_check /dev/newvg/logical1
```

- The last utility we wish to mention, xfs_growfs, is used to extend an XFS volume. Suppose we want to enlarge the space available to the XFS file system:

  a. First we extend logical volume1:

```
[root@localhost linux]# lvextend -L+1G /dev/newvg/logical1
```

  b. Then we can let the XFS file system grow to fill the expanded logical volume:

```
[root@localhost linux]# xfs_growfs /mnt/vol1/
meta-data=/mnt/vol1              isize=256    agcount=6, agsize=4267 blks
data     =                       bsize=4096   blocks=25600, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=0
naming   =version 2              bsize=4096
log      =internal               bsize=4096   blocks=1200
realtime =none                   extsz=65536  blocks=0, rtextents=0
data blocks changed from 25600 to 287744
```

Unfortunately, it is not possible to shrink an XFS file system as we did with ReiserFS.

For more information on XFS, please refer to:

http://linux-xfs.sgi.com/projects/xfs/index.html

4.5.3 JFS

The IBM JFS file system for Linux is only available as a set of patches against kernel versions 2.4.0 to 2.4.5. To obtain the software, download the tarball file from the JFS Web site at:

http://oss.software.ibm.com/developerworks/opensource/jfs

and unpack it under your /usr/src directory. The code is organized as a common patch and a number of different patches specifically related to individual kernel versions:

```
[root@localhost src]# tar xvzf jfs-1.0.0-patch.tar.gz
README
jfs-2.2.14-v1.0.0-patch
jfs-2.2.16-v1.0.0-patch
jfs-2.2.18-v1.0.0-patch
jfs-2.4.0-v1.0.0-patch
jfs-2.4.5-v1.0.0-patch
jfs-common-v1.0.0-patch
```

Applying the patches

To apply the JFS patches, download a vanilla kernel source tarball from:

http://www.kernel.org

An example would be kernel 2.4.2 srchive. Then unpack it in your /usr/src directory and issue these commands:

```
[root@localhost linux-2.4.2]# cd /usr/src/linux-2.4.2
[root@localhost linux-2.4.2]# patch -p1 < /usr/src/jfs-common-v1.0.0-patch
[root@localhost linux-2.4.2]# patch -p1 < /usr/src/jfs-2.4.0-v1.0.0-patch
```

For more information, see 4.3, “Compiling and patching” on page 51 and the kernel HOWTO at:

http://www.linuxdoc.org/HOWTO/Kernel-HOWTO.html

After the patches are applied, you have to configure the kernel. To enable the JFS, you have to set CONFIG_JFS_FS by selecting **JFS filesystem support** in the File systems window, as shown in Figure 4-9:

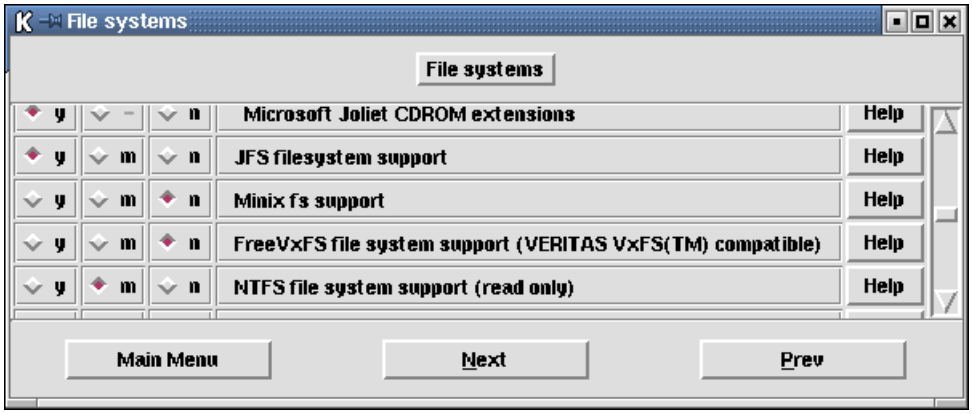


Figure 4-9 Setting kernel options for JFS

As with other file system support, you can choose to compile JFS either as a module or inside the kernel.

JFS utilities

To properly use JFS, you need to install the JFS utilities. There are two ways to do so:

- Build the rpm package and install from it
- Directly compile and install from the source

To build the rpm package you should:

1. Copy the JFS tar file to /usr/src/redhat/SOURCES/
2. Change directories to fs/jfs/utils/SPECS/ under your patched source tree. In our case, the directory is called /usr/src/linux-2.4/fs/jfs/utils/SPECS/

3. Execute **rpm** to build the package

```
[root@rancas SPECS]# rpm -ba jfsprogs.spec
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.56430
+ umask 022
+ cd /usr/src/redhat/BUILD
...
Wrote: /usr/src/redhat/SRPMS/jfsprogs-1.0.0-1.src.rpm
Wrote: /usr/src/redhat/RPMS/i386/jfsprogs-1.0.0-1.i386.rpm
Executing(%clean): /bin/sh -e /var/tmp/rpm-tmp.53861
+ umask 022
+ cd /usr/src/redhat/BUILD
+ rm -rf /var/tmp/jfsprogs-buildroot
+ exit 0
```

4. Install the package, which you can see, in our example, was created in /usr/src/redhat/RPMS/i386/:

```
[root@rancas utils]# cd /usr/src/redhat/RPMS/i386/
[root@rancas i386]# rpm -Uvh jfsprogs-1.0.0-1.i386.rpm
Preparing...                ########################################### [100%]
   1:jfsprogs               ########################################### [100%]
```

To build utilities from the code source, follow these steps:

1. Change directory to fs/jfs/utils/ under your patched source tree. In our case, the directory is called /usr/src/linux-2.4/fs/jfs/utils/
2. Execute **make** and **make install**:

If you chose to create an rpm package, use the rpm utility to verify that your system has the appropriate utilities installed:

```
[root@rancas i386]# rpm -ql jfsprogs
/sbin
/sbin/fsck.jfs
/sbin/logdump
/sbin/logredo
/sbin/mkfs.jfs
/sbin/xchkdmp
/sbin/xchklog
/sbin/xpeek
/usr/man
/usr/man/man8
/usr/man/man8/fsck.jfs.8.gz
/usr/man/man8/logdump.8.gz
/usr/man/man8/logredo.8.gz
/usr/man/man8/mkfs.jfs.8.gz
/usr/man/man8/xchkdmp.8.gz
/usr/man/man8/xchklog.8.gz
/usr/man/man8/xpeek.8.gz
/usr/share/doc/jfsprogs-1.0.0
/usr/share/doc/jfsprogs-1.0.0/jfs.txt
```

Creating a JFS file system

To create a JFS file system on a new logical volume, follow these steps, using your own partition and volume information where appropriate:

1. Use the **mkfs** command to create the new file system, using the `-t` flag to specify JFS:

```
[root@rancas i386]# mkfs -t jfs /dev/newvg/logical3
mkfs.jfs development version: $Name: v1_0_0 $
```

```
Warning!  All data on device /dev/newvg/logical3 will be lost!

Continue? (Y/N) y

Format completed successfully.

512000 kilobytes total disk space.
```

2. Just as for the other file systems, you can mount the JFS file system in a directory and create a text file as a simple test that it can be accessed:

```
[root@rancas i386]# mkdir /mnt/vol3
[root@rancas i386]# mount -t jfs /dev/newvg/logical3 /mnt/vol3/
[root@rancas i386]# echo "First file in JFS filesystem" > /mnt/vol3/firstfile
[root@rancas i386]# ls -l /mnt/vol3/
totale 4
-rw-r--r--    1 root     root           32 lug  5 11:22 firstfile
```

Checking the JFS file system

The fsck.jfs utility can be used to solve JFS problems. The command listing below shows a sample session using the utility. You should unmount the file system before using fsck.jfs:

```
[root@rancas /root]# umount /dev/newvg/logical3
[root@rancas /root]# fsck.jfs -d -f:2 /dev/newvg/logical3
The current device is:  /dev/newvg/logical3
(chklog) FSCK  The current device is:  /dev/newvg/logical3

(chklog) FSCK  Open(...READ/WRITE EXCLUSIVE...) returned rc = 0

(chklog) FSCK  Primary superblock is valid.

(chklog) FSCK  The type of file system for the device is JFS.

...

File system is clean.
(chklog) FSCK  Filesystem is clean.

(chklog) FSCK  All observed inconsistencies have been repaired.

(chklog) FSCK  Filesystem has been marked clean.

(chklog) FSCK **** Filesystem was modified. ****

(chklog) FSCK  processing terminated:  7/5/2001 11.28.1   with return code: 0.
```

fsck.jfs performs a number of useful functions. According to its definition from the man pages, it i*s used to replay the JFS transaction log, check a JFS formatted device for errors, and fix any errors found*.

The logdump utility

Another useful utility is logdump, which dumps log entries into a text file. Here is some sample output:

```
[root@rancas /root]# logdump /dev/newvg/logical3
Vol_Label : /dev/newvg/logical3
JFS_LOGDUMP: The current JFS log has been dumped into ./jfslog.dmp
[root@rancas /root]# cat jfslog.dmp
JOURNAL SUPERBLOCK:
-----------------------------------------------------
   magic number: x 87654321
```

```
   version     : x 1
   serial      : x 2
   size        : t 512 pages (4096 bytes/page)
   bsize       : t 4096 bytes/block
   l2bsize     : t 12
   flag        : x 10200900
   state       : x 0
   end         : x 2444

=====================================================

logrec d 0   Logaddr= x 2468   Nextaddr= x 2444   Backchain = x 0

****************************************************************
LOG_SYNCPT     (type = d 16384)     logtid = d 0

   data length = d 0
   sync = x 0
****************************************************************


----------------------------------------------------------------------
```

At this point in the development of JFS, the utility to resize JFS file systems is under test and not generally available. This function is documented in the TODO list on the JFS Web site.

4.6 Miscellaneous file systems

This section gives a brief overview of some of the other file systems that are available, from the point of view of kernel support.

4.6.1 NFS

Network File System support is already present in the Red Hat 7.1 standard installation and in the SGI extension. A good explanation of NFS is provided in *The Linux Network Administrator's Guide, Second Edition*, available from:

http://www.linuxdoc.org/guides.html

Additional information is available in the man pages and the NFS-HOWTO.

To configure NFS for kernel compilation, you need to set CONFIG_NFS_FS by selecting **NFS file system support** in the Network File Systems window within File systems in xconfig, as you can see in Figure 4-10:

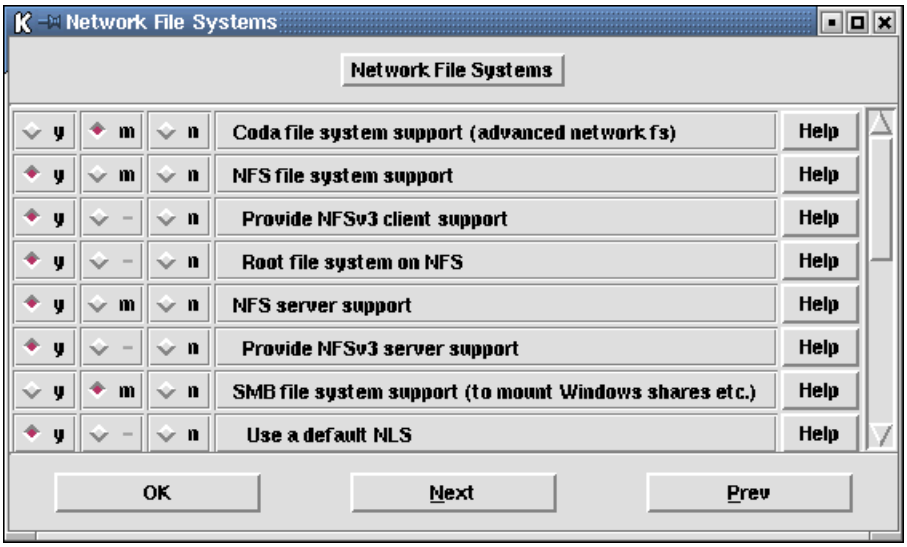


Figure 4-10 Setting kernel options for NFS support

To enable NFSv3 you have to set CONFIG_NFS_V3 by selecting **Provide NFSv3 client support** (for client-side support) and CONFIG_NFSD_V3 by selecting **Provide NFSv3 server support** (for server-side support).

4.6.2 Smbfs

Smbfs support is needed if you wish to mount Windows 95/98/NT/2000 shares and access them as a Linux directory. This feature is supported only if the Windows machines are using TCP/IP as the underlying transport protocol.

Smbfs support is already present in the Red Hat 7.1 standard installation, including the SGI extension.

To configure smbfs for kernel compilation, you need to set CONFIG_SMB_FS by selecting **SMB file system support (to mount Windows shares etc.)** in the Network File Systems window within File systems in xconfig. This option is shown in Figure 4-10.

To access smbfs shares, you need to install samba-client-2.0.7-36.rpm from Red Hat distribution or a later release.

4.6.3 Raw I/O

You do not usually want to access devices using raw I/O access. This feature is used in particular situations, as discussed briefly in 1.4.4, “Raw I/O devices” on page 22.

From the man pages for the **raw** command, we can read:

[…] Once bound to a block device, a raw device can be opened, read and written, just like the block device it is bound to. However, the raw device does not behave exactly like the block device. In particular, access to the raw device bypasses the kernel's block buffer cache entirely: all I/O is done directly to and from the address space of the process performing the I/O. If the underlying block device driver can support DMA, then no data copying at all is required to complete the I/O.

BUGS

[…] Raw I/O devices do not maintain cache coherency with the overwrite data already in the buffer cache, the buffer cache will no longer correspond to the contents of the actual storage device underneath. This is deliberate, but is regarded either a bug or a feature depending on who you ask!

As we explained, the raw access to block devices is done by a special raw I/O device association with the target block device. This is actually done using the raw utility that can be found in the util-linux-2.10s-12.rpm package from Red Hat.

You can query the binding of an existing raw device with the command:

```
raw -l /dev/raw/rawN
```

where `N` is an integer indicating the raw device (created under the /dev/raw/ directory) being queried. The command

```
raw -a
```

queries all raw devices.

There are 255 raw devices available, plus a master raw device, usually called rawctl (with minor number 0), used to control the bindings on the others. In the Red Hat standard distribution, the system comes with all raw devices ready to use in the /dev directory, but if you installed SGI's version, you have to set up the raw devices with the **mknod** command, for example:

```
mknod /dev/rawctl c 162 0
chmod 600 /dev/rawctl     # to allow root only access

mkdir /dev/raw
chmod 755 /dev/raw

mknod /dev/raw1 c 162 1
mknod /dev/raw2 c 162 2
mknod /dev/raw3 c 162 3
mknod /dev/raw4 c 162 4
```

To have access to a device through raw I/O, you must bind it to a raw device. This is done using either:

```
raw /dev/rawN <major> <minor>
```

using the block-device major and minor numbers, or

```
raw /dev/rawN /dev/<blockdev>
```

using an existing device in /dev, for example:

```
raw /dev/raw/raw1 /dev/scsi/host2/bus0/target0/lun0/disc
```

or

```
raw /dev/raw/raw1 8 0
```

Note: Permissions on the raw device are independent of the block device to which it is bound. This means it is necessary to set permissions correctly for whichever association is set.

We can summarize the steps we have discussed, using a practical example:

1. We create the raw directory under /dev and change its access rights:

   ```
   [root@localhost linux-2.4]# mkdir /dev/raw
   [root@localhost linux-2.4]# chmod 755 /dev/raw
   ```

2. Then we create the rawctl device file and at least one raw device under /dev/raw:

   ```
   [root@localhost linux-2.4]# mknod /dev/rawctl c 162 0
   [root@localhost linux-2.4]# mknod /dev/raw/raw1 c 162 1
   ```

3. To bind the first SCSI disk to the raw device, we have to use the `raw` utility:

   ```
   [root@localhost linux-2.4]# raw /dev/raw/raw1 /dev/scsi/host2/bus0/target0/lun0/disc
   /dev/raw/raw1:  bound to major 8, minor 0
   ```

4. To verify the bindings, we use:

   ```
   [root@localhost linux-2.4]# raw -a
   /dev/raw/raw1:  bound to major 8, minor 0
   ```

If you are using a Red Hat-based system, you can automate the process using the rawdevices service. You can configure this service to automatically assign raw devices as system boots. This can be done by configuring the /etc/sysconfig/rawdevices file. An example is given below:

```
# file /etc/sysconfig/rawdevices
# raw device bindings
# format:  <rawdev> <major> <minor>
#          <rawdev> <blockdev>
# raw device assignment: /dev/raw/raw1 /dev/scsi/host2/bus0/target0/lun0/disc
          /dev/raw/raw1 8 0
```

After the file is modified, you can update the service status by restarting the service:

```
[root@localhost linux-2.4.2]# /etc/init.d/rawdevices restart
Assigning devices:
           /dev/raw/raw1  -->   8 0
/dev/raw/raw1:  bound to major 8, minor 0
done
```

To have the rawdevices service start in your runlevel, verify that the appropriate link exists in the corresponding /etc/rcX.d directory (where X is your runlevel number).

4.7 Netfilter

The Netfilter framework is already present in the standard Red Hat 7.1 installation and in the SGI version. If you want to configure kernel compilation to include Netfilter, you need to set specific flags during kernel configuration.

To enable packet filtering you have to set CONFIG_NETFILTER by selecting **Network packet filtering (replaces ipchains)** in the Networking options window in xconfig, as shown in Figure 4-11:

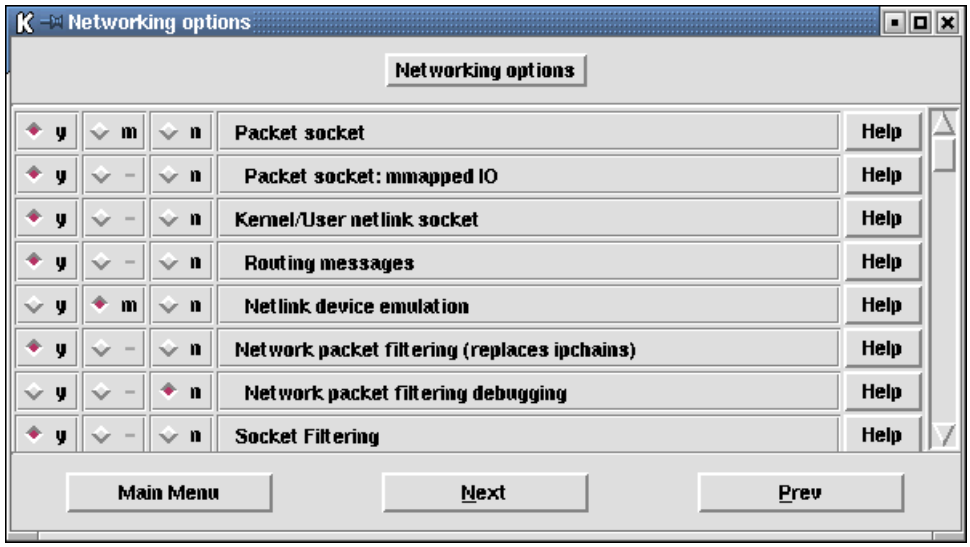


Figure 4-11 Enabling packet filtering

You could also set CONFIG_NETFILTER_DEBUG by selecting **Network packet filtering debugging** to enable additional messages from Netfilter, if you are having problems.

For detailed configuration of Netfilter, you have to select the **IP: Netfilter Configuration** submenu in the Networking options window. The resulting window is shown in Figure 4-12:

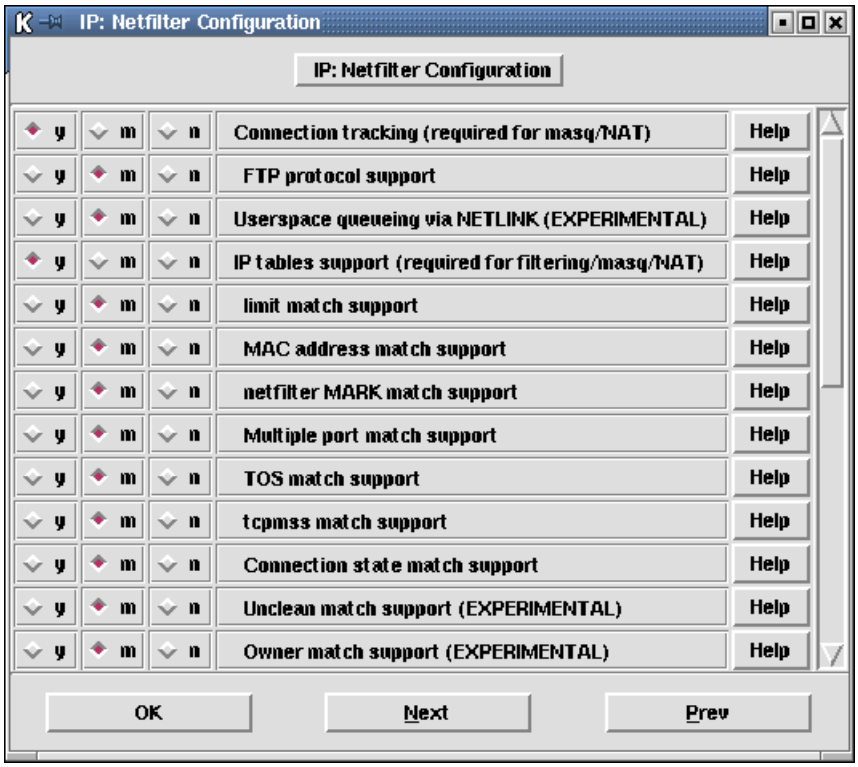


Figure 4-12 Kernel configuration options for Netfilter

A number of flags are available in the IP: Netfilter Configuration window. There are too many to describe in detail here, so we refer you to the online help for summaries of their functions. A more detailed explanation of the features relevant to this topic are given in 2.2, “Linux 2.4 firewalling and Network Address Translation” on page 30.

4.7.1 iptables under Red Hat 7.1 systems.

On a newly installed Red Hat 7.1 system, iptables is actually masqueraded as ipchains. As we said in 2.2.2, “IP Tables” on page 33, a compatibility module can be brought into play to use ipchains rules under Netfilter.

If we try to use iptables on a Red Hat 7.1 Linux box immediately after installation, we obtain output similar to that shown below:

```
[root@localhost linux]# iptables -L
/lib/modules/2.4.2-SGI_XFS_1.0/kernel/net/ipv4/netfilter/ip_tables.o: init_module: Device
or resource busy
Hint: insmod errors can be caused by incorrect module parameters, including invalid IO or
IRQ parameters
/lib/modules/2.4.2-SGI_XFS_1.0/kernel/net/ipv4/netfilter/ip_tables.o: insmod
/lib/modules/2.4.2-SGI_XFS_1.0/kernel/net/ipv4/netfilter/ip_tables.o failed
/lib/modules/2.4.2-SGI_XFS_1.0/kernel/net/ipv4/netfilter/ip_tables.o: insmod ip_tables
failed
iptables v1.2.1a: can't initialize iptables table `filter': iptables who? (do you need to
insmod?)
Perhaps iptables or your kernel needs to be upgraded.
```

This is due to the fact that the ipchains emulation module is running. To avoid this problem, follow these steps:

1. Stop the ipchains service, using this command:

```
[root@localhost linux]# /etc/init.d/ipchains stop
Flushing all chains: [  OK  ]
Removing user defined chains: [  OK  ]
Resetting built-in chains to the default ACCEPT policy:[  OK  ]
```

2. Then list the kernel modules that are loaded in memory; note that ipchains is present but unused:

```
[root@localhost linux]# lsmod
Module                  Size  Used by
...
ipchains               38304   0  (unused)
...
```

3. Now, remove the ipchains module using the **rmmod** command:

```
[root@localhost linux]# rmmod ipchains
```

4. At this point, you can start iptables:

```
[root@localhost linux]# /etc/init.d/iptables start
```

5. lsmod can be used once more to see if everything is as planned:

```
[root@localhost linux]# lsmod
Module                  Size  Used by
iptable_mangle          2272   0  (autoclean) (unused)
iptable_nat            15776   0  (autoclean) (unused)
ip_conntrack           15632   1  (autoclean) [iptable_nat]
iptable_filter          2304   0  (autoclean) (unused)
ip_tables              11072   5  [iptable_mangle iptable_nat iptable_filter]
...
```

   This list shows that all of the Netfilter elements we discussed in 2.2.1, “Netfilter framework” on page 30 are present.

6. It is now possible to list the tables’ contents:

```
[root@localhost linux]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

7. You can also add a new rule:

```
[root@localhost linux]# iptables -A FORWARD -p tcp --syn -m limit --limit 1/s -j ACCEPT
[root@localhost linux]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere           tcp flags:SYN,RST,AC
K/SYN limit: avg 1/sec burst 5

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination I
```

8. The last step is to modify the service startup at system boot. If you want to use your system at init level 3 (network, multiuser system with non-graphical login), for example, you have to modify the appropriate file in the /etc/rc3.d/ directory:

```
[root@localhost linux]# ls /etc/rc3.d/
K03rhnsd       K30mcserv     K65kadmin    S10network     S60lpd
K05innd        K34yppasswdd  K65kprop     S12syslog      S80isdn
K12mysqld      K35dhcpd      K65krb524    S13portmap     S80pppoe
K12postgresql  K35smb        K65krb5kdc   S14nfslock     S80sendmail
K15httpd       K35vncserver  K74nscd      S17keytable    S85gpm
K16rarpd       K40mars-nwe   K74ntpd      S20pcmcia      S90crond
K20bootparamd  K45arpwatch   K74ups       S20random      S90xfs
K20nfs         K45named      K74ypserv    S25netfs       S95anacron
K20rstatd      K50snmpd      K75gated     S26apmd        S99linuxconf
K20rusersd     K50tux        K96irda      S28autofs      S99local
K20rwalld      K54pxe        S05kudzu     S40atd
K20rwhod       K55routed     S06reconfig  S55sshd
K25squid       K61ldap       S08ipchains  S56rawdevices
K28amd         K65identd     S08iptables  S56xinetd
```

   Specifically, you have to disable ipchains initialization by renaming the underlined link in the /etc/rc3.d directory:

```
[root@localhost linux]# mv /etc/rc3.d/S08ipchains /etc/rc3.d/K08ipchains
```

   Renaming the file in this way will disable ipchains and allow iptables to be active at the next reboot.

4.8 Miscellaneous network features

In this last section, we examine several other important new network features with regard to kernel compilation.

4.8.1 Kernel Web servers: kHTTPd and TUX.

Linux kernel 2.4 comes with an integrated http server. The default kernel Web daemon is called kHTTPd, while the Red Hat kernel daemon is called TUX (Threaded linUX application protocol accelerator layer).

TUX

The TUX module is ready to use under Red Hat 7.1, if you are using the SGI installer. Make sure that you also install the TUX utilities from the tux-2.0.26-1.rpm package (or subsequent versions).

You can then enable TUX by starting the appropriate service:

```
[root@localhost networking]# /etc/init.d/tux start
Starting tux: [  OK  ]
```

When it is running, the daemon listens on the http port 80 for incoming requests, as we can see using the **netstat** command:

```
[root@localhost networking]# netstat -an | grep :80
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
```

Kernel configuration

To compile TUX from scratch, you need to apply the correct patch from:

http://people.redhat.com/~mingo/TUX-patches

or use the package kernel-source-x.y.z.rpm to install the source code on your system.

You also need to set CONFIG_HTTP by selecting **Threaded linUX application protocol accelerator layer (TUX)** in the Networking options window in xconfig (see TUX figure):

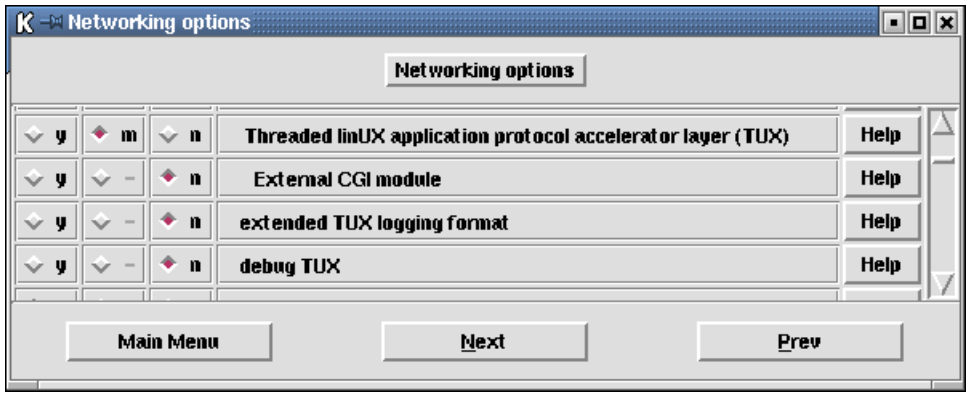


Figure 4-13 Configuring the kernel for TUX

You can also enable experimental support for starting CGI programs from within the kernel in this window if you set CONFIG_HTTP_EXTCGI. This can be done by selecting **External CGI module**. Two more options allow you to activate extended logging and debugging features by selecting **extended TUX logging format** and **debug TUX**.

To test TUX, you can use the Apache benchmark (ab) utility that is included with the apache-1.3.19-5.rpm package (among other versions):

```
[root@localhost networking]# ab -c 100 -n 1000 http://localhost/index.html
This is ApacheBench, Version 1.3c <$Revision: 1.44 $> apache-1.3
Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Copyright (c) 1998-2000 The Apache Group, http://www.apache.org/

Server Software:        TUX/2.0
Server Hostname:        localhost
Server Port:            80

Document Path:          /index.html
Document Length:        2890 bytes

Concurrency Level:      100
Time taken for tests:   0.530 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      3120612 bytes
HTML transferred:       2988260 bytes
Requests per second:    1886.79
Transfer rate:          5887.95 kb/s received

Connnection Times (ms)
              min   avg   max
Connect:        0    20    30
Processing:    17    27    52
Total:         17    47    82
```

More information about TUX can be found at these Web sites:

http://www.redhat.com/products/software/webservers/tux
http://lwn.net/2000/0907/a/tux-readme.php3

kHTTPd

To enable kHTTPd, you need to recompile the kernel. Under **`Code maturity level options`** in xconfig (see Figure 4-1 on page 53), you have to select **Prompt for development and/or incomplete code/drivers**, and in the Networking options window you must set CONFIG_KHTTPD by selecting **Kernel httpd acceleration (EXPERIMENTAL)**, as shown in Figure 4-14:

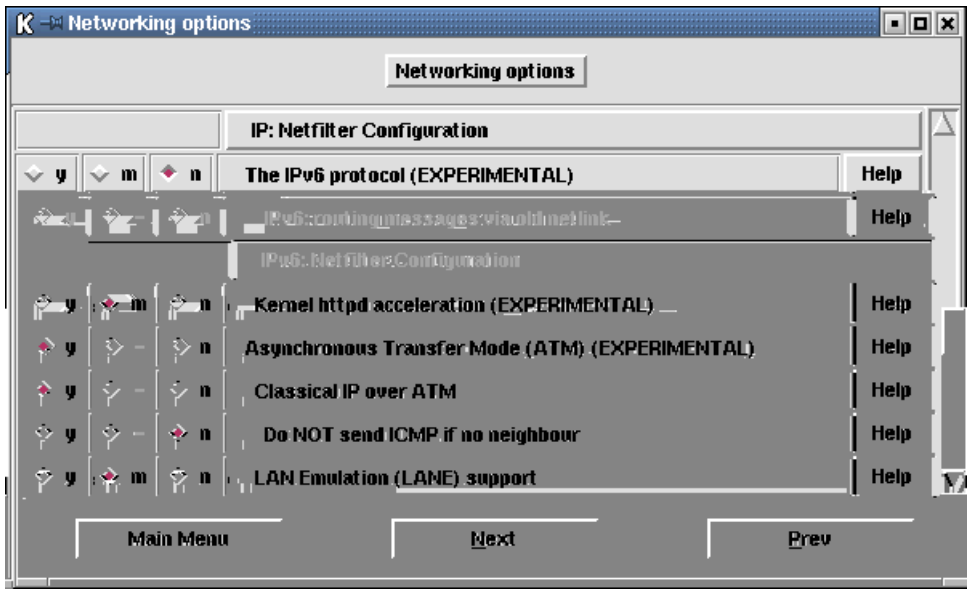


Figure 4-14 Setting the kHTTPd compilation option

After installing the kHTTPd module with insmod:

```
[root@localhost networking]insmod khttpd
```

KHTTPd can be configured through the /proc virtual file system, as follows:

- To start the daemon:

  ```
  [root@localhost networking]echo 1 > /proc/sys/net/khttpd/start
  ```

- To stop and unload the daemon., and then uninstall the module from the kernel:

  ```
  [root@localhost networking]echo 1 > /proc/sys/net/khttpd/stop
  [root@localhost networking]echo 1 > /proc/sys/net/khttpd/unload
  [root@localhost networking]rmmod httpd
  ```

- To set the kHTTPd listening port to 80:

  ```
  [root@localhost networking]echo 80 > /proc/sys/net/khttpd/serverport
  ```

- To set the port to which the standard http daemon (such as Apache) is listening:

  ```
  [root@localhost networking]echo 8080 > /proc/sys/net/khttpd/clientport
  ```

- To set the html pages base directory to /var/www/html/:

  ```
  [root@localhost networking]echo /var/www/html/  > /proc/sys/net/khttpd/documentroot
  ```

- To set the cgi directory to /var/www/html/:

  ```
  [root@localhost networking]echo /var/www/cgi/  > /proc/sys/net/khttpd/dynamic
  ```

Other features of kHTTPd can be configured in a similar fashion. To learn more about kHTTPd configuration, read the README file inside the net/khttpd/ directory under your source main tree, and visit:

http://www.fenrus.demon.nl/

4.8.2 ATM

Asynchronous Transfer Mode support is already present in the standard Red Hat 7.1 installation and in the SGI version. To compile ATM support in the kernel, you need to set many different configuration options, depending on your hardware requirements.

Here we describe the various flags and provide the related help information that is accessible through the configuration utilities. For a more detailed explanation of each feature, we refer you to the ATM on Linux Web site at:

http://lrcwww.epfl.ch/linux-atm

In the Networking options window, you find the following settings (see Figure 4-15):

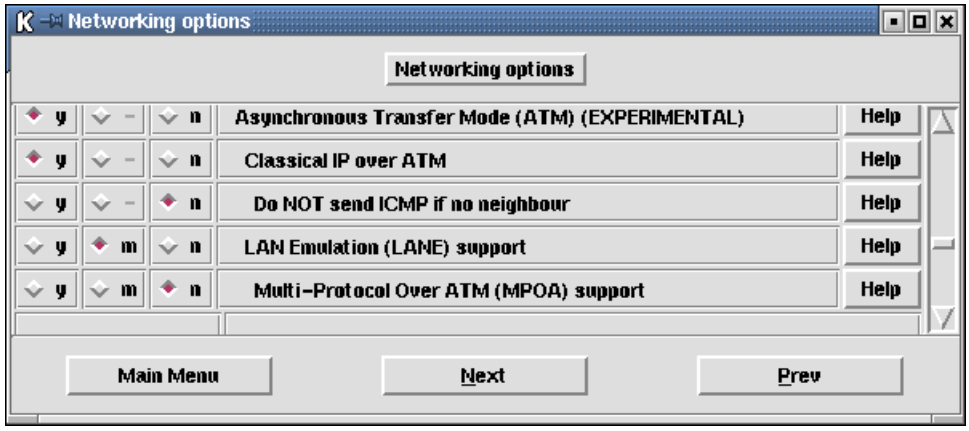


Figure 4-15 ATM settings for kernel compilation

- **Asynchronous Transfer Mode (ATM) (EXPERIMENTAL)**, which sets the CONFIG_ATM_ATM flag.

  ATM is a high-speed networking technology for Local Area Networks and Wide Area Networks. It uses a fixed packet size and is connection oriented, allowing for the negotiation of minimum bandwidth requirements.

- **Classical IP over ATM**, which sets the CONFIG_ATM_CLIP flag.

  Classical IP over ATM for permanent virtual circuits (PVCs) and switched virtual circuits (SVCs), supporting InARP and ATMARP. If you want to communicate with other IP hosts on your ATM network, you will typically either say Y here or to 'LAN Emulation (LANE)' below.

- **Do NOT send ICMP if no neighbour**, which sets the CONFIG_ATM_CLIP_NO_ICMP flag.

  Normally, an 'ICMP host unreachable' message is sent if a neighbour cannot be reached because there is no virtual circuit (VC) to it in the kernel's ATMARP table. This may cause problems when ATMARP table entries are briefly removed during revalidation. If you say Y here, packets to such neighbours are silently discarded instead.

- **LAN Emulation (LANE) support**, which sets the CONFIG_ATM_LANE flag.

  LAN Emulation emulates services of existing LANs across an ATM network. Besides operating as a normal ATM end station client, the Linux LANE client can also act as a proxy client bridging packets between ELAN and Ethernet segments. You need LANE if you want to try MPOA.

- **Multi-Protocol Over ATM (MPOA) support**, which sets the CONFIG_ATM_MPOA flag.

  Multi-Protocol Over ATM allows ATM edge devices such as routers, bridges and ATM attached hosts to establish direct ATM VCs across subnetwork boundaries. These shortcut connections bypass routers enhancing overall network performance.

Under **`Network device support`** in xconfig (see Figure 4-1 on page 53), in the **ATM drivers** subsection, there is an option called ATM over TCP. This sets the CONFIG_ATM_TCPATM flag and allows ATM communication over the TCP driver. This is mainly used for development and testing.

This window also offers support for a number of network interface card (NIC) drivers, including:

- Efficient Networks ENI155P
- Fujitsu FireStream (FS50/FS155)
- ZeitNet ZN1221/ZN1225
- IDT 77201 (NICStAR) (ForeRunnerLE)
- Madge Ambassador (Collage PCI 155 Server)
- Madge Horizon [Ultra] (Collage PCI 25 and Collage PCI 155 Client)
- Interphase ATM PCI x575/x525/x531
- FORE Systems 200E-series

4.8.3 Devfs

Devfs is enabled as standard if you use SGI's installation CD, but is not configured under the Red Hat 7.1 base installation. Successful devfs installation can be a complex process, so we recommend that you use the devfs documentation, which can be found at:

`http://www.atnf.csiro.au/~rgooch/linux/docs/devfs.html`

The documentation provides suggestions to help you solve specific problems you may encounter with devfsd, along with system library versions, secure tty configuration, Xfree86, unsupported drivers, and so on. Another useful document, which is focused on devfs and SCSI, can be found at:

`http://www.torque.net/sg/devfs_scsi.html`

This section is not intended to be a detailed guide to configuring the devfs feature, but gives an outline of the basic steps involved:

1. Before installing devfs, make sure the devfs utilities are already installed in your system by using the **`rpm`** command:

```
[root@localhost /root]# rpm -ql devfsd
/etc/devfsd.conf
/etc/modules.devfs
/sbin/devfsd
/usr/share/man/man8/devfsd.8.gz
```

The devfsd package includes the devfsd daemon, which ensures compatibility with old device names by dynamically creating appropriate symlinks. It also contains basic configuration files and a devfsd man entry. The latest release of the devfsd daemon can be found at:

http://www.atnf.csiro.au/~rgooch/linux/

2. You need to edit your main system boot script in order to start the devfsd daemon before any file system checks. In a Red Hat system, you need to edit /etc/rc.d/rc.sysinit. As an example, we reproduce the first part of the rc.sysinit script installed by SGI's installer:

```
#!/bin/bash
#
# /etc/rc.sysinit - run once at boot time
#
# Taken in part from Miquel van Smoorenburg's bcheckrc.
#

# Rerun ourselves through initlog
if [ -z "$IN_INITLOG" ]; then
 [ -f /sbin/initlog ] && exec /sbin/initlog $INITLOG_ARGS -r /etc/rc.sysinit
fi

# If we're using devfs, start devfsd now - we need the old device names
[ -e /dev/.devfsd -a -x /sbin/devfsd ] && /sbin/devfsd /dev
```

The last line in this sample shows the entry for the devfsd initialization and can be used as a prototype for your system.

3. To enable devfs on Red Hat 7.1 systems, you must compile the kernel (it comes by default with SGI's installer). The essential configuration option can be found under the File systems section of xconfig (see Figure 4-16):

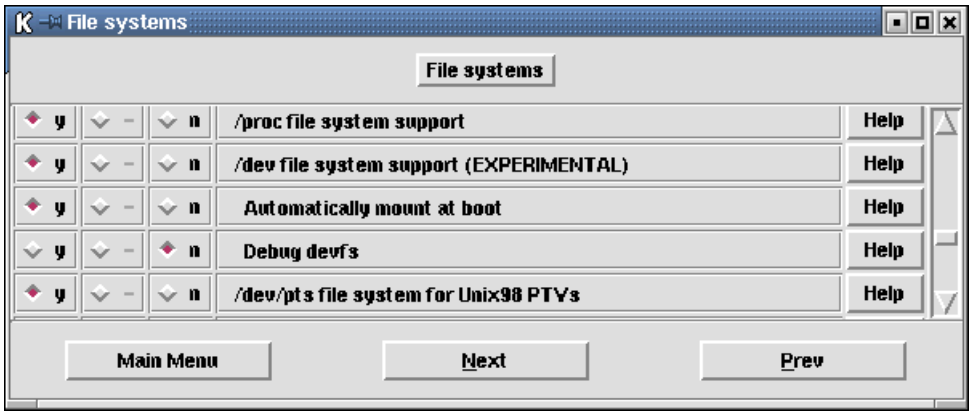


Figure 4-16 Enabling devfs support

You have to set the CONFIG_DEVFS_FS flag by selecting **/dev file system support (EXPERIMENTAL)** to compile devfs into the kernel code.

4. Set the CONFIG_DEVFS_MOUNT flag by selecting **Automatically mount at boot** if you want devfs to be mounted at boot time.

5. The last option to set, if you wish, is **Debug devfs** (setting the CONFIG_DEVFS_DEBUG flag), which enables verbose debug messages.

Boot-time options

The option CONFIG_DEVFS_MOUNT can be overridden by entering the `devfs=nomount` boot option at the lilo prompt or by including the line `append="devfs=nomount"` in your lilo.conf file. Conversely, the devfs=mount option can be used to mount devfs on the /dev directory at boot time if it is not mounted by default. For more information, see the bootparam and lilo.conf man pages.

Note: If devfs is mounted automatically at boot-time, and the devfsd daemon is not correctly configured, your system could be left in single user mode or rescue mode. This can happen because the /etc/fstab file refers to partitions using old names. Since devfs is mounted on /dev directory, it covers old names, preventing the system from mounting the affected partitions.

If you compile the devfs-enabled kernel with CONFIG_DEVFS_MOUNT disabled (you could also use the `nomount` option at boot-time), you can have a look at the devfs structure by mounting it on a directory other than /dev. Here is an example:

1. Make a new directory:

```
[root@localhost linux]# mkdir /devfs
```

2. Mount devfs on the new directory:

```
[root@localhost linux]# mount -n -t devfs none /devfs/
```

3. As an example, we can see how SCSI hardware is named under devfs. First of all, use the /proc filesystem to discover what SCSI devices are attached to the system:

```
[root@localhost linux]# cat /proc/scsi/scsi
Attached devices:
Host: scsi2 Channel: 00 Id: 00 Lun: 00
  Vendor:  IBM     Model:  SERVERAID       Rev:  1.0
  Type:   Direct-Access                    ANSI SCSI revision: 01
Host: scsi2 Channel: 00 Id: 01 Lun: 00
  Vendor:  IBM     Model:  SERVERAID       Rev:  1.0
  Type:   Direct-Access                    ANSI SCSI revision: 01
Host: scsi2 Channel: 00 Id: 15 Lun: 00
  Vendor:  IBM     Model:  SERVERAID       Rev:  1.0
  Type:   Processor                        ANSI SCSI revision: 01
Host: scsi2 Channel: 01 Id: 15 Lun: 00
  Vendor: IBM      Model: LN V1.2Tower     Rev: B004
  Type:   Processor                        ANSI SCSI revision: 02
```

4. Under the /devfs directory (or under /dev if devfs is installed by default in your system), you can find the scsi subdirectory, assuming your system has SCSI hardware installed:

```
[root@localhost linux]# du /devfs/scsi/ -a
0       /devfs/scsi/host2/bus0/target0/lun0/disc
0       /devfs/scsi/host2/bus0/target0/lun0/part1
0       /devfs/scsi/host2/bus0/target0/lun0/part2
0       /devfs/scsi/host2/bus0/target0/lun0/part5
0       /devfs/scsi/host2/bus0/target0/lun0/part6
0       /devfs/scsi/host2/bus0/target0/lun0/part7
...
0       /devfs/scsi/host2/bus0/target1/lun0/disc
0       /devfs/scsi/host2/bus0/target1/lun0/part1
...
```

As you can see, devfs creates a tree structure to map SCSI disks and the channel, ID and LUN numbers associated with them:

- `/devfs/scsi/host2/bus0/target0/lun0/disc` indicates the entire disk on channel 0, ID 0, LUN 0
- `/devfs/scsi/host2/bus0/target0/lun0/part1` indicates the first partition on the same disk, etc.

5. devfs maps devices in different ways. For example, each disk is mapped under /dev/discs/. We can see this by issuing an **ls** command:

```
[root@localhost linux]# ls /dev/discs/disc0/ -l
total 0
brw-------    1 root     root       8,   0 Dec 31  1969 disc
brw-------    1 root     root       8,   1 Dec 31  1969 part1
brw-------    1 root     root       8,   2 Dec 31  1969 part2
brw-------    1 root     root       8,   5 Dec 31  1969 part5
brw-------    1 root     root       8,   6 Dec 31  1969 part6
brw-------    1 root     root       8,   7 Dec 31  1969 part7
```

and also:

```
[root@localhost linux]# ls /dev/sd/c2b0t0u0* -l
lr-xr-xr-x    1 root     root           36 Jun  7 08:21 /dev/sd/c2b0t0u0 ->
../scsi/host2/bus0/target0/lun0/disc
lr-xr-xr-x    1 root     root           37 Jun  7 08:21 /dev/sd/c2b0t0u0p1 ->
../scsi/host2/bus0/target0/lun0/part1
lr-xr-xr-x    1 root     root           37 Jun  7 08:21 /dev/sd/c2b0t0u0p2 ->
../scsi/host2/bus0/target0/lun0/part2
lr-xr-xr-x    1 root     root           37 Jun  7 08:21 /dev/sd/c2b0t0u0p5 ->
../scsi/host2/bus0/target0/lun0/part5
lr-xr-xr-x    1 root     root           37 Jun  7 08:21 /dev/sd/c2b0t0u0p6 ->
../scsi/host2/bus0/target0/lun0/part6
lr-xr-xr-x    1 root     root           37 Jun  7 08:21 /dev/sd/c2b0t0u0p7 ->
../scsi/host2/bus0/target0/lun0/part7
```

and also using the old style names:

```
[root@localhost linux]# ls /dev/sda -l
lr-xr-xr-x    1 root     root           33 Jun  7 08:21 /dev/sda ->
scsi/host2/bus0/target0/lun0/disc
```

Special notices

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere.,The Power To Manage., Anything. Anywhere.,TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other

countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.