Hardware Management

Still another responsibility for the kernel is hardware management.

Any device that the Linux system must communicate with needs driver code inserted inside the kernel code.

The driver code allows the kernel to pass data back and forth to the device, acting as a middleman between applications and the hardware.

Hardware Management

There are two methods used for inserting device driver code in the Linux kernel.

- Drivers compiled in the kernel
- Driver modules added to the kernel

Previously, the only way to insert device driver code was to recompile the kernel.

Each time you added a new device to the system, you had to recompile the kernel code.

This process became even more inefficient as Linux kernels supported more hardware.

Fortunately, Linux developers devised a better method to insert driver code into the running kernel.

Programmers developed the concept of kernel modules to [1]allow you to insert driver code into a running kernel without having to recompile the kernel.

Hardware Management

Also, a [2]kernel module could be removed from the kernel when the device was finished being used.

This greatly simplified and expanded using hardware with Linux.

The Linux system identifies hardware devices as special files, called device files.

There are three different classifications of device files.

- Character
- Block
- Network

Hardware Management

Character device files are for devices that can only **handle data one character at a time**.

Most types of modems and terminals are created as character files.

Block files are for devices that can **handle data in large blocks at a time**, such as **disk drives**.

The **network** file types are used for devices that **use packets to send and receive data**.

This includes network cards and a special loopback device that allows the Linux system to communicate with itself using common network programming protocols.

Hardware Management

Linux creates **special files**, called **nodes**, for each device on the system.

All communication with the device is performed through the **device node**.

Each **node** has a **unique number pair** that identifies it to the Linux kernel.

The number pair includes a **major** and a minor device number.

Similar devices are grouped into the same major device number.

The **minor device** number is used to **identify a specific device within the major device group**.

Filesystem Management

Unlike some other operating systems, the Linux kernel can support different types of filesystems to read and write data to and from hard drives.

Besides having more than a dozen filesystems of its own, Linux can read and write to and from filesystems used by other operating systems, such as Microsoft Windows.

The kernel **must** be **compiled** with support for all types of filesystems that the system will use.

Table 1.2 lists the standard filesystems that a Linux system can use to read and write data.

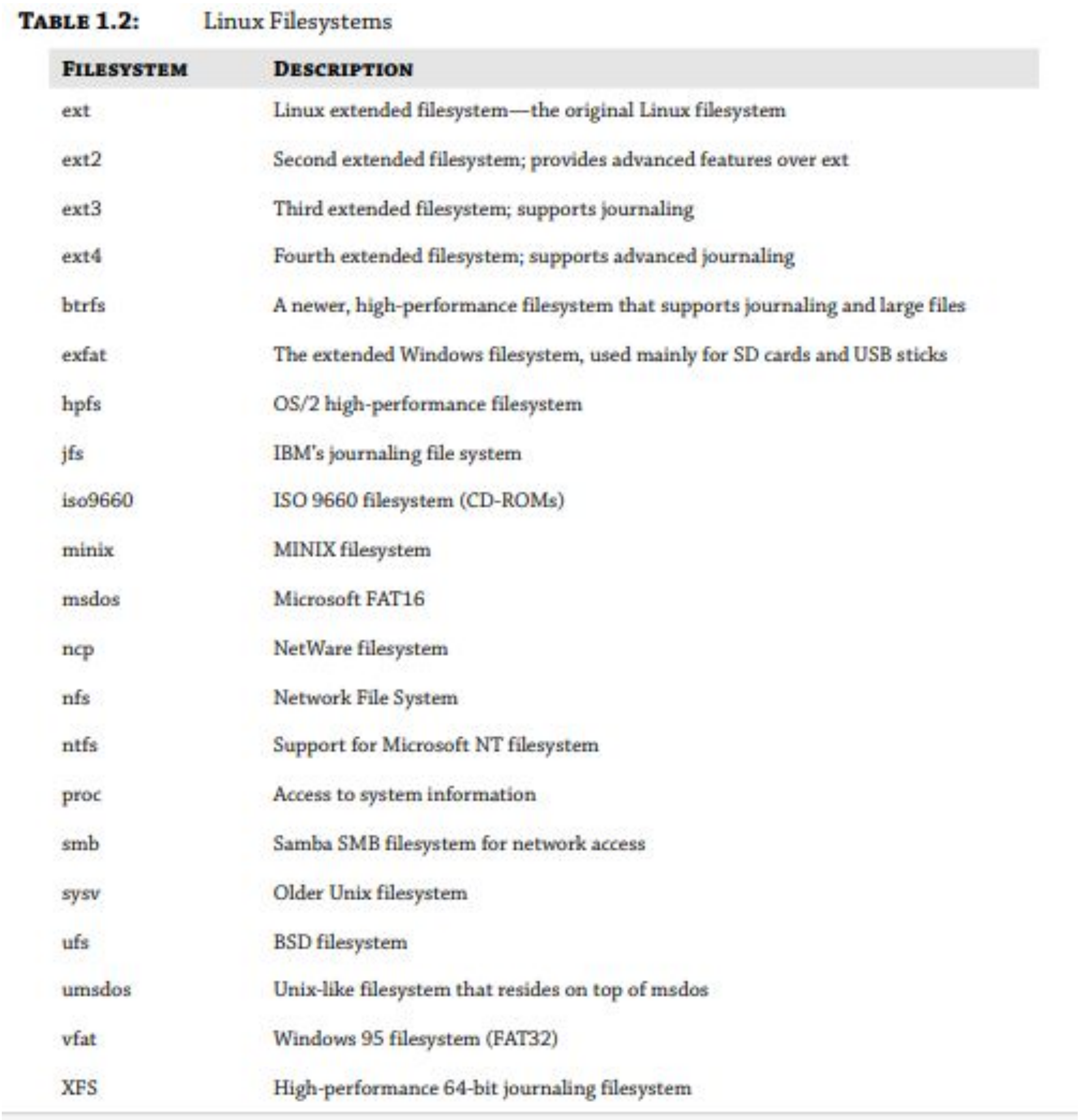

Table 1.2: Linux Filesystems

Filesystem	Description
ext	Linux extended filesystem—the original Linux filesystem
ext2	Second extended filesystem; provides advanced features over ext
ext3	Third extended filesystem; supports journaling
ext4	Fourth extended filesystem; supports advanced journaling
btrfs	A newer, high-performance filesystem that supports journaling and large files
exfat	The extended Windows filesystem, used mainly for SD cards and USB sticks
hpfs	OS/2 high-performance filesystem
jfs	IBM's journaling file system
iso9660	ISO 9660 filesystem (CD-ROMs)
minix	MINIX filesystem
msdos	Microsoft FAT16
ncp	NetWare filesystem
nfs	Network File System
ntfs	Support for Microsoft NT filesystem
proc	Access to system information
smb	Samba SMB filesystem for network access
sysv	Older Unix filesystem
ufs	BSD filesystem
umsdos	Unix-like filesystem that resides on top of msdos
vfat	Windows 95 filesystem (FAT32)
XFS	High-performance 64-bit journaling filesystem

Filesystem Management

Any hard drive that a Linux server accesses must be formatted using one of the filesystem types listed in Table 1.2.

The Linux kernel interfaces with each filesystem using the Virtual File System (VFS).

This provides a standard interface for the kernel to communicate with any type of filesystem. VFS caches information in memory as each filesystem is mounted and used.