Installable File System Overview

System Relationships

Installable File System (IFS) Mechanism defines the relationships among the operating system, the file systems, and the device drivers. The basic model of the system is represented in Figure 1-1.

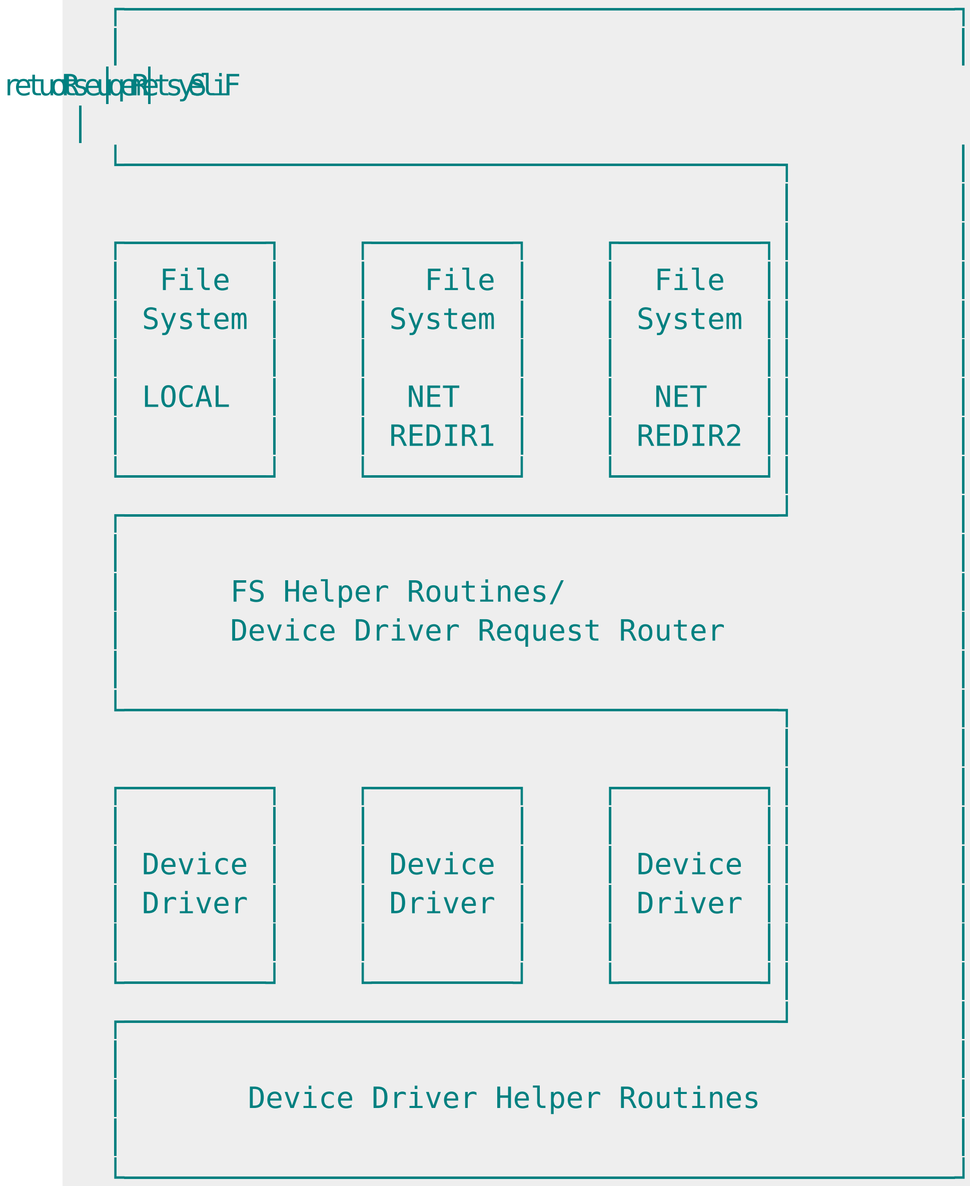
File System Request Router
File System LOCAL
File System NET REDIR1
File System NET REDIR2
FS Helper Routines/ Device Driver Request Router
Device Driver
Device Driver
Device Driver
Device Driver Helper Routines

Figure 1-1. System relationships for Installable File Systems

The file system request router directs file system function calls to the appropriate file system for processing.

The file systems manage file I/O and control the format of information on the storage media. An installable file system (FS) will be referred to as a file system driver (FSD).

The FS Helper Routines provide a variety of services to the file systems.

The device drivers manage physical I/O with devices. Device drivers do not understand the format of information on the media.

File I/O API

Standard file I/O is performed through the Standard File I/O API. The application makes a function call and the file system request router passes the request to the correct file system for processing. See Figure 1-2.

```
                   ┌─────────────┐
                   │ application │
                   └──────┬──────┘
                          ↓
            ┌─────────────┴─────────────┐  Dynamic
            │  Standard File I/O API    │  Link
            └─────────────┬─────────────┘  Library
                          ↓
┌───────────────────────────────────────────────┐
│          File System Request Router           │
└─────────────────────────────────────┐         │
                                      │         │
┌────────┐    ┌────────┐    ┌────────┐│         │
│ File   │    │  File  │    │  File  ││         │
│ System │    │ System │    │ System ││         │
└────────┘    └────────┘    └────────┘│         │
┌─────────────────────────────────────┘         │
│                                               │
└─────────────────────────────────────┐         │
                                      │         │
┌────────┐    ┌────────┐    ┌────────┐│         │
│ Device │    │ Device │    │ Device ││         │
│ Driver │    │ Driver │    │ Driver ││         │
└────────┘    └────────┘    └────────┘│         │
┌─────────────────────────────────────┘         │
│                                               │
└───────────────────────────────────────────────┘
```

Figure 1-2. Standard File I/O

New API may be provided by a file system to implement functions specific to the file system or not supplied through the standard file I/O interface. New API are provided in a dynamic link library that uses the DosFSCtl standard function call to communicate with the specific file system (FSD). See Figure 1-3.

```
                   ┌─────────────┐
                   │ application │
                   └──────┬──────┘
                           │
                          ↓
              ┌──────────────────────────┐
              │ Extended File I/O API    │ Dynamic Link Library
```

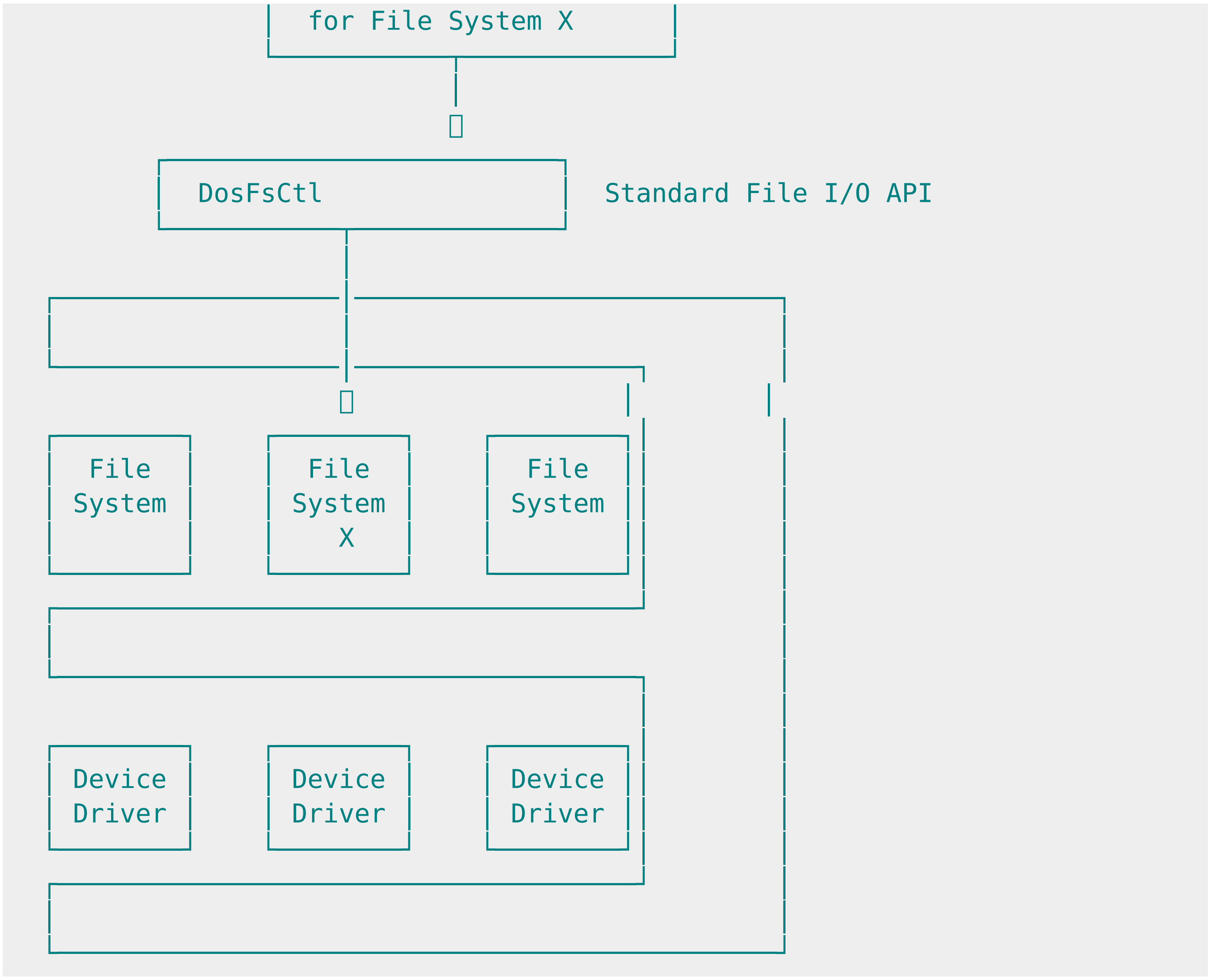

Figure 1-3. Extended File I/O

Buffer Management

In 2.0 the FAT buffer management helpers were removed because of lack of use by any 1.x FSD. FSDs should handle all buffer/cache management themselves.

The FSD moves all data requiring partial sector I/O between the application's buffers and its cache buffers. The FS helper routines initiate the I/O for local file systems.

Volume Management

Volume management (that is, detecting when the wrong volume is mounted and notifying the operator to take corrective action) is handled directly through OS/2 and the device driver. Each FSD is responsible for generating a volume label and 32-bit volume serial number. These are stored in a reserved location in logical sector zero at format time. Because an FSD is the only system component to touch this information, an FSD is not required to store it in a particular format. OS/2 calls the FSD to perform operations that might involve it. The FSD is required to update the volume parameter block (VPB) whenever the volume label or serial number is changed.

When the FSD passes an I/O request to an FS helper routine, the FSD passes the 32-bit volume serial number and the user's volume label (through the VPB). When the I/O is performed, OS/2 compares the requested volume serial number with the current volume serial number it maintains for the device. This is an in-storage test (no I/O required) performed by checking the drive parameter block's (DPB) VPB of the volume mounted on the drive. If unequal, OS/2 signals the critical error handler to prompt the user to insert the volume having the serial number and label specified.

When OS/2 detects a media change in a drive, or the first time a volume is accessed, OS/2 determines which FSD is responsible for managing I/O to that volume. OS/2 allocates a VPB and polls the installed FSDs (by calling the FS_MOUNT entry point) until an FSD indicates that it does recognize the media.

Note: The FAT FSD is the last in the list of installed FSDs and acts as the default FSD when no other FSD recognition takes place.

Connectivity

There are two classes of file system drivers:

- FSDs that use a block device driver to do I/O to a local or remote device. These are called local file systems.
- FSDs that access a remote system without a block device driver. These are called remote file systems.

The connection between a drive letter and a remote file system is achieved through a command interface provided with the FSD (FS_ATTACH).

When a local volume is first accessed, OS/2 sequentially asks each installed FSD to accept the media, by calling each FSD's FS_MOUNT entry point. If no FSD accepts the media, it is then assigned to the default FAT file system. Any further attempt that is made to access an unrecognized media, other than by FORMAT, results in an 'Invalid media format' message.

When a volume has been recognized, the relationship between drive, FSD, volume serial number, and volume label is remembered. The volume serial number and label are stored in the volume parameter block (VPB). The VPB is maintained by OS/2 for open files (I/O based on file-handles), searches, and buffer references. The VPB represents the media.

Subsequent requests for a volume that has been removed require polling the installed FSDs for volume recognition by calling FS_MOUNT. The volume serial number and volume label of the VPB returned by the recognizing FSD and the existing VPB are compared. If the test fails, OS/2 signals the critical error handler to prompt the user for the correct volume.

The connection between media and VPB is remembered until all open files on the volume are closed and search and cache buffer references are removed. Only volume changes cause a redetermination of the media at the time of next access.

IPL Mechanism

A primary DOS disk partition (type 1, 4, or 6) may be used to boot the system. The code for FSDs may reside in any partition readable by a previously installed FSD. An IFS partition must be a type 7 partition.

The OS/2 boot volume includes the following: Bootrecord and basic file system. The root directory of this volume will contain a mini-file system in OS2BOOT, a kernel loader in OS2LDR, the OS/2 kernel in OS2KRNL, and the CONFIG.SYS file.

Device drivers and FSDs are loaded in the order they appear in CONFIG.SYS and are considered elements of the same ordered set. Therefore, both device drivers and FSDs may be loaded from installed file systems as long as they are started in the proper order. For example:

DEVICE = c:\diskdriv.sys REM Block device D: is now defined. (diskdriv.sys controls this.) IFS = c:\fsd\newfsl.fsd REM If we assume that D: contains a fixed newfsl type partition, REM then we're now ready to use D: to load the device driver and REM FSD for E:. DEVICE = d:\root\dev\special.dev REM Block device e: is now defined. IFS = d:\root\fsd\special.fsd REM E: can now be read. DEVICE = e:\music

OS/2 Partition Access

Access to the OS/2 partition on a bootable, logically partitioned media is through the full OS/2 function set. See OS/2 Version 2.0 Physical Device Driver Reference for a detailed description of the disk partitioning design.

Permissions

There are no secure file system clients identified for OS/2 Version 2.0 incorporating the IFS architecture.

File Naming Conventions

See OS/2 Version 2.0 Programming Guide for a detailed description of OS/2 Version 2.0 file naming conventions.

Meta Character Processing

See OS/2 Version 2.0 Programming Guide for a detailed description of OS/2 Version 2.0 meta character processing.

FSD Pseudo-character Device Support

A pseudo-character device (single file device) may be redirected to an FSD. The behavior of this file is very similar to the behavior of a normal OS/2 character device. It may be read from (DosRead) and written to (DosWrite). The difference is that the DosChgFilePtr and DosFileLocks functions can also be applied to the file. The user would perceive this file as a device name for a non-existing device. This file is seen as a character device because the current drive and directory have no effect on the name. That is what happens in OS/2 today for character devices.

The format of an OS/2 pseudo-character device name is that of an ASCIIZ string in the format of an

OS/2 file name in a subdirectory called \DEV\. The pseudo device name XXX is accessible at the API level (DosQFSAttach) through the path name '\DEV\XXX'.

Family API Issues

Since the IFS Mechanism is not present in any release of DOS, FAPI will not be extended to support the new interfaces.

FSD Utilities

FSD Utility Support

Each FSD is required to provide a single .DLL executable module that supports the OS/2 FORMAT, CHKDSK, SYS, and RECOVER utilities. The FS-supported executable will be invoked by these utilities when performing a FORMAT, CHKDSK, SYS, or RECOVER function for that file system. The command line that was passed to the utility will be passed unchanged to the FS-specific executable.

The procedures that support these utilities reside in a file called U<fsdname>.DLL, where <fsdname> is the name returned by DosQFSAttach. If the file system utility support .DLL file is to reside on a FAT partition, then <fsdname> should be up to 7 bytes long.

FSD Utility Guidelines

The FSD utility procedures are expected to follow these guidelines:

- No preparation is done by the base utilities before they invoke the FSD utility procedure. Therefore, base utilities do not lock drives, parse names, open drives, etc. This allows maximum flexibility for the FSD.
- The FSD utility procedures are expected to follow the standard conventions for the operations that they are performing, for example, /F for CHKDSK implies “fix”.
- The FSD procedures may use stdin, stdout, and stderr, but should be aware that they may have been redirected to a file or device.
- It is the responsibility of the FSD procedures to worry about volumes being changed while the operation is in progress. The normal action would be to stop the operation when such a situation is detected.
- When the FSD procedures are called, they will be passed argc, argv, and envp, that they can use to determine the operations.
- FSD procedures are responsible for displaying relevant prompts and messages.
- FSD utility procedures must follow the standard convention of entering the target drive as specified for each utility.

FSD Utility Interfaces

All FSD utility procedures are called with the same arguments:

```
int far pascal CHKDSK(int argc, char far * far *argv,
char far * far *envp);

int far pascal FORMAT(int argc, char far * far *argv,
char far * far *envp);

int far pascal RECOVER(int argc, char far * far *argv,
char far * far *envp);

int far pascal SYS(int argc, char far * far *argv,
char far * far *envp);
```

where argc, argv, and envp have the same semantics as the corresponding variables in C.