Remote IPL/Bootable IFS

This chapter describes the OS/2 Version 2.0 boot architecture and extensions to the installable file system mechanism (IFSM) to enable booting from an FSD-managed volume, referred to as Bootable IFS (BIFS). If the volume is on a remote system, it is referred to as Remote IPL (RIPL).

The mini-FSD is similar to the FSD defined in this document. However, it has additional requirements for to allow reading of the boot drive before the base device drivers are loaded. These requirements are fully defined in the two interface sections of this chapter.

To satisfy its I/O requests, the mini-FSD may call the disk device device driver imbedded in OS2KRNL (BIFS case) or it may provide its own driver (RIPL case).

Along with the mini-FSD, the IFS SYS Utility is required to initialize an FSD-managed volume with whatever is required to satisfy the requirements of the mini-FSD and this document.

The IFS mechanism includes some additional calls which the mini-FSD may need while it is linked into the IFS chain.

Operational Description

FAT Boot Procedure

The following figure represents the major stages of the OS/2 Version 2.0 FAT boot procedure.

1. POST
2. BOOT SECTOR
3. OS2BOOT (OS2LDR loader)
4. OS2LDR
5. stage1 OS2KRNL
6. stage2
7. stage3

Figure 4-1. OS/2 Version 2.0 FAT boot procedure

- Powering-on the machine or pressing CTRL-ALT-DEL causes control to get transferred to the power-on-self-test (POST) code. This code initializes the interrupt vectors to get to the BIOS routines. It then scans the I/O adapters looking for and linking in any code which exists on them. It then executes an interrupt 19h (INT 19) which causes control to be transferred to the disk or diskette boot code.
- The INT 19h code reads the boot sector from disk or diskette into memory at 7C00H. Along with code, the boot sector contains a structure called the BIOS Parameter Block(BPB). The BPB contains information which describes how the disk is formatted. The boot code uses this information to load in the root directory and the FAT micro-IFS, which is kept inside the OS2BOOT file. After the micro-IFS is loaded the boot sector transfer control it via a far jump.
- OS2BOOT receives pointers to the RAM copies of the root directory and the BPB. Using the BPB information, OS2BOOT loads in the FAT table from the disk. Then using the root directory and the FAT table, the OS2LDR file is loaded into memory from disk. The inclusion of this micro-IFS in the FAT boot process has removed the requirement that the OS2LDR file be logically

contiguous on the FAT drive.
- OS2LDR contains the OS/2 loader. It relocates itself to the top of low memory, then scans the root directory for OS2KRNL and reads it into memory. After the required fixups are applied, control is transferred to OS2KRNL, along with a pointer to the BPB and the drive number.
- OS2KRNL contains the OS/2 kernel and initialization code. It switches to protected mode, relocates parts of itself to high memory, then scans the root directory for and reads in the base device drivers (stage 1). Once again, the BIOS interrupt 13h is used to read the disk, but mode switching must be done.
- OS2KRNL then switches to protection level 3 and loads some of the required dynamic link libraries (stage 2) followed by the device drivers and FSDs specified in CONFIG.SYS (stage 3). This is done with standard DOS calls and, therefore, goes through the regular file system and device drivers.

BIFS Boot Procedure

The following figure represents the major stages of the OS/2 Version 2.0 BIFS boot procedure.

1. POST
2. BlackBox (MicroFSD)
3. OS2LDR
4. stage1 OS2KRNL
5. stage2
6. stage3

Figure 4-2. OS/2 Version 2.0 BIFS boot procedure

- The major difference between this boot procedure and the FAT boot procedure is that there is no assumption of booting off of disk. OS/2 Version 2.0 does not define what should happen between when the POST code is run and when the OS2LDR program gains control.
- When OS2LDR receives control, it must be passed information about the current state of memory and pointers to the Open, Read, Close, and Terminate entry points of the micro-FSD. Included in the memory map information is the positions of the micro-FSD, mini-FSD, RIPL data, and the OS2LDR file itself.
- **Note**: This interface is defined in a next section of this chapter.
- As with the FAT boot procedure, the OS/2 loader relocates itself to the top of low memory, and with the help of the micro-FSD, scans the root directory for the OS2KRNL file. After reading OS2KRNL into memory and applying the required fixups, control is transferred to the kernel.
- When OS2KRNL receives control, it goes through the same initialization as before (stage 1) with a couple of exceptions. The module loader is called to load the mini-FSD from its memory image stored by OS2LDR in high memory to its final location at the top of low memory. Also, the mini-FSD is called to read the base device drivers (one at a time) through the stage 1 interfaces.
- Before any of the dynalinks are loaded, the mini-FSD will be linked into the IFS chain (it will be the only link in the chain) and asked to initialize through FS_INIT. The FS_INIT call marks the transition from stage 1 to stage 2.
- The dynalinks are then loaded using the stage 2 interfaces, followed by the device drivers and FSDs.
- The mini-FSD is required to support only a small number of the FSD system interfaces (the FS_xxxx calls). Therefore, the first FSD loaded must be the replacement for the mini-FSD.
- After the replacement FSD has been loaded, it is called at FS_INIT to initialize itself and take whatever action it needs to effect a smooth transition from the mini-FSD to the FSD. It then

replaces the mini-FSD in the IFS chain, as well as in any kernel data structures which keep a handle to the FSD (for example, the SFT, VPB). This replacement marks the transition from stage 2 to stage 3.
- From this point on, the system continues normally.

Interfaces

BlackBox/OS2LDR interface

When initially transferring control to OS2LDR from a “black box”, the following interface is defined:

DH boot mode flags:

bit 0 (**NOVOLIO**) on	indicates that the mini-FSD does not use *MFSH_DOVOLIO*.
bit 1 (**RIPL**) on	indicates that boot volume is not local (RIPL boot)
bit 2 (**MINIFSD**) on	indicates that a mini-FSD is present.
bit 3 (**RESERVED**)	
bit 4 (**MICROFSD**) on	indicates that a micro-FSD is present.
bits 5-7	are reserved and MUST be zero.

DL drive number for the boot disk. This parameter is ignored if either the **NOVOLIO** or **MINIFSD** bits are zero.

DS:SI is a pointer to the BOOT Media's BPB. This parameter is ignored if either the **NOVOLIO** or **MINIFSD** bits are zero.

ES:DI is a pointer to a filetable structure. The filetable structure has the following format:

```
struct FileTable {
    unsigned short ft_cfiles; /* # of entries in this table             */
    unsigned short ft_ldrseg; /* paragraph # where OS2LDR is loaded     */
    unsigned long  ft_ldrlen; /* length of OS2LDR in bytes              */
    unsigned short ft_museg;  /* paragraph # where microFSD is loaded   */
    unsigned long  ft_mulen;  /* length of microFSD in bytes            */
    unsigned short ft_mfsseg; /* paragraph # where miniFSD is loaded    */
    unsigned long  ft_mfslen; /* length of miniFSD in bytes             */
    unsigned short ft_ripseg; /* paragraph # where RIPL data is loaded  */
    unsigned long  ft_riplen; /* length of RIPL data in bytes           */

    /* The next four elements are pointers to microFSD entry points     */

    unsigned short (far *ft_muOpen)
                     (char far *pName, unsigned long far *pulFileSize);
    unsigned long (far *ft_muRead)
                     (long loffseek, char far *pBuf, unsigned long cbBuf);
    unsigned long (far *ft_muClose)(void);
    unsigned long (far *ft_muTerminate)(void);
}
```

The microFSD entry points interface is defined as follows:

mu_Open	- is passed a far pointer to name of file to be opened and a far pointer to a ULONG to return the file's size. The returned value (in AX) indicates success(0) or failure(non-0).
mu_Read	- is passed a seek offset, a far pointer to a data buffer, and the size of the data buffer. The returned value(in DX:AX) indicates the number of bytes actually read.
mu_Close	- has no parameters and expects no return value. It is a signal to the micro-FSD that the loader is done reading the current file.
mu_Terminate	- has no parameters and expects no return value. It is a signal to the micro-FSD that the loader has finished reading the boot drive.

The loader will call the micro-FSD in a Open-Read-Read-….-Read-Close sequence with each file read in from the boot drive.

miniFSD/OS2KRNL interface

- When called from OS2KRNL after being linked into the IFS chain, the interface will be as described in previous chapters of this document. Note that the *FS_INIT* interface for a mini-FSD has an additional parameter, as compared to the *FS_INIT* interface for an FSD.
- When called from OS2KRNL, before being linked into the IFS chain, the interface will be through the *MFS_xxxx* and *MFSH_xxxx* entry points. These interfaces are described in this chapter. Many of these interfaces parallel the interfaces defined for FSDs, while others are unique to the mini-FSD.
- The mini-FSD is built as a dynamic link library. Supplied functions are exported by making the function names public. Helper functions are imported by declaring the helper names external:far. It is required only to support reading files and will be called only in protect mode. The mini-FSD may NOT make dynamic link system calls at initialization time.
- Due to the special state of the system as it boots, the programming model for the mini-FSD during the state 1 time frame is somewhat different than the model for stage 2. This difference necessitates 2 different interfaces between OS/2 and the mini-FSD.
- During stage 1, all calls to the mini-FSD are to the *MFS_xxxx* functions. Only the *MFSH_xxxx* helper functions are available. These are the interfaces which are addressed in this document. Many of these interfaces parallel the interfaces defined for FSDs while others are unique to the mini-FSD.
- During stage 2, the mini-FSD is treated as a normal FSD. Calls are made to the *FS_xxxx* functions and all *FSH_xxxx* helper functions are available.
- During stage 3, the mini-FSD is given a chance to release resources (through a call to *MFS_TERM*) before being terminated.
- Transition from stage 1 to stage 2 is marked by calling the *FS_INIT* function in the mini-FSD. Transition from stage 2 to stage 3 is marked by calling *FS_INIT* in the FSD.

Figure 4-3 on page 4-6 shows the functions called during a typical boot sequence:

1. stage1
    1. MFS_INIT
    2. MFS_OPEN
    3. MFS_READ
    4. MFS_CHGFILEPTR
    5. MFS_CLOSE

2. stage2
    1. FS_INIT
    2. FS_MOUNT/ATTACH
    3. FS_OPEN
    4. FS_READ
    5. FS_CHGFILEPTR
3. stage3
    1. MFS_TERM

Figure 4-3. Typical boot sequence

- No files are open at the transition from stage 1 to stage 2. Also, only a single file at a time is open during stage 1. Files and volumes are open during the transition from stage 2 to stage 3 (the mini-FSD to the FSD). The FSD must do whatever is necessary for it to inherit them. The FSD will not receive mounts/attaches or opens for volumes and files which were mounted/attached and opened by the mini-FSD. Also, multiple files may be open simultaneously during stages 2 and 3.
- A special set of helper functions are available to the mini-FSD to support an imbedded device driver. This might be required for situations such as remote IPL where the boot volume is not readable through DOVOLIO. These special helper functions (referred to as imbedded device driver helpers) are available during all stages of the mini-FSD's life. Note that the list of error return codes for the helper functions is not exhaustive, but rather represents the most common errors returned.
- Because the mini-FSD is a new component added to the boot sequence, a new interface to OS2LDR is required.
- The name and attributes of the mini-FSD must match EXACTLY the name and attributes of the replacement FSD.
- Due to the instability of the system during initialization, any non-zero return code indicates an error has been encountered. The actual return code may not bake any sense in the context of the function called (for example, having ERROR_ACCESS_DENIED returned from a call to *MFSH_LOCK* when in fact an invalid selector was passed to the helper). It is also possible for the system to hang or reboot itself as a result of invalid parameters being passed to a helper function.

Stage 1 interfaces

The following functions must be made available by the mini-FSD. These functions will be called only during stage 1.

- MFS_CHGFILEPTR
- MFS_CLOSE
- MFS_INIT
- MFS_OPEN
- MFS_READ
- MFS_TERM

The following helper functions are available to the mini-FSD. These functions may be called only during stage 1.

- MFSH_DOVOLIO
- MFSH_INTERR

- MFSH_SEGALLOC
- MFSH_SEGFREE
- MFSH_SEGREALLOC

Stage 2 interfaces

- The intent of stage 2 is to use the mini-FSD as an FSD. Therefore, all the guidelines and interfaces specified in this document apply with the following exceptions.
- The following functions must be fully supported by the mini-FSD:

- FS_ATTACH (remote mini-FSD only)
- FS_ATTRIBUTE
- FS_CHGFILEPTR
- FS_CLOSE
- FS_COMMIT
- FS_INIT
- FS_IOCTL
- FS_MOUNT (local mini-FSD only)
- FS_NAME
- FS_OPENCREATE (existing file only)
- FS_PROCESSNAME
- FS_READ

- **Note** that since the mini-FSD is only required to support reading, *FS_OPENCREATE* need only support opening an existing file (not the create or replace options).
- None of the other functions required for FSDs are required for the mini-FSD but must be defined and should return the ERROR_UNSUPPORTED_FUNCTION return code.
- The full complement of helper functions specified in this document is available to the mini-FSD. However, the mini-FSD may NOT use any other dynamic link calls.

Stage 3 Interfaces

- The intent of stage 3 is to throw away the mini-FSD and use only the FSD.
- The following functions must be supported by the mini-FSD:
    - MFS_TERM

Imbedded Device Driver Helpers

The following helper functions are available to the mini-FSD and may be called during stage 1, 2, or 3. These helpers are counterparts for some of the device help functions and are intended for use by a device driver imbedded within the mini-FSD.

- MFSH_CALLRM
- MFSH_LOCK
- MFSH_PHYSTOVIRT
- MFSH_UNLOCK
- MFSH_UNPHYSTOVIRT
- MFSH_VIRTTOPHYS

Special Considerations

The size of the mini-FSD file image plus the RIPL data area may not exceed 62K. In addition, the memory requirements of the mini-FSD may not exceed 64K.

The mini-FSD is only required to support reading of a file. Therefore, any call to DosWrite (or other non-supported functions) which becomes redirected to the mini-FSD may be rejected. For this reason, it is required that the IFS= command which loads the FSD which will replace the mini-FSD be the first IFS= command in CONFIG.SYS. Also, only DEVICE= commands which load device drivers required by that FSD should appear before the first IFS= command.

If the mini-FSD needs to switch to real mode, it must use the MFSH_CALLRM function. This is required to keep OS/2 informed of the mode switching.

Each FSD which is bootable is required to provide their “black box” to load OS2LDR and the mini-FSD into memory before OS2LDR is given control.

Additionally, these FSDs are required to provide a single executable module in order to support the OS/2 SYS utility. The executable provided will be invoked by this utility when performing a SYS for that file system. The command line that was passed to the utility will be passed unchanged to the executable.

The supplied executable must do whatever is required to make the partition bootable. At the very least, it must install a boot sector. It also needs to install the “black box”, mini-FSD, OS2LDR and OS.