18

MS-DOS File Interface

An increasing number of computer systems provide a high-level environment that can switch from one operating system to another on the same machine. Moreover, they can share files between operating systems. For example, there are systems available commercially that support both MS-DOS and UNIX in this way.

Since PC-Xinu is run as an MS-DOS program, it seems reasonable that PC-Xinu should be able to access MS-DOS files. This chapter shows how to include such access in the framework of PC-Xinu's device structure. Because MS-DOS provides its own file system, managing a directory structure is not necessary; the only issues are buffering data and translating PC-Xinu I/O calls into appropriate MS-DOS access.

18.1 File Operations Available Through MS-DOS

MS-DOS supports several operations through the DOS function call mechanism. A DOS function call is made through an assembly language INT call of type *DOS*=21H. Following BIOS function call conventions, the AH register is loaded with the function number prior to making the *DOS* call. *DOS* function calls we will use are given in Figure 18.1.

INT DOS (DOS=21H)

Subfunction AH	Name	Description	Registers
3CH	DOS_CREAT	Create a DOS file	CX=[0] DX=pathname Returns: AX=DOS file handle
3DH	DOS_OPEN	Open a DOS file	AL=mode DX=pathname Returns: AX=DOS file handle
3EH	DOS_CLOSE	Close a DOS file	BX=DOS file handle
3FH	DOS_READ	Read from a DOS file	BX=DOS file handle CX=bytes to read DX=buffer address
40H	DOS_WRITE	Write to a DOS file	BX=DOS file handle CX=bytes to write DX=buffer address
42H	DOS_LSEEK	Move DOS file pointer	AL=method BX=DOS file handle CX:DX=(long) offset

Figure 18.1 DOS file I/O subfunctions

The value in brackets for DOS_CREAT is the one PC-Xinu uses. The other values depend on the request. In particular, the meanings of the *mode* parameter in subfunction *DOS_OPEN* are given in Figure 19.2. These modes correspond to the PC-Xinu open modes *FLREAD*, *FLWRITE* and *FLRW*. Note that files opened with *DOS_CREAT* default to read-write mode. *DOS* calls return with the carry bit of the FLAGS word set (1) upon error and reset (0) if no error has occurred.

Value	Name	Description
0	MSREAD	Read-only
1	MSWRITE	Write-only
2	MSRDWR	Read-write

Figure 18.2 DOS open modes

Access to *DOS* calls is made through a C library routine *intdos*. There are two parameters passed to *intdos*; both are pointers to structures (unions) corresponding to the 8088 registers AX, BX, CX, DX and FLAGS. The first parameter points to the structure

containing register values to be passed to the *DOS* function, and the second parameter points to the structure containing the register values upon return from the *DOS* function.

Observe that *DOS* functions are not re-entrant, so they must be made with PC-Xinu interrupt handling disabled. The *doscall* routine, defined in *doscall.c*, uses a pointer to a single structure for both input and output registers and returns the carry bit for error checking. The code for *doscall* is in file *doscall.c*.

```
/* doscall.c - doscall, dos_creat, dos_open, dos_close, dos_read,       */
/*              dos_write,dos_lseek                                     */

#include <dos.h>
#include <conf.h>
#include <kernel.h>

#define AX(r)           ((r).x.ax)
#define BX(r)           ((r).x.bx)
#define CX(r)           ((r).x.cx)
#define DX(r)           ((r).x.dx)
#define CFLAG(r)        ((r).x.cflag)
#define AH(r)           ((r).h.ah)
#define AL(r)           ((r).h.al)
#define BH(r)           ((r).h.bh)
#define BL(r)           ((r).h.bl)
#define CH(r)           ((r).h.ch)
#define CL(r)           ((r).h.cl)
#define DH(r)           ((r).h.dh)
#define DL(r)           ((r).h.dl)

#define DOS_CREAT       0x3c
#define DOS_OPEN        0x3d
#define DOS_CLOSE       0x3e
#define DOS_READ        0x3f
#define DOS_WRITE       0x40
#define DOS_LSEEK       0x42

/* alias for converting between longs & 2 ints.  This is highly machine */
/* and compiler dependent.  8088 longs are stored lowword/highword      */

struct L2I {
        int     lowword;
        int     highword;
};

/*------------------------------------------------------------------------
```

```
 *  doscall  --  call a dos function
 *------------------------------------------------------------------------
 */
doscall(regs)
union REGS *regs;
{
        intdos(regs,regs);      /* make the DOS call                    */
        return(CFLAG(*regs));   /* nonzero return means error           */
}

/*------------------------------------------------------------------------
 *  dos_creat  --  create a dos file
 *------------------------------------------------------------------------
 */
dos_creat(pathname,attr)
char *pathname; /* filename */
int attr;       /* file attributes */
{
        union REGS      regs;

        (char *) DX(regs) = pathname;
        CX(regs) = attr;
        AH(regs) = DOS_CREAT;
        if ( doscall(&regs) )
                return(SYSERR);
        return(AX(regs));               /* return the DOS file handle   */
}

/*------------------------------------------------------------------------
 *  dos_open  --  open a dos file
 *------------------------------------------------------------------------
 */
dos_open (pathname,mode)
char *pathname;         /* DOS filename */
int mode;               /* DOS file mode bits */
{
        union REGS      regs;

        (char *) DX(regs) = pathname;
        AL(regs) = mode;
        AH(regs) = DOS_OPEN;
        if ( doscall(&regs) )
                return(SYSERR);
        return(AX(regs));               /* return the DOS file handle   */
```

```
}

/*------------------------------------------------------------------------
 *  dos_close  --  close a dos file
 *------------------------------------------------------------------------
 */
dos_close (fd)
int fd;         /* file handle to close */
{
        union REGS      regs;

        BX(regs) = fd;
        AH(regs) = DOS_CLOSE;
        if ( doscall(&regs) )
                return(SYSERR);
        return(OK);
}

/*------------------------------------------------------------------------
 *  dos_read  --  read from a dos file
 *------------------------------------------------------------------------
 */
dos_read (fd,buffer,count)
int fd;         /* file handle to read from */
char *buffer;   /* destination buffer */
int count;      /* no. of bytes to transfer */
{
        union REGS      regs;

        BX(regs) = fd;
        (char *) DX(regs) = buffer;
        CX(regs) = count;
        AH(regs) = DOS_READ;
        if ( doscall(&regs) )
                return(SYSERR);
        return(AX(regs));               /* no. of bytes actually read */
}

/*------------------------------------------------------------------------
 *  dos_write  --  write to a dos file
 *------------------------------------------------------------------------
 */
dos_write (fd,buffer,count)
int fd;         /* file handle to write to */
```

```
char *buffer;   /* source buffer */
int count;      /* no. of bytest to transfer */
{
        union REGS      regs;

        BX(regs) = fd;
        (char *) DX(regs) = buffer;
        CX(regs) = count;
        AH(regs) = DOS_WRITE;
        if ( doscall(&regs) )
                return(SYSERR);
        return(AX(regs));               /* no. of bytes actually written*/
}

/*------------------------------------------------------------------------
 *  dos_lseek  --  perform an LSEEK on a dos file
 *------------------------------------------------------------------------
 */
long
dos_lseek(fd,offset,origin)
int fd;         /* file handle */
long offset;    /* offset into file */
int origin;     /* origin of seek - 0=beginning, 1=current, 2=end */
{
        union   REGS    regs;
        register struct L2I     *lp;

        BX(regs) = fd;
        lp = (struct L2I *) &offset;
        CX(regs) = lp->highword;
        DX(regs) = lp->lowword;
        AL(regs) = origin;
        AH(regs) = DOS_LSEEK;
        if ( doscall(&regs) )
                return( (long)SYSERR );
        lp->highword = DX(regs);
        lp->lowword  = AX(regs);
        return(offset);
}
```

Doscall.c also contains C-language calls for creating, opening, closing, reading, writing, and seeking to *DOS* files; these routines correspond to the *DOS* function calls described in Figure 14.1 and are translated by *doscall* into direct *DOS* calls. Note the similarity between these routines and their PC-Xinu counterparts.

18.2 Using The Device Switch Table For MS-DOS Files

To maintain the device-independent structure of PC-Xinu, MS-DOS files must correspond to entries in the device switch table just like other PC-Xinu devices. The implementation we have chosen is similar to the local file system described in Chapter 17. A master MS-DOS device is used to allocate from several pseudo-devices, each corresponding to an open MS-DOS file.

To avoid the expense of making *DOS* calls for single-character I/O, we have chosen to buffer MS-DOS files, using a structure similar to the PC-Xinu local file system. Each pseudo-device has a control block containing the buffer and associated information. File *mffile.h* contains the declarations.

```
/* mffile.h */

/* MS-DOS file structures used for accessing MS-DOS files in Xinu       */

#define EOF             -2              /* value returned on end-of-file*/
#define FLREAD          001             /* mf_mode bit for "read"       */
#define FLWRITE         002             /* mf_mode bit for "write"      */
#define FLRW            003             /* mf_mode bits for read+write  */
#define FLNEW           010             /* mf_mode bit for "new file"   */
#define FLOLD           020             /* mf_mode bit for "old file"   */
#define FLBINARY        040             /* mf_mode bit for "binary file"*/

#define DBUFSIZ         512             /* size of data buffers         */
#define DBMASK          (~(DBUFSIZ-1))  /* mask for stripping low bits  */

struct  mfblk   {                       /* file "device" control block  */
        int     mf_id;                  /* file's "device id" in devtab */
        int     mf_dev;                 /* file is on this master device*/
        int     mf_pid;                 /* process id accessing the file*/
        int     mf_handle;              /* MS-DOS file handle           */
        int     mf_mode;                /* FLREAD, FLWRITE, etc.        */
        long    mf_size;                /* file size in bytes           */
        long    mf_pos;                 /* current file position (bytes)*/
        long    mf_bpos;                /* position of first buffer byte*/
        Bool    mf_dch;                 /* has mf_buff been changed?    */
        char    *mf_bptr;               /* ptr to next char in mf_buff  */
        char    mf_buff[DBUFSIZ];       /* current data block for file  */
};

#ifdef  Nmf
extern  struct  mfblk   mftab[];
#endif

extern  int     dos_creat();
extern  int     dos_open();
extern  int     dos_close();
extern  int     dos_read();
extern  int     dos_write();
extern  long    dos_lseek();
```

Most of these fields are identical to their counterparts in file *file.h*. Field *mf_handle* is the MS-DOS file handle for the open file. The *mf_size* field holds the current size of the file in bytes. (For PC-Xinu local files, the size of the file is contained in the file's directory entry.) Finally, field *mf_bpos* is the MS-DOS file offset of the first byte of the current buffer; this value will always be a multiple of *DBUFSIZ*.

18.3 Establishing A Pseudo-Device

Making a connection to an MS-DOS file involves opening the file, allocating a pseudo-device, and initializing the control block. It should be no surprise that these activities are similar to their local file system counterparts.

MS-DOS files are opened with the same mode strings permitted for local files, and the *dfckmd* routine described in Chapter 17 is used for parsing the mode string. Once the mode string has been parsed, procedure *mfdsrch* attempts to open the file.

```
/* mfdsrch.c - mfdsrch */

#include <conf.h>
#include <kernel.h>
#include <mffile.h>

#define MSREAD  0
#define MSWRITE 1
#define MSRDWR  2

/*------------------------------------------------------------------------
 *  mfdsrch  --  search dos for given file name
 *------------------------------------------------------------------------
 */
mfdsrch(filenam, mbits)
char    *filenam;
int     mbits;
{
        int     fd;             /* MS-DOS file handle */
        int     mode;           /* MS-DOS mode bits */

        if ( strlen(filenam) <= 0 )
                return(SYSERR);
        mode = (mbits&FLRW) - 1;
        if ( mode<0 || mode>MSRDWR )
                return(SYSERR);
        if ( (fd=dos_open(filenam,mode)) == SYSERR ) {
                if ( mbits&FLOLD )
                        return(SYSERR);
                else    /* create a new file with no special attributes */
                        return( dos_creat(filenam,0) );
        }
        if ( mbits&FLNEW ) {
                dos_close(fd);
                return(SYSERR);
        }
        return(fd);
}
```

Note that the read/write modes for MS-DOS files (*MSREAD*, *MSWRITE*, and *MSRDWR*) are one less than their counterparts for PC-Xinu local files. *Mfdsrch* checks the file access against the mode bits passed in the *mbits* parameter and creates the file if it does not already exist. The value returned by a successful call to *mfdsrch* is the MS-DOS file handle of the opened file.

Once the MS-DOS file has been opened or created, a connection to a pseudo-device can be made. Procedure *mfalloc* allocates a pseudo-device from among the entries in the array *mftab*. The code is identical to that of *dfalloc* in Chapter 17.

```
/* mfalloc.c - mfalloc */

#include <conf.h>
#include <kernel.h>
#include <mffile.h>

/*------------------------------------------------------------------------
 *  mfalloc  --  allocate a device table entry for a dos file; return id
 *------------------------------------------------------------------------
 */
#ifdef  Nmf
mfalloc()               /* assume ints disabled, provided by caller     */
{
        int     i;

        for (i=0 ; i<Nmf ; i++)
                if (mftab[i].mf_pid == 0) {
                        mftab[i].mf_pid = getpid();
                        return(i);
                }
        return(SYSERR);
}
#endif
```

The high-level *open* procedure, applied to the MS-DOS master file device, maps into the *msopen* routine. *Msopen* uses the *dfckmd*, *mfdsrch*, and *mfalloc* routines to open the MS-DOS file and connect it to a pseudo-device.

```
/* msopen.c - msopen */

#include <conf.h>
#include <kernel.h>
#include <mffile.h>

/*------------------------------------------------------------------------
 *  msopen  --  open/create a dos file
 *------------------------------------------------------------------------
 */
msopen(devptr,filenam,mode)
struct devsw *devptr;
char *filenam;
char *mode;
{
        struct mfblk    *mfptr;
        int     mbits;
        int     findex;
        int     ps;
        int     fd;             /* DOS file handle */
        long    size;           /* file size */
        long    dos_lseek();

        if ( (mbits=dfckmd(mode)) == SYSERR )
                return(SYSERR);
        disable(ps);
        if ( (fd=mfdsrch(filenam,mbits)) == SYSERR ) {
                restore(ps);
                return(SYSERR);
        }
        /* determine file size */
        if ( (size=dos_lseek(fd,0L,2)) == (long)SYSERR ) {
                dos_close(fd);
                restore(ps);
                return(SYSERR);
        }
        if ( (findex=mfalloc()) == SYSERR ) {
                dos_close(fd);
                restore(ps);
                return(SYSERR);
        }
        mfptr = &mftab[findex];
        mfptr->mf_dev = devptr->dvnum;
        mfptr->mf_handle = fd;
```

```
        mfptr->mf_mode = mbits & (FLRW | FLBINARY);
        mfptr->mf_size = size;
        mfptr->mf_pos = 0L;
        mfptr->mf_bpos = 0L;
        mfptr->mf_dch = FALSE;
        if ( size > 0L) {
                dos_lseek(fd,0L,0);             /* reset to beginning */
                dos_read(fd,mfptr->mf_buff,DBUFSIZ);
                mfptr->mf_bptr = mfptr->mf_buff;
        } else
                mfptr->mf_bptr = &mfptr->mf_buff[DBUFSIZ];
        restore(ps);
        return(mfptr->mf_id);
}
```

Msopen sizes the file and fills in the control block by setting the current position to zero, and reading the first block of the file into the buffer if the file is nonempty. *Msopen* returns the pseudo-device id to the caller, so it can be used in operations like *read* and *write*.

Unlike the PC-Xinu file system, the MS-DOS file system is organized into a directory hierarchy. Files in MS-DOS can be named relative to the current directory or to the ''root'' of the file system. MS-DOS uses the backslash character '\' to separate directory names in the path from the root directory to a file. Since C uses the backslash character as an ''escape'' character in constant strings, care must be taken to use two backslashes in strings used for MS-DOS filenames. For example, to open the MS-DOS file *msopen.c* in the directory *\pcxinu\src* one would use the following call:

```
open(getdev("dos"),"\\pcxinu\\src\\msopen.c","ro");
```

18.4 MS-DOS Pseudo-Device Driver Routines

The pseudo-device driver routines for MS-DOS files are almost identical to their local file counterparts. The two auxiliary routines, *mfsflush and mfsetup*, are enough different that they deserve some comment.

File *mfsflush.c* contains the code for routine *mfsflush*, which writes the current buffer to the MS-DOS file if the buffer has changed. Observe that the *dos_write* routine is synchronous (i.e., it returns only after completing the write operation), making it unnecessary to copy the buffer contents into a temporary buffer before performing the *dos_write* operation.

```
/* mfsflush.c - mfsflush */

#include <conf.h>
#include <kernel.h>
#include <mffile.h>

/*------------------------------------------------------------------------
 *  mfsflush  --  flush data block for an MS-DOS file
 *------------------------------------------------------------------------
 */
mfsflush(mfptr)
struct  mfblk   *mfptr;
{
        int     bufcnt;
        long    dos_lseek();

        if ( mfptr->mf_dch == FALSE )
                return;
        bufcnt = DBUFSIZ;
        if (mfptr->mf_pos < mfptr->mf_bpos+DBUFSIZ)
                bufcnt = mfptr->mf_pos - mfptr->mf_bpos;
        dos_lseek(mfptr->mf_handle,mfptr->mf_bpos,0 );
        dos_write(mfptr->mf_handle,mfptr->mf_buff,bufcnt);
        mfptr->mf_dch = FALSE;
}
```

Mfsetup is much simpler than its local file counterpart *lfsetup*, because it does not need to contend with i-block management. The code is shown below.

```
/* mfsetup.c - mfsetup */

#include <conf.h>
#include <kernel.h>
#include <mffile.h>

/*------------------------------------------------------------------------
 *  mfsetup  --  set up appropriate data block in memory
 *------------------------------------------------------------------------
 */
mfsetup(mfptr)
struct  mfblk   *mfptr;
{
        int     offset;
```

```
        long    dos_lseek();

        mfptr->mf_bpos = (mfptr->mf_pos)&DBMASK;
        offset = mfptr->mf_pos - mfptr->mf_bpos;
        if ( mfptr->mf_bpos < mfptr->mf_size ) {
                dos_lseek(mfptr->mf_handle,mfptr->mf_bpos,0);
                dos_read(mfptr->mf_handle,mfptr->mf_buff,DBUFSIZ);
        }
        mfptr->mf_bptr = mfptr->mf_buff + offset;
        mfptr->mf_dch = FALSE;
}
```

18.4.1 MS-DOS File Input/Output And Seek Operations

The high-level MS-DOS file pseudo-device operations *mfseek*, *mfgetc*, *mfread*, *mfputc*, *mfwrite*, and *mfcntl* are given below without further comment.

```
/* mfgetc.c - mfgetc */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <mffile.h>

/*------------------------------------------------------------------------
 *  mfgetc  --  get next character from (buffered) disk file
 *------------------------------------------------------------------------
 */
mfgetc(devptr)
struct devsw *devptr;
{
        struct  mfblk   *mfptr;
        char    nextch;
        int     ps;

        disable(ps);
        mfptr = (struct mfblk *)devptr->dvioblk;
        if (mfptr->mf_pid!=currpid || !(mfptr->mf_mode&FLREAD)) {
                restore(ps);
                return(SYSERR);
        }
        if ( mfptr->mf_pos >= mfptr->mf_size ) {
                restore(ps);
                return(EOF);
        }
        if ( mfptr->mf_bptr >= &mfptr->mf_buff[DBUFSIZ] ) {
                mfsflush(mfptr);
                mfsetup(mfptr);
        }
        nextch = *(mfptr->mf_bptr)++;
        mfptr->mf_pos++;
        restore(ps);
        return( ((int)nextch) & 0xff );
}


/* mfread.c - mfread */

#include <conf.h>
#include <kernel.h>
#include <mffile.h>
```

```
/*------------------------------------------------------------------------
 *  mfread  --  read from a previously opened disk file
 *------------------------------------------------------------------------
 */
mfread(devptr, buff, count)
struct  devsw   *devptr;
char    *buff;
int     count;
{
        int     done;
        int     ichar;

        if (count < 0)
                return(SYSERR);
        for (done=0 ; done < count ; done++)
                if ( (ichar=mfgetc(devptr)) == SYSERR)
                        return(SYSERR);
                else if (ichar == EOF ) {       /* EOF before finished */
                        if (done == 0)
                                return(EOF);
                        else
                                return(done);
                } else
                        *buff++ = (char) ichar;
        return(done);
}
```

```
/* mfputc.c - mfputc */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <mffile.h>

/*------------------------------------------------------------------------
 *  mfputc  --  put a character onto a (buffered) disk file
 *------------------------------------------------------------------------
 */
mfputc(devptr, ch)
struct  devsw   *devptr;
char    ch;
{
        struct  mfblk   *mfptr;
        int     ps;

        disable(ps);
        mfptr = (struct mfblk *) devptr->dvioblk;
        if (mfptr->mf_pid != currpid || !(mfptr->mf_mode&FLWRITE)) {
                restore(ps);
                return(SYSERR);
        }
        if ( mfptr->mf_bptr >= &mfptr->mf_buff[DBUFSIZ] ) {
                mfsflush(mfptr);
                mfsetup(mfptr);
        }
        mfptr->mf_pos++;
        if ( mfptr->mf_pos > mfptr->mf_size )
                mfptr->mf_size = mfptr->mf_pos;
        *(mfptr->mf_bptr)++ = ch;
        mfptr->mf_dch = TRUE;
        restore(ps);
        return(OK);
}


/* mfwrite.c - mfwrite */

#include <conf.h>
#include <kernel.h>

/*------------------------------------------------------------------------
 *  mfwrite  --  write 'count' bytes onto a local disk file
```

```
 *------------------------------------------------------------------------
 */
mfwrite(devptr, buff, count)
struct  devsw   *devptr;
char    *buff;
int     count;
{
        int     i;

        if (count < 0)
                return(SYSERR);
        for (i=count; i>0 ; i--)
                if (mfputc(devptr, *buff++) == SYSERR)
                        return(SYSERR);
        return(count);
}
```

```
/* mfseek.c - mfseek */

#include <conf.h>
#include <kernel.h>
#include <mffile.h>

/*------------------------------------------------------------------------
 *  mfseek  --  seek to a specified position of a file
 *------------------------------------------------------------------------
 */
mfseek(devptr, offset)
struct  devsw   *devptr;
long    offset;
{
        struct  mfblk   *mfptr;
        int     retcode;
        int     ps;

        disable(ps);
        mfptr = (struct mfblk *)devptr->dvioblk;
        if ( mfptr->mf_mode & FLWRITE )
                mfsflush(mfptr);
        else if ( offset > mfptr->mf_size ) {
                restore(ps);
                return(SYSERR);
        }
        mfptr->mf_pos = offset;
        mfsetup(mfptr);
        restore(ps);
        return(OK);
}
```

18.4.2 Closing The Pseudo-Device

When a process finishes using an MS-DOS file, it must call *close* to flush unwritten buffers and detach the pseudo-device. The *mfclose* routine is similar to *lfclose*, except that *mfclose* does not need to contend with local file system directory management. The details are in file *mfclose.c*.

```
/* mfclose.c - mfclose */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <mffile.h>

/*------------------------------------------------------------------------
 *  mfclose  --  close a file by flushing output and freeing device slot
 *------------------------------------------------------------------------
 */
mfclose(devptr)
struct  devsw   *devptr;
{
        struct  mfblk   *mfptr;
        int     diskdev;
        int     ps;

        disable(ps);
        mfptr = (struct mfblk *) devptr->dvioblk;
        if (mfptr->mf_pid != currpid) {
                restore(ps);
                return(SYSERR);
        }
        if ( mfptr->mf_mode & FLWRITE )
                mfsflush(mfptr);
        dos_close(mfptr->mf_handle);
        mfptr->mf_pid = 0;
        restore(ps);
        return(OK);
}
```

18.4.3 MS-DOS Pseudo-Device Initialization

Procedure *msopen* initializes most of the entries in the MS-DOS pseudo-device control block when it opens a file, but some initialization is necessary at system startup. File *mfinit.c* contains the code for the pseudo-device initialization routine *mfinit*.

```
/* mfinit.c - mfinit */

#include <conf.h>
#include <kernel.h>
#include <mffile.h>

/*------------------------------------------------------------------------
 *  mfinit  --  mark disk file 'device' available at system startup
 *------------------------------------------------------------------------
 */
mfinit(devptr)
struct  devsw   *devptr;
{
        struct  mfblk   *mfptr;

        devptr->dvioblk = (char *)(mfptr = &mftab[devptr->dvminor]);
        mfptr->mf_pid = 0;
        mfptr->mf_id = devptr->dvnum;
        return(OK);
}
```

18.5 MS-DOS File System Control Operations

Since MS-DOS file access is synchronous, a *SYNC* control operation is unnecessary. We have provided a procedure stub *mscntl* which can be used to implement additional MS-DOS file system operations. The exercises explore some possibilities.

```
/* mscntl.c - mscntl */

#include <conf.h>
#include <kernel.h>
#include <mffile.h>

/*------------------------------------------------------------------------
 *  mscntl  --  ioctl for the MS-DOS file system
 *------------------------------------------------------------------------
 */
int mscntl(devptr, funct, arg)
struct devsw *devptr;
int funct;
char *arg;
{
        return(OK);
}
```

18.6 Summary

Extending PC-Xinu by adding device access to MS-DOS files provides additional functionality and flexibility. This chapter showed how MS-DOS files can be processed like PC-Xinu local files with a master MS-DOS device and several pseudo-devices. Buffering files within PC-Xinu substantially reduces the number of *DOS* function calls.

FOR FURTHER STUDY

A summary of DOS function calls is in Rollins [1985], Appendix B.2. Register conventions and calling details are given in the DOS reference manual.

EXERCISES

18.1 Redesign routines *mfread* and *mfwrite* to perform high-speed copies like the tty *readcopy* and *writcopy* routines.

18.2 Implement other MS-DOS file operations such as *unlink* and *mkdir*. Should these routines be coded into the *mscntl* routine?

18.3 Is there any advantage to using double buffering with MS-DOS files? Explain.

18.4 High-level MS-DOS and local file driver routines are almost identical. Merge the *mfblk* and *flblk structures (in mffile.h* and *file.h*, respectively) and parameterize the device-dependent auxiliary routines (such as *lfsetup and mfsetup*), so that both MS-DOS files and local files can use the same set of pseudo-devices and high-level driver routines. What advantage does this approach have? Are there any disadvantages?

18.5 Modify the file *open* and *close* routines to allocate and deallocate data buffers dynamically, using *getmem* and *freemem*, respectively.

18.6 Show how the dynamic buffer allocation described in the previous exercise can lead to deadlock.