10. File I/O

Background

In this lab we will look at a few aspects of File I/O and discuss the differences between structured and unstructured files. An unstructured file typically refers to a file that is simply a collection of bytes with no meanings to particular byte offsets within the file. A text file is an example of an unstructured file (contains ASCII data). In a structured file, individual bytes or groups of bytes are meant to contain defined data items such as characters, integers, strings, floating point numbers etc, with the ordering of the fields always the same for files of that particular type. The PC world contains many examples of structured files; .BMP (bitmap files) and .WAV (sound files) are two examples of structured files.

Objectives:

Understand:

A. Basic file open/read operations

B. Difference between ASCII files (unstructured) and Binary Files (structured)

Pre-Lab

Make sure that you read this this entire lab before you attend class.

Lab

A. Reading an ASCII File

The program below (*fdmp.asm*) illustrates how to open a file for reading and read the contents of the file. The file is then dumped to the screen as ASCII data.

```
.model  small
.586
.stack 200h

.data
fname db 128 dup (0)
filehandle dw 0

BUFSIZE equ 256
buffer  db BUFSIZE dup (?)
msgFileErr      db "Error opening file.",0
msgFileReadErr  db "Error reading file.",0

.code
       extrn  Crlf:proc, Writestring:proc

main    proc
       mov     ax,@data
       mov    ds,ax

       mov    di,offset fname
       call   getfname   ; get filename from DOS command line

       mov    dx,offset fname
       mov    al,0          ; AL=0 means open for reading
       mov    ah, 3Dh              ; file open
       int    21h
       jc      FileErr                 ; Error? Display error message and quit
       mov    filehandle,ax       ; save the filehandle

       ;; read first BUFSIZE bytes in file
       mov    ah,3fh
       mov    cx,BUFSIZE          ;number of bytes to read
       mov    dx,offset buffer
       mov    bx,filehandle
       int    21h
       jc     FileReadErr

       mov    cx,ax        ; save number of bytes actually read in CX
       mov    bx,offset buffer
;; print all characters that were read
lp1:
       mov    dl,[bx]
       mov    ah,2
       int    21h
       inc    bx
       loop   lp1
```

```
        mov    bx,filehandle       ;get filehandle back
        mov    ah,3Eh              ; close file
        int    21h

        jmp    mainexit

FileReadErr:
        mov    dx, offset msgFileReadErr
        call   Writestring
        jmp    mainexit

FileErr:
        mov     dx,offset msgFileErr
        call   Writestring

mainexit:
        Mov    ax, 4c00h    ;exit to DOS
        Int    21h
main    endp

getfname  proc
        ;; copy filename from DOS command line to ds:di
        push   ds
        push   ds
        pop    es     ; set ES to DS
        mov    ah,62h
        int    21h          ; get address of program segment prefix
        mov    ds,bx
        mov    si,80h
        mov    cl,[si]             ; get character count of cmd line args
        cmp    cl,0
        je     getfname_exit   ; exit if no filename
        inc    si           ; point at first character
getlp1:
        mov    al,[si]
        cmp    al,20h       ;space?
        jne    getsk1
        inc    si
        dec    cl
        jmp    getlp1
getsk1:
        cld
        rep    movsb
        inc    di
        mov    al,0
        mov    [di],0      ; null terminate
getfname_exit:
        pop    ds
        ret
getfname endp

end     main
```

This program opens the file whose pathname is passed on the DOS command line, reads the first BUFSIZE bytes from the file, and sends these bytes to the screen as ASCII data.

How does this program work?

1. The procedure *getfname* copies the filename passed on the DOS command line to the buffer pointed to by DS:DI. We could have hard-coded the pathname in the program (as was done for the previous lab) but this is more convenient for testing with multiple files. An example command line would be : "c:\data\fdmp c:\data\test.txt" where 'test.txt' is the file to open. ***Notice that you must specify the complete pathname to the file*** (if you don't use the complete pathname, you will get a DOS subsystem error). The DOS command line arguments are stored in a structure called the Program Segment Prefix whose segment address is returned in BX via the DOS Int 21h, AH=62h call. The number of characters after the command line is at offset 80H, the characters follow at offset 81H-127H. The program calls *getfname* with DS:DI pointing at the buffer labeled as '*fname*'.
2. The DOS function 21h, AH=3Dh is used to open the file. The filename must be passed to this function via DS:DX. The file can be opened for reading (AL=0), writing (AL=1) or read/write (AL=2). The carry flag will be set SET upon return if an error occurred. If no error occurred, then the *filehandle* for the file is returned in register AX. This *filehandle* must be used for all subsequent operations to the file.
3. The DOS function 21h, AH=3Fh is used to read data from the file. Register CX specifies the number of bytes to be read, DS:DX points to the input buffer where the bytes are to be stored, and BX = filehandle. The carry flag is set on return if an error occurred. If no error occurred, then AX will contain the actual number of bytes read (AX will not be equal to CX if the file has fewer than CX remaining bytes in it). Each file read starts where the last file read stopped.
4. The program uses a loop to print the data that was read to the screen using the DOS function 21h, AH=1.
5. The file is closed via the DOS function Int 21h, AH=3Eh. Register BX contains the filehandle for the file to be closed.

Note that the program prints out an error message and exits if one of the file functions returns an error. This is important - the program should not continue execution if a file error occurs.

Lab Question 1: Assemble the program and execute it using any text file that you want (such as the *fdmp.asm* file itself). Modify the program so that all of the characters in the file are sent in the screen. Do not change the buffer size; you will need to change the loop so that the program loops until no more characters can be read from the file. When dumping the characters to the screen, pause the character listing if a space character is entered, resumed if the space bar is hit again. Include the assembled listing of this program in your lab report.

B. Structured files

The WWW page for this lab contains links to five structured files (fmt0.dat, fmt1.dat, fmt2.dat, fmt3.dat, fmt4.dat) that have the following file structure:

1. Byte 0: format type. Valid values are 0,1,2,3,4.
2. Bytes 1,2: an unsigned 16-bit integer that specifies the number of *data records* in the file. The integer is stored in *little-endian* order (least significant byte first).
3. Bytes 3 to end of file are the data records. For type 0, a data record is simply a single byte. For type 1, each record is a 16-bit (2 byte) integer, stored in little-endian order. For type 2, each record is a 16-bit (2 byte) integer, stored in big-endian order. For type 3, each record is a 32-bit (4 byte) integer, stored in little-endian order. For type 4, each record is a 32-bit (4 byte) integer, stored in big-endian order.
4. The number of bytes in a file can be calculated as 1 + num_records * bytes-per-record.

The program 'axe.exe' on the Micro I lab PCs is a binary file editor that you can use to examine the contents of these files (axe.exe is also available for download from www.zdnet.com as part of the archive *advhexed.zip*). You should examine the contents of the files to make sure that you understand the format structure.

Lab Question 2: Write a program that will dump the contents of any file that follows the above format to the screen. The output for your program should look something like (output shown for each possible file type):

```
Fname: fmt0.dat
Format type: 0, Record length: 22
Data:
67 ('g')
6f ('o')
6f ('o')
64 ('d')
62 ('b')
79 ('y')
65 ('e')
20 (' ')
63 ('c')
72 ('r')
75 ('u')
65 ('e')
6c ('l')
20 (' ')
77 ('w')
6f ('o')
72 ('r')
6c ('l')
64 ('d')
2e ('.')
2e ('.')
2e ('.')
```

```
Fname: fmt1.dat
Format type: 1, Record length: 7
Data  :
000c ('12')
0022 ('34')
0041 ('65')
0022 ('34')
044c ('1100')
0e10 ('3600')
fb60 ('64352')
```

```
Fname: fmt2.dat
Format type: 2, Record length: 7
Data is :
000c ('12')
0022 ('34')
0041 ('65')
0022 ('34')
044c ('1100')
0e10 ('3600')
fb60 ('64352')
```

```
Fname: fmt3.dat
Format type: 3, Record length: 6
Data  :
0000000c
fffffffe
00233fd5
000001c8
00010000
fffabacf
```

```
Fname: fmt4.dat
Format type: 4, Record length: 6
Data :
0000000c
fffffffe
00233fd5
000001c8
00010000
fffabacf
```

Print the 16-bit values as both hex values and as unsigned decimal integers using the Irvine *Writeint* function. For the 32-bit values, you only need to print them in hex. To make it easier, you can make your file buffer 8K bytes and assume that you will never be passed a file larger than this (i.e., read all of the bytes from the file at one time). If your program is passed a file whose format byte is not in the range 0 to 4, print an error message. Also do error checking on file open and file reading as in the first example program.

Two perl scripts are also linked to the WWW page for this lab. The script 'mkfile.pl' can be used to create new test files in these formats (to see how to use this program, type 'perl mkfile.pl' on a UNIX machine and read the usage information). The script 'dpfile.pl' can be used to dump the contents of a file that follows these formatting rules.

Included an assembled, commented listing of your program in the lab report.

Lab Report

A. Describing What You Learned

Include the answers to all "**Lab Questions**" in your report.

B. Applying What You Learned

Demonstrate the programs you wrote for Lab Questions 1 and 2 to the TA. The TA may test your programs with data files that are different from the examples on the WWW page.