Introduction

In this assignment you will add details to the design of a file system, and then implementit in detail. The file system uses a variant of the scheme used by UNIX in that it allows for a hierarchical file directory and uses pointers to individual blocks of data.

The disk used in this assignment has 100 blocks (sectors) numbered from 0 to 99, and each contains 512 bytes of data. We are not concerned about the placement of the blocks on the disk, and therefore all aaccess to the disk should be done through two procedures (which you Will write) DREAD and DWRITE (which will be described below). Note that you need not actually do disk reads and writes; the entire disk can be simulated using a 50K block of memory.

The file system will be tested using a front end program that processes commands given as input data, These commands will exercise the file create, open, close, delete, read, write,· and seek functions of the file system.

When the end of the input data is reached, the following information will be· displayed: the directory (illustrating its hierarchical structure), the length of each file (in bytes) and the number of free, directory, and user data blocks.

Directory structure

The file directory in this system will be hierarchically ordered. The root directory ,always begins in block O, which has the following structure (a'la PL/I):

```
DECLARE 1 BLOCKO,
            2  BACK FIXED  BIN(31),      /* ALWAYS ZERO  IN THIS BLOCK*/
            2  FR\.ID FIXED BIN(31),     /* BLOCK NUMBER: OF SECOND OIRECTORY  BLOCK;  OR '2'.tRO *J
            2  FREE FIXED  BIN(31),      /* BLOCK NUMBER Of FIRST UNUSED BLOCK*/
            2  FILLER  CHARACTER(4),     /'*  UNUSED*I
            2  DIR  (31),                /* DIRECTORY ENTRIES*/
               3 TYPE  CHARACTER(1),3    /* 'F'  = FREE,  'D' ,= DIRECTORY,  'U' = USER  JATA */
               NAHE CHARACTER (9),       /* FILE NAME, LEFT: JUSTIFIED, BLANK  FILLED */
               3  LINK  FIXED  BIN(31),  /* BLOCK NUMB(R OF FIRST BLOCK OF FILE*/
               3  SIZE  FIXED  BIN(l5);  /* NUMBER OF BYTES USED IN THE LAST BLOCK OF THE FILE */
```

The remaining blocks in the root file directory, if any, will be linked to block zero using the FRWD entry. These blocks, and blocks in the subordinate directories will have, exactly the same structure as block zero, except that the FREE entry will be unused. This entry .in block zero is used to point to the first unused block on the disk. The unused blocks should form. a linked list (which must be initially created when the file system starts execution).

The TYPE entry of each directory entry determines what type of file is referenced. ‘F’ is used to indicate that an entry in the file directory is unused. 'D' indicates that the entry points to another (subordinate) directory, while ‘U’, indicates that the entry points to a user data file.

File names are given as strings of up to 9 alphabetic characters separated by virgules·

(slashes). For example, the following are all legal file names:

```
SAMPLE
SUB/SAMPLE
SUB1/SUB2/SAMPLE
```

The first file name would be located in the root directory as a user data file. The second file name would be found first in the root directory as a subordinate directory ca11ed SUB, then in the SUB directory as a user data file called SAMPLE. The third file name would involve two subordinate file directories, the first called SUBl located in the root directory, the second ca11ed SUB2 located in the SUB1 directory, and the last called SAMPLE located in the SUB2 directory as a user data file.

Each block in a data file, except the last, is considered full. The number of data bytes present in the last block of a data file is indicated by the SIZE field of the directory entry. Note that directory blocks always have exactly 31 entries, with unused entries marked by a type of field of 'F'.

Data File Fonnat

Data files also form linked lists. The structure of a data file block is as follows:

```
DECLARE 1 DATA_BLOCK,
          2 BACK FIXED BIN(31),     /* BLOCK NUMBER OF PREVIOUS BLOCK */
          2 FRWD FIXED BIN(31),     /* BLOCK NUMBER OF SUCCESSOR BLOCK*/
          2 USER_DATA CHAR(504);    /* USER DATA BYTES */
```

The first and last data blocks of a file will have the BACK and FRWD fields respectively, set to zero. These will not point to block zero (since that is always the first directory block) but will rather indicate the end of the linked data block list. Note further: that the maximum number of bytes in a data block is 504.

collll1ands

Each command that may be processed by the file system is matched by an input command line. The syntax of these command lines, and the processing to be performed, is as follows:

CREATE type name

"type" is either U or D, and indicates whether a User data file or a Directory file is to be created. "name" is a full file name in the form described above. Neither of these items is enclosed in quotes. The CREATE command should cause a new directory entry to be created for the type of file specified. If CREATE is given for an existing file, no error should occur, but the file should be deleted and then Recreated. CREATE also leaves the file in the same state as an OPEN in the output mode.

OPEN mode name

"mode" is either I, O, or U indicating the file named "name" is to be opened in the Input, Output, or Update mode. Input mode means that only READ and SEEK commands are permitted while Output mode means only WRITE commands are permitted. Update mode allows READ, WRITE, and SEEK commands. Associated with each open file is a pointer to the next byte to be read or written. Opening a file for input or update places the pointer at the first byte of the file, while opening a file for output places the pointer at the byte immediately after the last byte of the file.

CLOSE

This command causes the last Opened or Created file to be closed. No filename is given.

DELETE name

This command causes the named file to be deleted.

READ n

This command may only be used between an OPEN (in input or update mode) and 'the corresponding CLOSE. If possible, "n" bytes of data should be read and displyed. If fewer than "n" bytes remain before the end of file, then those bytes should be read, and displayed with a message indicating that the end of file was reached.

WRITE n 'data'

This command causes the first "n" data bytes from 'data' (actually enclosed, in single quotes in the command line) to be written to the file. If fewer than "n" bytes are given, then append sufficient blanks to 'data' to make "n" bytes. If it is impossible to write "n" bytes (because the disk is full) then an appropriate message should be issued, but the command should be otherwise treated as if the largest possible valve of "n" was specified. (That is, the remaining available disk space should be filled)

SEEK base offset

"base" is either -1, 0, or +1 indicating the beginning of the file, the current position in the file, or the end of file. "offset" is a signed integer indicating the number of bytes from the "base" that the file pointer should be moved. For example; "SEEK -1 0" is equivalent to a rewind, "SEEK +1 0" is equivalent to a position to end of file, and "SEEK 0 -5" positions the file pointer backward by five bytes.

You are guaranteed that all input data will have legal format. Further, the following syntax graph describes all possible sequences of input data:

```
__+-+__-->DELETE ----------------------------------------------------+


  !+!-----> CREATE-----+------>WRITE-----+--------> CLOSE--.---+!--+-->
  ! !                                                           !
  !                    +-<---<----<----<--+
  ! !                  !                  !                     !   !
  !+!------>OPEN----+--+-----> READ------+---+----> CLOSE------+!   !!
  ! !                                                           !   !
  !                    +----->WRITE ----- +
  !                 !  !!                 !     !                   !
  !                 ! +----->SEEK -------- +    !                   !
  !                 !+--<---<----<----<---<!-+  !                   !
  !                 !                           !                   !
  +------<-----<-----<-----<-----<-----<-----<-----<-----<----<-----+!
  !                                                                 !
```