The file system in this project is implemented on a 2MB section of storage. Storage is divided into blocks of 512 bytes. The total number of addressable blocks is 212, which requires 12 bit addresses, so each FAT entry has a size of two bytes. Each block of FAT addresses 28 blocks, so 24 blocks of FAT are needed. The first block of the file system contains the boot sector. The next 16 are FAT. By default, the next two are the root directory, but the length of the root directory is variable in the boot sector.

Two types are defined. The first is the data structure of an individual FAT listing. It contains a 100 character string for the filename, bits for lock and subdirectory indicator, 2 bytes for address of starting block, int length of file. The second type is an Open Directory Listing object, used to implement Working Directory and as a FILE struct for applications to use in file operations. It contains pointers to data block and FAT, position pointer for current place and current block, buffer mapped to current block, pointer to parent starting block, and counters for bytes written. It also contains pointer to 512 byte buffer that it keeps mapped to current block at all times. It will be used to read, write, lock, unlock, fseek…

Boot procedure: Read boot sector to determine length of FAT and root. Their sum, increased by one to account for the boot sector is saved as offset D. All future addressing will refer to block numbering as beginning from 0 at start of data area, so block number should be added to offset D to get true physical location. Create an Open Directory Listing object and set it to open the root directory.

## Functions

### File System Library Methods:

**AllocateBlock**(predecessor block): // For new file, pass null argument

Linear Search FAT for free block.

If no block found, then Error: Memory Full.

If predecessor in range [1, 212], point predecessor FAT listing to new block.

Return New Block Number

### Directory Listing Object Methods:

Methods that have no object in arguments or return work in Working Directory object.

**AdvancePosition**:

If current place = 512:

current block := AllocateBlock(current block)

Unmap memory and remap to current block.

current place := 1

Else, increment current place

**Backtrack**:

If current place = 0:

current block := linear search linked list from start block until finding FAT entry listing to current block

Unmap memory and map to current block.

current place = 512

Else:

Decrement current place

**ChangeDirectory**(subdir name):

Unmap buffer from current blocks.

If name indicates root, set all pointers to indicate start of root and return.

Scan for name in current file listing. If none found, Error: directory not found.

When listing found, find starting block. Change all pointers to that block and that FAT entry.

Map buffer to start of first block of file.

Return.

**OpenFile**(path, …):

Tokenize path into pathTokens.

If length of tokens > 1:

New Directory Listing object at root.

While pathTokens contains more than one token:

ChangeDirectory to first token, and remove it from pathTokens list.

Scan for final token in current directory listings. If directory listing not found, Error, File not Found

current block := directory listing start block

current place = 0

map memory to current block

Return Directory Listing object

**CreateFile**(path, …):

Tokenize path into pathTokens.

If length of tokens > 1:

New Directory Listing object at root.

While pathTokens contains more than one token:

ChangeDirectory to first token, and remove it from pathTokens list.

Scan for final token in current directory listings. If found:

If “-f” in arguments:

Set size to 0. Write EOF to first character in file data.

Else:

Error “File already exists”

Return.

Scan FAT for zero listings. When one is found, store block number as newFileAddr, and set FAT entry to 0xFFFF. Write EOF to first character in file data.

In current directory, write new directory listing, storing final token as name, newFileAddr as starting block, length as 0…

Return pointer to newFileAddr.

**CreateDirectory**(full path,…):

newFileAddr := CreateFile(full path,…)

In file listing at newFileAddr, set directory bit to true.

// Create new file, but store parent and set directory bit true.

**CloseFile**:

Unmap open buffer

Unlock file object

**WriteFile**(input string):

Allocate 512 byte Displaced Characters buffer

CharactersWritten := 0

For all characters c in input string:

# If EOF hasn’t been found, characters are still being displaced, and must be held.

If character at current blockCurrent place ≠ EOF:

then append current blockCurrent place to Displaced Characters

current blockCurrent place := c

Increment CharactersWritten

AdvancePosition

Else:

current blockCurrent place = c

AdvancePosition

current blockCurrent place + 1 = EOF

AdvancePosition

Increment CharactersWritten

# Loop Ends. All characters in input string written to data area. Deal with Displaced Characters

If Displace Characters, then WriteFile(Displaced Characters)

# Disregard return from recursive call. Characters already counted.

In file listing, increment size by CharactersWritten

Return CharactersWritten

**ReadFile**(N: Number of Characters):

ReadSize := max( N, file size – current position)

Allocate ReadBuffer of ReadSize bytes.

For index := 1 to ReadSize:

If character at current place = EOF, then return ReadBuffer

Append character at current place to ReadBuffer

AdvancePosition

Return ReadBuffer

**DeleteFile**(path):

Tokenize path into pathTokens.

If length of tokens > 1:

New Directory Listing object at root.

While pathTokens contains more than one token:

ChangeDirectory to first token, and remove it from pathTokens list.

Scan for final token in current directory listings. If directory listing not found, Error, File not Found

# List all blocks available in FAT

cleantblock := starting block

while value in cleaningblock ≠ FFFF:

nextblock = value stored in cleanblock

Set value at address in cleanblock to 0

Cleanblock := Nextblock

Zero out directory listing object

**SeekPosition**(N):

BlockinFile := N / 512

BlockPlace := N % 512

Linear Search Linked List in FAT for Nth block of file. Set current block to Nth block. If last block reached before N exhausted, return false.

while BlockPlace:

AdvancePosition

Decrement BlockPlace

If character at current blockcurrent position = EOF, then break

Return N – BlockPlace