**MP4 Report**

組別:os21team34

成員:江咏宸108030001、蔡懿晨107070035

| Contribution | |
| --- | --- |
| Tracecode與相關report | 蔡懿晨 |
| Implement與相關report | 江咏宸 |

Part I

(1) Explain how the NachOS FS manage and find free block space? Where is this information stored on the raw disk (which sector)?

A:

使用OpenFile \*freeMapFile 與Persistent Bitmap去管理和找free block space。由於persistent bitmap 最後是以file的形式呈現，所以剛開始要由PersistentBitmap FetchFrom()拿到disk儲存的 map，然後呼叫FindAndSet()找到free block space並設置已使用。同時也有Test()和Clear()，分別是支援測試該bit 是否被使用以及把該bit設置為沒使用。最後會呼叫WriteBack()寫回去disk上更新。

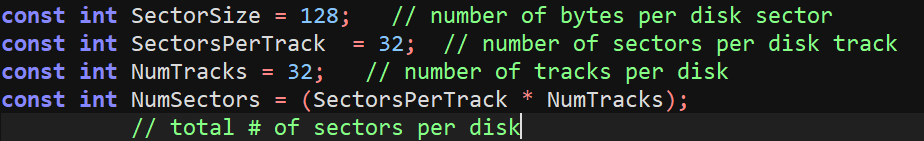
freeMapFile的file header儲存在sector 0，從file header可以找到file的data blocks。

(2) What is the maximum disk size that can be handled by the current implementation? Explain why.

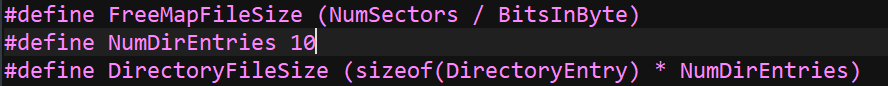
A:

在disk.h裡定義了一個disk的最大sector數=32\*32=1024

所以disk的大小是1024\*128=131072(bytes)



filesys.cc



filehdr.h



current implementation:

(sector size=block size in NachOs)

maximum disk size(sectors)=

bitmap hdr+directory hdr +

freeMapFile data block+

directoryFile data block +

directory entry number hdr+directory entry number\*maximum data blocks per file

=1+1+

Roundup(1024/8/128)(NumSectors/BitsInByte/SectorSize)+

Roundup(20\*10/128)(sizeof(DirectoryEntry)\*NumDirEntry/SectorSize)+

10+10\*(128-2\*4)/4 (NumDirEntries+NumDirEntries\*(SectorSize-2\*sizeof(int))/sizeof(int))

=2+1+2+10+300

=315(sectors)

maximum disk size in bytes=315\*128=40320(bytes)

(3) Explain how the NachOS FS manage the directory data structure? Where is this information stored on the raw disk (which sector)?

A:

使用OpenFile \*directoryFile。在disk儲存為file，讀取時使用directory data structure接收directory的內容(呼叫FetchFrom())。之後可以在memory對directory進行操作，像是:

Find(char\*name):從directory root找到名字是name的file

Add(char\*name,int newSector):在directory新增directory entry，file header 儲存在第newSector個sector上

Remove(char\*name):移除叫做name的file。

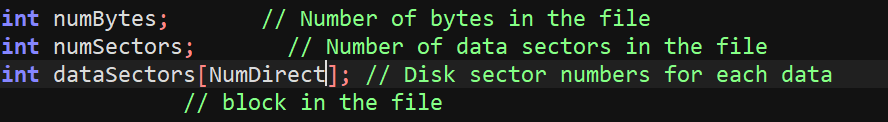
最後使用完directory的操作後，會呼叫WriteBack()寫回去disk上更新。

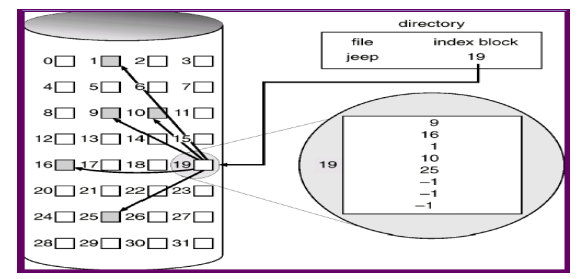
directoryFile file header儲存在sector 1。從file header可以找到所有的directory entry。

(4) Explain what information is stored in an inode, and use a figure to illustrate the disk allocation scheme of current implementation.

A:

filehdr:





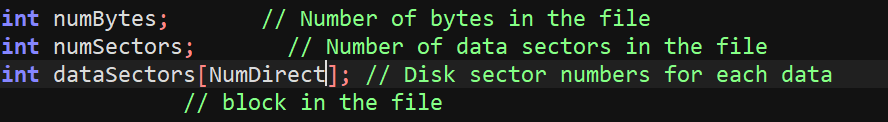
current implementation:

使用indexed allocation。directory entry儲存的是flie header的sector，然後再從file header找到所有的data blocks。Allocation會配和bitmap來使用，bitmap會記錄所有sector被使用的情形。file header會呼較 Allocate(PersistentBitmap \*bitMap, int fileSize)，而bitmap會檢查是否有沒被使用的sector，如果有則設置為已使用。

(5) Why is a file limited to 4KB in the current implementation?

A:

filehdr:



目前fileheader的大小被設置成剛好是1 sector size，所以換算下來file header可以容納

dataSector是NumDirect=30

30\*128+128(file header size)=3968，大約是4KB

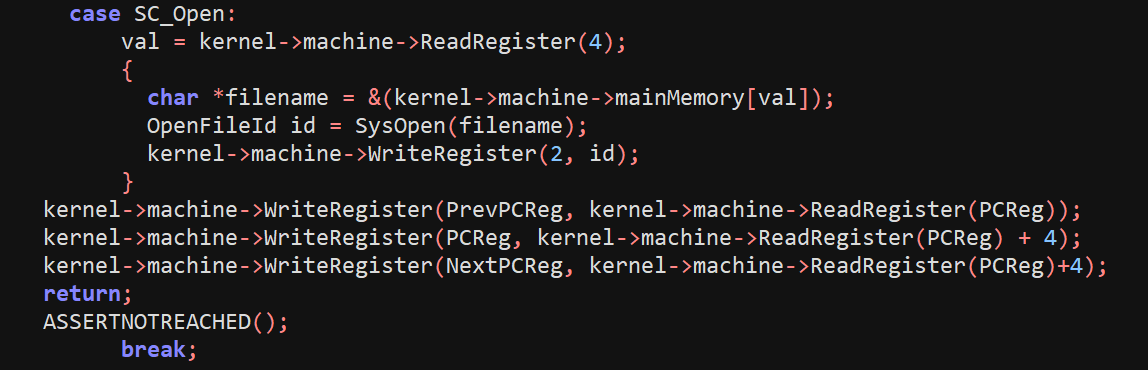
Part II.

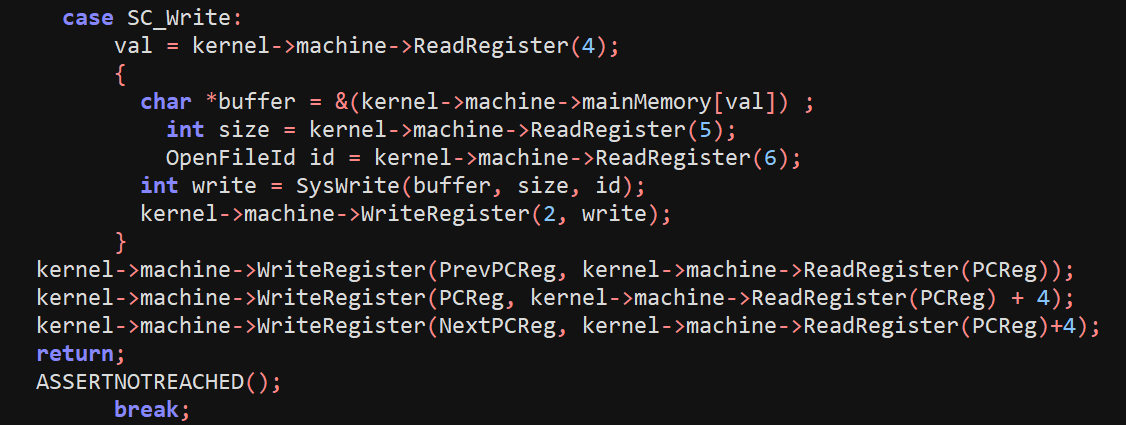
Modify the file system code to support file I/O system call and larger file size

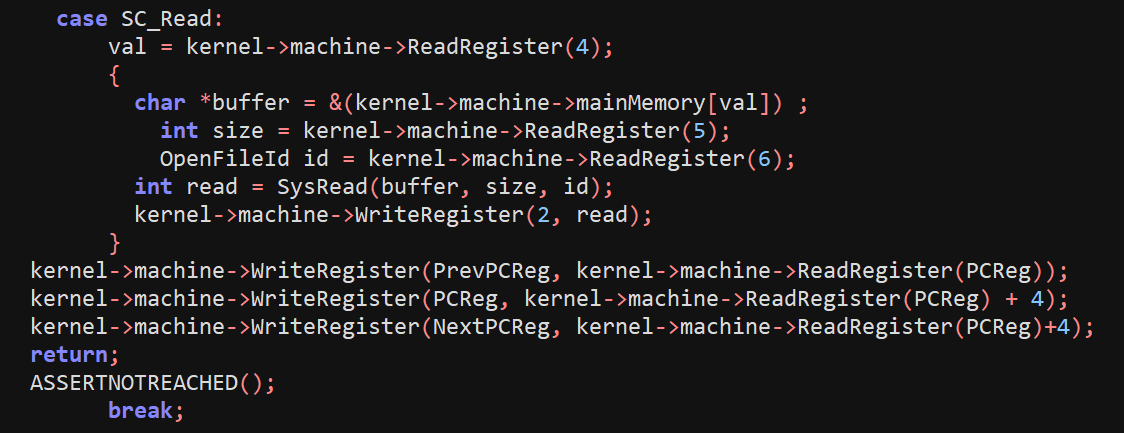
(1) Combine your MP1 file system call interface with NachOS FS

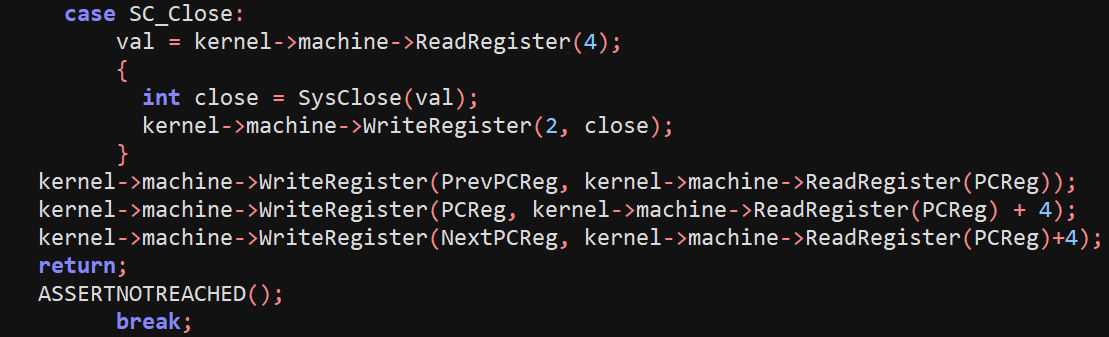
(2) Implement five system calls:

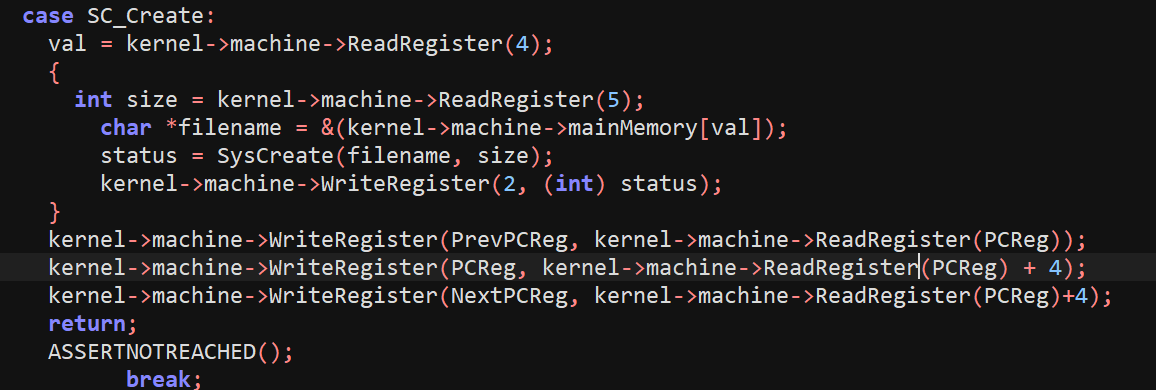
在exception.cc新增5個Case:







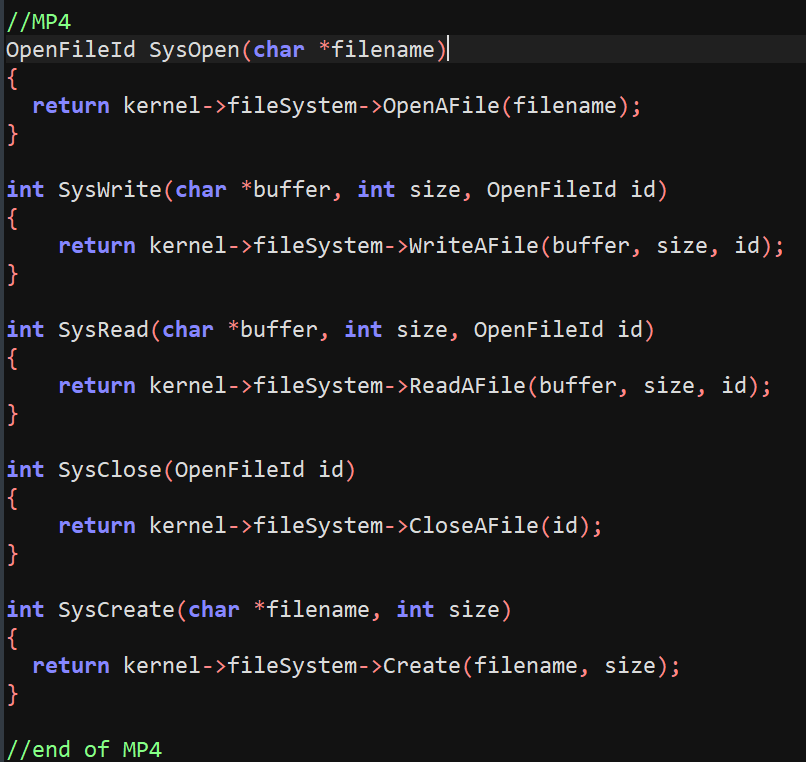




基本上跟MP1的內容一樣，都是讀取參數，丟到SysXXX system call，把output(status)寫回register。最後更新PrevPCReg,PCReg,NextPCReg。

差別在於SysCreate多了size這個參數，描述要創立的檔案大小。

在ksyscall.h新增上述對應的system call，由fileSystem 負責執行



5 system call 對應spec的關係:

int Create(char \*name, int size)->SysCreate

OpenFileId Open(char \*name)->SysOpen

int Read(char \*buf, int size, OpenFileId id)->SysRead

int Write(char \*buf, int size, OpenFileId id)->SysWrite

int Close(OpenFileId id)->SysClose

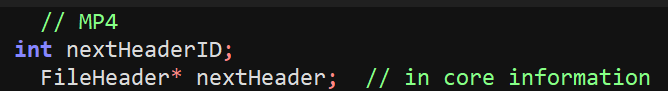
由於各funcion在FileSystem牽扯到path以及subdirectory的實作，故在Part III 再一起說明。

(3) Enhance the FS to let it support up to 32KB file size

filehdr.h

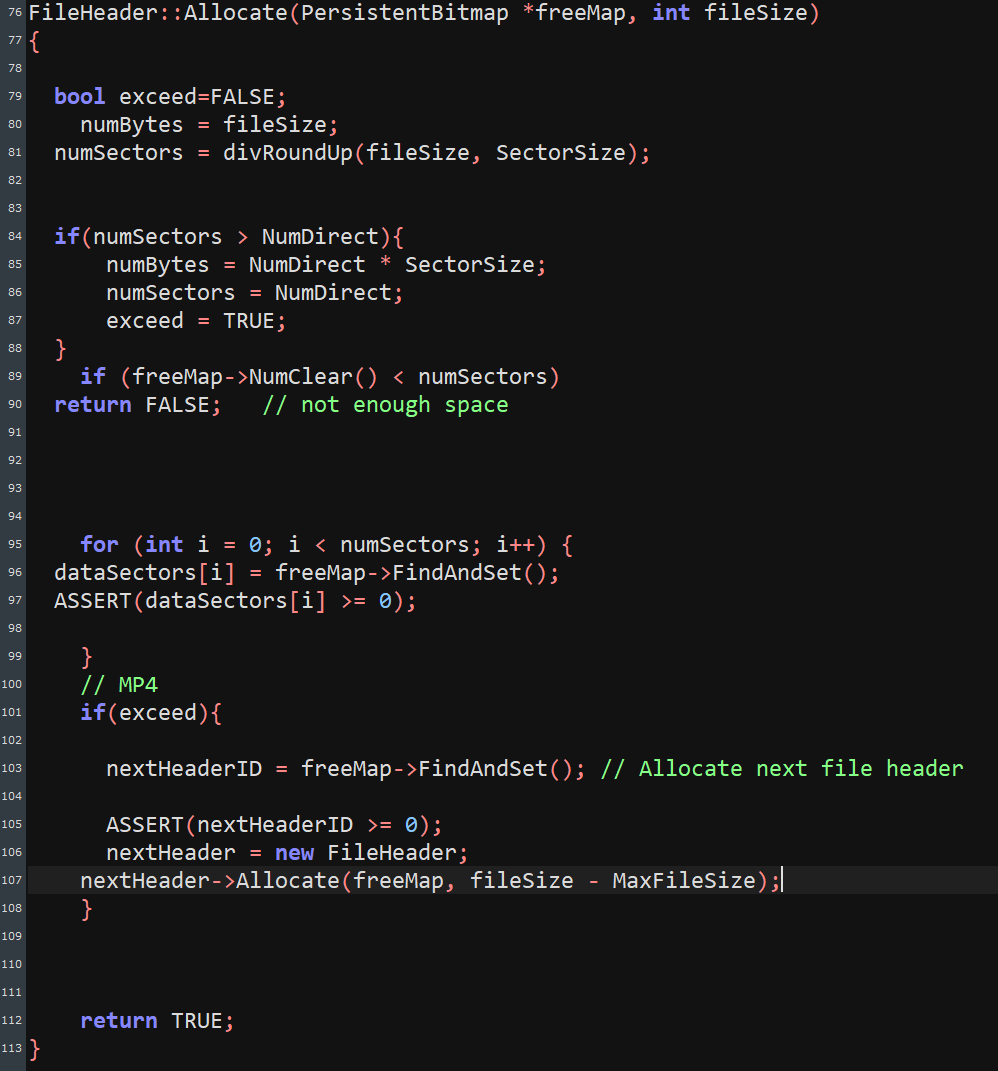


由於在FileHeader新增nextHeaderID這個variable 紀錄下個linked file header的sector number，所以NumDirect從原本-2\*sizeof(int)改為-3\*sizeof(int)。



同時在FileHeader新增nextHeader，說明檔案是否有下個linked file header(初始為NULL)，此為in core information。

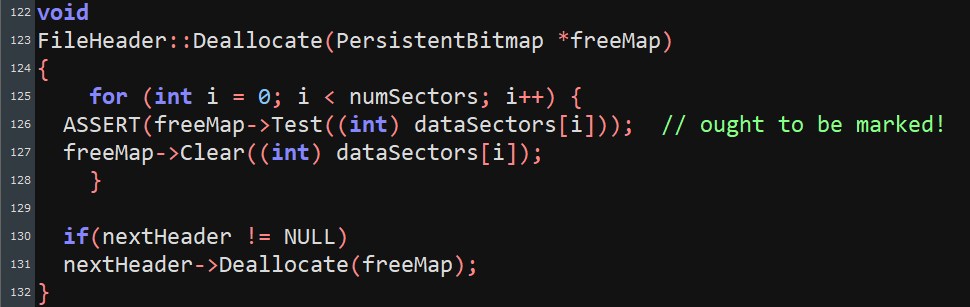
filehdr.cc



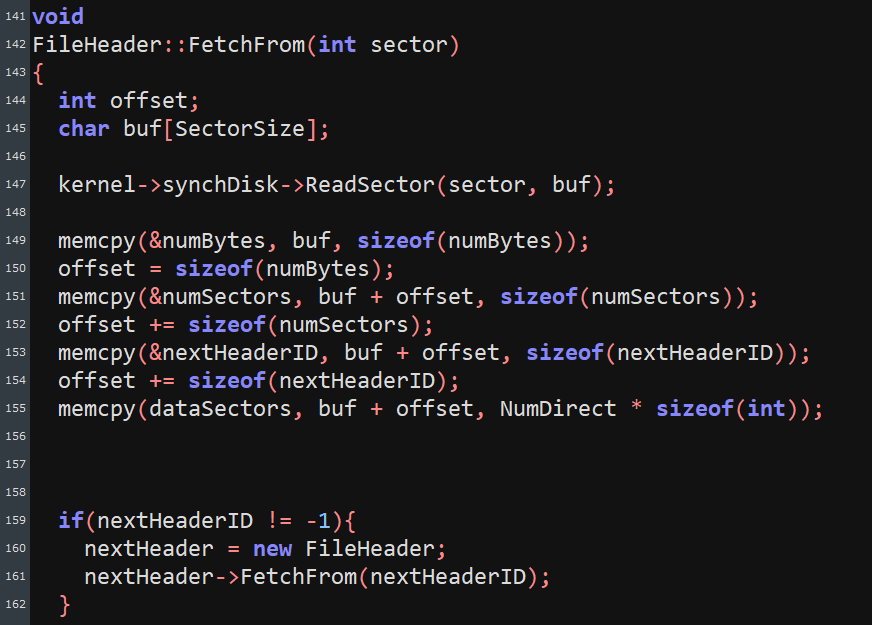
line 84~87:檢查所需的sector數量超過系統規定的NumDirect大小，超過則先設置要分配的NumBytes和NumSectors為FileHeader的上限

line 95~97:利用FindAndSet() allocate sectors and update freeMap，dataSectors會記錄使用了那些data blocks。

line 101~107:如果超過當前file header可以管理的sector數量，創造nextHeader並allocate剩餘的file size。



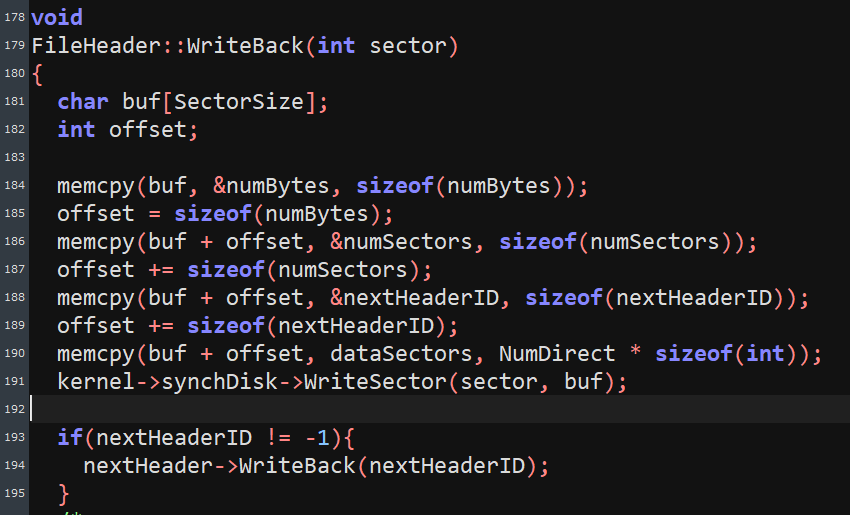
line 130+131:如果使用到nextHeader的話也要deallocate該file header所擁有的sector。



line147:讀取整個sector的資訊放到buf上

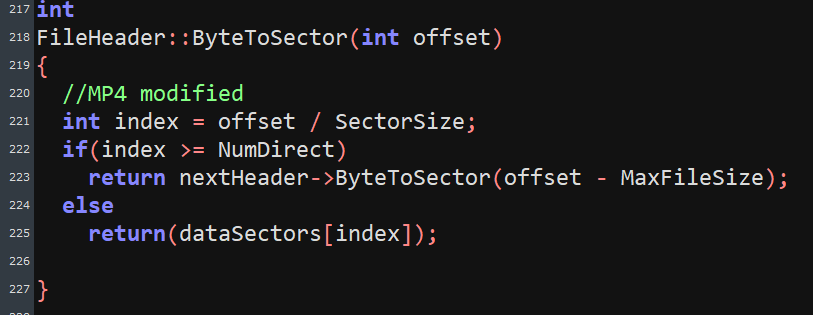
line149~155:個別更新numBytes,numSectors,nextHeaderID,dataSectors，每更新一次offset+=讀的資料大小，這要做的原因是因為FetchFrom()不需要讀整個sector的資訊。

line159~161:如果nextHeader有用到的話也要同步更新。



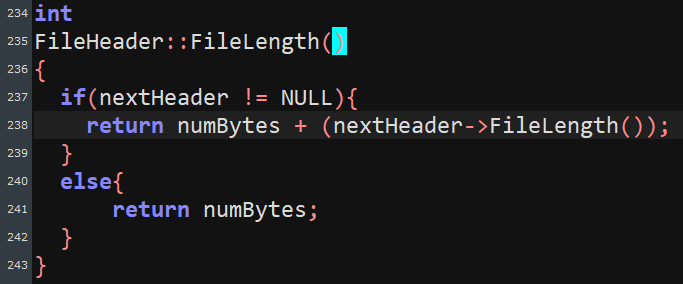
WriteBack()也是一樣只需要修改部分的資訊就好，因為不希望動到in core information。

使用buf接收部分的資訊然後寫回file header，如果使用到nextHeader的話也要同步更新。

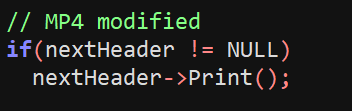


如果index超過NumDirect的話呼叫nextHeader的ByteToSector()計算sector number，

否則return dataSectors存放的sector number。



recursive呼叫FileLength()直到沒有使用nextHeader為止，return 一路上所有numBytes的總和



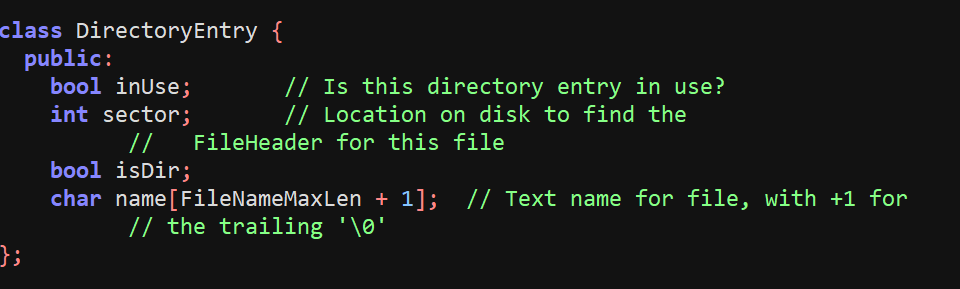
在Print()新增，讓nextHeader也可以show information。

Part III. Modify the file system code to support subdirectory

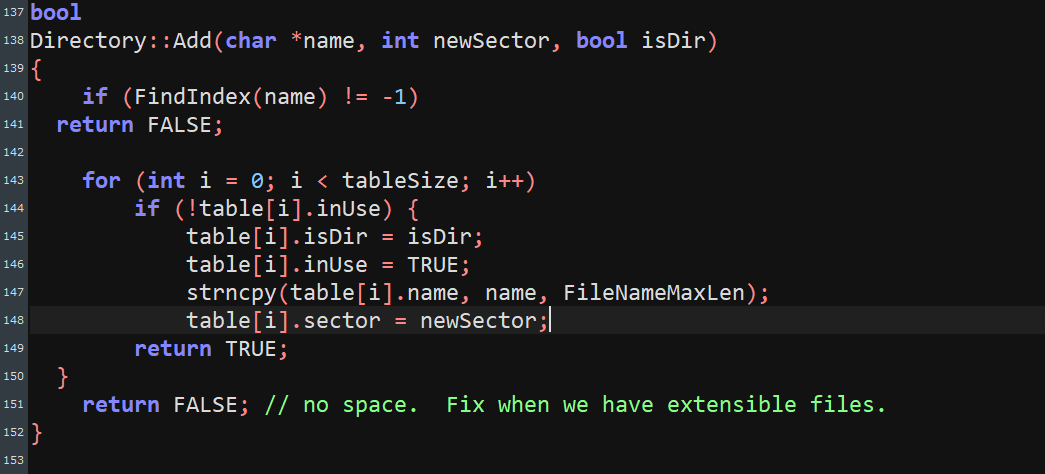
directory.h/directory.cc



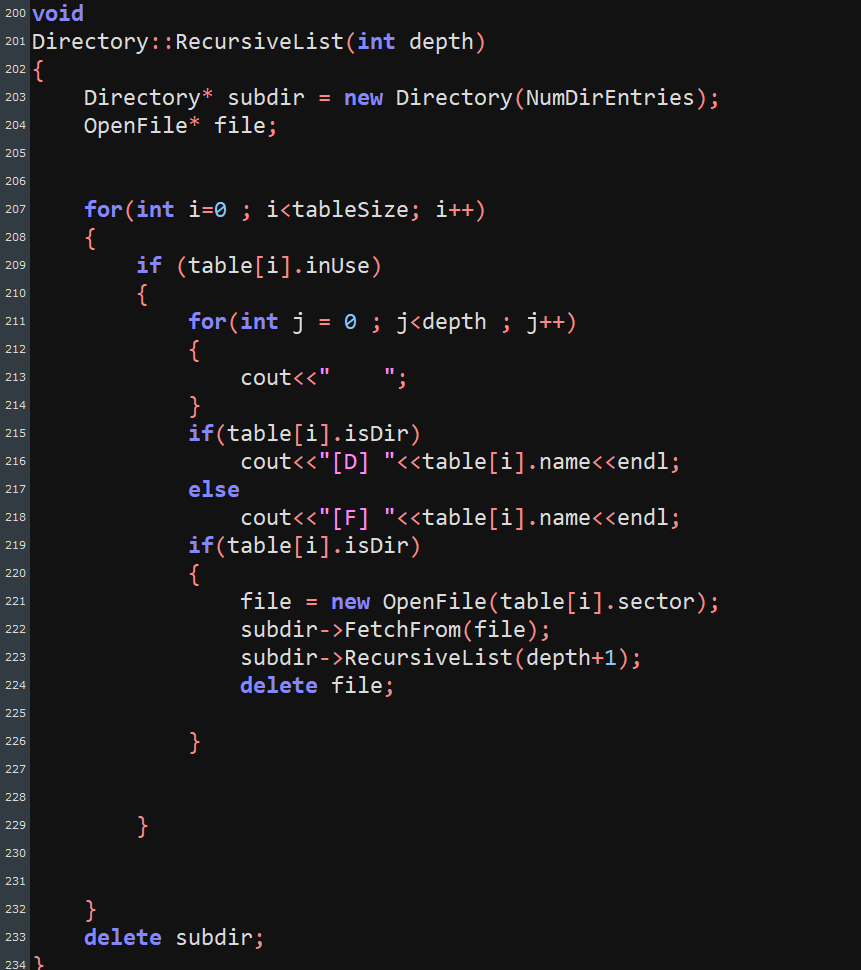
讓nachos可以"Support up to 64 files/subdirectories per directory"



在DirectoryEntry新增isDir去判斷是否是subdirectory or file



Add新增參數isDir，代表新增的是file還是directory



int depth 代表離root directory多遠，根據depth顯示多少的空格。

line207~224:掃過table，如果是file就印出來，是directory則是根據file header sector 取出subdirectory的內容並recursive 呼叫RecursiveList()。

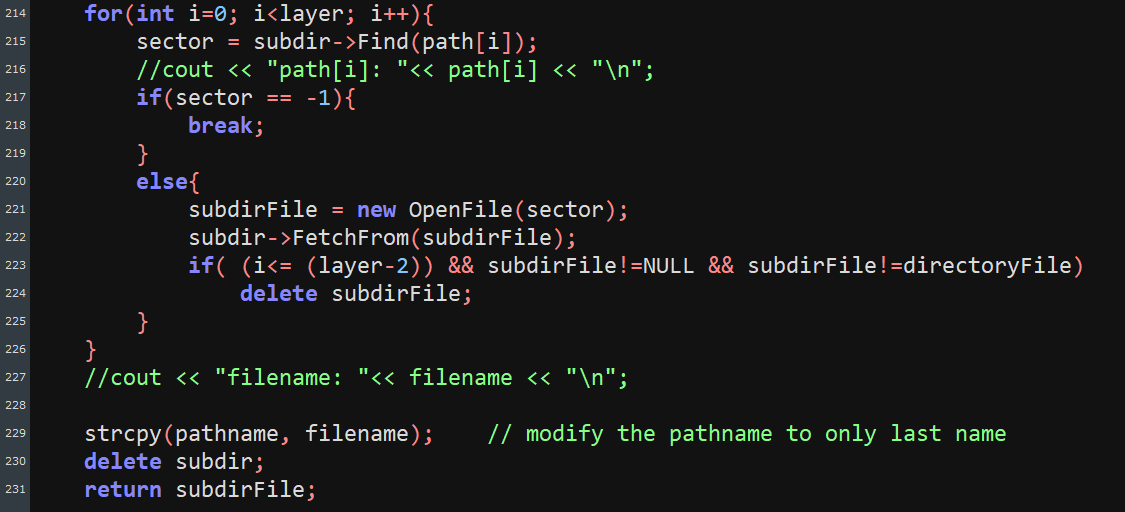
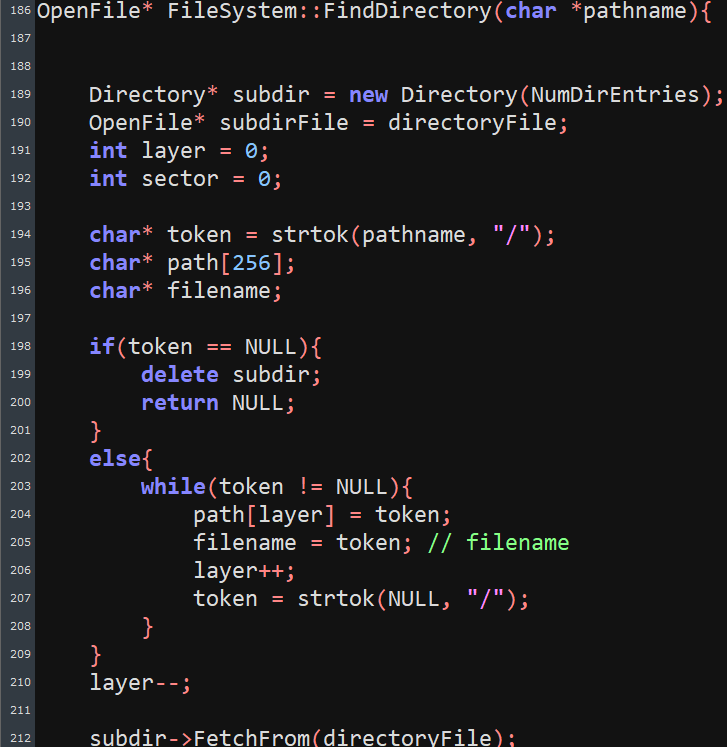
filesys.h/filesys.cc



設定可以同時開啟的檔案數



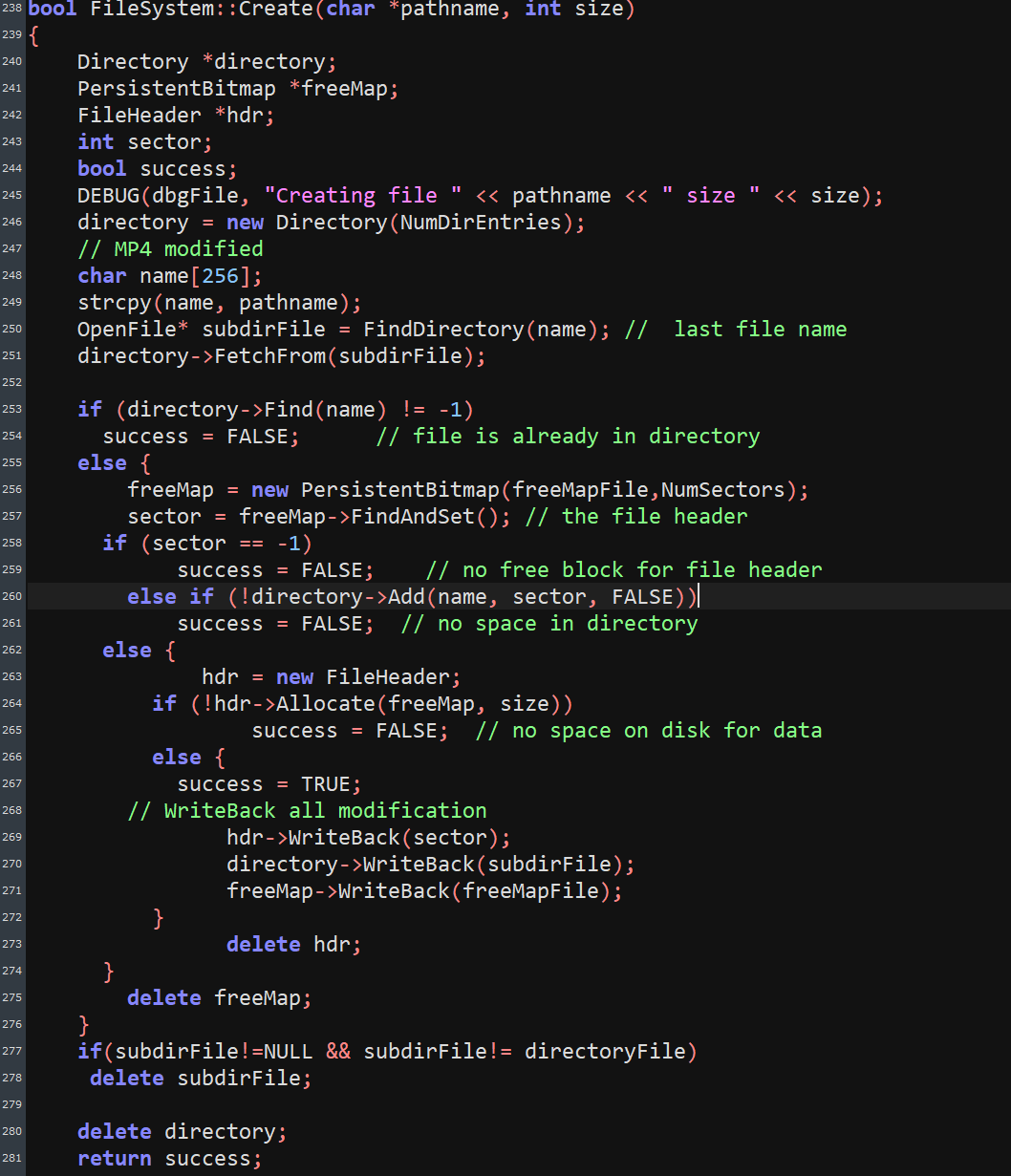
在FileSystem設定System Wide Open File Table



line 203~207呼叫strtok直到把pathname切割完(每個path[layer]代表directory，最後一個可能是file)

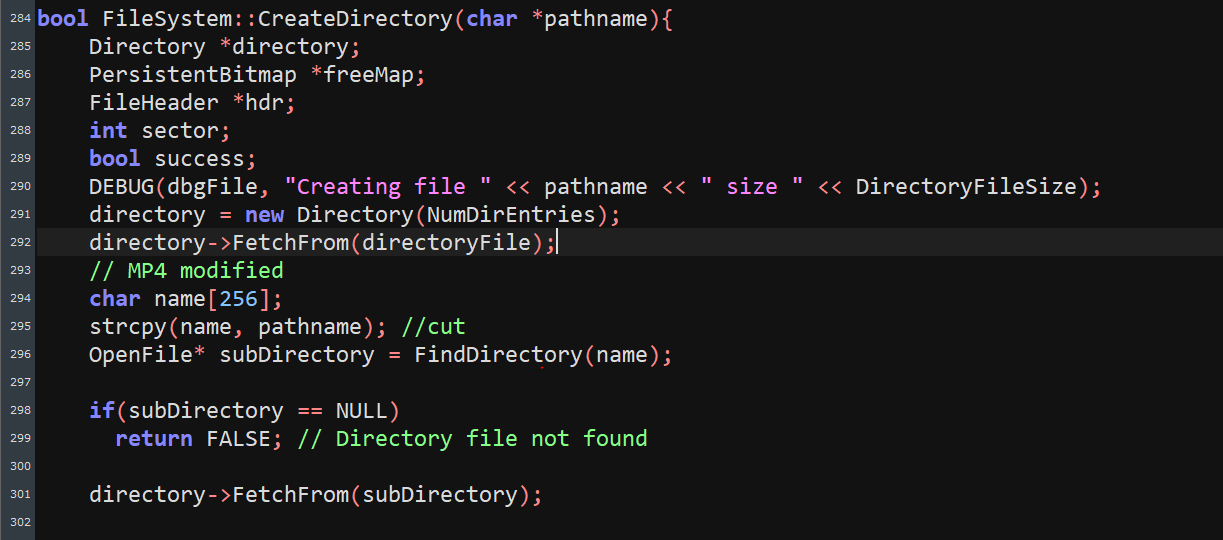
line214~224:根據每段的path[i]往下搜尋，根據搜到的file header sector創建新openfile來讓subdir得到下層的table，持續呼叫直到最後filename的前一個openfile停下。

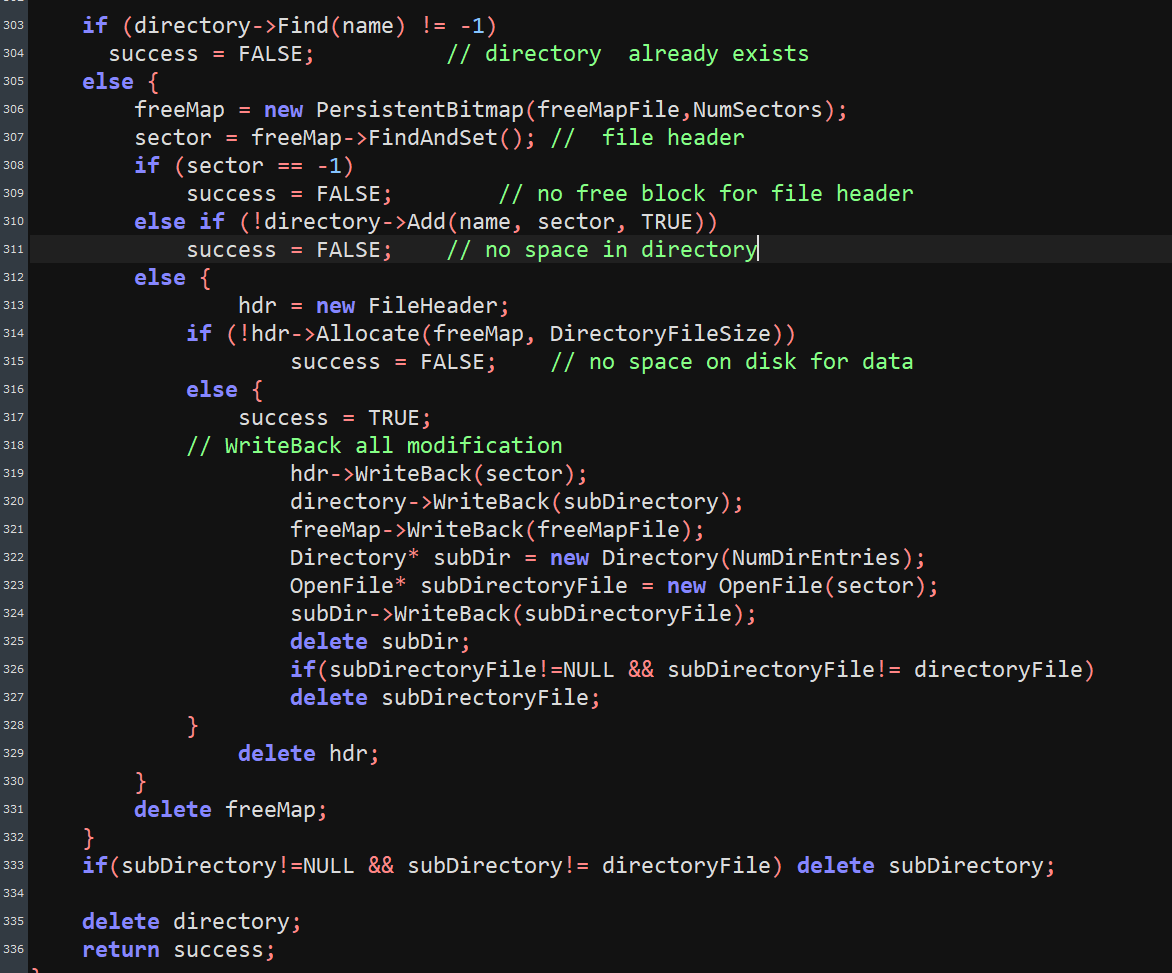
line229:修改pathname變成底層的filename。



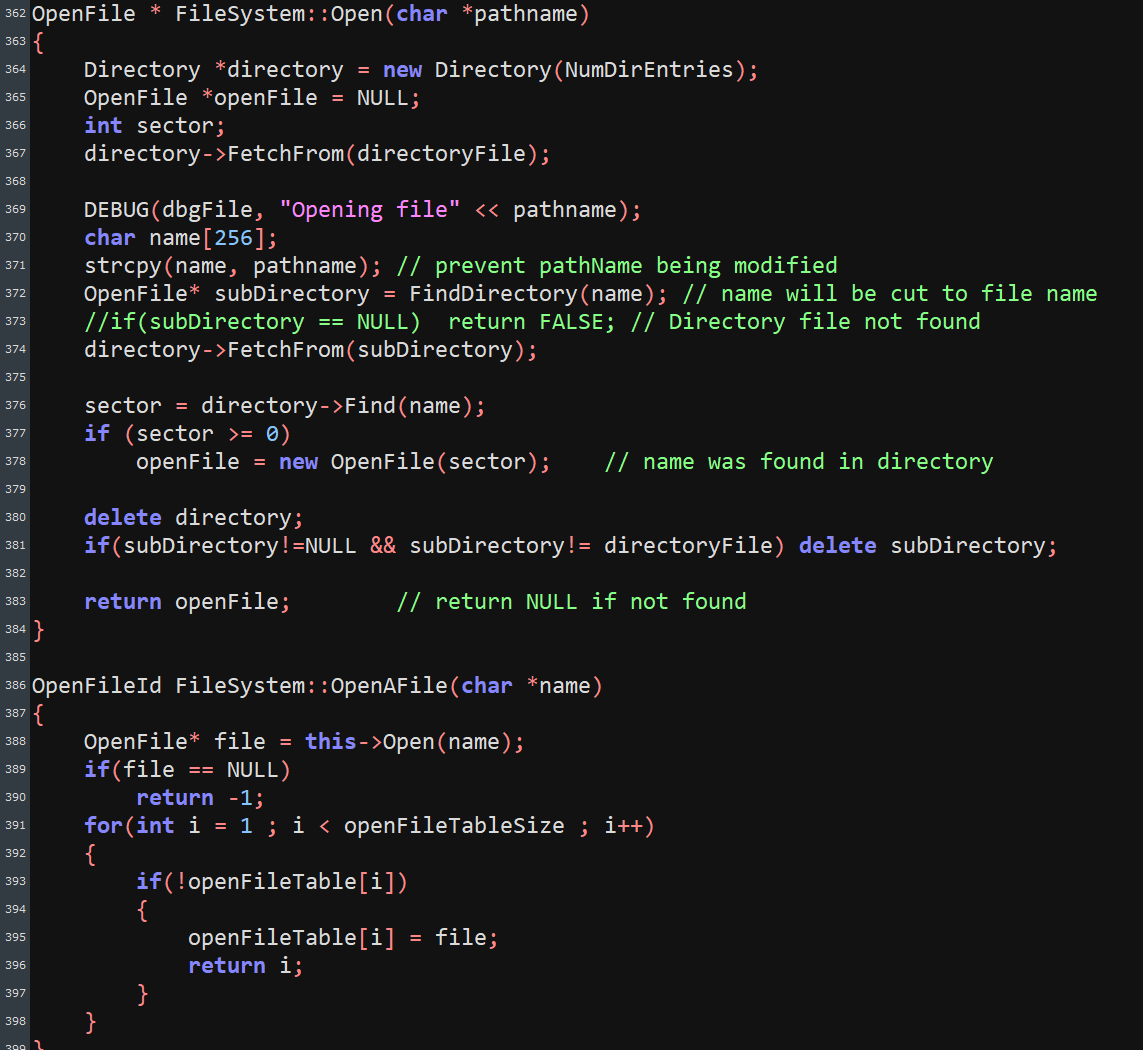
line249~251:把pathname讀到name，name的大小是255(spec)。FindDirectory(name)會把新增file的前一個directory的openfile找出來，並修改name變成最後file name。

line255~271:freeMap allocates file header sectors,adds new file to the directory,and allocates all data blocks for the file。最後WriteBack 所有的改動。





基本上跟Create()一樣，只是新增的是directory，以及最後WriteBack時候要多一次寫回新創的directory。

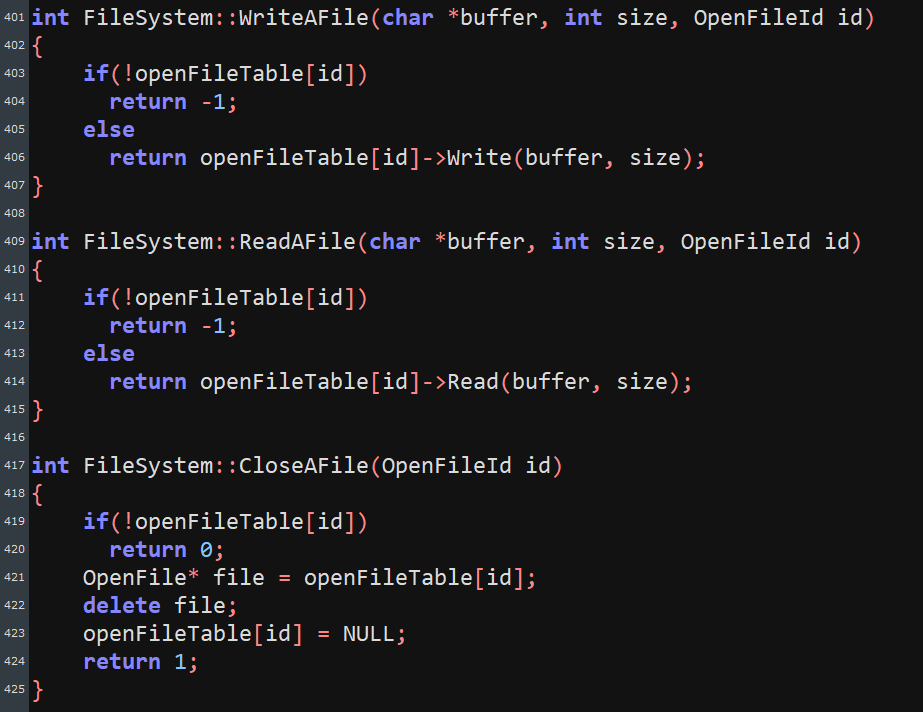


Open():

用FindDirectory()找到最後file前一個directory的openfile，接著創造directory去找file header sector，回傳該sector形成的openfile。

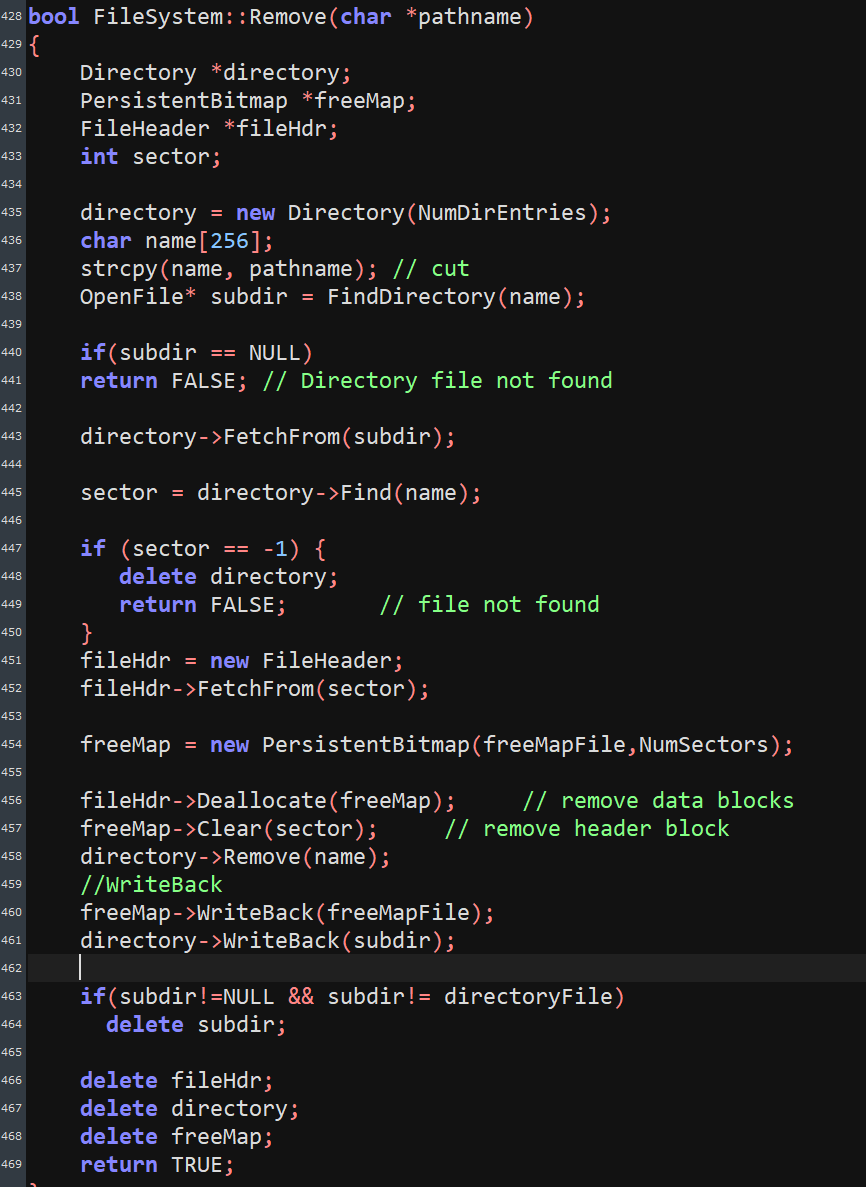
OpenAFile():

根據Open()回傳的openfile 放置在空的openFileTable上，return 放置的index。

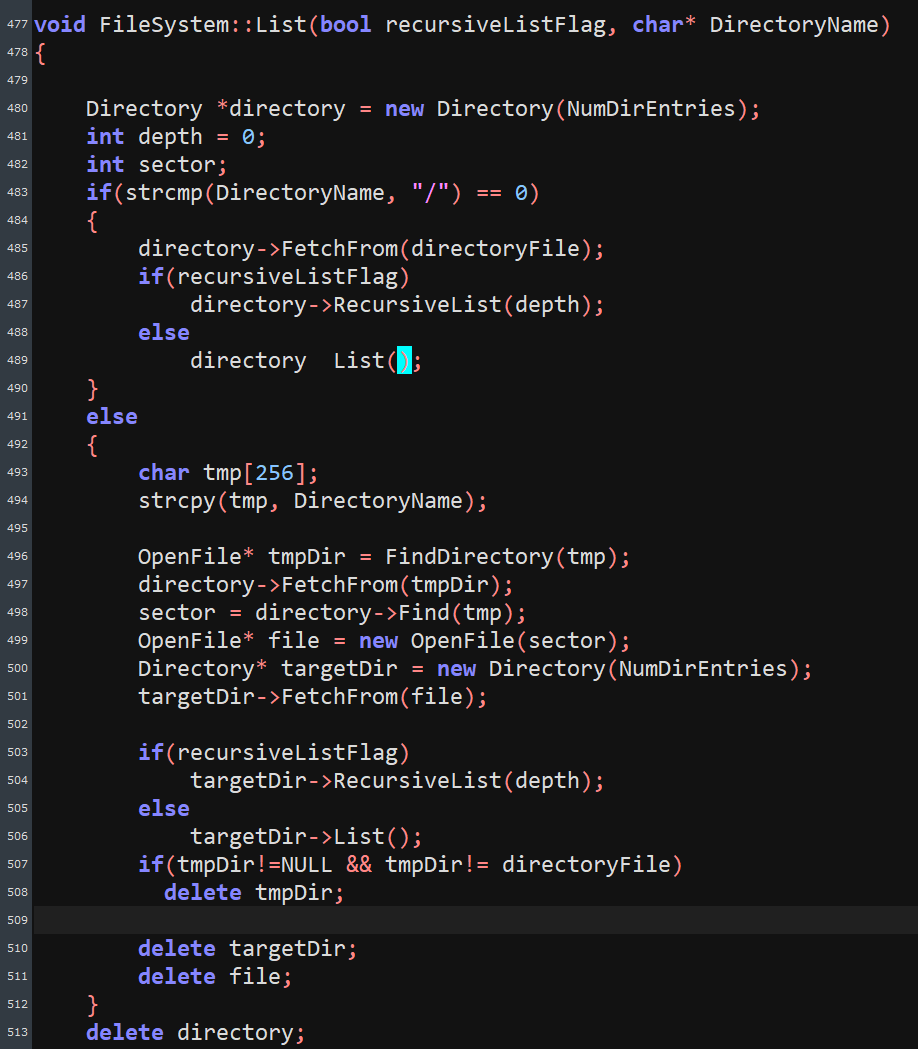


WriteAFile(),ReadAFile,CloseAFile:

在openFileTable找到對應的openfile並執行相關的操作。



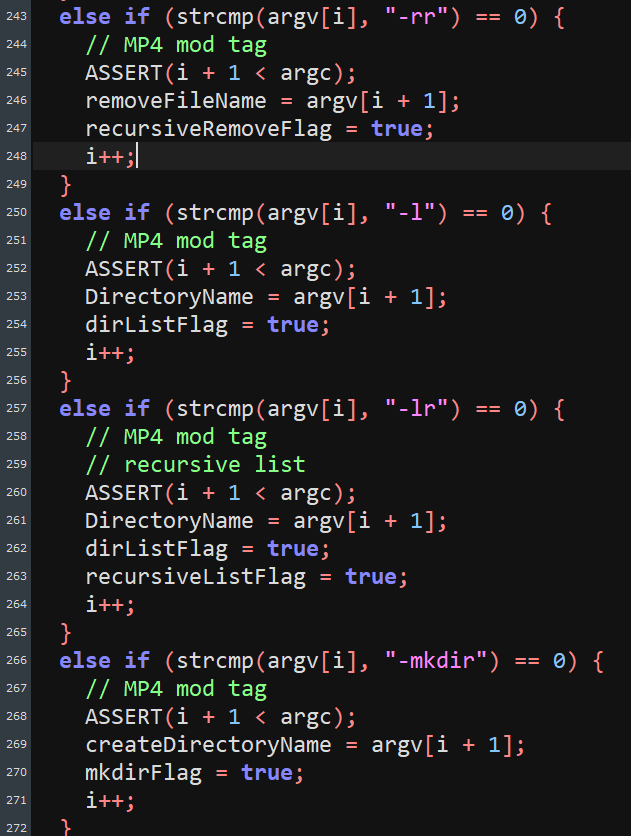
找到要移除file的的前一個subdirectory，找到file header sector並呼叫freeMap deallocate file header and data blocks，並移除directory的file，最後WriteBack freeMap+directory。

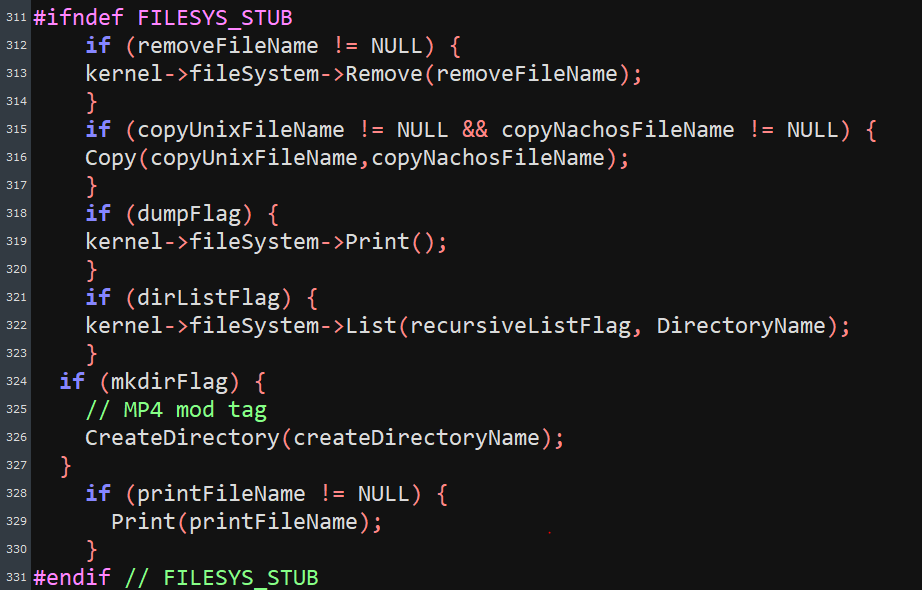


line483~489:如果是從root directory list的話根據recursiveListFlag直接呼叫RecursiveList()或是List()

line493~506:找到對應file/directory的file header sector並初始化為OpenFile\*file，並讓directory讀出file的內容並根據recursiveListFlag呼叫RecursiveList()或是List()。

main.cc





新增:

char \*createDirectoryName = NULL;

char \*DirectoryName = NULL;

bool mkdirFlag = false;

bool recursiveListFlag = false;

bool recursiveRemoveFlag = false;

並在上述新增的指令觸發時設置flag與接收參數，並根據設置的flag呼叫fileSystem的function。