**CS342301: Operating System**

**MP4: File System**

組別: 8

組員:

109062315洪聖祥

109062314張紘齊

|  |  |
| --- | --- |
| 項目 | 分工 |
| 5 system calls | 張紘齊、洪聖祥 |
| **BASIC** |  |
| File size increase to 64MB | 張紘齊 |
| Implement subdirectory | 洪聖祥 |
| **BONUS** |  |
| Multilevel header size | 張紘齊 |
| Recursive removal | 洪聖祥 |

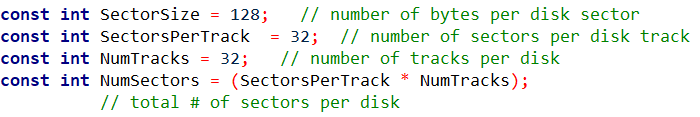
**Part I: Understanding NachOS file system**

* Explain how the NachOS FS manages and finds free block space? Where is this information stored on the raw disk (which sector)?

NachOS FileSystem中有個變數是freeMapFile，它被放在disk sector 0，而它會透過bitmap的方式記錄各個disk sector是否被使用，

* What is the maximum disk size that can be handled by the current implementation? Explain why

如果不考慮Magic Number佔的空間，maximum disk size為2^(7+5+5) = 128KB

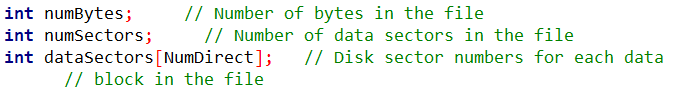




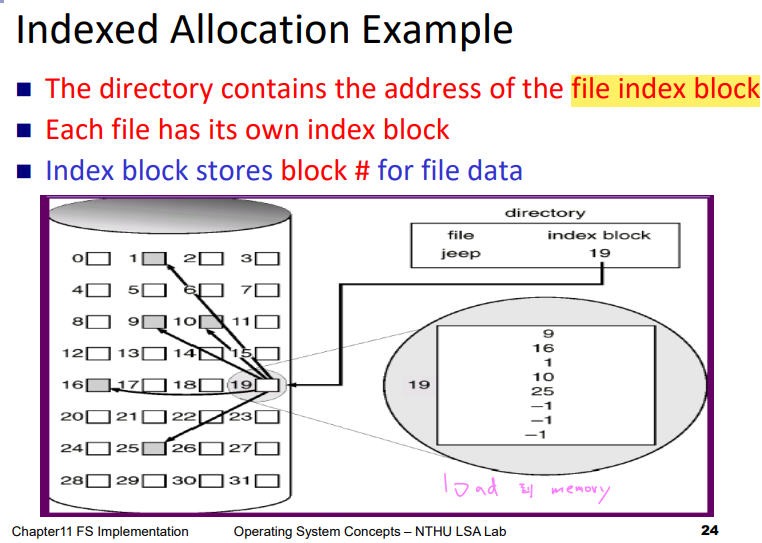
* Explain how the NachOS FS manages the directory data structure? Where is this information stored on the raw disk (which sector)?

NachOS FileSystem中有個變數是directoryFile，它被放在disk sector 1。目前它的結構是single level directory，且只有10 entries，而每個entry會記錄 (file name, sector #) pair

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



Allocation scheme為Index allocation

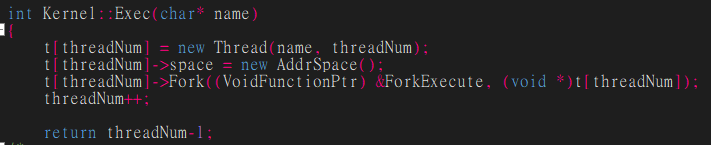


* Why is a file limited to 4KB in the current implementation?

在inode中，numBytes、numSectors與dataSectors的每一個entry (共30個entry)會各佔一個sector，因此可存file的大小為32\*128 = 4KB。若是考慮能存入的data size，則就只有30\*128 bytes

**Part II: Modify the file system code to support file I/O system calls and larger file size**

* How does Nachos allocate the memory space for a new thread(process)?

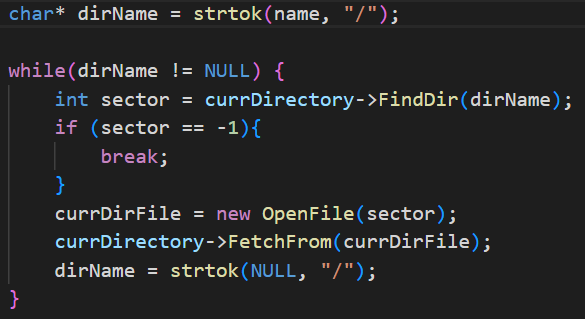


在Kernel::Exec中的第二行會new AddrSpace()，並把它放入thread中的space變數，也就是替thread allocate memory space

**Part III: Modify the file system code to support the subdirectory**

Basic part 輸入的指令都是absolute path，所以很多地方都要找到新建的 file/directory的parent directory (因為是新建的，所以目前在file system中沒有輸入指令的abolute path)或是如果absolute path有這個directory直接找到。因此我寫了一個code來處理絕對路徑並找到parent directory。我們要沿著絕對路徑從root directory往下找到要加directory的名字的parent directory，並在parent directory底下加入新的directory/file或是直接找到目標的directory。

以下是我的code (很多地方都用到同樣的code):



用一個迴圈從root directory 往下找到要加的parent directory。首先currDirectory是root directory，並看currDirectory有沒有找到相應名字的directory。如果有，會回傳找到的directory的file header的sector number，直到找到新建的 file/directory的parent directory。

* Create Directory

新建一個directory會呼叫bool FileSystem::CreateDirectory(char\* name)。接來會執行上面的code找到新建的directory的parent directory。找到之後，就跟之前一樣，從freeMap找有空的sector。分配DirectoryFileSize的空間，把file header寫回空的sector。Parent directory因為也有更跟table所以也要寫回disk。

* Copy File to NachOS file system

主要是把現存的file放到NachOS 的file system。在main.cc會呼叫Copy(char\* from, char\* to)。Copy()會呼叫kernel->fileSystem->Open()和kernel->filesystem->Create()。

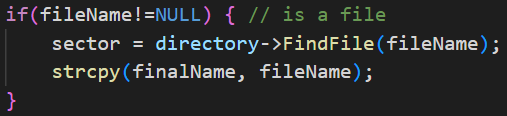
首先是 int FileSystem::Open(char\* name, int initialSize)。跟create directory一樣。先從絕對路徑找要新建的file的parent directory。跟上面的code 一樣，但是現在我們要建的是file而不是directory。因為我在額外寫了int Directory::FindDir(char\* name)，這個函式只會找名字符合的directory，所以經過while loop，跳出迴圈後的currDirectory一定是要新建的file的parent directory。找到parent directory後，操作就跟template一樣。從freeMap找空的sector number，在currDirectory加入我們要建的file，分配data block給新的file，把file header寫進剛剛從freeMap找的sector number，把currDirectory更新的資訊寫回disk (因為currDirectory加入了新的file有新的data)，最後把freeMap更新的資訊寫回disk(因為分配了block給新的file)。

再來是 OpenFile\* FileSystem::Open(char\* name)。一模一樣的操作，從絕對路徑找到新建的file的parent directory。接下來從parent directory找到剛剛新建的file。我額外寫了int Directory::FindFile(char\* name)，這個函式只會找符合名字的file，並回傳找到file 的file header的sector number。並回傳file header是這個sector number的OpenFile pointer。

* Remove a file

移除一個file會呼叫 bool FileSystem::Remove(char\* name, bool isRecur)。指令 -r 和 -rr 都會呼叫 bool FileSystem::Remove(char\* name, bool isRecur)。如果是 -r ，isRecur = False，如果是 -rr ，isRecur = True。

這裡先解釋 isRecur = False，isRecur = True在 Bonus Part解釋。這裡remove a file也是一樣會找到target file的parent directory，也是先執行上面的code找到parent directory。因為沒有messy operation(下面code fileName 對於remove a file fileName一定不是NULL)，所以找到parent directory後可以找target file。

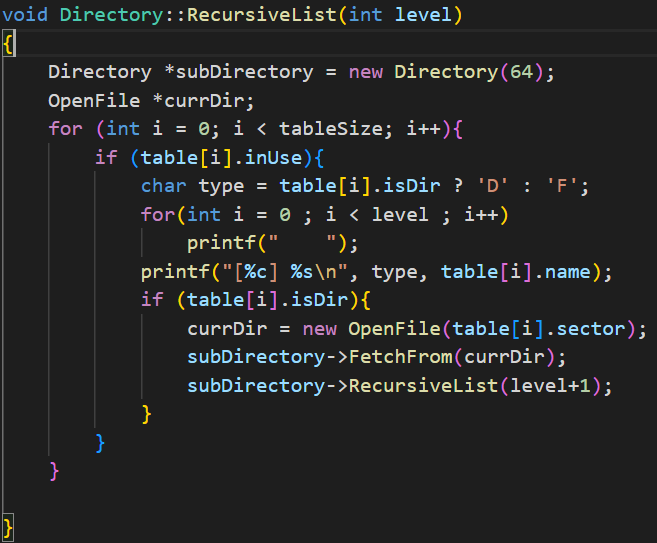


我額外寫Directory::FindFile(char\* name)來找file。找到target file 的file header的 sector，我們就可以remove這個file了。移除file header指到的data block，移除 header block。因為parent directory有更新資料(一個file移除)，所以寫回disk。freeMap也有更新資料( data block、header block被deallocate)，所以寫回disk。

* List a directory / List recursively

List a directory 和 List recuresively都會呼叫 void FileSystem::List(char\* name, bool isRecur)。如果輸入的指令是 -l，isRecur是False。如果輸入的指令是 -lr，isRecur是True。FileSystem::List(char\* name, bool isRecur) 首先一樣會執行上面的code。因為我們輸入指令的absolute path一定存在，所以我們會找到目標的directory。接下來會在目標directory list 或 recursively list，取決於 isRecur。

如果isRecur是False，呼叫Directory::List()。如果isRecur是True，呼叫新寫的void Directory::RecursiveList(int level)。



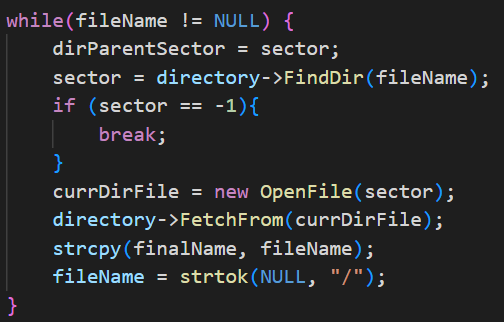
主要是用DFS的概念來做recursion，參數level是表示在第幾層directory。

**Bonus Assignment**

* Bonus1
* Bonus2: Recursively Remove

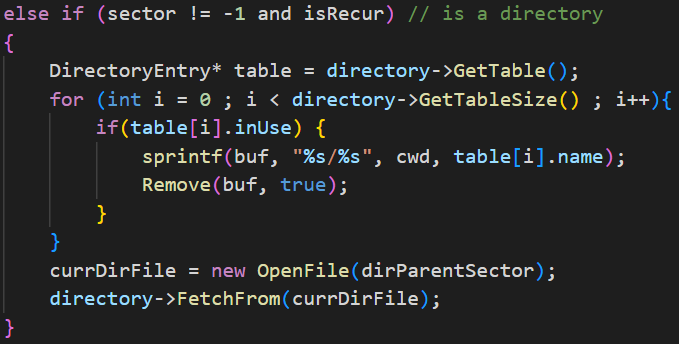
指令 -r 和 -rr 都會呼叫 bool FileSystem::Remove(char\* name, bool isRecur)如果是 -r ，isRecur = False，如果是 -rr ，isRecur = True。

這裡解釋isRecur = True，在Basic Part，我只解釋remove file的部分，但Recursive removal也可能remove directory，再從absolute path找到parent directory要修改一下。



我這裡多加一個叫dirParentSector的變數，這是為了要記錄taget directory的parent directory。因為remove operation的 absolute path一定存在，不像create file 或create directory，最後一個名字一定不存在，所以while loop如果最後一個名字是directory，finalName是absolute path的最後一個名字。例如 absolute path 是 /test/hw1/data 且data是directory，照上面的寫法，directory會是data這directory，但我們在remove玩data這個directory的subdirectory/file，也要remove data這個directory，而且parent directory也要更新資訊，所以用一個變數dirParentSector紀錄data directory的parent directory的file header的sector number。

剛剛有講要先remove 經過absolute path 最後的名字的directory 的subdirectory再移除本身directory。移除subdirectory用recursion的方式。



簡單來說就是看目前directory底下有沒有東西，如果有，path加上目前的path再加上底下東西的name，再來呼叫Remove()。接下來remove所有目前directory的subdirectory後，把directory指到parent directory。然後把absolute path最後一個名字的directory remove，parent directory也要更新資訊。

* Bonus3