

2021 OS MP4

TEAM15

108062217 傅詠軒

108062203江浩辰

Contribution:

江浩辰50%

傅詠軒50%

**Part I. Understanding NachOS file system**

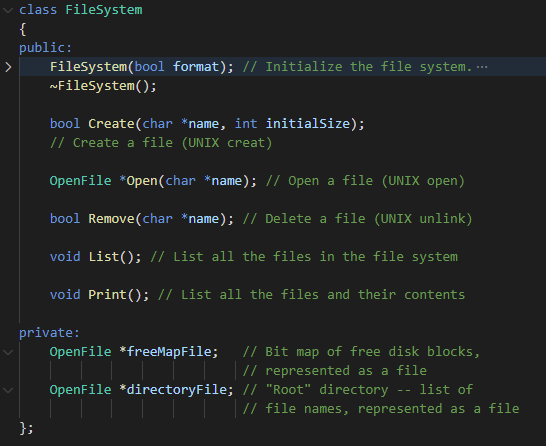
**(1) Explain how the NachOS FS manage and find free block space? Where is this information stored on the raw disk (which sector)?**

一開始由 Kernel:Initialize()初始化Filesystem。

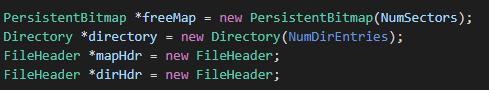
Filesystem中有兩個private變數，皆以file的形式儲存，並由OpenFile來管理，其中freeMapFile用來儲存free blocks。

freeMapFile : 用來記錄free disk blocks的bitmap.

directoryFile : 根目錄，用來記錄所有file的名稱。

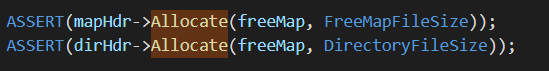


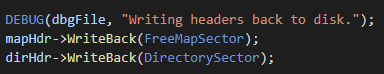
在初始化時，會consturct一個值全都為0的Bitmap → freeMap(初始化大小為NumSectors且值皆為0)，以及construct Bitmap file的FileHeader mapHdr。



把freeMap上的FreeMapSector的free block標成已使用，並allocate space給freeMapFile，之後把FileHeader寫回disk上，最後freeMapFile會以openfile的形式在Nachos上做使用。



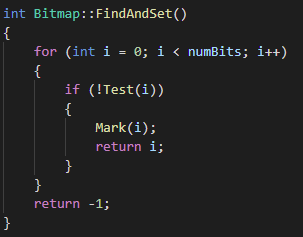
****





如果要管理free blocks則會透過freeMapFile的freeMap來管理。

如要find free block space的話，則會透過Bitmap::FindAndSet()完成，在FileHeader::Allocate分配blocks時，會根據此file需要的sectors數量(如果足夠的話)，去call Bitmap::FindAndSet()，尋找在bitmap中還free的bit number(by scan through)，也就是disk中的free sector，並把他Mark成used，回傳此sector number。



而管理free block space的freeMap的資訊存在raw disk上的哪個sector可從初始化的過程中得知，freeMap會寫回freeMapFile，而freeMapFile則在Allocate時放到了FreeMapSector的sector上，而FreeMapSector define = 0，因此可以知道會被存在Disk sector 0上

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

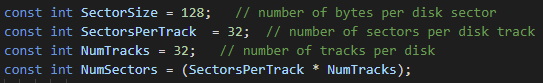
在disk.h內有定義了以下幾個變數：

(1) SectorSize = 128 = 一個sector可存幾bytes

(2) SectorsPerTrack = 32 = 一個track上有幾個sectors

(3) NumTracks = 32 = 一個disk上有幾個tracks

(4) NumSectors = SectorsPerTrack \* NumTracks = 1024 = 一個disk上總共有幾個sectors

****

因此maximum disk size = NumSectors \* SectorSize

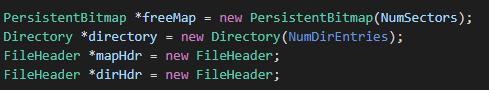
= 1024(sectors) \* 128(bytes) = 131072(bytes) = 128(KB)

但目前一個File的MaxFileSize = NumDirect \* SectorSize = 30 \* 128

= 3840(bytes)，而目前FS只支援最多10個Files，因此目前的implementation最多只有38400(bytes) is actually used。

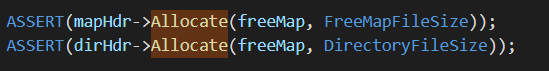
**(3) Explain how the NachOS FS manage the directory data structure? Where is this information stored on the raw disk (which sector)?**

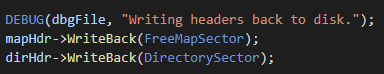
首先回答會store在哪個sector上，基本上跟(1)中的freeMapFile是同時被create跟initialize，在初始化時，會consturct一個能放NumDirEntries數(目前為10)的directory，，以及construct Directory file的FileHeader dirHdr。



把freeMap上的DirectorySector的free block標成已使用，並allocate space給directoryFile，之後把FileHeader寫回disk上，最後directoryFile會以openfile的形式在Nachos上做使用。



****



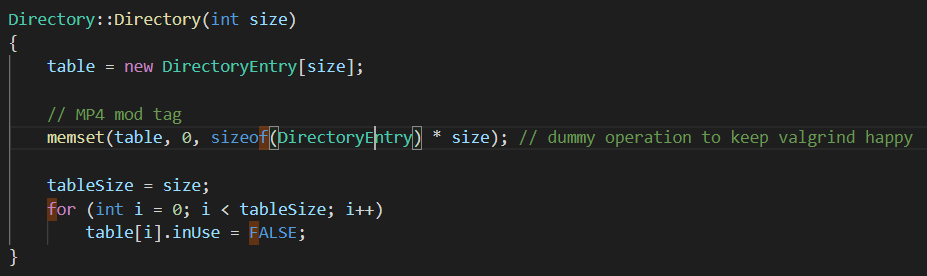


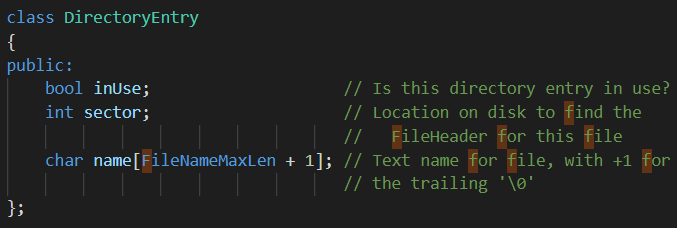
從這我們可以看到manage directory data structure的directoryFile會存在sector number為DirectorySector的sector上，而DirectorySector define = 1，因此可知會存在Sector 1上。

****接下來回答FS如何manage directory data structure。

首先在consturct的時候，會根據傳入的size，也就是directory的numbers of entires去建立一個為Directory Entry array的table，並把tableSize設成Directory的enties size。

每個DirectoryEntry包含的即為Directory內每個entry對應到的file的information，inUse: 此entry有沒有file、sector: 有file的話，其fileHeader在disk上的sector number、name: file name。一開始在建構Directory時會把所有Entry的inUse設為False代表都是free的。





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

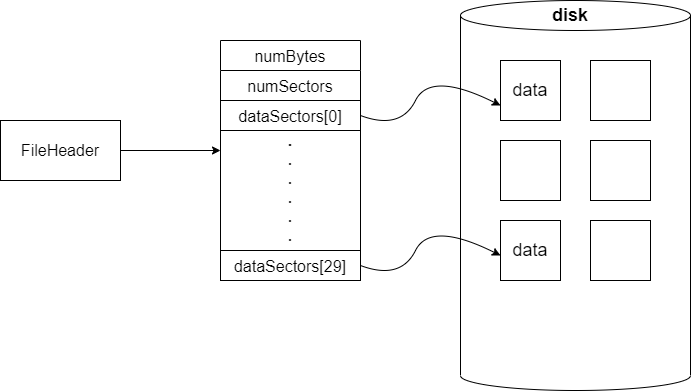
FileHeader即為NachOS的inode(FCB)，因此裡面存的information有：

numBytes: 此file存了多少bytes

numSectors: 此file使用的data sector數

dataSectors: 1D array，用來存data sectors的sector number

Current disk allocation scheme為Indexed Allocation，figure如下：



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

因為目前NachOS使用的是single indirect Indexed allocation，因此一個FileHeader(need a sector存放)會point到30個data blocks，也就是說每個file會limit在30個data blocks以內。

且因為NachOS視一個block為一個sector，30個data blocks

= 30個sectors = 30 \* 128 (一個sector有128 bytes)

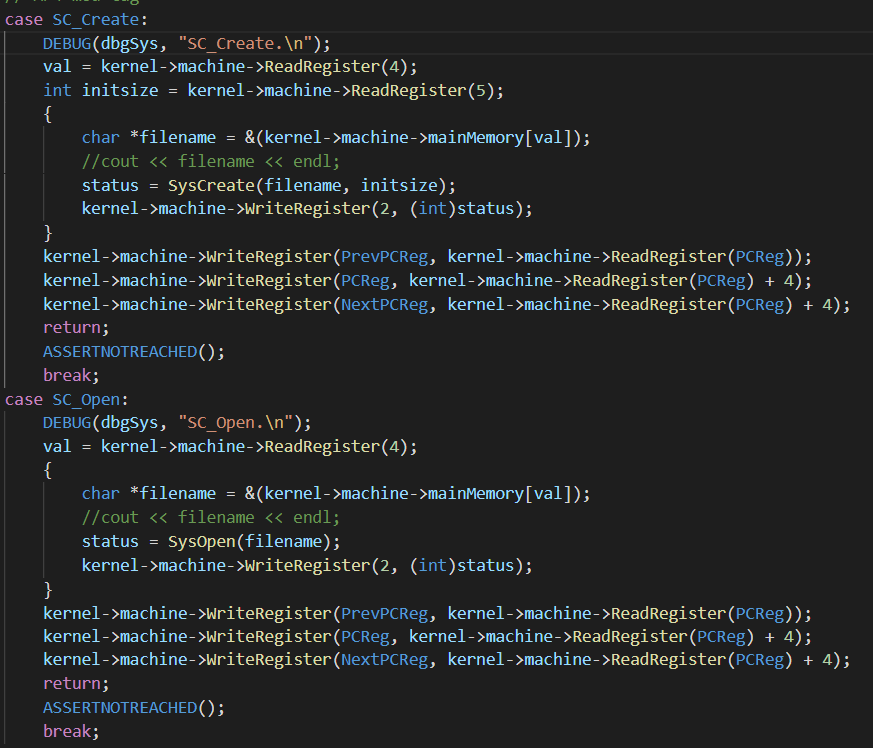
= 3840 (bytes) 4 (KB)

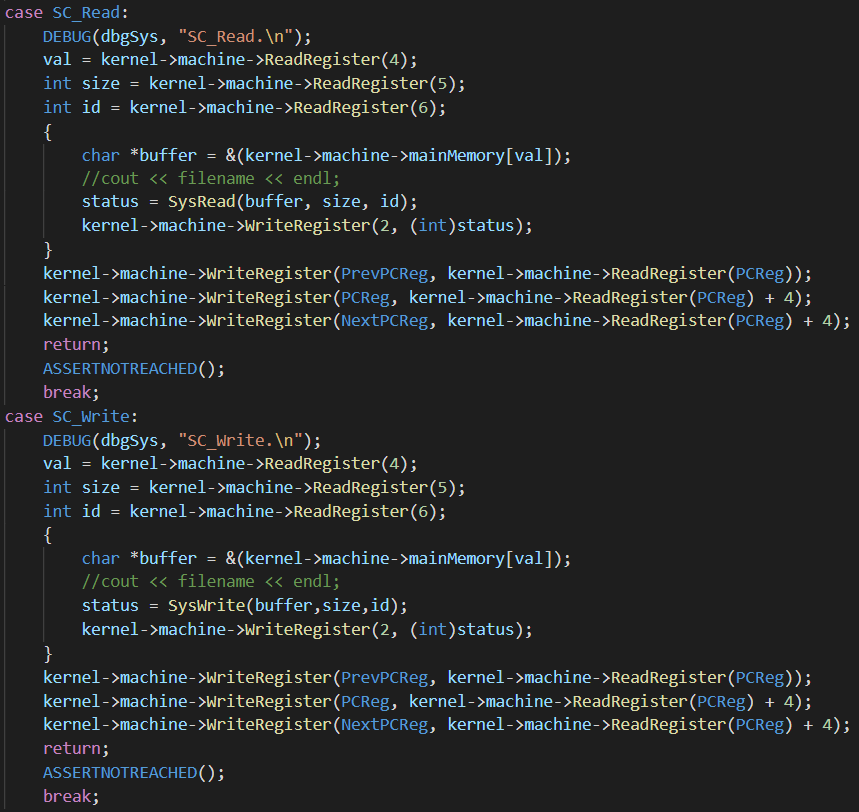
**Part II.** **Modify the file system code to support file I/O system call and larger file size**

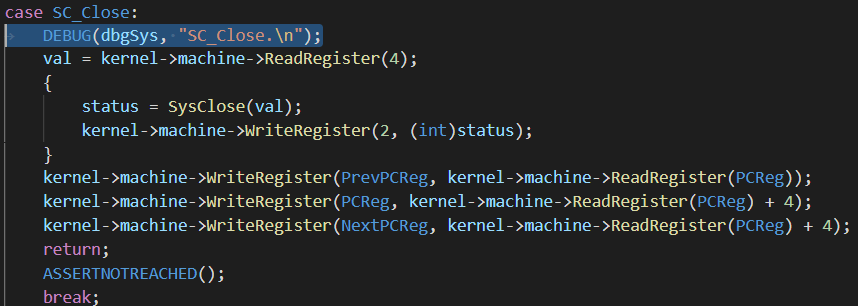
**II.1 Combine MP1 file system call interface with NachOS FS**

exception.cc:

作法與MP1相同，從Register跟Memomy讀取arguments後呼叫ksyscall.h內定義的kernel system call。

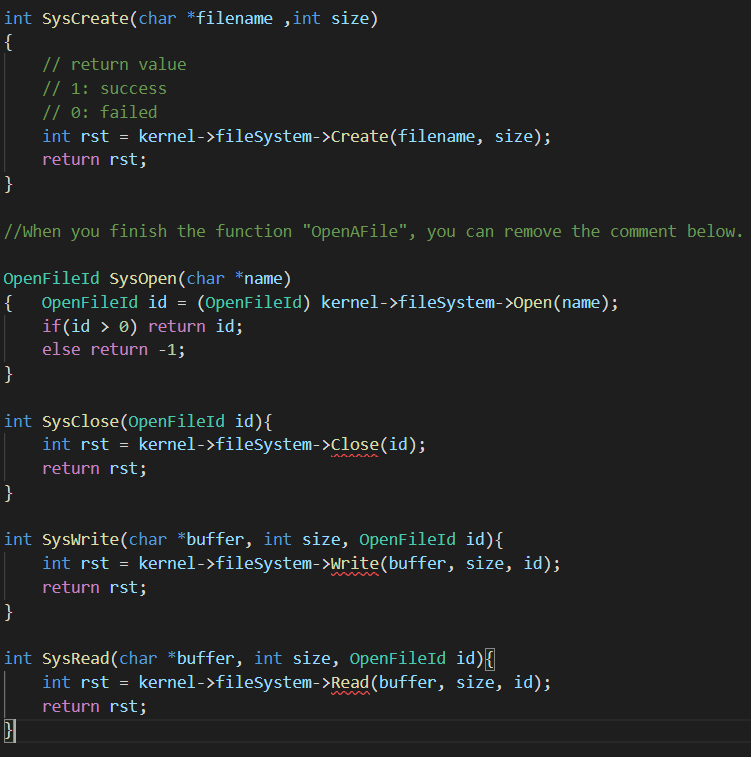


****



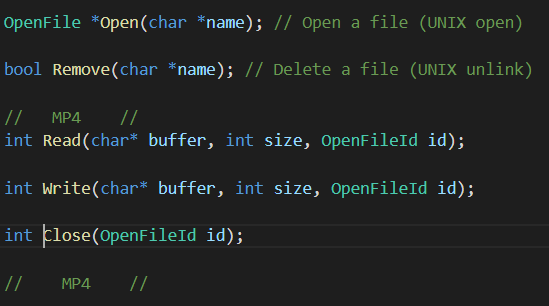
ksyscall.h

定義5個system call的kernel syscall interface，並把實作留到fileSystem。

****

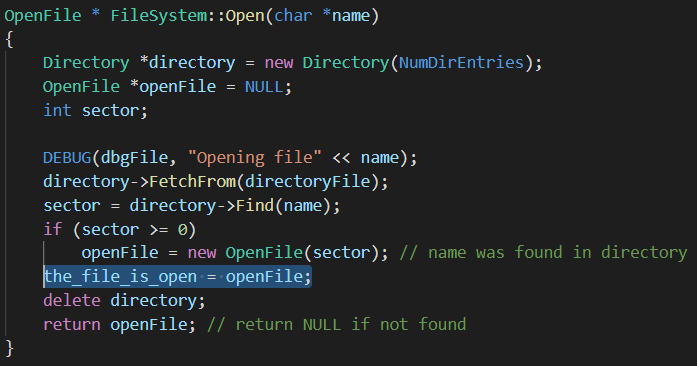
filesys.h:

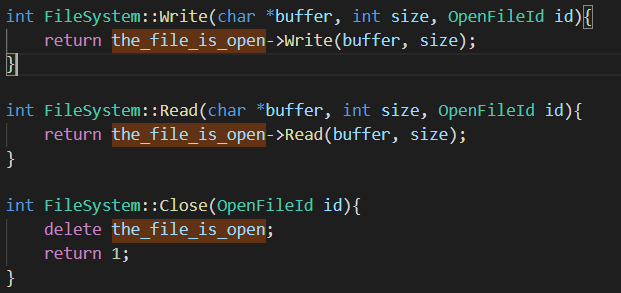
declare fileSystem member function Create, Read, Write, Open, Close。



****filesys.cc:

進行這些member function的實作:



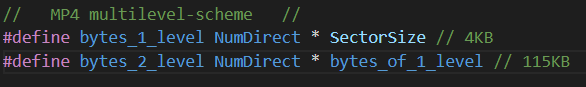


**II.2 Enhance the FS to let it support up to 32KB file size**

使用Multilevel Indexed Allocation來解決: 2-level可以達到115KB。

filehdr.h:

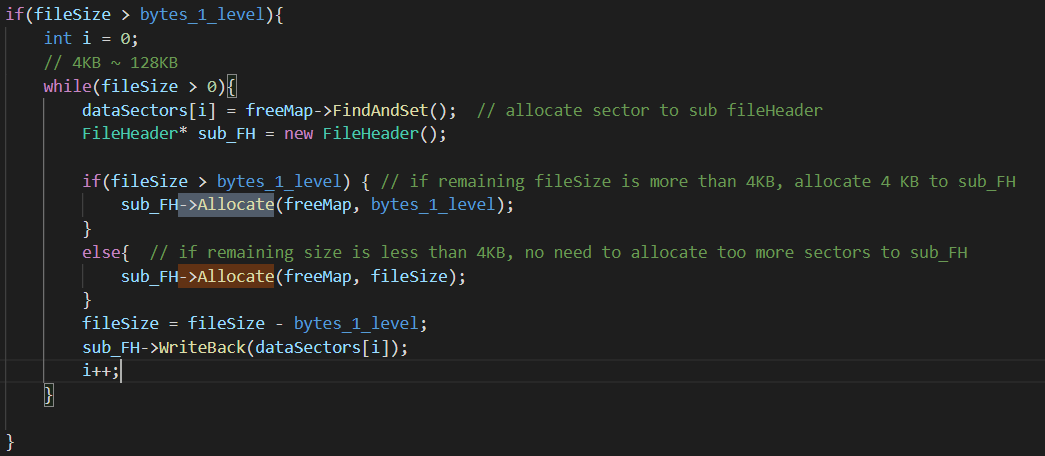
首先define 1-level跟2-level分別可放多少bytes的data。



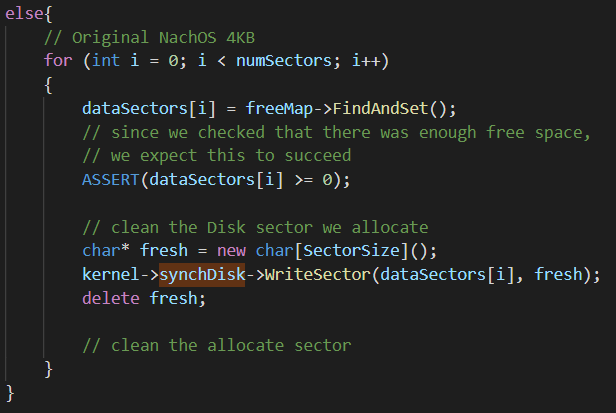
filehdr.cc:

首先改動allocate()使得變成multilevel indexed allocation，如果fileSize > 4KB，則需要allocate多個index block，一開始設i為0，從第一個index block開始allocate，進入while loop，每次iteration，如果fileSize仍>0，代表還要繼續分配index block。

每次iteration中，透過freeMap->FindAndSet()替index block i找到free sector後，new一個FileHeader object sub FileHeader，之後根據目前剩餘的fileSize決定分配多少fileSize的sectors給sub FileHeader，****然後把fileSize減掉4KB，把sub fileHeader寫回index block i，最後看要不要繼續分配。



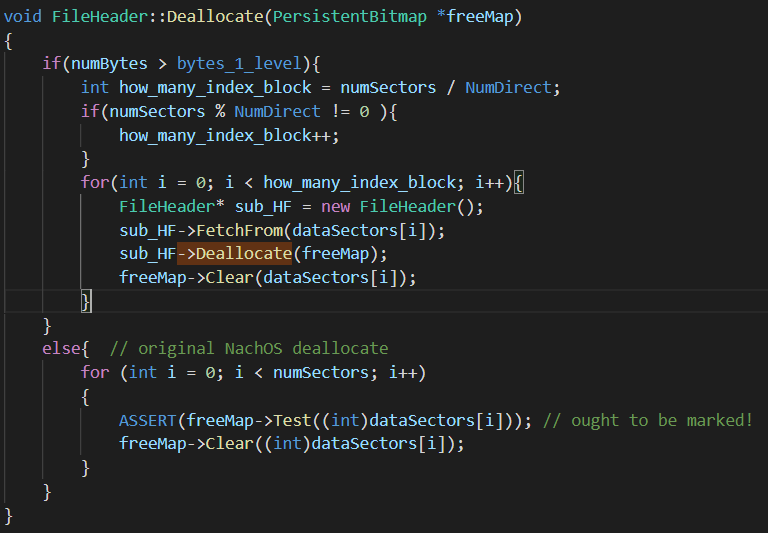
如果現在要分配的fileSize <= 4KB的話，則真正分配data sectors給FileHeader，直接根據NachOS原本的方式來allocate，並把分配到的sectors清空。



接著改動Deallocate()。

如果numBytes＞4KB，就要去把其底下所有index blocks指到的sectors全部deallocate，先算出此sector下有幾個index blocks存到how\_many\_index\_block中，接著不斷fetch index block的內容到FileHeader，並對此FileHeader進行deallocate，最後把此index block deallocate。

如果numBytes<4KB，則照原本NachOS deallocate的方式把所有的data sectors給allocate掉。

****

ByteToSector():

根據目前的numBytes判斷，如果＞4KB，則算出此offset的byte在哪個index sector包含的sectors上，接著透過FileHeader fetch出該index sector的content，接著算出該index sector的起始offset，往下再call一次ByteToSector，這次要找的offset是offset – index sector起始offset。

如果小於4KB，則透過NachOS原本算法即可。

