

NACHOS-4.0

MP4

一張含有 服飾 的圖片

自動產生的描述

第21組

106062173 吳東弦(一起 trace code,負責整理 report)

106062271 郭子豪(一起 TRACE CODE,負責實作 CODING)

# Understanding NachOS file system

　　NachOS的虛擬機以bit vector的方式來管理free block space，主要實作在Bitmap class（bitmap.h / bitmap.cc）以及繼承它的PersistentBitmap class（pbitmap.h / pbitmap.cc）中。

　　在Bitmap class中，Mark()會把傳進來的sector所對應的bit設為1，Clear()會把傳進來的sector所對應的bit設為0，FindAndSet()會根據bitmap找到free sectors並將這些sectors所對應的bit設為1。

　　而PersistentBitmap class則是NachOS真正實例化的bitmap，除了有Bitmap class的功能外，還能對虛擬機的disk進行讀寫，FetchFrom()會把disk上的 data block讀進memory中，WriteBack()則是把data從memory寫回disk上。

　　在disk上的file有file control block和data block，在NachOS中FCB即為FileHeader，讀檔時會把FileHeader載入記憶體中，再據此找到data block，而free block space實際上也是以file的形式存在disk上（freeMapFile）。

　　當FileSystem（filesys.h / filesys.cc）在NachOS中被new出來時，如果disk沒有要被format，就直接依照free block space之FileHeader所在的sector（FreeMapSector）載入記憶體中，保持open的狀態。

　　如果disk被format，則要重新建立free block space，先在記憶體中new出其FileHeader（mapHdr）以及對應的data（freeMap），再將指定要放mapHdr的sector標記起成「使用中（Mark()）」，並分配sector給freeMap，最後將兩者皆寫到disk上（WriteBack()），同時依照free block space之FileHeader所在的sector（FreeMapSector）載入記憶體中，保持open的狀態。

　　由此可知，free block space的FileHeader放在FreeMapSector中，也就是sector 0，而由於sector 1放directory的FCB，故free block space的data block（freeMapFile）放在sector 2。

　　在disk.h中可知，此disk有32個track，每個track有32個sector，共1024個sector，每個sector為128 Byte，故此disk共128 KB。

　　此外1024個sector對應freeMapFile為1024 bits，即128 Byte，由此可知freeMapFile恰放滿一個sector，解釋了上題freeMap為什麼只需要sector 2。

　　directory主要實作在Directory class和DirectoryEntry class中（directory.h / directory.cc）。Directory class為一張表（table），一格為一個DirectoryEntry，代表可放一個file，紀錄該格是否已被使用（inUse），如果已被佔用，該檔案的FileHeader放在哪個sector，檔名（name）為何。

　　在Directory class中，可以新增（Add()）、移除（Remove()）、表列（List()）檔案，以及找到檔案的FileHeader後（Find()），再將其讀進memory中（FetchFrom()），或是寫回disk上（WriteBack()）。

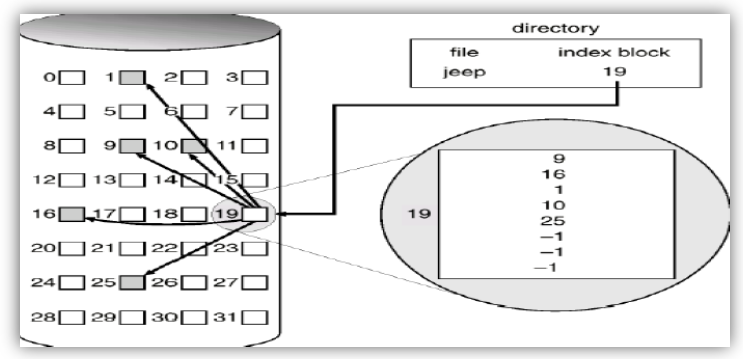
　　directory和free block space同樣也是以file的形式存在disk上。當FileSystem（filesys.h / filesys.cc）在NachOS中被new出來時，如果disk沒有要被format，就直接依照directory之FileHeader所在的sector（DirectorySector）載入記憶體中，保持open的狀態。

　　如果disk被format，則要重新建立directory，先在記憶體中new出其FileHeader（dirHdr）以及對應的data（directory），再將指定要放dirHdr的sector標記起成「使用中（Mark()）」，並分配sector給directory，最後將兩者皆寫到disk上（WriteBack()），同時依照directory之FileHeader所在的sector（DirectorySector）載入記憶體中，保持open的狀態。

　　由此可知，directory的FileHeader放在DirectorySector中，也就是sector 1，而由於sector 2放free block space的data block（freeMapFile），故directory的data block（directoryFile）放在sector 3和sector 4。

　 在NachOS中，inode就是FileHeader，存放檔案大小（numBytes）、檔案用了多少sector（numSectors），以及指向data block每個sector的entry（dataSectors[]）。

　　由於dataSectors[] 為一個陣列，由此可知NachOS的檔案系統是以Indexed Allocation來存放檔案的。



　　在FileHeader所在的sector中，以4個Byte（sizeof (int)）為一個單位，前4個Byte放file大小（numBytes），次4個Byte放file的data block使用了幾個sector（numSectors），剩下120個byte即30個entry（NumDirect），最多對應30個sector，因此一個file的data block最多為3.75 KB，大約為4 KB。

　　而由filesys.cc中可知，NachOS（唯一）的directory共10個entry，即可放10個檔案，共37.5 KB，表示目前的檔案系統只能映射37.5 KB。

　　此外，directory之entry的資料（DirectoryEntry），佔用了20 Byte （inUse + sector + name[]），10個entry共200 Byle，會超過一個sector的大小（128 Byte），解釋了第3題directoryFile為什麼需要sector 3和sector 4。

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

　　I/O system call呼叫的流程大致和MP1相同，Create的部分多傳了一個表示檔案大小（size）的參數，並讀入暫存器$5。

　　呼叫流程以file create為例，case SC\_Create（exception.\*）呼叫SysCreate()（ksyscall.h），SysCreate()呼叫kernelinterruptCreateFile()（interrupt.\*），CreateFile()呼叫kernelCreatFile()（kernel.\*），CreateFile()呼叫fileSystemCreate()（filesys.\*）。

　　因此，實作部分主要放在filesys.c的FileSystem class中，各個API也與MP1大致相同，一樣多新增一張表（fileDescriptorTable[]）方便記錄開啟過的檔案。

　　為了data block不增加額外的pointer，以及使FileHeader本身保持洽使用一個sector，以two-level Indexed Allocation來實作，主要實作在filehdr.cc中。

　　第一層index table （LevelOneTable[]）有30個entry（NumDirect），可對應到30張第二層index table，每張表有32個entry，實作上將其併成一張表（dataSectors[]），共對應960個sector（NumDirect \* NumberofSecondEntry），即最大可支援單檔120KB。

　　分別修改了Allocate()、Deallocate()、FetchFrom()、WriteBack()。

# Modify the file system code to support subdirectory

　　在directory.\*中，為了要recursively trace，需判斷當前資料夾的entry（DirectoryEntry）為file還是directory，因此新增了一個變數isDir，同時改寫List() 成能夠做recursively list。

　　在filesys.\*中，由於資料夾變成多層結構，多實做了Find\_My\_Directory()來判斷是否要繼續往下trace，並回傳當前資料夾的FileHeader。而原本的函數中，多傳了isDir判斷要創建的是資料夾還是檔案，並利用Find\_My\_Directory()找到當前資料夾再進行操作，另外List()多判斷是否要進行遞歸處理。

　　此外，在create directory時，在directory.\*多實做了一個Clean()，先將disk上指定的位置清零，再從記憶體寫回去（WriteBack()），避免因為disk的殘留值，導致在判斷DirectoryEntry的inUse時發生錯誤。

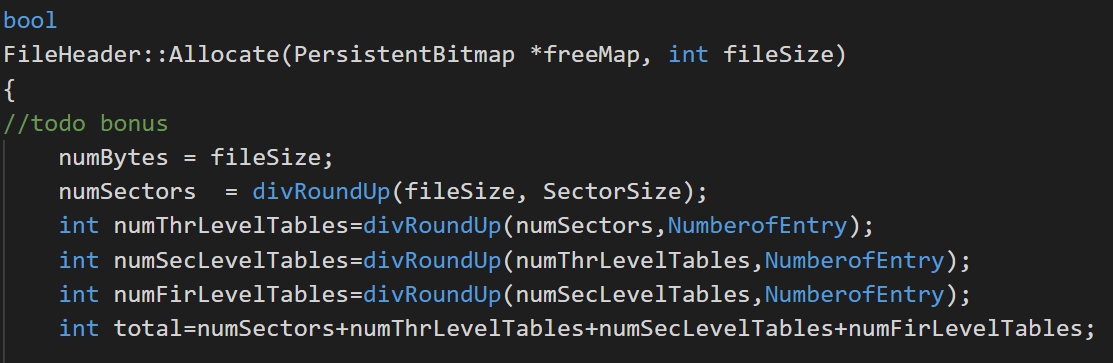
　　此題直接將filesys.h的 #define NumDirEntries改為64即可。

### Bonus

一張含有 文字 的圖片

自動產生的描述

* 先將disk變大成16384\*32\*128 = 67108864 = 64MB
* 再將fileheader 的 hierarchy變成四層使其可記錄30\*32\*32\*32\*128 = 125829120 = 120 MB 的檔案 只要將disk變大就可以紀錄120MB的檔案



* 因為我們 fileheader使用hieracy的結構紀錄其所占的sector
* 故將numFir + numSec + numThr + 1就會是總header需要的大小，而這個數字由上圖可知是由numsectors決定的，也就是檔案大小決定了header的大小，因為numsectors是變數，故超過三種以上

一張含有 螢幕擷取畫面, 文字 的圖片

自動產生的描述

* 這是多加在remove的部分，先判斷是否為資料夾（所以我們在dirctory新增了isDir(char\* name)來判斷要刪除的檔案是否為資料夾），接者如果是資料夾的話，我們找資料裡inUse的部份遞迴進去，就可以將這個資料夾內所有的資料刪除了。

# Summary & FeedBack

　　本次實作2-1大致與MP1相同；2-2由於採用Indexed Allocation所以bonus單檔64 MB不好實作；3-1為這次實作重點，概念並不難，但是很容易出bug，要不斷觀察disk的資訊。

　　其一是創建檔案與創建資料夾是判斷不一樣的東西，前者看free block space，後者看DirectoryEntry的inUse，要把directory在disk部分先清空再寫入。

　　其二是disk的format並不會真正把disk清零，導致雖然做了格式化，卻還有殘留值，所以得把整個disk刪掉再重建。