Operating System

MP4 : File System

Report

Team 20 member :

105060011 蔡悅承

105060016 謝承儒

Team member contribution :

Implement : 蔡悅承 謝承儒

Report : 蔡悅承 謝承儒

Part I.
Understanding NachOS file system

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

使用bitmap來記住sector的使用狀況，bitmap存在sector 0。

二. What is the maximum disk size can be handled by the current implementation? Explain why.

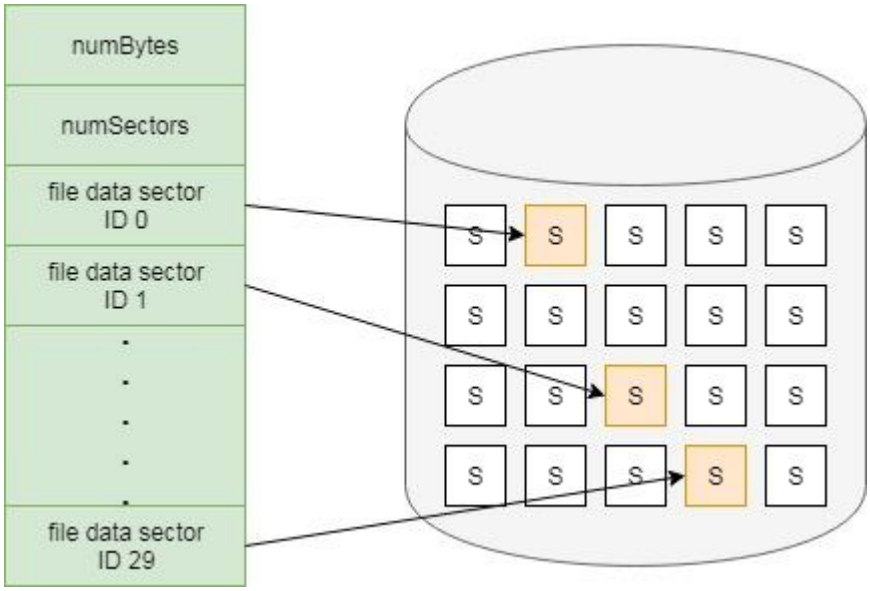
128KB = 32 * 32 * 128B

```
const int SectorSize = 128;   // number of bytes per disk sector
const int SectorsPerTrack  = 32;  // number of sectors per disk track
const int NumTracks = 32;   // number of tracks per disk
```

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

用一個file來儲存directory structure，存在sector 1。

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



allocation scheme類似於index，inode是FCB也是index sector，利用sector的ID直接找到data sector。

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

FCB也是用sector存，一個sector可以存32個int(128/4)，所以能記得32個int，扣除numBytes、numSectors，還可以存30個ID

=>30 * 128B = 3.75KB

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

一.Combine your MP1 file system call interface with NachOS FS

只須考慮開起一個file的情況=>不用maintain table

1. 修改部分

(1) ksyscall.h

新增相對應的System call function，大都直接用fileSystem裡的method。

a. int SysCreate(char *name, int size)
   Call fileSystem->Create(name, size)。

b. OpenFlieId SysOpen(char *filename)
   Call fileSystem->Open(filename)，若是null則回傳-1，反之有找到東西就回傳1。

c. int SysClose(int id)
   Call fileSystem->CloseFile(id)

d. int SysWrite(char *buffer, int size, int id)
   Call fileSystem->WriteFile(buffer, size, id)

e. int SysRead(char *buffer, int size, int id)
   Call fileSystem->ReadFile(buffer, size, id)

(2) filesys.h

基本上Call已經寫好的method

a. OpenFile* currentfile
   因為這次只需要考慮file只有一個的情形，就用該變數記住現在open哪個檔案。
   在Open時改變其值。

b. int CloseFile(int id)
   delete currentfile並再次currentfile=NULL，再回傳1

c. int WriteFile(char *buffer, int size, int id)
   Call currentfile->Write(buffer, size)

d. int ReadFile(char *buffer, int size, int id)
   Call currentfile->Read(buffer, size);

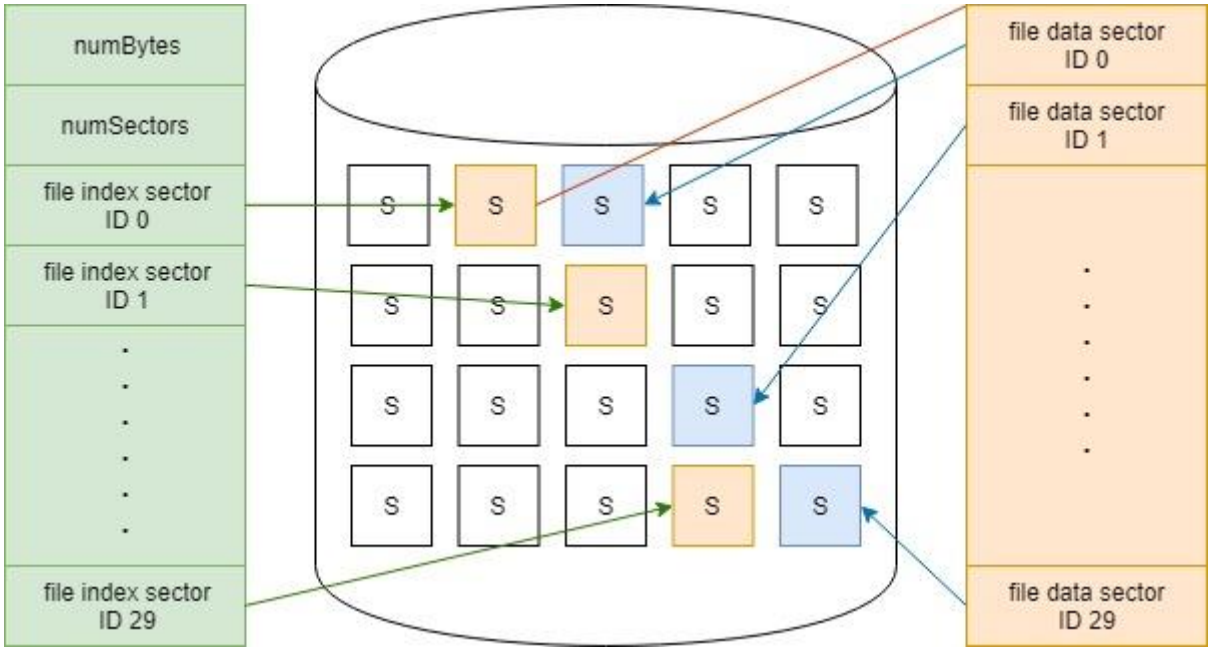
二. Enhance the FS to let it support up to 32KB file size

1. 修改想法

將本來直接指向data sector的作法，改為指向index sector，再從index sector中得到data sector的ID，就能得到data。

如此只需要8個index sector就能得32KB(=8*32*128B)的空間。

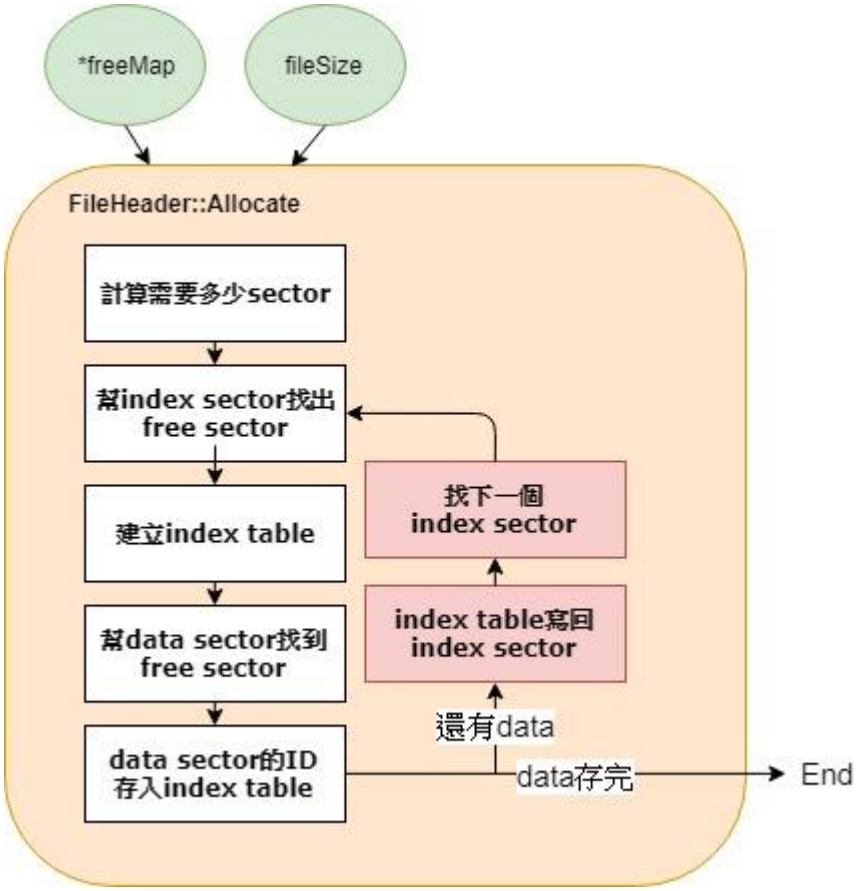
註. 把datasector[]當作index sector而不重新改名，避免其他有使用該變數的code segment出現bug。



2. 修改內容

(1) Allocate

a. 流程圖



b. 步驟

(a) 計算需要多少sector

$$\text{numSectors} = \frac{\text{fileSize}}{\text{sectorSize}}$$

(b) 從freeMap中找到free sector當作index sector。

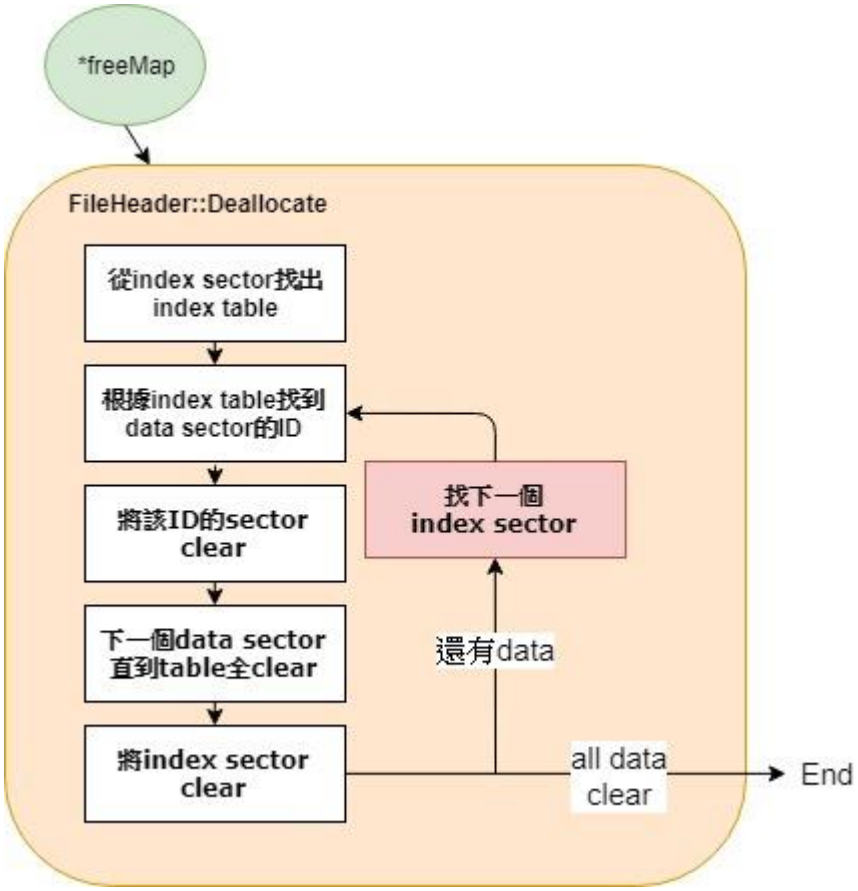
(c) 建立index table(index[32])

(d) 從freeMap中找到free sector當作data sector，並把該ID存進index[]

(e) 寫滿table後，將table寫回index sector。若是還有data，就回到步驟(b)；反之，就結束運行。

(2) Deallocate

a. 流程圖



b. 步驟

(a) 從index sector讀出index table。

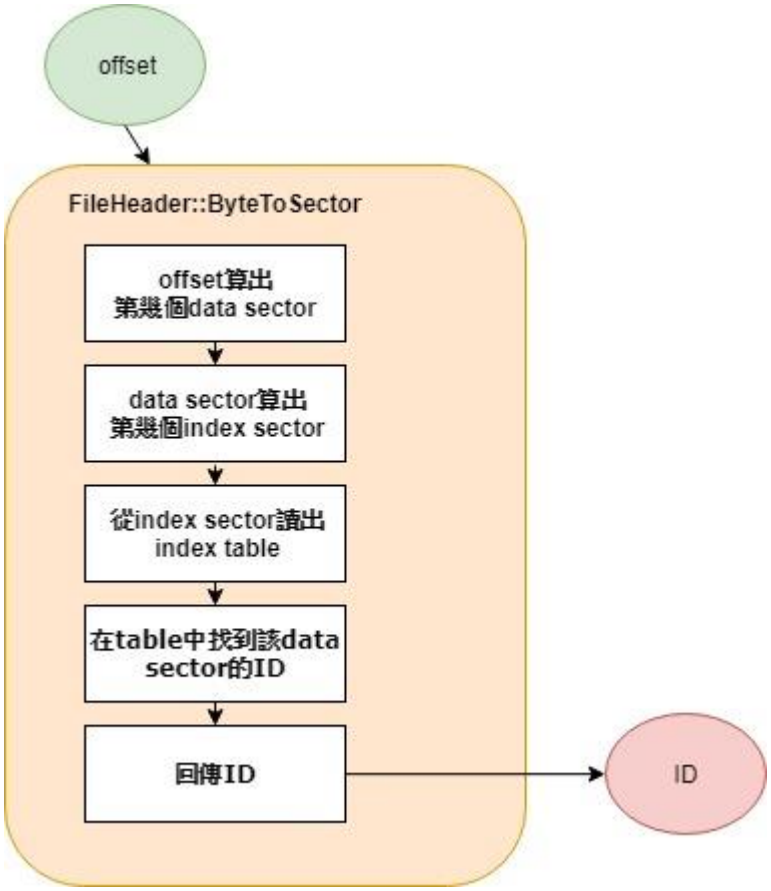
(b) 根據index table裡的data sector ID，將data sector一一清除掉。

(c) table clear完後，將index sector clear。

(d) 換下一個index sector重複步驟(a)~(c)，直到全部的index sector做完。

(3) ByteToSector

a. 流程圖



b. 步驟

(a) 由offset算出在哪個sector。

$$\text{whichDataSector} = \frac{\text{offset}}{\text{sectorSize}}$$

(b) 由whichDataSector算出在哪個index sector

$$\text{whichIndexSector} = \frac{\text{whichDataSector}}{32}$$

(c) 算出會在whichIndexSector的哪個位置

$$\text{offsetSector} = \text{whichDataSector} \ \% \ 32$$

(d) 讀出index table，並從第offsetSector個得到ID。

Part III. Modify the file system code to support subdirectory

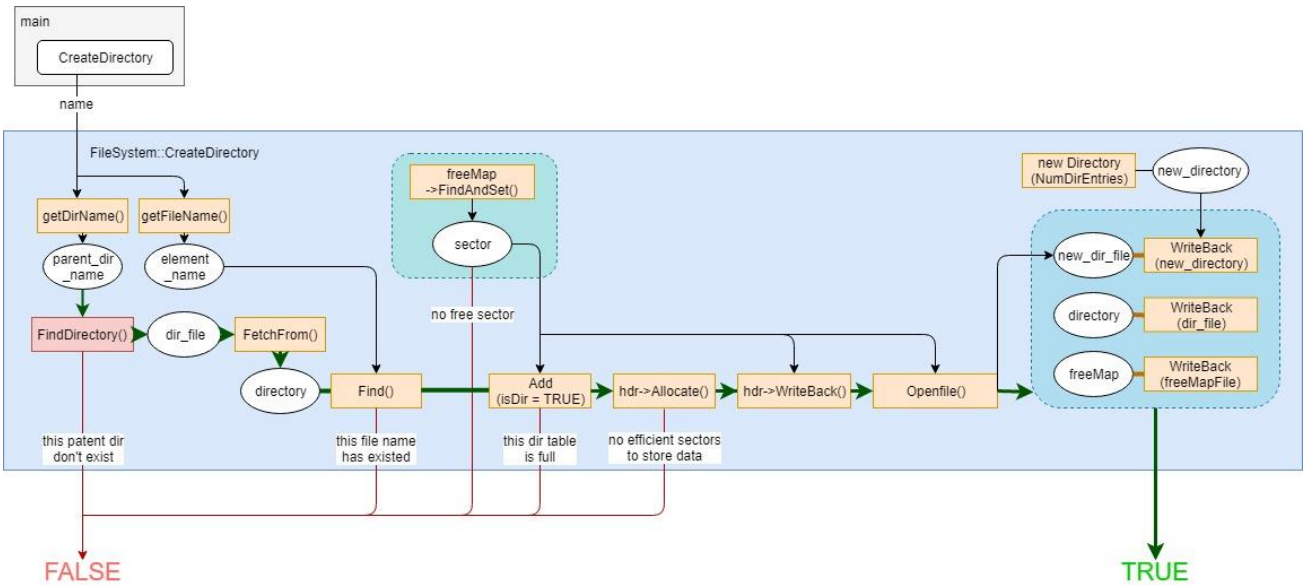
一.Implement the subdirectory structure

1.修改內容

(1) CreateDirectory

除此之外，Remove()、Open()、給file用的Create()也都有更改，但思路是相同的，故只說明該method。

a. 流程圖



b. 步驟

(a) 將傳進來的name分成parent_dir_name和element_name
parent_dir_name : 母目錄
element_name : 想創建的子目錄名稱

(b) FindDiretory(parent_dir_name)來判斷該母目錄是否存在，若存在就Fetch到directory

(c) 在directory中找element_name是否存在，不存在才能創建

(d) 找free sector給新目錄 => sector

(e) directory->Add(element_name, sector, TRUE)
將新目錄加進舊目錄的table

(f) 創建FCB(hdr)並Allocate給予新目錄，並把FCB寫進sector

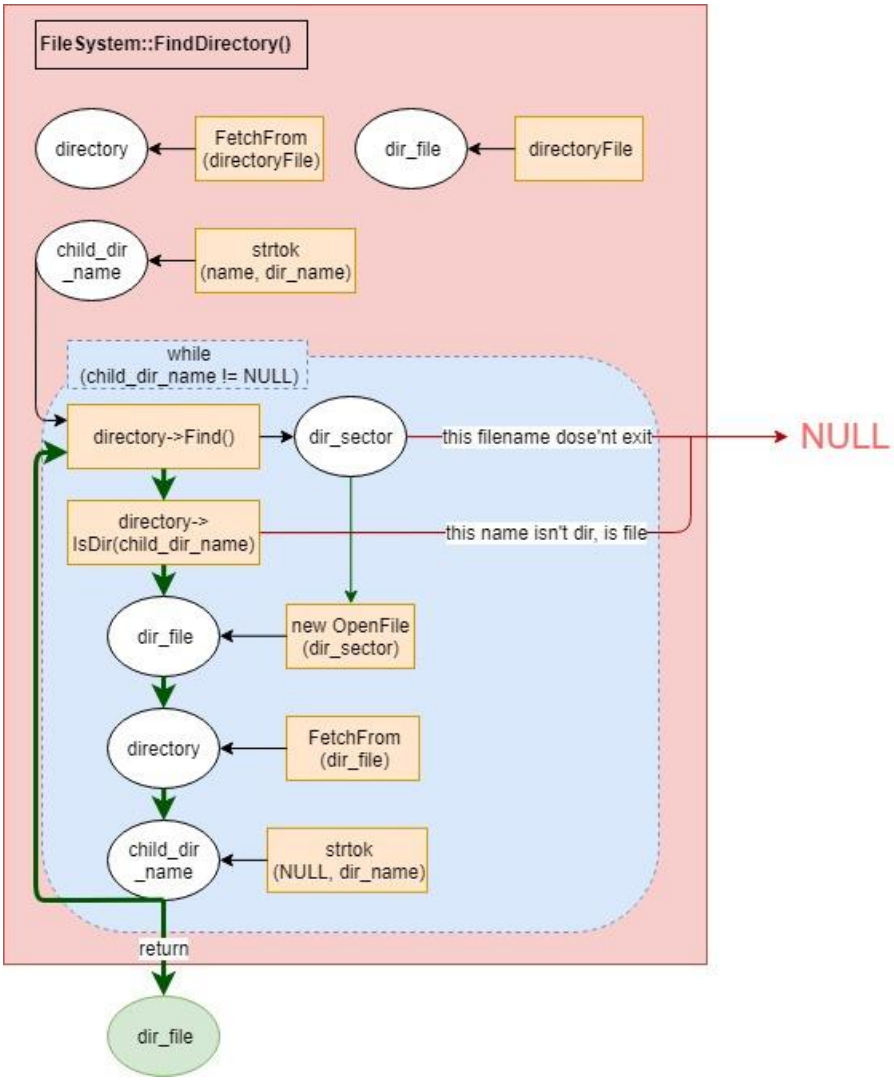
(g) 創建一個全新的Directory架構 => new_directory

(h) 根據FCB找出新目錄的data sector，並把new_directory寫進這些data sector來reset

(i) 把新增子目錄的母目錄寫回disk，以及更新freeMap

(2) FindDirectory

a. 流程圖



註. dir_name = “/”

b. 步驟

(a) 先將dir_file預設成root directory，並fetch到directory

(b) 使用strtok(name,dir_name)，從name切出下一個目錄的名字

Ex. /abc/ss/dfa => child_dir_name = abc

/ => child_dir_name = NULL(已經到最底)

(c) directory->Find(child_dir_name)，看是否存在child_dir_name為名的東西=>得到dir_sector(FCB所在)

(d) 判斷該東西是file還是directory

(e) 從dir_sector中讀出data block=>得到dir_file

(f) 進到下個資料夾 => 把dir_file fetch到directory

(g) 使用strtok(NULL,dir_name)，從child_dir_name切出下一個目錄的名字，回到步驟(c)

(3) List

a. Code segment

```
for (int i = 0; i < tableSize; i++) {
    printf("%s\n", table[i].name);
    if (table[i].isDir) {
       childDir_file = new OpenFile(table[i].sector);
       childDirectory->FetchFrom(childDir_file);
       childDirectory->RecursiveList(indent+1);
    }
}
```

依照table一個個print出來，但若該entry是directory，就再次call RecursiveList(indent+1)

註. RecursiveList(int indent)，indent是要印多少”-”(排版用)

二.Support up to 64 files/subdirectories per directory

```
#define NumDirEntries       64
```