# MP4: File System

Team 02


**Team member list**

106030009 葉蓁

106010010 劉艾薇

**Contributions**

共同 trace code、implement、寫 report

# 1. Understanding NachOS file system

## Trace the file system call and answer the following questions:

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

首先我們先來看 FS 建構的部分

**filesys/filesys.cc**

```
FileSystem::FileSystem(bool format)
{
    if (format)
    {
        PersistentBitmap *freeMap = new
PersistentBitmap(NumSectors);
        Directory *directory = new Directory(NumDirEntries);
        FileHeader *mapHdr = new FileHeader;
        FileHeader *dirHdr = new FileHeader;

        freeMap->Mark(FreeMapSector);
        freeMap->Mark(DirectorySector);
```

當 format 為 true 的時候，就會建立 freeMap 這個 bitmap，用來記錄 block 有沒有被使用，並且也先建立兩個 FileHeader，分別是 mapHdr 與 dirHdr，也就是這個 bitmap 的 header 跟 directory 的 header。接著使用 Mark()來把 FreeMapSector 與 DirectorySector 的位置標註為 1，代表這兩個位置要被使用，這兩個數代表的值有先被定義在上面：

```
#define FreeMapSector       0
#define DirectorySector     1
```

也就是說這個 freeMap 中 sector 0 跟 sector 1 已經被保留下來了。

```
ASSERT(mapHdr->Allocate(freeMap, FreeMapFileSize));
ASSERT(dirHdr->Allocate(freeMap, DirectoryFileSize));
```

接著要去 allocate 空間來做使用。

**filesys/filehdr.cc**

```
FileHeader::Allocate(PersistentBitmap *freeMap, int fileSize)
{
    numBytes = fileSize;
    numSectors  = divRoundUp(fileSize, SectorSize);
    if (freeMap->NumClear() < numSectors)
    return FALSE;       // not enough space

    for (int i = 0; i < numSectors; i++) {
    dataSectors[i] = freeMap->FindAndSet();
    }
    return TRUE;
}
```

首先先看 freeMap 中有沒有足夠的空間可以使用，夠的話，就一個一個用 FindAndSet()來找空間給剛剛那兩個 header 做使用

**lib/bitmap.cc**

```
Bitmap::FindAndSet()
{
    for (int i = 0; i < numBits; i++) {
    if (!Test(i)) {
        Mark(i);
        return i;
    }
    }
    return -1;
}
```

FindAndSet()就是從 freeMap 的頭開始看，看看有一個 bit 是沒有被使用的，有的話就把那個空間標註為被使用，並且把找到的那個 bit 回傳放到 dataSectors 的 array 中。基本上就是一一尋找空的 blocks，並且標註為被使用。這就是 FS 如何去管理並找 free block space 的過程！

然後我們就可以回到剛剛的 filesys/filesys.cc 繼續往下看

```
mapHdr->WriteBack(FreeMapSector);
dirHdr->WriteBack(DirectorySector);
```

接著要把剛剛定義的 hearders 寫回 disk 中，這樣之後要 open file 的時候才可以透過 header 找到並打開那個 file。這裡就可以回答第二部分的問題了！管理 free block space 的 freeMap 的 header 是放在 sector 0 裡面(剛剛上面有提過 FreeMapSector 為 0)！這樣在 bootup 的時候會比較容易被找到！

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

disk.cc 中對於 DiskSize 的定義如下

```
const int DiskSize = (MagicSize + (NumSectors * SectorSize));
```

其中：

MagicSize = 4 bytes

**machine/disk.cc**

```
const int MagicSize = sizeof(int);
```

NumSectors = 32*32 = 1024

**machine/disk.h**

```
const int NumSectors = (SectorsPerTrack * NumTracks);
const int SectorsPerTrack  = 32;
const int NumTracks = 32;
```

SectorSize = 128

**machine/disk.h**

```
const int SectorSize = 128;
```

=> DiskSize = ( 4 + ( 1024 * 128 ) =  131076 (bytes)
而另一種算法是把 magicSize 拿掉，因為她所占用的空間並不能拿來儲存資料，因此 DiskSize = 131072 bytes = 128 KB

**(3) Explain how the NachOS FS manage the directory data structure? Where is this information stored on the raw disk (which sector)?**
首先先回答第二部份的問題，因為這個在第一題已經帶到過了，所以我們只擷取部分的 code 解釋：

**filesys/filesys.cc**

```
FileHeader *dirHdr = new FileHeader;
freeMap->Mark(DirectorySector);
ASSERT(dirHdr->Allocate(freeMap, DirectoryFileSize));
dirHdr->WriteBack(DirectorySector);
```

先定義 directory 的 header，並且在 freeMap 上標註 DirectorySector 上為被使用的，並且分配空間給這個 header，在把它寫回 disk 上 DirectorySector 的位子，根據定義：

```
#define DirectorySector     1
```

因此管理 directory data strucuture 的 information 就在 sector 1 的地方！一樣是為了在 bootup 的時候比較容易被找到。

接著來解釋 FS 要怎麼管理 directory data strucuture，因此我們來看 Directory 建構的部分：

**filesys/directory.cc**

```
Directory::Directory(int size)
{
    table = new DirectoryEntry[size];
    memset(table, 0, sizeof(DirectoryEntry) * size);
    tableSize = size;
    for (int i = 0; i < tableSize; i++)
        table[i].inUse = FALSE;
}
```

在建構的時候，會建立一個紀錄 DirectoryEntry 的 table，紀錄的內容可以看下面一段 code。

**filesys/directory.h**

```
class DirectoryEntry
{
public:
    bool inUse;
    int sector;
    char name[FileNameMaxLen + 1];
};
```

inUse 紀錄各個 entry 是不是有被使用了，sector 是用來看這個 entry 紀錄的 file 的 FileHeader 在 disk 上的哪個 sector 中，最後 name[]是紀錄這個 file 的名稱。綜上所述，可以看出來 NachOS 採用的是 single indirect directory。

後面每次在操作 Create, Open file 等動作時，都會先去 directory 的 table 找，利用 directory->Find(filename)來確認目前的 directory 有沒有這個檔案了!

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

inode 就是 UNIX 的 file control block，在這裡也就是 file header，於是我們來看 FileHeader 的 class 裡面 private 部分：

**filesys/filehdr.h**

```
int numBytes;
int numSectors;
int dataSectors[NumDirect];
```

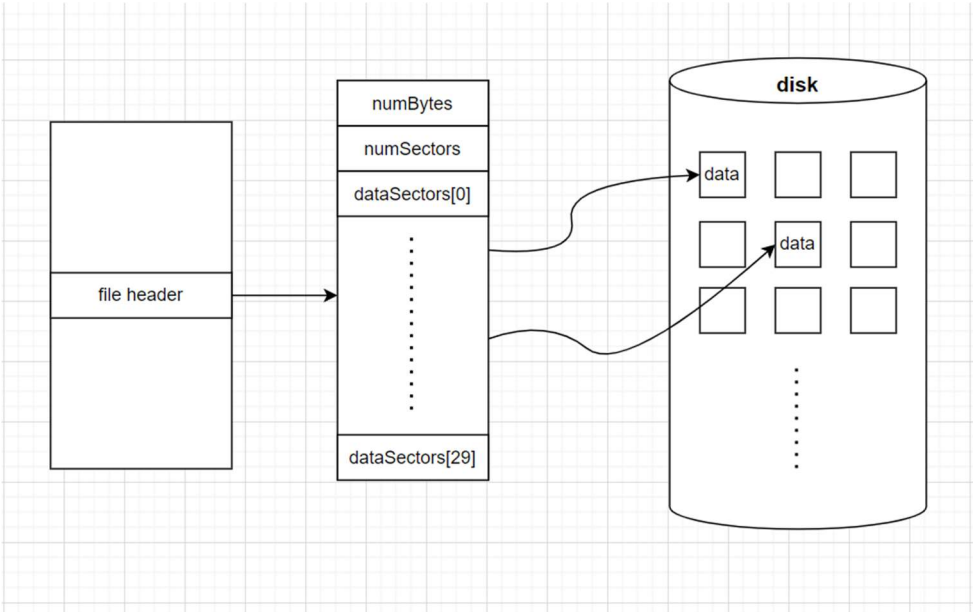
可以看到在一個 inode 中存有的資料有：

numBytes：一個 file 中有多少個 bytes

numSectors：一個 file 有多少個 data sector

dataSectors[]：紀錄 file 中每個 data block 的 disk sector numbers

而 NachOs 原本是使用 Indexed Allocation 的方式，並且他的 index block 是採用 single indirect，詳細圖示如下所示：




因為 FileHeader 本身剛好占一個 sector (128bytes)，扣掉 metadata numBytes, numSectors 各 4bytes 後 (128 - 4*2 = 120) 剩下 120 bytes, 一個 sector number 是 int 也需要 4bytes，所以真正存 dataSector number 的位子有 30 格! 一個 File Header 可以管理 30 個 data Sectors!(存 logical data blocks mapping 到 physical disk sector 的 number)

**(5) Why is a file limited to 4KB in the current implementation?**
Currently, nachOS uses single-indirect indexed allocation, which means an entry in the directory points to the file's header, and each file header(takes a sector) points to 30 data Sectors. So each file is limited to 30 (datablocks)
**Since nachOS assign one sector per block, 30 blocks takes:**
30 * 128 (bytes per sector) = 3840 = approximately = 4KB

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

### Part II - a

**System call:**
**userprog /ksycall.h**

```
29   /* ------------ MP4 ----------*/
30
31   int SysCreate(char *filename, int initialSize)
32   {
33       // return value
34       // 1: success
35       // 0: failed
36       return kernel->fileSystem->Create(filename, initialSize); //must cre
37   }
38
39   /* Open the Nachos file "name", and return an "OpenFileId" that can
40    * be used to read and write to the file.
41    */
42   OpenFileId SysOpen(char *name){
43       OpenFileId ofid = (OpenFileId) kernel->fileSystem->Open(name);
44       if(ofid>0) return ofid;
45       else return -1; // fail.
46       // return kernel->fileSystem->Open(name);
47   }
```

```
49   /* Write "size" bytes from "buffer" to the open file. ···
53   int SysWrite(char *buffer, int size, OpenFileId id){
54
55       return kernel->fileSystem->Write(buffer, size, id);
56   }
57
58   /* Read "size" bytes from the open file into "buffer".   ···
64   int SysRead(char *buffer, int size, OpenFileId id){
65       return kernel->fileSystem->Read(buffer, size, id);
66
67   }
68
69   /* Close the file, we're done reading and writing to it.
70    * Return 1 on success, negative error code on failure
71    */
72   int SysClose(OpenFileId id){
73       return kernel->fileSystem->Close(id);
74
75   }
76
```

**userprog / exception.cc**

作法如 MP1 做法：根據 MIPS convention 從 cpu register 和 memory 拿取正確的 argument，呼叫 kernel 的 system call implementation (接到 filesystem implementation)，結束後寫回 return value 並且 Advance Program Counter。

以 SC_Create 例：

```
// MP4 mod tag
/* -------------- MP4 --------------*/
    case SC_Create: // filename, initialSize
        DEBUG(dbgSys, "System call: SysCreate()\n");
        val = kernel->machine->ReadRegister(4); // address of "filename"

        filename = &(kernel->machine->mainMemory[val]);

        initialSize = kernel->machine->ReadRegister(5);
        status = SysCreate(filename, initialSize);
        kernel->machine->WriteRegister(2, (int)status);
        /* Modify return point */
        {
            /* set previous programm counter (debugging only)*/
            kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

            /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
            kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

            /* set next programm counter for brach execution */
            kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
        }
        cout << "Create file result is " << status << "\n";
        return;

        cout << "in exception\n";
        ASSERTNOTREACHED();
    break;
```

**SC_Open, SC_Read, SC_Write, SC_Close 都一樣，差別在呼叫不同的 ksyscall.h 中定義的 function。**

```
case SC_Open: // filename
    DEBUG(dbgSys, "System call: SysOpen()\n");
    val = kernel->machine->ReadRegister(4); // address of "filename"

    filename = &(kernel->machine->mainMemory[val]);

    status = SysOpen(filename);

    kernel->machine->WriteRegister(2, (int)status);
    /* Modify return point */
```

```
case SC_Read: // char *buffer, int size, OpenFileId id
    DEBUG(dbgSys, "System call: SysRead()\n");
    val = kernel->machine->ReadRegister(4); // address of "buffer_name"
    buffer_name = &(kernel->machine->mainMemory[val]);
    size = kernel->machine->ReadRegister(5);
    id = kernel->machine->ReadRegister(6);

    status = SysRead(buffer_name, size, id);

    kernel->machine->WriteRegister(2, (int)status);
    /* Modify return point */
```

```
case SC_Write: // char *buffer, int size, OpenFileId id
    DEBUG(dbgSys, "System call: SysWrite()\n");
    val = kernel->machine->ReadRegister(4); // address of "buffer_name"
    buffer_name = &(kernel->machine->mainMemory[val]);
    size = kernel->machine->ReadRegister(5);
    id = kernel->machine->ReadRegister(6);
    status = SysWrite(buffer_name, size, id);
    kernel->machine->WriteRegister(2, (int)status);
    /* Modify return point */
    {
        /* set previous programm counter (debugging only)*/
```

```
case SC_Close: // OpenFileId id.
    DEBUG(dbgSys, "System call: SysClose().\n");
    val = kernel->machine->ReadRegister(4); // OpenFileId id
    status = SysClose(val);
    kernel->machine->WriteRegister(2, (int)status);
    /* Modify return point */
```

**filesys / filesys.h**

(Part III 一併說明)

**filesys / filesys.cc**

```
/* ----------- MP4 ----------------- */

int FileSystem::Write(char *buffer, int size, OpenFileId id){
    return opfile->Write(buffer, size);
}

int FileSystem::Read(char *buffer, int size, OpenFileId id){
    return opfile->Read(buffer,size);
}

int FileSystem::Close(OpenFileId id){
    delete opfile;
    return 1;
}

/* --------------------------------- */
```

**threads/ main.cc :**

**“-p”, “-cp”, “-l”**

```
//    Filesystem-related flags:
//    -f forces the Nachos disk to be formatted
//    -cp copies a file from UNIX to Nachos
//    -p prints a Nachos file to stdout
//    -r removes a Nachos file from the file system
//    -l lists the contents of the Nachos directory
//    -D prints the contents of the entire file system
//
//  Note: the file system flags are not used if the stub filesystem
//        is being used
```

main 要改的地方是：RecursiveRemove 跟 Recursive List 的呼叫。還有 CreateDir 的實作。

```
339 #ifndef FILESYS_STUB
340     if (removeFileName != NULL)
341     {   /* ------- MP4 ---------- */
342         if(recursiveRemoveFlag) kernel->fileSystem->Remove(removeFileName, TRUE);
343         else kernel->fileSystem->Remove(removeFileName, FALSE);
344     }
345     if (copyUnixFileName != NULL && copyNachosFileName != NULL)
346     {
347         Copy(copyUnixFileName, copyNachosFileName);
```

```
353     if (dirListFlag)
354     {
355         /* ---------- MP4 ------------*/
356
357         if (recursiveListFlag){
358             kernel->fileSystem->List(listDirectoryName , TRUE);
359             // kernel->fileSystem->List(listDirectoryName , TRUE);  //TRUE means recursive!
360         }
361         else kernel->fileSystem->List(listDirectoryName , FALSE);
362         // kernel->fileSystem->List();
363     }
364     if (mkdirFlag)
365     {
366         // kernel->fileSystem->CreateDirectory(createDirectoryName);
367         CreateDirectory(createDirectoryName);
368     }
```

-mkdir : 直接呼叫 kernel->fileSystem->CreateDir(name)

```
162 //----------------------------------------------------------------------
163 // MP4 mod tag
164 // CreateDirectory
165 //      Create a new directory with "name"
166 //----------------------------------------------------------------------
167 static void CreateDirectory(char *name) //name:
168 {
169     DEBUG(dbgFile, "Create directory: "<<name<<" with size: "<<NumDirect * sizeof(DirectoryEntry));
170     kernel->fileSystem->CreateDir(name);
171 }
172
173 //----------------------------------------------------------------------
```

**threads/kernel.cc : “-f” Format 的 flag 在此定義**

```
#ifndef FILESYS_STUB
        } else if (strcmp(argv[i], "-f") == 0) {
            formatFlag = TRUE;
#endif
```

```
113 #ifdef FILESYS_STUB
114     fileSystem = new FileSystem();
115 #else
116     fileSystem = new FileSystem(formatFlag);
117 #endif // FILESYS_STUB
```

**另外 lib / debug.h 我們新增幾個 DEBUG 用的 flag:**

```
33  /* ------------- MP4 ---------------- */
34  const char dbgdir = 'l';
35  const char dbgFileRemove = 'r';
36  const char dbgFileCreate = 'c';
```

## Part II - b

Support Larger File Size:

我們使用 **Multi-level indexed allocation scheme**，至多可以提供到 4-level ( 30^4 * 128 = 100 MB (For Bonus I )

**filesys / filehdr.h**

首先定義每個 level 最多可以容納的檔案 bytes 數

```
27  /* ------- MP4 ---------- */
28  #define num_of_bytes_1level NumDirect*SectorSize // 30*128 ~= 4kB Original nachOS
29  #define num_of_bytes_2level NumDirect*NumDirect*SectorSize // 30*30*128= 128 KB
30  #define num_of_bytes_3level NumDirect*NumDirect*NumDirect*SectorSize // 30^3*128 = 3.5 MB
31  #define num_of_bytes_4level NumDirect*NumDirect*NumDirect*NumDirect*SectorSize // 30^4*128 = 100 MB
32
```

```
76      int get_num_of_bytes(){ return numBytes;}
77      int get_num_of_sectors(){    return numSectors;}
```

**filesys / filehdr.cc**

**Allocate() :**

allocate FileHeader 的 dataSectors => 對應的 disk sector

這邊因為使用多層分配，需視檔案的 fileSize 多大，決定要給幾層的 index Table。如下圖所示

```
77  bool FileHeader::Allocate(PersistentBitmap *freeMap, int fileSize)
78  {
79      numBytes = fileSize;
80      // KEY !!!  Below: use fileSize instead of private member. numByes!! Since
81
82      numSectors = divRoundUp(fileSize, SectorSize);  // need numSectors to suppo
83      // 1. Check whether still have free space.
84      DEBUG(dbgFileCreate, "This file needs "<<numSectors<<" sectors");
85      if (freeMap->NumClear() < numSectors)
86          return FALSE; // not enough space
87
88      if(fileSize > num_of_bytes_4level ){ // Four-level indirect (~100 MB)// sin
108     else if(fileSize > num_of_bytes_3level){ // tripple indirect (~3MB)···
125     else if(fileSize > num_of_bytes_2level){ // double indirect (~128KB)···
142     else if(fileSize > num_of_bytes_1level){ // double indirect (~4KB)···
160     else{   // single indirect (orignal nachos)···
186
187
188     return TRUE;
189 }
```

仔細看其中一個就好，因為其他都一樣。只有到最後一層(小於原本 nachOS 支援的 4KB)才真正分配 sector 給 dataSectors!

1. 如果 fileSize 大於 num_of_bytes_4_level : 需要分配多個 indextable
   先 create 一個 dataSector[i] (一個 entry in FCB) 要用來存 指向下一層 table 的 sector number!

2. 接著地回呼叫 sub_hdr->Allocate()，注意如果所剩的 bytes 數小於這層，直接呼叫 Allocate(fileSize)，若否呼叫(4_level_bytes) 才會正確執行!
3. 把 fileSize 減去剛剛分配掉的 4_level size (bytes)，下次 while-loop 的 fileSize 才是正確的。
4. 記得將結果 writeBack 到 sub_hdr 的 dataSectors[i]
   (寫回他的 on-disk FCB，下次從 disk FetchFrom 時才是正確的值!)
5. i++;

```
88    if(fileSize > num_of_bytes_4level ){ // Four-level indirect (~100 MB)// single indirect (orignal nachos)
89        DEBUG(dbgFileCreate, ">4level: "<<num_of_bytes_4level<<" bytes, need "<<fileSize<<" bytes");
90        int i=0; // index for dataSectors.
91        while(fileSize>0){  // still needs to allocate.
92            dataSectors[i] = freeMap->FindAndSet();
93            FileHeader* sub_hdr = new FileHeader(); // sub header
94            // This if-else is used for: when the last entry(dataSector) is assign, and its remain size(fileSize)<nu
95            // We just allocate space on Disk for the rest(remaining) bytes. (go else{})
96            if(fileSize >  num_of_bytes_4level){
97                sub_hdr->Allocate(freeMap, num_of_bytes_4level); // using the same bitmap to allocate
98            }
99            else{
100               sub_hdr->Allocate(freeMap, fileSize);
101           }
102           fileSize -= num_of_bytes_4level;    // Every while-loop we allocate some size, so remember to minus num_
103           sub_hdr->WriteBack(dataSectors[i]); // Write Back to disk.
104           i++; // move to the next dataSector.
105       }
106
107   }
```

**看 else :**

跟原本 nachOS single-indirect 分配一樣!

差別在第 177 ~ 180 處：要正確把 disk sector 清乾淨再分配!

這步超級關鍵，我們卡在這邊大概兩天以上 de 不出 bug

```
160   else{   // single indirect (orignal nachos)
161       DEBUG(dbgFileCreate, "<= 1level: "<<num_of_bytes_1level<<" bytes, need "<<fileSize<<" bytes");
162       #ifdef DEBUG_FHDR_ALLOCATE···
164       #endif
165
166       for (int i = 0; i < numSectors; i++)
167       {
168           // Allocate datablock here! One sector at a time
169           dataSectors[i] = freeMap->FindAndSet();
170           // since we checked that there was enough free space,
171           // we expect this to succee
172
173           DEBUG(dbgFileCreate, "dataSector["<<i<<"] takes sector "<<dataSectors[i]);
174           ASSERT(dataSectors[i] >= 0); // sector number should be >=0
175
176           ///////// MP4 KEY !!!!! CLEAN DISK HERE!!! ////////////
177           char* clean_data = new char[SectorSize]();
178           // memset(clean_data, 0, SectorSize);
179           kernel->synchDisk->WriteSector(dataSectors[i], clean_data);
180           delete clean_data;
181
182       }
183       DEBUG(dbgFileCreate, "Done allocating this level of fileheader.");
184
185   }
186
187
188   return TRUE;
189 }
```

**MP4 KEY !!　關鍵**

解決：執行完 partII_a.sh 後 執行 partIII.sh，會拿到髒的 disk sectors 的問題：

每次從 bitmap 獲得 free disk sectors 時：同時把該 disk->WriteSector() 寫入空的值!

```
174         ASSERT(dataSectors[i] >= 0); // sector number should be >=0
175
176         ///////// MP4 KEY !!!!! CLEAN DISK HERE!!! ////////////
177         char* clean_data = new char[SectorSize]();
178         // memset(clean_data, 0, SectorSize);
179         kernel->synchDisk->WriteSector(dataSectors[i], clean_data);
180         delete clean_data;
181
182     }
```

畢竟 DISK_0 也是一個 File, 用來模擬真實的硬碟。

注意到：bitmap 雖然是掌管 "Free Disk Sectors"，但當你使用 Deallocate FCB 的 dataSectors 時，或是 Clear 掉 FCB 使用的那個 sector，這些動作都只有將 “bitmap”這個 dataStructure 自己的 bit flush 掉！或是將 FileHeader 自己的 dataStructure reset 以及將 directory 的那個 file 占用的 entry 歸零。

並沒有真正將 disk 內容清洗！因此你若不清洗這格 sector，下次 allocate 時 bitmap 會以為這個 sector 是 free 的(事實上真的是 free)，但因為內容還是 dirty 的，留有上一次被寫入的內容，必須先歸零，再安心交給下個 file 使用!

**Deallocate():**

把 freeMap 中 FileHeader 的 dataSectors => 對應的 disk sector 的 bit 清掉 (從 occupied => free )

**分法跟 Allocate 一樣，看內部細節**

**NumDirect = 30 (一個 index table 最多 30 個 dataSectors)**

**for 迴圈指的是 "要往下跑幾次迴圈才真正清完所有的 dataSectors)**

**例如：numSectors = 1000，除以 30 = 33.多**

**因為 divRoundUp = > 要跑 34 次!**

```
220 void FileHeader::Deallocate(PersistentBitmap *freeMap)
221   {
222     if(numBytes > num_of_bytes_4level ){ // Four-level indirect (~100 MB)// sir
223         for (int i = 0; i < divRoundUp(numSectors, NumDirect) ; i++) // 要 deal
224         {
225             /*
226                 NumDirect = 30 (一個index table最多 30 個 dataSectors)
227                 for 迴圈指的是 "要往下跑幾次迴圈才真正清完所有的dataSectors)
228                 例如：numSectors = 1000，除以30 = 33.多
229                 因為divRoundUp = > 要跑 34次!
230             */
231             DEBUG(dbgFileRemove, "FileHDR: Deallocate(): > 4level");
232             FileHeader* subdir = new FileHeader();
233             // subdir->FetchFrom(dataSectors[dataSectors[i]]);
234             subdir->FetchFrom(dataSectors[i]);
235             // after fetchfrom [sector number of this sub-header], subhdr would
236             // 'numBytes' and 'numSectors' of contents below this level.
237             subdir->Deallocate(freeMap); // recursive.
238         }
239     }
```

```
62     else{   // single indirect (orignal nachos)
63         DEBUG(dbgFileRemove, "numBytes: "<<this->numBytes);
64       for (int i = 0; i < numSectors; i++)
65       {
66         DEBUG(dbgFileRemove,"Ready to test dataSectors["<<i<<"] : on-disk sector " <<dataSectors[i]);
67         // std::cout<<"Ready to test dataSectors["<<i<<"] : on-disk sector " <<dataSectors[i]<<"\n";
68         ASSERT(freeMap->Test((int)dataSectors[i])); // ought to be marked!
69         DEBUG(dbgFileRemove,"Test dataSectors["<<i<<"] true!(marked)");
70         // std::cout<<"Test dataSectors["<<i<<"] true!(marked)\n";
71         freeMap->Clear((int)dataSectors[i]);
72       }
73     }
74 }
```

**ByteToSector():**

給定 virtual dataSector number => map 到 physical sector number!

用 divRoundDown 是因為 dataSectors index start from 0! e.g. offset/num... = 1.7, 要去第二個 entry 繼續找，但因為第二個 entry 是 datasectors[1],所以用 roundown

```
347 int FileHeader::ByteToSector(int offset)
348   {
349     if(numBytes > num_of_bytes_4level){     // Level 1
350         FileHeader* subhdr = new FileHeader;
351         // subhdr->FetchFrom()
352         int entry_number = divRoundDown(offset,num_of_bytes_4level); // R
353         //dataSectors index start from 0! e.g. offset/num... = 1.7, 要去第
354         subhdr->FetchFrom(dataSectors[entry_number]);
355         subhdr->ByteToSector(offset-num_of_bytes_4level*entry_number);
356
357     }
358     else if(numBytes > num_of_bytes_3level){ // Level 2
359         FileHeader* subhdr = new FileHeader;
360         // subhdr->FetchFrom()
```

```
    }
    else{   // Level 5: True DataBlock
        return (dataSectors[offset / SectorSize]);
    }
```

**Print():**

Print 跟 byteToSector 一樣都是用 divRoundDown 概念

```
if(numBytes > num_of_bytes_4level){     // Level 1
    for(int i=0; i< numSectors/NumDirect; i++){ // If FULL: run 30 recursive calls
        OpenFile* opfile = new OpenFile(dataSectors[i]);
        FileHeader* subhdr = opfile->getHdr();
        subhdr->Print();
    }
}
else if(numBytes > num_of_bytes_3level){ // Level 2···
else if(numBytes > num_of_bytes_2level){ // Level 3···
else if(numBytes > num_of_bytes_1level){ // Level 4···
```

```
415 void FileHeader::Print()
416 {
417     int i, j, k;
418     char *data = new char[SectorSize];
419
420     printf("FileHeader contents.  File size: %d.  File blocks:\n", numBytes);
421     for (i = 0; i < numSectors; i++)
422         printf("%d ", dataSectors[i]);
423     printf("\nFile contents:\n");
424
425
426     if(numBytes > num_of_bytes_4level){     // Level 1···
433     else if(numBytes > num_of_bytes_3level){ // Level 2···
440     else if(numBytes > num_of_bytes_2level){ // Level 3···
447     else if(numBytes > num_of_bytes_1level){ // Level 4···
454     else{   // Level 5: True DataBlock
455         for (i = k = 0; i < numSectors; i++)
456         {
457             kernel->synchDisk->ReadSector(dataSectors[i], data);
458             for (j = 0; (j < SectorSize) && (k < numBytes); j++, k++)
459             {
460                 if ('\040' <= data[j] && data[j] <= '\176') // isprint(data[j])
461                     printf("%c", data[j]);
462                 else
463                     printf("\\%x", (unsigned char)data[j]);
464             }
465             printf("\n");
466         }
467     }
468     delete[] data;
469
470 }
471
```

## 3. Modify the file system code to support subdirectory

### Part III :

因為現在 傳入的參數 name 可能是很長的 path (例如 /t0/bb/f1)
要逐步拆解 pathname，直到最後一個 token 才是真正的 filename，並且要一步一步進入下一個 directory (因為我們使用 Multi-level Indirect)

**filesys / filesys.cc**

**Create(char* pathname):**
**CreateDir(char* pathname):**
直接看 CreateDir(), Create() 只須要改一行 code 的內容而已。
首先步驟跟原本 nachOS Create()類似，要從 root directory 開始往下找。我們用 strtok(name, "/") 來逐一拆解 token。

```
372  // For sub-dir
373  bool FileSystem::CreateDir(char *name)
374  {
375      Directory *directory;
376      PersistentBitmap *freeMap;
377      FileHeader *hdr;
378      int sector;
379      bool success;
380
381      DEBUG(dbgFile, "Creating dir file " << name << " size " << DirectoryFileSize);
382
383      directory = new Directory(NumDirEntries);
384      directory->FetchFrom(directoryFile); // directoryFiile 是 openfile,
385                                           // 讓新建的 directory（視為一個暫存的buffer）去
386
387      // Start searching from root dir...
388      OpenFile *file_temp; //  KEY! Should initialize file_temp!
389      file_temp = directoryFile;
390      char *token = strtok(name, "/");
391      char *prev_token = token;
392      OpenFile *last_level_dir_file = file_temp;
393      bool should_roll_back_to_last_dir = TRUE;
394
```

以下為拆解 tokenz 方法：在 while-loop 內，如果 directory 找不到這個 token 的 entry：表示還沒建立過，直接 break 掉，跳出 while 迴圈。此時 token 不是 NULL。

```
395     while( token!=NULL){
396         if(  (sector = directory->Find(token)) ==-1 ){
397             DEBUG(dbgFileCreate, "First create dir "<<token<<" in cur dir. Break");
398              should_roll_back_to_last_dir = FALSE;
399              break;
400         }
401         // Else: should traverse to the last layer!
402         std::cout<<"Existed dir: "<<token<<" its FCB at sector "<<sector<<"\n";
403         last_level_dir_file = file_temp;        // update last level file.
404 
405         file_temp = new OpenFile(sector);
406         directory->FetchFrom(file_temp);
407         prev_token = token; //update prev_token
408 
409 
410         token = strtok(NULL, "/"); // update toke to the next token after "/"
411         // std::cout<<"prev_token "<<prev_token<<", next token"<<token<<"\n";
412         DEBUG(dbgdir, "prev_token "<<prev_token);
413     }
414 
415     if(token==NULL){
416 
417         token = prev_token;
418     }
419 
420     // KEY !!
421     if(should_roll_back_to_last_dir) directory->FetchFrom(last_level_dir_file);
```

接著步驟跟原本 Create 一樣：先確認 directory 有沒有，沒有的話看 freeMap 有沒有空間放此 File(directory 也是一個 File)的 FCB (需要一個 sector)。最後看 disk 上所有 free sectors 夠不夠存你需要的 file 大小。

```
436     if (directory->Find(token) != -1){
437         std::cout<<"Err contruct the same dir \'"<<token<<"\' in cur dir. Break\n";
438         DEBUG(dbgFile, "ERROR!!!  Directory with the same name already exist in current dir. Abort!");
439         success = FALSE; // Dir with the same name already exist.
440     }
441     else // File hasn't been created! CAN create this file
442     {
443         freeMap = new PersistentBitmap(freeMapFile, NumSectors);// 新建的 freemap（視為一個暫存的buffer）去讀取file system
444          // step 2~4
445          // 先找第一個sector, 存fileheader用
446         // Debug for freeMap!
447         if (debug->IsEnabled('f'))···
452 
453         sector = freeMap->FindAndSet(); // find a sector to hold the file header
454         // #ifdef DEBUG_CREATE_DIR
455         DEBUG(dbgFileCreate, "Sector number for FCB of 'directory' "<<token<<" is "<<sector);
456         // #endif
457 
458         if (sector == -1){
459             DEBUG(dbgFile, "No free bit from bitmap. => No available free sector for your directory. Abort!");
460             success = FALSE; // no free block for file header
461         }
462             // 再看 root directory還有沒有entry可以存這個file
463         // else if (!directory->Add(token, sector))
464         else if (!directory->Add(token, sector, TRUE)){
465             DEBUG(dbgFile, "No free entry for your directory in current directory(Already 64 entries! FULL). Abort!");
466             success = FALSE; // no space in directory
467         }
468         else
469         {
```

以上檢查都完成且成功的話：使用 filehdr->Allocate()來逐一分配 sector => dataSectors[ ] 給這個新建的 directoryFile

**在把這個 file 加入 directory 某個 entry 時**，是使用：

```
    // 再看 root directory還有沒有entry可以存這個fil
// else if (!directory->Add(token, sector))
else if (!directory->Add(token, sector, TRUE)){
    DEBUG(dbgFile, "No free entry for your directo
    success = FALSE; // no space in directory
```

下面會再針對 directory.cc 中做說明。

```
        else
        {

            hdr = new FileHeader;
            DEBUG(dbgFileCreate, "Allocating file header for [dir] file: "<<token<<", FCB is at sector "<<sector);
            // 最後看 用header來allocate 'initialsize' 給這個file, disk空間還夠不夠！（用當前的bitmap:=freemap）
            if (!hdr->Allocate(freeMap, DirectoryFileSize)){
                DEBUG(dbgFile, "No enough [number of sectors] on disk for your directory! Abort!");
                success = FALSE; // no space on disk for data
            }
            else
            {
                //step 5.& 6.
                success = TRUE;
                #ifdef DEBUG_CREATE_DIR ···
                #endif

                hdr->WriteBack(sector);
                directory->WriteBack(file_temp);
                freeMap->WriteBack(freeMapFile);

                /* --------------------···


            }
            delete hdr;
        }
        delete freeMap;
    }
    delete directory;
    return success;
}
```

而 Create() 一個 File 的差別也只有在"加入 dir entry"時：

```
else if (!directory->Add(token, sector,FALSE)){
    std::cout<<"Err: No free entry for file \'"<<token
    success = FALSE; // no space in directory
}
```

**Open (char *pathname ):**

**分解 token 步驟跟上面類似：**

**把 name assign 成 token**

```
    if(token==NULL){
        name = prev_token;
    }
    else{
        #ifdef DEBUG_TOKEN ···
        #endif
        name = token;  // if token i
    }
    // std::cout<<"\nOpen file name:
```

在這層 directory 去找 token，找到的話回傳 OpenFile*指標
找不到的話回傳 NULL

```
685     sector = directory->Find(name); // return the sector contain this file's header
686
687 >   #ifdef DEBUG_OPENFILE ···
690     #endif
691
692     if (sector >= 0){
693         openFile = new OpenFile(sector); // name was found in directory
694         DEBUG(dbgFileCreate, "Success in finding file when Open(), sector: "<<sector<<
695     }
696     delete directory;
697     opfile = openFile;
698     return openFile; // return NULL if not found
699 }
```

**List()**

**跟上面依樣拆解 token，這邊要注意是 token 可能是 NULL!**
**因為可能會呼叫 list root (-l /)，所以如果 token 是 NULL，把 token assign 成 "/" 繼續執行。**

```
926 void FileSystem::List(char* dirname, bool Recursive)
927 {
928     Directory *directory = new Directory(NumDirEntries);
929     directory->FetchFrom(directoryFile); // first: take root dir!
930
931     OpenFile* dirFile = directoryFile;
932     int sector;
933
934     char *token = strtok(dirname, "/");
935     char *prev_token = dirname; // e.g. : "/t0" => get t0
936
937 >   while( token!=NULL){ ···
953
954     // if(token!=NULL)std::cout<<"Token before processed: "<<token<<"
955     // else std::cout<<"Token before processed is NULL\n";
956
957     if(token==NULL){    // only if pass "/"
958         token  = prev_token;
959         #ifdef DEBUG_TOKEN
960         std::cout<<"Traverse path to the last token\n";
961         #endif
962     }
963 >   #ifdef DEBUG_TOKEN ···
965     #endif
966
967     // if root dir.
968     if(strcmp(token, "/")==0) sector = DirectorySector;
```

接著看是否要 Recursive，要的話呼叫 directory->RecursiveList(0)
0 是 起始的 depth，從這層 dir 開始，depth 從 0 開始計算

```
975     if(Recursive){
976 
977         // std::cout<<"Checking: Current level :\n";
978         // directory->List();
979         // std::cout<<"Now passing RecursiveList(depth = 0);\n";
980 
981         directory->RecursiveList(0);  // start from depth 0.
982         std::cout<<"\n";
983     }
984     else  directory->List();
985     delete directory;
986 }
```

**Remove()**

Remove()跟 List()很像。**注意的地方是：**假如要 RecursiveList，要先進入下一層 dir，呼叫"他的"dir -> RecursiveRemove()。

**等到 return 後，回到這層 dir(他的上一層)，這樣才能夠從這層 dir 刪除他的 entry!!**

```
809 
810     freeMap = new PersistentBitmap(freeMapFile, NumSectors);
811 
812     if(Recursive){
813         if(directory->IsDir(token)){
814             // std::cout<<"token "<<token<<" is a directory!\n";
815             DEBUG(dbgFileRemove,  "token "<<token<<" is a directory!");
816             // 1. Enter this absolute dir. /t0/bb
817             directory->FetchFrom(actual_dir_opfile);
818             DEBUG(dbgFileRemove,"Enter "<<token<<" dir:" );
819             directory->List();
820 
821             directory->RecursiveRemove(token); // Pass original pathname (e.g. /t0).
822                                                // Since we want to traverse all 'absolute path name' under this (cur) directory!
823             // After return from recursive remove:
824             // cur dir: /bb
825             directory->WriteBack(actual_dir_opfile); // flush to disk   //////// KEY!!!!
826             DEBUG(dbgFileRemove,"Return from Directoyr::RecursiveRemove("<<token<<"), check dir "<<token );
827             // std::cout<<"Return from Directoyr::RecursiveRemove("<<token<<"), check dir "<<token<<"\n";
828             // list bb.
829             directory->List();
830 
```

```
832             // std::cout<<"Now, want to deallocate the header of "<<token<<". \n";
833             // fileHdr : /t0/bb
834             // Clear /t0/bb's bitmap
835             fileHdr->Deallocate(freeMap); // remove data blocks
836             // std::cout<<"Now, want to clear the header-bit (sector number ) on bitmap\n";
837 
838             // sector of /bb 's FCB.
839             freeMap->Clear(sector);       // remove header block
840 
841             // Note: Go back to the last level dir and remove /bb!
842             // Now: in /t0
843             directory->FetchFrom(last_level_dir_file);
844             // From t0 : remove /bb
845             directory->Remove(token);
846             // Write back result to /t0
847             directory->WriteBack(last_level_dir_file); // flush to disk   //////// KEY!!!!
848 
849             DEBUG(dbgFileRemove, "Success in RecursiveRemove dir "<<token);
850 
851         }
```

**filesys / filesys.h**

更改 Number of directory "Entries"!!原本只有 10，擴充到 64

```
42    /* MP4  */
43    #define NumDirEntries 64
```

Member function 部分：

新增幾個會用到的 function：

CreateDir() 建立子資料夾、Remove()跟 List()需多傳入 bool Recursive 決定是否要遞迴輸出。

Write(), Read(), Close() 用來接到 system call。

```
107     /* ---------- MP4 ---------- */
108     bool CreateDir(char *name);
109
110     // bool Create(char *name, int initialSize, bool isDir); // create a dir or a file.
111     // Create a file (UNIX creat)
112     bool Remove(char *name, bool Recursive); // Delete a file (UNIX unlink)  ; recursive remove or not
113
114     // void List(bool Recursive); // List all the files in the file system ; recursive list or not
115     void List(char* dirname, bool Recursive);
116
117     int Write(char *buffer, int size, OpenFileId id);
118     int Read(char *buffer, int size, OpenFileId id);
119     int Close(OpenFileId id);
120     bool CreateDirectory(char *createDirectoryName, int initialSize);
121     OpenFile *FindSubDir(char *filepath);
122     OpenFile* getFreeMapFile(){return freeMapFile;}
123
124     /* ------------------------ */
```

private： 新增 OpenFile* opfile 控制 read / write !

```
126   private:
127       OpenFile *freeMapFile;   // Bit map of free disk blocks,
128                                // represented as a file
129                                // Node: If freeMap Itself needs 500 dataBl
130
131       OpenFile *directoryFile; // "Root" directory -- list of
132                                // file names, represented as a file
133       OpenFile *opfile;        /* MP4 For read write.*/
```

**filesys / directory.cc**

Add() 新增一個 argument : isDir，用意是標示這個 entry 是 [D]或是 [F]

```
143   /*  MP4 */
144   bool Directory::Add(char *name, int newSector, bool isDir)
145   {
146       if (FindIndex(name) != -1)
147           return FALSE;
148
149       // Find free entry in the directory table, set inUse to TRUE and string copy!
150
151       for (int i = 0; i < tableSize; i++)
152           if (!table[i].inUse)
153           {
154               table[i].inUse = TRUE;
155               strncpy(table[i].name, name, FileNameMaxLen); // string copy: copy 'nam
156               table[i].sector = newSector;
157               table[i].isDir = isDir; /* ---------- MP4 --------- */
158               return TRUE;
159           }
160       return FALSE; // no space.  Fix when we have extensible files.
161   }
```

**List()**

大致維持原本，稍微修改輸出格式。

```
334 void Directory::List()
335 {
336     bool empty = TRUE;
337     // printf("Directoyr::List() tableSize: %d\n",tableSize);
338     for (int i = 0; i < tableSize; i++){
339         // std::cout<<"Entry "<<i<<"inUse: "<<table[i].inUse<<", name
340         if (table[i].inUse==TRUE){
341             empty = FALSE;
342             // printf("%s, %d\n",table[i].name, table[i].isDir);
343             if(table[i].isDir==TRUE)
344                 std::cout<<"[D] "<<table[i].name<<std::endl;
345             else std::cout<<"[F] "<<table[i].name<<std::endl;
346         }
347         // else std::cout<<"Entry "<<i<<" not used.\n";
348     }
349
350     if(empty) std::cout<<"The directory is empty\n";
351 }
```

**RecursiveList()**

記得要傳入參數 “depth"，因會影響輸出時的版面(要縮排幾格)

一樣掃過整個 dir，**如果 entry 是 dir，呼叫 subdir->RecursiveList。**

**記得每次宣告一個 Directory 的 pointer，都要用 FetchFrom(OpenFile* ) 從 disk 把這個 directory 的 File 內容讀出來!**

```
288 void Directory::RecursiveList(int depth)
289 {
290     // printf("Run Recursive list!\n");
291     bool empty = TRUE;
292     Directory *subdir = new Directory(NumDirEntries);
293     OpenFile *subdir_openfile;
294
295     DEBUG(dbgFile,"Directory::RecursiveList in depth:  "<<depth);
296
297     for(int i=0; i< tableSize; i++){
298         if(table[i].inUse){
299             std::cout<<"\n"; // New line in every (dir)table-item in each recursive call.
300             empty = FALSE;
301             for(int k=0; k<depth; k++) std::cout<<"  ";
302             if(table[i].isDir){
303                 printf("[D] %s", table[i].name);
304                 subdir_openfile = new OpenFile(table[i].sector);
305                 DEBUG(dbgFile, "table["<<i<<"] is dir, current_dir->FetchFrom( OpenFile* ("<<table[i].sector<<") )");
306                 subdir->FetchFrom(subdir_openfile); // KEY!  This openfile.sector should be clear when removing!!
307                 subdir->RecursiveList(depth+1);
308
309             }
310             else{
311                 printf("[F] %s", table[i].name);
312             }
313         }
314     }
315     if(empty){
316         for(int k=0; k<depth; k++) std::cout<<" ";
317         std::cout<<"(Empty directory)";
318     }
319     depth--; //remember to bring back the depth!
320 }
```

**Remove()**

維持原本的 Remove()，**新增遞迴函式 RecursiveRemove()**

```
190
191  bool Directory::Remove(char *name)
192  {
193      int i = FindIndex(name);
194
195      if (i == -1)
196          return FALSE; // name not in directory // No su
197      table[i].inUse = FALSE;
198      return TRUE;
199  }
200
201  void Directory::RecursiveRemove(char *name) ···
288
```

**RecursiveRemove()**

RecursiveRemove 和 Remove 很像，差別在：go through 每個 entry 時，如果這個 entry 是 sub directory，要呼叫該 subdirectory 自己的 RecursiveRemove()!

重點：記得 return from subdirectory->RecursiveRemove() 後

要刪除這個 subdirectory 的 entry! (使用這層 dir 的 Remove(next_dir)

```
200
201  void Directory::RecursiveRemove(char *name)
202  {
203      bool empty = TRUE;
204      // First enter current dir (e.g /t0/bb)
205      DEBUG(dbgFile, "Current in : "<<name);
206      for(int i=0; i<tableSize; i++){
207          if(table[i].inUse){
208              empty = FALSE;
209              /* KEY ! If have next dir: should record current dir, when return, use
210                  cur->dir->Remove(table[i].name);
211              */
212
213              if(table[i].isDir){···
259              else{ // is a fil.e···
280          }
281      }
282
283      if(empty) std::cout<<"This directory is empty.\n";
284      // return if empty dir.
285  }
286
```

有很多 DEBUG 訊息導致版面大 圖片小 不好意思!!

再下面放下比較重點的分解截圖!

```
213         if(table[i].isDir){
214             // kernel->fileSystem->Remove(path_under_this_dir, TRUE);
215             DEBUG(dbgFile, "entry "<<i<<" is Dir. name "<<table[i].name<<", sector "<<table[i].sector);
216             Directory* next_dir = new Directory(NumDirEntries);
217             OpenFile* next_dir_file = new OpenFile(table[i].sector);    // (FCB of the next_directory file. ) =>Open from FCB.
218             next_dir->FetchFrom(next_dir_file);
219 
220             PersistentBitmap* freeMap = new PersistentBitmap(kernel->fileSystem->getFreeMapFile() , NumSectors);
221             FileHeader* next_dirfile_tobeRemove = new FileHeader;
222             next_dirfile_tobeRemove->FetchFrom(table[i].sector);
223 
224             // First Remove dataSectors.
225             DEBUG(dbgFileRemove, "First Remove dataSectors of dir file "<<table[i].name);
226             next_dirfile_tobeRemove->Deallocate(freeMap); // remove data blocks
227             DEBUG(dbgFileRemove, "Second Remove the FileHeader. "<<table[i].name<<", on-disk FCB sector: "<<table[i].sector);
228             DEBUG(dbgFileRemove, "freeMap->Clear("<<table[i].sector<<")");
229             // Second Remove the FileHeader.
230             freeMap->Clear(table[i].sector);
231             // Third, remove its directory entries.
232             DEBUG(dbgFileRemove,  "Third, remove its directory entries of dir /"<<table[i].name<<", recursive call.");
233             next_dir->RecursiveRemove(table[i].name);
234 
235             /*  Write back the result of next_dir to its FCB. */
236             DEBUG(dbgFileRemove,  "Write back the result of next_dir /"<<table[i].name<<" to its FCB.");
237             next_dir->WriteBack(next_dir_file);
238             /*
239                 Remove next_dir 's FCB & dataSectors.
240             */
241             DEBUG(dbgFileRemove,  "From this dir, remove next_dir /"<<table[i].name);
242             this->Remove(table[i].name); /* KEY!!!! */
243             freeMap->WriteBack( kernel->fileSystem->getFreeMapFile() );
```

**1. 取得 next_dir 的 directory***

```
if(table[i].isDir){
    // kernel->fileSystem->Remove(path_under_this_dir, TRUE);
    DEBUG(dbgFile, "entry "<<i<<" is Dir. name "<<table[i].name<<", sect
    Directory* next_dir = new Directory(NumDirEntries);
    OpenFile* next_dir_file = new OpenFile(table[i].sector);    // (FCB
    next_dir->FetchFrom(next_dir_file);

    PersistentBitmap* freeMap = new PersistentBitmap(kernel->fileSystem-
    FileHeader* next_dirfile_tobeRemove = new FileHeader;
    next_dirfile_tobeRemove->FetchFrom(table[i].sector);
```

**2. Deallocate next_dir 的 FileHeader, bitmap 並遞迴呼叫他的 RecursiveRemove。Return 後要記得 writeback 到他的 dirFile**

```
// First Remove dataSectors.
DEBUG(dbgFileRemove, "First Remove dataSectors of dir file "<<table[i].name);
next_dirfile_tobeRemove->Deallocate(freeMap); // remove data blocks
DEBUG(dbgFileRemove, "Second Remove the FileHeader. "<<table[i].name<<", on-disk FCB sector: "<<table[i].sector);
DEBUG(dbgFileRemove, "freeMap->Clear("<<table[i].sector<<")");
// Second Remove the FileHeader.
freeMap->Clear(table[i].sector);
// Third, remove its directory entries.
DEBUG(dbgFileRemove,  "Third, remove its directory entries of dir /"<<table[i].name<<", recursive call.");
next_dir->RecursiveRemove(table[i].name);

/*  Write back the result of next_dir to its FCB. */
DEBUG(dbgFileRemove,  "Write back the result of next_dir /"<<table[i].name<<" to its FCB.");
next_dir->WriteBack(next_dir_file);
/*
```

**3. 回到這層 directory(/t0)，刪除 next directory (/t0/bb) 的 entry!**

```
/*
    Remove next_dir 's entry in current directory
*/
DEBUG(dbgFileRemove,  "From this dir, remove next_dir /"<<table[i].name);
this->Remove(table[i].name); /* KEY!!!! */
freeMap->WriteBack( kernel->fileSystem->getFreeMapFile() );

delete next_dir_file;
delete next_dir;
delete next_dirfile_tobeRemove;
delete freeMap;
```

**filesys / directory.h**

在 DirectoryEntry class: 新增 member bool isDir 判斷這個 entry 是 file 還是 subdirectory

```
32  class DirectoryEntry
33  {
34  public:
35      bool inUse;                     // Is this directory entry in use?
36      int sector;                     // Location on disk to find the
37                                      //   FileHeader for this file
38      char name[FileNameMaxLen + 1]; // Text name for file, with +1 for
39                                      // the trailing '\0'
40
41      /* MP4 */
42      bool isDir; // TRUE: is directory , FALSE: is a file.
43
44  };
```

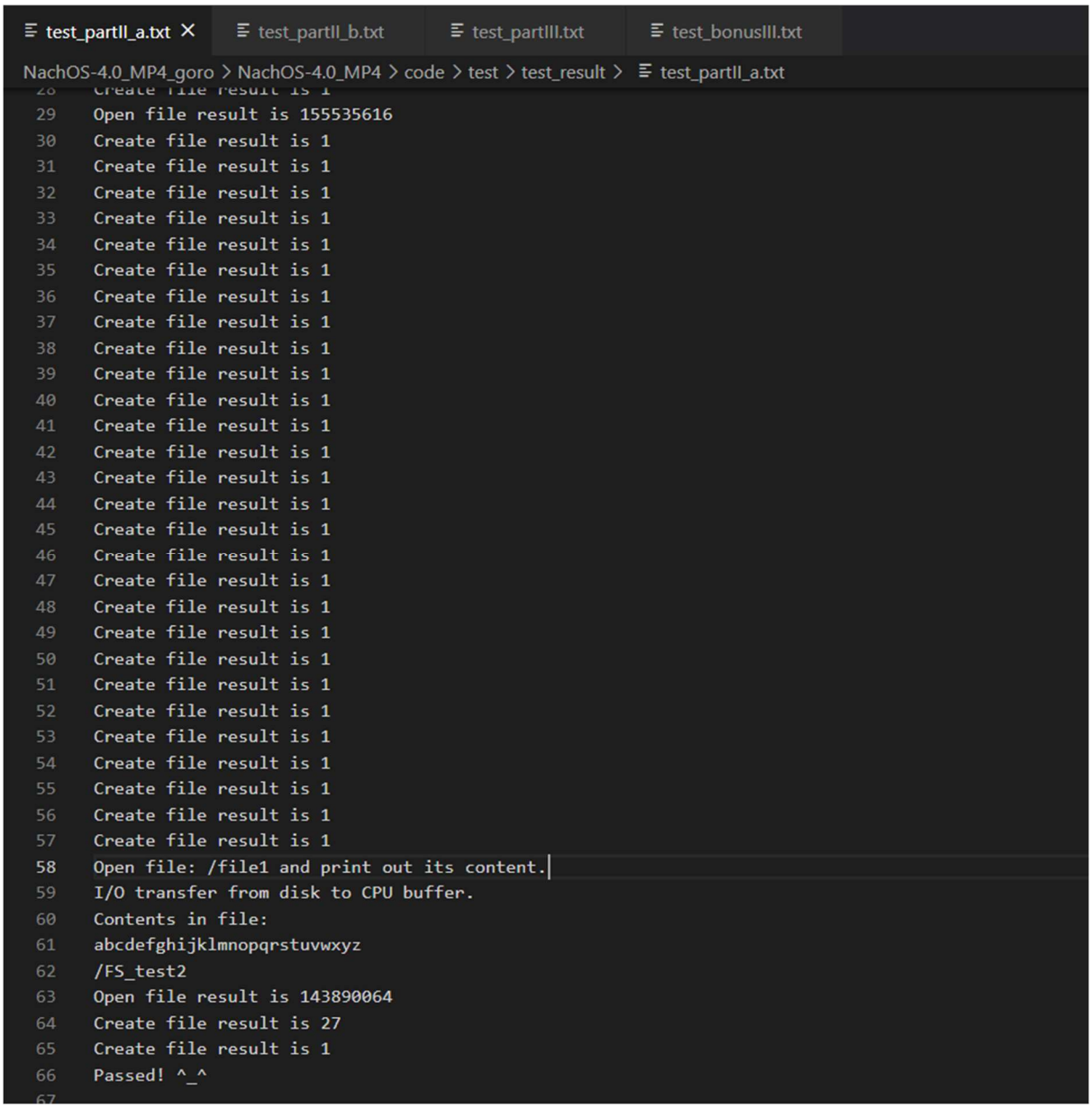
**新增一些好用的 member functions:**

```
80  /*---------- MP4---------------*/
81      bool Add(char *name, int newSector, bool isDir); // Add a file name into the directory
82      // New a entry for a 'file' or a 'sub-directory'
83
84      // bool Remove(char *name); // Remove a file from the directory
85      void RecursiveList(int depth);
86
87      void RecursiveRemove(char *name);
88
89      void List(int depth, bool Recursive);  // Print the names of all the files
90                  //  in the directory
91      // void Print(); // Verbose print of the contents
92                  //  of the directory -- all the file
93                  //  names and their contents.
94      bool IsDir(char* pathname){
95          int entry_index = this->FindIndex(pathname);
96          return this->table[entry_index].isDir;
97      };
98      int getTableSize(){return tableSize;}
99      void setTableSize(int size){tableSize = size;}
100     DirectoryEntry* getTable(){  return table;}
101
102
103 /*---------- MP4---------------*/
```

Directory 的 private members 則沒有改變，維持原樣。

# Implementation Results:

## PartII_a



# PartII_b

NachOS-4.0_MP4_goro > NachOS-4.0_MP4 > code > test > test_result > test_partII_b.txt

```
Total Num of Sectors: 512000
Need FreeMap Size: 64000
Need root Dir Size 1280
Formatting FS: allocating freemap dataSectors...
Formatting FS: allocating 'root directory' dataSectors...

----- Recursive List after copy two files -----

[F] 100
[F] 1000
Open file: /1000 and print out its content.
I/O transfer from disk to CPU buffer.
Contents in file:
000000001 000000002 000000003 000000004 000000005 000000006 000000007 000000008 000000009 000000010
000000011 000000012 000000013 000000014 000000015 000000016 000000017 000000018 000000019 000000020
000000021 000000022 000000023 000000024 000000025 000000026 000000027 000000028 000000029 000000030
000000031 000000032 000000033 000000034 000000035 000000036 000000037 000000038 000000039 000000040
000000041 000000042 000000043 000000044 000000045 000000046 000000047 000000048 000000049 000000050
000000051 000000052 000000053 000000054 000000055 000000056 000000057 000000058 000000059 000000060
000000061 000000062 000000063 000000064 000000065 000000066 000000067 000000068 000000069 000000070
000000071 000000072 000000073 000000074 000000075 000000076 000000077 000000078 000000079 000000080
000000081 000000082 000000083 000000084 000000085 000000086 000000087 000000088 000000089 000000090
000000091 000000092 000000093 000000094 000000095 000000096 000000097 000000098 000000099 000000100
000000101 000000102 000000103 000000104 000000105 000000106 000000107 000000108 000000109 000000110
000000111 000000112 000000113 000000114 000000115 000000116 000000117 000000118 000000119 000000120
000000121 000000122 000000123 000000124 000000125 000000126 000000127 000000128 000000129 000000130
000000131 000000132 000000133 000000134 000000135 000000136 000000137 000000138 000000139 000000140
000000141 000000142 000000143 000000144 000000145 000000146 000000147 000000148 000000149 000000150
000000151 000000152 000000153 000000154 000000155 000000156 000000157 000000158 000000159 000000160
000000161 000000162 000000163 000000164 000000165 000000166 000000167 000000168 000000169 000000170
000000171 000000172 000000173 000000174 000000175 000000176 000000177 000000178 000000179 000000180
000000181 000000182 000000183 000000184 000000185 000000186 000000187 000000188 000000189 000000190
000000191 000000192 000000193 000000194 000000195 000000196 000000197 000000198 000000199 000000200
000000201 000000202 000000203 000000204 000000205 000000206 000000207 000000208 000000209 000000210
000000211 000000212 000000213 000000214 000000215 000000216 000000217 000000218 000000219 000000220
000000221 000000222 000000223 000000224 000000225 000000226 000000227 000000228 000000229 000000230
000000231 000000232 000000233 000000234 000000235 000000236 000000237 000000238 000000239 000000240
000000241 000000242 000000243 000000244 000000245 000000246 000000247 000000248 000000249 000000250
000000251 000000252 000000253 000000254 000000255 000000256 000000257 000000258 000000259 000000260
```

NachOS-4.0_MP4_goro > NachOS-4.0_MP4 > code > test > test_result > test_partII_b.txt

```
000000841 000000842 000000843 000000844 000000845 000000846 000000847 000000848 000000849 000000850
000000851 000000852 000000853 000000854 000000855 000000856 000000857 000000858 000000859 000000860
000000861 000000862 000000863 000000864 000000865 000000866 000000867 000000868 000000869 000000870
000000871 000000872 000000873 000000874 000000875 000000876 000000877 000000878 000000879 000000880
000000881 000000882 000000883 000000884 000000885 000000886 000000887 000000888 000000889 000000890
000000891 000000892 000000893 000000894 000000895 000000896 000000897 000000898 000000899 000000900
000000901 000000902 000000903 000000904 000000905 000000906 000000907 000000908 000000909 000000910
000000911 000000912 000000913 000000914 000000915 000000916 000000917 000000918 000000919 000000920
000000921 000000922 000000923 000000924 000000925 000000926 000000927 000000928 000000929 000000930
000000931 000000932 000000933 000000934 000000935 000000936 000000937 000000938 000000939 000000940
000000941 000000942 000000943 000000944 000000945 000000946 000000947 000000948 000000949 000000950
000000951 000000952 000000953 000000954 000000955 000000956 000000957 000000958 000000959 000000960
000000961 000000962 000000963 000000964 000000965 000000966 000000967 000000968 000000969 000000970
000000971 000000972 000000973 000000974 000000975 000000976 000000977 000000978 000000979 000000980
000000981 000000982 000000983 000000984 000000985 000000986 000000987 000000988 000000989 000000990
000000991 000000992 000000993 000000994 000000995 000000996 000000997 000000998 000000999 000001000
========================================
Open file: /100 and print out its content.
I/O transfer from disk to CPU buffer.
Contents in file:
000000001 000000002 000000003 000000004 000000005 000000006 000000007 000000008 000000009 000000010
000000011 000000012 000000013 000000014 000000015 000000016 000000017 000000018 000000019 000000020
000000021 000000022 000000023 000000024 000000025 000000026 000000027 000000028 000000029 000000030
000000031 000000032 000000033 000000034 000000035 000000036 000000037 000000038 000000039 000000040
000000041 000000042 000000043 000000044 000000045 000000046 000000047 000000048 000000049 000000050
000000051 000000052 000000053 000000054 000000055 000000056 000000057 000000058 000000059 000000060
000000061 000000062 000000063 000000064 000000065 000000066 000000067 000000068 000000069 000000070
000000071 000000072 000000073 000000074 000000075 000000076 000000077 000000078 000000079 000000080
000000081 000000082 000000083 000000084 000000085 000000086 000000087 000000088 000000089 000000090
000000091 000000092 000000093 000000094 000000095 000000096 000000097 000000098 000000099 000000100
========================================
[F] 100
[F] 1000

```

## Part III .sh

FS_partIII.sh | test_partIII.txt | test_bonusIII.txt

NachOS-4.0_MP4_goro > NachOS-4.0_MP4 > code > test > FS_partIII.sh

```
echo -e "=========== Testing TA's FS_partIII.sh... ==========\n"
../build.linux/nachos -f # if format: get error!
echo -e "\n----------- Mkdir /t0-----------\n"
../build.linux/nachos -mkdir /t0
echo -e "\n----------- Mkdir /t1 -----------\n"
../build.linux/nachos -mkdir /t1
echo -e "\n----------- Mkdir /t2 -----------\n"
../build.linux/nachos -mkdir /t2
echo -e "\n----------- List root dir -----------\n"
../build.linux/nachos -l /
echo -e "\n----------- Test copy file -----------\n"
echo -e "----------- Copy file num_100.txt to nachOS FS /t0/f1: -----------\n"
echo -e "Copy file num_100.txt to nachOS FS /t0/f1: \n"
../build.linux/nachos -cp num_100.txt /t0/f1
echo -e "\n----------- Mkdir /t0/aa -----------\n"
../build.linux/nachos -mkdir /t0/aa
echo -e "\n----------- Mkdir /t0/bb -----------\n"
../build.linux/nachos -mkdir /t0/bb
echo -e "\n----------- Mkdir /t0/cc -----------\n"
../build.linux/nachos -mkdir /t0/cc
echo -e "\n---------- Copy file num_100.txt to nachOS FS /t0/bb/f1: ----------\n"
echo -e "Copy file num_100.txt to nachOS FS /t0/bb/f1: \n"
../build.linux/nachos -cp num_100.txt /t0/bb/f1
echo -e "\n---------- Copy file num_100.txt to nachOS FS /t0/bb/f2: ----------\n"
../build.linux/nachos -cp num_100.txt /t0/bb/f2
echo -e "\n---------- Copy file num_100.txt to nachOS FS /t0/bb/f3: ----------\n"
../build.linux/nachos -cp num_100.txt /t0/bb/f3
echo -e "\n---------- Copy file num_100.txt to nachOS FS /t0/bb/f4: ----------\n"
../build.linux/nachos -cp num_100.txt /t0/bb/f4
echo -e "\n---------- List root dir /: ---------- \n"
../build.linux/nachos -l /
echo -e "\n---------- List dir /t0: ---------- \n"
../build.linux/nachos -l /t0
echo -e "\n---------- Recursive List root dir /: ---------- \n"
```

```
33
34  echo -e "\n---------- Copy file num_100.txt to nachOS FS /t0/bb/f1: ----------\n"
35  echo -e "Copy file num_100.txt to nachOS FS /t0/bb/f1: \n"
36  ../build.linux/nachos -cp num_100.txt /t0/bb/f1
37
38  echo -e "\n---------- Copy file num_100.txt to nachOS FS /t0/bb/f2: ----------\n"
39  ../build.linux/nachos -cp num_100.txt /t0/bb/f2
40  echo -e "\n---------- Copy file num_100.txt to nachOS FS /t0/bb/f3: ----------\n"
41  ../build.linux/nachos -cp num_100.txt /t0/bb/f3
42  echo -e "\n---------- Copy file num_100.txt to nachOS FS /t0/bb/f4: ----------\n"
43  ../build.linux/nachos -cp num_100.txt /t0/bb/f4
44  echo -e "\n---------- List root dir /: ---------- \n"
45  ../build.linux/nachos -l /
46  echo -e "\n---------- List dir /t0: ---------- \n"
47  ../build.linux/nachos -l /t0
48  echo -e "\n---------- Recursive List root dir /: ---------- \n"
49  ../build.linux/nachos -lr /
50  echo -e "\n---------- Print content /t0/f1: ---------- \n"
51  ../build.linux/nachos -p /t0/f1
52  echo -e "\n---------- Print content /t0/bb/f3:----------  \n"
53  ../build.linux/nachos -p /t0/bb/f3
54  echo -e "\n---------- Test Remove() ("-r")  ----------\n"
55  echo -e "Before removing /t0/f: \n"
56  ../build.linux/nachos -l /t0
57  ../build.linux/nachos -r /t0/f1
58  echo -e "After removing /t0/f: \n"
59  ../build.linux/nachos -l /t0
```

# Part III .txt

NachOS-4.0_MP4_goro > NachOS-4.0_MP4 > code > test > test_result > test_partIII.txt

```
========== Testing TA's FS_partIII.sh... ==========

Total Num of Sectors: 512000
Need FreeMap Size: 64000
Need root Dir Size 1280
Formatting FS: allocating freemap dataSectors...
Formatting FS: allocating 'root directory' dataSectors...

---------- Mkdir /t0----------


---------- Mkdir /t1 ----------


---------- Mkdir /t2 ----------


---------- List root dir ----------

[D] t0
[D] t1
[D] t2

---------- Test copy file ----------

---------- Copy file num_100.txt to nachOS FS /t0/f1: ----------

Copy file num_100.txt to nachOS FS /t0/f1:


---------- Mkdir /t0/aa ----------

Existed dir: t0 its FCB at sector 529

---------- Mkdir /t0/bb ----------

Existed dir: t0 its FCB at sector 529

---------- Mkdir /t0/cc ----------

Existed dir: t0 its FCB at sector 529

---------- Copy file num_100.txt to nachOS FS /t0/bb/f1: ----------

Copy file num_100.txt to nachOS FS /t0/bb/f1:


---------- Copy file num_100.txt to nachOS FS /t0/bb/f2: ----------


---------- Copy file num_100.txt to nachOS FS /t0/bb/f3: ----------


---------- Copy file num_100.txt to nachOS FS /t0/bb/f4: ----------


---------- List root dir /: ----------

[D] t0
[D] t1
[D] t2

---------- List dir /t0: ----------

[F] f1
[D] aa
[D] bb
[D] cc

---------- Recursive List root dir /: ----------


[D] t0
  [F] f1
  [D] aa  (Empty directory)
  [D] bb
    [F] f1
    [F] f2
    [F] f3
    [F] f4
  [D] cc  (Empty directory)
[D] t1 (Empty directory)
[D] t2 (Empty directory)
```

```
------------ Print content /t0/f1: -------------

Open file: /t0/f1 and print out its content.
I/O transfer from disk to CPU buffer.
Contents in file:
000000001 000000002 000000003 000000004 000000005 000000006 000000007 000000008 000000009 000000010
000000011 000000012 000000013 000000014 000000015 000000016 000000017 000000018 000000019 000000020
000000021 000000022 000000023 000000024 000000025 000000026 000000027 000000028 000000029 000000030
000000031 000000032 000000033 000000034 000000035 000000036 000000037 000000038 000000039 000000040
000000041 000000042 000000043 000000044 000000045 000000046 000000047 000000048 000000049 000000050
000000051 000000052 000000053 000000054 000000055 000000056 000000057 000000058 000000059 000000060
000000061 000000062 000000063 000000064 000000065 000000066 000000067 000000068 000000069 000000070
000000071 000000072 000000073 000000074 000000075 000000076 000000077 000000078 000000079 000000080
000000081 000000082 000000083 000000084 000000085 000000086 000000087 000000088 000000089 000000090
000000091 000000092 000000093 000000094 000000095 000000096 000000097 000000098 000000099 000000100

------------ Print content /t0/bb/f3:------------

Open file: /t0/bb/f3 and print out its content.
I/O transfer from disk to CPU buffer.
Contents in file:
000000001 000000002 000000003 000000004 000000005 000000006 000000007 000000008 000000009 000000010
000000011 000000012 000000013 000000014 000000015 000000016 000000017 000000018 000000019 000000020
000000021 000000022 000000023 000000024 000000025 000000026 000000027 000000028 000000029 000000030
000000031 000000032 000000033 000000034 000000035 000000036 000000037 000000038 000000039 000000040
000000041 000000042 000000043 000000044 000000045 000000046 000000047 000000048 000000049 000000050
000000051 000000052 000000053 000000054 000000055 000000056 000000057 000000058 000000059 000000060
000000061 000000062 000000063 000000064 000000065 000000066 000000067 000000068 000000069 000000070
000000071 000000072 000000073 000000074 000000075 000000076 000000077 000000078 000000079 000000080
000000081 000000082 000000083 000000084 000000085 000000086 000000087 000000088 000000089 000000090
000000091 000000092 000000093 000000094 000000095 000000096 000000097 000000098 000000099 000000100

------------ Test Remove() (-r)  --------------

Before removing /t0/f:

[F] f1
[D] aa
[D] bb
[D] cc
After removing /t0/f:

[D] aa
[D] bb
[D] cc

```

## ➢ Bonus Assignment

### Bonus I :

由於我們的 Multi-level 支援到 4-level 以上，fileheader 可以一直長(往下個 level)，另外還要改一個參數：影響整個 nachOS disk simulator 的大小：machine / disk.h

Number of Tracks!一個檔案要能夠 64MB 的話，除以 128B = 需要 512000 個 sector。

因為 SectorPerTrack 只有 32，所以 512000/32 = 需要 16000 個 Track!

```
50  const int SectorSize = 128;   // number of bytes per disk sec
51  const int SectorsPerTrack  = 32;  // number of sectors per di
52  // const int NumTracks = 32;    // number of tracks per disk
53  const int NumTracks = 16000;    // number of tracks per disk
54  const int NumSectors = (SectorsPerTrack * NumTracks);
55                  // total # of sectors per disk
```

**Test script:**

NachOS-4.0_MP4_goro > NachOS-4.0_MP4 > code > test > FS_bonusI.sh

```
1   echo "======== Testing BonusI.sh ============="
2   ../build.linux/nachos -f
3
4
5   echo -e "\n----- Mkdir /dir1 -----"
6   ../build.linux/nachos -mkdir /dir1
7
8   echo -e "\n----- Before copy num_1000000.txt: -----"
9   ../build.linux/nachos -l /
10
11  echo -e "\n----- After copy num_1000000.txt to /dir1/bonus1  -----"
12
13  ../build.linux/nachos -cp num_1000000.txt /dir1/bonus1
14  ../build.linux/nachos -lr /
15
16  echo -e "\n----- Print the content in bonus1 -----"
17  ../build.linux/nachos -p /dir1/bonus1
18
19  echo -e "\n----- Done testing FS_bonusI.sh -----"
20
21
```

**FS_bonusI.sh**

```
/NachOS-4.0_MP4/code/test]$ ./FS_bonusI.sh >> test_result/bonusI.txt
```

## Result:

NachOS-4.0_MP4_goro > NachOS-4.0_MP4 > code > test > test_result > bonusI.txt

```
======== Testing BonusI.sh =============
Total Num of Sectors: 512000
Need FreeMap Size: 64000
Need root Dir Size 1280
Formatting FS: allocating freemap dataSectors...
Formatting FS: allocating 'root directory' dataSectors...

----- Mkdir /dir1 -----
Sector number for FCB of 'directory' dir1 is 529
Allocating file header for [dir] file: dir1, FCB is at sector 529

----- Before copy num_1000000.txt: -----
Token after processed: /
FileSystem::List(): Dir / Fecth from sector 1
[D] dir1

----- After copy num_1000000.txt to /dir1/bonus1  -----
Token after processed: /
FileSystem::List(): Dir / Fecth from sector 1
Checking: Current level :
[D] dir1
Now passing RecursiveList(depth = 0);

[D] dir1
  [F] bonus1

----- Print the content in bonus1 -----
Open file: /dir1/bonus1 and print out its content.
I/O transfer from disk to CPU buffer.
Contents in file:
000000001 000000002 000000003 000000004 000000005 000000006 000000007 000000008 000000009 000000010
000000011 000000012 000000013 000000014 000000015 000000016 000000017 000000018 000000019 000000020
000000021 000000022 000000023 000000024 000000025 000000026 000000027 000000028 000000029 000000030
000000031 000000032 000000033 000000034 000000035 000000036 000000037 000000038 000000039 000000040
000000041 000000042 000000043 000000044 000000045 000000046 000000047 000000048 000000049 000000050
000000051 000000052 000000053 000000054 000000055 000000056 000000057 000000058 000000059 000000060
000000061 000000062 000000063 000000064 000000065 000000066 000000067 000000068 000000069 000000070
000000071 000000072 000000073 000000074 000000075 000000076 000000077 000000078 000000079 000000080
000000081 000000082 000000083 000000084 000000085 000000086 000000087 000000088 000000089 000000090
000000091 000000092 000000093 000000094 000000095 000000096 000000097 000000098 000000099 000000100
000000101 000000102 000000103 000000104 000000105 000000106 000000107 000000108 000000109 000000110
000000111 000000112 000000113 000000114 000000115 000000116 000000117 000000118 000000119 000000120
000000121 000000122 000000123 000000124 000000125 000000126 000000127 000000128 000000129 000000130
000000131 000000132 000000133 000000134 000000135 000000136 000000137 000000138 000000139 000000140
000000141 000000142 000000143 000000144 000000145 000000146 000000147 000000148 000000149 000000150
000000151 000000152 000000153 000000154 000000155 000000156 000000157 000000158 000000159 000000160
000000161 000000162 000000163 000000164 000000165 000000166 000000167 000000168 000000169 000000170
000000171 000000172 000000173 000000174 000000175 000000176 000000177 000000178 000000179 000000180
000000181 000000182 000000183 000000184 000000185 000000186 000000187 000000188 000000189 000000190
000000191 000000192 000000193 000000194 000000195 000000196 000000197 000000198 000000199 000000200
000000201 000000202 000000203 000000204 000000205 000000206 000000207 000000208 000000209 000000210
000000211 000000212 000000213 000000214 000000215 000000216 000000217 000000218 000000219 000000220
```

NachOS-4.0_MP4_goro > NachOS-4.0_MP4 > code > test > test_result > bonusI.txt

```
100009  000999781 000999782 000999783 000999784 000999785 000999786 000999787 000999788 000999789 000999790
100010  000999791 000999792 000999793 000999794 000999795 000999796 000999797 000999798 000999799 000999800
100011  000999801 000999802 000999803 000999804 000999805 000999806 000999807 000999808 000999809 000999810
100012  000999811 000999812 000999813 000999814 000999815 000999816 000999817 000999818 000999819 000999820
100013  000999821 000999822 000999823 000999824 000999825 000999826 000999827 000999828 000999829 000999830
100014  000999831 000999832 000999833 000999834 000999835 000999836 000999837 000999838 000999839 000999840
100015  000999841 000999842 000999843 000999844 000999845 000999846 000999847 000999848 000999849 000999850
100016  000999851 000999852 000999853 000999854 000999855 000999856 000999857 000999858 000999859 000999860
100017  000999861 000999862 000999863 000999864 000999865 000999866 000999867 000999868 000999869 000999870
100018  000999871 000999872 000999873 000999874 000999875 000999876 000999877 000999878 000999879 000999880
100019  000999881 000999882 000999883 000999884 000999885 000999886 000999887 000999888 000999889 000999890
100020  000999891 000999892 000999893 000999894 000999895 000999896 000999897 000999898 000999899 000999900
100021  000999901 000999902 000999903 000999904 000999905 000999906 000999907 000999908 000999909 000999910
100022  000999911 000999912 000999913 000999914 000999915 000999916 000999917 000999918 000999919 000999920
100023  000999921 000999922 000999923 000999924 000999925 000999926 000999927 000999928 000999929 000999930
100024  000999931 000999932 000999933 000999934 000999935 000999936 000999937 000999938 000999939 000999940
100025  000999941 000999942 000999943 000999944 000999945 000999946 000999947 000999948 000999949 000999950
100026  000999951 000999952 000999953 000999954 000999955 000999956 000999957 000999958 000999959 000999960
100027  000999961 000999962 000999963 000999964 000999965 000999966 000999967 000999968 000999969 000999970
100028  000999971 000999972 000999973 000999974 000999975 000999976 000999977 000999978 000999979 000999980
100029  000999981 000999982 000999983 000999984 000999985 000999986 000999987 000999988 000999989 000999990
100030  000999991 000999992 000999993 000999994 000999995 000999996 000999997 000999998 000999999 001000000
100031
100032  ----- Done testing FS_bonusI.sh -----
100033
```

## Bonus III :

RecursiveRemove 的做法已在 PartIII 的 filesys.cc 與 directory.cc 中說明。在此僅展示成果

**Test script:**

**FS_bonusIII.sh**

NachOS-4.0_MP4_goro > NachOS-4.0_MP4 > code > test > FS_bonusIII.sh

```
echo -e "=========== Testing FS_bonusIII.sh... ===========\n"
echo -e "\n----------- Formatting disk... ------------\n"
../build.linux/nachos -f
echo -e "\n----------- Mkdir /t0------------\n"
../build.linux/nachos -mkdir /t0
echo -e "\n----------- Mkdir /t1 ------------\n"
../build.linux/nachos -mkdir /t1
echo -e "\n----------- Mkdir /t2 ------------\n"
../build.linux/nachos -mkdir /t2
echo -e "\n----------- Recursive List  to check root dir ------------\n"
../build.linux/nachos -lr /
echo -e "\n----------- Test copy file ------------\n"
echo -e "\------------ Copy file num_100.txt to nachOS FS /t0/f1: ------------\n"
echo -e "Copy file num_100.txt to nachOS FS /t0/f1: \n"
../build.linux/nachos -cp num_100.txt /t0/f1
echo -e "\n----------- Mkdir /t0/aa ------------\n"
../build.linux/nachos -mkdir /t0/aa
echo -e "\n----------- Mkdir /t0/bb ------------\n"
../build.linux/nachos -mkdir /t0/bb
echo -e "\n----------- Mkdir /t0/cc ------------\n"
../build.linux/nachos -mkdir /t0/cc
echo -e "\n------------ Copy file num_100.txt to nachOS FS /t0/bb/f1: ------------\n"
echo -e "Copy file num_100.txt to nachOS FS /t0/bb/f1: \n"
../build.linux/nachos -cp num_100.txt /t0/bb/f1

echo -e "\n------------ Copy file num_100.txt to nachOS FS /t0/bb/f2: ------------\n"
# echo -e "Copy file num_100.txt to nachOS FS /t0/bb/f2: \n"
../build.linux/nachos -cp num_100.txt /t0/bb/f2
echo -e "\n------------ Copy file num_100.txt to nachOS FS /t0/bb/f3: ------------\n"
# echo -e "Copy file num_100.txt to nachOS FS /t0/bb/f3: \n"
```

```
echo -e "\n------------ Copy file num_100.txt to nachOS FS /t0/bb/f3: ------------\n"
# echo -e "Copy file num_100.txt to nachOS FS /t0/bb/f3: \n"
../build.linux/nachos -cp num_100.txt /t0/bb/f3
echo -e "\n------------ Copy file num_100.txt to nachOS FS /t0/bb/f4: ------------\n"
# echo -e "Copy file num_100.txt to nachOS FS /t0/bb/f4: \n"
../build.linux/nachos -cp num_100.txt /t0/bb/f4
echo -e "\n------------ Recursive List root dir /: ------------ \n"
../build.linux/nachos -lr /
echo -e "\n------------ Recursive Remove /t0:  --------------\n"
../build.linux/nachos -rr /t0
echo -e "\n------------ After Recursive Remove /t0: --------------\n"
echo "The root dir woule be:"
../build.linux/nachos -lr /
```

**Result:**

NachOS-4.0_MP4_goro > NachOS-4.0_MP4 > code > test > test_result > test_bonusIII.txt

```
========== Testing FS_bonusIII.sh... ==========


---------- Formatting disk... ------------

Total Num of Sectors: 512000
Need FreeMap Size: 64000
Need root Dir Size 1280
Formatting FS: allocating freemap dataSectors...
Formatting FS: allocating 'root directory' dataSectors...

---------- Mkdir /t0------------


---------- Mkdir /t1 ------------


---------- Mkdir /t2 ------------


---------- Recursive List  to check root dir ------------


[D] t0 (Empty directory)
[D] t1 (Empty directory)
[D] t2 (Empty directory)

---------- Test copy file ------------

\------------ Copy file num_100.txt to nachOS FS /t0/f1: ------------

Copy file num_100.txt to nachOS FS /t0/f1:


---------- Mkdir /t0/aa ------------

Existed dir: t0 its FCB at sector 529

---------- Mkdir /t0/bb ------------

Existed dir: t0 its FCB at sector 529

---------- Mkdir /t0/cc ------------

Existed dir: t0 its FCB at sector 529

------------ Copy file num_100.txt to nachOS FS /t0/bb/f1: ------------

Copy file num_100.txt to nachOS FS /t0/bb/f1:


------------ Copy file num_100.txt to nachOS FS /t0/bb/f2: ------------


------------ Copy file num_100.txt to nachOS FS /t0/bb/f3: ------------


------------ Copy file num_100.txt to nachOS FS /t0/bb/f4: ------------


------------ Recursive List root dir /: ------------

```

```
63
64  [D] t0
65    [F] f1
66    [D] aa  (Empty directory)
67    [D] bb
68      [F] f1
69      [F] f2
70      [F] f3
71      [F] f4
72    [D] cc  (Empty directory)
73  [D] t1 (Empty directory)
74  [D] t2 (Empty directory)
75
76  ------------ Recursive Remove /t0:  --------------
77
78  [F] f1
79  [D] aa
80  [D] bb
81  [D] cc
82  This directory is empty.
83  This directory is empty.
84  The directory is empty
85
86  ------------ After Recursive Remove /t0: --------------
87
88  The root dir woule be:
89
90  [D] t1 (Empty directory)
91  [D] t2 (Empty directory)
```

## ➢ 心得

**葉蓁：**

透過這次作業，體認到平常用爽爽的 File system，背後是多少辛苦的人幫我們建立好的 QQ 真的要帶著感恩的心!

平常隨便按個 ls, mkdir, touch, cd ..等等，輕輕鬆鬆按下 tab 鍵就會跳出可能的選項，OS 在背後幫我們跑了多少 search、traverse directory 和判斷的過程，真的感激不盡!

這次幾個容易忘記的點：

1. directory 和 fileheader 都需要先 FetchFrom disk 才能拿到正確的值，改動完後記得要 WriteBack 到 disk 中!不然下次讀出來的值就錯啦!

2. directory->FetchFrom(OpenFile*), OpenFile* 是已經打開的 directory 的“檔案"! 存著 directory 的 table 的內容!

nachOS 的 file system 預設會一直打開兩個檔案：bitmap 跟 root directory，因此所有的子 directory 都是從 root dir 往下長的，要開啟子 dir 的檔案，都要經過 root dir 去找: 去 disk 上 table[index].sector 開啟此 OpenFile，再把這個 OpenFile 的指標傳給 directory 讓他去 FetchFrom disk!

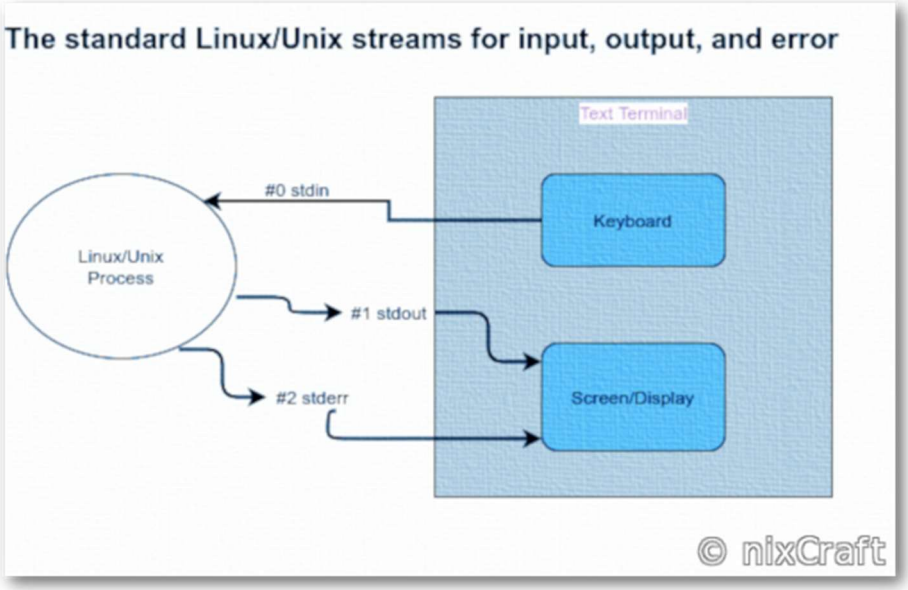
3.filedhr->FetchFrom(sector)，通常是用 dir 找到的那個 entry(table[i].sector)去把 fileheader(FCB /inode)的資料讀出來!

最後一次作業終於結束了! 放下重擔之感! 辛苦助教們這學期幫我們出作業、寫 spec、demo、看又臭又長的 report，還要一一解決我們寄信等等各種大小毛問題。真的非常感激! 也的確學到非常多東西

透過四次作業，了解到 system call, multiprogramming, virtual memory 的管理，CPU scheduling 以及 File system 的運作，os 真的做好多事情!! 還有很多事情是這學期沒時間用作業練習的(不過我大概也沒力氣了哈哈)，總之結論就是：帶著感恩的心使用電腦!!!

電腦當掉的時候不要怪他，可能是 deadlock，永遠記得一個觀念：
reboot 就好了! XD

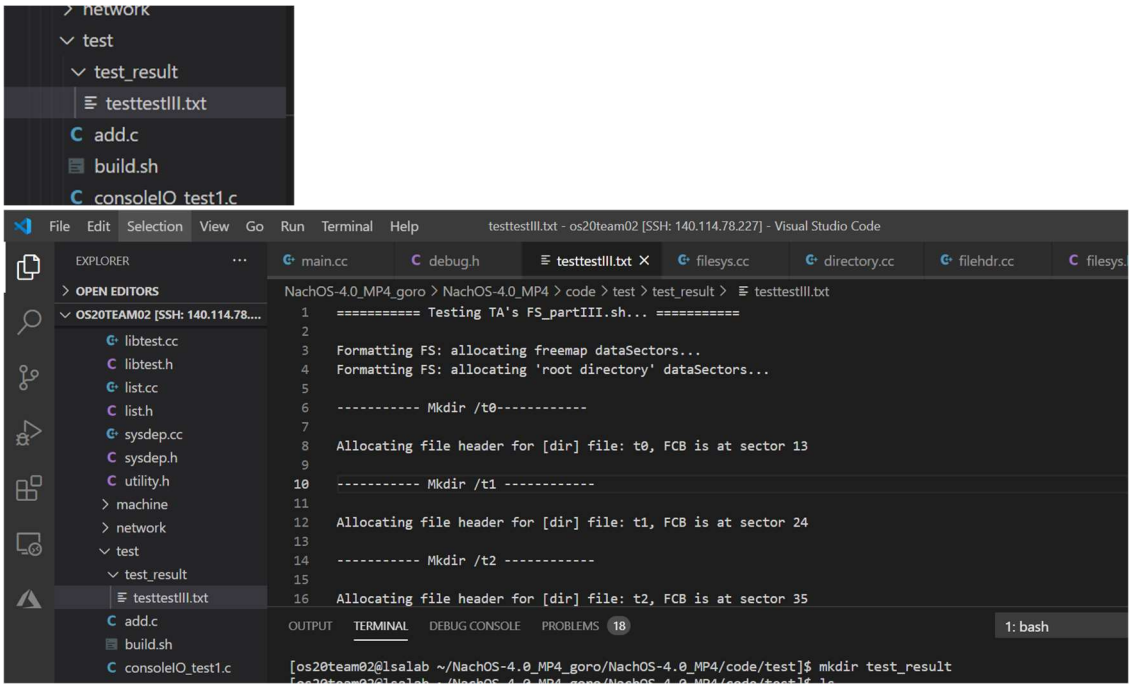
附註：這次發現一個好功能：
把 terminal output 存到 file 觀看
原本 Linux/Unix 預設 std output 會輸出到 terminal，若想存成檔案來看(改變 output streams)，可以用指令： command >> <filepath>




在測試測資時，只需要在 ./FS_partIII.sh 後面打: >> <file_path you want to save>
就會將 terminal output 全部 dump 到這個檔案，你可以再把那個結果的檔案打開
方便檢查錯誤。通常存成 xxx.txt。例如：

```
[os20team02@lsalab ~/NachOS-4.0_MP4_goro/NachOS-4.0_MP4/code/test]$ ./FS_partIII.sh >> test_result/testtestIII.txt
```

原本 stdout 的輸出就會變成 輸出到我們指定的 output file! 如下圖




**劉艾薇：**

本來不知道聽誰說（可能是老師）這次作業應該還好，還想說應該沒事，任由其他期末考延遲這次作業開始的時間，標準的 shortest deadline first，結果就是大崩潰，加上其實在上課的時候 file system 的地方自己就不是聽很懂了，現在想想都覺得到底是哪來的自信一直不開始做這次的 MP4，我真的是，哎呀呀

總之，感謝老師跟助教還有我們組長，覺得自己能撐到現在真的是奇蹟了，該來去放寒假了！希望助教們也能過一個愉快的寒假～