Part I. Understanding NachOS file system

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

   Class Bitmap and PersistentBitmap

   bitmap.h

   bitmap 維護一個 map，其存有哪個 sector 是 free 的資訊

   ```
   unsigned int *map;     // bit storage
   ```

   用 Mark 和 Clear 對 map 中的某個欄位做標記，mark 表該 sector 有人正在使用，clear 則是該 sector 無人正在使用

   ```
   void Mark(int which);      // Set the "nth" bit
   void Clear(int which);     // Clear the "nth" bit
   ```

   用 FindAndSet 找到無人使用的 sector 並回傳該 sector 位置，找不到則回傳-1

   ```
   int FindAndSet();    // Return the # of a clear bit, and as a side
                        // effect, set the bit.
                        // If no bits are clear, return -1.
   ```

   pbitmap.h

   PersistentBitmap 繼承 bitmap 並支援對 disk 的讀寫

   ```
   void FetchFrom(OpenFile *file);  // read bitmap from the disk
   void WriteBack(OpenFile *file);  // write bitmap contents to disk
   ```

   free map file 的建立

   filesys.cc

   1. new 一個 freeMap 紀錄哪個 sector 有被使用

      ```
      PersistentBitmap *freeMap = new PersistentBitmap(NumSectors);
      ```

   2. new 一個 mapHdr 為了可以在 disk 存放 freeMap file 的資料

      ```
      FileHeader *mapHdr = new FileHeader;
      ```

   3. 在 freeMap 上標記 sector 0 被 file header 使用

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

   4. 為 file header mapHdr 配置 data blocks

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

   5. 將 file header mapHdr 寫回去 sector FreeMapSector

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

   6. 打開 freeMapFile，並把 freeMap 寫到進去

      ```
      freeMapFile = new OpenFile(FreeMapSector);
      ```

   7. 之後 freeMapFile 保持 open 的狀態，直到他有被改變之後又要 WriteBack

      ```
      freeMap->WriteBack(freeMapFile);
      ```

   8. Delete freeMap、mapHdr 以確保不會有 memory leak

      ```
      delete freeMap;
      delete mapHdr;
      ```

bitmap 的 file header 放在 sector 0

filesys.h

```
#define FreeMapSector 0
```

2. What is the maximum disk size can be handled by the current Implementation? Explain why.

   因為有 32 個 track，且每個 track 有 32 個 sector，所以有 32*32 = 1024 個 sector。而每個 sector 大小為 128B，因此 maximum disk size 為 1024×128B = 2^10 × 2^7 = 2^17 B = 128 KB

disk.h

```
const int SectorSize = 128;   // number of bytes per disk sector
// number of sectors per disk track
const int SectorsPerTrack  = 32;
// number of tracks per disk 64*1024*1024/128/32
const int NumTracks = 32;
// total # of sectors per disk
const int NumSectors = (SectorsPerTrack * NumTracks);
```

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

Class Directory

Directory.h

Directory 維護一個 table，每個 entry 其存有<name, sector, inUse>，entry 中的 sector 為該 file 的 header 所存在在 raw disk 的 sector 位置

```
class DirectoryEntry {
  public:
    bool inUse;                 // Is this directory entry in use?
    int sector;                 // Location on disk to find the
                                // FileHeader for this file
    char name[FileNameMaxLen + 1]; // Text name for file, with +1
                                   // for the trailing '\0'
};
```

Add 和 Remove，新增或移除在目錄中的 file

```
// Add a subdir file into the directory
bool Add(char *name, int newSector, bool isDir);
// Remove a file from the directory
bool Remove(char *name);
```

directoryFile 的建立

Directory.cc

1. new 一個 directory 紀錄哪個 sector 有被使用

```
Directory *directory = new Directory(NumDirEntries);
```

2. new 一個 dirHdr 為了可以在 disk 存放 directory file 的資料

```
fileHeader *dirHdr = new FileHeader;
```

3. 在 directory 上標記 sector 1 被 file header 使用

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

4. 為 file header dirHdr 配置 data blocks

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

5. 將 file header dirHdr 寫回去 sector DirectorySector

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

6. 打開 directoryFile，並把 directory 寫到進去

```
directoryFile = new OpenFile(DirectorySector);
```

7. 之後 directoryFile 保持 open 的狀態，直到他有被改變之後又要 WriteBack

```
directory->WriteBack(directoryFile);
```

8. Delete freeMap、mapHdr 以確保不會有 memory leak

```
delete directory;
delete dirHdr;
```

Directory 的 file header 放在 sector 1

filesys.h

```
#define DirectorySector 1
```

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

file header(inode)

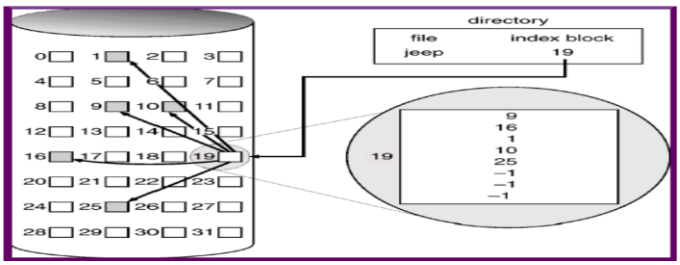
Data part

- **numBytes:** file 的 bytes
- **numSectors:** file 所需要的 sector 數
- **dataSectors:** 有 30 個 entry，表示該 datablock 所存在 raw disk 的 sector 位置

```
int numBytes;            // Number of bytes in the file
int numSectors;          // Number of data sectors in the file
// Disk sector numbers for each data
int dataSectors[NumDirect];
```

The disk allocation scheme of current implementation

原本的 allocation scheme 為 indexed，且只有用 directed blocks，如下方講義擷取圖



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

disk.h

```
const int SectorSize = 128;   // number of bytes per disk sector
```

filehdr.h

```
#define NumDirect ((SectorSize - 2 * sizeof(int)) / sizeof(int))
#define MaxFileSize (NumDirect * SectorSize)
```

SectorSize = 128
NumDirect = ((SectorSize - 2 × sizeof(int)) / sizeof(int)) = (128 - 2 × 4) / 4 = 30
Filesize = 30 × 128 =3840 B = 3.75 KB

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

1. Support file I/O system

   在 MP1 part2 我們已經有實作過 FILESYS_STUB 的 file I/O 相關的 system call interface，所以我們就按一樣的方法實作 FILESYS 的 Create、Open、Read、Write、Close 的 system call interface。

   但是為了符合 mp4 的 filesys 的 Create()有 initialSize 這個參數，所以在實作 Create 的 system call 的 interface 時統一都加上 initialSize 這個參數(exception.cc 會多從 register5 讀 initialSize 進來) 。

   另外，我們在 filesys.h 裡的 fileDescriptorTable 的 entry 數從原來的 20 變成 400，且多加了一個變數 openedNum，用來記錄現在有多少個檔案是打開的。openedNum 在 Open 時會++，在 Close 時會--

   filesys.h

   ```
   #define MAXFILENUM 400
   //fileID and openfile* map
   OpenFile* fileDescriptorTable[MAXFILENUM];
   int openedNum; // how many file be opended
   ```

   filesys.cc

   Check 是否開太多檔案

   ```
   if (openedNum == MAXFILENUM) // fileDescriptorTable has no space
   {
       cout << "   ---> Open fail\n\n";
       if (curDirFile != directoryFile)
           delete curDirFile; /* MP4 */
       delete directory;
       return make_pair((OpenFile *)NULL, -1);
   }
   ```

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

   利用 **linked index** 的方式來 support，link all the file header

   filehdr.h

   ```
   int nextFileHeaderSector;
   FileHeader* nextFileHeader;
   ```

   多加變數 nextFileHeader 用來指向下一個 header，nextFileHeaderSector 則是記錄下一個 header 存在哪一個 sector

   ```
   #define NumDirect ((SectorSize - 3 * sizeof(int)) / sizeof(int))
   ```

   因為在 header 中我們多加了一個 nextFileHeaderSector，所以再加上原本的 numBytes、numSectors，總共有 3 個 integer variable，因此只有 SectorSize - 3*sizeof(int)的空間給 dataSectors[NumDirect] 用

filehdr.cc

Constructor

```
FileHeader::FileHeader() {
    numBytes = -1;
    numSectors = -1;
    memset(dataSectors, -1, sizeof(dataSectors));
    nextFileHeader = NULL;
    nextFileHeaderSector = -1;
}
```

初始化

Destructor

```
FileHeader::~FileHeader() {
    if(nextFileHeader != NULL)
        delete nextFileHeader;
}
```

遞迴刪除 file header

Allocate

```
int FileHeader::Allocate(PersistentBitmap *freeMap, int fileSize) {
    int file_size = (int)(fileSize);
    int max_file_size = (int)(MaxFileSize);
    if(file_size > max_file_size){
        numBytes = max_file_size;
    }else{
        numBytes = file_size;
    }
    numSectors = divRoundUp(numBytes, SectorSize);
```

如果檔案需要的空間比 MaxFileSize (NumDirect * SectorSize)還大，那就代表這個 file 會把這個 header 的空間都用掉，所以把 numBytes 就是 MaxFileSize，之後再去計算需要多少 sectors

```
    if(freeMap->NumClear() < numSectors){
        return 0;
    }else{
        for(int i=0; i<numSectors; i++){
            dataSectors[i] = freeMap->FindAndSet();
            ASSERT(dataSectors[i] >= 0);
            // clean sector
            char clean[SectorSize];
            for(int j=0 ; j<SectorSize; j++)
                clean[j] = 0;
            kernel->synchDisk->WriteSector(dataSectors[i], clean);
        }
        if((file_size - max_file_size) > 0){
            nextFileHeaderSector = freeMap->FindAndSet();
            ASSERT(nextFileHeaderSector >= 0);
```

```
            nextFileHeader = new FileHeader;
            return SectorSize + nextFileHeader->Allocate(freeMap, file_size
- max_file_size);
        }
    }
    return SectorSize; // fileheader size
}
```

如果現今這個 header 的 freeMap 的剩餘 sector 數比需要的少就代表剩餘空間不夠 Allocate 失敗。若剩餘的 sector 數足夠就去 freeMap 裡找所需數量的對應空間並記錄 sector number 到 dataSector 裡，找完還要記得 initialize，把 sector 清乾淨後再 writeback 回去。

之後檢查若檔案大小比一個 header 的空間還大，那就代表需要下一個 header，因為這個 header 的 dataSectors 數不夠這個 file 用，所以先在 freeMap 裡先找塊空間存 nextFileHeaderSector，然後 new 一個新的 fileheader，再遞迴 allocate 來把剩下的檔案都 allocate 完

最後做完要 return file header size

Deallocate

```
void FileHeader::Deallocate(PersistentBitmap *freeMap) {
    for (int i = 0; i < numSectors; i++) {
        ASSERT(freeMap->Test((int)dataSectors[i])); // ought to
be marked!
        freeMap->Clear((int)dataSectors[i]);
    }
    if(nextFileHeader != NULL)
        nextFileHeader->Deallocate(freeMap);
}
```

遞迴 deallocate 把 linked list 上所有的 file header 的 dataSectors 都清空

FetchFrom

```
void FileHeader::FetchFrom(int sector) {
    char buf[SectorSize];
    kernel->synchDisk->ReadSector(sector, buf);
    int offset = sizeof(numBytes);
    memcpy(&numBytes, buf, sizeof(numBytes));
    memcpy(&numSectors, buf + offset, sizeof(numSectors));
    offset += sizeof(numSectors);
    memcpy(&nextFileHeaderSector,buf+offset, sizeof(nextFileHeaderSector));
    offset += sizeof(nextFileHeaderSector);
    memcpy(dataSectors, buf + offset, NumDirect * sizeof(int));
    if (nextFileHeaderSector != -1) {
        nextFileHeader = new FileHeader;
        nextFileHeader->FetchFrom(nextFileHeaderSector);
    }
}
```

Fetchfrom 是從 disk 把 file header 的 content 都拿出來，做法是把 numBytes, numSectors, nextFileHeaderSector, dataSector 依序拿出，然後用 nextFileSector 去 check 是否有下一個 file header，若有就遞迴去 fetchfrom

WriteBack

```
void FileHeader::WriteBack(int sector) {
    char buf[SectorSize];
    int offset = sizeof(numBytes);
    memcpy(buf, &numBytes, sizeof(numBytes));
    memcpy(buf + offset, &numSectors, sizeof(numSectors));
    offset += sizeof(numSectors);
    memcpy(buf+offset, &nextFileHeaderSector,sizeof(nextFileHeaderSector));
    offset += sizeof(nextFileHeaderSector);
    memcpy(buf + offset, dataSectors, NumDirect * sizeof(int));
    kernel->synchDisk->WriteSector(sector, buf);
    if (nextFileHeaderSector != -1) {
        nextFileHeader->WriteBack(nextFileHeaderSector);
    }
}
```

Writeback 就像是 Fetchfrom 的反向，是把 file header 的 content 都寫到 disk，做法是把 numBytes, numSectors, nextFileHeaderSector, dataSector 依序複製到一個 buffer 裡，然後用 WriteSector 把它寫回 disk，之後再用 nextFileSector 去 check 是否有下一個 file header，若有就遞迴去 Writeback，把所有的 file header 都寫回 disk

ByteToSector

```
int FileHeader::ByteToSector(int offset) {
    int sector = offset / SectorSize;
    if (sector < NumDirect) // within NumDirect
        return (dataSectors[sector]);
    else {
        return nextFileHeader->ByteToSector(offset - (int)MaxFileSize);
    }
}
```

找到 paticular byte 在 disk 上的哪一個 sector，先判斷這個 byte 需要的 sector 數是否超過 NumDirect，若是，就代表它是在 first file header 的 dataSectors[]範圍中，直接回傳對應的 sector。反之，遞迴去找正確的 sector number

FileLength

```
int FileHeader::FileLength() {
    int totalNumBytes = numBytes;

    if (nextFileHeader != NULL) {
        totalNumBytes += nextFileHeader->FileLength();
    }
```

```
    return totalNumBytes;
}
```

遞迴算出 file size

Print

```
void FileHeader::Print() {
    // numSectors total sectors
    int i, j, k;
    char *data = new char[SectorSize];
    // int tempNumSector = divRoundUp(numSectors, 32);
    printf("FileHeader contents. File size: %d. File blocks:\n",numBytes);
    for (i = 0; i < numSectors; i++) // dataIndex blocks
        printf("%d ", dataSectors[i]);
    printf("\nFile contents:\n");
    for (i = k = 0; i < numSectors; i++) {
        int dataIndex[numSectors - NumDirect];
        kernel->synchDisk->ReadSector(dataSectors[i], (char *)dataIndex);
        for (int l; l < numSectors - NumDirect; l++) {
            kernel->synchDisk->ReadSector(dataIndex[l], data);
            for (j = 0; (j < SectorSize) && (k < numBytes); j++, k++) {
                if ('\040' <= data[j] && data[j] <= '\176')
                    printf("%c", data[j]);
                else
                    printf("\\%x", (unsigned char)data[j]);
            }
            printf("\n");
        }
        printf("\n");
    }
    if (nextFileHeader != NULL)
        nextFileHeader->Print();
    delete[] data;
}
```

Recursively print the contents of the file header and the contents of all the data blocks pointed to by the file header.

Part III. Modify the file system code to support subdirectory

1. Support up to 64 files/subdirectories per directory

原本 NumDirEntries 10，每個目錄裡只能有 10 個 file，改成每個目錄能有 64 個 entry

```
// Support up to 64 files/subdirectories per directory
#define NumDirEntries 64
```

Subdirectory structure

class Directory and DirectoryEntry

原先的架構為只有一層根目錄，而現在需要改成有子目錄的架構，因此每個 entry 要紀錄著該 entry 是一般的 file，還是個 directory，作為區別。

```
class DirectoryEntry {
  public:
    bool isDir;                     // Is this entry is directory
    bool inUse;                     // Is this directory entry in use?
    int sector;                     // Location on disk to find the
                                    // FileHeader for this file
    char name[FileNameMaxLen + 1]; // Text name for file, with +1 for
                                    // the trailing '\0'
};
```

首先，為了因應 data structure 的改變，更改了 class Directory 原本的 function。

1. Add

在 Add 新的 file 進入目錄時，需要判斷該 file 是否為 directory，做 entry 的標記

```
bool Directory::Add(char *name, int newSector, bool isDir){
    if (FindIndex(name) != -1)
        return FALSE;

    for (int i = 0; i < tableSize; i++)
        if (!table[i].inUse) {
            table[i].isDir = (isDir == TRUE)?TRUE: FALSE;
            table[i].inUse = TRUE;
            strncpy(table[i].name, name, FileNameMaxLen);
            table[i].sector = newSector;
            return TRUE;
        }
    return FALSE; // no space.  Fix when we have extensible files.
}
```

2. List

原本的架構使得目錄只有一層，當加入子目錄的架構時，就不能照原本的單純尋訪每個 entry，這是因為現在的指令多了遞迴的 List。

實作方式：

參數部份

1. bool recursion，表示是否要遞迴的 List
2. int depth，表示現在距離根目錄的深度，可根據當前的深度印幾個 tab，以表示相對的深度

若是 recursion = FALSE，則按以往的作法依序尋訪每個 entry 即可，並印出他在哪個 entry、檔案名稱、屬性。
若是 recursion = TRUE，也是依序尋訪每個 entry，但當該 entry 記的是 directory，就必須遞迴下去，List subDirectory。
在 List subDirectory 之前，必須將子目錄的內容從 disk 讀出來，而他在 disk 的 sector 由他上一層的目錄 table 所紀錄

1. new directory for subDirectory
2. 打開對應的 subDirectory file
3. directory 讀取該 file 的內容(FetchFrom)，即可使用 directory 的 class
4. 使用完 directory 和 openfile，必須 delete 他們，以避免 memory leakage

```
void Directory::List(bool recursion, int depth) {
    for(int i=0; i<tableSize; i++){
        if(table[i].inUse == TRUE){
            for(int j=0; j<depth; j++)
                cout << "\t";

            cout << "[" << i << "] " << table[i].name << " ";
            if(table[i].isDir){
                cout << "D\n";
                if(recursion){
                    // fetch subdirectory from disk
                    Directory* subDir = new Directory(NumDirEntries);
                    OpenFile* subDirFile = new OpenFile(table[i].sector);
                    subDir->FetchFrom(subDirFile);

                    // recursion list directory
                    subDir->List(recursion, depth+1);

                    delete subDir;
                    delete subDirFile;
                }
            }else{
                cout << "F\n";
            }
        }
    }
}
```

3. IsDir

為了在 filesys.cc 實作方便，新增這個函式方便詢問該檔案是否為一個 directory file
實作方式：

參數部份

1. name，為查詢檔案名稱

利用 FindIndex 找到該 name 在目錄的第幾個 entry，並回傳該 entry 的 isDir

```
bool Directory::IsDir(char* name){
    int index = FindIndex(name);
    return table[index].isDir;
}
```

class Directory 的更改大致就到這裡，接著需要改 **filesys.cc**，由於目錄架構的改變，因此 class FileSystem 的 function 都有做些調整，和新增。

class FileSystem

新增部份

1. #define MAXFILENUM 400：至多只能有 400 個 file 被打開
2. fileDescriptorTable：一個 fileID 和 openfile*的對照表
3. FindSubDir

給定一串 path，能回傳該檔案的前一層目錄的 OpenFile*，並將傳入的 path 修改成 file name。像是說若是 path = /t0/f1，會回傳 t0 的 OpenFile*，並將傳入的 path 改成 f1。

實作方式：

參數部份

1. path，尋找其 subDirectory 的 file 路徑

為了解析由"/"所分隔的 path，我們使用了 strtok，每次可以得到其中一個子字串。我們先建立了 curDir 物件，表示現在在哪一層的目錄，首先必須從根目錄下去搜尋。

```
char *delim = "/";
char *token = strtok(subDirPath, delim);
char *nextToken = "";

OpenFile *curDirFile = directoryFile;
Directory *curDir = new Directory(NumDirEntries);
curDir->FetchFrom(directoryFile);
if (token == NULL) {
     delete curDir;
     return NULL;
}
```

一開始已經解析完第一個 token，接著先繼續解析下一個 token，之後的迴圈判斷說如果 nextToken 不是 NULL 且 token 的 file 是一個 directory file，才能繼續往下解析，因為至少也要解析到 token 那層目錄；跳出迴圈，則表示 token 為檔案名稱。

```
else {
        nextToken = strtok(NULL, delim);
        while (nextToken != NULL && curDir->IsDir(token)) {
            int sector = curDir->Find(token);
            if (sector == -1) {
                delete curDir;
                if (curDirFile != directoryFile)
                    delete curDirFile;
                return NULL;
            } else {
                if (curDirFile != directoryFile)
                    delete curDirFile;
                curDirFile = new OpenFile(sector);
                curDir->FetchFrom(curDirFile);
```

```
            }
            token = nextToken;
            nextToken = strtok(NULL, delim);
        }
        // end file
        strcpy(subDirPath, token);
        delete curDir;
        return curDirFile;
    }
```

調整部份

1. **Create**

實作方式：

參數部份

1. char *path，檔案創在哪個路徑上
2. int initialSize，檔案初始的大小
3. bool isDir，是否是個 directory file

要 create 的檔案若是 directory file，intialSize 統一改為 DirectoryFileSize。接著需要找到該檔案的前一層目錄，使用新增的 function FindSubDir，並從 disk 中 fetch 出來。

```
char targetPath[500];
strcpy(targetPath, path);
cout << "targetPath: " << targetPath << endl;
OpenFile *curDirFile = FindSubDir(targetPath);
if (curDirFile == NULL)
{
    delete directory;
    return FALSE;
}
directory->FetchFrom(curDirFile);
```

之後依序為 file header 找到 sector，加入到 direcotry 中，接著為 data blocks 找到足夠量的 sector，若是其中有一步的空間不足，則會 create fail，若沒有發生，最後要將有更改的內容全部寫回 disk 中。

2. **Open**

實作方式：

參數部份

1. char *path，開啟的檔案路徑

由於 kernel 那邊的 open 需要回傳的是 ID，所以我們將函式的回傳值改成 **pair<OpenFile *, OpenFileId>**，以滿足兩種需求。

找到該檔案的前一層目錄，使用新增的 function FindSubDir，並 disk 中 fetch 出來，接著在目錄 table 中找到該 file 的 file header 存在哪個 sector。

若是說當前以開啟的檔案數量已經達到最大值或是從該 sector new 一個 openfile 失敗，則都會開啟失敗。若中間都沒有失敗，接著在 fileDescriptorTable 找到空的 entry，存放新創的 openfile*，並將指標和 ID 打包回傳。

3. **Remove**

實作方式：

參數部份

1. char *path，開啟的檔案路徑

找到該檔案的前一層目錄，使用新增的 function FindSubDir，並 disk 中 fetch 出來，接著在目錄 table 中找到該 file 的 file header 存在哪個 sector。
file header 從 sector fetch 內容，做 Deallocate，清掉 data blocks 在 freeMap 原本標記使用的 sector，接著清掉 file header 佔用的 sector。若過程中沒有發生錯誤，最後要將有更改的內容全部寫回 disk 中。

4. **List**

實作方式：

參數部份

1. bool recursion，表示是否遞迴 List
2. char *dirPath，list 的路徑

若是要 List “/”，可以直接 call directory 的 List

```
if (strcmp(dirPath, "/") == 0)
{ // root directory
    Directory *directory = new Directory(NumDirEntries);
    directory->FetchFrom(directoryFile);
    cout << "List  \"/\"" << endl;
    directory->List(recursion, 0);
    delete directory;
    return;
}
```

若是其他種狀況，需要找到要 List 目錄的前一層目錄，使用新增的 function FindSubDir，並 disk 中 fetch 出來，接著在目錄 table 中找到該 directory 的 file header 存在哪個 sector。
從 sector new openfile，並新 new 一個 directory fetch 該 openfile，可以得到要 List 目錄的 directory 物件，call directory 的 List。

```
else
{
    char targetPath[500]; // FindSubDir will modified path into filename
    strcpy(targetPath, dirPath);

    // subDirFile is the directory containing the target directory file
    OpenFile *conDirFile = FindSubDir(targetPath);
    if (conDirFile == NULL) // no such dir
        return;
    Directory *conDir = new Directory(NumDirEntries);
    conDir->FetchFrom(conDirFile);

    int targetSector = conDir->Find(targetPath);
    ASSERT(targetSector >= 0);
    OpenFile *targetDirFile = new OpenFile(targetSector);
    Directory *targetDir = new Directory(NumDirEntries);
    targetDir->FetchFrom(targetDirFile);

    cout << "List \"" << targetPath << "\"" << endl;
    targetDir->List(recursion, 0);

    delete targetDirFile;
    delete targetDir;
    if (conDirFile != directoryFile)
        delete conDirFile; // not delete root directory file
    delete conDir;
}
```

Bonus I. Enhance the NachOS to support even larger file size

(1) Extend the disk from 128KB to 64MB

```
const int SectorSize = 128;                             // number of bytes per disk sector
const int SectorsPerTrack = 32;                         // number of sectors per disk track
const int NumTracks = 16384;                            // number of tracks per disk 64*1024*1024/128/32
const int NumSectors = (SectorsPerTrack * NumTracks); // total # of sectors per disk
```

原先 disk 最大只能支援 128KB 是因為 DiskSize = SectorSize*NumTracks*SectorsPerTrack 且 NumTracks=32 所以只能支援 128*32*32/1024=128KB，若要 extend disk 就要加大 NumTrack 或是 SectorsPerTrack。

我們選擇加大 NumTracks 成 64(MB)*1024(KB/MB)*1024(KB/B)/128/32=16384

(2) Support up to 64MB single file

我們原本的實作方式就可以 support

Bonus II. Recursive Operations on Directories

Support recursive remove of a directory

main.cc

```
if (removeFileName != NULL) {
    kernel->fileSystem->Remove(recursiveRemoveFlag,removeFileName);
}
```

傳 recursiveRemoveFlag 這個參數給 fileSystem->Remove

若下 nachos command "-rr" ，recursiveRemoveFlag=true，就要進行 recursive remove

fileSystem.cc

```
if (directory->IsDir(targetPath) && recursive)
{
    // fetch subdirectory from disk
    Directory *subDir = new Directory(NumDirEntries);
    OpenFile *subDirFile = new OpenFile(sector);
    subDir->FetchFrom(subDirFile);

    char targetPath[255];
    strcpy(targetPath, path);
    int offset = strlen(targetPath);
    targetPath[offset] = '/';

    // Remove all things in the target directory
    for (int i = 0; i < subDir->tableSize; i++)
    {
        if (subDir->table[i].inUse)
        {
            //update tragetPath
            strcpy(targetPath + offset + 1, subDir->table[i].name);
            Remove(recursive, targetPath);
        }
    }
    delete subDir;
    delete subDirFile;
}
```

先判斷要 remove 的這個 file 是不是 directory，如果是的話就要先幫 path 加上'/'，然後遞迴 remove 這個 directory 底下所有的 entry ，使用完 directory 和 openfile，必須 delete 他們，以避免 memory leakage

Result

```
========================================
Start removing file: bb
Start removing file: f2
   ---> Remove success

Start removing file: f3
   ---> Remove success

Start removing file: f4
   ---> Remove success

   ---> Remove success

List  "/"
[0] t0 D
        [0] f1 F
        [3] cc D
[1] t1 D
[2] t2 D
========================================
```

分工

104062101 劉芸瑄 code、report
104062121 陳品媛 code、report