OS team 26

CS342301: Operating System
MP4: File System

Team member & contribution:

I. **江佩霖 111062118**
   - **i.** Trace code
   - **ii.** Implement function
   - **iii.** 測試, Debug

II. **陳庭竣 111020025**
   - **i.** Trace code
   - **ii.** Implement function
   - **iii.** 測試, Debug

I. Understanding NachOS file system

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

i. Trace code:

```
FileSystem::FileSystem(bool format) // file system constructor
```

- 建構子由 kernel::initialize()呼叫，當下達指令-f 時，會將 fileSystem = new FileSystem(formatFlag);設置為 true，執行初始化硬碟的動作。
- 建構子根據 format 值決定是否要初始化 disk，由靜態成員變數 freeMapFile, directoryFile 存取 OpenFile 物件(分別為 sector 0 - FreeMapSector, 1 - DirectorySector)

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

- 當 format = true(初始化硬碟)，會呼叫 PersistentBitmap(NumSectors)初始化 bitmap。
- PersistentBitmap 大多繼承 Bitmap，NachOS 主要利用 bitmap 來記錄哪裡有 free block space。

```
Bitmap::Bitmap(int numItems) // Bitmap constructer
```

- Initialize a bitmap with "numItems" bits, so that every bit is clear.
- 建構子會先創一個 map，將使用情形紀錄在 map 中，0 表示該位置的 block 未被使用，1 則是已經被使用(初始化時 sectors 會先設置為 0)。

```
int Bitmap::FindAndSet()
```

- Return the number of the first bit which is clear.
- 呼叫 Test(int i)尋找首個未被占用的 bit，若迴圈結束還未找到空的 bit，回傳-1。
- 若 Test(int i)找到有 bit 未使用，則呼叫 Mark(int i)將此 bit 標示為使用並回傳。

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

- File system 建構子初始化會創建 FileHeader，透過 Mark()函數將 FreeMapSector(0)標示為使用，接著對 FileHeader 呼叫 Allocate()函數。

```
bool FileHeader::Allocate(PersistentBitmap *freeMap, int fileSize)
#define FreeMapFileSize (NumSectors / BitsInByte)
```

- 透過 Allocate()在 disk 上分配空間，初始化 FileHeader 的資訊。
- PersistentBitmap *freeMap 對應到每個區域的使用狀況，透過傳入的 fileSize 大小計算需要的 Sector 數量。
- 檢查區域中是否有足夠的區塊可以 Allocate，若有則透過迴圈分配磁碟區塊，記錄到 dataSectors 陣列中。

ii. Answer :

Manages and find free block space

- NachOS 主要利用 bitmap 管理 block space。
- 函數 FindAndSet()會根據 bitmap 中紀錄的使用情況來找到 free block space，並透過 Allocate()要到一定數量的 sector，並記錄在 dataSectors 陣列中。

Where is this information stored on the raw disk (which sector) ?

- 在檔案系統格式化（format）時，分配 freeMapFile，並初始化磁碟空間使用的位圖。
- freeMapFile 的 file header 放 sector 0 (FreeMapSector)
- Bitmap 的資訊存在 allocate 的 sector 2

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

i. Trace code :

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

- disk 包含 NumTracks(32) 個 tracks，每個 track 包含 SectorsPerTrack(32) 個 sectors。各個 sector 的大小為 SectorSize(128) bytes

```
const int MagicNumber = 0x456789ab;
const int MagicSize = sizeof(int);
const int DiskSize = (MagicSize + (NumSectors * SectorSize));
```

- NachOS DiskSize = (MagicSize + (NumSectors * SectorSize))

ii. Answer

- NachOS maximum disk size = (sizeof(int) + (32 * 32 * 128)) , 128KB

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

i. Trace code :

```
Directory *directory = new Directory(NumDirEntries);
Directory::Directory(int size){
    table = new DirectoryEntry[size];
    /* skip */
}
```

- 硬碟初始化時，會呼叫 new Directory 來儲存 filename。
  - NachOS 把 directory 當成一個 file 來管理，有 header 和 dataSectors。Directory 用來記錄檔案名稱和檔案 fileHeader 位置。
  - Directory 的建構子中創建了 DirectoryEntry 陣列, 記錄每個 entry 的使用狀態, 紀錄檔案的名稱與 file header sector, 以維護 file name。

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

  - File system 建構子初始化會創建 FileHeader，透過 Mark()函數將 DirectorySector(1) 標示為使用，接著對 FileHeader 呼叫 Allocate()函數。

```
bool FileHeader::Allocate(PersistentBitmap *freeMap, int fileSize)
#define DirectoryFileSize (sizeof(DirectoryEntry) * NumDirEntries)
```

- 透過 Allocate()在 disk 上分配空間給 directory。

ii. Answer :

Manages directory data structure

  - 主要作為 file 來管理，有 header 和 dataSectors。
  - directoryFile 的 file header 存在 DirectorySector 中(sector 1)。

Where is this information stored on the raw disk (which sector) ?

  - 在檔案系統格式化（format）時，分配 freeMapFile，並初始化磁碟空間使用的位圖。

- directoryFile 的 file header 放 sector 1 (DirectorySector)
- Bitmap 的資訊存在 allocate 的 sector 2

- directory 的資訊存在 allocate 的 sector 3、4 (根據 #define，需要 20*10 = 200 bytes = 兩個 sector)

4. What information is stored in an inode? Use a figure to illustrate the disk allocation scheme of the current implementation.

i. Trace code :

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

- FileHeader 的 private variable 說明了 i-node 的資訊。

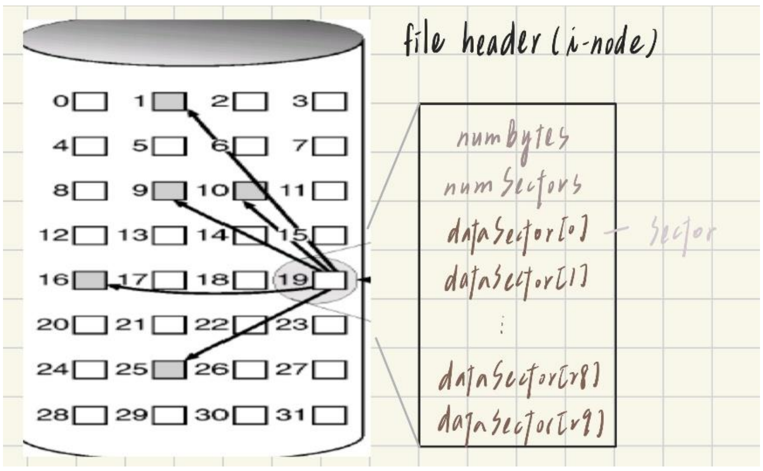
```
#define NumDirect ((SectorSize - 2 * sizeof(int)) / sizeof(int))
```

ii. Answer :

- 最多可以存放 NumDirect 個 sector。
- 因 file header 只存在 1 個 sector, 減去 numBytes 和 numSectors 兩變數的空間後

(128-2*4)/4=30

最多可以存到 30 個 sector



5. What is the maximum file size that can be handled by the current implementation? Explain why

i. Trace code :

```
#define MaxFileSize (NumDirect * SectorSize)
```

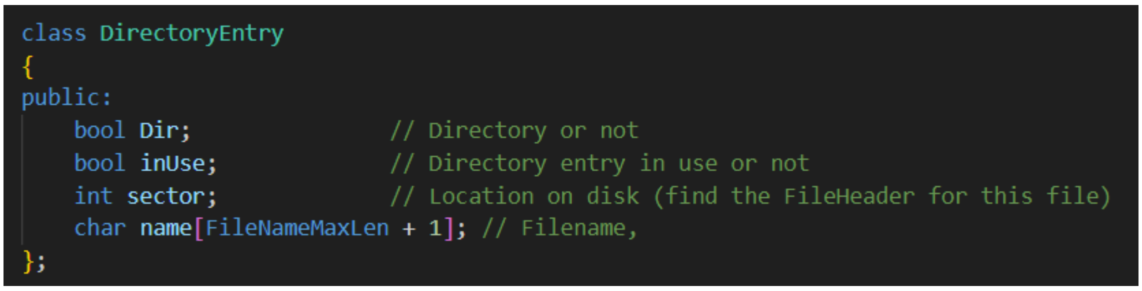
- 目前 file fileHeader 只有單層(從 Allocate()的迴圈可看出)，且一個 fileHeader 能使用的空間僅 32 sector 大小。
- 扣除 numBytes, numSectors 變數空間，僅有 30sector 可使用(同問題 4)。

ii. Answer :

- 30*128 = 4KB 左右(3.75KB)

II. Implement the I/O system calls in NachOS

1. Part I : Modify the file system code to support file I/O system calls and larger file size

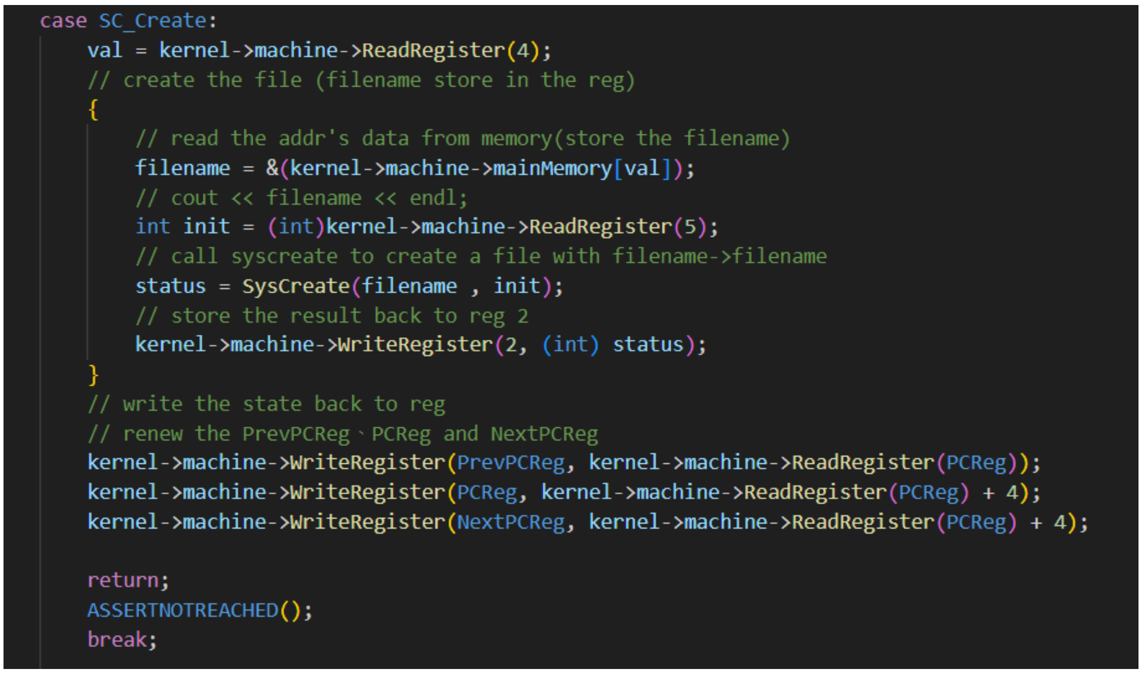


```
class DirectoryEntry
{
public:
    bool Dir;                  // Directory or not
    bool inUse;                // Directory entry in use or not
    int sector;                // Location on disk (find the FileHeader for this file)
    char name[FileNameMaxLen + 1]; // Filename,
};
```

Directory table 為一個 DirectoryEntry object，內容包含 Dir(紀錄是否為 directory)、inUse(紀錄此 entry 是否已被使用)、sector(紀錄該 file/directory 在 disk 上的位置)、name(紀錄該 file/directory 的名稱)。

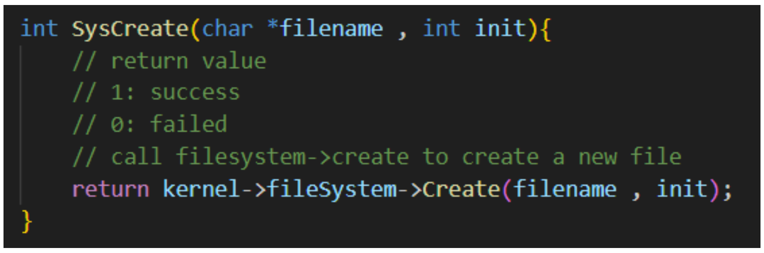
(1) Combine your MP1 file system call interface with NachOS FS to implement five system calls:

Create



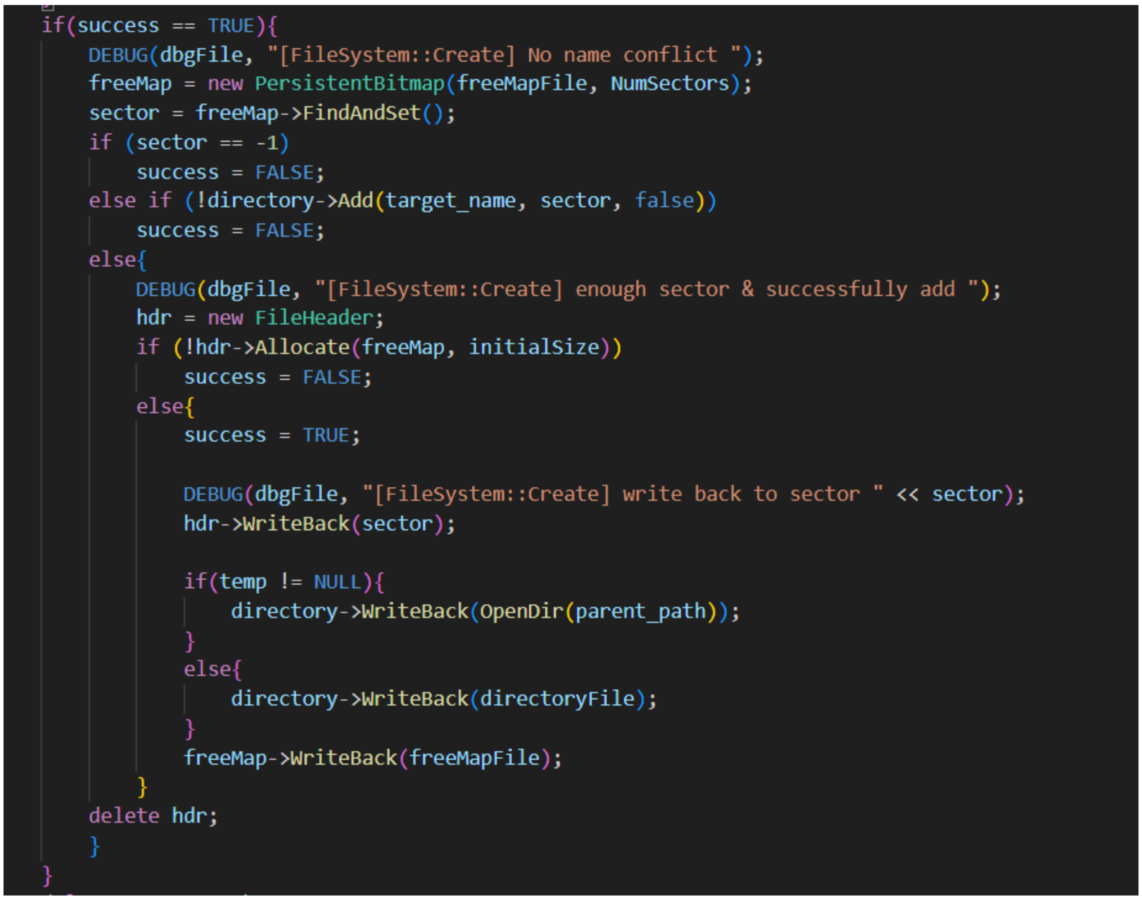
```
    case SC_Create:
        val = kernel->machine->ReadRegister(4);
        // create the file (filename store in the reg)
        {
            // read the addr's data from memory(store the filename)
            filename = &(kernel->machine->mainMemory[val]);
            // cout << filename << endl;
            int init = (int)kernel->machine->ReadRegister(5);
            // call syscreate to create a file with filename->filename
            status = SysCreate(filename , init);
            // store the result back to reg 2
            kernel->machine->WriteRegister(2, (int) status);
        }
        // write the state back to reg
        // renew the PrevPCReg、PCReg and NextPCReg
        kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
        kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
        kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);

        return;
        ASSERTNOTREACHED();
        break;
```

➢ 在 ExceptionHandler()中，複製 MP1 實作的部分，呼叫 Syscreate()。



```
int SysCreate(char *filename , int init){
    // return value
    // 1: success
    // 0: failed
    // call filesystem->create to create a new file
    return kernel->fileSystem->Create(filename , init);
}
```

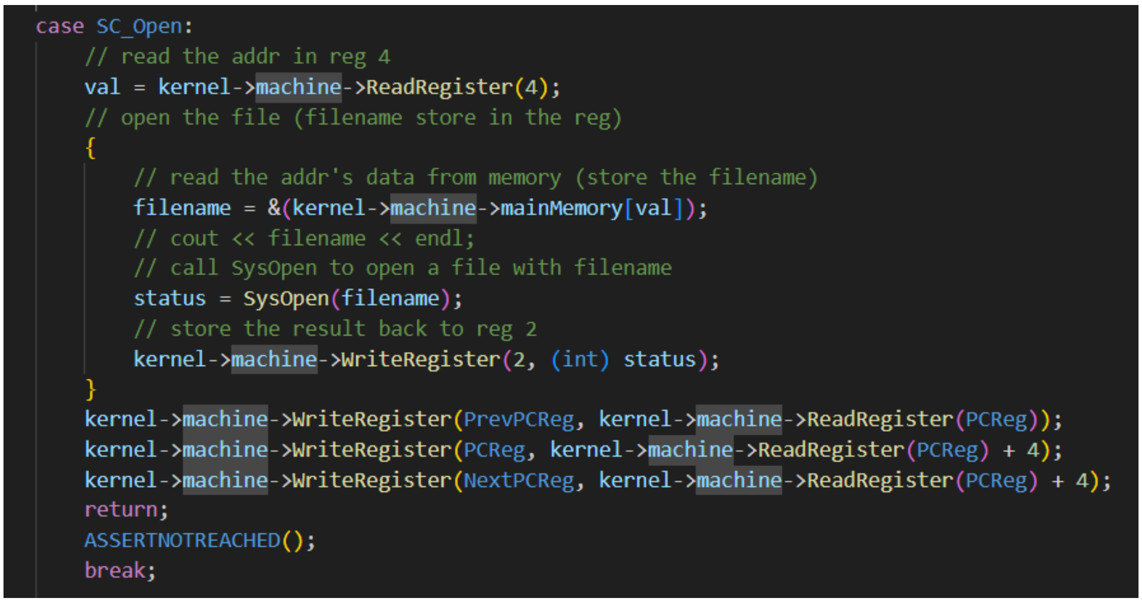
➢ 在 ksyscal.h 的 Syscreate()中，呼叫 Create()。



```
if(success == TRUE){
    DEBUG(dbgFile, "[FileSystem::Create] No name conflict ");
    freeMap = new PersistentBitmap(freeMapFile, NumSectors);
    sector = freeMap->FindAndSet();
    if (sector == -1)
        success = FALSE;
    else if (!directory->Add(target_name, sector, false))
        success = FALSE;
    else{
        DEBUG(dbgFile, "[FileSystem::Create] enough sector & successfully add ");
        hdr = new FileHeader;
        if (!hdr->Allocate(freeMap, initialSize))
            success = FALSE;
        else{
            success = TRUE;

            DEBUG(dbgFile, "[FileSystem::Create] write back to sector " << sector);
            hdr->WriteBack(sector);

            if(temp != NULL){
                directory->WriteBack(OpenDir(parent_path));
            }
            else{
                directory->WriteBack(directoryFile);
            }
            freeMap->WriteBack(freeMapFile);
        }
    delete hdr;
    }
}
```

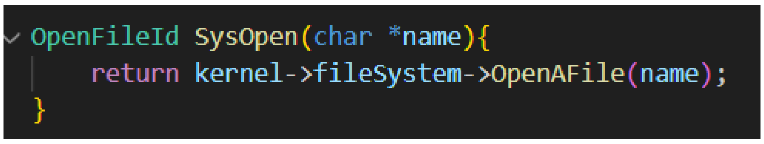
➢ 在 filesys.cc 的 Create()中，完成檔案創建動作。

Open



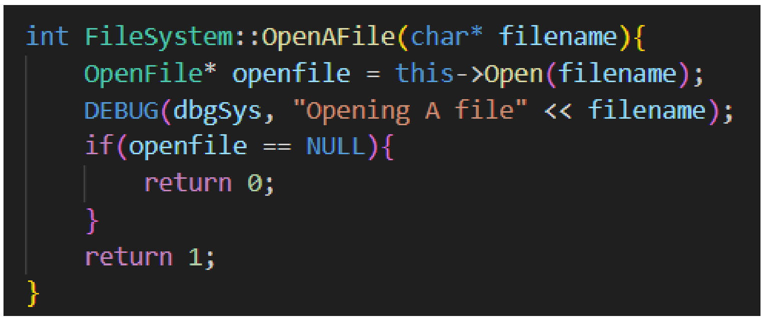
```
case SC_Open:
    // read the addr in reg 4
    val = kernel->machine->ReadRegister(4);
    // open the file (filename store in the reg)
    {
        // read the addr's data from memory (store the filename)
        filename = &(kernel->machine->mainMemory[val]);
        // cout << filename << endl;
        // call SysOpen to open a file with filename
        status = SysOpen(filename);
        // store the result back to reg 2
        kernel->machine->WriteRegister(2, (int) status);
    }
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
    return;
    ASSERTNOTREACHED();
    break;
```

➢ 在 ExceptionHandler()中，複製 MP1 實作的部分，呼叫 SysOpen()。



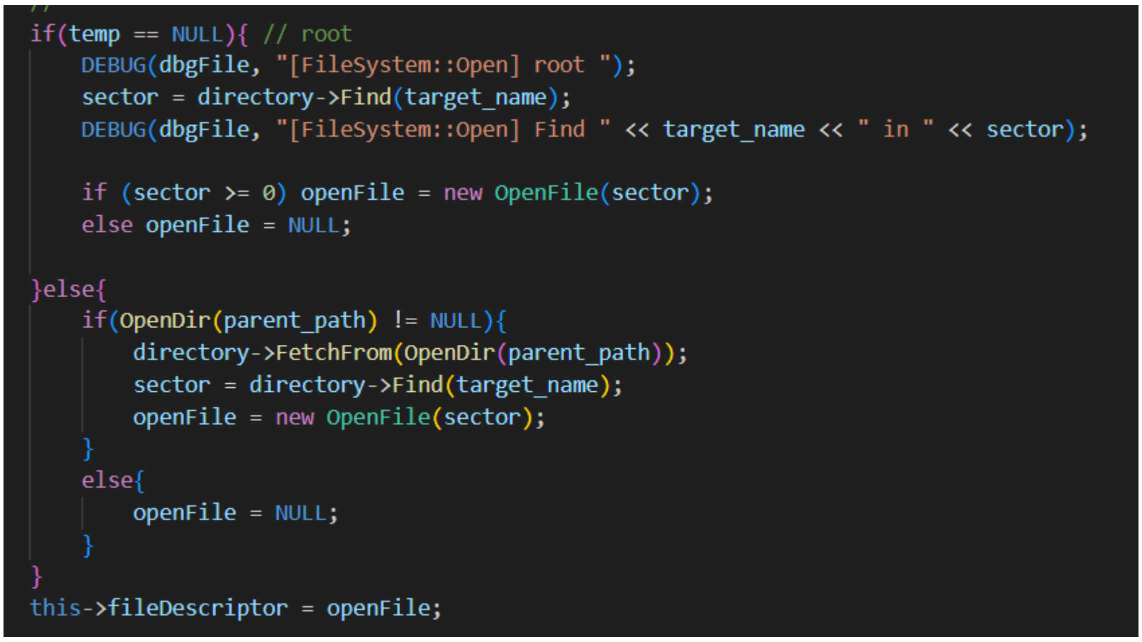
```
OpenFileId SysOpen(char *name){
    return kernel->fileSystem->OpenAFile(name);
}
```

➢ 在 ksyscal.h 的 SysOpen()中，呼叫 OpenAFile()。



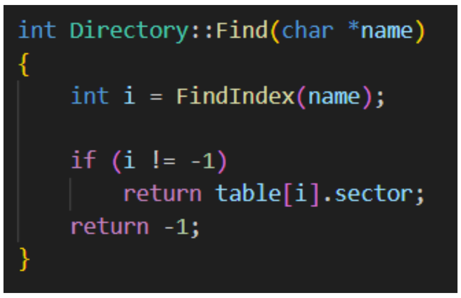
```
int FileSystem::OpenAFile(char* filename){
    OpenFile* openfile = this->Open(filename);
    DEBUG(dbgSys, "Opening A file" << filename);
    if(openfile == NULL){
        return 0;
    }
    return 1;
}
```

➢ 在 filesys.cc 的 OpenAFile()中，呼叫 Open()。

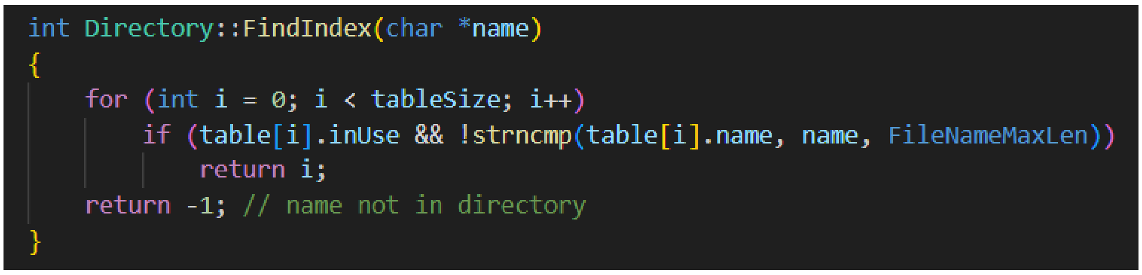


```
if(temp == NULL){ // root
    DEBUG(dbgFile, "[FileSystem::Open] root ");
    sector = directory->Find(target_name);
    DEBUG(dbgFile, "[FileSystem::Open] Find " << target_name << " in " << sector);

    if (sector >= 0) openFile = new OpenFile(sector);
    else openFile = NULL;

}else{
    if(OpenDir(parent_path) != NULL){
        directory->FetchFrom(OpenDir(parent_path));
        sector = directory->Find(target_name);
        openFile = new OpenFile(sector);
    }
    else{
        openFile = NULL;
    }
}
this->fileDescriptor = openFile;
```

➢ 在 Open()中，若 sector 回傳值為 -1，代表該檔案不存在，回傳 NULL；若不為 -1，則傳入 sector number，並 new 一個 OpenFile。(Open()實作過程中會呼叫 Find())。



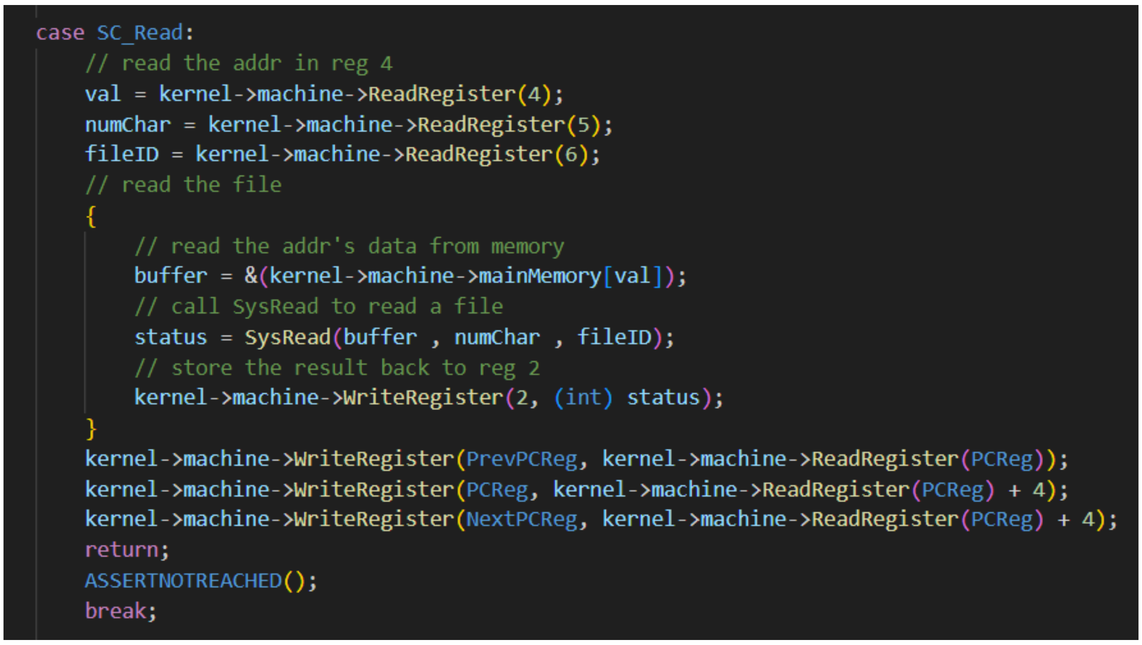
```
int Directory::Find(char *name)
{
    int i = FindIndex(name);

    if (i != -1)
        return table[i].sector;
    return -1;
}
```



```
int Directory::FindIndex(char *name)
{
    for (int i = 0; i < tableSize; i++)
        if (table[i].inUse && !strncmp(table[i].name, name, FileNameMaxLen))
            return i;
    return -1; // name not in directory
}
```

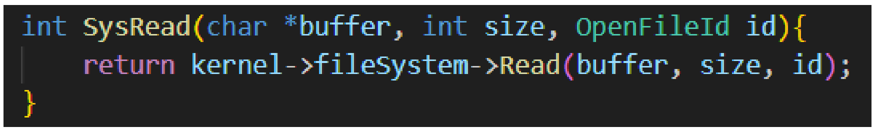
➢ 在 Find()中，會呼叫 FindIndex()，此 function 可以找到名稱符合傳入名稱的 entry，並回傳該 entry 的 index，進而回傳該檔案名稱對應的 sector。

Read



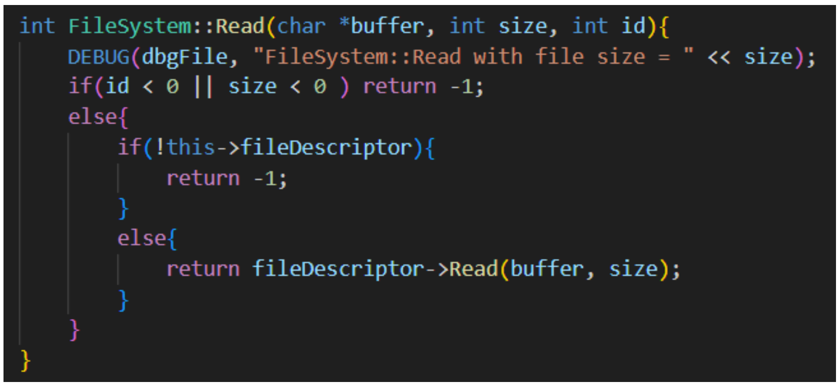
```
case SC_Read:
    // read the addr in reg 4
    val = kernel->machine->ReadRegister(4);
    numChar = kernel->machine->ReadRegister(5);
    fileID = kernel->machine->ReadRegister(6);
    // read the file
    {
        // read the addr's data from memory
        buffer = &(kernel->machine->mainMemory[val]);
        // call SysRead to read a file
        status = SysRead(buffer , numChar , fileID);
        // store the result back to reg 2
        kernel->machine->WriteRegister(2, (int) status);
    }
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
    return;
    ASSERTNOTREACHED();
    break;
```

- 在 ExceptionHandler()中，複製 MP1 實作的部分，呼叫 SysRead()。



```
int SysRead(char *buffer, int size, OpenFileId id){
    return kernel->fileSystem->Read(buffer, size, id);
}
```

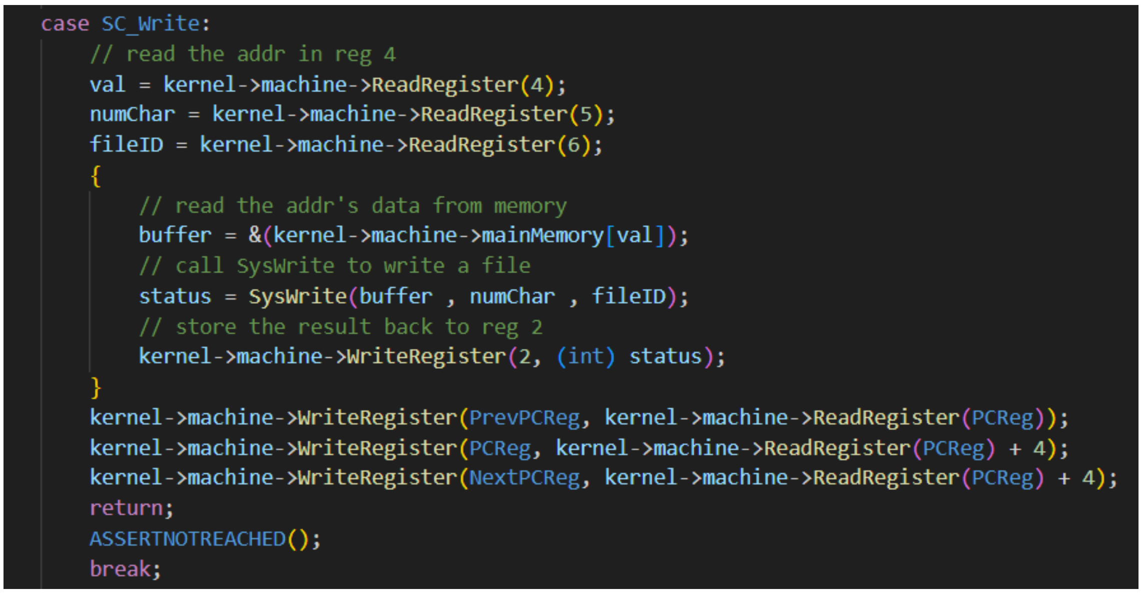
- 在 ksyscal.h 的 SysRead()中，呼叫 Read()。



```
int FileSystem::Read(char *buffer, int size, int id){
    DEBUG(dbgFile, "FileSystem::Read with file size = " << size);
    if(id < 0 || size < 0 ) return -1;
    else{
        if(!this->fileDescriptor){
            return -1;
        }
        else{
            return fileDescriptor->Read(buffer, size);
        }
    }
}
```

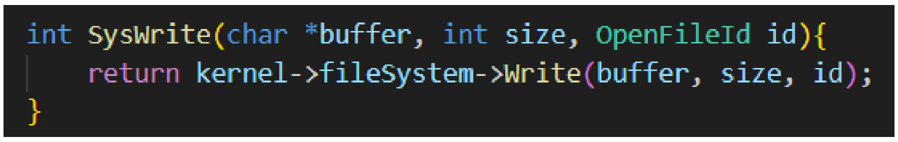
- 在 filesys.cc 的 Read()中，呼叫 Read()，以完成檔案讀取動作。

Write



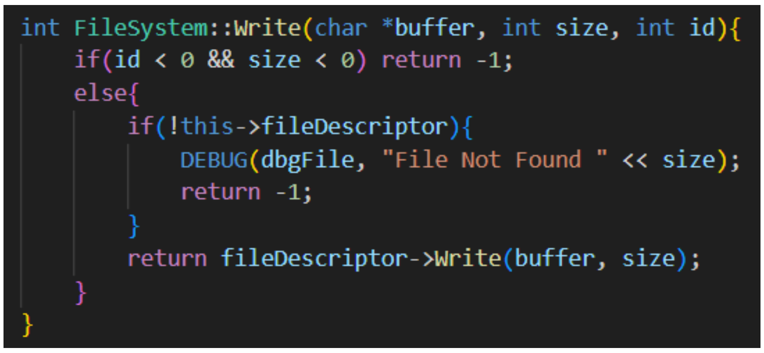
```
case SC_Write:
    // read the addr in reg 4
    val = kernel->machine->ReadRegister(4);
    numChar = kernel->machine->ReadRegister(5);
    fileID = kernel->machine->ReadRegister(6);
    {
        // read the addr's data from memory
        buffer = &(kernel->machine->mainMemory[val]);
        // call SysWrite to write a file
        status = SysWrite(buffer , numChar , fileID);
        // store the result back to reg 2
        kernel->machine->WriteRegister(2, (int) status);
    }
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
    return;
    ASSERTNOTREACHED();
    break;
```

➢ 在 ExceptionHandler()中，複製 MP1 實作的部分，呼叫 SysWrite()。



```
int SysWrite(char *buffer, int size, OpenFileId id){
    return kernel->fileSystem->Write(buffer, size, id);
}
```

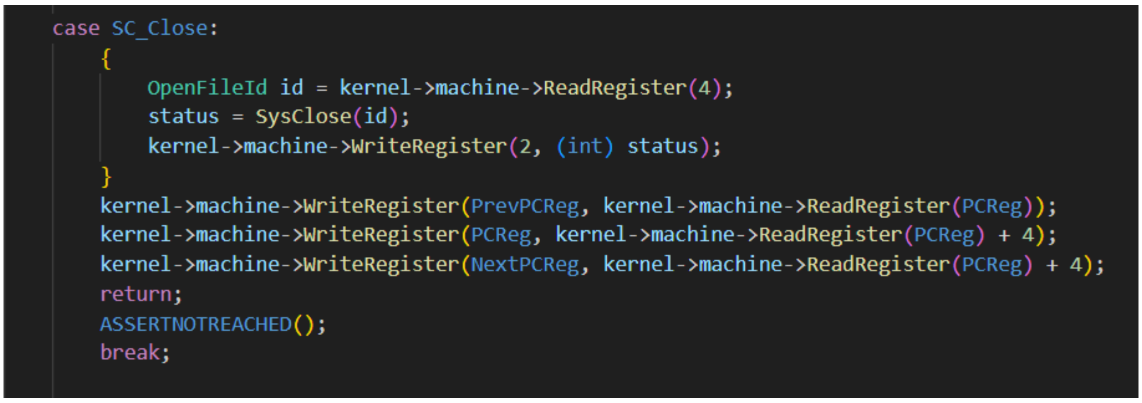
➢ 在 ksyscal.h 的 SysWrite()中，呼叫 Write()。



```
int FileSystem::Write(char *buffer, int size, int id){
    if(id < 0 && size < 0) return -1;
    else{
        if(!this->fileDescriptor){
            DEBUG(dbgFile, "File Not Found " << size);
            return -1;
        }
        return fileDescriptor->Write(buffer, size);
    }
}
```

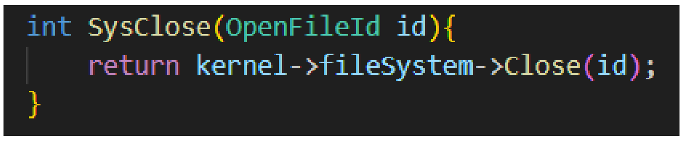
➢ 在 filesys.cc 的 Write()中，呼叫 Write()，以完成檔案寫入動作。

Close



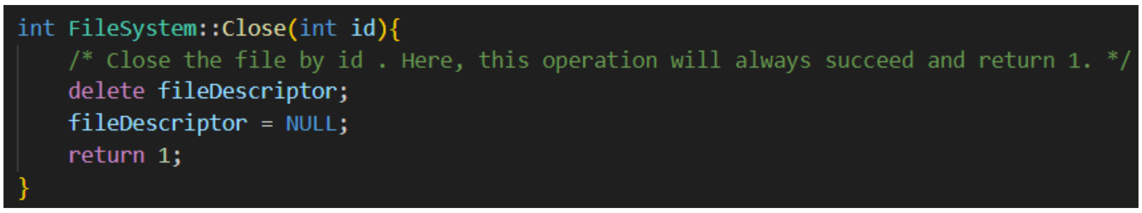
```
case SC_Close:
    {
        OpenFileId id = kernel->machine->ReadRegister(4);
        status = SysClose(id);
        kernel->machine->WriteRegister(2, (int) status);
    }
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
    return;
    ASSERTNOTREACHED();
    break;
```

➢ 在 ExceptionHandler()中，複製 MP1 實作的部分，呼叫 SysClose()。



```
int SysClose(OpenFileId id){
    return kernel->fileSystem->Close(id);
}
```

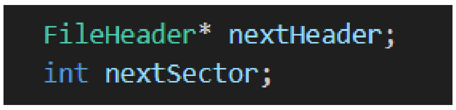
- 在 ksyscal.h 的 SysClose()中，呼叫 Close()。



```
int FileSystem::Close(int id){
    /* Close the file by id . Here, this operation will always succeed and return 1. */
    delete fileDescriptor;
    fileDescriptor = NULL;
    return 1;
}
```

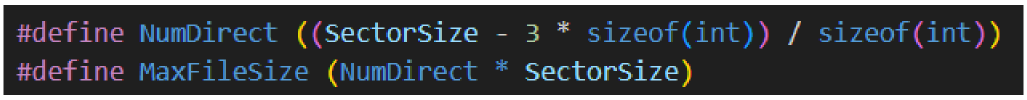
- 在 filesys.cc 的 Close()中，以完成檔案關閉動作。

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



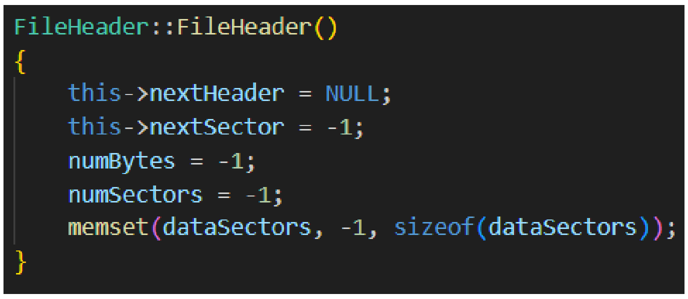
```
FileHeader* nextHeader;
int nextSector;
```

- 在 FileHeader class 中定義兩個 private 變數，分別為 nextHeader(link list 中，連接下一個 FileHeader object 的 pointer)及 nextSector(紀錄下一個 FileHeader object 的 sector number)。



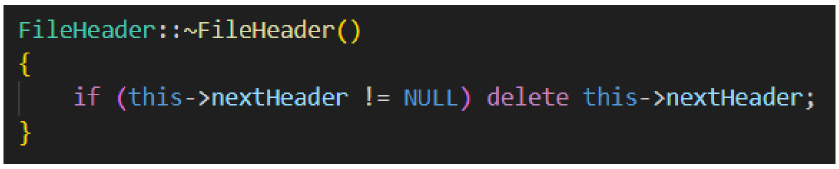
```
#define NumDirect ((SectorSize - 3 * sizeof(int)) / sizeof(int))
#define MaxFileSize (NumDirect * SectorSize)
```

- 更改 NumDirect 值，因每個 FIleHeader object 的 dataSector 大小固定，此時需要多存 nextSector，故須調整 NumDirect。



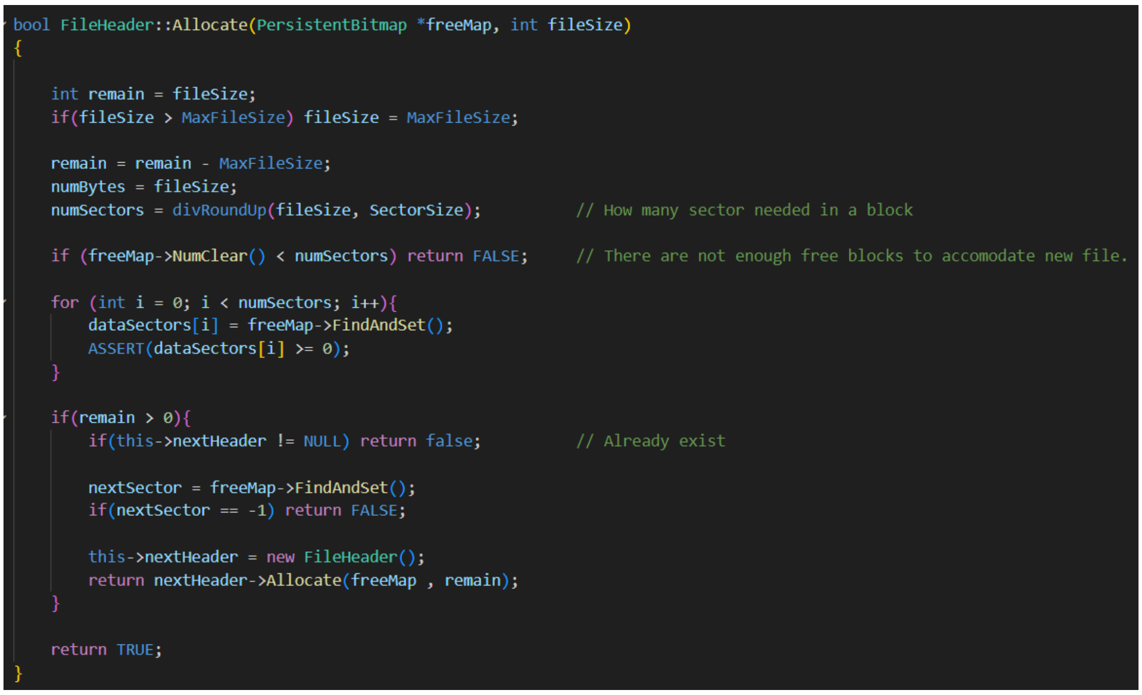
```
FileHeader::FileHeader()
{
    this->nextHeader = NULL;
    this->nextSector = -1;
    numBytes = -1;
    numSectors = -1;
    memset(dataSectors, -1, sizeof(dataSectors));
}
```

- 在 FileHeader()中，initialize nextHeader 及 numSector。



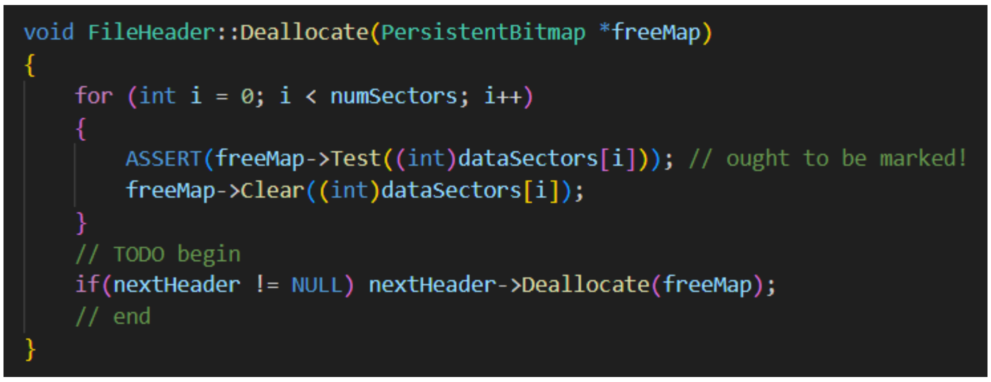
```
FileHeader::~FileHeader()
{
    if (this->nextHeader != NULL) delete this->nextHeader;
}
```

- 在~FileHeader()中，destruct nextHeader。



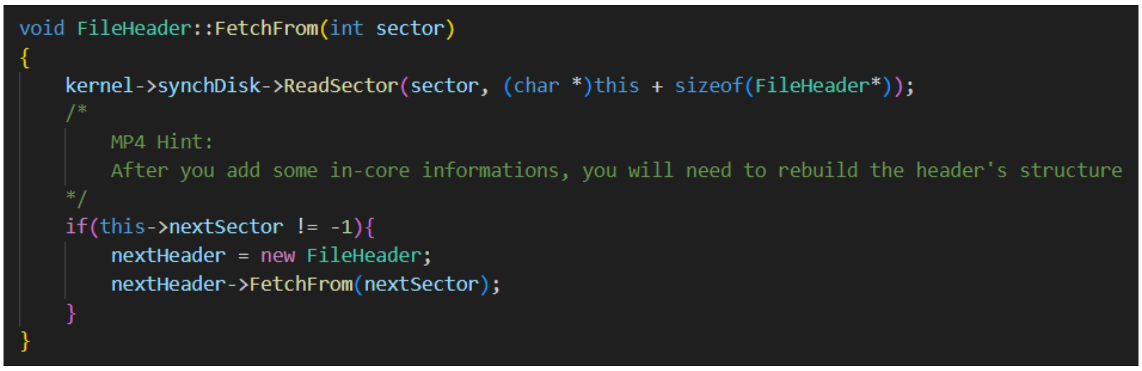
```
bool FileHeader::Allocate(PersistentBitmap *freeMap, int fileSize)
{

    int remain = fileSize;
    if(fileSize > MaxFileSize) fileSize = MaxFileSize;

    remain = remain - MaxFileSize;
    numBytes = fileSize;
    numSectors = divRoundUp(fileSize, SectorSize);          // How many sector needed in a block

    if (freeMap->NumClear() < numSectors) return FALSE;     // There are not enough free blocks to accomodate new file.

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

    if(remain > 0){
        if(this->nextHeader != NULL) return false;          // Already exist

        nextSector = freeMap->FindAndSet();
        if(nextSector == -1) return FALSE;

        this->nextHeader = new FileHeader();
        return nextHeader->Allocate(freeMap , remain);
    }

    return TRUE;
}
```

➢ 在 Allocate()中，需要檢測 fileSize 是否大於 MaxFileSize，若大於，代表檔案大小過大，需要用到下一個 FileHeader，因此，遞回呼叫 Allocate()直到分配完成。



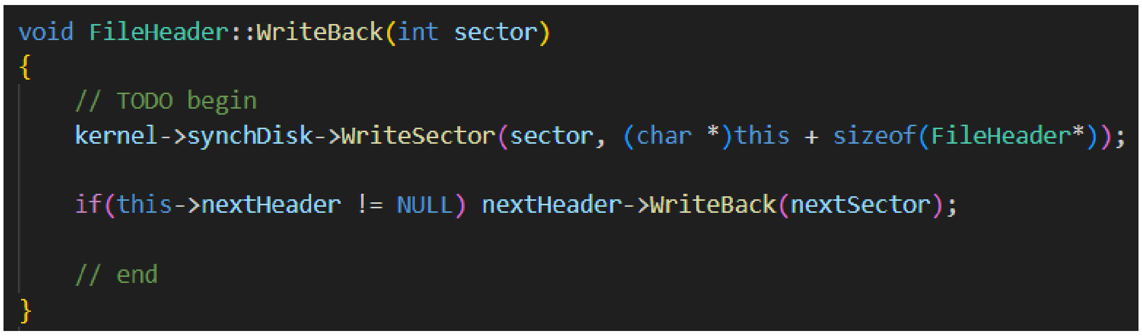
```
void FileHeader::Deallocate(PersistentBitmap *freeMap)
{
    for (int i = 0; i < numSectors; i++)
    {
        ASSERT(freeMap->Test((int)dataSectors[i])); // ought to be marked!
        freeMap->Clear((int)dataSectors[i]);
    }
    // TODO begin
    if(nextHeader != NULL) nextHeader->Deallocate(freeMap);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

➢ 在 Deallocate 中，會把所有使用到的 nextHeader 遞迴 deallocate。



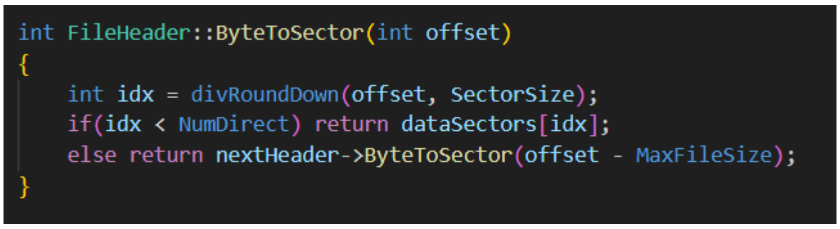
```
void FileHeader::FetchFrom(int sector)
{
    kernel->synchDisk->ReadSector(sector, (char *)this + sizeof(FileHeader*));
    /*
        MP4 Hint:
        After you add some in-core informations, you will need to rebuild the header's structure
    */
    if(this->nextSector != -1){
        nextHeader = new FileHeader;
        nextHeader->FetchFrom(nextSector);
    }
}
```

➢ 在 FetchFrom 中，若有 nextHeader，遞迴呼叫將 fileHeader 內容從 disk 獲取出來。



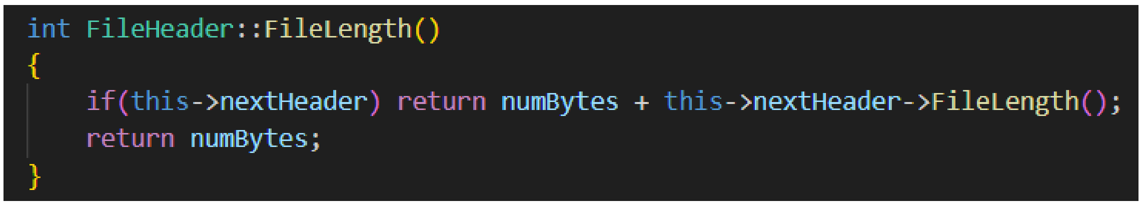
```
void FileHeader::WriteBack(int sector)
{
    // TODO begin
    kernel->synchDisk->WriteSector(sector, (char *)this + sizeof(FileHeader*));

    if(this->nextHeader != NULL) nextHeader->WriteBack(nextSector);

    // end
}
```

➢ 在 WriteBack 中，若有 nextHeader，遞迴呼叫將更改資料寫回 disk。



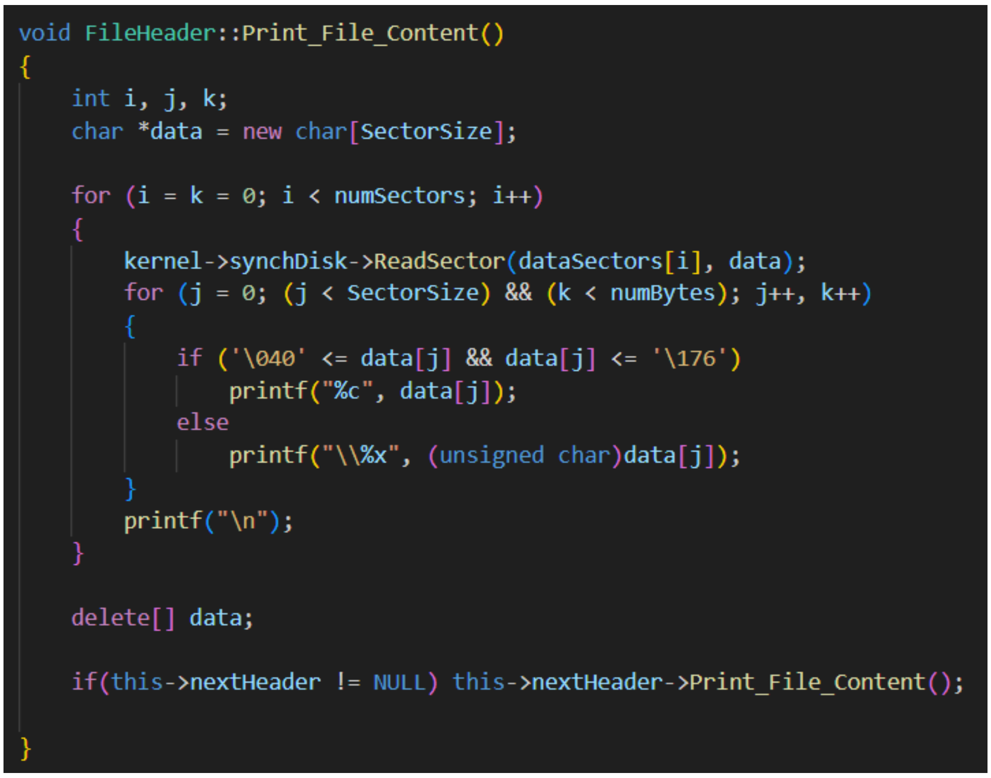
```
int FileHeader::ByteToSector(int offset)
{
    int idx = divRoundDown(offset, SectorSize);
    if(idx < NumDirect) return dataSectors[idx];
    else return nextHeader->ByteToSector(offset - MaxFileSize);
}
```

➢ 在 ByteToSector 中，若 idx < NumDirect，代表 offset 範圍在 link list 的 head，回傳 dataSectors[idx]，否則進行遞迴呼叫。



```
int FileHeader::FileLength()
{
    if(this->nextHeader) return numBytes + this->nextHeader->FileLength();
    return numBytes;
}
```

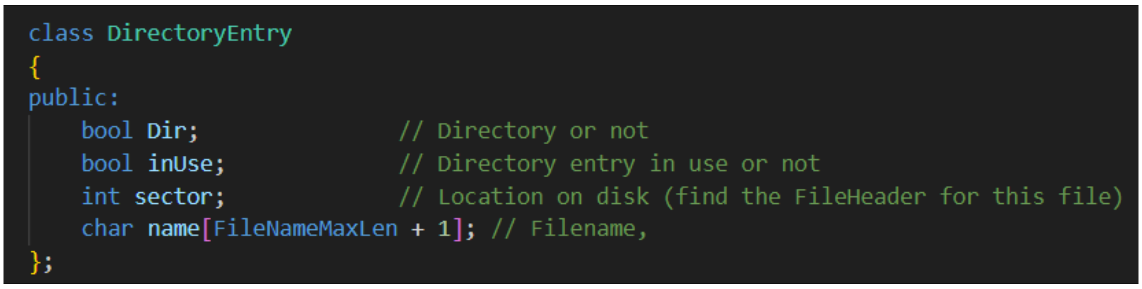
➢ 在 FileLength()中，若有 nextHeader，遞迴呼叫將整個 link list 的 numBytes 加起來。



```
void FileHeader::Print_File_Content()
{
    int i, j, k;
    char *data = new char[SectorSize];

    for (i = k = 0; i < numSectors; i++)
    {
        kernel->synchDisk->ReadSector(dataSectors[i], 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");
    }

    delete[] data;

    if(this->nextHeader != NULL) this->nextHeader->Print_File_Content();

}
```

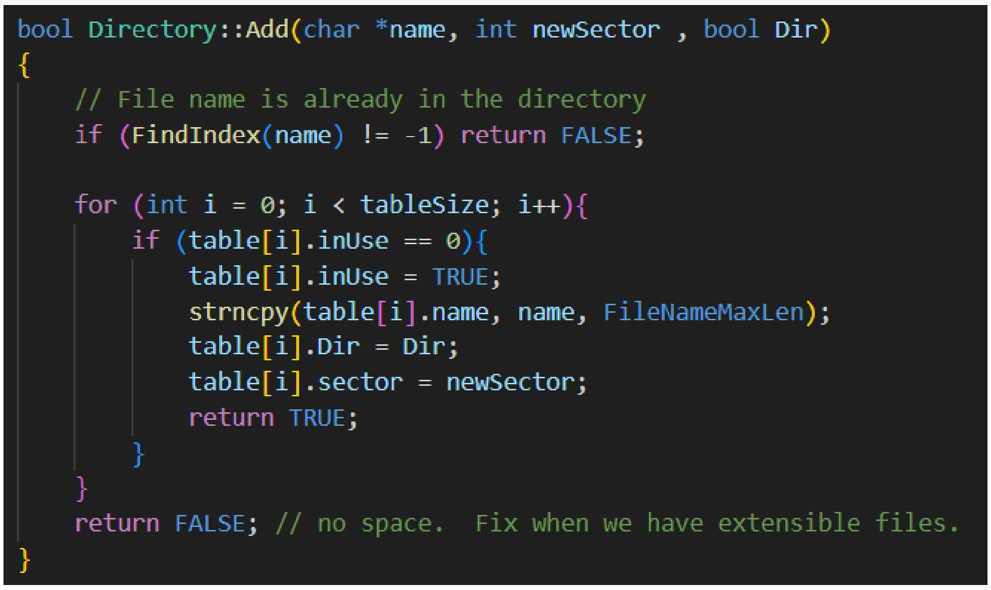
➢ 在 FileHeader()中，若有 nextHeader，遞迴呼叫將 file conent 印出。

2. Part II : Modify the file system code to support the subdirectory



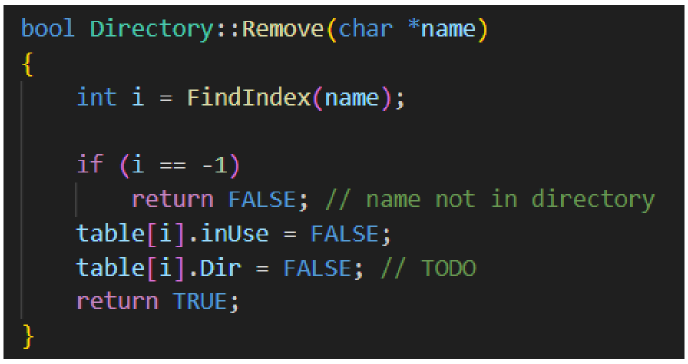
```
class DirectoryEntry
{
public:
    bool Dir;               // Directory or not
    bool inUse;             // Directory entry in use or not
    int sector;             // Location on disk (find the FileHeader for this file)
    char name[FileNameMaxLen + 1]; // Filename,
};
```

➢ 使用 Dir 紀錄使否為 directory。



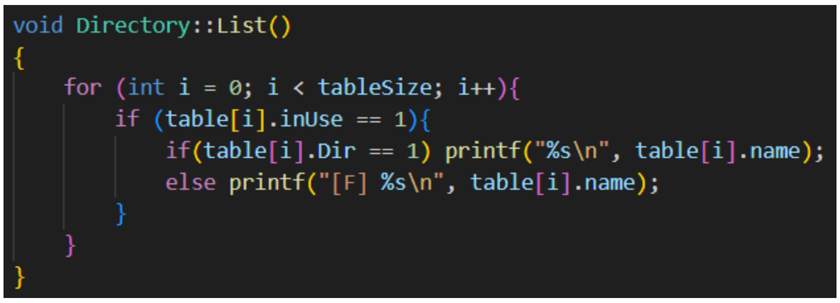
```
bool Directory::Add(char *name, int newSector , bool Dir)
{
    // File name is already in the directory
    if (FindIndex(name) != -1) return FALSE;

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

➢ 在 Add()中，加入 Dir 紀錄新增的為檔案還是資料夾。



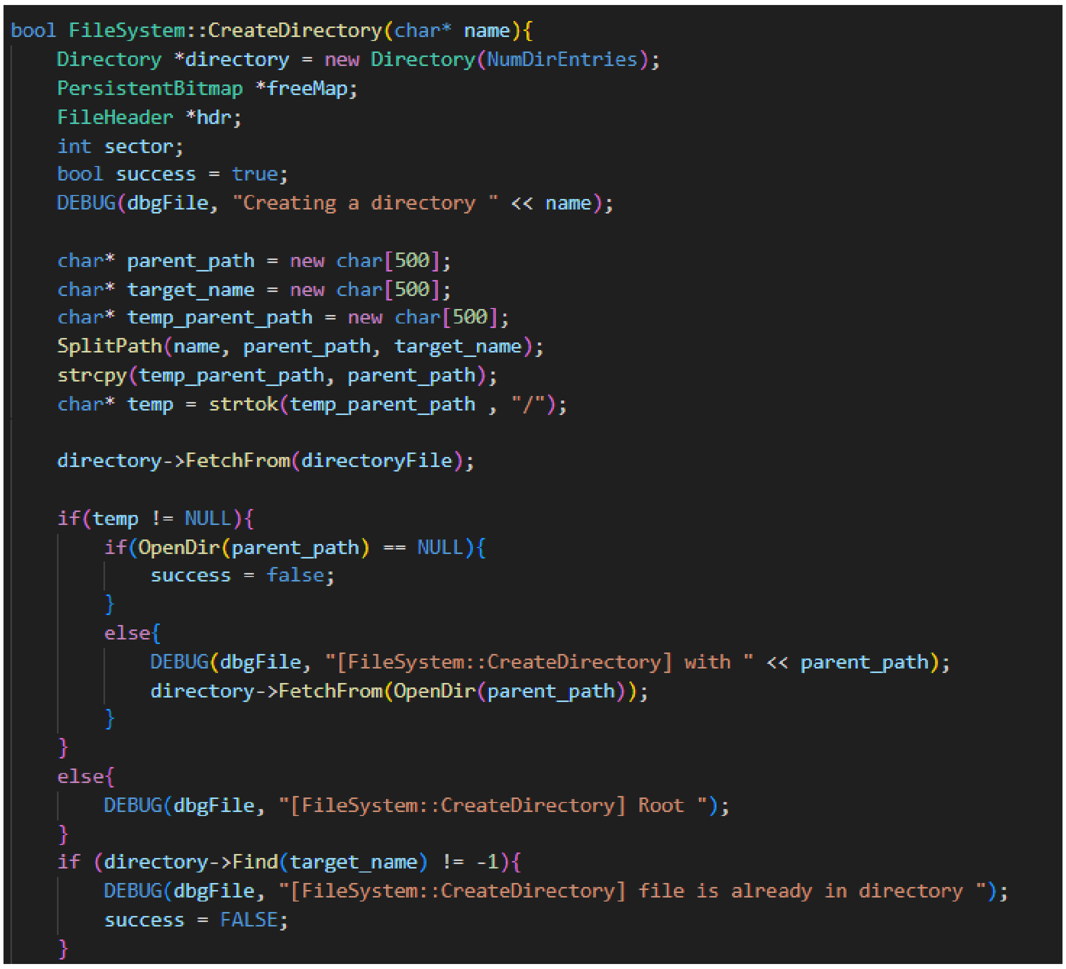
```
bool Directory::Remove(char *name)
{
    int i = FindIndex(name);

    if (i == -1)
        return FALSE; // name not in directory
    table[i].inUse = FALSE;
    table[i].Dir = FALSE; // TODO
    return TRUE;
}
```

➢ 在 Remove()中，Dir 設為 False。

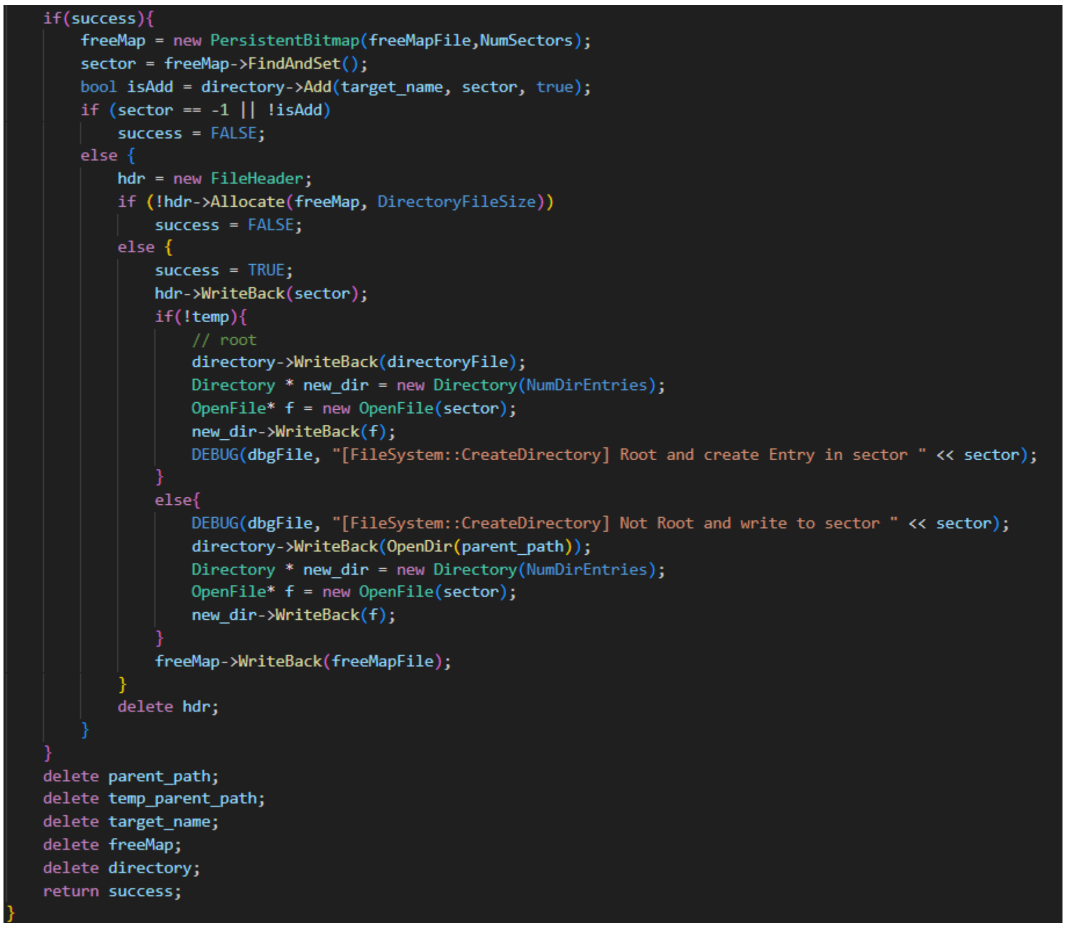


```
void Directory::List()
{
    for (int i = 0; i < tableSize; i++){
        if (table[i].inUse == 1){
            if(table[i].Dir == 1) printf("%s\n", table[i].name);
            else printf("[F] %s\n", table[i].name);
        }
    }
}
```

➢ 在 List()中，若 Dir 為 True，印出資料夾資訊，否則，印出檔案資訊。

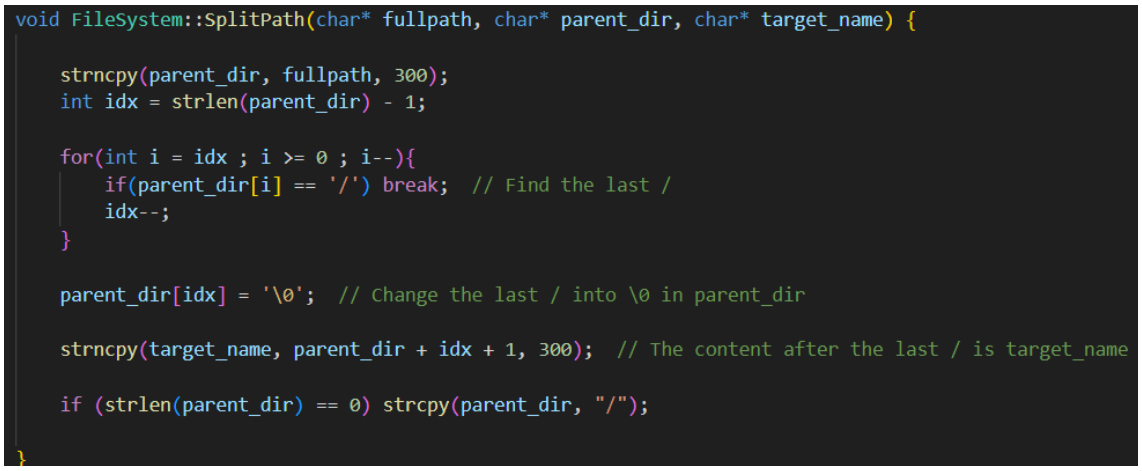


```
bool FileSystem::CreateDirectory(char* name){
    Directory *directory = new Directory(NumDirEntries);
    PersistentBitmap *freeMap;
    FileHeader *hdr;
    int sector;
    bool success = true;
    DEBUG(dbgFile, "Creating a directory " << name);

    char* parent_path = new char[500];
    char* target_name = new char[500];
    char* temp_parent_path = new char[500];
    SplitPath(name, parent_path, target_name);
    strcpy(temp_parent_path, parent_path);
    char* temp = strtok(temp_parent_path , "/");

    directory->FetchFrom(directoryFile);

    if(temp != NULL){
        if(OpenDir(parent_path) == NULL){
            success = false;
        }
        else{
            DEBUG(dbgFile, "[FileSystem::CreateDirectory] with " << parent_path);
            directory->FetchFrom(OpenDir(parent_path));
        }
    }
    else{
        DEBUG(dbgFile, "[FileSystem::CreateDirectory] Root ");
    }
    if (directory->Find(target_name) != -1){
        DEBUG(dbgFile, "[FileSystem::CreateDirectory] file is already in directory ");
        success = FALSE;
    }
```



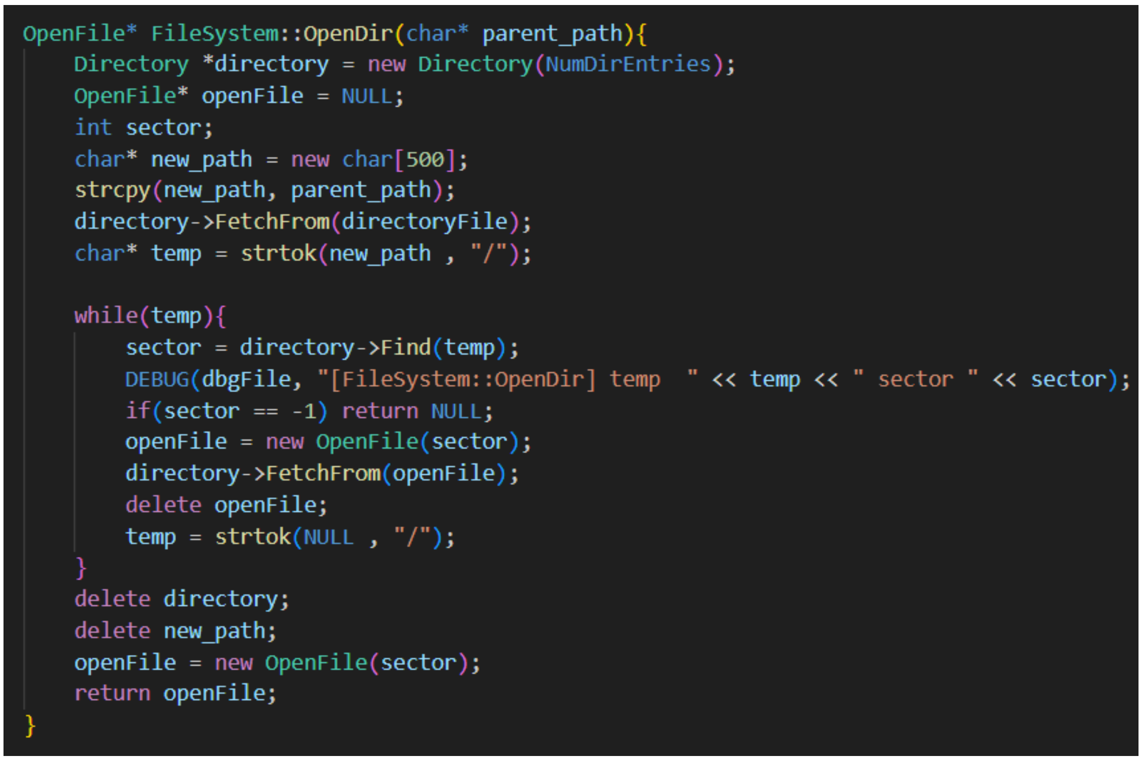
```
    if(success){
        freeMap = new PersistentBitmap(freeMapFile,NumSectors);
        sector = freeMap->FindAndSet();
        bool isAdd = directory->Add(target_name, sector, true);
        if (sector == -1 || !isAdd)
            success = FALSE;
        else {
            hdr = new FileHeader;
            if (!hdr->Allocate(freeMap, DirectoryFileSize))
                success = FALSE;
            else {
                success = TRUE;
                hdr->WriteBack(sector);
                if(!temp){
                    // root
                    directory->WriteBack(directoryFile);
                    Directory * new_dir = new Directory(NumDirEntries);
                    OpenFile* f = new OpenFile(sector);
                    new_dir->WriteBack(f);
                    DEBUG(dbgFile, "[FileSystem::CreateDirectory] Root and create Entry in sector " << sector);
                }
                else{
                    DEBUG(dbgFile, "[FileSystem::CreateDirectory] Not Root and write to sector " << sector);
                    directory->WriteBack(OpenDir(parent_path));
                    Directory * new_dir = new Directory(NumDirEntries);
                    OpenFile* f = new OpenFile(sector);
                    new_dir->WriteBack(f);
                }
                freeMap->WriteBack(freeMapFile);
            }
            delete hdr;
        }
    }
    delete parent_path;
    delete temp_parent_path;
    delete target_name;
    delete freeMap;
    delete directory;
    return success;
}
```

- 新增 CreateDirector 以創建資料夾，實作過程與原先的 Create 類似，需判斷 parent directory 是否存在、parent directory 是否有同名的檔案或資料夾、parent directory 是否有足夠空間、是否有足夠空間分配給 header、是否有足夠空間分配給 directory，以上條件滿足後，即 new directory，將 file sectors 給這個 directory，若此 directory 為 root，回寫至 directoryFile，若非 root，回寫至 parent_path。



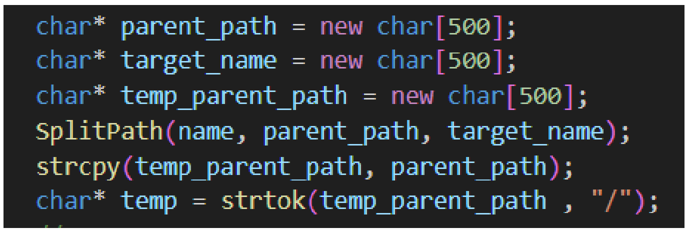
```
void FileSystem::SplitPath(char* fullpath, char* parent_dir, char* target_name) {

    strncpy(parent_dir, fullpath, 300);
    int idx = strlen(parent_dir) - 1;

    for(int i = idx ; i >= 0 ; i--){
        if(parent_dir[i] == '/') break;  // Find the last /
        idx--;
    }

    parent_dir[idx] = '\0';  // Change the last / into \0 in parent_dir

    strncpy(target_name, parent_dir + idx + 1, 300);  // The content after the last / is target_name

    if (strlen(parent_dir) == 0) strcpy(parent_dir, "/");

}
```

- SplitPath()將 fullpath 分割成 parent directory 及 target_name，實作方式為找到最後一個/，/前內容為 parent directory，/後為 target_name。



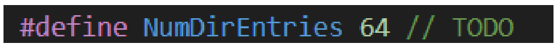
```
OpenFile* FileSystem::OpenDir(char* parent_path){
    Directory *directory = new Directory(NumDirEntries);
    OpenFile* openFile = NULL;
    int sector;
    char* new_path = new char[500];
    strcpy(new_path, parent_path);
    directory->FetchFrom(directoryFile);
    char* temp = strtok(new_path , "/");

    while(temp){
        sector = directory->Find(temp);
        DEBUG(dbgFile, "[FileSystem::OpenDir] temp  " << temp << " sector " << sector);
        if(sector == -1) return NULL;
        openFile = new OpenFile(sector);
        directory->FetchFrom(openFile);
        delete openFile;
        temp = strtok(NULL , "/");
    }
    delete directory;
    delete new_path;
    openFile = new OpenFile(sector);
    return openFile;
}
```

- OpenDir()可獲得 parent directory 的 sector。



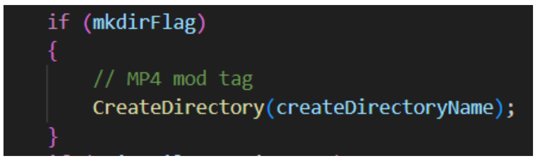
```
char* parent_path = new char[500];
char* target_name = new char[500];
char* temp_parent_path = new char[500];
SplitPath(name, parent_path, target_name);
strcpy(temp_parent_path, parent_path);
char* temp = strtok(temp_parent_path , "/");
```

- 因傳入 absolute，為了支援 subdirectory，必須在 CreateDirectory、Create、Open、 Remove、List、RecursiveList 中加入以下程式碼，並透過 OpenDir(parent_dir)來拿取 parent directory 的 table。



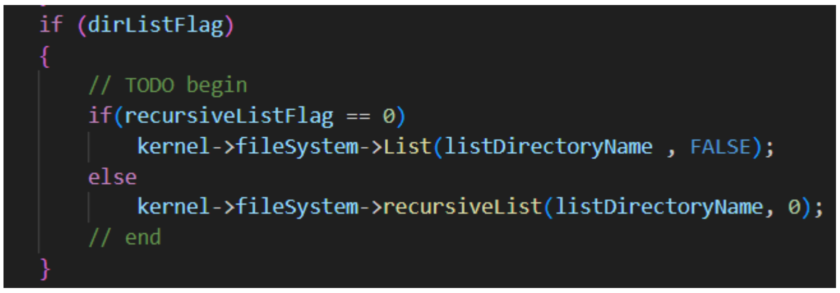
```
#define NumDirEntries 64 // TODO
```

➢ 每個 directory 有 64 個 file/subdirectories。



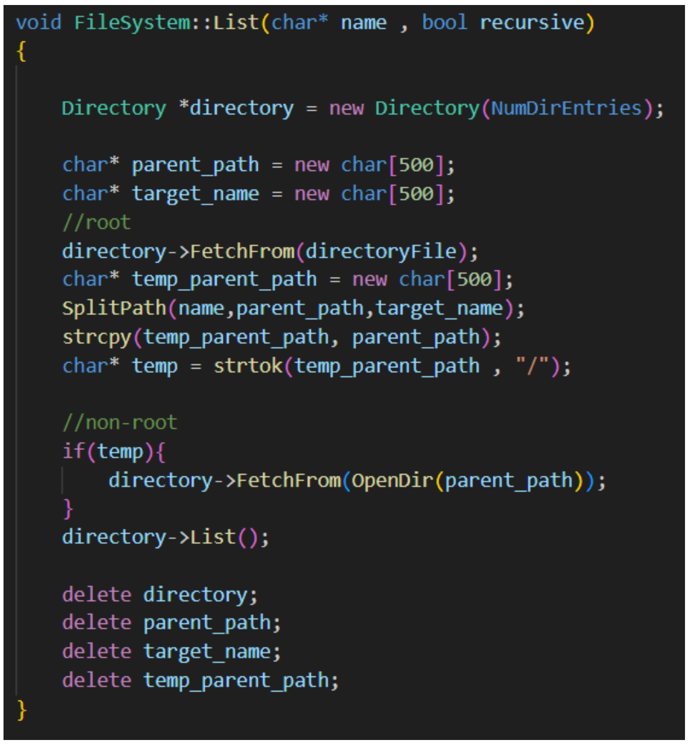
```
if (mkdirFlag)
{
    // MP4 mod tag
    CreateDirectory(createDirectoryName);
}
```

➢ 在 main.cc 中，收到 mkdir 即將 mkdirFlag 設為 TRUE，並呼叫 CreateDirectory()。



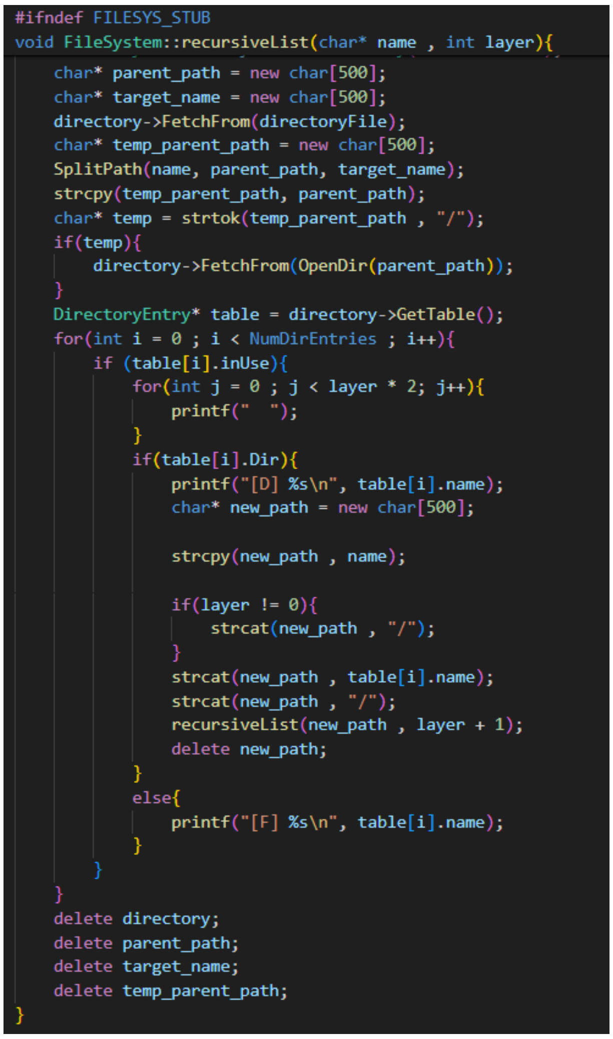
```
if (dirListFlag)
{
    // TODO begin
    if(recursiveListFlag == 0)
        kernel->fileSystem->List(listDirectoryName , FALSE);
    else
        kernel->fileSystem->recursiveList(listDirectoryName, 0);
    // end
}
```

➢ 若 dirListFlag 為 TRUE，list 為 recursiveList 時，呼叫 recursiveList()，非 recursiveList 時，呼叫 List()。



```
void FileSystem::List(char* name , bool recursive)
{

    Directory *directory = new Directory(NumDirEntries);

    char* parent_path = new char[500];
    char* target_name = new char[500];
    //root
    directory->FetchFrom(directoryFile);
    char* temp_parent_path = new char[500];
    SplitPath(name,parent_path,target_name);
    strcpy(temp_parent_path, parent_path);
    char* temp = strtok(temp_parent_path , "/");

    //non-root
    if(temp){
        directory->FetchFrom(OpenDir(parent_path));
    }
    directory->List();

    delete directory;
    delete parent_path;
    delete target_name;
    delete temp_parent_path;
}
```

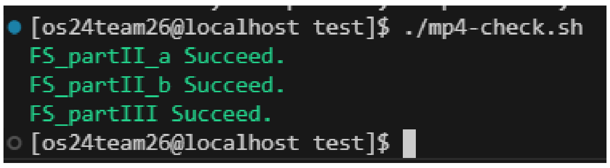
➢ 在 List()中，若為 root directory，獲取 directoryFile(table)資訊，呼叫 directory->list()，若為 non-root directory，呼叫 parent directory list。



```
#ifndef FILESYS_STUB
void FileSystem::recursiveList(char* name , int layer){
    char* parent_path = new char[500];
    char* target_name = new char[500];
    directory->FetchFrom(directoryFile);
    char* temp_parent_path = new char[500];
    SplitPath(name, parent_path, target_name);
    strcpy(temp_parent_path, parent_path);
    char* temp = strtok(temp_parent_path , "/");
    if(temp){
        directory->FetchFrom(OpenDir(parent_path));
    }
    DirectoryEntry* table = directory->GetTable();
    for(int i = 0 ; i < NumDirEntries ; i++){
        if (table[i].inUse){
            for(int j = 0 ; j < layer * 2; j++){
                printf("  ");
            }
            if(table[i].Dir){
                printf("[D] %s\n", table[i].name);
                char* new_path = new char[500];

                strcpy(new_path , name);

                if(layer != 0){
                    strcat(new_path , "/");
                }
                strcat(new_path , table[i].name);
                strcat(new_path , "/");
                recursiveList(new_path , layer + 1);
                delete new_path;
            }
            else{
                printf("[F] %s\n", table[i].name);
            }
        }
    }
    delete directory;
    delete parent_path;
    delete target_name;
    delete temp_parent_path;
}
```

➢ 在 reciursiveList()中，與 list 實作相似，若在 traverse 時，遇到 Dir 為 TRUE，就遞迴呼叫 reciursiveList()，進到下層 layer。

3. Result

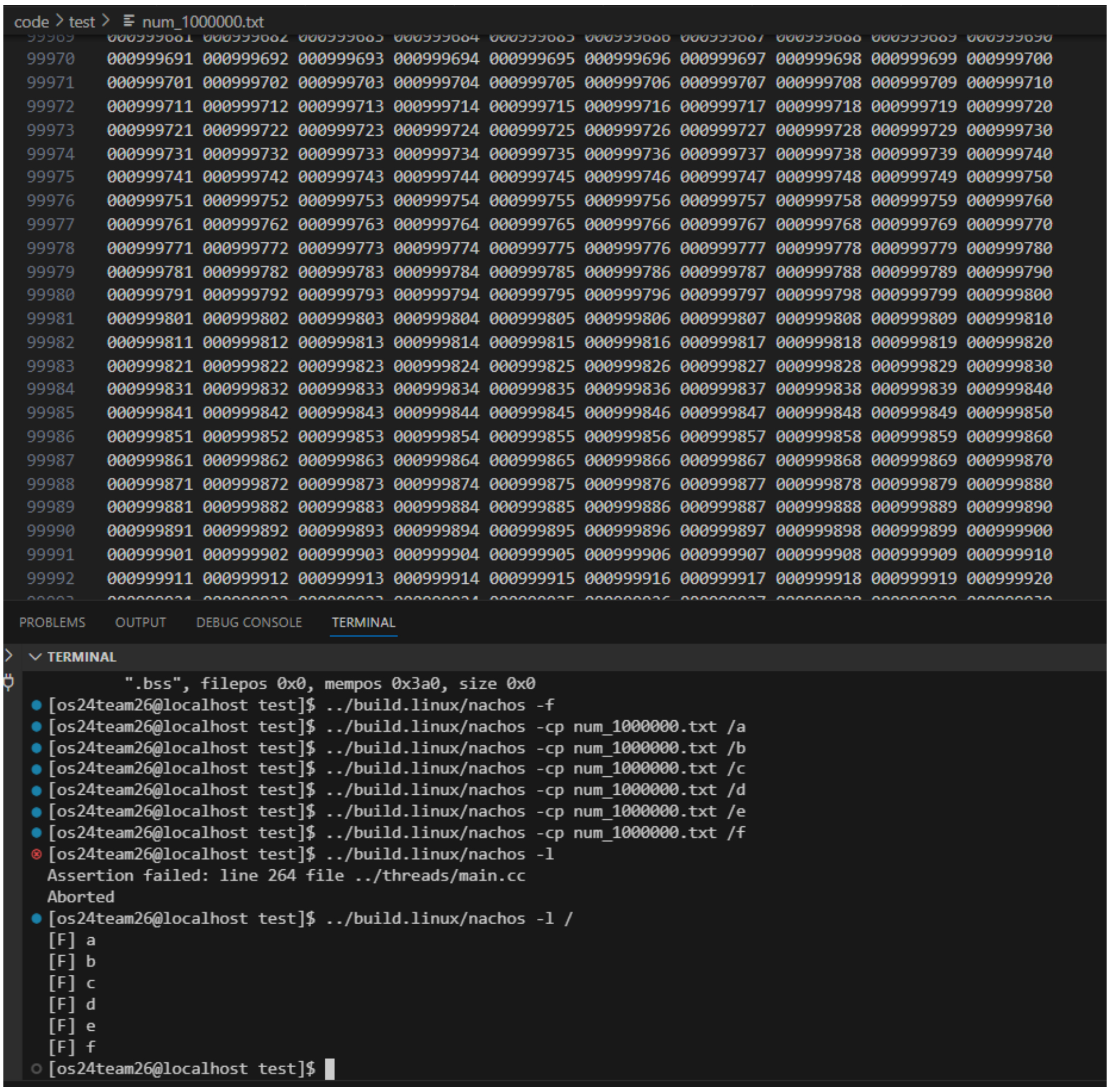


```
[os24team26@localhost test]$ ./mp4-check.sh
FS_partII_a Succeed.
FS_partII_b Succeed.
FS_partIII Succeed.
[os24team26@localhost test]$ 
```

4. Bonus

(1) Enhance the NachOS to support even larger file size

```
// MP4 Hint: DO NOT change the SectorSize, but other constants are allowed
const int SectorSize = 128;    // number of bytes per disk sector
const int SectorsPerTrack  = 16384; // number of sectors per disk track
const int NumTracks = 32;    // number of tracks per disk
const int NumSectors = (SectorsPerTrack * NumTracks); // total # of sectors per disk
```



code > test > num_1000000.txt

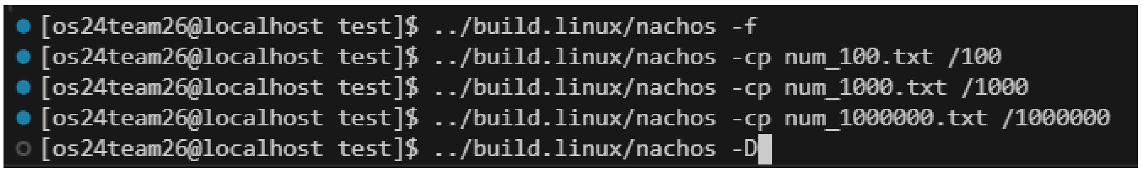
```
000999691 000999692 000999693 000999694 000999695 000999696 000999697 000999698 000999699 000999700
000999701 000999702 000999703 000999704 000999705 000999706 000999707 000999708 000999709 000999710
000999711 000999712 000999713 000999714 000999715 000999716 000999717 000999718 000999719 000999720
000999721 000999722 000999723 000999724 000999725 000999726 000999727 000999728 000999729 000999730
000999731 000999732 000999733 000999734 000999735 000999736 000999737 000999738 000999739 000999740
000999741 000999742 000999743 000999744 000999745 000999746 000999747 000999748 000999749 000999750
000999751 000999752 000999753 000999754 000999755 000999756 000999757 000999758 000999759 000999760
000999761 000999762 000999763 000999764 000999765 000999766 000999767 000999768 000999769 000999770
000999771 000999772 000999773 000999774 000999775 000999776 000999777 000999778 000999779 000999780
000999781 000999782 000999783 000999784 000999785 000999786 000999787 000999788 000999789 000999790
000999791 000999792 000999793 000999794 000999795 000999796 000999797 000999798 000999799 000999800
000999801 000999802 000999803 000999804 000999805 000999806 000999807 000999808 000999809 000999810
000999811 000999812 000999813 000999814 000999815 000999816 000999817 000999818 000999819 000999820
000999821 000999822 000999823 000999824 000999825 000999826 000999827 000999828 000999829 000999830
000999831 000999832 000999833 000999834 000999835 000999836 000999837 000999838 000999839 000999840
000999841 000999842 000999843 000999844 000999845 000999846 000999847 000999848 000999849 000999850
000999851 000999852 000999853 000999854 000999855 000999856 000999857 000999858 000999859 000999860
000999861 000999862 000999863 000999864 000999865 000999866 000999867 000999868 000999869 000999870
000999871 000999872 000999873 000999874 000999875 000999876 000999877 000999878 000999879 000999880
000999881 000999882 000999883 000999884 000999885 000999886 000999887 000999888 000999889 000999890
000999891 000999892 000999893 000999894 000999895 000999896 000999897 000999898 000999899 000999900
000999901 000999902 000999903 000999904 000999905 000999906 000999907 000999908 000999909 000999910
000999911 000999912 000999913 000999914 000999915 000999916 000999917 000999918 000999919 000999920
```

```
        ".bss", filepos 0x0, mempos 0x3a0, size 0x0
[os24team26@localhost test]$ ../build.linux/nachos -f
[os24team26@localhost test]$ ../build.linux/nachos -cp num_1000000.txt /a
[os24team26@localhost test]$ ../build.linux/nachos -cp num_1000000.txt /b
[os24team26@localhost test]$ ../build.linux/nachos -cp num_1000000.txt /c
[os24team26@localhost test]$ ../build.linux/nachos -cp num_1000000.txt /d
[os24team26@localhost test]$ ../build.linux/nachos -cp num_1000000.txt /e
[os24team26@localhost test]$ ../build.linux/nachos -cp num_1000000.txt /f
[os24team26@localhost test]$ ../build.linux/nachos -l
Assertion failed: line 264 file ../threads/main.cc
Aborted
[os24team26@localhost test]$ ../build.linux/nachos -l /
[F] a
[F] b
[F] c
[F] d
[F] e
[F] f
[os24team26@localhost test]$ 
```

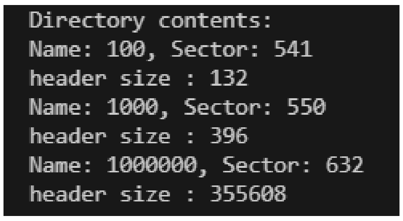
- ➢ 將 SectorPerTrack 更改為 16384，使得 disk 大小為 128 * 16384 * 32 = 64MB，可以支援 64MB 的單一檔案。
- ➢ 複製 6 個 10MB 的檔案，可以 list 出正確的檔案，代表 disk 空間以成功擴展。

(2) Multi-level header size

```
void FileHeader::Print()
{
    int fileLen = FileLength();
    printf("header size : %d \n", sizeof(FileHeader) * divRoundUp(fileLen, MaxFileSize));

}
```



```
● [os24team26@localhost test]$ ../build.linux/nachos -f
● [os24team26@localhost test]$ ../build.linux/nachos -cp num_100.txt /100
● [os24team26@localhost test]$ ../build.linux/nachos -cp num_1000.txt /1000
● [os24team26@localhost test]$ ../build.linux/nachos -cp num_1000000.txt /1000000
○ [os24team26@localhost test]$ ../build.linux/nachos -D
```



```
Directory contents:
Name: 100, Sector: 541
header size : 132
Name: 1000, Sector: 550
header size : 396
Name: 1000000, Sector: 632
header size : 355608
```

- 在 print()中，算出該個 file 所使用到的 FileHeader 數量，再乘上 MaxFileSize，即為該 file 所使用的總 header size。
- 將三個大小不同的檔案放入，會發現越小的檔案使用到的 header size 越小，反之，越大檔案使用到的 header size 越大。