Operating System
MP4 Report

組別：25

陳凱揚：Trace code、Implement、Report

簡佩如：Trace code、Implement、Report

1. Understanding NachOS file system

(1) Explain how the NachOS FS manage and find free block space?

Nachos 透過 Bitmap 來管理 free blocks。首先在 kernel 中收到指令為 “-f” 時，formatFlag 會設為 TRUE，讓 fileSystem 建立時格式化 disk。當沒有格式化時，會直接打開 sector 0 所在的檔案，在 Nachos 運行時保持開啟，存留在記憶體中；當需要格式化時，會去初始化 NumSectors bits 的 PersistentBitmap，並 new 名為 mapHdr、dirHdr 的 FileHeader，用來 allocate 空間給 freeMap 且寫回 disk（圖 1-1）。

```
86          PersistentBitmap *freeMap = new PersistentBitmap(NumSectors);
87          Directory *directory = new Directory(NumDirEntries);
88          FileHeader *mapHdr = new FileHeader;
89          FileHeader *dirHdr = new FileHeader;
```

▲ 圖 1-1

此外，PersistentBitmap 繼承了 Bitmap，可以透過 Bitmap 找到 free block，以下可以看到管理方法會在一開始初始所有 bit 為 0（0 代表未使用、1 代表使用中）（圖 1-2），並且使用 FindAndSet()找到一個 bit 為 0 的 index（圖 1-3），透過 Mark()將其設為 1，再回傳 index，如果找不到救回傳 -1。

```
21  Bitmap::Bitmap(int numItems)
22  {
23      int i;
24
25      ASSERT(numItems > 0);
26
27      numBits = numItems;
28      numWords = divRoundUp(numBits, BitsInWord);
29      map = new unsigned int[numWords];
30      for (i = 0; i < numWords; i++)
31      {
32          map[i] = 0; // initialize map to keep Pu
33      }
34      for (i = 0; i < numBits; i++)
35      {
36          Clear(i);
37      }
```

▲ 圖 1-2

```
112 int Bitmap::FindAndSet()
113 {
114     for (int i = 0; i < numBits; i++)
115     {
116         if (!Test(i))
117         {
118             Mark(i);
119             return i;
120         }
121     }
122     return -1;
```

▲ 圖 1-3

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

一開始會透過 Mark()將 FreeMapSector（sector 0）設為使用中，之後才不會再用到這個 sector。接著 mapHdr 會 allocate data block 給 freeMap，而 FreeMapFileSize 是 freemap 的大小，為 NumSectors/BitsInByte（圖 1-4）。

```
freeMap->Mark(FreeMapSector);
ASSERT(mapHdr->Allocate(freeMap, FreeMapFileSize));
#define FreeMapFileSize (NumSectors / BitsInByte)
```

▲ 圖 1-4

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

在 dick.h 中可以看到每個 sector 有 128 個 bytes，每個 track 有 32 個 sectors，disk 有 32 個 tracks（圖 1-5）。而在 disk.cc 中可以看到 DiskSize 大小為 4+(32*32)*128 = KB。其中 MagicSize 是用來驗證，防止不小心把有用的 file 當成 disk（圖 1-6）。

```
50  const int SectorSize = 128;   // number of bytes per
51  const int SectorsPerTrack  = 32;  // number of sectors
52  const int NumTracks = 32;   // number of tracks per di
53  const int NumSectors = (SectorsPerTrack * NumTracks);
```

▲ 圖 1-5

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

▲ 圖 1-6

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

與 freeMap 很像，在建立 FileSystem 時，當不需格式化時，直接打開 sector 1 所在的檔案，保留在記憶體中；當需要格式化時，建立 root directory 初始化底下的 DirectoryEntry，將每個 entry 的 inUse 設為 0（圖 1-7），並 allocate 空間且 Mark()所使用的空間，DirectoryFileSize 為 sizeof(DirectoryEntry)*NumDirEntries（圖 1-8）。

```
37  Directory::Directory(int size)
38  {
39      table = new DirectoryEntry[size];
40
41      // MP4 mod tag
42      memset(table, 0, sizeof(DirectoryEntry) * size);
43
44      tableSize = size;
45      for (int i = 0; i < tableSize; i++)
46          table[i].inUse = FALSE;
47  }
```

▲ 圖 1-7

```
freeMap->Mark(DirectorySector);
ASSERT(dirHdr->Allocate(freeMap, DirectoryFileSize));
#define DirectoryFileSize (sizeof(DirectoryEntry) * NumDirEntries)
```

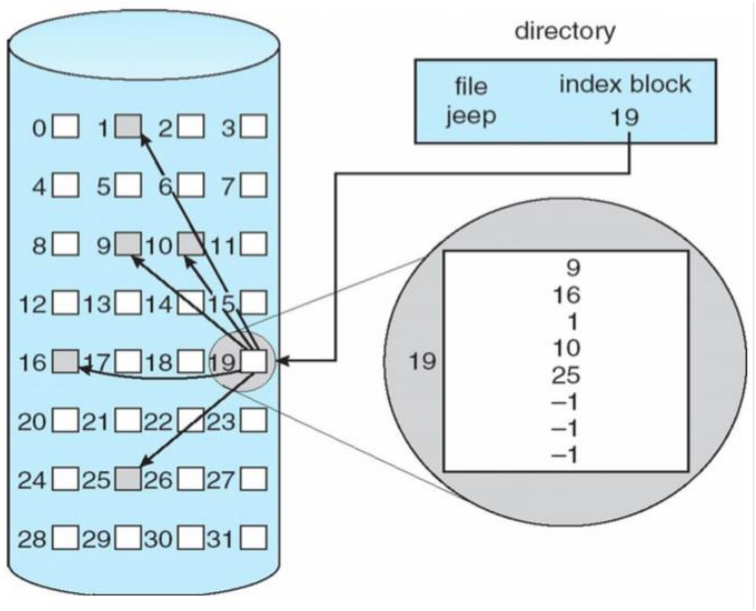
▲ 圖 1-8

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

在 filehdr.h 的 FileHeader 這個 class 可以看到 inode 所儲存的相關資訊（圖 1-9），其中 numBytes 為檔案的大小，numSectors 為檔案使用了多少 sector，dataSectors 為儲存 index 的 table，Nachos 使用 direct index scheme（圖 1-10）。

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

▲ 圖 1-9



▲ 圖 1-10

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

由於目前為 direct index allocation，file header 只能使用 1 個 sector 的大小，所以能儲存的 index 有限。而在 filehdr.h 中可以看到 MaxFileSize 為 NumDirect*SectorSize，其中 NumDirect 為(SectorSize-2*sizeof(int))/sizeof(int)，扣掉的 2 個 int 為 numBytes 和 numSectors（圖 1-11）。NumDirect 為(128-2*4)/4 = 30，則 MaxFileSize = 30*128，約為 4KB。

```
20    #define NumDirect ((SectorSize - 2 * sizeof(int)) / sizeof(int))
21    #define MaxFileSize1 (NumDirect * SectorSize)
```

▲ 圖 1-11

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

(1) Combine your MP1 file system call interface with NachOS FS

與 MP1 大致相同，user 呼叫 system call 後，進入 syscall.h（這裡原先就建好 interface 了），接著程式執行後，會進到 exception.cc 對應的 case（圖 2-1，以 SC_Create 為例），再呼叫 ksyscall.h 中對應的 function（圖 2-2，以 SysCreate() 為例），最後呼叫 fileSystem 下對應的 function（圖 2-3）。

```
94  // ****************** MP4 ****************** //
95  #else
96          case SC_Create:
97              val = kernel->machine->ReadRegister(4);
98              size = kernel->machine->ReadRegister(5);
99              {
100                 char *filename = &(kernel->machine->mainMemory[val]);
101                 //cout << filename << endl;
102                 status = SysCreate(filename, size);
103                 kernel->machine->WriteRegister(2, (int)status);
104             }
105             kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
106             kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
107             kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
108             return;
109             ASSERTNOTREACHED();
110             break;
```

▲ 圖 2-1

```
36  // ****************** MP4 ****************** //
37  #else
38  int SysCreate(char *name, int size)
39  {
40      return kernel->fileSystem->CreateAFile(name, size);
41  }
```

▲ 圖 2-2

```
112     int CreateAFile(char *name, int size);
113     OpenFileId OpenAFile(char *name);
114     int WriteFile(char *buffer, int size, OpenFileId id);
115     int ReadFile(char *buffer, int size, OpenFileId id);
116     int CloseFile(OpenFileId id);
```

▲ 圖 2-3

(2) Implement five system calls

與 MP1 相同（圖 2-4），只是多了 CreateAFile()，且要在 fileSystem 下新增 fileDescriptorTable 來儲存開啟的檔案。

```
335  // ****************** MP4 ****************** //
336  int FileSystem::CreateAFile(char *name, int size){
337      return Create(name, size);
338  }
339  OpenFileId FileSystem::OpenAFile(char *name){
340      OpenFile *file = Open(name);
341      if(file == NULL) return -1;
342      for(int i = 0; i < 20; i++){
343          if(fileDescriptorTable[i] == NULL){
344              fileDescriptorTable[i] = file;
345              return i+1;
346          }
347      }
348      delete file;
349      return -1;
350  }
351  int FileSystem::WriteFile(char *buffer, int size, OpenFileId id){
352      if(id<1 || id>20 || size<0 || fileDescriptorTable[id-1]==NULL) return -1;
353      return fileDescriptorTable[id-1]->Write(buffer, size);
354  }
355  int FileSystem::ReadFile(char *buffer, int size, OpenFileId id){
356      if(id<1 || id>20 || size<0 || fileDescriptorTable[id-1]==NULL) return -1;
357      return fileDescriptorTable[id-1]->Read(buffer, size);
358  }
359  int FileSystem::CloseFile(OpenFileId id){
360      if(id<1 || id>20 || fileDescriptorTable[id-1]==NULL) return -1;
361      delete fileDescriptorTable[id-1];
362      fileDescriptorTable[id-1] = NULL;
363      return 1;
364  }
```

▲ 圖 2-4

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

在這邊我將原先的 index scheme，加強成 linked index scheme，使檔案的長度不受限制。首先在 filehdr.h 下，我新增了 nextHeader 和 nextHeaderSector 等 2 個 private 變數（圖 2-5），用以儲存下一個 index block 的資訊，其中注意的是 nextHeader 一定要寫在第一個，因為他不需要存回 disk 中，放在第一個用來之後 WriteBack()時可以直接跳過，細節會在後面再做解釋。此外，NumDirect 也必須更新（圖 2-6），扣除 nextHeaderSector 的大小，使需要存回 disk 的大小與 sector 大小保持相同。

```
86      // ****************** MP4 ****************** //
87      FileHeader *nextHeader;
88      int nextHeaderSector;
89      // ****************** MP4 ****************** //
```

▲ 圖 2-5

```
// ****************** MP4 ****************** //
#define NumDirect ((SectorSize - 3 * sizeof(int)) / sizeof(int))
// ****************** MP4 ****************** //
```

▲ 圖 2-6

接著開始修改每個函式，首先在 constructor 初始化（圖 2-7）、destructor delete 掉（圖 2-8）。

```
39  FileHeader::FileHeader()
40  {
41      // ****************** MP4 ****************** //
42      nextHeaderSector = -1;
43      nextHeader = NULL;
44      // ****************** MP4 ****************** //
```

▲ 圖 2-7

```
57  FileHeader::~FileHeader()
58  {
59      // nothing to do now
60      if(nextHeader != NULL) delete nextHeader;
61  }
```

▲ 圖 2-8

在 Allocate()中（圖 2-9），如果發現一個 block 存不下所有檔案的話，就會去尋找一個新的 sector，儲存下一個 file header，並繼續往下呼叫這個 header 的 Allocate()儲存剩餘的檔案。Deallocate()則是會繼續往下一個 file header 呼叫 Deallocate()（圖 2-10）。

```
74  bool FileHeader::Allocate(PersistentBitmap *freeMap, int fileSize)
75  {
76      // ****************** MP4 ****************** //
77      numBytes = MaxFileSize<fileSize?MaxFileSize:fileSize;
78      numSectors = divRoundUp(numBytes, SectorSize);
79      if (freeMap->NumClear() < numSectors){
80          return FALSE; // not enough space
81      }
82
83      for (int i = 0; i < numSectors; i++)
84      {
85          dataSectors[i] = freeMap->FindAndSet();
86          // since we checked that there was enough free space,
87          // we expect this to succeed
88          ASSERT(dataSectors[i] >= 0);
89      }
90
91      fileSize -= numBytes;
92      if(fileSize > 0){
93          if((nextHeaderSector = freeMap->FindAndSet()) == -1) return FALSE;
94          nextHeader = new FileHeader;
95          return nextHeader->Allocate(freeMap, fileSize);
96      }
97      return TRUE;
98      // ****************** MP4 ****************** //
99  }
```

▲ 圖 2-9

```
115     // ****************** MP4 ****************** //
116     if(nextHeader != NULL) nextHeader->Deallocate(freeMap);
117     // ****************** MP4 ****************** //
```

▲ 圖 2-10

在 FetchFrom()中（圖 2-11），同樣會繼續遞迴呼叫下一個 file header 的 FetchFrom()，其中注意的是這邊從 disk 讀出來的資料並不包含 nextHeader，所以需要以 nextHeaderSector 判斷是否有下一個 header，若有的話就 new 一個新的 FileHeader。在 WriteBack()時（圖 2-12），會將 this 加上 sizeof(FileHeader*)用來跳過前面所提到的不需存回 disk 的 nextHeader，因此在剛剛的 FetchFrom()裡，this 同樣會加上 sizeof(FileHeader*)，用來跳過這個位置。

```
127  void FileHeader::FetchFrom(int sector)
128  {
129      // ******************* MP4 ******************* //
130      kernel->synchDisk->ReadSector(sector, (char *)this+sizeof(FileHeader*));
131      if(nextHeaderSector != -1){
132          nextHeader = new FileHeader;
133          nextHeader->FetchFrom(nextHeaderSector);
134      }
135      // ******************* MP4 ******************* //
```

▲ 圖 2-11

```
149  void FileHeader::WriteBack(int sector)
150  {
151      // ******************* MP4 ******************* //
152      kernel->synchDisk->WriteSector(sector, (char *)this+sizeof(FileHeader*));
153      if(nextHeader != NULL) nextHeader->WriteBack(nextHeaderSector);
154      // ******************* MP4 ******************* //
```

▲ 圖 2-12

在 ByteToSector()中（圖 2-13），如果 offset 超過這個 block 儲存的上限，便會遞迴的向下尋找，並將傳入的 offset 減掉 MaxFileSize。而 FileLength()也是遞迴的加上後面的 bytes 再回傳（圖 2-14）。

```
175  int FileHeader::ByteToSector(int offset)
176  {
177      // ******************* MP4 ******************* //
178      int idx = divRoundDown(offset, SectorSize);
179      if(idx < NumDirect) return dataSectors[idx];
180      else return nextHeader->ByteToSector(offset-MaxFileSize);
181      // ******************* MP4 ******************* //
182  }
```

▲ 圖 2-13

```
189  int FileHeader::FileLength()
190  {
191      // ******************* MP4 ******************* //
192      if(nextHeader != NULL) return numBytes+nextHeader->FileLength();
193      else return numBytes;
194      // ******************* MP4 ******************* //
195  }
```

▲ 圖 2-14

在 Print()中，為了讓 block 資訊和 content 資訊能夠分開輸出，因此新增 PrintBlock()和 PrintContent()（圖 2-15），用來分別在 Print()裡呼叫遞迴。而這兩個 function 與原來的輸出方式皆相同，只是在最後加上遞迴呼叫（圖 2-16）。

```
66      // ******************** MP4 ******************** //
67      void PrintBlock();
68      void PrintContent();
69      // ******************** MP4 ******************** //
```

▲ 圖 2-15

```
203 // ******************** MP4 ******************** //
204 void FileHeader::Print()
205 {
206     printf("\nFileHeader contents.  File size: %d. Header size: %d.  File blocks:\n", \
207         FileLength(), sizeof(FileHeader)*divRoundUp(FileLength(), MaxFileSize));
208     PrintBlock();
209     printf("\nFile contents:\n");
210     PrintContent();
211 }
212
213 void FileHeader::PrintBlock(){
214     for (int i = 0; i < numSectors; i++)
215         printf("%d ", dataSectors[i]);
216     if(nextHeader != NULL) nextHeader->PrintBlock();
217 }
218
219 void FileHeader::PrintContent(){
220     int i, j, k;
221     char *data = new char[SectorSize];
222     for (i = k = 0; i < numSectors; i++)
223     {
224         kernel->synchDisk->ReadSector(dataSectors[i], data);
225         for (j = 0; (j < SectorSize) && (k < numBytes); j++, k++)
226         {
227             if ('\040' <= data[j] && data[j] <= '\176') // isprint(data[j])
228                 printf("%c", data[j]);
229             else
230                 printf("\\%x", (unsigned char)data[j]);
231         }
232         printf("\n");
233     }
234     delete[] data;
235     if(nextHeader != NULL) nextHeader->PrintContent();
236 }
237 // ******************** MP4 ******************** //
```

▲ 圖 2-16

3. Modify the file system code to support subdirectory

(1) Implement the subdirectory structure

要實作出 subdirectory 的結構，從底層到上層的 DirectoryEntry、Directory、FileSystem 等都需要有對應的修改，這邊會從最底層開始解釋實作過程。

首先，在 DirectoryEntry 新增一個 member variable 為 isDir，用來判斷此 entry 的資料是 directory 還是 file（圖 3-1）。

```
32  class DirectoryEntry
33  {
34  public:
35      bool inUse;
36      int sector;
37
38      // ******************** MP4
39      bool isDir;
40      // ******************** MP4
```

▲ 圖 3-1

在 Directory 中（圖 3-2），將原先 Add()多傳入一個參數為 isDir，並新增 RecursiveList()來達到 “-lr” 的功能，而 GetDirSector()的功能為傳入一個相對路徑，回傳此檔案所在的 sector。而在 Add()中（圖 3-3），找到 entry 後，新增設定 isDir 的資料。

```
69      // ******************** MP4 ********************
70      bool Add(char *name, int newSector, bool isDir);
71
72      void RecursiveList(int layer);
73      int GetDirSector(char *name);
```

▲ 圖 3-2

```
134     for (int i = 0; i < tableSize; i++)
135         if (!table[i].inUse)
136         {
137             table[i].inUse = TRUE;
138             strncpy(table[i].name, name,
139             table[i].sector = newSector;
140             // ******************** MP4 *
141             table[i].isDir = isDir;
142             // ******************** MP4 *
143             return TRUE;
```

▲ 圖 3-3

在一般的 List()中（圖 3-4），修改輸出為指定的格式，用 isDir 判斷是 directory 還是 file。

在 RecursiveList()中（圖 3-5），在每個 entry 輸出前，會先以 layer 來判斷目前在第幾層，加上對應數量的 indent，且在輸出後，若是 directory 的話，便繼續向下遞迴呼叫 RecursiveList()，並將 layer 加 1。

```
171  void Directory::List()
172  {
173      // ****************** MP4 ****************** //
174      for(int i = 0; i < tableSize; i++){
175          if(table[i].inUse){
176              printf("[%c] %s\n", (table[i].isDir?'D':'F'), table[i].name);
177          }
178      }
179      // ****************** MP4 ****************** //
```

▲ 圖 3-4

```
204  // ****************** MP4 ****************** //
205  void Directory::RecursiveList(int layer){
206      Directory *subDirectory = new Directory(NumDirEntries);
207      for(int i = 0; i < tableSize; i++){
208          if(table[i].inUse){
209              for(int j = 0; j < layer; j++) printf("\t");
210              printf("[%c] %s\n", (table[i].isDir?'D':'F'), table[i].name);
211              if(table[i].isDir){
212                  OpenFile *file = new OpenFile(table[i].sector);
213                  subDirectory->FetchFrom(file);
214                  subDirectory->RecursiveList(layer+1);
215                  delete file;
```

▲ 圖 3-5

在 GetDirSector()中的步驟主要為字串的處理（圖 3-6），例如：name = “/t0/t1”，會先取出/t0，並找出其 entry 的 index，接著建立 t0 的 directory，遞迴呼叫 GetDirSector()，此時傳入的 name 為/t1，直到整個字串結束後回傳 sector 位置。此外，在這邊會假設除了 root 外，所有的 name 結尾不會有 “/”，因此當 name = “/”，即為 root，回傳 1 代表 root 的資料存在 sector 1。

```
220  int Directory::GetDirSector(char *name){
221      if(!strcmp(name, "/")) return 1;
222      int idx = 1;
223      while(name[idx]!='\0' && name[idx]!='/') idx++;
224
225      char entry[256];
226      strncpy(entry, name+1, idx-1);
227      entry[idx-1] = '\0';
228
229      int i = FindIndex(entry), sector = -1;
230      if(name[idx] != '\0'){
231          Directory *directory = new Directory(NumDirEntries);
232          OpenFile *file = new OpenFile(table[i].sector);
233          directory->FetchFrom(file);
234          sector = directory->GetDirSector(name+idx);
235          delete directory;
236          delete file;
237      }
238      else sector = table[i].sector;
239
240      return sector;
241  }
```

▲ 圖 3-6

接著在 FileSystem 中新增 3 個 function（圖 3-7、圖 3-8），CreateDirectory() 用來使 kernel 呼叫並建立 direcrtory，RecursiveList()也是由 kernel 所呼叫，SplitPath()的功能為將傳進的字串切成兩部分，如：name = “/t0/t1/t2，切成 ”/t0/t1” 和 ”t2”。

```
118     bool CreateDirectory(char *name);
119     void RecursiveList(char *name);
120     // ****************** MP4 ****************** //
```

▲ 圖 3-7

```
126 private:
127     // ****************** MP4 ****************** //
128     void SplitPath(char *path, char *dirName, char *fileName);
```

▲ 圖 3-8

首先在 Create()中（圖 3-9、圖 3-10），傳入的 name 不再是一定在 root 下了，因此需要先使用 SplitPath()，將 name 切為所在 directory 名稱及 file 名稱，並且先從 root 中呼叫 GetDirSector()來找到此 directory 所在的 sector 並開啟後，再依照原先的步驟，在這個 directory 下建立新的 file。

```
178 bool FileSystem::Create(char *name, int initialSize)
179 {
180     // ****************** MP4 ****************** //
181     Directory *root, *directory;
182     PersistentBitmap *freeMap;
183     OpenFile *file;
184     FileHeader *hdr;
185     int sector, dirSector;
186     bool success;
187     char dirName[256], fileName[10];
188     SplitPath(name, dirName, fileName);
189
190     DEBUG(dbgFile, "Creating file " << fileName << " siz
191
192     root = new Directory(NumDirEntries);
193     directory = new Directory(NumDirEntries);
194     root->FetchFrom(directoryFile);
195     dirSector = root->GetDirSector(dirName);
196     file = new OpenFile(dirSector);
197     directory->FetchFrom(file);
```

▲ 圖 3-9

```
199     if (directory->Find(fileName) != -1)
200         success = FALSE; // file is already in directory
201     else
202     {
203         freeMap = new PersistentBitmap(freeMapFile, NumSectors);
204         sector = freeMap->FindAndSet(); // find a sector to hold
205         if (sector == -1)
206             success = FALSE; // no free block for file header
207         else if (!directory->Add(fileName, sector, FALSE))
208             success = FALSE; // no space in directory
209         else
210         {
211             hdr = new FileHeader;
212             if (!hdr->Allocate(freeMap, initialSize)){
213                 success = FALSE; // no space on disk for data
214             }
215             else
216             {
217                 success = TRUE;
218                 // everthing worked, flush all changes back to di
219                 hdr->WriteBack(sector);
220                 directory->WriteBack(file);
221                 freeMap->WriteBack(freeMapFile);
222             }
223             delete hdr;
224         }
225         delete freeMap;
226     }
227     delete root;
228     delete directory;
229     delete file;
230     return success;
231     // ****************** MP4 ****************** //
```

▲ 圖 3-10

在 Open()中（圖 3-11），將原先的 Find()改為使用 GetDirSector()，因為路徑同樣可能是多層的。

```
244 OpenFile * FileSystem::Open(char *name)
245 {
246     Directory *directory = new Directory(NumDirEntries);
247     OpenFile *openFile = NULL;
248     int sector;
249
250     DEBUG(dbgFile, "Opening file" << name);
251     directory->FetchFrom(directoryFile);
252     // ****************** MP4 ****************** //
253     sector = directory->GetDirSector(name);
254     // ****************** MP4 ****************** //
255     if (sector >= 0)
256         openFile = new OpenFile(sector); // name was four
257     delete directory;
258     return openFile; // return NULL if not found
```

▲ 圖 3-11

在 CreateDirectory()中（圖 3-12），與 Create()相似，都是先切開路徑並找到 directory，差別在於第 393 行時，Add()的參數 isDir 要設為 TRUE，代表這是一個 directory，還有第 408~411 行時，要將此 sector 的資料初始化為 directory，並且存回 disk 裡（圖 3-13）。

```
366  bool FileSystem::CreateDirectory(char *name){
367      Directory *root, *directory;
368      PersistentBitmap *freeMap;
369      OpenFile *file;
370      FileHeader *hdr;
371      int sector, dirSector;
372      bool success;
373      char dirName[256], fileName[10];
374      SplitPath(name, dirName, fileName);
375
376      DEBUG(dbgFile, "Creating Directory " << file
377
378      root = new Directory(NumDirEntries);
379      directory = new Directory(NumDirEntries);
380      root->FetchFrom(directoryFile);
381      dirSector = root->GetDirSector(dirName);
382      file = new OpenFile(dirSector);
383      directory->FetchFrom(file);
```

▲ 圖 3-12

```
385      if (directory->Find(fileName) != -1)
386          success = FALSE; // file is already in directory
387      else
388      {
389          freeMap = new PersistentBitmap(freeMapFile, NumSectors);
390          sector = freeMap->FindAndSet(); // find a sector to hold t
391          if (sector == -1)
392              success = FALSE; // no free block for file header
393          else if (!directory->Add(fileName, sector, TRUE))
394              success = FALSE; // no space in directory
395          else
396          {
397              hdr = new FileHeader;
398              if (!hdr->Allocate(freeMap, DirectoryFileSize)){
399                  success = FALSE; // no space on disk for data
400              }
401              else
402              {
403                  success = TRUE;
404                  // everthing worked, flush all changes back to dis
405                  hdr->WriteBack(sector);
406                  directory->WriteBack(file);
407                  freeMap->WriteBack(freeMapFile);
408                  OpenFile *f = new OpenFile(sector);
409                  Directory *d = new Directory(NumDirEntries);
410                  d->WriteBack(f);
411              }
412              delete hdr;
413          }
414          delete freeMap;
415      }
```

▲ 圖 3-13

在 RecursiveList()中（圖 3-14），同樣先找到此 directory 的 sector 並開啟後，呼叫 directory 的 RecursiveList()，傳入 0 代表現在在第 0 層。

```
422   void FileSystem::RecursiveList(char *name){
423       Directory *root = new Directory(NumDirEntries);
424       Directory *directory = new Directory(NumDirEntries);
425       root->FetchFrom(directoryFile);
426       int sector = root->GetDirSector(name);
427       OpenFile *file = new OpenFile(sector);
428       directory->FetchFrom(file);
429       directory->RecursiveList(0);
430       delete root;
431       delete directory;
432       delete file;
433   }
```

▲ 圖 3-14

在 SplitPath()中（圖 3-15），主要在處理字串，直接將切開後的兩個字串，複製進 dirName 和 fileName 裡。

在 List()中（圖 3-16），與 RecursiveList()大致相同，只是在最後改為呼叫此 directory 的 List()就好，不需要 recursive。

```
435   void FileSystem::SplitPath(char *path, char *dirName, char *fileName){
436       int len = strlen(path), i;
437       for(i = len-1; i >= 0; i--){
438           if(path[i] == '/') break;
439       }
440       strncpy(dirName, path, i);
441       strncpy(fileName, path+i+1, len-(i+1));
442       dirName[i] = fileName[len-(i+1)] = '\0';
443       if(i == 0) dirName[0] = '/', dirName[1] = '\0';
444   }
445   // ****************** MP4 ****************** //
```

▲ 圖 3-15

```
452   void FileSystem::List(char *name)
453   {
454       // ****************** MP4 ****************** //
455       Directory *directory = new Directory(NumDirEntries);
456       OpenFile *file;
457
458       directory->FetchFrom(directoryFile);
459       int sector = directory->GetDirSector(name);
460       file = new OpenFile(sector);
461       directory->FetchFrom(file);
462       directory->List();
463       delete file;
464       delete directory;
465       // ****************** MP4 ****************** //
```

▲ 圖 3-16

將以上 FileSystem 的功能都做好後，main()裡面就可以根據 “-l” 或 “lr”，呼叫 List()或 RecursiveList()（圖 3-17），在 CreateDirectory()中同樣呼叫 fileSystem 的 CreateDirectory()（圖 3-18）。

```
343     if (dirListFlag)
344     {
345         // ****************** MP4 ****************** //
346         if(recursiveListFlag) kernel->fileSystem->RecursiveList(listDirectoryName);
347         else kernel->fileSystem->List(listDirectoryName);
348         // ****************** MP4 ****************** //
349     }
```

▲ 圖 3-17

```
154   static void CreateDirectory(char *name)
155   {
156       // MP4 Assignment
157       // ******************** MP4 ******************** //
158       kernel->fileSystem->CreateDirectory(name);
159       // ******************** MP4 ******************** //
160   }
```

▲ 圖 3-18

(2) Support up to 64 files/subdirectories per directory

將 NumDirEntries 設為 64，即可在 directory 下放入 64 個 file。

```
42  #define FreeMapFileSize (NumSectors / BitsInByte)
43  // ******************** MP4 ******************** //
44  #define NumDirEntries 64
45  // ******************** MP4 ******************** //
46  #define DirectoryFileSize (sizeof(DirectoryEntry) * NumDirEntries)
```

▲ 圖 3-19

4. Bonus

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

我們在 part2 所採用的是 allocation 方式為 linked index scheme，理論上 file size 不管多大，header 都能夠裝得下。而從 disk.h 中可以發現原本的 disk 大小為 128*32*32 = 128KB，因此可以知道限制是在 disk 的大小，因此只需要改變 NumTracks 的大小為 32*512 = 16384（圖 4-1），disk 的大小即為 64MB，就可以使一個 file 的大小最高可到 64MB。

```
50    const int SectorSize = 128;   // number of bytes per disk sector
51    const int SectorsPerTrack  = 32;  // number of sectors per disk track
52    // ******************** MP4 ******************** //
53    const int NumTracks = 16384;     // number of tracks per disk
54    // ******************** MP4 ******************** //
55    const int NumSectors = (SectorsPerTrack * NumTracks);
```

▲ 圖 4-1

(2) Multi-level header size

在 FileHeader 的 Print()中（圖 4-2），我新增了 header size 的顯示作為驗證，由於我們使用的是 linked index scheme，越大的 file size 會有越大的 header size。header size 可由 file size 計算得到，每個 header 最多只能儲存 MaxFileSize 大小的檔案，算出所需幾個 FileHeader 後，再乘上 sizeof(FileHeader)，即為檔案的 header 所佔據的空間。

以下附上 bonus2_1.txt（5KB）、bonus2_2.txt（10KB）、bonus2_3.txt（15KB）等三個不同大小的檔案，複製進 nachos 後，可看出具有三種不同大小的 header，分別為 264B、396B、660B，也就是 2 個、3 個、5 個 FileHeader（圖 4-3）。

```
204   void FileHeader::Print()
205   {
206       printf("\nFileHeader contents.  File size: %d. Header size: %d.  File blocks:\n", \
207           FileLength(), sizeof(FileHeader)*divRoundUp(FileLength(), MaxFileSize));
208       PrintBlock();
209       printf("\nFile contents:\n");
210       PrintContent();
```

▲ 圖 4-2

```
Name: b1, Sector: 13
FileHeader contents.  File size: 5048. Header size: 264.  File blocks:
14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 44 45 46 47 48 49 50 51 52 53 54
File contents:
```

```
Name: b2, Sector: 55
FileHeader contents.  File size: 10098. Header size: 396.  File blocks:
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 86 87 88 89 90 91 92 93 94 95 96 97
 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 116 117 118 119 120 121 122 123 124 125 126 127 128 129
 130 131 132 133 134 135 136
File contents:
```

```
Name: b3, Sector: 137
FileHeader contents.  File size: 15148. Header size: 660.  File blocks:
138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 168 16
9 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 198 199 200
201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 228 229 230 231 23
2 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 258 259 260
File contents:
```

▲ 圖 4-3

(3) Recursive operations on directories

與 part3 一樣，需要從底層的 Directory 開始修改，到上層的 FileSystem。首先在 Directory 新增 FindIsDir()，功能為以檔案名稱搜尋，回傳此檔案是否為 directory（圖 4-5），GetTableSize()和 GetTable()分別回傳對應的資訊（圖 4-4）。

```
74      bool FindIsDir(char *name);
75      int GetTableSize() {return tableSize;}
76      DirectoryEntry* GetTable() {return table;}
77      // ******************** MP4 ******************** //
```

▲ 圖 4-4

```
242   bool Directory::FindIsDir(char *name){
243       int i = FindIndex(name);
244       return table[i].isDir;
245   }
246   // ******************** MP4 **********
```

▲ 圖 4-5

在 FileSystem 的 Remove()中我新增了一個參數判斷是否需要 recursive 的 remove（圖 4-6），一開始與 Create()相似，先對路徑做處理，找到 directory 及要 remove 的 directory 或 file 的名稱。接著如果需要 recursive remove 且為 directory 時，就先取得他的 directoryEntry table，並開始遞迴呼叫 Remove()，remove 掉每個 entry()（圖 4-7）。遞迴結束後，即可開始刪除此 directory 或 file，移除掉 data block 和 header block（圖 4-8）。

```
275   bool FileSystem::Remove(char *name, bool recursive)
276   {
277       // ******************** MP4 ******************** //
278       Directory *root, *directory;
279       PersistentBitmap *freeMap;
280       OpenFile *file;
281       FileHeader *fileHdr;
282       int sector, dirSector;
283       char dirName[256], fileName[10];
284       SplitPath(name, dirName, fileName);
285
286       root = new Directory(NumDirEntries);
287       directory = new Directory(NumDirEntries);
288       root->FetchFrom(directoryFile);
289       dirSector = root->GetDirSector(dirName);
290       file = new OpenFile(dirSector);
291       directory->FetchFrom(file);
292
293       sector = directory->Find(fileName);
```

▲ 圖 4-6

```
295     if(recursive){
296         bool isDir = directory->FindIsDir(fileName);
297         if(isDir){
298             OpenFile *f = new OpenFile(sector);
299             Directory *d = new Directory(NumDirEntries);
300             d->FetchFrom(f);
301             int tableSize = d->GetTableSize();
302             DirectoryEntry *table = d->GetTable();
303             for(int i = 0; i < tableSize; i++){
304                 if(table[i].inUse){
305                     char curName[256] = {};
306                     strcpy(curName, name);
307                     strcat(curName, "/");
308                     strcat(curName, table[i].name);
309                     Remove(curName, TRUE);
310                 }
311             }
312         }
313     }
```

▲ 圖 4-7

```
315     fileHdr = new FileHeader;
316     fileHdr->FetchFrom(sector);
317
318     freeMap = new PersistentBitmap(freeMapFile, NumSectors);
319
320     fileHdr->Deallocate(freeMap); // remove data blocks
321     freeMap->Clear(sector);       // remove header block
322     directory->Remove(fileName);
323
324     freeMap->WriteBack(freeMapFile);     // flush to disk
325     directory->WriteBack(file); // flush to disk
326     delete fileHdr;
327     delete root;
328     delete directory;
329     delete freeMap;
330     delete file;
331     return TRUE;
332     // ****************** MP4 ****************** //
```

▲ 圖 4-8

最後在 main()裡，傳入 recursiveListFlag，表示是否需要遞迴（圖 4-9）。

```
329     if (removeFileName != NULL)
330     {
331         // ****************** MP4 ****************** //
332         kernel->fileSystem->Remove(removeFileName, recursiveListFlag);
333         // ****************** MP4 ****************** //
334     }
```

▲ 圖 4-9