OS MP4 Report

Team21

葉宥忻 109062301

陳禹勳 109062134

contribution: 一起

Trace code

Kernel::Initialize()

```
115    #else
116        fileSystem = new FileSystem(formatFlag);
```

創一個新的FileSystem，formatFlag若為true則進行格式化(代表現在disk為空的，所以要initialize disk 讓他有empty directory和bitmap for free sectors)；若為false則直接打開directory和bitmap就好。

FileSystem::FileSystem()

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

86、87行: 創新的bitmap和directory

88、89行: bitmap和directory也是file的一種，所以需要fileheader(FCB)

```
58    #define FreeMapSector 0
59    #define DirectorySector 1
```

```
93          // First, allocate space for FileHeaders for the directory and bitmap
94          // (make sure no one else grabs these!)
95          freeMap->Mark(FreeMapSector);
96          freeMap->Mark(DirectorySector);
```

把剛創好的bitmap中第0個sector(for FreeMap fileheader)，和第1個sector(for directory fileheader)標記起來，以後不能放其他file

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

在disk上分配sector給bit map和directory

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

把bit map和 directory的fileheader寫回disk，在open file前要先做這個，因為Open指令會把fileheader從disk讀出來

```
117         freeMapFile = new OpenFile(FreeMapSector);
118         directoryFile = new OpenFile(DirectorySector);
```

接著打開它們

```
127         freeMap->WriteBack(freeMapFile); // flush changes to disk
128         directory->WriteBack(directoryFile);
```

再把它們寫回disk

```
140     else
141     {
142         // if we are not formatting the disk, just open the files representing
143         // the bitmap and directory; these are left open while Nachos is running
144         freeMapFile = new OpenFile(FreeMapSector);
145         directoryFile = new OpenFile(DirectorySector);
```

如果format為false，表示之前創過這兩個檔案了，直接打開就好

bool FileSystem::Create(char *name, int initialSize)

```
190  int FileSystem::Create(char *name, int initialSize)
191  {
192      Directory *directory;
193      PersistentBitmap *freeMap;
194      FileHeader *hdr;
195      int sector;
196      bool success;
197
198      DEBUG(dbgFile, "Creating file " << name << " size " << initialSize);
199
200      directory = new Directory(NumDirEntries);
201      directory->FetchFrom(directoryFile);
202
203      if (directory->Find(name) != -1)
204          success = FALSE; // file is already in directory
205      else
206      {
207          freeMap = new PersistentBitmap(freeMapFile, NumSectors);
208          sector = freeMap->FindAndSet(); // find a sector to hold the file header
209          if (sector == -1)
210              success = FALSE; // no free block for file header
211          else if (!directory->Add(name, sector))
212              success = FALSE; // no space in directory
213          else
214          {
215              hdr = new FileHeader;
216              if (!hdr->Allocate(freeMap, initialSize))
217                  success = FALSE; // no space on disk for data
218              else
219              {
220                  success = TRUE;
221                  // everthing worked, flush all changes back to disk
222                  hdr->WriteBack(sector);
223                  directory->WriteBack(directoryFile);
224                  freeMap->WriteBack(freeMapFile);
225              }
226              delete hdr;
227          }
228          delete freeMap;
229      }
230      delete directory;
231      return success;
232  }
```

201把在disk裡的dirctory讀出來

203行檢查有沒有同名的file

208行檢查有沒有空的sector可以存fileheader，有的話回傳

211行檢查directory有沒有位置

216行檢查disk有沒有空位

若以上都成功的話，就把剛創的file header、修改過的directory和bitmap寫回disk

OpenFile * FileSystem::Open(char *name)

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

Open a file for reading and writing.

253行在directory找所要的fileheader的位置，255行Open a file whose header is located at "sector" on the disk(把找到的fileheader 帶到memory)

bool FileSystem::Remove(char *name)

```
290  bool FileSystem::Remove(char *name)
291  {
292      Directory *directory;
293      PersistentBitmap *freeMap;
294      FileHeader *fileHdr;
295      int sector;
296
297      directory = new Directory(NumDirEntries);
298      directory->FetchFrom(directoryFile);
299      sector = directory->Find(name);
300      if (sector == -1)
301      {
302          delete directory;
303          return FALSE; // file not found
304      }
305      fileHdr = new FileHeader;
306      fileHdr->FetchFrom(sector);
307
308      freeMap = new PersistentBitmap(freeMapFile, NumSectors);
309
310      fileHdr->Deallocate(freeMap); // remove data blocks
311      freeMap->Clear(sector);       // remove header block
312      directory->Remove(name);
313
314      freeMap->WriteBack(freeMapFile);     // flush to disk
315      directory->WriteBack(directoryFile); // flush to disk
316      delete fileHdr;
317      delete directory;
318      delete freeMap;
319      return TRUE;
320  }
```

把file從file system中刪除。

300行檢查有沒有這個name的file

310行移除這個file的所有data block

311行把header的sector清空(在bitmap的值從1設成 0)

312行清掉directory的entry

314、315把更新過的bitmap和directory寫回disk

FileHeader::Allocate()

```
69  bool FileHeader::Allocate(PersistentBitmap *freeMap, int fileSize)
70  {
71      numBytes = fileSize;
72      numSectors = divRoundUp(fileSize, SectorSize);
73      if (freeMap->NumClear() < numSectors)
74          return FALSE; // not enough space
75
76      for (int i = 0; i < numSectors; i++)
77      {
78          dataSectors[i] = freeMap->FindAndSet();
79          // since we checked that there was enough free space,
80          // we expect this to succeed
81          ASSERT(dataSectors[i] >= 0);
82      }
83      return TRUE;
84  }
```

初始化fileheader(FCB)。72行先算需要的sector數量，73行檢查bitmap裡有沒有足夠可用的sector，夠的話執行76行。

76行: dataSectors陣列存的是該file用到哪些sector，這裡的意思是在bitmap中找到可用的sector，然後把它給該file。

void FileHeader::Deallocate(PersistentBitmap *freeMap)

```
93  void FileHeader::Deallocate(PersistentBitmap *freeMap)
94  {
95      for (int i = 0; i < numSectors; i++)
96      {
97          ASSERT(freeMap->Test((int)dataSectors[i])); // ought to be marked!
98          freeMap->Clear((int)dataSectors[i]);
99      }
100 }
```

用for loop把file用到的那些data block release

FileHeader::WriteBack(int sector)

```
126  void FileHeader::WriteBack(int sector)
127  {
128      kernel->synchDisk->WriteSector(sector, (char *)this);
129
130      /*
131          MP4 Hint:
132          After you add some in-core informations, you may not want to write all fields into disk.
133          Use this instead:
134          char buf[SectorSize];
135          memcpy(buf + offset, &dataToBeWritten, sizeof(dataToBeWritten));
136          ...
137      */
138  }
```

把改過的filehdr寫回disk的sector

OpenFile::OpenFile(int sector)

```
29  OpenFile::OpenFile(int sector)
30  {
31      hdr = new FileHeader;
32      hdr->FetchFrom(sector);
33      seekPosition = 0;
34  }
```

紀錄file的file head，初始化seekPosition

void SyncDisk::ReadSector(int sectorNumber, char *data)

```
56  void SynchDisk::ReadSector(int sectorNumber, char *data)
57  {
58      lock->Acquire(); // only one disk I/O at a time
59      disk->ReadRequest(sectorNumber, data);
60      semaphore->P(); // wait for interrupt
61      lock->Release();
62  }
```

```
73 void SynchDisk::WriteSector(int sectorNumber, char *data)
74 {
75     lock->Acquire(); // only one disk I/O at a time
76     disk->WriteRequest(sectorNumber, data);
77     semaphore->P(); // wait for interrupt
78     lock->Release();
79 }
```

SyncDisk會使用semaphore和lock讓readSector和writeSector可以正常執行

Part I. Understanding Nachos file system

(1) Explain how the NachOS FS manages and finds free block space?

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

Ans:

```
#define FreeMapSector 0
```

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

```
OpenFile *freeMapFile;   // Bit map of free disk blocks,
                         // represented as a file
```

```
freeMap = new PersistentBitmap(freeMapFile, NumSectors);
```

```
int numBits;         // number of bits in the bitmap
int numWords;        // number of words of bitmap storage
                     // (rounded up if numBits is not a
                     //  multiple of the number of bits in
                     //  a word)
unsigned int *map; // bit storage     You, 2 weeks ago • m
```

file system用上述trace的方式初始化後，Bitmap fileheader放在sector0，在construct file system時就會open bitmapfile，需要存取bitmap時construct PersistenBimap，freemap中map的bit會記錄哪些sector用過(記為1)，哪些沒用過(記為0)

```
57  void Bitmap::Mark(int which)
58  {
59      ASSERT(which ≥ 0 && which < numBits);
60
61      map[which / BitsInWord] |= 1 << (which % BitsInWord);
62
63      ASSERT(Test(which));
64  }
```

將map第n個bit設成1

```
73  void Bitmap::Clear(int which)
74  {
75      ASSERT(which ≥ 0 && which < numBits);
76
77      map[which / BitsInWord] &= ~(1 << (which % BitsInWord));
78
79      ASSERT(!Test(which));
80  }
```

Clear map的第n個bit

```
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;
123  }
```

找bitmap第一個0，也就是空的sector。若有找到則把它設成1，表示該sector被佔用了

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

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

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

Disk size = SectorSize* SectorsPerTrack * NumTrack + MagicSize

32*32*128 = 128KB (+4 bytes for MagicSize)

(3) Explain how the NachOS FS manages the directory data structure?

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

```
#define DirectorySector 1
```

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

```
OpenFile *directoryFile; // "Root" directory -- list of
                         // file names, represented as a file
```

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

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

file system用上述trace的方式初始化後，Directory fileheader放在sector1，在construct file system時就會open directoryFile，需要存取Directory時construct Directory並Fetch from directoryFile

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

Directory的table記錄所有directoryEntry，並初始化每個entry的inUse為FALSE

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

每個DirectoryEntry會記錄是否被使用(inUse)，fileheader所在sector(sector)，以及file name(name)

```
108 int Directory::Find(char *name)
109 {
110     int i = FindIndex(name);
111
112     if (i != -1)
113         return table[i].sector;
114     return -1;
115 }
```

Directory::Find可以幫我們找file name對應的sector

```
91  int Directory::FindIndex(char *name)
92  {
93      for (int i = 0; i < tableSize; i++)
94          if (table[i].inUse && !strncmp(table[i].name, name, FileNameMaxLen))
95              return i;
96      return -1; // name not in directory
97  }
```

Directory::FindIndex可以找filename是在table的哪個index

```
128 bool Directory::Add(char *name, int newSector)
129 {
130     if (FindIndex(name) != -1)
131         return FALSE;
132
133     for (int i = 0; i < tableSize; i++)
134         if (!table[i].inUse)
135         {
136             table[i].inUse = TRUE;
137             strncpy(table[i].name, name, FileNameMaxLen);
138             table[i].sector = newSector;
139             return TRUE;
140         }
141     return FALSE; // no space.  Fix when we have extensible files.
142 }
```

Directory::Add先檢查file name是否已經存在，再尋找可用的的table entry，寫入對應的file name和newSector number

```
152  bool Directory::Remove(char *name)
153  {
154      int i = FindIndex(name);
155
156      if (i == -1)
157          return FALSE; // name not in directory
158      table[i].inUse = FALSE;
159      return TRUE;
160  }
```

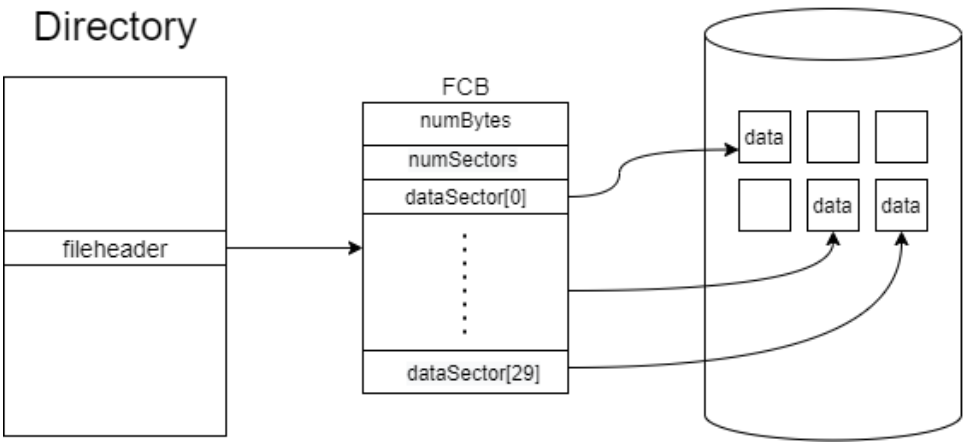
Directory::Remove幫我們free directory的table(inUse = False)

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

filehdr.h

```
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
```

numBytes紀錄file用到的Byte數; numSector紀錄file用到的sector數; dataSectors[]紀錄用到的Sector在disk上的位置。



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

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

因為現在disk的allocate方式是index，而現在每個Fileheader裡面能放最多的index數量為((128 - 2 * 4) / 4) = 30個，也就是一個file最多只能用30個sector。所以每個file的容量是30*128(bytes per sector) 約4KB

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

(2)

exception.cc

```
79          case SC_Create:
80              val = kernel->machine->ReadRegister(4);
81              {
82                  char* filename = &(kernel->machine->mainMemory[val]);
83                  // MP4
84                  int size = kernel->machine->ReadRegister(5);
85                  status = SysCreate(filename, size);
86                  kernel->machine->WriteRegister(2, (int)status);
87              }
88              kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
89              kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
90              kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
91              return;
92              ASSERTNOTREACHED();
93              break;
```

新增SC_create、SC_write、SC_open等system call

ksyscall.cc

```
29  int SysCreate(char *filename, int size)
30  {
31      // return value
32      // 1: success
33      // 0: failed
34      return kernel->fileSystem->Create(filename, size);
35  }
```

新增SysCreate()、SysOpen()、Syswrite()，會去kernal->fileSystem呼叫在fileSystem實做的Create、Open等function

filesys.h

```
125     // MP4
126     OpenFile *opfile;
```

opfile表在filesyetem已開啟的檔案

filesys.cc

```
369 // MP4
370 OpenFileId FileSystem::OpenAFile(char *name){
371     opfile = Open(name);
372     return 1;
373 }
374
375 // MP4
376 int FileSystem::Read(char *buffer, int size, OpenFileId id){
377     return opfile->Read(buffer, size);
378 }
379
380 // MP4
381 int FileSystem::Write(char *buffer, int size, OpenFileId id){
382     return opfile->Write(buffer, size);
383 }
384
385 // MP4
386 int FileSystem::Close(OpenFileId id){
387     opfile = NULL;
388     return 1;
389 }
```

filesystem的read、write會往下傳到Openfile.cc，然後在執行那邊的read、write；Close則是把在filesystem打開的opfile設成NULL

(3) Allocate()

filehdr.h

```
24  #define MaxFileSizeLevel1 (NumDirect * SectorSize)  // 30 * 128
25  #define MaxFileSizeLevel2 (NumDirect * NumDirect * SectorSize)  // 30^2 * 128
26  #define MaxFileSizeLevel3 (NumDirect * NumDirect * NumDirect * SectorSize)  // 30^3 * 128
27  #define MaxFileSizeLevel4 (NumDirect * NumDirect * NumDirect * NumDirect * SectorSize)  // 30^4 * 128
```

我們是用multiLevel的index allocation實作，先定義不同level的filesize

filehdr.cc

```
69  bool FileHeader::Allocate(PersistentBitmap *freeMap, int fileSize)
70  {
71      numBytes = fileSize;
72
73      // MP4
74      numSectors = divRoundUp(fileSize, SectorSize);
75      int totalNumSectors = numSectors;
76      if (numBytes > MaxFileSizeLevel3) {
77          totalNumSectors += divRoundUp(numSectors, NumDirect);
78          totalNumSectors += divRoundUp(numSectors, NumDirect*NumDirect);
79          totalNumSectors += divRoundUp(numSectors, NumDirect*NumDirect*NumDirect);
80      }else if (numBytes > MaxFileSizeLevel2) {
81          totalNumSectors += divRoundUp(numSectors, NumDirect);
82          totalNumSectors += divRoundUp(numSectors, NumDirect*NumDirect);
83      }else if (numBytes > MaxFileSizeLevel1) {
84          totalNumSectors += divRoundUp(numSectors, NumDirect);
85      }
86      if (numSectors > NumDirect) {
87          numSectors = NumDirect;
88      }
89      if (freeMap->NumClear() < totalNumSectors) {
90          return FALSE;
91      }
```

74行把file需要的sector算出來，76~84把在不同level的所需之總共sector算出來，86行意思是:若需要的Sector超一個FCB能放的sector，則把FCB所有空間都拿來放sector，90行看disk空間夠不夠，不夠的話就直接回傳FALSE

```
98  >   if (numBytes > MaxFileSizeLevel3){···
118 >   else if (numBytes > MaxFileSizeLevel2) {···
138 >   else if (numBytes > MaxFileSizeLevel1) {···
158     else {
159         for (int i = 0; i < numSectors; i++)
160         {
161             dataSectors[i] = freeMap->FindAndSet();
162             // since we checked that there was enough free space,
163             // we expect this to succeed
164             ASSERT(dataSectors[i] >= 0);
165         }
166     }
```

接著再根據不同Filesize去遞回呼叫Allocate，以level3為例(其他level做法一樣):

```
98      if (numBytes > MaxFileSizeLevel3){
99          for (int i = 0; i < numSectors; i++) {
100             if (fileSize <= 0) break;
101
102             dataSectors[i] = freeMap->FindAndSet();
103             FileHeader* subHdr = new FileHeader();
104
105             if (fileSize >= MaxFileSizeLevel3) {
106                 subHdr->Allocate(freeMap, MaxFileSizeLevel3);
107                 fileSize -= MaxFileSizeLevel3;
108                 subHdr->WriteBack(dataSectors[i]);
109             }else {
110                 subHdr->Allocate(freeMap, fileSize);
111                 fileSize -= fileSize;
112                 subHdr->WriteBack(dataSectors[i]);
113             }
114
115             ASSERT(dataSectors[i] >= 0);
116         }
117     }
```

在for迴圈裡面，先去bitmap找空的sector給dataSector[i]，接著看fileSize有沒有大於MaxFileSizeLevel3，若有的話就把一個FCB最多所能放data的數量，全部拿去放下一層的FCB；若沒有的話就只需要把剩下的fileSize allocate進subHdr就好。

107、111行把原fileSize更新，也就是扣掉那些已分配給Level3的部分，108、112行把subHdr寫回disk。

```
158     else {
159         for (int i = 0; i < numSectors; i++)
160         {
161             dataSectors[i] = freeMap->FindAndSet();
162             // since we checked that there was enough free space,
163             // we expect this to succeed
164             ASSERT(dataSectors[i] >= 0);
165         }
166     }
```

直到最下層，才是資料真正放在sector上

Deallocate

```
186  void FileHeader::Deallocate(PersistentBitmap *freeMap)
187  {
188      if (numBytes <= MaxFileSizeLevel1) {
189          for (int i = 0; i < numSectors; i++){
190              ASSERT(freeMap->Test((int)dataSectors[i])); // ought to be marked!
191              freeMap->Clear((int)dataSectors[i]);
192          }
193      }else {
194          for (int i = 0; i < numSectors; i++) {
195              ASSERT(freeMap->Test((int)dataSectors[i])); // ought to be marked!
196
197              FileHeader* subHdr = new FileHeader;
198              subHdr->FetchFrom(dataSectors[i]);
199              subHdr->Deallocate(freeMap);
200
201              freeMap->Clear((int)dataSectors[i]);
202          }
203      }
```

188行: 如果numBytes 小於等於level1的容量的話表示只有用到一層，因此用for迴圈把每個dataSector清掉就好；否則的話會去dataSector[i]取得下一層的dataSector，用遞回呼叫的方式，從最底層往上清。

ByteToSector()

```
259  int FileHeader::ByteToSector(int offset)
260  {
261      if (numBytes > MaxFileSizeLevel3) {
262          FileHeader *subHdr = new FileHeader;
263          int dataSectorIndex = offset / MaxFileSizeLevel3;
264          subHdr->FetchFrom(dataSectors[dataSectorIndex]);
265
266          return subHdr->ByteToSector(offset - MaxFileSizeLevel3 * dataSectorIndex);
267      }
268      else if (numBytes > MaxFileSizeLevel2) {
269          FileHeader *subHdr = new FileHeader;
270          int dataSectorIndex = offset / MaxFileSizeLevel2;
271          subHdr->FetchFrom(dataSectors[dataSectorIndex]);
272
273          return subHdr->ByteToSector(offset - MaxFileSizeLevel2 * dataSectorIndex);
274      }
275      else if (numBytes > MaxFileSizeLevel1) {
276          FileHeader *subHdr = new FileHeader;
277          int dataSectorIndex = offset / MaxFileSizeLevel1;
278          subHdr->FetchFrom(dataSectors[dataSectorIndex]);
279
280          return subHdr->ByteToSector(offset - MaxFileSizeLevel1 * dataSectorIndex);
281      }
282      else {
283          return dataSectors[offset / SectorSize];
284      }
285      // return (dataSectors[offset / SectorSize]);
286  }
```

把virtual sector number轉成 physical sector number。除了最下層，每層都會把offset扣掉在該層已經allocate過的filesize然後繼續呼叫ByteToScetor往下層找。到最下層之後就直接回傳dataSector[offset/ SectorSize]。例如offset = 3、SectorSize=2，因為3/2 = 1.5，所以該byte就會放在dataSector[1](從dataSector[0]開始數)。

Print()

```
304  void FileHeader::Print()
305  {
306      int i, j, k;
307      char *data = new char[SectorSize];
308
309      printf("FileHeader contents.  File size: %d.  File blocks:\n", numBytes);
310      for (i = 0; i < numSectors; i++)
311          printf("%d ", dataSectors[i]);
312      printf("\nFile contents:\n");
313
314      for (i = k = 0; i < numSectors; i++)
315      {
316          if (numBytes > MaxFileSizeLevel1) {
317              FileHeader *subHdr = new FileHeader;
318              subHdr->FetchFrom(dataSectors[i]);
319              subHdr->Print();
320          }
321          else {
322              kernel->synchDisk->ReadSector(dataSectors[i], data);
323              for (j = 0; (j < SectorSize) && (k < numBytes); j++, k++)
324              {
325                  if ('\040' <= data[j] && data[j] <= '\176') // isprint(data[j])
326                      printf("%c", data[j]);
327                  else
328                      printf("\\%x", (unsigned char)data[j]);
329              }
330              printf("\n");
331
332              delete[] data;
333          }
334      }
335  }
```

和ByteToSector類似，地回呼叫到最下層後把content print出來

Part III. Modify the file system code to support the subdirectory

CreateDirectory()

```
191  void FileSystem::CreateDirectory(char *name)
192  {
193      // Root Directory
194      Directory *directory;
195      directory = new Directory(NumDirEntries);
196      directory->FetchFrom(directoryFile);
197      OpenFile *currDirFile = directoryFile;
198
199       PersistentBitmap *freeMap = new PersistentBitmap(freeMapFile, NumSectors);
200      int sector;
```

195、196行: 創一個directory，去接整個fileSystem的diretory，也就是root dircetory。197行: currDirFile指的是現在trace到了的Dircetory的pointer，194行: directory指的是現在trace到的dirctory其表格實際的樣子，所以之後可以用 int Find(token) 來找表格上有沒有我們要的subDirectory

接下來就是把path name切成token

```
203      char *token = strtok(name, "/");
204      while (true) {
205          if (directory->Find(token) != -1) {
206              sector = directory->Find(token);
207
208              DEBUG(dbgFile, token << " at sector num " << sector);
209
210              OpenFile* subDirFile = new OpenFile(sector);
211              directory->FetchFrom(subDirFile);
212              currDirFile = subDirFile;
213          }else {
```

從第一個token開始，206行: 若找到名為token的Directory，就呼叫Find()去找這個Directory在哪一個sector上，210行: 接著用這個sector去打開subDirFile，在211行把其Table交給directory，212行把currDirFile設成找到的subDirFile。

Create()

```
262     while (true) {
263         token = strtok(NULL, "/");
264         if (token == NULL) {
265             break;
266         }
267 
268         sector = directory->Find(prevToken);
269 
270         OpenFile* subDirFile = new OpenFile(sector);
271         directory->FetchFrom(subDirFile);
272         currDirFile = subDirFile;
273 
274         DEBUG(dbgFile, prevToken << " at sector num " << sector);
275 
276         prevToken = token;
277     }
```

拆token的方式和CreateDirectory一樣，差別在會記錄prevToken，因為我們要在最後Create名為token的file，否則跳出迴圈時token會變NULL的話就找不到了。

```
281     if (directory->Find(prevToken) != -1)
282         success = FALSE; // file is already in directory
283     else
284     {
285         freeMap = new PersistentBitmap(freeMapFile, NumSectors);
286         sector = freeMap->FindAndSet(); // find a sector to hold the file header
287 
288         if (sector == -1)
289             success = FALSE; // no free block for file header
290         else if (!directory->Add(prevToken, sector, FALSE))
291             success = FALSE; // no space in directory
292         else
293         {
294             hdr = new FileHeader;
295             if (!hdr->Allocate(freeMap, initialSize))
296                 success = FALSE; // no space on disk for data
297             else
298             {
299                 success = TRUE;
300                 // everthing worked, flush all changes back to disk
301                 DEBUG(dbgFile, "file's FCB at " << sector);
302 
303                 hdr->WriteBack(sector);
304                 directory->WriteBack(currDirFile);
305                 freeMap->WriteBack(freeMapFile);
306             }
307             delete hdr;
308         }
309         delete freeMap;
310     }
311     delete directory;
```

接下來Create動作就和之前一樣了

```
279     DEBUG(dbgFile, "filename is " << prevToken);
280
281     if (directory->Find(prevToken) != -1)
282         success = FALSE; // file is already in directory
283     else
284     {
285         freeMap = new PersistentBitmap(freeMapFile, NumSectors);
286         sector = freeMap->FindAndSet(); // find a sector to hold the file header
287
288         if (sector == -1)
289             success = FALSE; // no free block for file header
290         else if (!directory->Add(prevToken, sector, FALSE))
291             success = FALSE; // no space in directory
292         else
293         {
294             hdr = new FileHeader;
295             if (!hdr->Allocate(freeMap, initialSize))
296                 success = FALSE; // no space on disk for data
297             else
298             {
299                 success = TRUE;
300                 // everthing worked, flush all changes back to disk
301                 DEBUG(dbgFile, "file's FCB at " << sector);
302
303                 hdr->WriteBack(sector);
304                 directory->WriteBack(currDirFile);
305                 freeMap->WriteBack(freeMapFile);
306             }
307             delete hdr;
308         }
309         delete freeMap;
310     }
311     delete directory;
```

撞名、空間不夠的話就return FALSE，都沒問題的話就allocate空間給 File，最後更動完directory、bitmap等等後就寫回disk。

Open()

```
339     while (true) {
340         token = strtok(NULL, "/");
341         if (token == NULL) {
342             break;
343         }
344 
345         sector = directory->Find(prevToken);
346 
347         DEBUG(dbgFile, prevToken << " at sector num " << sector);
348 
349         OpenFile* subDirFile = new OpenFile(sector);
350         directory->FetchFrom(subDirFile);
351 
352         prevToken = token;
353     }
```

```
359     sector = directory->Find(prevToken);
```

也是切成token一路往下找，也要記錄prevToken，因為要在最後打開名為last token的file，沒紀錄prevToken的話，最後跳出while迴圈token就變成NULL，也就找不到了。

Copy() - main.cc

```
// MP4
char saveFileName[300];
strcpy(saveFileName, to);
```

在main.cc中Copy會先create再進行open，會改到to的value，使open的時候發生錯誤，所以要先把to存起來

Remove()

```
418     while (true) {
419         token = strtok(NULL, "/");
420         if (token == NULL) {
421             break;
422         }
423 
424         // subDir header sector number
425         sector = directory->Find(prevToken);
426 
427         DEBUG(dbgFile, prevToken << " at sector num " << sector);
428 
429         OpenFile* subDirFile = new OpenFile(sector);
430         directory->FetchFrom(subDirFile);
431         currDirFile = subDirFile;
432 
433         prevToken = token;
434     }
```

一樣切token，且會記錄prevToken(最後要刪的file)和要刪的file的directory

```
438     directory = new Directory(NumDirEntries);
439     directory->FetchFrom(currDirFile);
440     sector = directory->Find(prevToken);
441     if (sector == -1)
442     {
443         delete directory;
444         return FALSE; // file not found
445     }
446     fileHdr = new FileHeader;
447     fileHdr->FetchFrom(sector);
448
449     freeMap = new PersistentBitmap(freeMapFile, NumSectors);
450
451     fileHdr->Deallocate(freeMap); // remove data blocks
452     freeMap->Clear(sector);       // remove header block
453     directory->Remove(prevToken);
454
455     freeMap->WriteBack(freeMapFile);     // flush to disk
456     directory->WriteBack(currDirFile); // flush to disk
457     delete fileHdr;
458     delete directory;
459     delete freeMap;
460     return TRUE;
461 }
```

440行~444行: 如果在最後的directory底下找不到名為prevToken的file就retuen false；找到的話就 remove data blocks(451行)、remove header block(452行)、remove its entry in directory(453行)，最後把改動寫回disk、delete刪掉local變數。

List()

```
479     while (true) {
480         if (token == NULL) {
481             break;
482         }
483 
484         sector = directory->Find(token);
485 
486         DEBUG(dbgFile, token << " at sector num " << sector);
487 
488         OpenFile* subDirFile = new OpenFile(sector);
489         directory->FetchFrom(subDirFile);
490 
491         token = strtok(NULL, "/");
492     }
493 
494     directory->List();
495     delete directory;
```

trace到最後一個token後，呼叫directory -> List去印出名為pathName中的最後一個token的directory下之每個file或directory。(如pathName是 /a/b/c，就印出在資料夾c底下的每個directory或file)。

```
168 void Directory::List()
169 {
170     // for (int i = 0; i < tableSize; i++)
171     //     if (table[i].inUse){
172     //         printf("%s\n", table[i].name);
173     //     }
174 
175     for (int i = 0; i < tableSize; i++){
176         if (table[i].inUse){
177             // MP4
178             char type = table[i].isDir ? 'D' : 'F';
179             printf("[%c]: %s\n", type, table[i].name);
180         }
181     }
182 }
```

把table放哪些directory或file印出來

filesys.cc

```
65    #define NumDirEntries 64
```

把directory最多可放的entry改成64個

directory.h

```
32    class DirectoryEntry
33    {
34    public:
35        bool inUse;
36        bool isDir;
```

```
67        bool Add(char *name, int newSector, bool isDir);
```

table新增一個isDir欄位，記錄他是directory還是file，且在add的時候也要記錄是加了directory還是file。

RecursiveList()

```
276           else if (strcmp(argv[i], "-lr") == 0)
277           {
278               // MP4 mod tag
279               // recursive list
280               ASSERT(i + 1 < argc);
281               listDirectoryName = argv[i + 1];
282               // dirListFlag = true;
283               recursiveListFlag = true;
284               i++;
285           }
```

```
353       if (recursiveListFlag)
354       {
355           kernel->fileSystem->RecursiveList(listDirectoryName);
356       }
```

```
498   void FileSystem::RecursiveList(char *name)
499   {
```

```
523         directory->RecursiveList(0);
524         delete directory;
525     }
```

和前面一樣，先拆解pathName，進到最後一層directory之後，再去呼叫directory -> RecursiveList()，把該directory的所有subDirectory和file印出來。

```
184 void Directory::RecursiveList(int level)
185 {
186     Directory *subDir = new Directory(10);
187     OpenFile *subDirFile;
188 
189     for (int i = 0; i < tableSize; i++){
190         if (table[i].inUse){
191             if (table[i].isDir) {
192                 for (int k = 0; k < level; k++) {
193                     printf("  ");
194                 }
195 
196                 printf("[D]: %s\n", table[i].name);
197 
198                 subDirFile = new OpenFile(table[i].sector);
199                 subDir->FetchFrom(subDirFile);
200                 subDir->RecursiveList(level+1);
```

for迴圈走過table的每個entry。若entry為directory，則先印出資料夾的名字(196行)，再用遞回的方式，往下印此資料夾底下的directory

```
202             }else {
203                 for (int k = 0; k < level; k++) {
204                     printf("  ");
205                 }
206                 printf("[F]: %s\n", table[i].name);
207             }
```

若為File，則直接印出來。level為空格的數量

BonusI. Enhance the NachOS to support even larger file size

disk.h

```
54    const int NumTracks = 32768;
```

filehdr.

```
27    #define MaxFileSizeLevel4 (NumDirect * NumDirect * NumDirect * NumDirect * SectorSize)  // 30^4 * 128
```

把NumTracks改成32768(128MB)，因為如果設成64MB，就會放不下64MB的file。剩下就是用前面提過的muti-level index的方式實作，因為最高level為4，故最大filesize為(30^4) * 128 約64MB

BonusII. Multi-level header size

```
int totalHdrSectors = 1;
if (numBytes > MaxFileSizeLevel3) {
    totalHdrSectors += divRoundUp(numSectors, NumDirect);
    totalHdrSectors += divRoundUp(numSectors, NumDirect*NumDirect);
    totalHdrSectors += divRoundUp(numSectors, NumDirect*NumDirect*NumDirect);
}else if (numBytes > MaxFileSizeLevel2) {
    totalHdrSectors += divRoundUp(numSectors, NumDirect);
    totalHdrSectors += divRoundUp(numSectors, NumDirect*NumDirect);
}else if (numBytes > MaxFileSizeLevel1) {
    totalHdrSectors += divRoundUp(numSectors, NumDirect);
}
DEBUG(dbgFile, "BonusII: Use " << totalHdrSectors << " blocks for header");
```

我們使用Combined Scheme根據file大小，讓他使用direct、single indirect等，增加file header數量讓file可以有更多的data sector

以下為num_100.txt、num_1000.txt、num_1000000.txt不同大小file總共使用到的file header數量

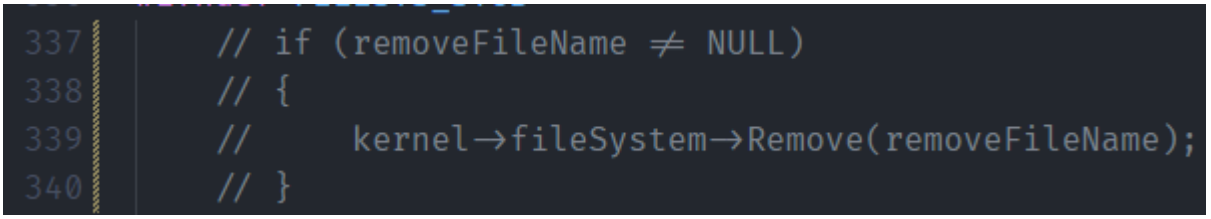
```
BonusII: Use 1 blocks for header
```

```
BonusII: Use 4 blocks for header
```

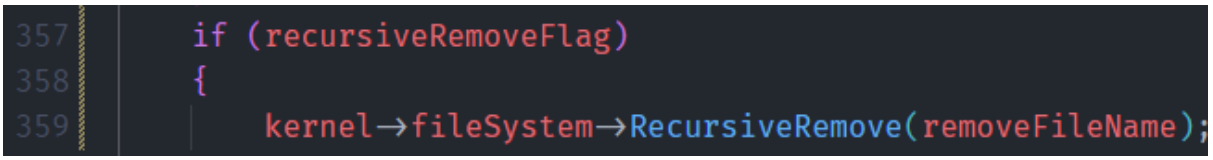
```
BonusII: Use 2696 blocks for header
```

BonusIII. Recursive operations on directories

main.cc



```
337     // if (removeFileName != NULL)
338     // {
339     //     kernel->fileSystem->Remove(removeFileName);
340     // }
```



```
357     if (recursiveRemoveFlag)
358     {
359         kernel->fileSystem->RecursiveRemove(removeFileName);
```

根據flag -rr呼叫RecursiveRemove

FileSystem::RecursiveRemove(char *name)

```
527 void FileSystem::RecursiveRemove(char *name)
528 {
529     Directory *directory = new Directory(NumDirEntries);
530     directory->FetchFrom(directoryFile);
531
532     // MP4
533     char *token = strtok(name, "/");
534     int sector;
535     char *prevToken = token;
536     OpenFile *currDirFile = directoryFile;
537
538     while (true) {
539         token = strtok(NULL, "/");
540         if (token == NULL) {
541             break;
542         }
543
544         // subDir header sector number
545         sector = directory->Find(prevToken);
546         You, 8 seconds ago • Uncommitted changes
547         DEBUG(dbgFile, prevToken << " at sector num " << sector);
548
549         OpenFile* subDirFile = new OpenFile(sector);
550         directory->FetchFrom(subDirFile);
551         currDirFile = subDirFile;
552
553         prevToken = token;
554     }
555
556     DEBUG(dbgFile, "file/Directory to delete: " << prevToken);
557     directory->RecursiveRemove(prevToken, freeMapFile, currDirFile);
558
559     delete directory;
560 }
```

529行：fetch root directory

538-554：不停向下尋找到我們要delete的file/directory的前一個directory

557行：呼叫Directory::RecursiveRemove去將目標file/directory清除，傳入目標file/directory的name(prevToken)、freeMapfile、currDirFile

Directory::RecursiveRemove

```
223     int index = FindIndex(name);
```

先看要刪除的file/directory是在他的parent directory table的哪個entry

```
225     if (table[index].isDir) {
226         subDirHdr->FetchFrom(table[index].sector);
227         subDirFile = new OpenFile(table[index].sector);
228         subDir->FetchFrom(subDirFile);
229 
230         for (int i = 0; i < subDir->tableSize; i++) {
231             if (subDir->table[i].inUse) {
232                 subDir->RecursiveRemove(subDir->table[i].name, freeMapFile, subDirFile);
233             }
234         }
235 
236         int sector = table[index].sector;
237 
238         subDirHdr->Deallocate(freeMap); // remove data blocks
239         freeMap->Clear(sector);       // remove header block
240         this->Remove(name);
241 
242         freeMap->WriteBack(freeMapFile);
243         WriteBack(dirFile);
244     }
```

如果目標是個directory,

226-228行:fetch他的header、DirFile、directory structure

230-234行:對他底下的檔案和資料夾做遞迴remove

238行:free掉他的data所使用的sector

239行:free掉他header使用的sector

240行:free掉目標在parent directory占用的table entry

242-243行:更新freeMapFile和dirFile

```
245     else {
246         int sector = table[index].sector;
247 
248         fileHdr = new FileHeader;
249         fileHdr->FetchFrom(sector);
250 
251         fileHdr->Deallocate(freeMap); // remove data blocks
252         freeMap->Clear(sector);       // remove header block
253         this->Remove(name);
254 
255         freeMap->WriteBack(freeMapFile);     // flush to disk
256         WriteBack(dirFile); // flush to disk
257     }
```

如果目標是個file

248-249行:fetch他的fileheader

251行：free掉他的data所使用的sector

252行：free掉他header使用的sector

253行：free掉目標在parent directory占用的table entry

255-256行：更新freeMapFile和dirFile

心得

經過這次作業，我們學到很多東西，像是怎麼增加fileMaxSize、怎麼維護bitmap和directory等等。這次有很多小地方要注意，尤其是main.cc的部分，常常因為main.cc有地方沒有改而卡了很久