MP4_Report_Team11

Team Member

- 資工大三 108062213 顏浩昀
- 電資大三 108061250 吳長錡

Contributions

Work	Member
Trace code	顏浩昀&吳長錡
Implement	顏浩昀&吳長錡
Debug	顏浩昀&吳長錡
report	顏浩昀&吳長錡

PartI Understanding NachOS file system

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

- NachOS的FileSystem使用Bitmap管理Free blocks
- NachOS的Physical Disk以Sector為最小存取單位
- Sector從0開始編號，使用一個table紀錄每個Sector是否被佔用，若為0表示沒有被佔用，1表示已被佔用
- 首先從Kernel.cc開始觀察

VOID KERNEL::INITIALIZE()

```
fileSystem = new FileSystem(formatFlag);
```

- 在Initialize函式中，可以看到file system的初始化
- formatFlag可以觀察Kernel()

KERNEL::KERNEL(INT ARGC, CHAR **ARGV)

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

- 可以發現當指令下達-f時，formatFlag = True
- 接著進入filesys.cc中觀察FileSystem(bool format)

FILESYSTEM::FILESYSTEM(BOOL FORMAT)

```
if (format) {
  PersistentBitmap *freeMap = new PersistentBitmap(NumSectors);
} else {
  // if we are not formatting the disk, just open the files representing
  // the bitmap and directory; these are left open while Nachos is running
  freeMapFile = new OpenFile(FreeMapSector);
  directoryFile = new OpenFile(DirectorySector);
}
```

- 可以發現format == True時，表示我們需要初始化磁碟來包含一個空的路徑，以及一個bitmap的磁區(幾乎但非全部的時區都標示為可以使用)
- When format == False => we are not formatting the disk, just open the files representing the bitmap and directory

- 接著我們開始觀察bitmap的部分
- NachOS主要利用bitmap來記錄free block space

BITMAP::BITMAP(INT NUMITEMS)

```
Bitmap::Bitmap(int numItems)
{
    int i;

    ASSERT(numItems > 0);

    numBits = numItems;
    numWords = divRoundUp(numBits, BitsInWord);
    map = new unsigned int[numWords];
    for (i = 0; i < numWords; i++)
    {
        map[i] = 0; // initialize map to keep Purify happy
    }
    for (i = 0; i < numBits; i++)
    {
        Clear(i);
    }
}
```

- Bitmap(int numItems)初始化bitmap，將使用情形記錄在map裡，0表示該位置的block還未被佔用，1表示已經被使用了。
- 接著觀察FindAndSet()是用來尋找free block space的function

INT BITMAP::FINDANDSET()

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

- 利用Test()搭配for迴圈，判斷每個bit是否已經被佔用，若有救回傳True，反之則為False
- 利用Mark()設置成使用
- 回頭觀察filesys.cc的FileSystem(bool format)

FILESYSTEM::FILESYSTEM(BOOL FORMAT)

```
FileSystem::FileSystem(bool format) {
  DEBUG(dbgFile, "Initializing the file system.");
  if (format) {
    PersistentBitmap *freeMap = new PersistentBitmap(NumSectors);
    Directory *directory = new Directory(NumDirEntries);
    FileHeader *mapHdr = new FileHeader;
    FileHeader *dirHdr = new FileHeader;

    DEBUG(dbgFile, "Formatting the file system.");

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

    // Second, allocate space for the data blocks containing the contents
    // of the directory and bitmap files.  There better be enough space!

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

    // Flush the bitmap and directory FileHeaders back to disk
    // We need to do this before we can "Open" the file, since open
    // reads the file header off of disk (and currently the disk has garbage
    // on it!).
```

- freeMap->Mark(FreeMapSector)是指bitmap header放在

disk 的第"FreeMapSector"個sector

- 因為它的FileHeader要固定放在FreeMapSector，所以FreeMapSector只能被freeMap的header使用，要先把FreeMapSector標示成已被使用，再來去allocate freeMap的資料時才不會用到FreeMapSector
- mapHdr->Allocate(freeMap, FreeMapFileSize)決定freeMap的資料要放在哪些sector
- freeMap是剛剛new而且已經改過的PersistentBitmap
- FreeMapFileSize是freeMap的大小，算法是總共有幾個sectors除以BitInBytes
- filesys.cc中有定義FreeMapSector跟DirectorySector的值，分別為0跟1
- 其他FileSystem函式：在FileSystem::Create/Remove/Print需要用到freeMap的時候，是先new一個PersistentBitmap，用freeMapFile初始化它，然後用這個一開始跟freeMapFile，之後FreeMapFile也是會一直開著用來找free space，然後用這個一開始跟freeMapFile一樣的新的freeMap去 Create/Remove/Print，成功且有更新freeMap的話再寫回到freeMapFile，否則就丟掉這次的動作
- Create的時候用freeMap->FindAndSet()找header要放的位置和更新freeMap，用fileHeader->Allocate(PersistentBitmap *freeMap, int fileSize)找資料放的位置和更新freeMap
- Remove用FileHeader::Deallocate(PersistentBitmap *freeMap, int fileSize)更新 freeMap把空間釋放出來，再用Clear釋放fileHeader空間

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

DISK.H

```
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
```

- 觀察disk.h可以發現，NachOS模擬的disk包含32個tracks
- 每個tracks包含32個sectors
- 每個sectors的大小為128bytes
- 接著觀察disk.c

DISK.C

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

- 可以發現到DiskSize = (MagicSize + (NumSectors * SectorSize))大約等於128KB
- MagicNumber的用處是為了減少我們不小心將有用的文件視為磁盤的可能性
- 如果文件已經存在，讀取前四個字節驗證是否包含預期的NachOS MagicNumber，如果檢查失敗則終止
- 這就是DiskSize前面要加MagicNumber的原因

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

- NachOS FS把directory當成一個file來管理，有header和dataSectors
- Data是Directory，用來記錄檔案名稱和檔案fileHeader的位置
- 和Part(1)的free block space相似，這裡我們直接從filesys.cc的code開始trace起

FILESYSTEM(BOOL FORMAT)

```
if (format) {
PersistentBitmap *freeMap = new PersistentBitmap(NumSectors); Directory *
```

- 當format = False，就用directoryFile = OpenFile(DirectorySector)到DirectorySector去讀header，把代表directory的file打開放在directoryFile，而且這個file會再NachOS跑的期間一直存在，就可以用這個file去找fileHeader
- 當format = True，可以看到FileSystem(bool format)new

一個 Directory 去紀錄 fileName和fileHeader location

- **接著觀察directory.cc** (http://xn--directory-f74q779dzv3gpyo.cc)，觀察Directory(int size)函式

DIRECTORY::DIRECTORY(INT SIZE)

```
Directory::Directory(int size) {
  table = new DirectoryEntry[size];

  // MP4 mod tag
  memset(table, 0, sizeof(DirectoryEntry) * size);
  // dummy operation to keep valgrind happy

  tableSize = size;
  for (int i = 0; i < tableSize; i++) {
    table[i].inUse = FALSE;
    table[i].isDir = FALSE;
  }
}
```

- 一開始directory的每個entry都還沒被用掉
- **回到filesys.cc** (http://xn--filesys-5t0ly16a.cc)

FILESYSTEM(BOOL FORMAT)

```
1  freeMap->Mark(DirectorySector);
2  ASSERT(dirHdr->Allocate(freeMap, DirectoryFileSize));
```

- 從Mark()裡面可以知道這行的意思是directory占用第"DirectorySector"個sector
- 因為它的FileHeader要固定放在DirectorySector，所以DirectorySector只能被directory的header使用，要先把DirectorySector標示成已被使用，再來去allocate directory的data時才不會用到 DirectorySector
- 用mapHdr->Allocate(freeMap, DirectoryFileSize)決定directory的data要放在哪些sector
- freeMap是剛剛new而且已經改過的PersistentBitmap，DirectoryFileSize是directory的大小，算法及數值定義都寫在filesys.cc裡

```
#define FreeMapSector 0
#define DirectorySector 1

#define NumDirEntries 10
#define DirectoryFileSize (sizeof(DirectoryEntry) * NumDirEntries)
```

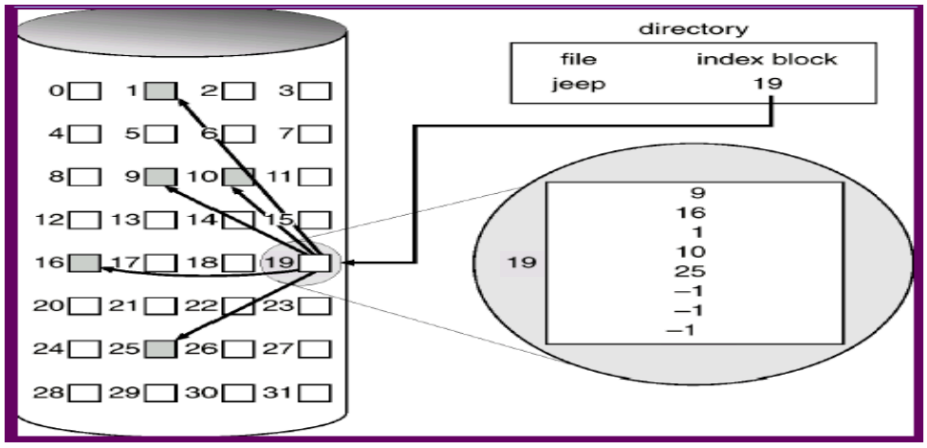
(4) Explain what information is stored in an inode, and use a figure to illustrate the disk allocation scheme of current implementation.

- inode是用來管理disk file header，所以我們要觀察filehrd.cc和filehrd.h
- 要知道inode存什麼資訊，我們需要觀察FileHeader Class的private data

CLASS FILEHEADER

```
class FileHeader {
    private:
        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
};
```

- 可以看到在inode中，存了numBytes: 這個file有幾個bytes
- numSectors: 這個file要用到幾個dataSector
- dataSectors[NumDirect]: 每個dataBlock在哪個sector



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

- 觀察filehrd.cc裡的Allocate()及filehrd.h中的定義

FILEHEADER::ALLOCATE()

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

FILEHRD.H

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

- 由Allocate中可發現，現在的file fileHeader只有一層=>32個sector size
- 從filehrd.h可以知道NumDirect要扣掉numBytes、numSectors = 30
- MaxFileSize = 30 * 128 < 4KB

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

System call:

USERPROG/KSYCALL.H

```
int SysCreate(char *filename, int initialSize) {
  return kernel->fileSystem->Create(filename, initialSize);
}

OpenFileId SysOpen(char *name) {
  OpenFileId openFileId = (OpenFileId)kernel->fileSystem->Open(name);
  if (openFileId > 0)
    return openFileId;
  else
    return -1;
}

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

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

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

int SysAdd(int op1, int op2) { return op1 + op2; }
// MP4 mod tag
case SC_Create:
  val = kernel->machine->ReadRegister(4);
  filename = &(kernel->machine->mainMemory[val]);
  initialSize = kernel->machine->ReadRegister(5);
  status = SysCreate(filename, initialSize);
  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;
```

- 根據MIPS convention從cpu register和memory拿取正確的argument
- 呼叫kernel的system call implementation(接到filesystem implementation)
- 結束後寫回return value
- 這裡以SC_Create為例，SC_Open, Read, Write, Close都一樣，只是呼叫在ksyscall.h中定義的不同funciton

FILESYS/ FILESYS.H (PARTIII 一併說明)

FILESYS/ FILESYS.C

```
// MP4
int FileSystem::Write(char *buffer, int size, OpenFileId id) {
  return opfile->Write(buffer, size);
}

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

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

THREADS/ MAIN.C

“-p”,“-cp”,“-l”

```
//    Filesystem-related flags:
//    -f forces the Nachos disk to be formatted
//    -cp copies a file from UNIX to Nachos
//    -p prints a Nachos file to stdout
//    -r removes a Nachos file from the file system
//    -l lists the contents of the Nachos directory
//    -D prints the contents of the entire file system
} else if (strcmp(argv[i], "-rr") == 0) {
  // MP4 mod tag
  ASSERT(i + 1 < argc);
  removeFileName = argv[i + 1];
  recursiveRemoveFlag = true;
  i++;
} else if (strcmp(argv[i], "-l") == 0) {
  // MP4 mod tag
  ASSERT(i + 1 < argc);
  listDirectoryName = argv[i + 1];
  dirListFlag = true;
  i++;
} else if (strcmp(argv[i], "-lr") == 0) {
  // MP4 mod tag
  // recursive list
  ASSERT(i + 1 < argc);
```

```
    listDirectoryName = argv[i + 1];
    dirListFlag = true;
    recursiveListFlag = true;
    i++;
} else if (strcmp(argv[i], "-mkdir") == 0) {
    // MP4 mod tag
    ASSERT(i + 1 < argc);
    createDirectoryName = argv[i + 1];
    mkdirFlag = true;
    i++;
// MP4 mod tag
char *createDirectoryName = NULL;
char *listDirectoryName = NULL;
bool mkdirFlag = false;
bool recursiveListFlag = false;
bool recursiveRemoveFlag = false;
```

- 要改的地方是RecursiveRemove跟RecursiveList的呼叫
- CreateDir的實作

```
//----------------------------------------------
// MP4 mod tag
// CreateDirectory
//      Create a new directory with "name"
//----------------------------------------------
static void CreateDirectory(char *name) {
    // MP4 Assignment
    kernel->fileSystem->CreateDir(name);
}
```

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

THREADS/ KERNEL.C

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

- "-f"Format的flag定義於此

Supprot Larger File Size

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

FILESYS/ FILEHDR.H

```
// MP4
#define numOfBytesLevel1 NumDirect *SectorSize
#define numOfBytesLevel2 NumDirect *NumDirect *SectorSize
#define numOfBytesLevel3 NumDirect *NumDirect *NumDirect *SectorSize
#define numOfBytesLevel4 NumDirect *NumDirect *NumDirect *NumDirect *SectorSize
// MP4
int GetNumOfBytes() { return numBytes; }

int GetNumOfSectors() { return numSectors; }
```

- 首先定義每個level最多可以容納的Bytes數

FILESYS/ FILEHDR.C/ ALLOCATE()

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

  if (fileSize > numOfBytesLevel4) {…
  } else if (fileSize > numOfBytesLevel3) {…
  } else if (fileSize > numOfBytesLevel2) {…
  } else if (fileSize > numOfBytesLevel1) {…
  } else {
    for (int i = 0; i < numSectors; i++) {
      dataSectors[i] = freeMap->FindAndSet();

      ASSERT(dataSectors[i] >= 0);

      char *clean_data = new char[SectorSize]();
      kernel->synchDisk->WriteSector(dataSectors[i], clean_data);
      delete clean_data;
    }
  }

  return TRUE;
}
```

- allocate FileHeader的dataSectors所對應的disk sector
- 視fileSize決定要給幾層的index table

```
if (fileSize > numOfBytesLevel4) {
  int i = 0;
  while (fileSize > 0) {
    dataSectors[i] = freeMap->FindAndSet();
    FileHeader *sub_hdr = new FileHeader();
    if (fileSize > numOfBytesLevel4) {
      sub_hdr->Allocate(freeMap, numOfBytesLevel4);
    } else {
      sub_hdr->Allocate(freeMap, fileSize);
    }
    fileSize -= numOfBytesLevel4;
    sub_hdr->WriteBack(dataSectors[i]);
    i++;
  }
```

- 每一層的架構都相同所以只截取一層來講解
  - if fileSize > numOfBytesLevel4: 需要分配多個index table先create一個dataSector[i](一個entry in FCB)存指向下一層table的sector number
  - call sub_hdr->Allocate()，如果所剩的bytes數小於這層直接call Allocate(fileSize)，若大於call

Allocate(numOfBytesLevel4)

- fileSize -= 剛剛分配掉的numOfBytesLevel4
- 將結果writeBack(寫回他的on-disk FCB)

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

    ASSERT(dataSectors[i] >= 0);

    char *clean_data = new char[SectorSize]();
    kernel->synchDisk->WriteSector(dataSectors[i], clean_data);
    delete clean_data;
  }
}
```

- else的部分跟原本NachOS single-indirect的分配一樣
- 但要將disk sector清乾淨後再分配
- 每次從bitmap獲得free disk sectors時，同時把該disk->WriteSector()寫入空的值

FILESYS/ FILEHDR.C/ DEALLOCATE()

```
void FileHeader::Deallocate(PersistentBitmap *freeMap) {
  if (numBytes > numOfBytesLevel1) {
    for (int i = 0; i < divRoundUp(numSectors, NumDirect); i++) {
      FileHeader *sub_hdr = new FileHeader();
      sub_hdr->FetchFrom(dataSectors[i]);
      sub_hdr->Deallocate(freeMap);
    }
  } else {
    for (int i = 0; i < numSectors; i++) {
      ASSERT(freeMap->Test((int)dataSectors[i]));
      freeMap->Clear((int)dataSectors[i]);
    }
  }
}
```

- 把freeMap中FileHeader的dataSectors所對應的bit清掉
- 方法與Allocate相同
- NumDirect = 30(一個index table最多30個dataSectors)
- for loop是只要往下跑幾次回圈才真正清完所有的dataSectors

FILESYS/ FILEHDR.C/ BYTESTOSECTOR()

```
int FileHeader::ByteToSector(int offset) {
  if (numBytes > numOfBytesLevel4) {
    FileHeader *sub_hdr = new FileHeader;
    int entry_number = divRoundDown(offset, numOfBytesLevel4);
    sub_hdr->FetchFrom(dataSectors[entry_number]);
    sub_hdr->ByteToSector(offset - numOfBytesLevel4 * entry_number);
  } else if (numBytes > numOfBytesLevel3) {
    FileHeader *sub_hdr = new FileHeader;
    int entry_number = divRoundDown(offset, numOfBytesLevel3);
    sub_hdr->FetchFrom(dataSectors[entry_number]);
    sub_hdr->ByteToSector(offset - numOfBytesLevel3 * entry_number);
  } else if (numBytes > numOfBytesLevel2) {
    FileHeader *sub_hdr = new FileHeader;
    int entry_number = divRoundDown(offset, numOfBytesLevel2);
    sub_hdr->FetchFrom(dataSectors[entry_number]);
    sub_hdr->ByteToSector(offset - numOfBytesLevel2 * entry_number);
  } else if (numBytes > numOfBytesLevel1) {
    FileHeader *sub_hdr = new FileHeader;
    int entry_number = divRoundDown(offset, numOfBytesLevel1);
    sub_hdr->FetchFrom(dataSectors[entry_number]);
    sub_hdr->ByteToSector(offset - numOfBytesLevel1 * entry_number);
  } else {
    return (dataSectors[offset / SectorSize]);
  }
}
```

- 給定virtual dataSector number map to physical sector number
- use divRoundDown because dataSectors index start from 0. For example, offset/num... = 1.7,find second entry, but second entry is dataSectors[1] => use roundDown

FILESYS/ FILEHDR.C/ PRINT()

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

  printf("FileHeader contents.  File size: %d.  File blocks:\n", numBytes);
  for (i = 0; i < numSectors; i++)
    printf("%d ", dataSectors[i]);
  printf("\nFile contents:\n");

  if (numBytes > numOfBytesLevel1) {
    for (int i = 0; i < numSectors / NumDirect; i++) {
      OpenFile *openFile = new OpenFile(dataSectors[i]);
      FileHeader *sub_hdr = openFile->GetHdr();
      sub_hdr->Print();
    }
  } else {
    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') // isprint(data[j])
          printf("%c", data[j]);
        else
          printf("\\%x", (unsigned char)data[j]);
      }
      printf("\n");
    }
  }
  delete[] data;
}
```

- Use the same DivRoundDown idea with bytesToSector()

PartIII Modify the file system code to support subdirectory

- 傳入的參數可能是很長的path(/t0/bb/f1)，所以需要逐步拆解pathname，直到最後一個token才是真正的filename，並且要一步步進入下一個directory

filesys/ filesys.c

CREATE(CHAR *NAME)

CREATEDIR(CHAR *NAME)

- 直接看CreateDir(), Create()只需改一行code而已

```
// For sub directory
bool FileSystem::CreateDir(char *name) {
  Directory *directory;
  PersistentBitmap *freeMap;
  FileHeader *hdr;
  int sector;
  bool success;

  DEBUG(dbgFile, "Creating dir " << name << " size " << DirectoryFileSize);

  directory = new Directory(NumDirEntries);
  directory->FetchFrom(directoryFile);

  OpenFile *file_temp = directoryFile;
  OpenFile *last_level_dir_file = file_temp;
  bool should_roll_back_to_last_level_dir = TRUE;
  char *token = strtok(name, "/");
  char *prev_token = token;
```

- 首先從root開始往下找，使用strtok(name, “/”)來拆解

```
while (token != NULL) {
  sector = directory->Find(token);
  if (sector == -1) {
    should_roll_back_to_last_level_dir = FALSE;
    break;
  }

  last_level_dir_file = file_temp;

  file_temp = new OpenFile(sector);
  directory->FetchFrom(file_temp);

  prev_token = token;
  token = strtok(NULL, "/");
}

if (token == NULL)
  token = prev_token; // if traverse to the end.
```

- 接著開始拆解token
- in while loop, if directory cannot find this token entry => haven’t create => break, token != NULL

```
if (should_roll_back_to_last_level_dir)
  directory->FetchFrom(last_level_dir_file);

if (directory->Find(token) != -1)
  success = FALSE; // file is already in directory
else {
  freeMap = new PersistentBitmap(freeMapFile, NumSectors);
  sector = freeMap->FindAndSet(); // find a sector to hold the file header
  if (sector == -1)
    success = FALSE; // no free block for file header
  else if (!directory->Add(token, sector, TRUE))
    success = FALSE; // no space in directory
  else {
    hdr = new FileHeader;
    if (!hdr->Allocate(freeMap, DirectoryFileSize))
      success = FALSE; // no space on disk for data
    else {
      success = TRUE;
      // everthing worked, flush all changes back to disk
      hdr->WriteBack(sector);
      directory->WriteBack(file_temp);
      freeMap->WriteBack(freeMapFile);
    }
    delete hdr;
  }
  delete freeMap;
}
delete directory;
return success;
```

- 確認directory有沒有，沒有的話看freeMap有沒有空間放此file的FCB
- 最後看disk上所有free sector夠不夠存file
- 以上都檢查完成且成功的話，使用filehdr->Allocate()逐一分配sector
- 再將file add to a directory entry時，使用：

```
else if (!directory->Add(token, sector, TRUE))
```

- 而在Create()一個file時：

```
else if (!directory->Add(token, sector, FALSE))
```

OPEN(CHAR *NAME)

```
OpenFile *FileSystem::Open(char *name) {
  Directory *directory = new Directory(NumDirEntries);
  directory->FetchFrom(directoryFile);
  OpenFile *openFile = NULL;
  int sector;

  OpenFile *file_temp = directoryFile;

  char *token = strtok(name, "/");
  char *prev_token = token;

  while (token != NULL) {
    if ((sector = directory->Find(token)) == -1 ||
        directory->IsDir(token) == FALSE) {
      break;
    }
    file_temp = new OpenFile(sector);
    directory->FetchFrom(file_temp);
    prev_token = token;
    token = strtok(NULL, "/");
  }

  if (token == NULL)
    name = prev_token;
  else
    name = token;

  sector = directory->Find(name);

  if (sector >= 0) {
    openFile = new OpenFile(sector);
  }

  delete directory;
  opfile = openFile;
  return openFile; // return NULL if not found
}
```

- 分解token方法與上面類似
- 把name assign成token
- 在這層directory去找token，找到就回傳OpenFile poiter
- 找不到return NULL

LIST(CHAR *DIRNAME, BOOL RECURSIVE)

```
void FileSystem::List(char *dirname, bool recursive) {
  Directory *directory = new Directory(NumDirEntries);
  directory->FetchFrom(directoryFile);

  OpenFile *dirFile = directoryFile;
  int sector;

  char *token = strtok(dirname, "/");
  char *prev_token = dirname;

  while (token != NULL) {
    if ((sector = directory->Find(token)) == -1) {
      printf("No such file or directory: %s\n", token);
      return;
    }

    dirFile = new OpenFile(sector);
    directory->FetchFrom(dirFile);
    prev_token = token;
    token = strtok(NULL, "/");
  }

  if (token == NULL)
    token = prev_token;

  if (strcmp(token, "/") == 0)
    sector = DirectorySector;

  if (recursive)
    directory->RecursiveList(0);
  else
    directory->List();
  delete directory;
}
```

- 使用同樣的方法拆解token，這裏token可能是NULL
- 因為可能會call list root(-l/)，所以如果token == NULL => token = “/”
- 接著看是否要recursive
- 如果要的話call directory->RecursiveList(0)
- 0是起始的depth，從這層directory開始，depth從0開始計算

REMOVE()

```
bool FileSystem::Remove(char *name, bool recursive) {
  Directory *directory;
  PersistentBitmap *freeMap;
  FileHeader *fileHdr;
  int sector, lastSector;

  directory = new Directory(NumDirEntries);
  directory->FetchFrom(directoryFile);
  OpenFile *file_temp = directoryFile;
  OpenFile *last_level_dir_file = file_temp;
  bool should_roll_back_to_last_level_dir = TRUE;

  char *token = strtok(name, "/");
  char *prev_token = token;
```

```
while (token != NULL) {
  lastSector = sector;
  sector = directory->Find(token);
  if (sector == -1) {
    delete directory;
    return FALSE;
  }
  last_level_dir_file = file_temp;
  file_temp = new OpenFile(sector);
  directory->FetchFrom(file_temp);

  prev_token = token;
  token = strtok(NULL, "/");
}

OpenFile *actual_dir_opfile = file_temp;

if (token == NULL) {
  token = prev_token;
  should_roll_back_to_last_level_dir = TRUE;
  directory->FetchFrom(last_level_dir_file);
}
fileHdr = new FileHeader;
fileHdr->FetchFrom(sector);

freeMap = new PersistentBitmap(freeMapFile, NumSectors);
if (recursive) {
  if (directory->IsDir(token)) {
    directory->FetchFrom(actual_dir_opfile);
    directory->List();

    directory->RecursiveRemove(token);
    directory->WriteBack(actual_dir_opfile);
    directory->List();

    fileHdr->Deallocate(freeMap);
    freeMap->Clear(sector);
    directory->FetchFrom(last_level_dir_file);
    directory->Remove(token);
    directory->WriteBack(last_level_dir_file);
  } else {
    fileHdr->Deallocate(freeMap);
    freeMap->Clear(sector);
    directory->Remove(token);
    directory->WriteBack(last_level_dir_file);
  }
} else {
  if (directory->IsDir(token)) {
    return FALSE;
  }
  fileHdr->Deallocate(freeMap);
  freeMap->Clear(sector);
  directory->Remove(token);
  directory->WriteBack(last_level_dir_file);
}
freeMap->WriteBack(freeMapFile);
```

```
    delete fileHdr;
    delete directory;
    delete freeMap;
    return TRUE;
}
```

- 跟List()很像
- 如果要RecursiveList，要先進入下一層directory，呼叫“他”的directory->RecursiveRemove()
- 等到return後，回到這層dir(他的上一層)，這樣才能從這層directory刪除他的entry

filesys/ filesys.h

```
// MP4
#define NumDirEntries 64
```

- 更改Number of directory Entries(10 -> 64)

```
    // MP4
    bool CreateDir(char *name);

    bool Remove(char *name, bool Recursive);

    void List(char *dirname, bool Recursive);

    int Write(char *buffer, int size, OpenFileId id);

    int Read(char *buffer, int size, OpenFileId id);

    int Close(OpenFileId id);

    bool CreateDirectory(char *createDirectoryName, int initialSize);

    OpenFile *FindSubDir(char *filepath);

    OpenFile *getFreeMapFile() { return freeMapFile; }

  private:
    OpenFile *freeMapFile;   // Bit map of free disk blocks,
                             // represented as a file
    OpenFile *directoryFile; // "Root" directory -- list of
                             // file names, represented as a file
    OpenFile *opfile;        // MP4
```

- 新增一些member function and private data
- CreateDir()建立子資料夾
- Remove() and List() 傳入bool recursive決定是否需要地回輸出
- Write(), Read(), Close()用來接到system call

- OpemFile opfile用來控制read/write

filesys/ directory.c

ADD()

```
// MP4
bool Directory::Add(char *name, int newSector, bool isDir) {
  if (FindIndex(name) != -1)
    return FALSE;

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

- 新增一個argument isDir，表示這個entry是[D] or [F]

LIST()

```
void Directory::List() {
  bool empty = TRUE;
  for (int i = 0; i < tableSize; i++) {
    if (table[i].inUse) {
      empty = FALSE;
      if (table[i].isDir)
        cout << "[D] " << table[i].name << endl;
      else
        cout << "[F] " << table[i].name << endl;
    }
  }
  if (empty)
    cout << "The directory is empty." << endl;
}
```

- 更改輸出格式

RECURSIVELIST()

```
void Directory::RecursiveList(int depth) {
  bool empty = TRUE;
  Directory *subdir = new Directory(NumDirEntries);
  OpenFile *subdir_openfile;

  DEBUG(dbgFile, "Directory::RecursiveList in depth: " << depth);

  for (int i = 0; i < tableSize; i++) {
    if (table[i].inUse) {
      cout << "\n";
      empty = FALSE;
      for (int k = 0; k < depth; k++)
        cout << "  ";

      if (table[i].isDir) {
        printf("[D] %s", table[i].name);
        subdir_openfile = new OpenFile(table[i].sector);
        subdir->FetchFrom(subdir_openfile);
        subdir->RecursiveList(depth + 1);
      } else {
        printf("[F] %s", table[i].name);
      }
    }
  }

  if (empty) {
    for (int k = 0; k < depth; k++)
      cout << " ";
    cout << "(Empty directory)";
  }
  depth--;
}
```

- 傳入depth，控制輸出格式
- 掃過整個directory，如果entry是directory，call subdir->RecursiveList
- 每次宣告一個Directory pointer need to FetchFrom(OpenFile*)，從disk把這個directory的file讀出來

REMOVE()

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

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

- 維持原本Remove

RecursiveRemove()

```
void Directory::RecursiveRemove(char *name) {
  bool empty = TRUE;
  cout << "Current directory: " << name << endl;
  for (int i = 0; i < tableSize; i++) {
    if (table[i].inUse) {
      empty = FALSE;
      if (table[i].isDir) {
        cout << "entry " << i << " is Dir. name " << table[i].name
             << ", sector " << table[i].sector << ""
             << "\n";
        Directory *next_dir = new Directory(NumDirEntries);
        OpenFile *next_dir_file = new OpenFile(table[i].sector);
        next_dir->FetchFrom(next_dir_file);

        PersistentBitmap *freeMap = new PersistentBitmap(
            kernel->fileSystem->getFreeMapFile(), NumSectors);
        FileHeader *next_dirfile_tobeRemove = new FileHeader;
        next_dirfile_tobeRemove->FetchFrom(table[i].sector);
        next_dirfile_tobeRemove->Deallocate(freeMap);
        freeMap->Clear(table[i].sector);
        next_dir->RecursiveRemove(table[i].name);
        next_dir->WriteBack(next_dir_file);
        this->Remove(table[i].name);
        freeMap->WriteBack(kernel->fileSystem->getFreeMapFile());

        delete next_dir_file;
        delete next_dir;
        delete next_dirfile_tobeRemove;
        delete freeMap;
      } else {
        cout << "entry " << i << " is a File. name " << table[i].name
             << ", sector " << table[i].sector << ""
             << "\n";

        PersistentBitmap *freeMap = new PersistentBitmap(
            kernel->fileSystem->getFreeMapFile(), NumSectors);
        FileHeader *fileHdr_of_file_tobeRemove = new FileHeader;
        fileHdr_of_file_tobeRemove->FetchFrom(table[i].sector);
        fileHdr_of_file_tobeRemove->Deallocate(freeMap);
        freeMap->Clear(table[i].sector);
        if (!Remove(table[i].name))
          cout << "Error in removing file: " << name << "\n";
        freeMap->WriteBack(kernel->fileSystem->getFreeMapFile());
      }
    }
  }

  if (empty)
    cout << "This directory is empty.\n";
}
```

- 和Remove很像，差別在於go through every entry時，if this entry is sub directory => call subdirectory own RecursiveRemove()
- After return from subdirectory->RecursiveRemove() =>

delete this subdirectory entry (use this directory's Remove(next_dir))

- get next_dir directory*
- Deallocate next_dir's FileHeader, bitmap, recursive call it's RecursiveRemove()
- After return need to writeBack to it's dirFile
- Back to directory(/t0), delete next directory(/t0/bb)'s entry

filesys/ directory.h

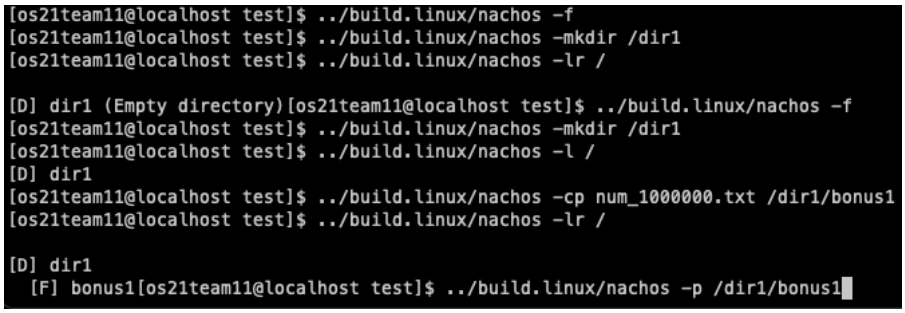
```
class DirectoryEntry {
public:
  bool inUse;                   // Is this directory entry in use?
  int sector;                   // Location on disk to find the
                                //   FileHeader for this file
  char name[FileNameMaxLen + 1]; // Text name for file, with +1 for
                                // the trailing '\0'
  // MP4
  bool isDir;
};
 // MP4
 bool Add(char *name, int newSector,
          bool isDir); // Add a file name into the directory

 void RecursiveList(int depth);

 void RecursiveRemove(char *name);

 void List(int depth, bool Recursive);

 bool IsDir(char *pathname) {
   int entry_index = this->FindIndex(pathname);
   return this->table[entry_index].isDir;
 }
 int getTableSize() { return tableSize; }
 void setTableSize(int size) { tableSize = size; }
 DirectoryEntry *getTable() { return table; }
```

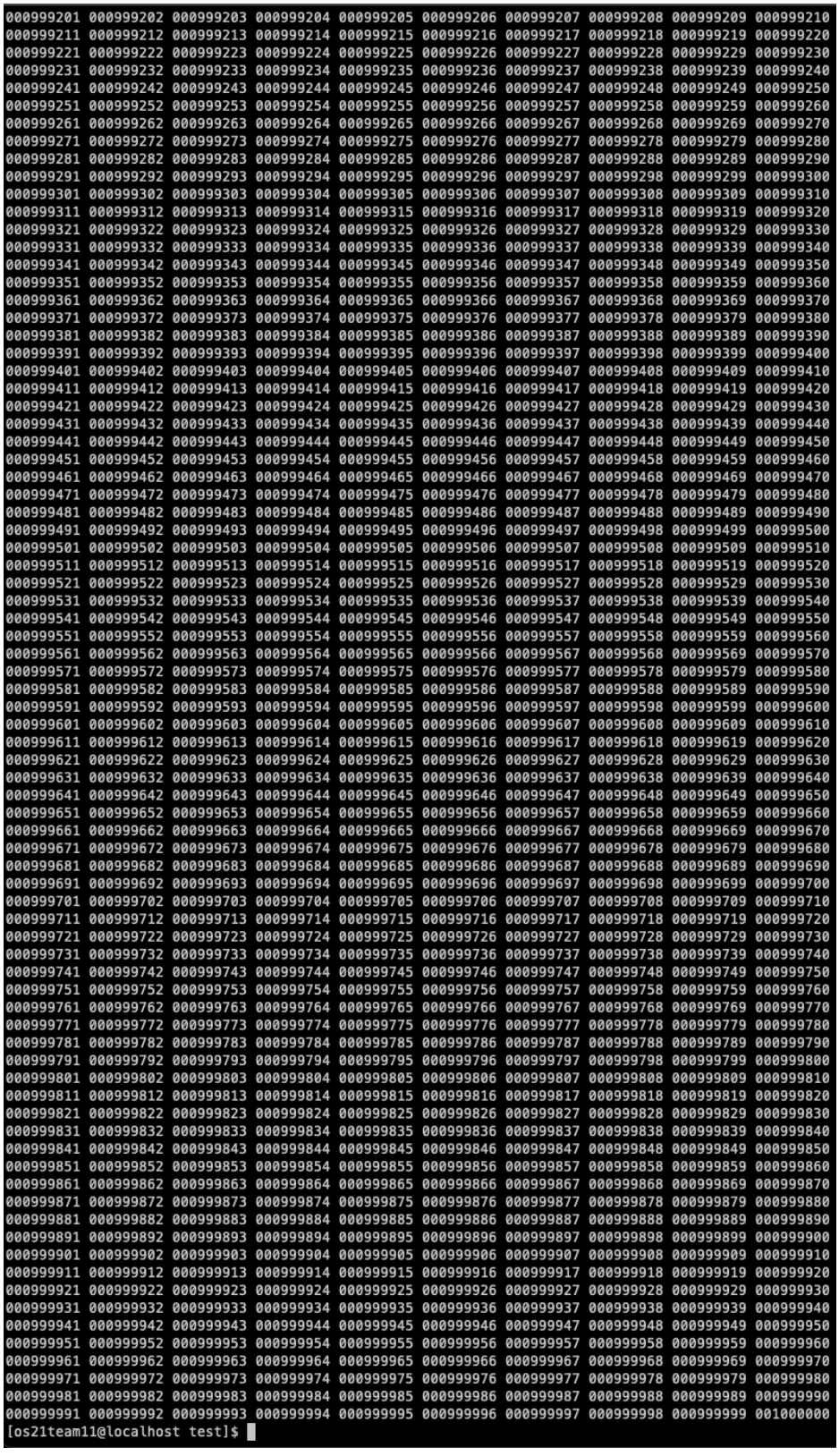
- Add a new member bool isDir in DirectoryEntry class, to see this entry is a file or subdirectory
- Addo some member functions

Bonus Assignment

Bounsl

- machine/ disk.h 更改Number of Tracks 32 -> 16000
- 一個檔案要64MB 除以128B => 需要512000個sectors
- SecotrsPerTrack只有32 => 512000 / 32 = 16000個Track





- RecursiveRemove的做法已在PartIII介紹過

```
[os21team11@localhost test]$ ../build.linux/nachos -f
[os21team11@localhost test]$ ../build.linux/nachos -mkdir /t0
[os21team11@localhost test]$ ../build.linux/nachos -mkdir /t1
[os21team11@localhost test]$ ../build.linux/nachos -mkdir /t2
[os21team11@localhost test]$ ../build.linux/nachos -lr /

[D] t0 (Empty directory)
[D] t1 (Empty directory)
[D] t2 (Empty directory)[os21team11@localhost test]$ ../build.linux/nachos -cp num_100.txt /t0/f1
[os21team11@localhost test]$ ../build.linux/nachos -mkdir /t0/aa
[os21team11@localhost test]$ ../build.linux/nachos -mkdir /t0/bb
[os21team11@localhost test]$ ../build.linux/nachos -mkdir /t0/cc
[os21team11@localhost test]$ ../build.linux/nachos -cp num_100.txt /t0/bb/f1
[os21team11@localhost test]$ ../build.linux/nachos -cp num_100.txt /t0/bb/f2
[os21team11@localhost test]$ ../build.linux/nachos -cp num_100.txt /t0/bb/f3
[os21team11@localhost test]$ ../build.linux/nachos -cp num_100.txt /t0/bb/f4
[os21team11@localhost test]$ ../build.linux/nachos -lr /

[D] t0
  [F] f1
  [D] aa  (Empty directory)
  [D] bb
    [F] f1
    [F] f2
    [F] f3
    [F] f4
  [D] cc  (Empty directory)
[D] t1 (Empty directory)
[D] t2 (Empty directory)[os21team11@localhost test]$ ../build.linux/nachos -rr /t0
[F] f1
[D] aa
[D] bb
[D] cc
Current directory: t0
entry 0 is a File. name f1, sector 530
entry 1 is Dir. name aa, sector 539
Current directory: aa
This directory is empty.
entry 2 is Dir. name bb, sector 542
Current directory: bb
entry 0 is a File. name f1, sector 548
entry 1 is a File. name f2, sector 557
entry 2 is a File. name f3, sector 566
entry 3 is a File. name f4, sector 575
entry 3 is Dir. name cc, sector 545
Current directory: cc
This directory is empty.
The directory is empty.
[os21team11@localhost test]$ ../build.linux/nachos -lr /

[D] t1 (Empty directory)
[D] t2 (Empty directory)[os21team11@localhost test]$
```