操作系统实验报告 3

林祥，PB16020923

1. 主要步骤及核心代码解释说明

（1） FileHeader::Allocate 的实现

a) 这个函数是为文件头对应的文件分配存储的扇区。

b) 首先是计算需要的扇区数，如果小于空闲扇区，则返回错误。

c) 然后对直接块分配扇区

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

d) **（洞 1）**为间接块分配扇区。doneSec 是前面直接块已经分配的扇区数。为每个 indirectSector 及其指向的 NumInDirectIndex 个数据块分配扇区，内外循环始终要判断是否已经到达该文件所需的扇区数 numSectors。

```
int doneSec = i;
for (int j = 0; j < numIndirectSectors && doneSec < numSectors; j++){
    int sectors[NumInDirectIndex];
    indirectSectors[j] = freeMap->FindAndSet();
    for(int k = 0; k < NumInDirectIndex && doneSec < numSectors; k++, doneSec++){
        sectors[k] = freeMap->FindAndSet();
    }
    kernel->synchDisk->WriteSector(indirectSectors[j], (char *)sectors);
}
```

（2） FileHeader::Deallocate 的实现

a) 这个函数是释放文件头对应的文件的扇区。

b) 类似地，先释放直接块，**这里只是标记为空闲，实际数据并未删除。**

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

c) **（洞 2）**再释放间接块。释放每个 indirectSector 指向的 NumInDirectIndex 个数据块的扇区，再释放 indirectSector 自身所在的扇区。

```
int doneSec = i;
for (int j = 0; j < numIndirectSectors && doneSec < numSectors; j++){
    int sectors[NumInDirectIndex];
    kernel->synchDisk->ReadSector(indirectSectors[j], (char *)sectors);
```

```
    for(int k = 0; k < NumInDirectIndex && doneSec < numSectors; k++, doneSec++){
        freeMap->Clear(sectors[k]);
    }
    freeMap->Clear(indirectSectors[j]);
}
```

（3） FileHeader::ByteToSector 的实现

a) 这个函数通过文件内部的偏移字节数计算出对应的扇区号

b) **（洞 3）**要分两种情况：

i. 一是偏移量还在直接块能存储的范围内，那么可以直接查询 dataSectors 数组得到对应的扇区号，在 dataSectors 数组中的下标就是 offset / SectorSize（向下取整）。

ii. 二是偏移量在间接块存储的范围，那么偏移量减去直接块可存储的大小，差值除去每个间接块可存储的文件大小，可以算出 indirectIndex，表示在哪个间接存储的的访问范围内，再算出是该间接块内部的第几个直接块，具体代码见下面，不好表述。

```
int sectors[NumInDirectIndex]; //direct sectors under inderect sector
int indirectIndex = (offset - DirectSize) / InDirectSectorSize;
int index;
if(offset < DirectSize){
    index = offset / SectorSize;
    return dataSectors[index];
}
else{
    index = (offset - DirectSize - InDirectSectorSize * indirectIndex) / SectorSize;
    kernel->synchDisk->ReadSector(indirectSectors[indirectIndex], (char *)sectors);
    return sectors[index];
}
```

（4） FileHeader::expandFile 的实现

a) 这个函数是为了解决 WriteAt 的时候出现文件大小变大的情况。

b) 先算出需要的总扇区数，以及需要的间接扇区数，如果空闲扇区不够，返回错误。

c) 然后在已有扇区的基础上，继续分配直接块，如果已有扇区数以及超过的直接块的存储范围，则这个循环不会执行。

```
for (i = numSectors; i < NumDirect && i < numSec;)
    if (dataSectors[i] == -1){
        dataSectors[i] = freeMap->FindAndSet();
        i++;
    }
```

d) **（洞 4）**在间接块部分继续分配，这时有两种情况：

i. 之前的间接块尾部没有填满，需要先读出最后一个间接块，找到第一个空闲的数据块，从这里开始分配新扇区，直到该间接块的每个数据块全部有对应扇区。然后下一次循环就变成下面的情况。

ii. 之前的间接块尾部填满了，那么这个间接块就是一个新的间接块，直接按 Allocate 函数的方法来实现。

```
for (int j = 0; j < indirectsNeed && doneSec < numSec; j++){
    int sectors[NumInDirectIndex]; //index table
    clearIndexTable(sectors);
    if (indirectSectors[j] == -1){//a new empty indirectSector
        indirectSectors[j] = freeMap->FindAndSet();
        for(int k = 0; k < NumInDirectIndex && doneSec < numSec; k++, doneSec++){
            sectors[k] = freeMap->FindAndSet();
        }
        kernel->synchDisk->WriteSector(indirectSectors[j], (char *)sectors);
    }
    else{  //a full or no empty indirectSector
        kernel->synchDisk->ReadSector(indirectSectors[j], (char *)sectors);
        int k = 0;
        for(; k < NumInDirectIndex; k++){
            if(sectors[k] == -1) break;
        }
        for(; k < NumInDirectIndex && doneSec < numSec; k++, doneSec++){
            sectors[k] = freeMap->FindAndSet();
        }
        kernel->synchDisk->WriteSector(indirectSectors[j], (char *)sectors);
    }
}
```

（5） OpenFile::WriteAt 的实现

a) 这个函数实现文件的在任意位置读写，Write 函数就是调用了该函数。

b) 首先要判断写入后文件大小是否变大

c) **（洞 5）**变大则调用 expandFile 函数扩大文件，这里需要从 kernel 中获得文件系统和它的 freeMapFile 对象，并且在文件大小拓展后，更新后的 freeMap 要写回 freeMapFile 文件。

```
int numSec=(position+numBytes+SectorSize-1)/SectorSize;
OpenFile* freeMapFile = new OpenFile(1);
freeMapFile=kernel->fileSystem->getFreeMapFile();
PersistentBitmap* freeMap = new PersistentBitmap(freeMapFile,NumSectors);
```

```
freeMap->FetchFrom(freeMapFile);
if ((position + numBytes) > fileLength){
    hdr->expandFile(numSec,freeMap);
    hdr->WriteBack(headerSector);
    freeMap->WriteBack(freeMapFile);
}
```

d) 计算要写入区间对应的扇区，判断头尾部分是否有不完整的扇区，如果有，需要将相应的部分读取出来到 buf

e) 使用 bcopy 函数将要写入的数据复制到 buf 相应位置，然后使用 kernel->synchDisk->WriteSector 将 buf 写回扇区。

（6） FileSystem::Open 的实现

a) 这个函数实现在 nachos 上的虚拟文件系统上打开一个文件。

b) 首先将 name 包含的各级目录名解析出来

c) **（洞 9）**逐级目录跳转置文件所在目录。currentDirectory 是当前目录，不断使用 Find 函数寻找子目录及其文件头所在扇区，然后新建一个 OpenFile 对象表示子目录对应的文件，从中读取出目录结构赋值给 currentDirectory，如此循环，直到到达最深层次的目录。

```
for (i = 0; i < (int)folderStrVec.size() - 1; i++) {
    int childdirfilehdrsector = currentDirectory->Find((char
                                *)folderStrVec[i].c_str(), false);
    if(childdirfilehdrsector == -1) return NULL;
    currentDirectoryFile = new OpenFile(childdirfilehdrsector);
 //directly new OpenFile, rather than use FileSystem::Open --------lx
    currentDirectory->FetchFrom(currentDirectoryFile);
}
```

d) **（洞 6）**在最深层次的目录下用 Find 方法寻找目标文件，如果找到，为其创建一个 OpenFile 对象，并分配文件描述符，用 addFile 加到文件系统的打开文件列表。

```
if(sector != -1){
    for(i = 3; i < MaxOpenFile; i++){
        if(getFile(i) == NULL) break;
    }
    if(i != MaxOpenFile) {
        openFile = new OpenFile(sector);
        openFile->setId(i);
        addFile(i,openFile);
    }else return NULL;
}
```

（7） FileSystem::Create 的实现

a) 这个函数实现在文件系统中创建一个文件。

b) 首先将 name 包含的各级目录名解析出来

c) **（洞 7）**逐级目录跳转置文件所在目录。currentDirectory 是当前目录，不断使用 Find 函数寻找子目录及其文件头所在扇区，然后新建一个 OpenFile 对象表示子目录对应的文件，从中读取出目录结构赋值给 currentDirectory，如此循环，直到到达最深层次的目录。代码与上面 FileSystem::Open**（洞 9）**类似

d) 判断当前目录下是否已经存在要创建的文件（同名）。

e) 不存在则用 freeMap->FindAndSet()为新文件的文件头分配扇区 sector，当然，要判断空闲扇区是否足够。

f) 把新文件的 name 和其文件头对应的 sector 作为一个目录项加入目录中，当然，要判断目录是否已满。

g) 使用新文件的文件头对象的 Allocate 方法为文件分配扇区。

h) 将相关对象（freeMap、hdr、currentDirectory）写回对应文件，并释放对象。

（8） FileSystem::CreateFolder 的实现**（洞 8）**

a) 这个函数实现在文件系统中创建一个目录。

b) 这个函数的实现类似 FileSystem::Create，仅有两处不同：

i. 一个是文件的初始大小，Allocate 的时候大小为 DirectoryFileSize

ii. 扇区分配完成后，需要初始化目录结构并写回。

```
Directory *createdDirectory = new Directory(NumDirEntries);
OpenFile *createdDirectoryFile = new OpenFile(sector);
createdDirectory->WriteBack(createdDirectoryFile);
delete createdDirectory;
delete createdDirectoryFile;
```

（9） FileSystem::Remove 的实现

a) 这个函数实现在文件系统中删除一个文件。

b) 首先将 name 包含的各级目录名解析出来

c) **（洞 10）**逐级目录跳转置文件所在目录。currentDirectory 是当前目录，不断使用 Find 函数寻找子目录及其文件头所在扇区，然后新建一个 OpenFile 对象表示子目录对应的文件，从中读取出目录结构赋值给 currentDirectory，如此循环，直到到达最深层次的目录。代码与上面 FileSystem::Open**（洞 9）**类似

d) 判断文件是否存在

e) 存在则读取文件头，调用 Deallocate 方法释放文件的扇区。

f) freeMap 标记文件头所在扇区为空闲，currentDirectory 标记对应目录项为空闲。

g) 写回 freeMap、currentDirectory 对应的文件对象。

（10） FileSystem::Recover 的实现

a) 这个函数实现在 nachos 文件系统中恢复一个文件到 Linux 中。

b) 首先将 name 包含的各级目录名解析出来

c) **（洞 11）**逐级目录跳转置文件所在目录。currentDirectory 是当前目录，不断使用 Find 函数寻找子目录及其文件头所在扇区，然后新建一个 OpenFile 对象表示子目录对应的文件，从中读取出目录结构赋值给 currentDirectory，如此循环，直到到达最深层次的目录。代码与上面 FileSystem::Open**（洞 9）**类似。

d) 在当前目录下 Find 要恢复的文件名，justcomparename 选择 true，表示只要文件名相同就算找到，因为被删除的文件对应 inUse=false。

e) 如果找到要恢复的文件目录项，尝试读取对应扇区（文件头）。

```
recoverfilename = (char *) folderStrVec[i].c_str();
recoverfilehdrsector = currentDirectory->Find(recoverfilename, true);
//just compare name, ignore "inUse"--------lx
if (recoverfilehdrsector == -1){
   delete currentDirectory;
   if (currentDirectoryFile != directoryFile)
      delete currentDirectoryFile;
   return FALSE; // file not found
}
recoverfile = new OpenFile(recoverfilehdrsector);
```

f) 判断文件长度是否小于 0（显然不能恢复），如果大于 0，读取对应长度的内容到 buf 中（不能保证数据和原来一样了），写回 Linux 中指定的文件中。

```
char *buffer = new char[recoverfile->Length()];
recoverfile->ReadAt(buffer,recoverfile->Length(),0);
FILE *out = fopen(dstName, "w");
fwrite(buffer,sizeof(char),recoverfile->Length(),out);
fclose(out);
```

2. 实验运行结果截图及分析说明

a) 阶段 1 测试

测试方法解释：

（1）toTestFileSys.sh，这是一个 nachos 的 shell，它先格式化 nachos 磁盘，然后从 Linux 中复制 prince.txt 和 testFileSys 文件到 nachos 磁盘中。

testFileSys 是一个 nachos 可执行文件，它将打开 prince.txt，创建并打开 hello 文件，从 prince.txt 读取 512 字节，重复十次写入 hello，然后关闭两个文件。打印所有文件的信息。

（2）运行./toTestFileSys.sh > testOutcome.txt; cat testOutcome.txt 将得到 shell 文件的执行情况（在 test/testOutcome.txt 文件中并且显示）。

测试结果分析：

测试结果与标准结果相同，正确显示了各个文件的内容和大小，其中 hello 文件的大小为 5120，与预期一致。

结果过长，此处不放出截图。

b) 阶段 2 测试

测试方法解释：

（1）toTestDirectoryAndRecovery.sh，这是一个 nachos 的 shell，它先格式化 nachos 磁盘，然后从 Linux 中复制 testDirectory 文件到 nachos 磁盘中。

testDirectory 是一个 nachos 可执行文件，它创建 folder1 目录，创建 folder1/folder2，folder2/folder3，创建 folder1/folder2/file，写入十次"we write contents to folder1/folder2/file\n"文本，关闭文件。

删除文件 file。

恢复文件 file 到 Recovery.txt。

（2）运行./ toTestDirectoryAndRecovery.sh; cat Recovery.txt;

```
lin@ubuntu:~/nachos/NachOS-4.0/code/test$ ./toTestDirectoryAndRecovery.sh
create folder folder2 failed
Machine halting!

Ticks: total 864562, idle 859592, system 4680, user 290
Disk I/O: reads 85, writes 62
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
n, I went to sleep on the sand, a thousand miles from any human habitation. I was more isolaMachine halting!

Ticks: total 32520, idle 32080, system 440, user 0
Disk I/O: reads 15, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
Machine halting!

Ticks: total 46020, idle 45560, system 460, user 0
Disk I/O: reads 13, writes 3
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
Machine halting!

Ticks: total 32520, idle 32080, system 440, user 0
Disk I/O: reads 15, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
lin@ubuntu:~/nachos/NachOS-4.0/code/test$
```

```
lin@ubuntu:~/nachos/NachOS-4.0/code/test$ cat recovery.txt
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
we write contents to folder1/folder2/file
n, I went to sleep on the sand, a thousand miles from any human habitation. I was more isolalin@ubuntu:~/nachos/NachOS-4.0/code/test$
```

测试结果分析：

测试结果与标准结果相同，创建 folder2/folder3 会失败，因为没有写全路径名。Recovery.txt 的最后一行是阶段 1 测试的残留信息。

3．实验过程中遇到的问题及解决方法

（1）遇到问题：

测试结果为：

```
lin@ubuntu:~/nachos/NachOS-4.0/code/test$ ./toTestFileSys.sh > testOutcome.txt; cat testOutcome.txt
Assertion failed: line 139 file ../machine/disk.cc
./toTestFileSys.sh: line 3:  8701 Aborted                 (core dumped) $nachos -f
Assertion failed: line 119 file ../machine/disk.cc
./toTestFileSys.sh: line 4:  8703 Aborted                 (core dumped) $nachos -cp prince.txt prince.txt
Assertion failed: line 119 file ../machine/disk.cc
./toTestFileSys.sh: line 5:  8705 Aborted                 (core dumped) $nachos -cp testFileSys testFileSys
Assertion failed: line 119 file ../machine/disk.cc
./toTestFileSys.sh: line 6:  8714 Aborted                 (core dumped) $nachos -x testFileSys
lin@ubuntu:~/nachos/NachOS-4.0/code/test$
```

分析问题：

提示全部是 core dumped，发生在 disk.cc 的 139 行。

到 disk.cc 查看：

```
133 void
134 Disk::WriteRequest(int sectorNumber, char* data)
135 {
136     int ticks = ComputeLatency(sectorNumber, TRUE);
137     printf("--------------------------secnum: %d--------------------------\n",sectorNumber);
138     ASSERT(!active);
139     ASSERT((sectorNumber >= 0) && (sectorNumber < NumSectors));
140
```

ASSERT 发生的条件是：对磁盘发出读请求的时候 sectorNumber 小于 0 或者大于最大扇区数，sectorNumber 是从上级函数传进来的，但无法确定计算出错的位置。

增加一个输出显示 sectorNumber，到 gdb 下单步调试（使用命令 n）。

```
(gdb) n
96              freeMap->Mark(DirectorySector);
(gdb) n
101             ASSERT(mapHdr->Allocate(freeMap, FreeMapFileSize));
(gdb) n
102             ASSERT(dirHdr->Allocate(freeMap, DirectoryFileSize));
(gdb) n
108             DEBUG(dbgFile, "Writing headers back to disk.");
(gdb) n
109             mapHdr->WriteBack(FreeMapSector);
(gdb) n
--------------------------secnum: 0-------------------------
110             dirHdr->WriteBack(DirectorySector);
(gdb) n
--------------------------secnum: 1-------------------------
116             freeMapFile = new OpenFile(FreeMapSector);
(gdb) n
117             directoryFile = new OpenFile(DirectorySector);
(gdb) n
125             DEBUG(dbgFile, "Writing bitmap and directory back to disk.");
(gdb) n
126             freeMap->WriteBack(freeMapFile); // flush changes to disk
(gdb) n
--------------------------secnum: 0-------------------------
--------------------------secnum: 0-------------------------
--------------------------secnum: -1073746072-------------------------
Assertion failed: line 139 file ../machine/disk.cc

Program received signal SIGABRT, Aborted.
0xb7fdccf9 in ?? ()
(gdb)
```

问题出现在 Writeback，但这里并没有计算 sectorNumber
进一步地，用 gdb 下用命令 s 单步进入函数

```
(gdb) n
212             int sector = hdr->ByteToSector(i*SectorSize);
(gdb) n
214                                             &buf[(i - firstSector) * SectorSize]);
(gdb) n
--------------------------secnum: 0-------------------------
210         for (i = firstSector; i <= lastSector; i++)
(gdb) n
212             int sector = hdr->ByteToSector(i*SectorSize);
(gdb) n
214                                             &buf[(i - firstSector) * SectorSize]);
(gdb) n
--------------------------secnum: 0-------------------------
210         for (i = firstSector; i <= lastSector; i++)
(gdb) n
212             int sector = hdr->ByteToSector(i*SectorSize);
(gdb) n
214                                             &buf[(i - firstSector) * SectorSize]);
(gdb) n
--------------------------secnum: -1073746072-------------------------
Assertion failed: line 139 file ../machine/disk.cc

Program received signal SIGABRT, Aborted.
0xb7fdccf9 in ?? ()
(gdb)
```

经过多次单步和进入，确定出错位置在 ByteToSector 函数

```
if(offset < DirectSize){
    index = offset / SectorSize;
    return dataSectors[index];
}
else{
    index = (offset - DirectSize - InDirectSectorSize * indirectIndex) / SectorSize;
    kernel->synchDisk->ReadSector(indirectSectors[indirectIndex], (char *)sectors);

    //printf("-------------------------index: %d sectors[index]: %d--------------------
    ASSERT(sectors[index] >= 0 && sectors[index] < NumSectors);//-----lx
    return sectors[index];
}
```

经查，是在 offset < DirectSize 时忘记返回，而将 return sectors[index]放在分支判断之外，改正为上图即可。

（2）遇到问题：

测试结果为：

```
Machine halting!

Ticks: total 409020, idle 407800, system 1220, user 0
Disk I/O: reads 17, writes 24
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
file did not open
read failed
file did not open
write failed!
file did not open
write failed!
file did not open
write failed!
file did not open
write failed!
file did not open
write failed!
file did not open
write failed!
file did not open
write failed!
file did not open
write failed!
file did not open
write failed!
file did not open
write failed!
file did not open
write failed!
close file failed!
close file failed!
```

分析问题：

仔细查看各文件，file did not open 提示是出现在 filesys.cc 的 FileSystem::Read 函数中。然而 ksyscall.h 的 SysOpen 中的错误提示

```
if(file==NULL) {
  printf("open %s error\n",filename);
  return -1;
}
```

并没有出现，因此文件打开是成功的，并且分配了 fid。

后来仔细思考，问题在于 FileSystem::Open 中，没有为新的 openFile 对象设置 fid 值，

```
if(sector != -1){
    for(i = 3; i < MaxOpenFile; i++){
        if(getFile(i) == NULL) break;
    }
    if(i != MaxOpenFile) {
        openFile = new OpenFile(sector);
        openFile->setId(i);
        addFile(i,openFile);
    }
}
```

（3）遇到问题：

测试结果出现 sectorNumber 比总扇区数还多的情况

```
---------------------------secnum: 4---------------------------
---------------------------secnum: 5---------------------------
---------------------------secnum: 6---------------------------
---------------------------secnum: 7---------------------------
---------------------------secnum: 8---------------------------
---------------------------secnum: 9---------------------------
---------------------------secnum: 0---------------------------
---------------------------secnum: 17--------------------------
---------------------------secnum: 0---------------------------
---------------------------secnum: 247540----------------------
lin@ubuntu:~/nachos/NachOS-4.0/code/test$
```

解决问题：

expandFile 的参数传错了，直接传入了 position + numBytes，应该改为(position + numBytes + SectorSize - 1) / SectorSize;

```
if ((position + numBytes) > fileLength){
    PersistentBitmap *freeMap = new PersistentBitmap(kernel->fileSystem->getFreeMapFile(), NumSectors);
    //"NumSectors" is the total sector of disk, diff from "numSectors" ---lx
    int newnumSec = (position + numBytes + SectorSize - 1) / SectorSize;
    if(hdr->expandFile(newnumSec, freeMap) == -1) return 0;
    hdr->WriteBack(headerSector);
    freeMap->WriteBack(kernel->fileSystem->getFreeMapFile());

}
```