Lab3 Nachos 文件系统

唐凯成，PB16001695

一、实验目的

1、运用所学文件系统知识，将 Nachos 的文件系统修改为模拟磁盘的工作模式，并将对应的系统操作移植到虚拟文件系统中。
2、理解掌握有关文件系统多级索引的相关知识，实现 Nachos 内的多级索引。
3、根据所学有关多级目录的知识，拓展原 Nachos 中的单级目录结构。
4、学习有关文件恢复的相关知识，编写对应恢复文件的接口函数。

二、实验过程

1、添加多级索引

（1）拓展 Nachos 的单级索引结构：

对于 Nachos 原有文件系统的单级索引，其在 code/filesys/filehdr.h 中申明了如右图的单级索引结构，每个文件都会有一个对应的文件头指针，存储对应文件里的所有数据存放的扇区位置。其中的 numBytes 代表文件的大小，numSectors 代表文件总共占据的扇区数，dataSectors 整型数组用来存储每个扇区对应在 bitmap 中的位置，一个位置存储 128B 信息，以此来构成单级索引。

```
private:
  int numBytes;
  int numSectors;
  int dataSectors[NumDirect];
```

为了拓展文件的最大大小（原文件系统内最大文件只能为：128B*32=4MB），加入二级索引，拓展后的文件头类如右图，主要加入了 4 个 numIndirectSectors，即对应的二级索引。每个二级索引下都存有 32 个一级索引，因为二级索引所对应的磁盘内的一个扇区大小也是 128B，一级索引对应的每个数字均为 4B，故一共能存储 32 个一级索引的 bitmap 位置。经过拓展后的 Nachos 最大文件大小变为；（25+4*32）*128B=19MB。

```
private:
  int numBytes;
  int numSectors;
  int dataSectors[NumDirect];

  /* ++++ LAB3 ++++ */
  int numIndirectSectors;
  int IndirectSectors[MaxIndirect];
  /* +++++++++++++++ */
```

（2）多级索引算法的基本实现：

1）文件扇区的分配和清除

对于从 unix 文件系统中复制过来的文件，nachos 虚拟文件系统中需要在复制前先给文件分配所需的扇区和对应的一级二级索引信息，这些信息均保存在文件的头指针文件中。具体的实现由 code/filesys/filehdr.cc 中的 Allocate 函数和 Deallocate 函数完成。由于具体源码已在实验包中发送，故这里仅介绍大致的算法思路。在原来的单级索引扇区的分配中，只需要调用 code/filesys/bitmap.h 里的 FindAndSet()即可，该函数用以找到位图中还未被别的文件占据的扇区，再将其对应的扇区号保存在 dataSectors 数组中即可。

但对于拓展后的二级索引，则要需加入间接索引块所存放的一级索引的写入，在实际实现时，需先计算该文件需要扇区数，若扇区数较多（在本次设计中直接索引为 25 个），大于直接索引的个数，即需要分配间接索引。每次分配间接索引的步骤为：首先在 bitmap 里给二级索引分配一个扇区，然后继续为该文件分配普通的数据扇区（小与等于 32 个），分配完成后，直接将普通扇区对应的 bitmap 的位置数据写入之前分配的特殊的二级索引对应的扇区中，此时使用 WriteSector 函数完成对数据的写回。

Deallocate 函数的实现恰好与 Allocate 函数相反，只需使用 bitmap 中的 Clear 函数，将之前分配的扇区的位图清除即可，对于二级索引，需要先将其内存储的一级缩影从对应扇区读出，使用 ReadSector 实现，再将对应一级和二级索引的位图均 Clear 掉即可。

2）实现查找文件内数据内容对应的扇区

为了后续查找数据的快捷，此处先实现了一个由文件中数据位置，返回对应文件中扇区号的函数 ByteToSector，仍在 code/filesys/filehdr.h 中。该函数的实现主要分为两部分：

一是直接索引部分，直接查看要查找的数据位置大小是否大于 25*128B，若不大于该值，只需返回对应 dataSectors[offset/SectorSize]内的值即可。

二是间接索引部分，若要查找的位置大于 25*128B，则需再继续去对应的间接索引里进行查找，此时先确定对应间接索引的数字，每个间接索引最多存储 32*128B 的内容。确认间接索引的位置以后再从对应扇区中读取对应的一级索引，从而返回对应实际扇区号。

3）写入文件时文件的拓展

对于原 Nachos 的文件系统来说，当写入文件大于一级索引所能存储的总大小时，即无法继续写入文件。而新增加了多级索引的 Nachos 可以处理这一问题，当文件所需写入内容的大小大于当前文件大小时，需为该文件分配新的扇区，主要还是分三种情况：

一是之前文件拥有的扇区还未越过直接索引部分，此时优先为该文件分配直接索引。

二是直接索引已经分配完，需继续分配间接索引，此时类似 Allocate 函数的操作，先分配二级再分配一级索引，最后将一级索引的临时数组 sectors 写入对应二级索引的扇区中。

三是之前的二级索引内的一级索引还有剩余，则应从有剩余的位置继续分配。此时先读取对应二级索引内的一级索引，然后查找到对应第一个索引对应扇区编号为-1 的位置，即代表未分配扇区，注意这里也提醒我们在之前为二级索引分配一级索引前，应该先将那个临时的 sectors 数组内每个格子内的编号先写为-1，代表未分配，此处也可直接用 clearIndexTable 函数完成对 sectors 数组的初始化。（此处存在一个写回和文件大小的问题，还有读取问题）

另外在完成上述分配后，需要将对应文件的文件头和此时更新后的 Bitmap 写回一次，再继续进行后续的写入操作。

4）实现打开文件的 Open 函数

在已经完成好整个多级索引结构后，最后实现打开一个文件时的 Open 函数，该函数的申明位于 code/filesys/filesys.cc 中。在 Nachos 文件系统中，有专门的打开的文件类 OpenFile，每当一个文件被打开时，都会有一个 OpenFile 的类指针指向它。OpenFile 类中定义了各种文件操作，如写入读出等。具体变量的定义如右图，其中 hdr 为此时打开文件对应的头指针，由此即可完成对整个打开文件的各数去位置的读取和定位。

```
private:
  FileHeader *hdr;
  int seekPosition;
      //LAB3
      int file;   //file id
      int headerSector;
```

理解了 OpenFile 类后 Open 函数的实现即较容易，只需由文件名在文件目录中找到对应存储的的头文件扇区，并由该扇区创一个新的 OpenFile 类即可。需要注意的是，在文件系统中还有一个特别定义的键值对结构 openedfile，用来存储此时正处在打开状态的文件对应的文件编号和 OpenFile 类指针，故对一个文件读写时直接来该结构中寻找对应的指针即可。

2、添加多级目录

（1）新建多级目录的结构

在 Nachos 原文件系统中，目录结构也为简单的单级目录，即所有虚拟磁盘中的文件均在一个根目录下，且该目录可以存取的上限文件数为 10。新增的多级目录大致实现了在根目录下可创建新的子目录，子目录里也可创建子目录，只有一个要求为在每个目录下存取文件和

子目录总数要小于等于 10 个。为了添加该多级目录结构，实际上并不需要修改太多原目录设计中的结构，如右图所示，其中 DiretoryEntry 目录中一个文件信息的结构体，用以存储文件的三个信息，如右下图，其中 inUse 表示该文件当前是否还为可读取状态，主要用于文件恢复中，而 sector 即为该文件头指针位置，注意子目录的 sector 即为该目录结构存储的扇区编号，有了该编号即可直接从磁盘中读取对应子目录中的信息。name 则表示对应的文件名或子目录名。每个目录都是一个这种结构，因此每个子目录文件都是一个占据一个扇区的只有文件头的文件，目录之间由 DiretoryEntry 中的 sector 号呈一个树状的连接状态（并非实际的树状链表式连接）。

```
private:
  int tableSize;
  DirectoryEntry *table;
```

```
class DirectoryEntry {
  public:
    bool inUse;
    int sector;
    char name[FileNameMaxLen + 1];
```

（2）实现子各种文件操作时多层目录的读入

多级目录的主要实现即在于一步步找到对应树状图内部的文件。因此每当确认一个文件位置之后，由读入位置中的“/”将其子目录一个个划分出来，再从根目录开始，一步步查找下一子目录，并进入下一层子目录，直到到达对应文件的子目录停止。

该过程的实现主要步骤即为，由之前获得的目录 sector 号读入该扇区号下的文件，再从该文件中读取目录类 Directory（若是根目录直接读取 1 号 sector 对应的文件即可），然后从其中的 table 数组中查找 name 为所需子目录的对应 DiretoryEntry，此时调用对应 Directory 类中的 Find 函数即可，找到后提取其中的 sector 信息，继续重复此循环，即可实现多级目录的跨越，找到最最终文件存在的子目录。

（3）实现子目录的创建

最后还要解决的问题就是子目录的创建，此过程和新建一个文件类似，即需要先进入需要创建新目录的子目录中，此时创建从 Bitmap 中找到一个空的新扇区给该目录文件，以该扇区新建一个文件，同时新建一个目录类，并将该空目录写入新建的目录文件中，即可完成一个新的子目录的创建。

3、实现文件恢复

对于 Nachos 中文件的删除，实际上只是将原目录中的 inUse 一项由 true 改为 false，即无法再查找到该文件目录，但在新的文件被写入之前，其目录信息和文件信息均不会丢失。因此为了实现文件恢复，只需要修改一下原 Find 函数即可，即设置一个查找模式，直接在目录中查找文件名而无需检查该文件的 inUse 是否还为 true，这样即可找到还未被新文件覆盖的已经被删除了的文件的头指针，由此即可读出整个文件的信息。核心 Find 函数的修改如下：

```
int
Directory::FindIndex(char *name, bool justCompareName)
{
  if (justCompareName) {//当只需要寻找名字时采取此类搜索，用以恢复文件时的查找
    for (int i = 0; i < tableSize; i++)
      if (!strncmp(table[i].name, name, FileNameMaxLen))
        return i;
    return -1;    // name not in directory
  }
  for (int i = 0; i < tableSize; i++)//恢复以外的文件操作时的查找目录
    if (table[i].inUse && !strncmp(table[i].name, name, FileNameMaxLen))
      return i;
  return -1;      // name not in directory
}
```

三、实验结果

（1）有关多级索引的测试：

由于该测试结果太长且不美观，故已放在实验文件中，此处就不再截图。

讨论一下其测试内容：

由右图可见测试指令，其中-f 首先初始化虚拟文件系统，两个-cp 均是将文件从 unix 文件系统中拷贝进 Nachos，然后-x 为在 nachos 虚拟文件系统中执行 testFileSys 文件。

```
#!/bin/bash
nachos='../build.linux/nachos'
$nachos -f
$nachos -cp prince.txt prince.txt
$nachos -cp testFileSys testFileSys
$nachos -x testFileSys
```

关于 testFileSys 文件内容如右图，其中先新建文件 hello，再打开该文件和另一个已经复制进 nachos 虚拟磁盘中的 prince.txt 文件，再将该文件中的前 512 位读出，存入 str 字符数组中。最后重复 10 次将该字符串写入 hello 文件中。

```
#include "syscall.h"
#include "func.h"

int main(){
    int i;
    int fd1;
    int fd2;
    char str[512];

    Create("hello");
    fd1=Open("prince.txt");
    fd2=Open("hello");

    //Read 512B from fd1, Write 5120B to fd2
    Read(str,512,fd1);
    for(i=0;i<10;i++)
    Write(str,512,fd2);
    Close(fd1);
    Close(fd2);

    //Print file system state
    Print();
    return 0;
}
```

该测试对于文件的新建、nachos 磁盘中扇区分配、文件的拓展和打开均有测试。

最终的 Print 指令会依次打印出当前虚拟磁盘中的所有文件，以及目录文件、Bitmap 文件和当前位图的分配状态。

（2）多级目录和文件恢复的测试：

测试的指令如右图，可见其先初始化文件系统，然后拷贝入对应 testDirectory 文件并运行它，再打印文件系统中位置为 folder1/folder2 下的 file 文件，再删除该文件，最后将该文件恢复到 recovery.txt 中。

```
#!/bin/bash
nachos='../build.linux/nachos'
$nachos -f -cp testDirectory testDirectory -x testDirectory
#print contents
$nachos -p folder1/folder2/file
#delete file
$nachos -r folder1/folder2/file
#recover file to recovery.txt
$nachos -recover folder1/folder2/file recovery.txt
```

对于 testDirectory 文件，其新建了 folder1、folder2 两个子目录，并在其下新建了 file 文件，并向 file 文件中写入了 10 个句子。

运行结果截图如下

```
tkc@ubuntu:~/Desktop/lab3pt1/NachOS-4.0-lab3/code/test$ ./toTestDirectoryAndRecovery.sh
create folder folder2 failed
Machine halting!

Ticks: total 928562, idle 923352, system 4920, user 290
Disk I/O: reads 89, writes 66
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
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
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
```

四、实验中的问题与分析

1、多处关于文件写回的问题

本次实验中花费时间 debug 最长的就是多个文件的新建与写回问题，很多次对于文件头指针、Openfile、Directory、Bitmap 类的新建与写回不正确导致出现了大量奇怪的 bug。

现将各种新建与写回整理如下：

1）FileHeader 类：

FileHeader 类相对简单，只需在申明后直接 new 分配空间：

FileHeader *hdr=new FileHeader;

如此即可完成新建，在一系列操作后再使用内置的 WriteBack 将其写回到头文件分配的对应 sector 中，即如下语句：

hdr->WriteBack(sector);

2）OpenFile 类：

新建一个 OpenFile 类时，也需有该文件对应的 sector，其实等价于找到这个文件头指针存放的 sector，即和 1）中类似，故新建时为：

currentFile=new OpenFile(sector);

由于其代表一个打开的文件类，所以不会对文件有修改，故无需写回。

3）对于 Bitmap 和 Directory 类：

此两类操作几乎一样故放在一起讨论，这两个类都必须从文件中读入其内容，故在新建其之前，现在由对应头文件指针的 sector 号找到对应存放该结构的文件，在从文件中由读取函数读取对应的位置，具体语句如下：

Directory *newDirectory = new Directory(NumDirEntries);

OpenFile *newdirectoryFile = new OpenFile(sector);

newDirectory->FetchFrom(newdirectoryFile);

在写回时同样不需要写回文件，直接写回其目录即可：

newDirectory->WriteBack(newdirectoryFile);

2、对于实验阶段 2 中的写入文件后文件末尾存在少量乱码的问题

如助教最开始给出的截图中最后一行出现了乱码，该问题实际上是在文件拓展过程中直接分配的新 sector 中可能残留有之前文件的内容而没有被清空所导致，此时为了解决该问题，在文件拓展的函数中新加入一个写入语句，每次新分配一个扇区时都先使用对应函数 WriteSector 将一个空白的字符数组 blank 写入扇区对应的磁盘中，即可解决该问题。

```
char blank[SectorSize];
int i;
for(i=0;i<SectorSize;i++)
  blank[i]='\0';
// allocate direct sectors
for (i = numSectors; i < NumDirect && i < numSec;)//从一级直接索引开始分配
    if (dataSectors[i] == -1)//此处判断意义不大 一直成立 如果判断错误会进入死循环
    {
        dataSectors[i] = freeMap->FindAndSet();//分配一级直接索引
        kernel->synchDisk->WriteSector(dataSectors[i], (char*)blank);
       // printf("add sector: %d\n",dataSectors[i] );
        i++;
    }
```

五、实验总结与收获

本次实验中完成了对 Nachos 系统文件系统的一系列学习，更加清晰地认识了文件系统中的每一个类的结构和作用，并亲自体验了对于多级索引和目录的编写，设计了文件恢复的算法。整体上大大加深了我对文件系统的理解。有点可惜都是最后一个实验啦，希望以后的学习中还能继续更加深入地了解操作系统！

参考文献:

[1] Abraham Silberschatz. 操作系统概念. 高等教育出版社, 2007.3.
[2] Nachos 中文教程
[3]OS 实验手册-lab3