山东大学 软件 学院

**操作系统课程设计** 实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202022300316 | 姓名：王丁 | | 班级：软工20.6 |
| 学号：202000300358 | 姓名：梁思睿 | | 班级：软工20.6 |
| 实验编号：Lab4 | | | |
| 实验题目：扩展文件系统 | | | |
| 实验学时：4 | | 实验日期：2022.xx.xx | |
| 实验目的：  1. 扩展Nachos的基本文件系统。Nachos的文件系统是一个简单并且能力有限的系统，限制之一就是文件的大小是不可扩展的。通过扩展，使得文件的大小是可变的。在扩展写入文件内容时，一边写入，一边动态调整文件的长度及所占用的数据扇区。  2. 增加Nachos文件的最后修改时间，并在执行./nachos -D命令时显示。Nachos文件头中存储文件最后修改时间，时间值是从UTC 1970年1月1日00:00:00来的秒数(精确到1秒)，占用原来numSectors的存储位置(从磁盘存储空间效率上考虑，文件头中已经有了文件长度字节数，无需再存储文件内容占用的扇区数)。 | | | |
| 硬件环境：  设备名称 LAPTOP-UUHJO71O  处理器　　AMD Ryzen 5 4600U with Radeon Graphics 2.10 GHz  机带 RAM 16.0 GB (15.4 GB 可用)  设备 ID 63412E57-FC9E-4F92-A869-B35FDEF3303C  产品 ID 00342-35837-59360-AAOEM  系统类型 64 位操作系统, 基于 x64 的处理器  硬盘 512GB SSD | | | |
| 软件环境：  宿主机：Windows 10 21H2 64位  虚拟机软件：VMware Workstation Pro 16.1.2 build-17966106  Linux：Ubuntu 14.04.6 LTS Desktop i386 (Trusty Tahr)  gcc/g++：(Ubuntu 4.8.4-2ubuntu1~14.04.4) 4.8.4  MIPS交叉编译器：gcc-2.8.1-mips.tar.gz  Nachos：Nachos-3.4-UALR-2022 | | | |
| 实验步骤与内容：  **概述**  要想完成本次实验，需要先阅读并理解nachos原有的文件系统框架及其实现方式。本次实验所需前置知识较多，但大部分可以通过回忆上学期内容、阅读源代码来掌握。总体来说不太难，但也需要耐心、细心。  **Nachos文件系统分析**    首先根据PPT的框架梳理源码，会变得简单一些。  先看 filesys.h。这个文件提供了一些基本的文件操作，例如Open(), Create(), Remove() 等等。  再来看fileheader是什么。经过阅读源码不难得知，Fileheader实际上就相当于Unix里面的inode，存储了一个文件除了文件名以外的全部信息，包括文件大小、在磁盘的哪些分区存储等等。  要想操作一个文件，首先得把它打开。Openfile 类定义了几个打开了的文件的操作，包括读写、获取文件长度等。这次我们需要添加一个append() 的方法，使得文件支持追加操作。  然后我们需要知道nachos模拟一个物理磁盘的方式。关于磁盘，nachos用一个名字为DISK的文件来模拟整个磁盘。通过阅读源码可知，磁盘共有32个track，每个track有32个sector，每个sector有128个字节。Sector是磁盘分配的最小单位，也是IO的最基本单位。  所有对磁盘步的管理任务在 Nachos 文件系统中都是由文件 synchdisk.h 中定义的 SynchDisk 类完成的，本次实验中我们不需要对其进行修改。  那么这些磁盘分配的元信息在哪里呢？其实是在bitmap当中。Bitmap 是一个占用一个sector的特殊文件，它的头文件在磁盘的第0个sector 当中。在开发的过程中，只需要 OpenFile \*mapFile = new OpenFile(0); 这样操作就可以读取bitmap到内存中，以供获取或修改磁盘的分配信息。  那么万事俱备，直接开始我们的开发。   1. **扩展Nachos的基本文件系统，使得文件的大小是可变的。在扩展写入文件内容时，一边写入，一边动态调整文件的长度及所占用的数据扇区。**   首先我们需要知道：为什么原来的文件系统大小是不可变的？通过阅读源码可知，nachos文件的结构如下所示：     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  文件只具有1级索引，而且占用的sector数量有限（文件头的大小不能超过一个sector），且没有任何扩展文件的方法。这三个原因，导致了原有的nachos文件系统能力比较有限。在本次实验中，我们将加入append方法。在下一个实验中，我们将改进文件索引方式为二级索引。  那么首先需要先给 FIleHeader 类增加一个能够追加扇区的功能，来了新文件的时候追加 扇区分两种情况讨论。  （1）上次分配的最后一个扇区没有用完，且已有扇区能盛得下新来的文件，那么此时不需要追加额外的扇区。  （2）当前的扇区盛不下了，需要追加新的扇区，用 bitmap 分配管理。  代码如下：  bool FileHeader::Append(BitMap \*freeMap, int fileSize) {      int lastSectorSpace = SectorSize - (numBytes % SectorSize);      if (fileSize > lastSectorSpace) {          fileSize -= lastSectorSpace;  // Put in the last sector, as much as possible          int newNumSectors = divRoundUp(fileSize, SectorSize);  // update necessary new sectors            if (freeMap->NumClear() < newNumSectors) return FALSE;  // not enough space          int oldSectorNum = divRoundUp(numBytes, SectorSize);  // old sector number          numBytes += (fileSize + lastSectorSpace);          for (int i = oldSectorNum; i < oldSectorNum  + newNumSectors; i++) {              dataSectors[i] = freeMap->Find();          }      } else {          numBytes += fileSize;      }      return TRUE;  }  然后需要增加文件的实际大小。我们需要修改 Openfile。这里有两点需要注意，一是 BitMap 文件头所在的扇区是 0（上面已经分析过），找 BitMap 就去 0 号扇区。 二是 bitmap 我们在此处修改过了，要把它写回磁盘。  bool OpenFile::AppendSize(int numBytes) {      printf("Appending!\n");      OpenFile \*mapFile = new OpenFile(0);  // bitmap header is in sector 0      BitMap \*freeMap = new BitMap(NumSectors);      freeMap->FetchFrom(mapFile);  // now freeMap is initialized with its content in disk.      if (!hdr->Append(freeMap, numBytes)) {          return FALSE;  // no enough space left      }      // map is updated, so we need to write it back to disk      freeMap->WriteBack(mapFile);      delete mapFile;      delete freeMap;      return TRUE;  }  由于我们修改了 Inode 结点，要把修改后的 Inode 写回原来它在的扇区， 根据fstest的提示，给OpenFile类添加WriteBack方法，  同时需要注意，OpenFile 里本来是不知道 Inode 原来在几号扇区的，给OpenFile 添加一个成员变量 headSector 来记录下文件 Inode 的头扇区号，并在构造函数里初始化它。  WriteBack 方法的实现非常简单，hdr 调用自己的WriteBack就可以，参数传入 Inode 所在扇区号。  Openfile.h:     private:      FileHeader \*hdr;   // Header for this file      int seekPosition;  // Current position within the file      int headSector;    // Sector number of the file header  Openfile.c:  OpenFile::OpenFile(int sector) {      hdr = new FileHeader;      hdr->FetchFrom(sector);      seekPosition = 0;      headSector = sector;  }  void OpenFile::WriteBack() { hdr->WriteBack(headSector); }  最后修改测试文件ftest.c即可。  void Append(char \*from, char \*to, int half) {  ... ...      if (half) start = start / 2;      openFile->Seek(start);      if (!(openFile->AppendSize(fileLength))) {          printf("Append: couldn't append to file %s:  File is too big.\n", to);          fclose(fp);          return;      }      // Append the data in TransferSize chunks      buffer = new char[TransferSize];      while ((amountRead = fread(buffer, sizeof(char),  TransferSize, fp)) > 0) {          int result;          //  printf("start value: %d,  amountRead %d, ",          // start, amountRead);          //  result = openFile->WriteAt(buffer, amountRead, start);          result = openFile->Write(buffer, amountRead);          //  printf("result of write: %d\n", result);          ASSERT(result == amountRead);          //  start += amountRead;          //  ASSERT(start == openFile->Length());      }      delete[] buffer;      //  Write the inode back to the disk, because we have changed it       openFile->WriteBack();       printf("inodes have been written back\n");      // Close the UNIX and the Nachos files      delete openFile;      fclose(fp);  }  最后进行测试，结果如下。  首先我们新建文件系统，打印磁盘信息。  可以看到，磁盘内容和实验指导书中给出的空磁盘内容一致：    然后放进文件big：    看一下磁盘：    big文件成功放进去了。接下来append一个小文件：    看一下磁盘，可以看到已经成功append。    文件append功能添加成功。   1. **增加Nachos文件的最后修改时间，并在执行./nachos -D命令时显示**   根据实验报告的提示，nachos文件中有个冗余的字段numSectors，我们把它改成一个时间戳就行。这个时间戳是从UTC 1970年1月1日00:00:00来的秒数(精确到1秒)。  那么我们怎么获取文件的sector数目呢？封装成一个函数就好。      int getSecNum() { return divRoundUp(numBytes, SectorSize); }  功能和原来的numSectors一样。  那么何时更改这个字段呢？当然在创建和修改的时候都需要更新。  Nachos没有提供获取墙上时间的函数接口，所以我们不得不使用Unix系统的time  函数。这个函数返回一个time\_t类型，可以转成int类型，这个数值就是UTC标准时间。  void FileHeader::updateTime() {      // time is sec num from UTC 1970.1.1 00:00:00      time\_t now = time(NULL);      int nowSec = (int)now;      lastUpdatedTime = nowSec;  }  在FileHeader::Allocate(),FileHeader::WriteBack(), Openfile::AppendSize(), Openfile::WriteAt() 当中，都需要添加更新时间戳的逻辑。  bool FileHeader::Allocate(BitMap \*freeMap, int fileSize) {      numBytes = fileSize;      updateTime();      // getSecNum() = divRoundUp(fileSize, SectorSize);      if (freeMap->NumClear() < getSecNum()) return FALSE;  // not enough space      for (int i = 0; i < getSecNum(); i++) dataSectors[i] = freeMap->Find();      return TRUE;  }  int OpenFile::WriteAt(char \*from, int numBytes, int position) {      hdr->updateTime();      int fileLength = hdr->FileLength();  ... ...  bool OpenFile::AppendSize(int numBytes) {      hdr->updateTime();  ... ...  那么大功告成，跑一下看看。  首先建立磁盘、看下磁盘信息。    时间戳已经显示出来。现在加个文件。      可以看到时间戳正常更新。  那么时间戳的功能也添加完毕，实验4圆满完成。 | | | |
| 结论分析与体会：  通过这次试验，我们了解了nachos模拟物理磁盘的方式，以及模拟出的物理磁盘的相关参数，充分地认识了Inode 结点的结构和每个部分的作用，熟练的掌握了bitmap的各种方法，。除了每个fileheader/openfile的操作外，我们还理清了他们之间的层次和管理关系，成功的实现了文件扩展功能。  本次实验收获颇丰，尤其是让我们通过实践掌握了操作系统的文件管理功能。我们对操作系统的知识特别是索引结点、位示图等有了更深层次的理解，并且为下个实验打好了基础。 | | | |