山东大学 软件 学院

操作系统 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202000300358 | 姓名：梁思睿 | | 班级：20.6 |
| 实验题目： 具有二级索引的文件系统(Lab5) | | | |
| 实验学时：2 | | 实验日期：2022.10.26 | |
| 实验目的：  1. Nachos系统原有的文件系统只支持单级索引，最大能存取30 \* 128 = 3840字节大小的文件。本实验将在理解原文件系统的组织结构基础上扩展原有的文件系统，在Lab4的基础上，设计并实现具有二级索引的文件系统。  2. 为Nachos增加命令行选项-DI。执行./nachos -DI时显示Nachos磁盘的以下信息：Nachos磁盘的总体大小，已使用空间大小，空闲空间大小，普通文件数目，全部普通文件的总字节数，全部普通文件占用的空间大小(不包括文件头占用的，但加上普通文件数据扇区的内碎片)，总内碎片字节数(仅计普通文件数据扇区造成的)。  3. 若要求为Nachos文件增加rwx权限(可读，可写，可执行)，请给出在Nachos中实现的具体方法(不要求实现可运行的代码。在实验报告中用文字描述即可，必要时可在文字中结合关键代码片段、数据结构、对象等说明)。 | | | |
| 硬件环境：  设备名称 Three  处理器 Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz  机带 RAM 16.0 GB (15.8 GB 可用)  设备 ID 8CA8307A-62B3-42B1-AEE1-871F308ACDF3  产品 ID 00342-35669-83174-AAOEM  系统类型 64 位操作系统, 基于 x64 的处理器  笔和触控 为 10 触摸点提供触控支持 | | | |
| 软件环境：  宿主机：Windows 10 21H1 64位  虚拟机软件：WSL2  Linux:Ubuntu20.04.1LTS (GNU/Linux 5.4.72-microsoft-standard-WSL2 x86\_64)  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-LW | | | |
| 实验步骤与内容：   1. 实验要求与解析   Nachos系统原有的文件系统只支持单级索引，最大能存取30 \* 128 = 3840字节大小的文件。本实验将在理解原文件系统的组织结构基础上扩展原有的文件系统，在Lab4的基础上，设计并实现具有二级索引的文件系统。  实现方法设想：  IMG_256  一级inode前两块仍然存储numBytes和numSectors的信息，后30块本来应该存储扇区信息，在一级索引下最后一块（下标29）应该存储二级索引inode所在的位置（扇区号），所以此时一级索引能用于存放数据内容的扇区数量是29，而二级索引一共32块（NumDirect+1）应该全部存储扇区信息。所以扩展后的文件长度应该是扩大后的文件最大长度（29+#2）\*128 = 7808 Byte  此外我们希望二级索引是动态产生的——当文件大小不需要它时，一级索引块的最后一项设为-1，此时不存在二级索引块。当文件大小增长到一级索引无法支持时，再分配一个新的块存储二级索引，并将其扇区号存储入一级索引块的最后一项。这个需要使用判断条件实现。  我们可能用到二级文件索引的情况有以下两种：   1. 第一次放入的文件就超过了一级索引能够分配的最大大小，用到了二级索引 2. 第一次放入的文件没有使用到二级索引，一级索引最后一个块为-1，但对该文件追加文件的过程用到了二级索引 3. 部分关键源代码分析   **bitmap.cc文件**  BitMap::BitMap(int nitems)  {  numBits = nitems;  numWords = divRoundUp(numBits, BitsInWord);  map = new unsigned int[numWords];  for (int i = 0; i < numBits; i++)  Clear(i);  }  观察相关成员变量：   1. numBits保存了能表示的扇号的位数，numWords保存了组成位示图的字数。map是一个int型指针，私有成员变量map指向一个保存位示图的内存   int BitMap::Find() {  for (int i = 0; i < numBits; i++)  if (!Test(i)) {  Mark(i);  return i;  }  return -1;  }  可以看出，find()的作用是返回找到的第一个空闲位的索引，同时将该位置为“1”    void BitMap::FetchFrom(OpenFile \*file) {  file->ReadAt((char \*)map, numWords \* sizeof(unsigned), 0);  }  void BitMap::WriteBack(OpenFile \*file) {  file->WriteAt((char \*)map, numWords \* sizeof(unsigned), 0);  }  由于内存是易失性的，所以对应硬盘的位示图需要作为一个文件保存在磁盘上。它作为内核一个特殊文件被管理。FetchFrom和WriteBack用于完成这一目的。  Inode节点：  Inode是文件系统的重要数据结构，它保存了除文件名以外的一个文件用到的所有管理信息。在基本的Nachos系统中使用了File Header类实现了简单的一级索引分配方法及其结构。具有三个私有成员变量：  private:  int numBytes; 表示文件中的字节数  int numSectors; 文件中的数据扇区数  int dataSectors[NumDirect]; 每个数据的磁盘扇区号码  每个Inode节点被读入内存后成为一个FileHeader类，在后面的代码中我们了解到使用一个hdr指针指向了读入的Inode并操作  Allocate函数：  在Nachos文件系统中要建立一个新文件时首先调用FileHeader类的成员函数Allocate(BitMap \*BitMap, int fileSize)这个函数根据指定的文件长度fileSize换算出文件数据要占用的磁盘区数，并将申请到的自由扇区好记到扇区索引表dataSectors中。即完成了一个Nachos文件Inode 的初始化工作。  Nachos文件操作命令   1. 建立磁盘：nachos -f 2. 查看磁盘信息： nachos -D 3. 向磁盘中拷贝一个文件： nachos -cp 路径 文件名 4. 向磁盘中的文件追加文件： nachos -ap 路径 文件名   **问题一：Nachos系统原有的文件系统只支持单级索引，最大能存取30 \* 128 = 3840字节大小的文件。本实验将在理解原文件系统的组织结构基础上扩展原有的文件系统，在Lab4的基础上，设计并实现具有二级索引的文件系统**  实现思路  我们是为了让Nachos能存更大的文件所以扩展的二级索引，第一步应该是把原来对“最大”的定义给改了。  同时由于我们分配方式变成了二级索引，所以FileHeader的Allocate分配方法要修改，分为以下两种场景：   1. 一级索引够分 2. 一级索引不够分，用到了二级索引   我们是在实验四的基础上进行二级索引的扩展，所以本来的文件追加功能还是要有的，所以扩展文件的时候可能会用到二级索引。我们的二级索引是动态产生的，类比一级inode结点，如果把它看成个对象的话应该只有一个成员变量dataSectors，大小为32，就够了，它不应该存在一级inode里，应该实现“动态产生”，要用的时候再把这个二级inode的dataSectors弄出来放到某个扇区存起来。   1. 第一步：修改最大文件大小   首先对Filehdr.h中define的文件最大大小进行修改：  #define NumDirect (int)((SectorSize - 2 \* sizeof(int)) / sizeof(int))  #define NumDirect2 (int)(SectorSize / sizeof(int))  #define MaxFileSize ((NumDirect + NumDirect2) \* SectorSize)  一级索引inode节点的-2不应该修改，其逻辑上能存储的数据块的扇区数从30变成29，相对于32-3，由于一级索引的前两块存储了numBytes和numSectors的信息，最后一块要么是-1要么是二级索引节点所在的扇区号，所以能存数据的一共是32-3=29块  一级索引的inode的dataSectors大小本来是使用NumDirect定义的，dataSectors应该还是30而不是29  二级索引全部用来存储数据块所在的扇区号，所以一共32块，不用减   1. 修改Alloctae函数   我们修改第一次放入文件用的Alloctae函数，思路如下：   1. 先判断这个文件整个磁盘是不是能装得下，不能则拒绝放入 2. 判断这个文件用二级索引能不能装得下，不能就拒绝放入 3. 如果一级索引够用就只用一级索引，并且将一级索引的最后一位设为-1 4. 如果一级索引不够用，就把一级索引分配完，之后给二级索引的inode建立出来的并在磁盘上找扇区(bitmap->Find( ))存起来，存的内容就是二级索引的dataSector数组，取名为dataSector2 5. 函数返回   bool FileHeader::Allocate(BitMap \*freeMap, int fileSize){      numBytes = fileSize;      printf("fileSize: %d \n", fileSize);      //numSectors = divRoundUp(fileSize, SectorSize);      printf("numSectors: %d \n", getSecNum());      //updateTime();        if (freeMap->NumClear() < getSecNum())          return FALSE;  // not enough space      else if(NumDirect+NumDirect2<=getSecNum())          return FALSE;  // second index do not have enough space      int lastIndex = NumDirect-1; //the last one of the first index      if(getSecNum()<lastIndex){          for(int i=0; i< getSecNum(); i++){              dataSectors[i] = freeMap->Find();  //Allocate sectors          }          dataSectors[lastIndex] = -1;  //the last one of the first inode is -1      printf("\*\*\*\*\*\*\*\*\*\*\*\*Now the last is: %d\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n",dataSectors[lastIndex]);        }else{          for(int i=0; i<lastIndex; i++){              dataSectors[i] = freeMap->Find();  //Allocate sectors          }          dataSectors[lastIndex] = freeMap->Find();  //find a sector for the second inode          int dataSector2[NumDirect2];          for(int i=0; i< getSecNum()-NumDirect; i++){              dataSector2[i] = freeMap->Find();  //Allocate sectors          }      synchDisk -> WriteSector(dataSectors[lastIndex],(char\*)dataSector2); //write back to the disk      }      return TRUE;  }   1. 修改Deallocate函数   由于修改了分配函数，因此Deallocate函数也需要修改，修改的思路和Allocate是类似的两种情况：   1. 没有用到二级索引，只用了一级，就普通释放 2. 用了二级索引：要先把二级索引在磁盘上上的inode2找出来才能根据inode2去释放二级索引分配的扇区块   void FileHeader::Deallocate(BitMap \*freeMap) {      int lastIndex = NumDirect - 1;      if(dataSectors[lastIndex]==-1){//没用到二级索引的时候的释放          for(int i=0;i<getSecNum();i++){              ASSERT(freeMap->Test((int)dataSectors[i]));              freeMap->Clear((int)dataSectors[i]);          }      } else{//用到了二级索引          int i;          for(i=0;i<lastIndex;i++){//先把一级索引释放了              ASSERT(freeMap->Test((int)dataSectors[i]));              freeMap->Clear((int)dataSectors[i]);          }          int dataSector2[NumDirect2];          synchDisk->ReadSector(dataSectors[lastIndex],(char\*)dataSector2);//先找到二级索引节点          freeMap->Clear((int)dataSectors[lastIndex]);          for(i;i<getSecNum();i++){              freeMap->Clear((int)dataSector2[i-lastIndex]);//释放二级索引节点内容          }      }  }   1. 修改Append   由于文件大小是可扩展的，所以在追加文件的过程中可能用到二级索引，因此AppendSectors需要修改，分为以下情况   1. 如果追加文件的大小能够放在已经分配的扇区里则不需要追加扇区 2. 如果追加文件的大小超出了磁盘大小，则拒绝追加 3. 如果追加文件后文件总大小超出了二级索引能承受的最大大小——拒绝追加 4. 如果本来的文件没有用到二级索引，则分为两种情况   一级索引够用，则在一级索引中追加  一级索引不够用用到了二级索引，则使用Allocate分配二级索引   1. 原来的文件以及用到了二级索引，但是追加的文件不会超过二级索引能够承受的大小，则对二级索引追加 2. 函数返回   注意：想对二级索引进行操作，要先根据以及索引dataSectors的最后一块找到二级索引inode2拿出二级索引的inode2，之后才能对二级索引操作  bool FileHeader::Append(BitMap \*freeMap, int fileSize) {      printf("Append:\n");      int restFileSize = SectorSize\*getSecNum() - numBytes; // the size of the sector which haven't be used      if (restFileSize >= fileSize) {      numBytes = numBytes + fileSize;      printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*the current sector is enough\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n");          return TRUE;      }else{      printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*the current sector is not enough\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n");          int moreFileSize = fileSize - restFileSize; // the current sector is not enough, need to be extend      printf("in Append: moreFileSize: %d \n",moreFileSize);            if(freeMap->NumClear() < divRoundUp(moreFileSize,SectorSize)){              return FALSE;          }else if(NumDirect + NumDirect2 <= getSecNum() + divRoundUp(moreFileSize,SectorSize)){ // the disk is enough but the sum of the first inode and th second is not enough          return FALSE;          }          int i = getSecNum();          numBytes = numBytes + fileSize;            //int AppendSector = divRoundUp(moreFileSize,SectorSize); //caculate the sector numbers need to be extend          //numSectors = numSectors + AppendSector;          int lastIndex = NumDirect - 1;          if(dataSectors[lastIndex] == -1){              if(getSecNum()<lastIndex){          printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*now the first inode is enough\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n");          for(;i<getSecNum();i++){              dataSectors[i] = freeMap -> Find();          }          }else{  // extend the second inode          printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*now the first inode is not enough\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n");          for(;i<lastIndex;i++){              dataSectors[i] = freeMap -> Find(); // allocate the first inode          }          dataSectors[lastIndex] = freeMap -> Find();  // set the sector of the second inode to the last of the first inode          int dataSector2[NumDirect2];          for(;i<getSecNum();i++){              dataSector2[i-lastIndex] = freeMap -> Find(); // allocate the second inode          }          synchDisk -> WriteSector(dataSectors[lastIndex],(char\*)dataSector2);  //write back          }      }else{// have use the second inode, but the extend files haven't exceed the size of second inode          int dataSector2[NumDirect2];          synchDisk -> ReadSector(dataSectors[lastIndex],(char\*)dataSector2);  //find the second inode          for(;i<getSecNum();i++){              dataSector2[i-lastIndex] = freeMap -> Find(); // allocate the second inode          }          synchDisk -> WriteSector(dataSectors[lastIndex],(char\*)dataSector2);  //write back      }      }      return TRUE;  }   1. 修改ByteToSector   int FileHeader::ByteToSector(int offset) {      int lastIndex = NumDirect - 1;      if(offset/SectorSize<lastIndex){          return (dataSectors[offset/SectorSize]);      }else{          int dataSectors2[NumDirect2];          synchDisk->ReadSector(dataSectors[lastIndex], (char \*)dataSectors2);          return (dataSectors2[offset / SectorSize - lastIndex]);      }  }  测试结果：  可以成功创建磁盘    将宿主机中的huge文件拷贝到Nachos模拟磁盘上  ./nachos -cp test/huge huge  可以观察到此时的last仍然为-1，仍然使用一级索引    输入./nachos -D 打印Nachos文件系统的内容    追加一个huge    输入./nachos -D 打印Nachos文件系统的内容    可见追加成功  **步骤二：为Nachos增加命令行选项-DI。执行./nachos -DI时显示Nachos磁盘的以下信息：Nachos磁盘的总体大小，已使用空间大小，空闲空间大小，普通文件数目，全部普通文件的总字节数，全部普通文件占用的空间大小(不包括文件头占用的，但加上普通文件数据扇区的内碎片)，总内碎片字节数(仅计普通文件数据扇区造成的)。**  实现思路：  需要统计的内容：   1. Nachos磁盘总体大小：可以从disk.h文件中获取NumSectors 2. 已经使用的空间大小：可以通过新建BitMap，并且通过FetchFrom(freeMapFile)方法获取位图并逐个统计使用个数 3. 空闲空间大小：Nachos磁盘总体大小 - 已经使用的空间大小 4. 普通文件数目：通过directory->FetchFrom(directoryFile)获取目录，通过遍历目录获取全部文件及其相关信息 5. 全部普通文件的总字节数：遍历目录获取FileHeader，通过FileHeader获取文件字节数hdr->FileLength()加和 6. 全部普通文件占用的空间大小：遍历目录获取FileHeader，通过FileHeader获取hdr->getSecNum()，通过获取的sector数目\*128计算全部普通文件占用的空间大小 7. 内存碎片字节数 = 全部普通文件占用的空间大小 - 全部普通文件的总字节数   相关代码：   1. main.cc中添加 -DI指令   else if (!strcmp(\*argv, "-DI")){              fileSystem->DiskMessage();  }   1. FileSystem中添加方法：   void  FileSystem::DiskMessage() {      // printf("Bitmap DiskMessage: \n");      FileHeader \*bitHdr = new FileHeader;      FileHeader \*dirHdr = new FileHeader;      BitMap \*freeMap = new BitMap(NumSectors);      Directory \*directory = new Directory(NumDirEntries);  printf("Disk size: %d sectors, %d  bytes. \n", NumSectors, NumSectors\*128);  //计算磁盘空间的大小      freeMap->FetchFrom(freeMapFile);      int usedBits = freeMap->getUsedBitsNumber();      printf("Used: %d  sectors, %d  bytes.\n",usedBits,usedBits\*128);  printf("Free: %d  sectors, %d  bytes.\n",NumSectors-usedBits,(NumSectors-usedBits)\*128);  // 遍历位图计算使用的数目，总数目-使用的数目=空闲数目      directory->FetchFrom(directoryFile);      directory->DiskMessage();      delete directory;     // printf("Disk size: %d sectors, %d bytes.",getSecNum(),numBytes);      //printf("Used:  sectors, %d bytes. \n", numBits);  }   1. BitMap中添加getUsedBitsNumber方法   int  BitMap::getUsedBitsNumber()  {      int count = 0;      for (int i = 0; i < numBits; i++){          if (Test(i)){              count++; // 如果使用了，则计数+1          }      }      return  count; //返回计数  }   1. Directory添加方法DiskMessage   void  Directory::DiskMessage(){      FileHeader \*hdr = new FileHeader;      BitMap \*bitmap;      int sectorSum = 0;      int fileSize = 0;      int count=0;      int sectorsFragmentation = 0;      int fragmentation = 0;      for (int i = 0; i < tableSize; i++)          if (table[i].inUse) {              count++;              // printf("Name: %s, Sector: %d\n", table[i].name, table[i].sector);              hdr->FetchFrom(table[i].sector);              sectorSum += hdr->getSecNum();              fileSize += hdr->FileLength(); //分别统计每个文件的大小和占用的sector数目              if(fileSize%128 == 0){                  sectorsFragmentation += 0; //如果文件大小刚好是128倍数，则没有碎片              }else{  //存在碎片，通过sector数目\*128 - 文件大小计算碎片大小                  fragmentation += hdr->getSecNum()\*128 - hdr->FileLength();                  sectorsFragmentation +=  1; //计算碎片大小及占用空间              }              //hdr->Print();          }      printf("%d bytes in %d files, occupy %d bytes(%d sectors). \n", fileSize, count, sectorSum\*128, sectorSum);      printf("%d bytes of internal fragmentation in %d sectors. \n", fragmentation, sectorsFragmentation);      delete hdr;  }  测试结果：  格式化模拟磁盘    第一次写入：    第二次追加：    查看磁盘：    输出磁盘信息：    继续添加文件：        再次查看磁盘：    对比n5screen:    完全一致  删除文件：    再次查看磁盘：    恢复初始状态  步骤三：若要求为Nachos文件增加rwx权限(可读，可写，可执行)，请给出在Nachos中实现的具体方法(不要求实现可运行的代码。在实验报告中用文字描述即可，必要时可在文字中结合关键代码片段、数据结构、对象等说明)。   1. 修改FileSystem的create方法   在其中添加设置权限的方法，并且添加一个参数power，将power值默认设为7（即可读、可写、可执行）  bool  FileSystem::Create(char \*name, int initialSize, int power)  {      Directory \*directory;      BitMap \*freeMap;      FileHeader \*hdr;      int sector;      bool success;      DEBUG('f', "Creating file %s, size %d\n", name, initialSize);      directory = new Directory(NumDirEntries);      directory->FetchFrom(directoryFile);      if (directory->Find(name) != -1)        success = FALSE;          // file is already in directory      else {          freeMap = new BitMap(NumSectors);          freeMap->FetchFrom(freeMapFile);          sector = freeMap->Find();   // find a sector to hold the file header          if (sector == -1)              success = FALSE;        // no free block for file header          else if (!directory->Add(name, sector))              success = FALSE;    // no space in directory          else {                  hdr = new FileHeader; //新建文件                  setPower(power);              if (!hdr->Allocate(freeMap, initialSize))                      success = FALSE;    // no space on disk for data              else {                      success = TRUE;              // everthing worked, flush all changes back to disk                      hdr->WriteBack(sector);                      directory->WriteBack(directoryFile);                      freeMap->WriteBack(freeMapFile);              }                  delete hdr;          }          delete freeMap;      }      delete directory;      return success;  }   1. 新增setPower方法   根据创建文件时传入的参数为其设置权限  bool FileSystem:: setPower(int x){      printf("设置权限：");      int a,b,c;      if(x>=4){          x-=4;          a=1;      }      if(x>2){          x-=2;          b=1;      }      c=x;        if(a==1){          r = true;          printf("r");      }      if(b==1){          w = true;          printf("w");      }      if(c==1){          x = true;          printf("x \n");      }  }   1. 在fileheader中添加参数及方法       bool getW();      bool getR();      bool getX();     private:      bool w;      bool r;      bool x;    具体实现：  bool FileHeader:: setPower(int x){      int a,b,c;      if(x>=4){          x-=4;          a=1;      }      if(x>2){          x-=2;          b=1;      }      c=x;        if(a==1){          r = true;      }      if(b==1){          w = true;      }      if(c==1){          x = true;      }      printf("%d,%d,%d",a,b,c);  }  bool FileHeader::getW(){      return w;  }  bool FileHeader::getR(){      return r;  }  bool FileHeader::getX(){      return r;  }   1. 在Openfile的ReadAt和WriteAt中添加判断   int OpenFile::ReadAt(char \*into, int numBytes, int position) {      if(hdr->getR()){          printf("您有读取权限/n");      }else{          printf("您没有读取权限!/n");          return NULL;      }  ...  }  int OpenFile::WriteAt(char \*from, int numBytes, int position) {      if(hdr->getW()){          printf("您有写权限/n");      }else{          printf("您没有写权限/n");          return NULL;  }  ...  }   1. 在progtest.cc中修改   void  StartProcess(char \*filename)  {      OpenFile \*executable = fileSystem->Open(filename);      if(!executable->getX()){          printf("您没有执行权限！");          ASSERT(FALSE);      }  ...  } | | | |
| 结论分析与体会：  本次实验过程中我进一步熟悉了整个 nachos 文件系统层次结构，并完全掌握了 nachos 文件系统的层次结构中文件模块组织层次全部内容。已经能够熟练使用 BitMap、Openfile 和 FileHeader 的相关函数并能够做出修改（如 AppendSector 方法等）。  我彻底弄懂了索引结点的结构以及索引结点的使用，并且能够做到文件自行追加二级索引。这次实验还让我更加熟悉了 Linux 命令行操作 nachos 文件系统的相关指令如创建文件./nachos cp test/xxx，追加文件 ap，建磁盘-f，显示磁盘信息-D 等等。  此外我还学习到了如何添加命令行指令，并且输出相应的信息，在完成这一部分的过程中，我对文件系统的结构、文件中存储碎片的产生等知识点有了更深刻的理解。  最后一部分，探究如何为nachos添加读写权限，我更加清晰的了解到了nachos读写文件的全过程，需要通过哪些部分的调用最终实现对磁盘的读写。 | | | |