|  |
| --- |
| 实验目的：  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。 |
| 硬件环境：  惠普品牌型号笔记本 Intel Core i5-8300 CPU 8GB内存 512GB SSD |
| 软件环境：  宿主机：Windows 10 21H1 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-LW |
| 实验步骤与内容： 4.1 实验内容 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。 4.2 实验思路 首先我们必须理解Nachos文件为什么是不可扩展的？   * 一个文件的大小是在创建时定的，在header，allocate时候，传入参数fileSize，按fileSize分配恰好合适大小的扇区数量 * 在openfile的write时，当写的位置超过fileSize时，会直接切掉超出部分，只是write不超出部分 * 所以一个文件的大小从创建开始是不会改变的   接下来就是文件拓展的思路：  就是在接收文件长度的时候，判断一下文件长度是否大于磁盘规定的位图中的位数，如果大于的话，就进行拓展操作，重新对位图中的内容进行编写，如果小于或等于的话，就保持之前的操作。这样的话，就可以完美实现文件的拓展操作。 4.3 实验代码4.3.1 给FileHeader增加Extend（int newSize） 判断输入的所有长度是否超过扇区规定的长度，如果超过长度就进行扩展操作  bool FileHeader::Extend(int newSize) {   if(newSize<numBytes)return FALSE; //if not a extend operation   if(newSize==numBytes)return TRUE; //if size not change   int newNumSectors = divRoundUp(newSize, SectorSize); //the number of sectors the new size need to be allocated.  if(newNumSectors == numSectors){  numBytes = newSize;  return TRUE; //if number of sectors new size need equals the   }   int diffSector = newNumSectors - numSectors;   OpenFile \*bitmapfile = new OpenFile(0);  BitMap \*freeMap;  freeMap = new BitMap(NumSectors);  freeMap->FetchFrom(bitmapfile); // printf("debug in fhdr extend where new Sector=%d \n",freeMap->NumClear());  if(newNumSectors>NumDirect||freeMap->NumClear()< diffSector)return FALSE; //if disk is full or file size is too big.   //allocate the new sectors and store them into file header  int i;  for(i = numSectors; i<newNumSectors; i++)  {  dataSectors[i] = freeMap->Find();  }  freeMap->WriteBack(bitmapfile);  numBytes = newSize;  numSectors = newNumSectors;   return TRUE; } 4.3.2 更改OpenFile中的writeAt函数 writeAt函数的空间不够时，将触发文件扩展  if (numBytes <= 0) {// For original Nachos file system // if ((numBytes <= 0) || (position > fileLength)) // For lab4 ...  return 0;  } // check request  if ((position + numBytes) > fileLength) {  hdr->Extend(position + numBytes);  //printf("extend secess");  } 4.3.3 在OpenFile中增加writeback函数，更新磁盘内容 void OpenFile::WriteBack() {  hdr->WriteBack(hdrSector);  } |
| 结论分析与体会：  实验结果  ./nachos -f ./nachos -cp test/big big ./nachos -D    ./nachos -ap test/small big ./nachos -D    ./nachos -hap test/small big ./nachos -D`    从输出结果来看，拓展之后的文件系统在保证了基本的cp命令执行不出错的情况下，增加了ap和hap两个命令，让原来固定大小的文件系统变成了大小可变的文件系统。  本次实验在掌握了nachos基本文件命令用法的情况下，对其功能进行了拓展，增加了文件可变的功能，而实现这个功能的前提也是足够了解nachos磁盘系统相关的源代码，因此通过扩展这个功能，我对nachos文件系统相关的部分有了更深的认识，也对操作系统中所讲的文件系统相关的内容进行了很好的回顾。  其次就是Nachos文件系统机制研究  1、 Nachos 文件系统的组织结构  磁盘其实就是，unix下的一个名字叫Disk的文件，其中I/O控制层，DISK模块完成的工作实际上是将文件指针定位到合适位置，读写Disk的文件。基本文件层SynchDisk，给‘磁盘’读写加锁，保证同步。  2、例子：Nachos如何创建文件，读写文件  1)创建文件  文件系统最高层——Filesys调用它的成员函数Create(char \*name, int initialSize)  然后访问Directory,判断是否还有空目录项  访问Bitmap，查看是否有空扇区，来放置header  如果有空间，增加新目录项，创建新的FileHeader，在hdr中给文件分配空间，再次访问Bitmap，检查是否还有空扇区  如果空间足够，文件allocate成功，将header写回磁盘（位置是第三步分配的扇区），将目录写回磁盘  2）读写文件  文件系统最高层——Filesys调用它的成员函数Open(char \*name)  然后访问Directory，根据文件名Find到该文件i-node扇区位置  OpenFile利用i-node位置，完成文件打开操作  在OpenFile里调用 Write(char \*from, int numBytes)，开始写文件  写的时候，会访问header，查看FileSize等属性  写文件调用SynchDisk，SynchDisk再调用Disk  3、Nachos的文件为什么是不可扩展的  由前两步，对文件创建，读写的操作流程，可以看出。  一个文件的大小是在创建时定的，在header，allocate时候，传入参数fileSize，按fileSize分配恰好合适大小的扇区数量  在openfile的write时，当写的位置超过fileSize时，会直接切掉超出部分，只是write不超出部分，所以一个文件的大小从创建开始是不会改变的。 |