山东大学 软件 学院

**操作系统课程设计** 实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202000300125 | 姓名：贾星宇 | | 班级：计软20.5 |
| 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年11月15日星期二 | |
| 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：  HUAWEI matebook14 2020笔记本  Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz CPU  8GB内存  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：  根据各个实验的内容，以及实现的过程，可写的包括但不限于：  解决问题的思路；  实现步骤；  关键源代码及注释(尽量以文本格式，且必须与提交的源代码一致)；  程序说明，特别是自己认为的精彩之处；  调试过程及记录；  运行结果(可文本格式，必要时抓屏)；  等等…   1. 对文件系统的理解：   首先浏览lab4文件夹中的main函数，找到与文件系统部分有关的语法：    可以发现与文件系统相关的命令皆定义在此处，这也是文件系统的入口。如，-cp命令可以从Unix系统中选择文件并复制到nachos系统中；-ap命令定义了追加一个 Unix 文件的内容到一个已存在的 Nachos 的文件中，也就是我们要实现的命令之一；同时，-hap命令定义了重写一个Unix文件的内容到一个已存在的nacho文件中。  输入**-cp命令**后，可以看到main函数中首先执行了ASSERT方法判断参数给出的数量是否大于2，如果满足要求后，则执行Copy方法将第一个参数对应的Unix系统中文件读取到nachos系统中，并命名为第二个参数。Copy函数如下所示：    在此方法中，首先打开UNIX文件，随后将文件系统的长度存储到fileLength参数中，随后利用Create创建新文件。Create方法效仿了UNIX系统的创建方法：确保文件不存在、为文件头分配扇区、为文件的数据块分配磁盘空间、为目录添加名称、在磁盘上存储新文件头、将更新刷新到bitmap和目录中，并存储到磁盘上。在Create方法中实现了创建一个与源文件相同长度的Nachos文件。    （图：Create方法）  下一步利用Open方法打开了创建的新文件，并存储到openFile中。  随后利用缓冲将数据以TransferSize大小的块利用while循环不断将文件内容利用Write方法写入openFile中。随后关闭fp指针指向的源文件，删除创建的无用指针，完成了Copy操作。  在Write方法中，调用WriteAt方法：    在此方法中，将form中的字符从文件的position开始，一共写numBytes个。首先判断给的初始位置是否合法（大于0且小于文件长度），随后如果从此位置开始写，写完总长度大于文件长度，则截取到文件长度。随后得到新的numBytes，并计算写进的块个数，随后复制到缓冲中并利用缓冲写回磁盘。  可以看到，在复制操作时，首先打开了需要复制的文件，然后读取到此文件大小，然后在nachos系统中创建了新的空文件并规定了文件大小，随后利用buffer将源文件数据复制到新创建的文件中，最后清理内存并结束。   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。   在一开始，计划根据n4areadme文件对lab4源代码进行修改，但对文件中所写内容理解欠佳，所以让我们顺着程序执行的数据流方向去分析如何实现：  扩展的第一个命令为-ap，在main函数中找到此命令的入口：    此方法体在确保了健壮性（argc>2）后便进入了**Append方法**开始执行，所以我们来到Append方法，这也是我们重点关注的方法。此方法首先利用fopen方法打开了第一个参数中所携带的文件名对应的文件，并将FILE指针保存到**fp**中。Fp即我们需要存储到nachos系统中的文件。随后获得文件长度存储到fileLength中，并判断此文件长度是否为0。随后判断要append to的文件是否存在，如果通过Open方法打开为NULL，则利用Create方法创建一份文件。此时，**openFile**便存储了需要append操作的文件的FILE指针。    （图：Append前期准备）  随后便开始Append操作:首先利用start存储要从何处开始写入新的字符，如果half为1，则从文件中间位置开始写，否则从文件末尾开始写。利用Seek方法将当前文件位置存储为start位置。创建新的缓冲buffer，为长度为10的字符数组。随后利用了fread函数进行while循环：  C 库函数 size\_t fread(void \*ptr, size\_t size, size\_t nmemb, FILE \*stream) 从给定流 stream 读取数据到 ptr 所指向的数组中。  因此(amountRead = fread(buffer, sizeof(char), TransferSize, fp)) > 0含义为从源文件fp中读取数据到buffer中，读取了TransferSize个字符。如果读取到的大于0，即还没有读完，则利用Write方法将buffer写入目标文件openFile中，写入的位置先前通过Seek方法定位到了。  写完成后，删除缓冲数组和打开的要写入的文件，关闭打开的UNIX的文件结束。    （图：Append操作）  整个方法感觉完整流畅，看来没有需要大幅修改的代码，因此涉及到修改的话，首先要检查的地方是while循环中的Write方法，通过此方法把一次循环得到的缓冲结果写入openFile中，标记了写入的字长为amountRead。因此我们定位到Write：    依旧没有发现需要修改的地方，但其中有个重点方法WriteAt，进去看看。  WriteAt方法为将from所指向的字符串从position位置开始写入文件，共计写入长度为numBytes：    在这里我们看到了需要做的修改：进入方法后需要判断输入是否合法，原程序判断如果起始位置position>=文件长度，则输入不合法。但如果要进行可调整文件长度的扩充，那么有可能存在从文件末尾处开始扩充。我们也看到下一行已经有了相应的注释提示如何修改，我们只需要将注释打开即可。  同时，下一行判断添加了numBytes长度的字符串后长度是否超过了文件长度，如果超过了则让numBytes变小，相当于截取到≤文件长度位置。但如果进行可变长度的扩充，这两行也需要注释掉。但是，我们还需要做一些其他的操作。  可以知道，先前文件长度不可扩展时，这儿当输入的长度达到最大文件长度后只需要截断就可以，但是现在我们需要扩展，即如果超过了先前的文件长度，那么我们就修改源文件的文件头信息，让文件长度及其对应的扇区等信息写入源文件。因此我们在FileHeader类中添加方法**setNumBytes**，并在WriteAt中做如下修改：    然后我们前往FileHeader类中写修改文件头的方法：    **这是最为关键的一个方法，核心思想为175行到179行，即如果可以扩展，则分配额外需要的扇区，并扩展扇区数和字节数。在开始进行了相应的判断：如果字节数小于现有的字节数，则不需要扩展，进入此方法表示产生了错误。如果字节数相等，也不需要扩展，但此时没有错误。如果字节数多了，但扇区数不需要变，则只需要修改字节数即可；如果现在的扇区数多余最多的扇区数或者需要扩大的扇区数多余剩余的扇区数则返回错误。**  WriteAt函数下方的代码为对内存块的操作以及复制文件到磁盘，不需要做较大的修改，因此我们回到Write之前的Append方法中继续向下看。  根据《操作系统课程设计指导教程》的提示以及相关注释，我们看到当通过while循环执行完写文件到磁盘后，还需要写入文件的inode，即文件头信息。我们打开如下两行注释：    可以发现我们需要在openFile中实现一个WriteBack方法，这刚好与n4areadme文件中的提示相对应。此方法用于修改文件头，核心功能为修改文件长度为新长度。  既然是修改文件头，我们进入FileHeader类查看详情：    可以发现类中存在方法WriteBack，输入为磁盘扇区号，方法实现了将文件头的修改写回磁盘。那么我们只需要在OpenFile类中添加此方法调用FileHeader中的写回方法即可。同时，要添加sector参数到OpenFile中，表示文件头所在扇区。可以看到，构造函数中已经有了相关提示。      添加写回方法：    有意思的是，起初构造函数中没有32行，但是创建文件时系统始终会把找到的文件头的扇区数返回给此构造方法。  **根据n4areadme文件，我们还需要依次实现-hap和-nap。**  -hap为从文件的一半处重新写，而且调用了相同的Append方法，只是half为1。因此我们去看WriteAt方法，如果现在文件位置在half处，那么一开始写的时候不需要扩展，正常运行不会被if捕获。如果需要扩展则扩展，无影响，所以已经完成。  随后是-nap方法。此方法实现了从nachos中取出文件复制到相应的文件中，调用的NAppend方法。我们进入此方法，方法第一个参数表示从哪儿复制，第二个参数表示依附到哪个文件。方法整体框架类似于Append，因此我们按照上述方法先修改NAppend，解开WriteBack的两行注释。由于先前已经修改了Write使其满足扩展，因此我们无需做其他操作。两者区别主要在于读from文件用的fread方法还是Read方法。  随后进行测试：首先make并删掉先前DISK，随后依次输入：  ./nachos -f:  ./nachos -cp test/big big    ./nachos -D    ./nachos -ap test/small big    ./nachos -D    ./nachos -hap test/small big    ./nachos -D    ./nachos -ap test/small small    ./nachos -nap small small2  解决问题：如果small为cp的，则没问题；如果small为ap的，则有问题。    ./nachos -D    **随后我们实现n4breadme文件中的要求，在OpenFile类析构函数~OpenFile中自动写回文件头，且不判断文件长度或文件头是否改变。**  我们将Append和NAppend方法中的注释恢复，并在析构函数中添加：    如此便实现了在关闭文件时候自动协会文件头的操作。  随后实现n4creadme，要求在析构函数中判断是否修改过文件头长度，如果长度改变则写回。因此我们做如下修改：    在WriteAt中：    给文件头新增属性ifChanged，默认为false，如果需要扩充openFile，则将其ifChanged设置为true，在写回后设置为false。  可以验证，在扩充文件后才后打印written back。    在n4dreadme中，要求在FileHeader类的析构函数~FileHeader中，判断文件长度改变时自动写回文件头。注意这儿是FileHeader类，我们前往此类构造析构函数    值得注意的是，为了能够写回，我们添加了新属性sector记录文件头所在扇区。   1. 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)。   首先我们发现输入-D后调用的是fileSystem->Print();，随后我们顺着此方法，根据screen文件提示，我们找到了FileHeader类中的print方法：    在此方法中，打印了对应文件头的文件长度和所占扇区数，根据提示，我们将在此处新增打印时间的语句，打印效果：  File modification time: Thu Aug 12 00:38:17 2021.  这时，我记起了先前实现的，当修改文件后自动写回文件头的方法，在OpenFile类的析构函数中:    那么，我们也可以在if语句中顺便将文件最后修改时间（即获得当前时间）写入hdr的numSectors中。根据提示man 2 time，我们得到了C语言获取自1970年1月1日00:00:00来的秒数的方法：  time\_t seconds; seconds = time(NULL);    同时，由于numSectors属性为私有，新增set方法：    随后，当创建文件或者文件被修改时，设置ifChanged为true，就完成了每次创建文件及修改文件头时候写入当前时间到numSectors。随后我们前往文件头的print方法并修改。主要修改为根据numSectors中记录的秒数计算当前的时间，并以字符串的形式随着文件信息一同打印出：    需要注意的是，由于我们占用了numSectors属性，所以在类FileHeader中要用到此属性的时候，需要先根据字节数除以块大小并向上取整计算出numSectors，随后计算。为了统一，我采用如下块将需要用到numSectors属性的代码框出：  **int temp = numSectors;**  **numSectors = divRoundUp(numBytes, SectorSize);**  **//代码**  **numSectors = temp;**  至此，便实现了打印文件信息时打印修改时间的功能，效果如下：    根据readme文件得到的最后结果：    需要注意的是，当复制文件时，**需要将源文件时间复制到新文件：**  如small2复制于small： | | | |
| 结论分析与体会：  此实验是我目前为止耗时最长的一个实验，但正是这种厚积薄发的感觉，令我学到了许多新的经验和技能。  首先是对文件系统的深入了解，文件头、比特文件、文件体等等。  其次我明白了不能单单根据readme提示或者单单根据实验大纲提示做内容，需要将两者结合。此实验首先应该通读readme和实验大纲提示的指导教程，随后从程序运行的数据流角度入手，根据程序要求去写，遇到问题了再去查看参考文件，这是效率比较高的一种做法。  同时，也应该学会正确的debug，比如通过多处打印操作查看程序开在了哪儿或者数据在哪个地方出现了错误。  此致    （图：利用打印实现bug的查找） | | | |