|  |
| --- |
| **《操作系统》课程实习（设计）报告** |

|  |
| --- |
| **题目： WFS文件系统** |

|  |  |  |  |
| --- | --- | --- | --- |
| 学生姓名 | 李瀚宇 | 电话 | 18902833039 |
| 学 号 | 202144011308 | | |
| 学 院 | 计算机科学与工程学院 | | |
| 专 业 | 计算机科学与技术 | | |
| 班 级 | 计科4213 | | |
| 成 绩 |  | | |
| 日 期 |  | | |

**广东海洋大学学生课程实习（设计）报告**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 选题背景 FUSE (https://github.com/libfuse/libfuse) 是一个 Linux 内核扩展，它允许用户在用户空间实现一个文件系统。WFS是基于Fuse框架制作的一个简易的文件系统。  FUSE由三个部分组成：linux内核模块、FUSE库 以及mount 工具。用户关心的只是FUSE库和mount工具，内核模块仅仅提供kernel的接入口，给了文件系统一个框架，而文件系统本身的主要实现代示位于用户空间中。FUSE库给用户提供了编程的接口，而mount工具则用于挂在用户编写的文件系统。  FUSE起初是为了研究AVFS(A Virtual Filesystem)而设计的，而现在已经成为 SourceForge的一个独立项目，目前适用的平台有Linux, FreeBSD, NetBSD, OpenSolaris和 Mac OS X。  官方的linux kernel版本到 2.6.14 才添加了FUSE模块，因此 2.4 的内核模块下，用户如果要在FUSE中创建一个文件系统，需要先安装一个FUSE内核模块，然后使用 FUSE库和API来创建。  WFS 文件系统是一个 FUSE 应用，也就是说WFS是利用FUSE框架创建的一个可以在linux上运行的真正的文件系统。 **设计内容和要求**系统需求描述 目前，WFS目前已经实现了最简单最基础的功能，通过框架给予的接口和数据结构，将不足之处完成：   1. 文件目录里缺乏文件权限、文件主、时间日期等信息，请予以补全。 2. 文件的大小目前最大只有一个磁盘块大小，请实现文件的大小可以是多块。 3. WFS目前只支持一级目录，即所以有的文件均在根目录下，请在WFS下实现多级目录。 4. 按照目前WFS目前的设计，文件的访问是一种顺序访问的方式，请改进WFS的数据访问方式为随机访问。  框架总体设计 WFS 文件系统是一个 FUSE 应用，也就是说WFS是利用FUSE框架创建的一个可以在linux上运行的真正的文件系统，下面是对WFS的基本介绍：  WFS 文件系统实现在一个大小为5M的磁盘映像文件diskimg上，即这个diskimg文件作为磁盘设备承载WFS。将这个文件放在/tmp目录下。WFS 文件系统的布局如下。假定磁盘的每块大小为 512 字节。所以平均每 block 大小为 512 bytes，然后整个5MB文件共 10240 块：     * 超级块是文件系统的第一块，描述整个文件系统。其结构如下：   *// 超级块中记录的，大小为 24 bytes（3个long），占用1块磁盘块*  struct super\_block  {      long fs\_size; *// size of file system, in blocks（以块为单位）*      long first\_blk; *// first block of root directory（根目录的起始块位置，以块为单位）*      long bitmap; *// size of bitmap, in blocks（以块为单位）*  };   * 目录   目录也被同时处理为文件。每个目录项格式如下：  *// 记录文件信息的数据结构,统一存放在目录文件里面，也就是说目录文件里面存的全部都是这个结构，大小为 40 bytes，占用1块磁盘块*  struct file\_directory  {      char fname[MAX\_FILENAME + 1]; *// 文件名 (plus space for nul)*      char fext[MAX\_EXTENSION + 1]; *// 扩展名 (plus space for nul)*      time\_t atime; */\* 上次访问时间 \*/*      time\_t mtime; */\*上次修改时间 \*/*  *// time\_t ctime;                 /\* 上次文件状态改变时间 \*/*      int uid; *//*      int mode; *//*      size\_t fsize; *// 文件大小（file size）*      long nStartBlock; *// 目录开始块位置（where the first block is on disk）*      int flag; *// indicate type of file. 0:for unused; 1:for file; 2:for directory*  };   * 文件名格式为8.3。所谓8.3格式短文件名规范，就是型如 ABCDEFGH（目录）或者ABCDEFGH.exe（文件）这样的名称——“8”是指文件名或目录名的主体部分小于等于8个字节；“3”是指文件名的扩展名部分小于等于3个字节。另外还有一点，就是8.3文件名的有效字符不包括空格等特殊字符。 * WFS文件由有一系列链接的块组成，每一块对应于一个磁盘块，其格式如下：   *// 文件内容存放用到的数据结构，大小为 512 bytes，占用1块磁盘块*  struct data\_block  {      size\_t size; *// 文件使用了这个块里面的多少Bytes*      long nNextBlock; *// （该文件太大了，一块装不下，所以要有下一块的地址）   long的大小为4Byte*      char data[MAX\_DATA\_IN\_BLOCK]; *// And all the rest of the space in the block can be used for actual data storage.*  };   * WFS的中存在一个基于内存的文件索引结构：   *// 目录文件索引*  struct file\_index  {      char path[50]; *// 文件目录*      struct file\_directory \*file\_dir; *// 文件信息*  };  由此可以看出，这个WFS的结构更像FAT，与ext2文件系统的结构有一定的差距。   * WFS实现的函数（系统调用）如下表所示：  |  |  |  | | --- | --- | --- | | WFS\_getattr | Description: | This function should look up the input path to determine if it is a directory or a file. If it is a directory, return the appropriate permissions. If it is a file, return the appropriate permissions as well as the actual size. This size must be accurate since it is used to determine EOF and thus read may not be called. | | UNIX  Equivalent: | man -s 2 stat | | 返回值: | 0 on success, with a correctly set structure  -ENOENT if the file is not found | | WFS\_readdir | Description: | This function should look up the input path, ensuring that it is a directory, and then list the contents.  To list the contents, you need to use the filler() function. For example: filler(buf, ".", NULL, 0); adds the current directory to the listing generated by ls -a In general, you will only need to change the second parameter to be the name of the file or directory you want to add to the listing. | | UNIX  Equivalent: | man -s 2 readdir  However it’s not exactly equivalent | | 返回值: | 0 on success  -ENOENT if the directory is not valid or found | | WFS\_mkdir | Description: | This function should add the new directory to the root level, and should update the .目录 file | | UNIX  Equivalent: | man -s 2 mkdir | |  | 0 on success  -ENAMETOOLONG if the name is beyond 8 chars -EPERM if the directory is not under the root dir only  -EEXIST if the directory already exists | | WFS\_rmdir |  | Deletes an empty directory | | UNIX  Equivalent: | man -s 2 rmdir | | 返回值: | 0 read on success  -ENOTEMPTY if the directory is not empty  -ENOENT if the directory is not found  -ENOTDIR if the path is not a directory | | WFS\_mknod | Description: | This function should add a new file to a subdirectory, and should update the .目录 file appropriately with the modified directory entry structure. | | UNIX  Equivalent: | man -s 2 mknod | | 返回值: | 0 on success  -ENAMETOOLONG if the name is beyond 8.3 chars  -EPERM if the file is trying to be created in the root dir  -EEXIST if the file already exists | | WFS\_write | Description: | This function should write the data in buf into the file denoted by path, starting at offset. | | UNIX  Equivalent: | man -s 2 write | | 返回值: | size on success  -EFBIG if the offset is beyond the file size (but handle appends) | | WFS\_read | Description: | This function should read the data in the file denoted by path into buf, starting at offset. | | UNIX  Equivalent: | man -s 2 read | | 返回值: | size read on success  -EISDIR if the path is a directory | | WFS\_unlink | Description: | Delete a file | | UNIX  Equivalent: | man -s 2 unlink | | 返回值: | 0 read on success  -EISDIR if the path is a directory  -ENOENT if the file is not found |  系统设计与实现get\_fd\_to\_attr根据文件的路径，到相应的目录寻找该文件信息流程图展示具体实现步骤 首先更加path确认是否为根目录，更具目录进一步递归查找其父目录，获取到当前目录所在的块，最后在这个目录块中查找文件，查找成功把这个文件放入索引并返回。 getattr获取文件属性（包括目录的） 基于fuse框架的函数声明如下：  static int ufs\_getattr(const char \*path, struct stat \*stbuf);  其操作是通过路径path，找到所处文件的属性赋值给stbuf 流程图展示具体实现步骤 调用get\_fd\_to\_attr 将path路径递归的展开，获取其file属性，根据属性，填充stat目录文件。 创建文件夹或目录 实现的函数声明如下：   |  | | --- | | static int ufs\_mkdir(const char \*path, mode\_t mode);  static int WFS\_mknod(const char \*path, mode\_t mode, dev\_t dev) |   为增强系统的耦合性，以及利用linux中目录以及文件都是基于文件，两个函数实际上调用的调用创建文件函数，同个函数的不同参数情况。  create\_file\_dir(const char \*path, int flag)  通过使用这个函数来创建一个文件或者目录 流程图展示具体实现步骤 调用create\_file\_dir后，首先查找该目录的起始块，然后在这个目录链中查找是否存在同名的文件，若没找到就在这个目录块中创建一个新的目录项，如果超过限额就新建一个快，在新块中创建新的目录项，完成创建文件返回。 删除文件或目录 实现的函数声明如下：   |  | | --- | | static int WFS\_unlink(const char \*path)  static int WFS\_rmdir(const char \*path) |   删除文件或目录的具体实现同样为同一个函数的不同参数，调用remove\_file\_dir这个函数实现。 流程图  具体实现步骤  1. 获取path获取该文件夹父文件夹的属性 2. 通过文件夹属性，判断是否为空文件夹。如果非空则返回错误 3. 初始化该path文件夹存放子文件夹/文件的磁盘块block。并将其释放（即在bitmap\_block中相应位修改成0）。 4. 修改父文件夹的相应属性信息并写回磁盘。 5. 删除对应缓存的索引  向文件写内容 WFS\_write 实现的函数声明如下：   |  | | --- | | static int ufs\_write(const char \*path, const char \*buf, size\_t size,  off\_t offset, struct fuse\_file\_info \*fi); |   其操作将buf里大小为size的内容，从path指定文件的内容的起始块后的第offset字节写入。 流程图实现步骤  1. 获取path对应文件的属性 2. 通过offset跳过文件内容的m个block后开始写 3. 顺序查找文件内容的第m块磁盘，通过size得知一共需要写多少块磁盘 4. 若第m块磁盘不够写，则创建一个新块，将剩余buf内容写入后续块，直到写完为止。 5. 修改path对应文件的属性并写回磁盘。  读取文件内容 ufs\_read 实现的函数声明如下：   |  | | --- | | static int ufs\_read(const char \*path, char \*buf, size\_t size, off\_t offset,struct fuse\_file\_info \*fi) |   其操作是根据路径path找到文件（非文件夹）起始位置，再偏移offset长度开始读取数据到buf中，并返回文件大小。 流程图实现步骤  1. 通过path找到文件属性。 2. 通过文件属性找到文件内容的起始磁盘块。 3. 按offset得出跳过磁盘块的数目，并获取最终需要写的磁盘块位置。 4. 从指定磁盘开始读取size内容并赋值给buf（如果跨磁盘块，则继续读后续磁盘块，直至读完size长度，或读完整个文件）  源程序清单 该代码总共有9个文件组成分别有以下作用   |  |  |  | | --- | --- | --- | | 序号 | 文件名 | 作用 | | 1 | Makefile | 编译脚本文件 | | 2 | WFS.c | 核心代码文件 | | 3 | WFS.h | 重要数据结构，函数声明 | | 4 | Hashmap.c | Hash函数库 | | 5 | Hashmap.h | Hash函数库接口 | | 6 | Hash.c | 哈希辅助函数定义 | | 7 | Log.c | 日记函数 | | 8 | Init\_disk.c | Wfs初始化函数 |  运行结果 使用初始化diskimg，放置于/tmp目录进行测试，按照一下流程对其进行测试 运行ls指令 使用ls -l指令，系统显示了原始镜像的3个基础文件 在根目录创建文件 执行echo hello > dd.txt && ls -l 指令，系统可以显示创建的文件信息 读取文件 使用cat ./dd.txt指令，系统显示文件内容为hello 删除文件 使用指令rm -rf ./dd.txt 文件被删除，最后通过ls -l指令查看文件状态，确认文件已经被删除 创建跨区文件 使用字符串生成器生成一串长600的文字，使用指令存入ff.txt中  文件大小等信息正确，最后使用cat对其进行访问  确认写入成功。 二级目录测试 使用mkdir gg 指令创建子目录  进入gg 在目录中创建一个文件，并读取  说明文件系统中存在二级目录系统 设计总结体会。 这次课程设计采用fuse文件系统框架，并在Linux系统中进行编码调试。在这次课程设计中，我还加深了Linux中编译操作、Makefile文件的编写一节linux文件系统块等结果等有了更深层次的理解。为了研究fuse的代码的实现方式，在网上查找相应资料，通过example文件夹中的几个例子，了解了相关fuse框架的接口，并一点点修改各个功能。通过原有的基于的框架代码，逐渐改写成一个完整的系统。  同时，这个系统也存在着不少缺乏之处，其一是删除文件中，缺乏块的回收机制，在多次创建和删除文件后会导致目录块冗余，导致空间和时间被浪费，其二是代码部分地方存在一些bug，细节仍然不够完善等。 |

|  |
| --- |
|  |

**评 语：**

|  |  |
| --- | --- |
| 指导教师 | 日期 |

**附：**

**《操作系统课程设计》**

**课程设计内容及考核要求、评分标准**

**一、选题**

在给定的选题中，选择一个题目作为课程设计内容进行实现：

①WFS文件系统；②筛素数进程版；③筛素数线程版

**二、课程设计内容及考核要求**

(1)要求用操作系统编程思想来完成系统的设计；系统设计要具有实用性。

(2)编程要简单清楚，实用，功能较全。

(3)课程设计完成后，每人上交一份设计报告，必须用专用的课程设计报告纸书写，要求具有以下内容：题目，原理叙述、设计有关的基本规则，流程图，源程序清单，运行结果，调试情况分析，设计总结体会。

(4)课程设计报告严禁抄袭，否则按不及格论。

(5)采用报告评阅、代码程序运行和回答问题的方式进行考核，结合课程设计报告进行成绩评定。

**三、评分标准**

本课程设计主要考核综合程序开发和课程设计报告。具体评分标准如下：

1．文档结构合理（10%）

2. 综合程序开发（60%）

（1）需求分析（5）

（2）界面设计（10）

（3）程序功能及实现（30）

（4）程序代码及理解、设计与开发（5）

3．论述详细（20%）

4．格式规范（10%）

**四、附录：**

课程实习（设计）报告要求3000字数以上。