****

课程设计报告书

**一个用户级文件系统的设计与实现**

**学 院 计算机科学与工程学院**

**专 业 计算机科学与技术**

**学生姓名 骆明楠**

**学生学号 201330551358**

**指导教师 吴一民**

**课程编号 145158**

**课程学分 2**

**起始日期 2016年1月**

|  |  |
| --- | --- |
| 教  师  评  语 | 教师签名：  日期： |
| 成  绩  评  定 |  |
| 备  注 |  |

**一个用户级文件系统的设计与实现**

## 一**、选题背景**

FUSE（用户空间文件系统）是这样一个框架：它使得FUSE用户在用户态下编写文件系统成为可能，而不必和内核打交道。

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。

解决的问题是，通过fuse提供的接口，创建一个u\_fs 文件系统。

## 二、方案论证(设计理念)

1. 安装fuse 2.7.0文件系统框架。FUSE使用fuse\_operations来给用户提供编程结构，让用户通过注册自己编写的函数到该结构体来实现自己的文件系统。模块化开发，简化了编码。
2. 完成的功能有：

* 以某文件作为我们磁盘，组织磁盘的文件管理，其中包括Super block、Bitmap block、Data block。采用位图方式记录磁盘块使用情况。

其结构如下

|  |  |  |
| --- | --- | --- |
| Super block  （1 block） | Bitmap block  (1280 blocks) | Data block  (all the rest blocks) |

* 需要实现接口函数:

|  |  |  |
| --- | --- | --- |
| getattr | readdir | mkdir |
| rmdir | mknod | write |
| read | unlink | open |
| flush | truncate |  |

* 编写Makefile文件，简化编译步骤

1. 运行环境：Linux系统。

**三、过程论述**

1. 在Linux系统中安装fuse 2.7.0文件系统框架。

|  |
| --- |
| 进入fuse 2.7.0文件夹，输入如下命令即可：  ./configure --disable-kernel-module  make  make install |

1. 创建一个二进制文件作为磁盘

此处用某5M的文件，重命名为newdisk，作为该文件系统的磁盘。

1. 初始化磁盘：
2. 由于文件原先有内容，故先将其内容全清空，以防止干扰文件系统运行。
3. 根据Super block、Bitmap block、Data block的结构，进行初始化磁盘。

其中一个block的结构为:

|  |
| --- |
| struct u\_fs\_disk\_block { // 512 bytes  int size; //how many bytes are bveing used in this block  //The next idsk block,if needed This is the next pointer in the linked allocation list  long nNextBlock;  //And all the rest of the space in the block can be used for actual data sotorage.  char data[MAX\_DATA\_IN\_BLOCK];  }; |

Super block： 结构如下

|  |
| --- |
| struct super\_block { //24 bytes  long fs\_size; //sizes of file system, in blocks  long first\_blk; //first block of root directory  long bitmap; //size of bitmap,in blocks  }; |

首先获取磁盘大小，从而赋值fs\_size，由于super block占1 block，bitmap block 占1280 blocks，故赋值first\_blk、bitmap分别为1281、1280.

将该结构写入磁盘，即super block初始化完毕。

Bitmap block:

此处记录整个磁盘的占用情况。由于super block、bitmap block均已占用，且data block中为root directory预定了一个block，故bitmap需要将这1+1280+1个block置1(即已占用)，此处bitmap采用的是位图记录算法，故需将bitmap block中的前1282个bit置为1，其余置为0。即可初始化bitmap block完成。

Data block：

此处存放文件或者文件夹的属性以及内容。文件/文件夹的结构如下:

|  |
| --- |
| struct u\_fs\_file\_directory { //64 bytes  char fname[MAX\_FILENAME + 1]; //filename(plus space for nul)  char fext[MAX\_EXTENSION + 1]; //extension(plus space for nul)  long fsize; //file size  long nStartBlock; //where the first block is on disk  int flag; //indicate type of file. 0:for unused.1:for file. 2:for directory  }; |

在data block的第一个block中需要初始化root directory的属性信息。开始root directory没有子文件，故size=0，nNextblock=-1 data为空。写入相应磁盘位置，即可完成data block的初始化。

1. 各接口的功能实现：

getattr：

实现的函数声明如下：

|  |
| --- |
| 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。

具体实现步骤：

|  |
| --- |
| 1. 获取path父目录的属性(存放于u\_fs\_file\_directory中)： 2. 通过path获取父目录的路径。 3. 读取super block，从而找到data block的起始位置（或直接定位为第1282 block处）。 4. 读取root directory所处块（即第1282块）的内容(存放于u\_fs\_disk\_block 结构中)。该block中存放的都是u\_fs\_file\_direcotry。依次读取该block的文件属性，通过文件的属性判断文件夹名称等于父目录的名称。如果确定为该父目录，则获取父目录的属性。   二、 通过父目录属性，获取path对应的文件属性：  1. 通过父目录属性，获取父目录的存放子文件属性的第一块block的位置。  2. 读取上述的block内容，循环赋值u\_fs\_file\_direcotry，查看其文件名是否为path所指文件名。如果不是，则继续查找block的后续块。  3. 如果找到path，则赋值文件属性u\_fs\_file\_direcotry。  三、根据path对应文件的类型，进行stat的赋值（如权限、文件大小等），并返回给fuse框架。 |

readdir：

实现的函数声明如下：

|  |
| --- |
| static int ufs\_readdir(const char\*path, void \*buf, fuse\_fill\_dir\_t filler, off\_t offset, struct fuse\_file\_info\*fi) |

其操作是通过路径path，找到其全部子文件/子文件夹，将其名称通过filler函数填充进buf中。

具体实现步骤：

|  |
| --- |
| 1. 获取path获取该文件夹的属性(操作类似getattr函数) 2. 每一个文件夹都有. 和 .. 子文件夹，将其添入buf 3. 通过path文件夹的属性，获取子文件/文件夹的存放位置 4. 循环赋值，获取子文件/文件夹的属性，并将其名称通过filler函数添加进buf中 5. 如果path的子文件/文件夹属性存放大小不止一块，则循环获取后续块，执行步骤3，直到u\_fs\_disk\_block-> nStartBlock为-1 |

mkdir：

实现的函数声明如下：

|  |
| --- |
| static int ufs\_mkdir(const char \*path, mode\_t mode); |

其操作是通过路径path，增加新的文件夹：

具体实现操作：

|  |
| --- |
| 1. 通过path，找到其父目录名称 2. 通过父目录名称，找到父目录的属性 3. 通过父目录的属性，找到父目录的子文件/文件夹的存放块的其实位置。 4. 在存放块中，顺序查找空闲位置，以添入path新文件属性 5. 如果当前块已满。则继续赋值后续块，继续执行步骤4，如果没有后续块，则寻找一块新的block（空闲块）作为后续块，写入path新文件的属性。 6. 修改父目录的属性信息（如子文件/文件夹大小），并写回磁盘。 7. 寻找n块连续空闲块步骤： 8. 读入bitmap\_block块，从其1282bit开始向后查找（因为磁盘的前1282块都已经占用了） 9. 寻找连续n个bit都是0的位置，并保存查找到的最大连续空闲块max\_num。 10. 如果max\_num等于n，则查找成功并返回。否则继续查找。 11. 如果有后续块，则继续赋值查找，执行b步骤，如果没有后续块，且尚未找到n块连续块，则返回找到的最大空闲块max\_num |

rmdir：

实现的函数声明如下：

|  |
| --- |
| static int ufs\_rmdir(const char \*path)； |

其操作是将path路径对应的空文件夹删除

具体实现步骤：

|  |
| --- |
| 1. 获取path获取该文件夹父文件夹的属性 2. 获取path的文件夹的属性 3. 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 4. 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）。 5. 初始化path文件夹属性的存放位置（如果清空该位置后可空出一个磁盘块，则回收磁盘块。 6. 修改父文件夹的相应属性信息并写回磁盘。 |

mknod：

实现的函数声明如下：

|  |
| --- |
| static int ufs\_mknod(const char \*path, mode\_t mode, dev\_t rdev)； |

其操作是根据path创建新文件。

具体实现步骤：

|  |
| --- |
| 1. 与mkdir函数类似，只是u\_fs\_file\_directory->flag应该为1（表示文件） 2. 创建好文件后需要寻找新的磁盘块，用来存放该文件内容(与mkdir类似，只是mkdir中新的磁盘块是用来存放子文件/文件夹属性) |

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。

具体实现步骤：

|  |
| --- |
| 1. 获取path对应文件的属性 2. 通过offset得知需要跳过文件内容的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，则继续将剩余buf内容写入后续块。如果全部后续块也不够写，则按剩余内容申请相应的空闲块，继续写（如果申请空闲块不够，则继续申请，直至写完buf内容或写满磁盘为止） 5. 修改path对应文件的属性并写回磁盘。 |

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。

具体实现步骤：

|  |
| --- |
| 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） |

unlink：

实现的函数声明如下：

|  |
| --- |
| static int ufs\_unlink(const char \*path)； |

其操作是是将path对应的文件删除。

具体实现步骤：

|  |
| --- |
| 1. 获取path获取该文件夹父文件夹的属性 2. 获取path的文件的属性 3. 遍历path文件内容的磁盘块，将磁盘块都初始化，并释放。 4. 初始化path文件属性的存放位置（如果清空该位置后可空出磁盘块，则回收该磁盘块） 5. 修改父文件夹的相应属性信息并写回磁盘。 |

open：

实现的函数声明如下：

|  |
| --- |
| static int ufs\_open(const char\*path, struct fuse\_file\_info \*fi); |

此处不用实现其操作，将其返回让fuse操作即可。

flush：

实现的函数声明如下：

|  |
| --- |
| static int ufs\_flush(const char\*path, struct fuse\_file\_info \*fi) |

此处不用实现其操作，将其返回让fuse操作即可。

truncate：

实现的函数声明如下：

|  |
| --- |
| static int ufs\_truncate(const char \*path, off\_t size); |

此处不用实现其操作，将其返回让fuse操作即可。

编写makefile

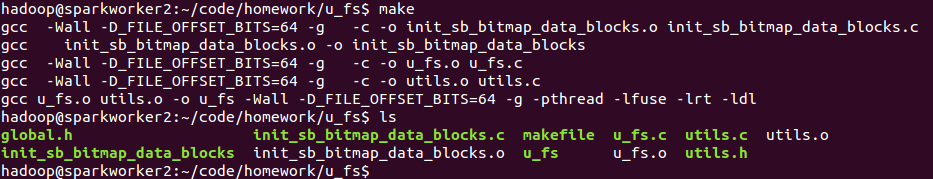
## 三、结果分析

1. 挂载文件系统：

* 先准备好一个5M文件，命名为~/code/homework/newdisk
* 查看当前文件：



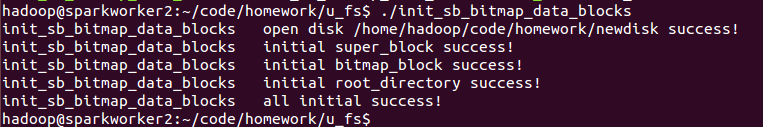
* 编译文件 make



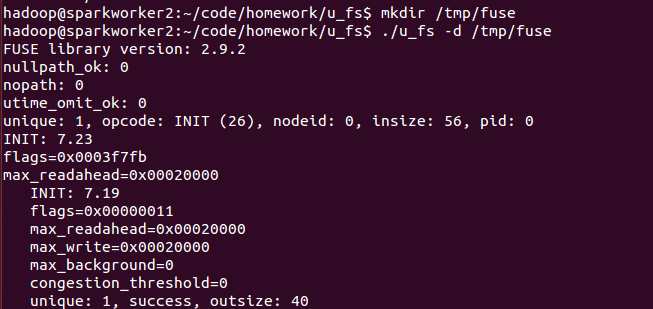
此处可能有warning，可忽略。

* 初始化磁盘（5M磁盘）

直接运行init\_sb\_bitmap\_data\_blocks程序



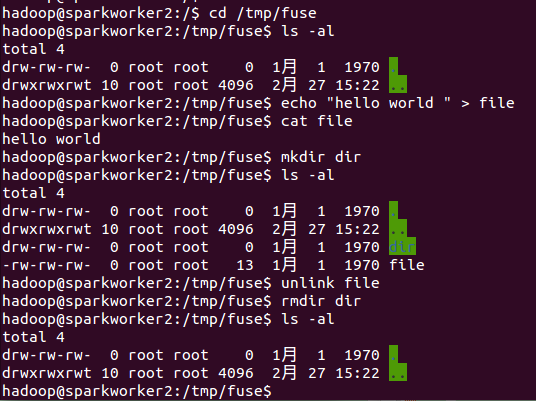
* 创建新文件夹（创建挂载点），并挂载文件系统：



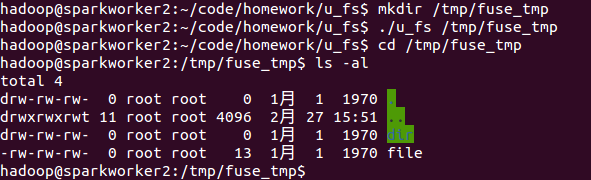
* 由于开启调试模式，故另开终端进行测试：

测试内容有

|  |  |
| --- | --- |
| 执行命令 | 调用的接口 |
| echo "hello world " > file | mknod 、write |
| cat file | open、read |
| mkdir dir | mkdir |
| ls -al | readdir、getattr |
| unlink file | unlink |
| rmdir dir | rmdir |
| fusermount –u /tmp/fuse | 卸载挂载点 |



重新挂载文件系统到另外一个文件夹（/tmp/fuse\_tmp）中，发现文件依旧在。即文件系统可用。



## **四、课程设计总结**

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，难度比以前课程设计的要大。

为了研究fuse的代码的实现方式，在网上查找相应资料，且调出fuse源代码研究其原理。通过example文件夹中的几个例子，一点点修改各个功能。期间询问了师兄关于某些功能的代码编写，也仿写了其他一些接口代码，经过两个星期的努力，终于做出了该文件系统。

整个课程设计过程中，收获最大的就是Linux的文件系统实现代码流程，以及C语言的编码框架及结构。此文件系统采用位图方式记录磁盘块占用情况，在查询空闲磁盘块中，采用了最先符合方式进行分配磁盘，其中可以进一步优化：如采用最差匹配、最优匹配磁盘等方式，或修改磁盘块的大小以来减少碎片等等。

在整个功能设计中，代码尚存冗余，在调用某些功能函数时候，存在部分调用做了些无谓的工作，从而使整个运行速度有所减缓。再次，此文件系统没有碎片管理、整合功能，在多次创建修改文件/文件夹后将会导致该文件系统操作缓慢。

此次课程设计，我还加深了Linux中编译操作、Makefile文件的编写，在Linux系统中进行开发其他C/C++项目或测试时，感觉异常舒服及流畅。

## 五、参考文献

[1]Andrew S.Tanenbaum（著），陈向群，马洪兵（译） 现代操作系统（原书第三版） [M]. 北京：机械工业出版社.2009.7

[2] 鸟哥 著；王世江 改编 Linux私房菜［M］．北京：人民邮电出版社． 2014.6

[3] W. Richard Stevens，Stephen A. Rago 著；戚正伟，张亚英，尤晋元 译 Unix环境高级编程 ［M］．北京：人民邮电出版社． 2014.6

[4] 侯捷 著；STL 源码剖析 ［M］．北京：华中科技大学出版社． 2007.6

[5] UC技术博客 FUSE源码剖析［EB/0L］

http://tech.uc.cn/?p=1597

[6] FUSE(Filesystem in Userspace)简介和使用［EB/0L］

http://blog.csdn.net/jiangandhe/article/details/5739391