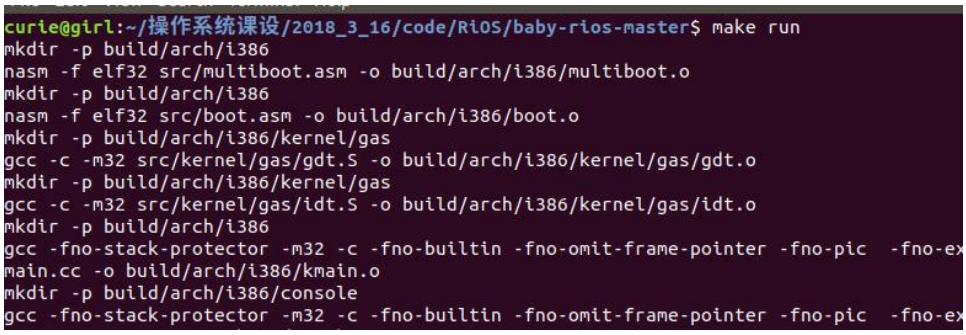
《计算机操作系统课程设计》

测试报告（自测）

题目名称	基于 x86 架构的操作系统之文件系统设计与实现					
院系	信息学院	班级	计科 151	测试报告时间	2018/3/16	
指导老师	姜海燕	助教姓名	徐灿		电话 QQ	
姓名 学号	邱日 19215116	手机 QQ	手机:18260068503 qq:3103204417		申请成绩	A+
姓名 学号		手机 QQ			申请成绩	

计划完成任务与功能（需细化）	1.设备管理,自下而上,从裸机硬件出发,编写 I/O 设备驱动程序,为文件系统提供读写硬盘的系统接口．(已完成) 2.8253 和 8259A 的初始化,建立 32 个不可屏蔽中断异常的处理和响应程序．(已完成) 3.内存管理,包括内存分配和回收,为文件系统编写做好准备.(已完成) 4.字符块设备，包括键盘的响应．(已完成) 5.硬盘分区表的探测和初始化(已完成) 6.inode,superblock 结构体的建立(已完成) 7.磁盘的格式化．(已完成) 8.终端 shell 的编写．(已完成) 9.建立目录文件(mkdir)和建立普通文件(touch) 10.进入目录(cd) 11.显示文件内容(cat) 12.系统调用,读写硬盘(已完成） 13.目录的删除及普通文件的删除(rmdir rm) 14.inode 位图的分配与回收（已完成） 15.完成类似 L inux 中 hexdump 命令的功能，显示任意磁盘扇区的内容（已完成） 16.创建文件 17.磁盘空闲区基于成组链接方法的管理（初步完成） 18.将底层操作包装成 write 和 read 函数等 总结：文件系统和设备管理联系紧密，前期先做设备管理相关的工作，目前设备管理和内存管理已经做完，向文件系统的设计奠定了基础.inode 位图和块位图的初始化经过这几天的编写也已经完成，磁盘空闲区基于成组链接的管理初步完成.

安装使用说明	1.本次实验环境采用 Ubuntu 18.10 2. cd 到代码目录敲 make 命令，完成编译，并自动启动 qemu 虚拟机 
裸机硬件部件仿真设计	采用成组链接管理空闲块.采用物理硬件 **参数设定** 目前磁盘 10MB, 磁盘前一部分是一些控制信息，我们设数据区前 8MB 放除了专用块之外的空闲块,810242=16384sectors,设数据区(专用块)一共 8*1024 块，64 块划分成一组,一共 128 块,专用块和普通的空闲块并没有什么本质上的不同. **初始化** 初始化时，若之前未初始化，先指定第一组的第一块为专用块，把此块复制到内存专用块中;如果已经初始化，从磁盘加载超级块到内存，得到专用块的块号．所有组的第一块相互链接，类似一个顺序表，这些组的第一块第一项存空闲块计数，第二项存下一块的块号，当专用块用完时，它就指定它的下一块是专用块，并在超级块中更改专用块的块号．组号写代码时从 0 开始编号， 0 到 1 2 7

通用数据结构设计	存于硬盘的超级块和内存中的超级块结构 **struct super_block{** **u16 s_ninodes;** **u16 s_capacity_blks; /*capacity count in blocks*/** **u16 s_startsect;** **u16 s_zone_bitmap_blks; /*according to Prof Jiang,we will not use this policy (data block bitmap) anymore.*/** **u16 s_inode_bitmap_blks; /*num of blks that bitmap takes up*/** **u16 s_inode_blks;** **u16 s_firstdatazone;** **u16 s_specific_blk_nr_group; /*成组链接专用块对应磁盘上的组号*/** **u16 s_magic; /*ri_fs magic:0x88*/** **/*These are only in memeory*/** **/*code here.addtional info in memory*/** **};**

```
struct d_super_block
{
	u16 s_ninodes;
	u16 s_capacity_blks;		/*capacity count in blocks*/
	u16 s_startsect;
	u16 s_zone_bitmap_blks;		/*according to Prof Jiang,we will not use this policy (data block bitmap) anymore.*/
	u16 s_inode_bitmap_blks;	/*num of blks that bitmap takes up*/
	u16 s_inode_blks;
	u16 s_firstdatazone;
	u16 s_specific_blk_nr_group;	/*成组链接专用块对应磁盘上的组号*/
	u16 s_magic;			/*ri_fs magic:0x88*/
};
存于硬盘的 d_inode 和内存中的 m_inode 结构
struct m_inode
{
	u8 i_mode;			/*file type(dir/normal) and attribute(rwx)*/
	u8 i_size;
	u8 i_uid;		/*user id*/
	u8 i_gid;		/*group id*/
	u8 i_nlinks;			/*num of files that link to it*/
	u8 padding0;
	u32 i_creat_time;
	u16 i_zone[10];
	u16 i_ino;			/*inode id 号   (bitmap)*/
	u32 padding1[8];		/*占位   8*32 个字节*/
/* ok,let's make sizeof(d_inode) exactly equal to 64,that's 512bits,
 * a sector can put exactly 8 of d_inode.
 * if we attemp to extend the m_inode and d_inode,make sure that
 * they are in sync with each other,and adjust the fields and paddings
 * without changing the sizeof(d_inode)
 */

/*请控制好 d_inode 的大小以及与 m_inode 同步性．这里设置几个 padding 的意义在于占位，
 *我把 d_inode  的大小控制在 8*6+32+16*10+16+32*8=512 bits,这样一个扇区 512*8=4096bits,
 *正好可以放 8 个 d_inode,尽量避免跨扇区操作 inode;
 */

/*
 * zone[0~6]: direct block
 * zone[7]:    single indirect block
 * zone[8]:    double indirect block
```

```
 * zone[9]:     trible indirect block
 */
/*These are only in memeory*/
      u32 i_access_time;
      u8 i_mount;
      u8 i_dev;            /*hd0 or hd1*/
      u8 i_dirty;
      u8 i_updated;
      struct task_struct *i_wait;   /*not implement yet*/
}__attribute__((packed));
/*一定要加，不然字节对不齐，会多用空间*/

struct d_inode{
      u8 i_mode;              /*file type(dir/normal) and attribute(rwx)*/
      u8 i_size;
      u8 i_uid;            /*user id*/
      u8 i_gid;            /*group id*/
      u8 i_nlinks;              /*num of files that link to it*/
      u8 padding0;
      u32 i_creat_time;
      u16 i_zone[10];
      u16 i_ino;              /*inode id 号*/
      u32 padding1[8];
}__attribute__((packed));
目录项
struct dir_entry{
      u32 inode;
      u8 name[MAX_NAME_LEN
```

功能测试	可依次执行如下命令，结果正确: **ls** **cd home** **cd qiuri** **cd /** **cd var** **ls** **cd /** **cd etc** **ls** **cd getty** **ls** **cd /** **mkdir dir1/dir2/dir3** **ls** **cd dir1/dir2/dir3** **ls** **cd ..** **ls** **clear** **cd /** **4help** **pwd** **cd home/qiuri** **pwd** **info superblock** **info disk** **info grouping**

功能测试	**ls** **cd ..** **ls** **rmdir qiuri** **ls** **cd ..** **ls** **rmdir home** **ls** **cat hamlet.txt** **cat jane.txt** **ls** **rm hamlet.txt**

```
[ OK ]  file system has been successfully initialized.                118357
[root@localhost]# ls
 .  ..  usr  lib  dev  home  var  etc  jane.txt  hamlet.txt
[root@localhost]# cd home
 cd home
[qiuri@/home]# ls
 .  ..  qiuri
[qiuri@/home]# cd qiuri
 cd qiuri
[qiuri@/home/qiuri]# cd /
[root@localhost]# cd var
 cd var
[qiuri@/var]# ls
 .  ..
[qiuri@/var]# cd /
[root@localhost]# cd etc
 cd etc
[qiuri@/etc]# ls
 .  ..  getty  passwd
[qiuri@/etc]# cd getty
 cd getty
[qiuri@/etc/getty]# ls
 .  ..
[qiuri@/etc/getty]# cd /
[root@localhost]# mkdir dir1/dir2/dir3
```

优点总结（

1.本套程序具有很强的系统性，虽然是做一个文件系统，但涉及操作系统课本的大部分章节，具体涉及到处理器管理，中断和异常的处理，系统调用的实现；涉及存储管理，基于连续空间分配和回收存储空间；涉及设备管理，编写了键盘驱动、字符显示设备的驱动、ATA 硬盘驱动，整个系统能够操纵屏幕键盘和读写硬盘；当然，涉及文件系统，涉及基于位图的空间分配回收和基于成组链接的磁盘空间分配与回收，以及文件目录树等的实现.

2.个人认为本次实验难度很大，我的工作量也非常大，以上所说各章节内容，除了文件系统还在继续做之外，其他已经彻底完成. 比如设备管理和存储管理等等，每一个单独拿出来都可以作为一次课程设计的内容. 这对我自己也是很大的挑战，从放假开始就一天不停地阅读相关理论书籍，编写相关代码，目前已经编写了三四千行代码. 其中编写和调试的过程也是很艰辛的.

```
curie@girl:~/gitcode/OS_exp/final_exp/19215116_邱日/Code/baby-rios$ cloc .
      87 text files.
      86 unique files.
      11 files ignored.

github.com/AlDanial/cloc v 1.74  T=1.62 s (47.6 files/s, 4642.1 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C++                             30            318            589           2909
Markdown                         9            483              0           1289
C/C++ Header                    29            233            184            976
Assembly                         4             51             32            167
make                             2             48             59            111
JSON                             1              1              0             19
Python                           1              1              5             17
C                                1              1              0             14
-------------------------------------------------------------------------------
SUM:                            77           1136            869           5502
-------------------------------------------------------------------------------
```

不足及改进建议	**1.目前文件系统还没彻底写完，上层操作的接口还没进行封装，需要一段时间继续做下去.** **2.之前准备用位图管理 inode 和数据区，且代码已经编写完，后来应姜老师要求，将位图管理数据区改为成组链接，也基本做完，还需要在细节上调整一下.**