哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2020 年秋季学期

课程名称：操作系统

实验名称：简单文件系统的设计与实现

实验性质：设计型实验

实验时间：11 月 11 日地点：T2210

学生班级：1801101

学生学号：180110115

学生姓名：方澳阳

评阅教师：

报告成绩：

实验与创新实践教育中心印制

2020 年 10 月

实验目的

以Linux系统中的EXT2文件系统为例，熟悉该文件系统内部数据结构的组织方式和基本处理流程，在此基础上设计并实现一个简单的文件系统。

实验环境

环境: **ubuntu 20.04**

IDE: **clion 2020.3**

语言：**C**

标准：**C11**

实验内容

（1）实现青春版 `Ext2` 文件系统。
系统结构参考 `ext2` 系统即可，不需要严格按照 `ext2` 设计，可简化，可自由发挥，但必须是模拟了文件系统，并能完成如下功能：

1. 创建文件/文件夹（数据块可预分配）；
2. 读取文件夹内容；
3. 复制文件；
4. 关闭系统；

系统关闭后，再次进入该系统还能还原出上次关闭时系统内的文件部署。

（2）为实现的文件系统实现简单的 shell 以及 shell 命令以展示实现的功能。 可参考实现以下shell命令：

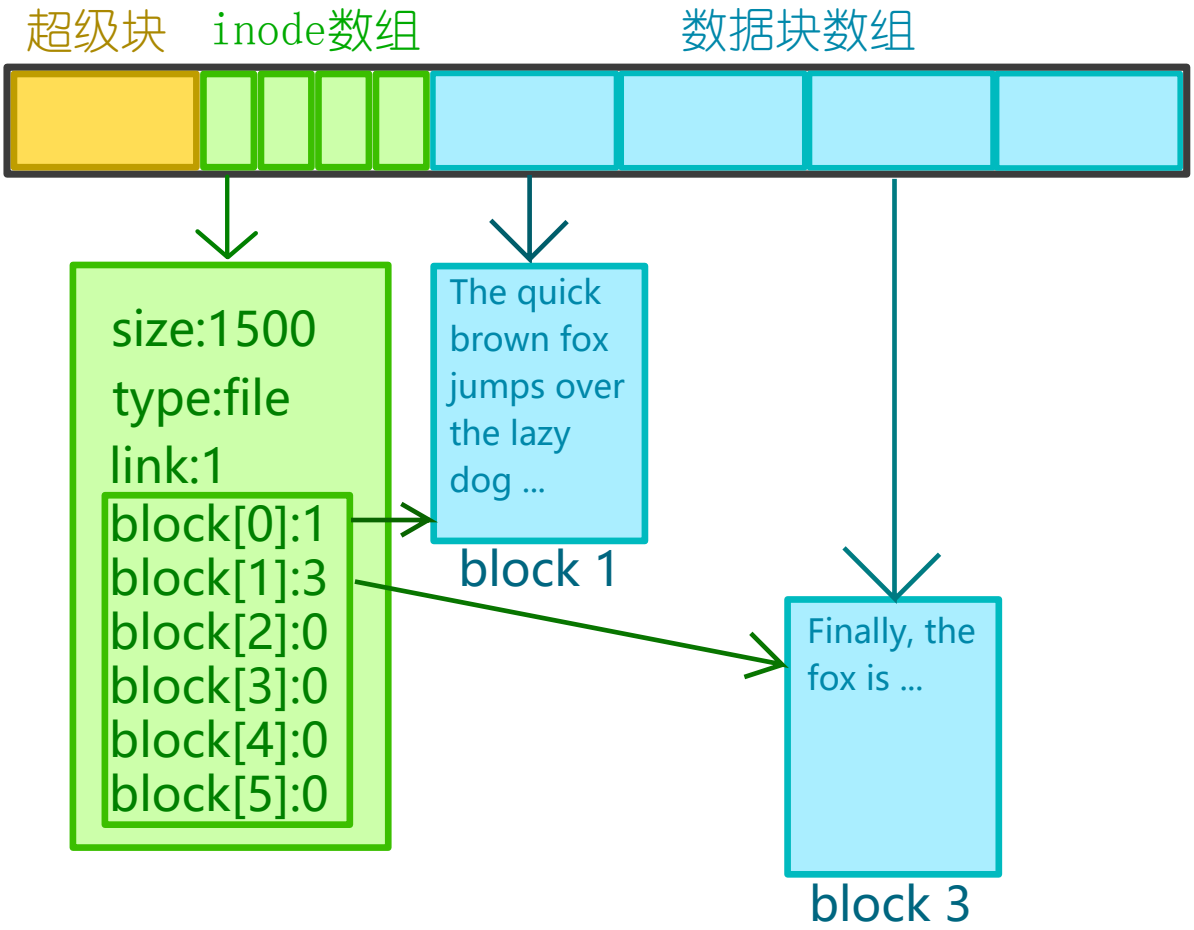
1. `ls` - 展示读取文件夹内容
2. `mkdir` - 创建文件夹
3. `touch` - 创建文件
4. `cp` - 复制文件
5. `shutdown` - 关闭系统

实验过程

通过阅读实验手册，我们可以大致了解本次实验的内容。数据结构已经由指导书给出，如 `inode`，`dirItem`，`superBlock` 等。我们需要做的是将这些数据结构连接起来，互动起来。

对于 `Ext2` 文件系统，需要了解并掌握其的工作原理。需要注意的是，目录和文件的存放类型都是 `inode`，通过 `file_type` 字段来区分。但是不同的是，如果一个 `inode` 是文件类型的，他指向的一些 `block` 是 **全都是专属于这个文件的**，文件越大，占的 `block` 数量越多。

如下图所示



而如果`inode`类型是文件夹。则该`inode`指向的`block`里存放的是一堆`dir_item`。`dir_item`同样也有着文件和文件夹两种类型。通过`dir_item`结构体中的`type`字段标明。如果是文件，则这个`dir_item`指向了`inode`数组中的某个`inode`，从而指向正确的文件。如果是文件夹，也同理，再指向`inode`数组中的某个`inode`。然后重复这个过程，直到找到正确的文件或文件夹。

换句话说，在切换多级目录的时候，比如`cd /tmp/testDir`时，需要在`inode`和`block`之间来回跳跃两次。首先在根目录的`inode`指向的`block`中，找到`tmp`所在的`dir_item`，然后根据这个`dir_item`找到`tmp`文件夹所在的`inode`，再在`tmp`的`inode`指向的`block`中，找到`testDir`所在的`dir_item`，再找到`testDir`所在的`inode`。此时如果要进行`ls`命令，就可以根据当前目录，即`testDir`的`inode`，找到对应的`block`，然后打印出在这个文件夹下所有的信息。

不过`cd`命令并没有在实验要求范围之内。

设计接口

根据实验内容要求，可以分析成以下内容。

在首次启动文件系统时，需要初始化。这部分的初始化选项可以交给用户，作为一个格式化磁盘的选项。清除所有数据，重写超级块的内容。

对于新建文件的命令，需要以下步骤

1. 判断在哪里创建一个文件，如果是根目录，则执行第二步，否则执行第四步
2. 创建一个`inode`。
3. 为`inode`分配6个数据块的指针，结束
4. 遍历所有类型是目录的`inode`，去里面找第一层文件夹的名字。
   比如`touch config/zsh/zsh_config.txt`，需要遍历所有在当前目录下(即根目录下)对应的`block`，在这些`block`中遍历，直到找到名字是`config`的`dir_item`，然后再根据这个`dir_item`中的`inode_id`找到对应的`inode`。此时当前目录相当于是在`config/`目录下了，一开始是在根目录下。重复以上步骤，直到找到`zsh`所在的inode，然后去这个`inode`的对应的`block`

中追加一个`dir_item`，修改`valid`号，修改名称，设置目录项类型为文件。在`inode`数组中追加一个`inode`项，为其指定6个block，将这个inode的id给`dir_item`。

5. 结束。

从上面的几个步骤可以归纳出几个函数。

```
1. createInode(); // 新建inode
2. createDirItem(); // 新建Diritem
3. findFolder(); // 遍历给定目录(输入一个id)对应的block中遍历，直到找到名字是xxx的dir_item，返回inodeid
```

其中，在寻找文件时,与寻找文件夹的步骤是一样的，所以可以合并成同一个函数。具体看`fs.h`中的定义。

对于创建文件夹的命令，与创建文件如出一辙，只需将类型改为文件夹即可。

对于`ls`命令，也差不多，只需要找到那个文件夹的id号之后，在其`block`中打印出所有的是`valid`的`dir_item`项即可。

还剩最后一个复制文件的命令。假设已经找到了对应的目录，则只需要在当前目录里的`block`中找到源文件，拷贝一份（新增）相关信息（`inode`），在这个目录下的`block`新建一个`diritem`，将这个新增的`inode`的`id`填到这个新建的`diritem`中。

实现过程

本项目共有以下文件

```
fs.h        // 系统的定义及实现
fs.c
util.h      // 一些工具函数
util.c
disk.h      // 实验给的模拟磁盘读写.我封装了一些函数
disk.c
testModule.h //测试模块
testModule.c
main.c
```

`util`提供了一些位操作的支持，如下面的前三个函数；提供了字符串的处理，如字符串分割、去掉字符串两边的空格、去掉回车；提供了打印位图的函数。

```
// util.h
int bit_isset(const uint32_t *array, uint32_t index);

void bit_set(uint32_t *array, uint32_t index);

void bit_clear(uint32_t *array, uint32_t index);

char *simple_tok(char *p, char d);

char *trim(char *c);

void rmEnter(char *c);

void printBit(uint32_t *array, uint32_t size);
```

`disk`主要添加了4个封装后的读写函数，将读写磁盘封装到`disk_read\disk_write`中；并且封装出一个读写一整个`blcok(1024byte)`的函数，方便使用。

```
void disk_write_whole_block(unsigned int block_num, char *buf);

void disk_read_whole_block(unsigned int block_num, char *buf);

/**
 * 封装后的disk_write_block,读写安全
 * @param block_num
 * @param buf
 */
void disk_write(unsigned int block_num, char *buf);

/**
 * 封装后的disk_write_block,读写安全
 * @param block_num
 * @param buf
 */
void disk_read(unsigned int block_num, char *buf);
```

`fs`主要定义了一些文件系统的操作。看函数名即可知道其作用。

```
void initExt2();

void printSuperBlock(const sp_block *sp_block_buf);

void printInode(iNode node, FILE *fp);

int createInode(uint32_t blockNum, uint32_t size, uint16_t file_type,
uint16_t link);

int createDirItem(uint32_t blockNum, uint32_t inode_id, uint8_t type, char
name[121]);

uint32_t findFolderOrFile(uint32_t curDirInode, char name[121], int type);

int touch(char *dir);

int mkdir(char *folderName);

int ls(char *dir);

int cp(char *source, char *target);

void shutdown();
/**
 *
 * @param inodeNumber
 * @return 返回第inodeNumber个inode，在哪个block
 */
uint32_t getBlockNum(uint32_t inodeNumber);

/**
 *
 * @param inodeNumber
```

```
 * @return 返回第inodeNumber个inode，在某个block中的第几个
 */
uint32_t getInodeNum(uint32_t inodeNumber);

/**
 *
 * @param blockNum 在第几块block
 * @param inodeNumInBlock 在块内的第几个inode
 * @return 总的第几个inode
 */
uint32_t getTotalInodeNum(uint32_t blockNum, uint32_t inodeNumInBlock);
```

测试模块主要测试了初始化超级块、字符串分割、创建一个inode、创建一个diritem、对于位操作的正确性。

因为实验要求的指令都是以上述功能为基础的，因此需要保证上述功能的正确性。

```
void testInitSP();

void testSep(char *dir);

void testCreateInode();

void testCreateDirItem();

void testBitSet();
```

感想

本来打算是在`windows`上写的，因为我认为我写的这些代码，在`linux`以及`windows`平台上都有相应的库函数，并没有用到一些`linux`独有的函数。但是在实验的过程中，出现了一下的现象：

第五次调用`printIN`,创建第五个`inode`时,出现磁盘读写错误

出错路径为:

```
createInode -> disk_read_whole_block -> disk_read -> disk_read_block
    -> fread
```

接下来继续调用`disk_read_whole_block`中的第二个`disk_read`，出现`open_disk`错误，`disk!=0`导致出错。但不应该出现这种错误，在上一次调用`disk_read`的时候，已经关闭了磁盘访问了。

在我自己查看了自己的代码之后并没有发现错误。后来我去问了助教，发现在助教的`macbook pro`平台上可以正常运行，并不会出现读写错误。后来我在我自己的`linux`笔记本上运行，也没有发生错误。这也让我非常迷惑。但由于时间原因，我并没有继续深究，而将写代码的平台转向了`linux`。

在完成实验后，我再次尝试了在`windows`平台上运行，是可以正常运行的。但是格式化输出有点丑，因为时间原因也不想再改了。

本次实验是我进入大学以来写的最酣畅淋漓的实验，想要形容这种感觉的话，做其他的实验，感觉自己是在一个迷宫里面，每走一步都会发现一些新的东西，学到一些新的东西。这样的结果固然好，但是越到后面会发现自己之前写的代码有问题，有很多本来可以这样做，但是却那样做了的事。在这个时候，最恰当的事应该是重构代码，但通常因为课程时间压力，学生(就是我)，根本没有时间去思考重构代码的

问题。然后当我发现一个之前写的bug，我并不能直接去改他，我只能通过增加一个函数，来修正我的结果。就导致代码量越来越大，做的事却只有一点点。以上都是我编译原理实验的感想。

而这次操作系统实验，可以说是在编译原理实验、计网实验、之前的几个操作系统实验、软件工程课的锻炼下写出来的。这就不是在迷宫里了，是一个自顶向下、自己设计实现一个软件的过程。在搞清楚`Ext2`文件系统的原理以及实验要求后，我就可以模拟一下场景，用软件工程的话来说就是提出需求；从而可以设计自己需要怎么样的函数。由于C语言是面向过程的，因此这里也没有一些面向对象的设计了。

当提出了文件系统需要的函数之后，就可以设计他们之间的接口：需要通过什么样的数据结构，传入什么样的值，返回什么样的值。在明确了以上内容之后，编写代码的思路就非常清晰了。在完成一些重要的函数之后，还需要进行测试。比如创建`inode`、`dirItem`，以及对位图的操作。而且必须要保证单元测试是正确的，才能使用这些函数。否则会导致`debug`非常的困难。

简明用户手册

如果使用`clion`运行，则只需打开该工程项目，在`File`选项栏中选择`Reload cmake project`即可在`linux`,`macOs`,`windows`平台上运行。

如果使用`shell`运行，则进入`cmake-build-debug/`文件夹中，终端输入`./Ext_FS`即可运行。

进入界面后，系统提示是否格式化磁盘。

```
Hello, World! Do you want do reformat the disk? [y/n]
```

选择`y`则会进入一个新的文件系统。

该文件系统可以正常使用`ls,touch,mkdir,cp`四个常见的`linux`命令，以及一个`shutdown`关机命令。

使用示例如下：

进入系统并创建一些文件和文件夹

```
Hello, World! Do you want do reformat the disk? [y/n]
y
> ls
name                    type            inode_id
> touch tx
> touch wx
> mkdir qq
> mkdir f1
> ls
name                    type            inode_id
tx                      File            3
wx                      File            4
qq                      Folder          5
f1                      Folder          6
```

在子目录中创建文件

```
> touch qq/file1
> touch qq/file2
> mkdir qq/folder
> ls qq/
name                    type          inode_id
file1                   File          7
file2                   File          8
folder                  Folder        9
```

拷贝文件示例

```
> cp wx f1/wx_copyed
> ls f1/
name                     type          inode_id
wx_copyed                File          10

> cp f1/wx_copyed qq/wx_d
> ls qq/
name                     type          inode_id
file1                    File          7
file2                    File          8
folder                   Folder        9
wx_d                     File          11
```

注意

在使用`ls`命令查看子目录的文件、文件夹时，必须在路径末尾加上`/`，如`ls f1/`，`ls f1/f2/f3/`，而不能直接使用`ls f1/f2`。