



# 操作系统原理实验报告

## 实验十：文件系统

数据科学与计算机学院 17大数据与人工智能

17341015 陈鸿峰

### 一、实验目的

- 学习文件操作的方法，掌握其实现方法。
- 利用FAT12文件系统实现原型操作系统的文件管理控制命令。
- 扩展MyOS的系统调用，实现进程中文件创建、文件打开、文件读、文件写和文件关闭等。

### 二、实验要求

原型保留原有特征的基础上，设计满足下列要求的新原型操作系统：

1. 参考DOS的命令功能，实现文件管理控制命令`cd`、`dir`、`del`、`copy`等。
2. 扩展MyOS的系统调用，在C语言中，程序可以调用文件创建、文件打开、文件读、文件写和文件关闭等系统调用。
3. 编写一个测试的这些系统调用的C语言用户程序，编译产生可执行程序进行测试。

### 三、实验环境

具体环境选择原因已在实验一报告中说明。

- Windows 10系统 + Ubuntu 18.04(LTS)子系统
- gcc 7.3.0 + nasm 2.13.02 + GNU ld (Binutils) 2.3.0
- GNU Make 4.1
- Oracle VM VirtualBox 6.0.6
- Bochs 2.6.9
- Sublime Text 3 + Visual Studio Code 1.33.1

虚拟机配置：内存4M，1.44M虚拟软盘引导，1.44M虚拟硬盘。

### 四、实验方案

本次实验继续沿用实验六保护模式的操作系统。用户程序都以ELF文件格式读入，并且在用户态(ring 3)下执行。

## 1. FAT12文件系统

由于虚拟出来的硬盘容量较小，且没有那么多数据需要管理，故本实验采用FAT12文件系统。

### (i) 基本理论

针对1.44M的虚拟软盘/硬盘，FAT12将其划分为如图1所示的几个部分。

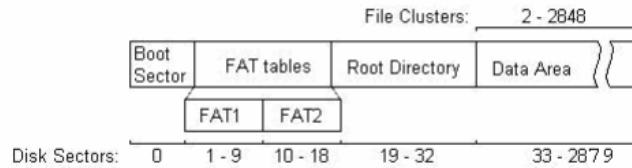


图 1: FAT12磁盘划分<sup>1</sup>

- 引导程序区：0-1扇区
- 文件分配表(File Allocation Table)1：1-9扇区
- 文件分配表2：10-18扇区
- 根目录：19-32扇区
- 文件数据区：33扇区以后

注意上述所指的都是逻辑扇区。

对于引导程序区，我用C的结构体封装了其各个字节的语义，如下。

```
// Boot Sector (BS) & BPB (BIOS Parameter Block)
typedef struct bootsect
{
    uint8_t      Jump[3]; // 3
    char         BS_OEMName[8]; // 11
    uint16_t     BPB_BytesPerSector; // 13
    uint8_t      BPB_SectorsPerCluster; // 14
    uint16_t     BPB_ReservedSectors; // 16
    uint8_t      BPB_NumFATs; // 17
    uint16_t     BPB_RootDirectoryEntries; // 19
    uint16_t     BPB_LogicalSectors; // 21
    uint8_t      BPB_MediumDescriptorByte; // 22
    uint16_t     BPB_SectorsPerFat; // 24
    uint16_t     BPB_SectorsPerTrack; // 26
    uint16_t     BPB_NumHeads; // 28
    uint32_t     BPB_HiddenSectors; // 32
    uint8_t      code[480]; // the last 2B is signature "AA55h"
} __attribute__((packed)) bootsect_t;
```

文件分配表即所谓的FAT，是一个数组模拟的链表，存放文件的组织方式。由于用12位二进制位表示一个簇(cluster)的下标<sup>2</sup>，故称为FAT12文件系统。如图2所示，由于FAT12涉及到半字节的处理，故会比较麻烦。通常是3字节3字节进行处理，图2的上图是物理FAT结构，下图是逻辑FAT结构，需要经过映射才得到真正的FAT。

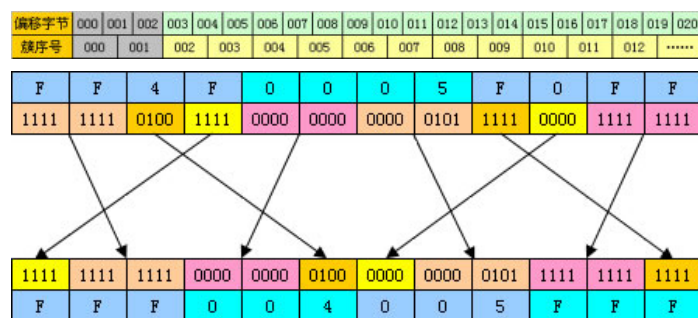


图 2: 文件分配表<sup>3</sup>

前面三个字节F0 FF FF是存储介质和标识符，故实际的文件存储从2号簇开始。FAT数组内存放的内容指向下一个簇的下标，如果是0F00-0FF7则代表坏簇（文件结尾）。

如一个文件1000B，需要占用2个簇/扇区，则一种可能的方式是

Index	0	1	2	3
FAT	FF0	FFF	003	FF7

在C头文件中我分别创建了物理和逻辑两个不同的结构体

```
typedef struct phys_fat
{
    uint8_t entry [FAT_PHYS_SIZE * FAT_SECTOR_SIZE]; // 4608
} __attribute__((packed)) phys_fat12_t;
static phys_fat12_t phys_fat;

typedef struct logic_fat
{
    // only the first 3B are used (24bits)
    // the first two clusters (0,1) are useless
    uint32_t entry [FAT_NUM_ENTRY]; // 3072
} __attribute__((packed)) fat12_t;
static fat12_t fat;
```

接着是文件目录项，如图3所示。

<sup>2</sup>FAT12中一个簇与一个扇区大小相同

图 3: FAT12文件目录项<sup>4</sup>

同样可以创建对应的结构体FileEntry，由于图3已经十分清晰，这里不再赘述。但注意日期、时间与整型的转换，我在fat12.h头文件中创建了四个辅助函数来帮助转换。

最后是根目录，一共有224项。

```
typedef struct root_directory
{
    FileEntry_t entry[FAT_SECTOR_SIZE / sizeof(FileEntry_t) * FAT_ROOT_SIZE]; //
    ↪ 224
} __attribute__((packed)) fat_root_t;
static fat_root_t root_dir;
```

在FAT12文件系统中，目录与文件都是一个entry，只是属性不同，都至少占用一个簇。而子目录将存储在数据区中，一个簇最多存储16项（因每个项占32B）。

下面将详细叙述具体的文件操作指令实施，由于是Linux环境，故对应的都是Linux的命令行语句的实施。

## (ii) 文件系统初始化

按照上述四个部分次序依次将FAT12的头部读入内存，同时设置了当前目录/簇curr\_dir和当前路径curr\_path，方便索引。由于一开始在根目录，故初始化curr\_path为"/"。

```
bool fat12_init()
{
    // Boot sector
    read_sectors((uintptr_t)&bootsector, FAT_BOOTSECTOR_START, FAT_BOOTSECTOR_SIZE);

    // FAT sectors
    read_sectors((uintptr_t)&phys_fat, FAT_ENTRY_START, FAT_PHYS_SIZE);

    // Converts the FAT into the logical structures (array of word).
    for (int i = 0, j = 0; i < 4608; i += 3)
    {
```

```

        fat.entry[ j++ ] = (phys_fat.entry[i] + (phys_fat.entry[i+1] << 8)) & 0
        ↪ x0FFF;
        fat.entry[ j++ ] = (phys_fat.entry[i+1] + (phys_fat.entry[i+2] << 8)) >> 4;
    }

    // root directory
    read_sectors((uintptr_t)&root_dir,FAT_ROOT_REGION_START,FAT_ROOT_SIZE);

    // set current path
    curr_dir = FAT_ROOT_REGION_START;
    strcpy(curr_path,"/");

    return true;
}

```

### (iii) 目录查看

目录查看对应的是Linux中的ls指令。

首先实施fat12\_show\_file\_entry, 将对应的FileEntry\_t\* f输出。由于主要是字符串输出操作, 故这里不再展示, 但注意只有文件和目录需要输出, 对于删除的文件name首字节为0xE5的也不需要展示。

由于根目录与其他子目录大小不同, 故需要分别处理, 下面的操作也是类似, 需要注意。

```

void fat12_ls()
{
    if (curr_dir == FAT_ROOT_REGION_START)
        for (int i = 0; i < FAT_NUM_ROOT_ENTRY; ++i)
            fat12_show_file_entry(&root_dir.entry[i]);
    else
        for (int i = 0; i < FAT_NUM_DIR_ENTRY; ++i)
            fat12_show_file_entry(&subdir.entry[i]);
}

```

### (iv) 目录切换

目录切换对应的是Linux中的cd指令。

首先需要实施curr\_dir和curr\_dir的重赋值。

```

void fat12_set_dir(int new_dir, char* action)
{
    if (curr_dir == FAT_ROOT_REGION_START){
        curr_dir = new_dir;
        strcpy(curr_path,"/");
        strcat(curr_path,action);
    } else {

```

```

    if (new_dir == 0)
        new_dir = FAT_ROOT_REGION_START;
    curr_dir = new_dir;

    if (strcmp(action, ".") == 0)
        ; // do nothing
    else if (strcmp(action, "..") == 0) {
        reverse(curr_path);
        char* tmp_path = curr_path;
        strsep(&tmp_path, "/");
        reverse(tmp_path);
        strcpy(curr_path, tmp_path);
        if (curr_dir == FAT_ROOT_REGION_START)
            strcpy(curr_path, "/");
    } else {
        strcat(curr_path, "/");
        strcat(curr_path, action);
    }
}
}

```

然后找到对应的目录项，并将该目录项对应的簇读入内存。注意.和..在FAT文件系统中也对应着一个项，即与其他文件/文件夹没有区别。

```

bool fat12_cd(char* folder)
{
    FileEntry_t* fe = fat12_find_entry(folder);
    if (fe == NULL){
        put_error("No such file or directory!");
        return false;
    }
    fat12_set_dir(fe->startCluster, folder);
    fat12_read_clusters((char*)&subdir, fe->startCluster);
    return true;
}

```

#### (v) 文件读入

这是执行用户程序的关键，对每一个目录项的名字都进行匹配，如果匹配上则读取文件内容。由于前两个簇不适用，故簇号到实际的硬盘逻辑扇区号需要经过 $\text{clust} + \text{FAT\_DATA\_REGION\_START} - 2$ 进行转换。

```

// start from the @cstart-th cluster, read the whole file to @buf
// return the number of bytes read
int fat12_read_clusters(char* buf, uint32_t cstart)

```

```

{
    int i;
    uint32_t clust = cstart;
    for (i = 0; true; ++i)
    {
        uintptr_t addr = (uintptr_t) buf + i * FAT_SECTOR_SIZE;
        // remember to recalculate cstart
        read_sectors(addr, clust + FAT_DATA_REGION_START - 2, 1);
        enable();
        if (fat12_next_sector(&clust, clust) == false)
            break;
    }
    return i;
}

bool fat12_read_file(char* filename, char* addr, int* size)
{
    for (int i = 0; i < FAT_NUM_ROOT_ENTRY; ++i){
        char entry_name[13]; // with dot(.)!
        fat12_construct_file_name(entry_name, &root_dir.entry[i]);
        if (strcmp(filename, entry_name) != 0)
            continue;
        fat12_read_clusters(addr, root_dir.entry[i].startCluster);
        if (size != NULL)
            *size = root_dir.entry[i].fileLength;
        return true;
    }

    put_error("FAT No this file!");

    return false;
}

```

#### (vi) 文件删除

删除操作对应Linux中的rm指令。

删除的操作较为麻烦，虽然只需将目录项的name第0个字节设为0xE5即可，但需要重新写回磁盘。同时，FAT对应的簇也应改为坏簇，也要写回磁盘

```

bool fat12_rm(char* filename)
{
    FileEntry_t* fe = fat12_find_entry(filename);
    if (fe == NULL){
        put_error("No such file or directory!");
    }
}

```

```

        return false;
    }
    fe->name[0] = 0x000000E5;
    if (curr_dir == FAT_ROOT_REGION_START)
        write_sectors((uintptr_t)&root_dir, FAT_ROOT_REGION_START, FAT_ROOT_SIZE);
    else
        fat12_write_clusters((char*)&subdir, curr_dir);
    fat12_del_fat_entry(fe->startCluster);
    return true;
}

```

### (vii) 文件创建

文件的创建则较为繁琐，按照如下流程：

1. 计算需要多少个簇，记为numCluster
2. 从FAT中寻找numCluster个未用的簇，同时记录下头地址，并创建起整个链表串
3. 边创建FAT也要边将文件对应的内容写入对应的簇/扇区中
4. 最后修改目录项，将文件信息填入

碍于时间关系，这里修改过的内容都会被直接写入硬盘，尽管效率不是很高。

```

void fat12_create_file(uintptr_t addr, int size, char* name)
{
    int numCluster = (size % FAT_SECTOR_SIZE == 0 ?
        size / FAT_SECTOR_SIZE :
        size / FAT_SECTOR_SIZE + 1);

    int cnt = 0, prevIndex = -1, cstart = 0;
    for (int i = 2; i < FAT_NUM_ENTRY; ++i) // do NOT start from 0!
        if (fat.entry[i] == 0){
            if (cnt == 0)
                cstart = i;
            if (prevIndex != -1)
                fat.entry[prevIndex] = i;
            prevIndex = i;
            write_sectors(addr+i*FAT_SECTOR_SIZE, i+FAT_DATA_REGION_START-2, 1);
            cnt++;
            if (cnt == numCluster){
                fat.entry[i] = 0xFFFF;
                break;
            }
        }
    fat12_write_back_fat();

    for (int i = 0; i < FAT_NUM_ROOT_ENTRY; ++i)

```



```

    if (root_dir.entry[i].name[0] == 0x00000000){
        char tmp_name[20];
        strcpy(tmp_name,name);
        char* ext = tmp_name;
        strsep(&ext,".");
        strcpy(root_dir.entry[i].name,tmp_name);
        for (int j = strlen(root_dir.entry[i].name); j < 8; ++j)
            root_dir.entry[i].name[j] = ' ';
        strcpy(root_dir.entry[i].extension,ext);
        for (int j = strlen(root_dir.entry[i].extension); j < 3; ++j)
            root_dir.entry[i].extension[j] = ' ';
        root_dir.entry[i].attribute.archive = true;
        root_dir.entry[i].startCluster = cstart;
        root_dir.entry[i].fileLength = size;
        root_dir.entry[i].date = date_to_int(2019,6,23);
        root_dir.entry[i].time = time_to_int(0,0,0);
        break;
    }
    write_sectors((uintptr_t)&root_dir,FAT_ROOT_REGION_START,FAT_ROOT_SIZE);
}

```

#### (viii) 文件复制

复制对应着Linux的cp指令。

有了前面文件创建的基础，文件复制实施起来就很简单了。只需读取文件，然后拷贝一份写入即可。

```

bool fat12_cp(char* src, char* dst)
{
    char buf[512 * 30];
    int size = 0;
    if (fat12_read_file(src,buf,&size)){
        fat12_create_file((uintptr_t)buf,size,dst);
        return true;
    }
    return false;
}

```

## 2. 虚拟文件系统

这一部分是文件系统更高层的抽象管理，更加方便C用户程序的调用。这里文件系统采用int 0x82号中断，已经做了API的封装。

对于文件管理，我创建了如下的结构体，同时创建了一个维护文件的全局数组file\_list。

```
typedef struct _FILE {

    char        name[32];
    char        mode[4];
    char        buffer[1024];
    char*       pos;
    uint32_t    fileLength;
    uint32_t    startCluster;
    uint32_t    currentCluster;

}FILE, *PFILE;

static FILE file_list[MAX_FILE_NUM];
```

### (i) 文件打开与关闭

文件打开首先要判断文件是否存在，如果文件不存在，且为写模式，则创建新文件。然后将文件对应的簇读入到文件结构体的缓冲数组中。

为避免频繁的硬盘读写，故文件内容都是统一读入buffer进行处理，然后在文件关闭时再将buffer内容写回硬盘。

```
FILE* do_fopen(const char *pname, const char *mode)
{
    FILE* file = NULL;
    for (int i = 0; i < MAX_FILE_NUM; ++i)
        if (strcmp(file_list[i].mode, "u") == 0){
            file = &file_list[i];
            break;
        }
    if (file == NULL)
        return NULL;
    strcpy(file->name, pname);
    FileEntry_t* fe = fat12_find_entry((char*)pname);
    if (fe == NULL && strcmp(mode, "w") == 0){
        strcpy(file->mode, mode);
        return file;
    } else if (fe == NULL && strcmp(mode, "r") == 0)
        return NULL;
    file->fileLength = fe->fileLength;
    file->currentCluster = file->startCluster = fe->startCluster;
    strcpy(file->mode, mode);
    fat12_read_clusters(file->buffer, file->startCluster);
    return file;
}
```

```

int do_fclose(FILE* fp)
{
    if (strcmp(fp->mode,"w") == 0)
        fat12_create_file((uintptr_t)fp->buffer,fp->pos-fp->buffer,fp->name);
    strcpy(fp->mode,"u");
    enable();
    return 0;
}

```

## (ii) 文件读写

我实现了以下几个函数，都比较简单，主要是内存的操作。

```

int do_fread(void *buf, int size, int count, FILE *fp)
{
    int numBytes = size * count;
    memcpy((char*)buf,fp->buffer,numBytes);
    return count;
}

int do_fwrite(void *buf, int size, int count, FILE *fp)
{
    if (strcmp(fp->mode,"w") != 0)
        return 0;
    int numBytes = size * count;
    memcpy(fp->pos,(const char*)buf,numBytes);
    fp->pos = fp->pos + numBytes;
    return count;
}

char *do_fgets(char *str, int count, FILE *fp)
{
    memcpy((char*)str,fp->buffer,count);
    return str;
}

int do_fputs(const char *str, FILE *fp)
{
    if (strcmp(fp->mode,"w") != 0)
        return 0;
    memcpy(fp->pos,str,strlen(str));
    fp->pos = fp->pos + strlen(str);
    return 1;
}

```

五、实验结果

我将所有的用户程序都放入创建的虚拟硬盘中<sup>5</sup>，并创建了两个测试目录，如图4所示。

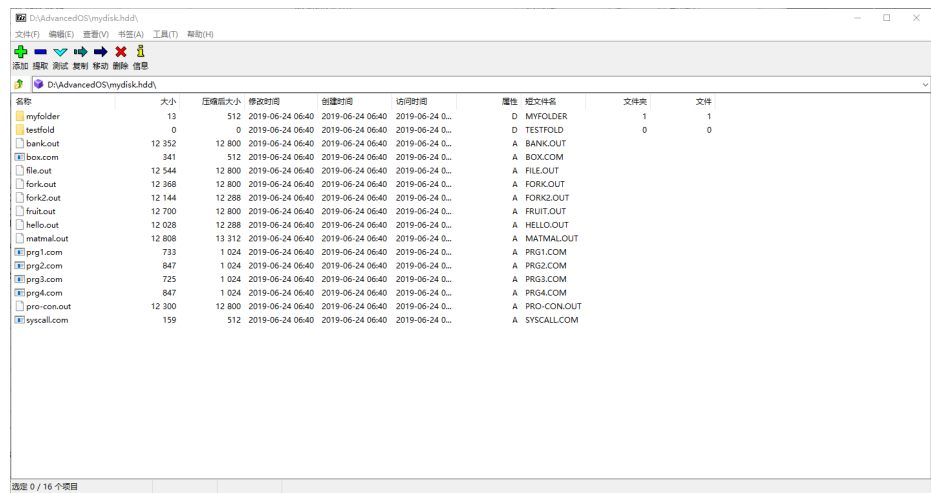
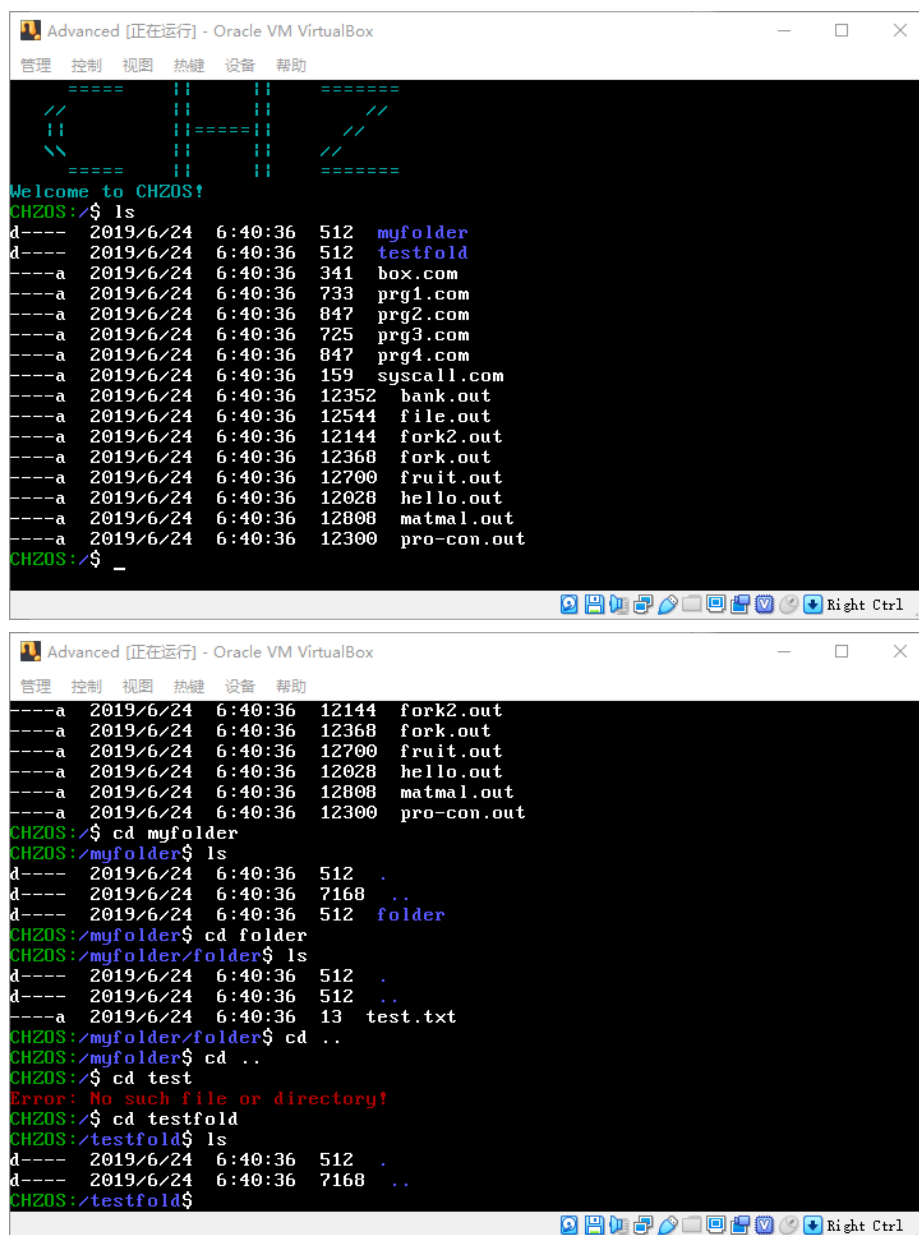


图 4: 用7ZIP查看虚拟硬盘内容

从图5中可以看到，随着目录的切换，前面的文件夹显示也会跟着改变。向下的文件夹索引及向上的..文件夹索引也都没有问题。同时，想要访问不存在的文件夹时会及时报错。

<sup>5</sup>利用Linux的mount语句可以把虚拟硬盘挂载后，然后很方便地执行文件操作。



```
Advanced [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助

=====
//      ||      ||      //
||      ||=====||
\\      ||      ||      //
=====

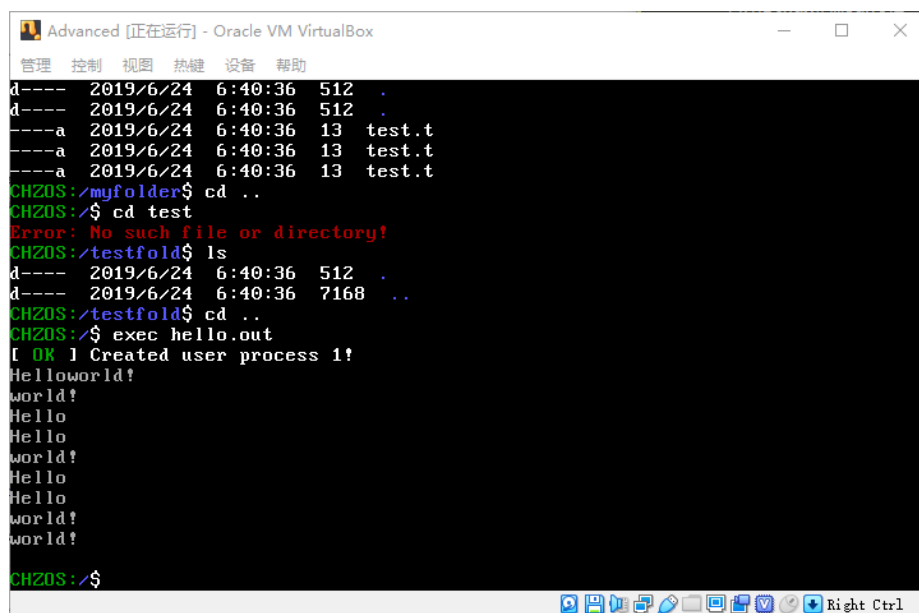
Welcome to CHZOS!
CHZOS:/$ ls
d---- 2019/6/24 6:40:36 512 myfolder
d---- 2019/6/24 6:40:36 512 testfold
----a 2019/6/24 6:40:36 341 box.com
----a 2019/6/24 6:40:36 733 prg1.com
----a 2019/6/24 6:40:36 847 prg2.com
----a 2019/6/24 6:40:36 725 prg3.com
----a 2019/6/24 6:40:36 847 prg4.com
----a 2019/6/24 6:40:36 159 syscall.com
----a 2019/6/24 6:40:36 12352 bank.out
----a 2019/6/24 6:40:36 12544 file.out
----a 2019/6/24 6:40:36 12144 fork2.out
----a 2019/6/24 6:40:36 12368 fork.out
----a 2019/6/24 6:40:36 12700 fruit.out
----a 2019/6/24 6:40:36 12028 hello.out
----a 2019/6/24 6:40:36 12808 matmal.out
----a 2019/6/24 6:40:36 12300 pro-con.out
CHZOS:/$ _

Advanced [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助

----a 2019/6/24 6:40:36 12144 fork2.out
----a 2019/6/24 6:40:36 12368 fork.out
----a 2019/6/24 6:40:36 12700 fruit.out
----a 2019/6/24 6:40:36 12028 hello.out
----a 2019/6/24 6:40:36 12808 matmal.out
----a 2019/6/24 6:40:36 12300 pro-con.out
CHZOS:/$ cd myfolder
CHZOS:/myfolder$ ls
d---- 2019/6/24 6:40:36 512 .
d---- 2019/6/24 6:40:36 7168 ..
d---- 2019/6/24 6:40:36 512 folder
CHZOS:/myfolder$ cd folder
CHZOS:/myfolder/folder$ ls
d---- 2019/6/24 6:40:36 512 .
d---- 2019/6/24 6:40:36 512 ..
----a 2019/6/24 6:40:36 13 test.txt
CHZOS:/myfolder/folder$ cd ..
CHZOS:/myfolder$ cd ..
CHZOS:/$ cd test
Error: No such file or directory!
CHZOS:/$ cd testfold
CHZOS:/testfold$ ls
d---- 2019/6/24 6:40:36 512 .
d---- 2019/6/24 6:40:36 7168 ..
CHZOS:/testfold$
```

图 5: 目录切换与目录查看

图6展示了上一次实验的多线程程序能够正常执行。这里涉及到FAT12的文件索引、多簇的文件读入、ELF文件的解析、进程的创建与执行、多线程管理等多个内容，非常复杂，但我的操作系统依然能够正常运行！



```

d----- 2019/6/24  6:40:36    512  .
d----- 2019/6/24  6:40:36    512  .
-a----- 2019/6/24  6:40:36      13  test.t
-a----- 2019/6/24  6:40:36      13  test.t
-a----- 2019/6/24  6:40:36      13  test.t
CHZOS:/myfolder$ cd ..
CHZOS:/$ cd test
Error: No such file or directory!
CHZOS:/testfold$ ls
d----- 2019/6/24  6:40:36    512  .
d----- 2019/6/24  6:40:36   7168  ..
CHZOS:/testfold$ cd ..
CHZOS:/$ exec hello.out
[ OK ] Created user process 1!
Hello world!
world!
Hello
Hello
Hello
world!
Hello
Hello
Hello
world!
world!
CHZOS:/$

```

图 6: 正常加载并执行多线程程序

下面的用户程序file.c将测试文件的基本功能。先创建一个文件testmsg.txt，然后往里写内容，最后读出并关闭文件。

```

#include "stdio.h"
#include "file.h"

int main()
{
    // Declare the file pointer
    FILE* fp;

    // Get the data to be written in file
    char dataToBeWritten[50] = "This is a test message!";

    // Open the non-existing file with fopen()
    fp = fopen("testmsg.txt", "w");

    // Check if this fp is null
    // which maybe if the file does not exist
    if (fp == NULL)
    {
        printf("testmsg.txt file failed to open!");
    }
    else
    {
        printf("The file is now opened.\n");
    }
}

```

```

// Write the dataToBeWritten into the file
if (strlen(dataToBeWritten) > 0)
{
    // writing in the file using fputs()
    fputs(dataToBeWritten, fp);
    // fputs("\n", fp);
}

char dataToBeRead[50];
fgets(dataToBeRead, strlen(dataToBeWritten), fp);
dataToBeRead[strlen(dataToBeWritten)] = '\0';

printf("==== Data read out =====\n\n");
printf("%s\n\n", dataToBeRead);
printf("=====\n\n");

// Closing the file using fclose()
fclose(fp);

printf("Data successfully written in file testmsg.txt\n");
printf("The file is now closed.\n");
}
return 0;
}

```

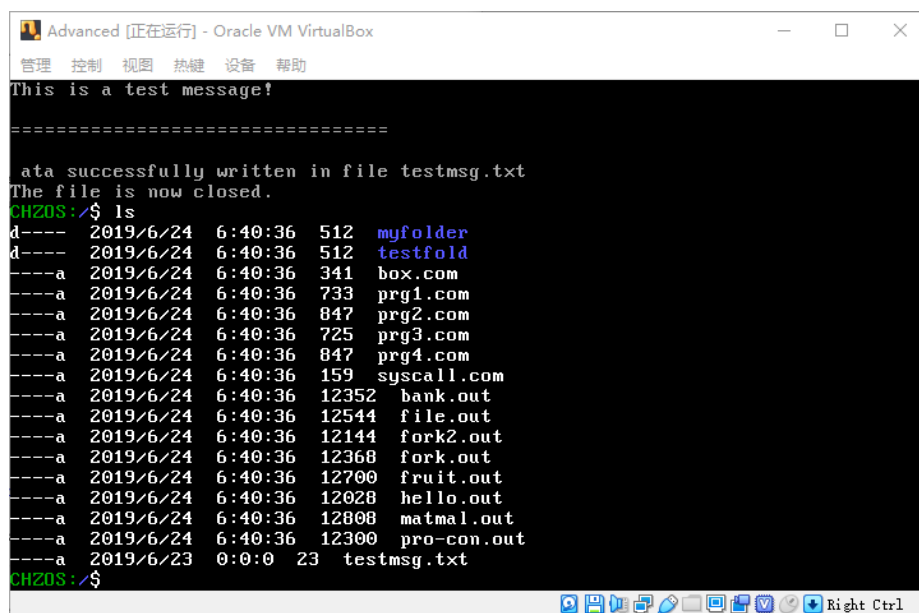
```

Advanced [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
CHZOS:/testfold$ cd ..
CHZOS:/testfold$ exec hello.out
[ OK ] Created user process 1!
Helloworld!
world!
Hello
Hello
Hello
world!
Hello
Hello
world!
world!

CHZOS:/testfold$ exec file.out
[ OK ] Created user process 4!
The file is now opened.
==== Data read out =====
This is a test message!
=====
Data successfully written in file testmsg.txt
The file is now closed.
CHZOS:/testfold$

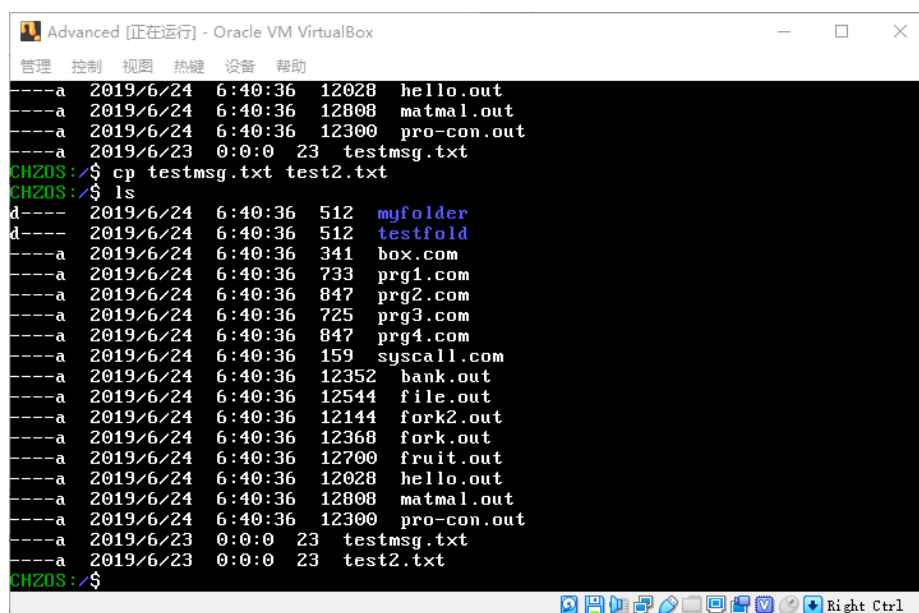
```

图 7: C程序输出结果, 注意Data read out那一段是文本写入文件后再读出来的, 可以看到结果正确



```
Advanced [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
This is a test message!
=====
ata successfully written in file testmsg.txt
The file is now closed.
CHZOS:/$ ls
d---- 2019/6/24 6:40:36 512 myfolder
d---- 2019/6/24 6:40:36 512 testfold
----a 2019/6/24 6:40:36 341 box.com
----a 2019/6/24 6:40:36 733 prg1.com
----a 2019/6/24 6:40:36 847 prg2.com
----a 2019/6/24 6:40:36 725 prg3.com
----a 2019/6/24 6:40:36 847 prg4.com
----a 2019/6/24 6:40:36 159 syscall.com
----a 2019/6/24 6:40:36 12352 bank.out
----a 2019/6/24 6:40:36 12544 file.out
----a 2019/6/24 6:40:36 12144 fork2.out
----a 2019/6/24 6:40:36 12368 fork.out
----a 2019/6/24 6:40:36 12700 fruit.out
----a 2019/6/24 6:40:36 12028 hello.out
----a 2019/6/24 6:40:36 12808 matmal.out
----a 2019/6/24 6:40:36 12300 pro-con.out
----a 2019/6/23 0:0:0 23 testmsg.txt
CHZOS:/$
```

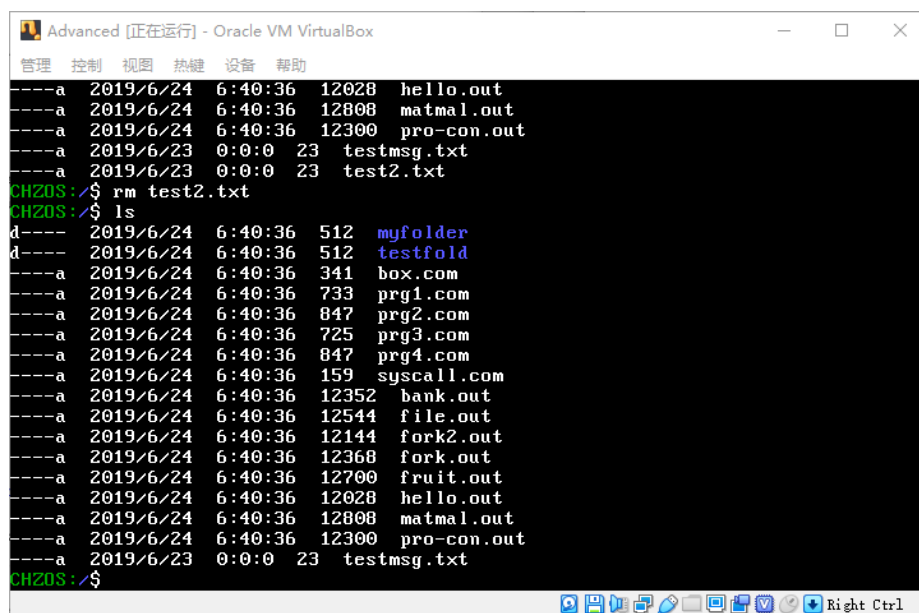
图 8: 程序结束后查看目录, 确实存在新的文件testmsg.txt, 由于没有外部时钟的支持, 故这里我给文件创建设置的时间会存在不准确, 但不影响整体的正确性



```
Advanced [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
----a 2019/6/24 6:40:36 12028 hello.out
----a 2019/6/24 6:40:36 12808 matmal.out
----a 2019/6/24 6:40:36 12300 pro-con.out
----a 2019/6/23 0:0:0 23 testmsg.txt
CHZOS:/$ cp testmsg.txt test2.txt
CHZOS:/$ ls
d---- 2019/6/24 6:40:36 512 myfolder
d---- 2019/6/24 6:40:36 512 testfold
----a 2019/6/24 6:40:36 341 box.com
----a 2019/6/24 6:40:36 733 prg1.com
----a 2019/6/24 6:40:36 847 prg2.com
----a 2019/6/24 6:40:36 725 prg3.com
----a 2019/6/24 6:40:36 847 prg4.com
----a 2019/6/24 6:40:36 159 syscall.com
----a 2019/6/24 6:40:36 12352 bank.out
----a 2019/6/24 6:40:36 12544 file.out
----a 2019/6/24 6:40:36 12144 fork2.out
----a 2019/6/24 6:40:36 12368 fork.out
----a 2019/6/24 6:40:36 12700 fruit.out
----a 2019/6/24 6:40:36 12028 hello.out
----a 2019/6/24 6:40:36 12808 matmal.out
----a 2019/6/24 6:40:36 12300 pro-con.out
----a 2019/6/23 0:0:0 23 testmsg.txt
----a 2019/6/23 0:0:0 23 test2.txt
CHZOS:/$
```

图 9: 拷贝一份文件test2.txt, 可以看见文件属性相同, 大小也相同





```

Advanced [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
----a 2019/6/24 6:40:36 12028 hello.out
----a 2019/6/24 6:40:36 12808 matmal.out
----a 2019/6/24 6:40:36 12300 pro-con.out
----a 2019/6/23 0:0:0 23 testmsg.txt
----a 2019/6/23 0:0:0 23 test2.txt
CHZOS:/$ rm test2.txt
CHZOS:/$ ls
d---- 2019/6/24 6:40:36 512 myfolder
d---- 2019/6/24 6:40:36 512 testfold
----a 2019/6/24 6:40:36 341 box.com
----a 2019/6/24 6:40:36 733 prg1.com
----a 2019/6/24 6:40:36 847 prg2.com
----a 2019/6/24 6:40:36 725 prg3.com
----a 2019/6/24 6:40:36 847 prg4.com
----a 2019/6/24 6:40:36 159 syscall.com
----a 2019/6/24 6:40:36 12352 bank.out
----a 2019/6/24 6:40:36 12544 file.out
----a 2019/6/24 6:40:36 12144 fork2.out
----a 2019/6/24 6:40:36 12368 fork.out
----a 2019/6/24 6:40:36 12700 fruit.out
----a 2019/6/24 6:40:36 12028 hello.out
----a 2019/6/24 6:40:36 12808 matmal.out
----a 2019/6/24 6:40:36 12300 pro-con.out
----a 2019/6/23 0:0:0 23 testmsg.txt
CHZOS:/$

```

图 10: 删除该拷贝test2.txt，文件确实从目录中消失了

上述实验已经将我实施的所有文件操作演示了一遍，均取得了预期的结果。

## 六、实验总结

单是一个文件系统就增加了我操作系统近2000行代码，足以见得文件系统的工程量之大。而且网上几乎没有合适的参考资料，因为每个操作系统的底层都不一样，对文件系统的实施也就完全不一样。其实文件系统并不难，按部就班进行实施就好了。但有很多细节需要注意，一不小心操作系统很可能就会崩溃，毕竟所有的文件读写、执行都是建立在文件系统之上的。

由于Windows 10的子系统WSL不支持FAT12的mount操作，故我只能将我的操作系统放上云服务器上进行调试，来来回回折腾非常耗费时间，也大大延长了我的调试进度。不过好在用户程序不太需要进行修改，故只需编译一次，写入虚拟硬盘一次即可（这也是将具体实施与用户API相分离的好处）。

由于时间所限，本次实验的文件系统只实现了一些基本的功能，且还不够完善，还有很多细节未考虑，也还有很多后续工作没完成。但不管怎样，我的操作系统也算是完成了大部分的功能，差不多要给它画上尾声了。

## 七、参考资料

操作系统实现完整资料：

1. OS Development Series, <http://www.broken Thorn.com/Resources/OSDevIndex.html>
2. Roll your own toy UNIX-clone OS, [http://www.jamesmolloy.co.uk/tutorial\\_html/](http://www.jamesmolloy.co.uk/tutorial_html/)
3. The little book about OS development, <http://littleosbook.github.io/>

4. Writing a Simple Operating System from Scratch, [http://www.cs.bham.ac.uk/~exr/lectures/opsys/10\\_11/lectures/os-dev.pdf](http://www.cs.bham.ac.uk/~exr/lectures/opsys/10_11/lectures/os-dev.pdf)
5. Intel® 64 and IA-32 Architectures Software Developer's Manual
6. UCore OS Lab, [https://github.com/chyyuu/ucore\\_os\\_lab](https://github.com/chyyuu/ucore_os_lab)
7. CMU CS 15-410, Operating System Design and Implementation, <https://www.cs.cmu.edu/~410/>
8. 李忠, 王晓波, 余洁, 《x86汇编语言-从实模式到保护模式》, 电子工业出版社, 2013

#### 本次实验参考资料:

1. FAT12文件系统之引导扇区结构, [http://blog.sina.com.cn/s/blog\\_3edcf6b80100cr08.html](http://blog.sina.com.cn/s/blog_3edcf6b80100cr08.html)
2. FAT12文件系统之数据存储方式详解, [http://blog.sina.com.cn/s/blog\\_3edcf6b80100crz1.html](http://blog.sina.com.cn/s/blog_3edcf6b80100crz1.html)
3. An Overview of FAT12, <http://www.disc.ua.es/~gil/FAT12Description.pdf>
4. FAT12实施, <https://github.com/imtypist/fat12>

## 附录 A. 程序清单

### 1. 内核核心代码

序号	文件	描述
1	bootloader.asm	主引导程序
2	kernel_entry.asm	内核汇编入口程序
3	kernel.c	内核C入口程序
4	Makefile	自动编译指令文件
5	bootflpy.img	引导程序/内核软盘
6	mydisk.hdd	虚拟硬盘
7	bochsrc.bxrc	Bochs配置文件

## 2. 内核头文件

序号	文件	描述
1	disk_load.inc	BIOS读取磁盘
2	show.inc	常用汇编字符显示
3	gdt.inc	汇编全局描述符表
4	gdt.h	C全局描述符表
5	idt.h	中断描述符表
6	hal.h	硬件抽象层
6.1	pic.h	可编程中断控制器
6.2	pit.h	可编程区间计时器
6.3	keyboard.h	键盘处理
6.4	tss.h	任务状态段
6.5	ide.h	硬盘读取
7	io.h	I/O编程
8	exception.h	异常处理
9	syscall.h	系统调用
10	task.h	多进程设施
11	user.h	用户程序处理
12	terminal.h	Shell
13	scancode.h	扫描码
14	stdio.h	标准输入输出
15	string.h	字符串处理
16	elf.h	ELF文件处理
17	api.h	进程管理API
18	semaphore.h	信号量机制
19	systhread.h	线程模型
20	pthread.h	线程管理API
21	fat12.h	FAT12文件系统
22	sysfile.h	虚拟文件系统
23	file.h	文件管理API

## 3. 用户程序

用户程序都放置在usr文件夹中。

序号	文件	描述
1-4	prgX.asm	飞翔字符用户程序
5	box.asm	画框用户程序
6	sys_test.asm	系统中断测试
7	fork_test.c	进程分支测试
8	fork2.c	进程多分支测试
9	bank.c	银行存取款测试
10	fruit.c	父子祝福水果测试
11	prod_cons.c	消费者生产者模型测试
12	hello_world_thread.c	多线程Hello_world测试
13	matmul.c	多线程矩阵乘法测试
14	file.c	文件读写测试

## 附录 B. 系统调用清单

int 0x80功能号	功能
0	输出OS Logo
1	睡眠100ms
10	fork
11	wait
12	exit
13	get_pid
20	get_sem
21	sem_wait
22	sem_signal
23	free_sem
100	返回内核Shell

int 0x81功能号	功能	int 0x82功能号	功能
0	pthread_create	0	fopen
1	pthread_join	1	fclose
2	pthread_self	2	fread
3	pthread_exit	3	fwrite
		4	fgets
		5	fputs