《操作系统实验》实验报告

实验名称：文件系统

姓名：胡育玮

学号：171860574

邮箱：yeevee@qq.com

班级：17 级计算机科学与技术系 2 班

一、实验过程

（1）step1：创建目录和文件。

```
stringCpy("fs.bin", driver, NAME_LENGTH - 1);
format(driver, SECTOR_NUM, SECTORS_PER_BLOCK);
```

一开始要调用 format()来将 GroupHeader、根目录等信息写入虚拟磁盘文件 fs.bin 中。

```
stringCpy("/boot", destDirPath, NAME_LENGTH - 1);
mkdir(driver, destDirPath);

stringCpy("/dev", destDirPath, NAME_LENGTH - 1);
mkdir(driver, destDirPath);

stringCpy("/usr", destDirPath, NAME_LENGTH - 1);
mkdir(driver, destDirPath);

stringCpy("/boot/initrd", destFilePath, NAME_LENGTH - 1);
// touch(driver, destFilePath);

stringCpy("./uMain.elf", srcFilePath, NAME_LENGTH - 1);
cp(driver, srcFilePath, destFilePath);
//touch(driver, "/boot/initrd");

stringCpy("/dev/stdin", destFilePath, NAME_LENGTH - 1);
touch(driver, destFilePath);

stringCpy("/dev/stdout", destFilePath, NAME_LENGTH - 1);
touch(driver, destFilePath);
```

根据课程网站，一共要创建 3 个目录和三个文件，其中 /boot/initrd 文件应该要由 uMain.elf 复制而来。上图使用了 func.c 中的 mkdir 函数来创建目录，使用 cp 来拷贝文件内容，使用 touch 来创建文件。

（2）step2：open 系统调用：

逻辑: 在 ret == 0，也即文件（设备）存在的情况下：

1. 先判断是不是设备，若是且找到匹配的设备（用 inodeoffset 来匹配），然后返回设备号：

```
for (i = 0; i < MAX_DEV_NUM; i++)
{
    if (dev[i].inodeOffset == destInodeOffset)
    {
        if (dev[i].state == 0)
            dev[i].state = 1;
        pcb[current].regs.eax = i;
        return;
    }
}
```

2. 若不是设备，则是文件，此时在 file[]数组中找 state 为 0 的空表项，给 file 表项赋值：offset 为 0（新打开文件，故为 0），flags 为 open 的参数，state 置 1，inodeoffset 置为相应的值。

由于打开不同文件时可能指定的 flag 不一样，故每次使用 open 来打开文件，不管这个文件是否已被打开，都要创建新的打开文件表项来存放相关信息。因此此处无需判断文件是否已经被打开。

```
for (i = 0; i < MAX_FILE_NUM; i++)
{
    if (file[i].state == 0)
    {
        file[i].state = 1;
        file[i].inodeOffset = destInodeOffset;
        file[i].offset = 0;
        file[i].flags = sf->edx; // a mix of O_DIRECTORY, O_CREATE, O_READ, O_WRITE
        pcb[current].regs.eax = i + MAX_DEV_NUM;
        return;
    }
}
```

3. 若找不到空的 file 项，则打开失败，返回-1：

```
if (i == MAX_FILE_NUM)
    {
        pcb[current].regs.eax = -1;
        return;
    }
```

若 ret !=0，也即该文件不存在：需要创建文件：

1.
```
//create a new file
if ((sf->edx & 4) == 0) //no O_create
{
    //printf("Not allowed to create new file!\n");
    pcb[current].regs.eax = -1;
    return;
}
```

若 open 的参数中未指定 create，则不允许创建文件。

2.

```
length = stringLen(str);
if (str[length - 1] == '/')
{ // last byte of destDirPath is '/'
    cond = 1;
    *(str + length - 1) = 0;
}

ret = stringChrR((const char *)str, '/', &size);
if (ret == -1)
{ // no '/' in destFilePath
    //printf("Incorrect destination file path.\n");
    if (cond == 1)
        *(str + length - 1) = '/';
    pcb[current].regs.eax = -1;
    return;
}

tmp = *(str + size + 1);
*(str + size + 1) = 0;
```

去掉文件名末尾的“/”，去掉最后一个/后面的字符，得到父目录的地址。

3.

```
ret = readInode(&sBlock, gDesc, &fatherInode, &fatherInodeOffset, (const char *)str); //not su
*(str + size + 1) = tmp;
if (ret == -1)
{
    //printf("Failed to read father inode.\n");
    if (cond == 1)
        *(str + length - 1) = '/';
    pcb[current].regs.eax = -1;
    return;
}

int a = 0;
if ((sf->edx & 8) == 0) //no O_dir
    a = REGULAR_TYPE;
else
    a = DIRECTORY_TYPE;

ret = allocInode(&sBlock, gDesc, // safe operation, none of the parameters would be modified
                 &fatherInode, fatherInodeOffset,
                 &destInode, &destInodeOffset, str + size + 1, a);
```

得到父目录的 iNode，然后使用 allocInode 接口来分配新的 iNode。

```
int i;
for (i = 0; i < MAX_FILE_NUM; i++)
{
    if (file[i].state == 0) //file not in use
    {
        //set process and sys open-file table
        file[i].state = 1;
        file[i].inodeOffset = destInodeOffset;
        file[i].offset = 0;
        file[i].flags = sf->edx; // a mix of O_DIRECTORY, O_CREATE, O_

        //if (i + MAX_DEV_NUM == 12)
        putChar(i + '0');
        putChar('\n');
        putChar('\n');
        pcb[current].regs.eax = i + MAX_DEV_NUM;

        //if ()
        return;
    }
}
```

4. 将新创建的文件的 inode 等信息记录在一个空的 file 表项中。

（3）step3：write 调用：

由于磁盘只能以块为单位进行读写，故要写文件则必须先读出文件的 offset 处所在的物理块，然后在后续添加，然后写回磁盘。注意，有可能在添加之后，需要写回磁盘的内容超出了一个物理块的大小，此时需要新分配物理块来写入。

1.

```
//uint8_t buffer[sBlock.blockSize];
ret = readBlock(&sBlock, &inode, quotient, buffer);
if (ret != 0)
{
    pcb[current].regs.eax = -1;
    return;
}
```

quotient 即为 offset 处所在的物理块号。此处读入该物理块到 buffer 中。

2.

```
//2. add contents to the buffer
if (size <= sBlock.blockSize - remainder) //bytes to w
{
    for (i = 0; i < size; i++)
    {
        buffer[i + remainder] = str[i];
    }
```

将额外的内容写入到 buffer 中。

3.

```
//3. write back the buffer
ret = writeBlock(&sBlock, &inode, quotient, buffer);
if (ret != 0)
{
    pcb[current].regs.eax = -1;
    return;
}

//4. modify other tables
//tables to modify: (1)file offset in file (2)file size in inode table  (3)bl
file[sf->ecx - MAX_DEV_NUM].offset += size;
inode.size += size;
diskWrite(&inode, sizeof(Inode), 1, file[sf->ecx - MAX_DEV_NUM].inodeOffset);
//diskRead(&inode, sizeof(Inode), 1, file[sf->ecx - MAX_DEV_NUM].inodeOffset)
```

将 buffer 写回到磁盘原物理块上。然后修改 offset 值，修改 inode 记录的文件大小，并将修改后的 iNode 写回 iNode

区。

4. 若是写入后的 buffer 将超出单个物理块的容量，则新分配一个物理块。先将能填满原有物理块（文件第 offset 个字节所在的物理块，块号为 quotient）的部分写入原物理块，然后再将剩余部分写入新分配的物理块。同样要注意修改 iNode 和 offset 的值。

（4）step4：read 调用：

1.

```
//1. consider if read size > file size
if (size > inode.size - file[sf->ecx - MAX_DEV_NUM].offset)
{
    size = inode.size - file[sf->ecx - MAX_DEV_NUM].offset;
}
```

若要读的字节数大于文件总大小 – Offset，则将 size 设为文件总大小 – Offset，只读这一点字节数。

2.

```
int off = remainder;
i = 0;
while (i < size)
{
    ret = readBlock(&sBlock, &inode, quotient, buffer);
    if (ret != 0)
    {
        pcb[current].regs.eax = -1;
        return;
    }

    while (off < sBlock.blockSize && i < size)
    {
        str[i] = buffer[off];
        i++;
        off++;
    }

    quotient++;
    off = 0;
}
```

循环读入：当一个物理块的字节被全部读完时，读入下一个物理块到 buffer 中，然后继续把 buffer 中的内容转移到 str 中。

3.
```
file[sf->ecx - MAX_DEV_NUM].offset += size;
pcb[current].regs.eax = size;
return;
```

设置 offset 的值。由于没有像 write 一样增添文件内容，故无需修改 inode。

（5）step5: seek 调用：

1.

```
*/
if ((file[sf->ecx - MAX_DEV_NUM].flags >> 1) % 2 == 0 && file[sf->ecx - MAX_DEV_NUM].flags % 2 == 0)
{ //XXX if O_READ and write is not set
    pcb[current].regs.eax = -1;
    return;
}
//                  a1:ecx  a2:edx  a3: ebx  a4:esi
```

若文件在打开时，没有谁知 O_read 或 O_write，则不允许进行 seek 操作。

2.

```
int off = sf->edx;
int whence = sf->ebx;

Inode inode;
diskRead(&inode, sizeof(Inode), 1, file[sf->ecx - MAX_DEV_NUM].inodeOffset);

if (whence == 1)
{
    off += file[sf->ecx - MAX_DEV_NUM].offset;
}
if (off < 0)
    off = 0;
if (off > inode.size)
    off = inode.size;

file[sf->ecx - MAX_DEV_NUM].offset = off;
pcb[current].regs.eax = 0;
```

由调用时的参数得知 whence 和 offset 的值。若 whence 等于 1 则最后要设置成的文件 offset = 参数 offset + 原来的文件 offset。做这个加法后还要判断是否超出了文件的大小限制，如大于文件长度，或小于 0。进行相应处理后，把文件 offset 设为这个计算后的参数 offset。

（6）step6：close 调用：

```
if (ind >= MAX_DEV_NUM && ind < MAX_DEV_NUM + MAX_FILE_NUM)
{
    ind -= MAX_DEV_NUM;
    if (file[ind].state == 0)
    {
        pcb[current].regs.eax = -1;
        return;
    }

    file[ind].state = 0;
}

else
{
    if (dev[ind].state == 0)
    {
        pcb[current].regs.eax = -1;
        return;
    }

    dev[ind].state = 0;
}
```

遍历打开文件表或设备表，若发现相应的表项已经处于关闭状态（state == 0），则错误，close 失败；否则将其 state 设为 1，成功。

（7）step7：remove 调用：

1. 若 ret ！= 0， 文件不存在，则错误，无法 remove：

```
if (ret != 0) //file doesn't exist
{
    pcb[current].regs.eax = -1;
    return;
}
```

2.

```
//readInode(&sBlock, gDesc, &destInode, &destInodeOffset, str);
ret = readInode(&sBlock, gDesc, &fatherInode, &fatherInodeOffset, (const char *)str);
*(str + size + 1) = tmp;
if (ret == -1)
{
    if (cond == 1)
        *(str + length - 1) = '/';
    pcb[current].regs.eax = -1;
    return;
}
```

读取父目录的 iNode。

3.

```
ret = freeInode(&sBlock, gDesc, &fatherInode, fatherInodeOffset, &destInode, &destInodeOffset, str + size + 1,
destFiletype);
```

调用 freeInode 接口来把已经分配的 inode 及物理块回收。

4.

```
for (i = 0; i < MAX_DEV_NUM; i++)
{
    if (dev[i].inodeOffset == destInodeOffset && dev[i].state == 1)
    {
        dev[i].state = 0;
    }
}
for (i = 0; i < MAX_FILE_NUM; i++)
{
    if (file[i].state == 1 && file[i].inodeOffset == destInodeOffset)
    {
        file[i].state = 0;
        file[i].inodeOffset = 0;
        file[i].offset = 0;
        file[i].flags = 0;
    }
}
```

将相应的打开文件表项或设备表项清除。

（8）step8：ls 调用：

1.

```
printf("ls %s, ", destFilePath);
int fd = open(destFilePath, O_READ);

printf("ls fd: %d\n", fd);
if (fd == -1)
{
    return 0;
}

DirEntry direntry;
int read_num = 0;
read_num = read(fd, (uint8_t *)&direntry, 128);
```

打开文件所在路径，用 DirEntry 结构体来解读打开的文件。

2.

```
while (read_num > 0)
{
    //i++;
    //printf("read_num = %d\n", read_num);
    if (direntry.inode != 0)
    {
        printf("%s ", direntry.name);
    }
    read_num = read(fd, (uint8_t *)&direntry, 128);
}
printf("\n");
close(fd);
```

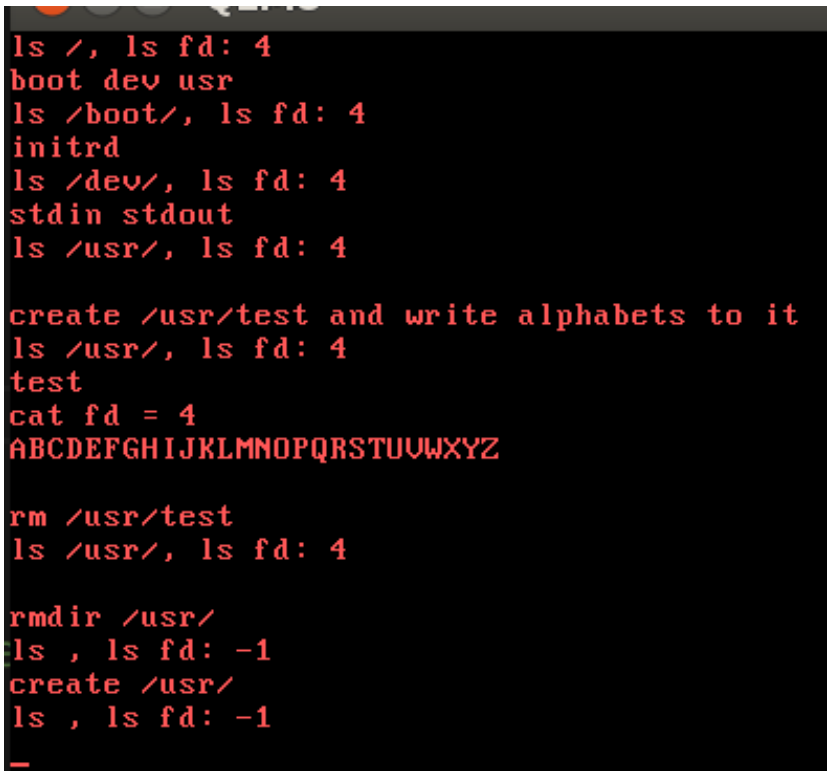
读取目录项中的内容（文件或目录名），然后打印出来。

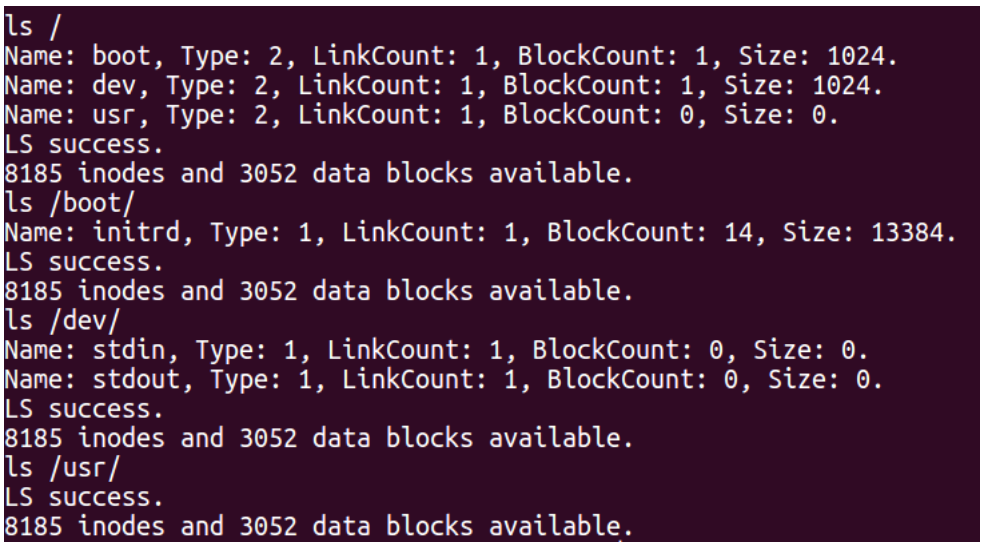
（9）step9：cat 调用：

1. 打开文件；读出文件内容到缓存中；打印缓存内容；关闭文件。

```
char str[LEN];
int fd = open(destFilePath, O_READ);
printf("cat fd = %d\n", fd);

if (fd == -1)
{
    return 0;
}

int ret = read(fd, (uint8_t*)str, LEN);

str[ret] = '\0';
printf("%s\n", str);
close(fd);

return 0;
```

（10）最终实现效果：





最后的效果有一点瑕疵，remove 函数有一点点问题。