



# 课程实验报告

实验名称 文件系统

课程名称 操作系统

院 系 计算机科学与技术系

学 号 191220129

姓 名 邢尚禹

邮 箱 191220129@smail.nju.edu.cn

实验日期 2021 年 6 月

# 目录

<b>1</b>	<b>实验进度</b>	<b>2</b>
<b>2</b>	<b>实验思路 and 过程</b>	<b>2</b>
2.1	open . . . . .	2
2.2	write . . . . .	5
2.3	read . . . . .	7
2.4	seek . . . . .	8
2.5	close . . . . .	9
2.6	remove . . . . .	10
<b>3</b>	<b>实验结果</b>	<b>12</b>
<b>4</b>	<b>问题与思考</b>	<b>12</b>
<b>5</b>	<b>建议</b>	<b>13</b>

# 1 实验进度

已完成所有内容。

## 2 实验思路 and 过程

本次实验主要实现几个系统调用。

### 2.1 open

syscallOpen 函数要对 pcb 和全局的文件描述符做一些修改，注意要充分考虑各种情况，如文件是否存在，文件是常规文件，设备文件还是目录，flag 是否匹配等。其执行流程大致如下。

- 如果文件存在:

检查文件类型和打开文件的 flag 是否匹配;

检查文件是否已经被打开，此处不允许同一个文件被打开两次;

判断文件类型，如果是设备文件，直接返回文件 id; 如果是普通文件，修改 pcb 和全局文件描述符。

```
1 if (((sf->edx >> 3) & 1) == 0 && destInode.type ==  
    DIRECTORY_TYPE) || (((sf->edx >> 3) & 1) == 1 &&  
    destInode.type == REGULAR_TYPE))  
2 {  
3     pcb[current].regs.eax = -1;  
4     return;  
5 }  
6 for (i = 0; i < MAX_FILE_NUM; ++i)  
7     if (file[i].inodeOffset == destInodeOffset && file[  
        i].state == 1)  
8     {  
9         pcb[current].regs.eax = -1;  
10        return;  
11    }  
12 for (i = 0; i < MAX_DEV_NUM; ++i)
```

```

13         if (dev[i].inodeOffset == destInodeOffset && dev[i]
14             ].state == 1)
15         {
16             pcb[current].regs.eax = i;
17             return;
18         }
19     for (i = 0; i < MAX_FILE_NUM; ++i)
20     {
21         if (file[i].state == 0)
22         {
23             file[i].state = 1;
24             file[i].flags = sf->edx;
25             file[i].inodeOffset = destInodeOffset;
26             file[i].offset = 0;
27             pcb[current].regs.eax = MAX_DEV_NUM + i;
28             return;
29         }
30     }
31     pcb[current].regs.eax = -1;

```

- 如果文件不存在：

检查 flag 是否有 create 权限；

根据是普通文件还是目录做 inode 分配，写入磁盘；

建立新的文件描述符。

```

1  if ((sf->edx >> 2) % 2 == 0)
2  {
3      pcb[current].regs.eax = -1;
4      return;
5  }
6  if ((sf->edx >> 3) % 2 == 0)
7  {
8      if (stringChrR(str, '/', &size) == -1)
9      {
10         pcb[current].regs.eax = -1;
11         return;
12     }
13     char parent_path[128];
14     for (int i = 0; i < size + 1; ++i)

```

```

15         parent_path[i] = *(str + i);
16     parent_path[size + 1] = 0;
17     ret = readInode(&sBlock, gDesc, &fatherInode, &
18         fatherInodeOffset, parent_path);
19     if (ret == -1)
20     {
21         pcb[current].regs.eax = -1;
22         return;
23     }
24     ret = allocInode(&sBlock, gDesc, &fatherInode,
25         fatherInodeOffset, &destInode, &destInodeOffset,
26         str + size + 1, REGULAR_TYPE);
27 }
28 else
29 {
30     length = strlen(str);
31     if (str[length - 1] == '/')
32     {
33         cond = 1;
34         str[length - 1] = 0;
35     }
36     ret = strchrR(str, '/', &size);
37     if (ret == -1)
38     {
39         pcb[current].regs.eax = -1;
40         return;
41     }
42     char parent_path[128];
43     for (int i = 0; i < size + 1; ++i)
44         parent_path[i] = *(str + i);
45     parent_path[size + 1] = 0;
46     ret = readInode(&sBlock, gDesc, &fatherInode, &
47         fatherInodeOffset, parent_path);
48     if (ret == -1)
49     {
50         if (cond == 1)
51             str[length - 1] = '/';
52         pcb[current].regs.eax = -1;

```

```

49         return;
50     }
51     ret = allocInode(&sBlock, gDesc, &fatherInode,
                    fatherInodeOffset, &destInode, &destInodeOffset,
                    str + size + 1, DIRECTORY_TYPE);
52     if (cond == 1)
53         str[length - 1] = '/';
54 }
55 if (ret == -1)
56 {
57     pcb[current].regs.eax = -1;
58     return;
59 }
60 for (i = 0; i < MAX_FILE_NUM; i++)
61     if (file[i].state == 0)
62     {
63         file[i].state = 1;
64         file[i].inodeOffset = destInodeOffset;
65         file[i].offset = 0;
66         file[i].flags = sf->edx;
67         pcb[current].regs.eax = MAX_DEV_NUM + i;
68         return;
69     }
70 pcb[current].regs.eax = -1; // create success but no
    available file[]

```

在实现中，要求出一个文件的父目录，可以直接从后到前查找符号‘/’，前面的字符串就是父路径，复制到一个新的缓冲区即可。框架代码提供了 stringChrR 和 stringLen 函数，用起来很方便。

## 2.2 write

syscallWriteFile 需要把指定的内容从缓冲区写入磁盘，注意每次需要写一个 block，故要设置两个变量，一个指示当前的字节数，一个指示当前的 block 位置。如果写入后的大小超出了 inode 限定的范围，则需要调用 allocBlock 分配一个新的 block。如果 allocBlock 失败，需要立刻将已经写

入的部分更新到文件描述符里。最后，调用 diskWrite 写入磁盘。

```
1 if (size <= 0)
2 {
3     pcb[current].regs.eax = 0;
4     return;
5 }
6 if (quotient < inode.blockCount)
7     readBlock(&sBlock, &inode, quotient, buffer);
8 while (i < size)
9 {
10     buffer[(remainder + i) % sBlock.blockSize] = str[i];
11     ++i;
12     if ((remainder + i) % sBlock.blockSize == 0)
13     {
14         if (quotient + j == inode.blockCount)
15         {
16             ret = allocBlock(&sBlock, gDesc, &inode,
17                             file[sf->ecx - MAX_DEV_NUM].
18                             inodeOffset);
19             if (ret == -1)
20             {
21                 inode.size = inode.blockCount *
22                             sBlock.blockSize;
23                 diskWrite(&inode, sizeof(Inode),
24                           1, file[sf->ecx -
25                               MAX_DEV_NUM].inodeOffset);
26                 pcb[current].regs.eax = inode.
27                     size - file[sf->ecx -
28                                 MAX_DEV_NUM].offset;
29                 file[sf->ecx - MAX_DEV_NUM].
30                     offset = inode.size;
31                 return;
32             }
33         }
34         writeBlock(&sBlock, &inode, quotient + j, buffer
35                   );
36         ++j;
37     }
38 }
```

```

28         if (quotient + j < inode.blockCount)
29             readBlock(&sBlock, &inode, quotient + j,
                       buffer);
30     }
31 }
32 if (quotient + j == inode.blockCount)
33 {
34     ret = allocBlock(&sBlock, gDesc, &inode, file[sf->ecx -
              MAX_DEV_NUM].inodeOffset);
35     if (ret == -1)
36     {
37         inode.size = inode.blockCount * sBlock.blockSize
              ;
38         diskWrite(&inode, sizeof(Inode), 1, file[sf->ecx
              - MAX_DEV_NUM].inodeOffset);
39         pcb[current].regs.eax = inode.size - file[sf->
              ecx - MAX_DEV_NUM].offset;
40         file[sf->ecx - MAX_DEV_NUM].offset = inode.size;
41         return;
42     }
43 }

```

## 2.3 read

syscallReadFile 的实现相对简单，采用与 write 相同的两个变量指示位置，直接调用 readBlock 即可。

```

1 if (size <= 0)
2 {
3     pcb[current].regs.eax = 0;
4     return;
5 }
6 if (size > inode.size - file[sf->ecx - MAX_DEV_NUM].offset)
7     size = inode.size - file[sf->ecx - MAX_DEV_NUM].offset;
8 readBlock(&sBlock, &inode, quotient, buffer);
9 ++j;
10 while (i < size)
11 {

```



```

12     str[i] = buffer[(remainder + i) % sBlock.blockSize];
13     ++i;
14     if ((remainder + i) % sBlock.blockSize == 0)
15     {
16         readBlock(&sBlock, &inode, quotient + j, buffer)
17         ;
18         ++j;
19     }
20 pcb[current].regs.eax = size;
21 file[sf->ecx - MAX_DEV_NUM].offset += size;

```

## 2.4 seek

根据三种不同的基准 (negin, current, end), 直接修改全局文件描述符的偏移量。

```

1 uint32_t cur_off;
2 switch (sf->ebx)
3 { // whence
4 case SEEK_SET:
5     if (offset >= 0 && offset <= inode.size)
6     {
7         file[sf->ecx - MAX_DEV_NUM].offset = offset;
8         pcb[current].regs.eax = 0;
9     }
10    else
11        pcb[current].regs.eax = -1;
12    break;
13 case SEEK_CUR:
14     cur_off = file[sf->ecx - MAX_DEV_NUM].offset;
15     if (cur_off + offset >= 0 && cur_off + offset <= inode.size)
16     {
17         file[sf->ecx - MAX_DEV_NUM].offset += offset;
18         pcb[current].regs.eax = 0;
19     }
20    else
21        pcb[current].regs.eax = -1;

```

```

22     break;
23 case SEEK_END:
24     if (offset + inode.size >= 0 && offset + inode.size <= inode
        .size)
25     {
26         file[sf->ecx - MAX_DEV_NUM].offset = offset + inode.size
            ;
27         pcb[current].regs.eax = 0;
28     }
29     else
30         pcb[current].regs.eax = -1;
31     break;
32 default:
33     break;
34 }

```

## 2.5 close

先检查文件是否已被打开和是否是普通文件，如果是，直接将对应的描述符清零即可。

```

1 int i = (int)sf->ecx;
2 if (i < MAX_DEV_NUM || i >= MAX_DEV_NUM + MAX_FILE_NUM)
3 {
4     pcb[current].regs.eax = -1;
5     return;
6 }
7 if (file[i - MAX_DEV_NUM].state == 0)
8 {
9     pcb[current].regs.eax = -1;
10    return;
11 }
12 file[i - MAX_DEV_NUM].state = 0;
13 file[i - MAX_DEV_NUM].inodeOffset = 0;
14 file[i - MAX_DEV_NUM].offset = 0;
15 file[i - MAX_DEV_NUM].flags = 0;
16 pcb[current].regs.eax = 0;

```

## 2.6 remove

remove 的关键操作就是 freeInode，不过需要注意常规文件和目录文件的区别以及不能删除设备文件。另外，对于目录文件，如果最后一个字符是 '/'，要单独处理。获取父目录的方法与 write 相同。

```
1 for (i = 0; i < MAX_DEV_NUM; ++i)
2     if (dev[i].inodeOffset == destInodeOffset)
3     {
4         pcb[current].regs.eax = -1;
5         return;
6     }
7 for (i = 0; i < MAX_FILE_NUM; ++i)
8     if (file[i].inodeOffset == destInodeOffset && file[i].
9         state == 1)
10    {
11        pcb[current].regs.eax = -1;
12        return;
13    }
14 if (destInode.type == REGULAR_TYPE)
15 {
16     if (stringChrR(str, '/', &size) == -1)
17     {
18         pcb[current].regs.eax = -1;
19         return;
20     }
21     char parent_path[128];
22     for (int i = 0; i < size + 1; ++i)
23         parent_path[i] = *(str + i);
24     parent_path[size + 1] = 0;
25     ret = readInode(&sBlock, gDesc, &fatherInode, &
26         fatherInodeOffset, parent_path);
27     if (ret == -1)
28     {
29         pcb[current].regs.eax = -1;
30         return;
31     }
32     ret = freeInode(&sBlock, gDesc, &fatherInode,
```

```

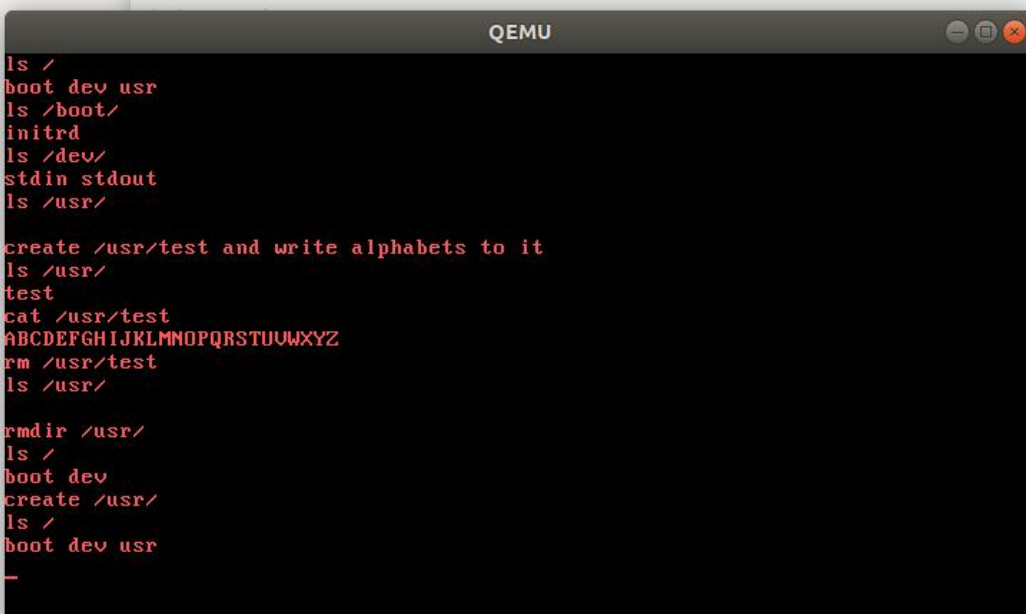
        fatherInodeOffset, &destInode, &destInodeOffset, str
        + size + 1, REGULAR_TYPE);
31 }
32 else if (destInode.type == DIRECTORY_TYPE)
33 {
34     length = stringLen(str);
35     if (str[length - 1] == '/')
36     {
37         cond = 1;
38         str[length - 1] = 0;
39     }
40     if (stringChrR(str, '/', &size) == -1)
41     {
42         pcb[current].regs.eax = -1;
43         return;
44     }
45     char parent_path[128];
46     for (int i = 0; i < size + 1; ++i)
47         parent_path[i] = *(str + i);
48     parent_path[size + 1] = 0;
49     ret = readInode(&sBlock, gDesc, &fatherInode, &
        fatherInodeOffset, parent_path);
50     if (ret == -1)
51     {
52         if (cond == 1)
53             str[length - 1] = '/';
54         pcb[current].regs.eax = -1;
55         return;
56     }
57     ret = freeInode(&sBlock, gDesc, &fatherInode,
        fatherInodeOffset, &destInode, &destInodeOffset, str
        + size + 1, DIRECTORY_TYPE);
58     if (cond == 1)
59         str[length - 1] = '/';
60 }
61 if (ret == -1)
62 {
63     pcb[current].regs.eax = -1;

```

```
64         return;
65     }
66     pcb[current].regs.eax = 0;
67     return;
```

### 3 实验结果

所有的系统调用都可以正确运行：



```
ls /
boot dev usr
ls /boot/
initrd
ls /dev/
stdin stdout
ls /usr/

create /usr/test and write alphabets to it
ls /usr/
test
cat /usr/test
ABCDEFGHIJKLMNOPQRSTUVWXYZ
rm /usr/test
ls /usr/

rmdir /usr/
ls /
boot dev
create /usr/
ls /
boot dev usr
```

图 1: 运行结果

### 4 问题与思考

1. 20.04 版本完全无法正常编译运行本次实验，即使代码正确，也会在第一个 `ls` 调用处卡住，尝试调试也没能发现问题。由于之前也出现过类似的问题（但通过修改 `bootloader` 和 `kvm` 的加载过程以及在 `Makefile` 中加编译参数可以解决），我就借用了同学的 18.04 编译，发现可以正常运行。暂时没有找到相关的解决方案。

2. 在框架代码中有很多工具函数，如 `sreingLen`, `stringChrR` 等，使用很方便；也有关键性的函数接口，如 `freeInode` 等。但是，这些函数没有具体的说明（参数的字符串格式等）。一开始我错误地调用了这些函数，产生了一些 bug，还很难找到，耗费了很多时间。

## 5 建议

1. 建议将官方的实验环境升级到 20.04LTS 版本。因为自己安装系统，一般都会选择最新版本 20.04。这样不容易产生 gcc 版本问题，可以节省很多时间；
2. 建议为框架代码提供的工具函数和关键性的函数接口提供一个清单，写清楚有哪些可以用、参数和返回值的含义等，这样学生做起来体验更好，不用把时间花在意义不大的 debug 上。