

同济大学计算机系

操作系统课程实验报告



学 号 2251557

姓 名 代文波

专 业 计算机科学与技术

授课老师 方钰

实验七：UNIX V6++文件系统

一 实验目的

结合课程所学知识，通过在 UNIX V6++实验环境中编写使用文件管理相关的系统调用或库函数的应用程序，进一步了解 UNIX 文件管理的工作过程。

二 实验设备及工具

已配置好 UNIX V6++运行和调试环境的 PC 机一台。

三 预备知识

(1) 在 UNIX V6++的/lib/file.c 文件中了解 UNIX V6++支持的所有和文件管理有关的库函数。

(2) 复习利用 fork, wait 和 exit 如何进行多进程编程及父子进程间的同步。

(3) 熟悉 UNIX 文件系统的内存打开结构和父子进程对文件打开结构的共享。

四 实验内容

4.1. 熟悉 UNIX 文件系统的接口

编写可执行程序 fileText, 实现以下功能：

(1) 进程在根目录下创建文件“/Jessy”，创建时设置三类用户对该文件都有读写和可执行的权限；（补充：这里我都是按照可读可写权限来创建文件了）

(2) 向其中写入字符串“Hello World! ”

(3) 进程将“/Jessy”文件的内容读出，屏幕打印，以判断写入的是否正确。

4.1.1 在 program 文件夹中加入一个名为 fileTest.c 的文件

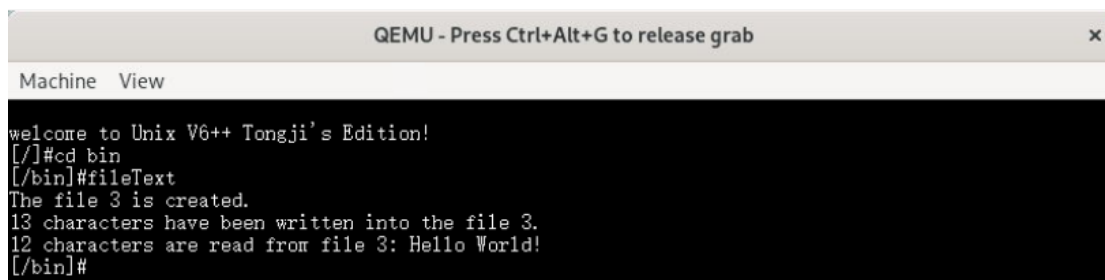
```
C fileText.c U X
programs > C fileText.c > main10
1  #include <stdio.h>
2  #include <sys.h>
3  #include <file.h>
4  void main1()
5  {
6      char data1[13]="Hello World!";
7      char data2[13];
8      int fd = 0;
9      int count = 0;
10
11     fd = creat("Jessy",0666);
12     if (fd>0)
13     {
14         printf("The file %d is created.\n",fd);
15     }
16     else
17     {
18         printf("The file can not be created.\n");
19     }
20
21     count = write(fd, data1, 13);
22     if (count == 13)
23     {
24         printf("%d characters have been written into the file %d.\n", count,fd);
25     }
26     else
27     {
28         printf("The file can not be written successfully.\n");
29     }
30     close(fd);
31
32     fd = open("Jessy",3);          //以可读可写的方式打开文件
33     count = read(fd, data2, 12);
34     printf("%d characters are read from file %d: %s.", count, fd, data2);
35     printf("\n");
36     close(fd);
37 }
```

4.1.2 重新编译运行 UNIX V6++代码

```
问题 输出 调试控制台 终端 端口
[vesper_center_279@archlinux unix-v6pp-tongji]$ make all
```

```
问题 输出 调试控制台 终端 端口
[bin] > [info] 切换路径。
[bin/..] > [info 9] 创建文件夹: etc
[bin/..] > [info] 切换路径。
[bin/../etc] > [info 5] 上传成功: v6pp_splash.bmp
[bin/../etc] > [info] 切换路径。
[bin/../etc/..] > [info 5] 上传成功: Shell.exe
[bin/../etc/..] > bye!
cp target/img-workspace/c.img target/
build success (unix-v6pp-tongji).
```

4.1.3 程序运行结果



```
QEMU - Press Ctrl+Alt+G to release grab
Machine View
welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#fileText
The file 3 is created.
13 characters have been written into the file 3.
12 characters are read from file 3: Hello World!
[/bin]#
```

4.1.4 问题一

文件创建成功之后, 为什么没有直接完成读写操作, 而是写过之后, 先关闭, 再重新打开?

(提示: 读者可以尝试在代码 1 中将写操作完成之后的关闭文件和打开文件两句代码注释掉, 将得到如下图所示的错误输出。



```
QEMU
Machine View
welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#fileTest
The file 3 is created.
12 characters have been written into the file 3.
-1 characters are read from file 3:
[/bin]#
```

尝试解释其中的原因, 并从 UNIX V6++ 的代码中找到依据。)

解答:

文件创建成功后, 会自动以可写的方式打开文件, 这时可以对文件进行写操作, 但是不可以对文件进行读操作。而后面我们要先把文件内容读到 data2, 然后把 data2 再写到文件里面, 所以只能先关闭文件, 再用可读可写模式打开文件。

UNIX V6++ 代码解释:

从下面代码的第 77 行可以看到, 在文件成功创建后, 会以 File::FWRITE (可写权限) 打开文件, 等待写入信息。

```
C fileText.c U C FileManager.h FileManager.cpp X
src > fs > FileManager.cpp > Creat()
47  /*
48  * 功能: 创建一个新的文件
49  * 效果: 建立打开文件结构, 内存1节点开锁、i_count 为正数 (应该是 1)
50  */
51  void FileManager::Creat()
52  {
53      Inode* pInode;
54      User& u = Kernel::Instance().GetUser();
55      unsigned int newACCMODE = u.u_arg[1] & (Inode::IRWXU|Inode::IRWXG|Inode::IRWXO);
56
57      /* 搜索目录的模式为1, 表示创建; 若父目录不可写, 出错返回 */
58      pInode = this->NameI(NextChar, FileManager::CREATE);
59      /* 没有找到相应的Inode, 或NameI出错 */
60      if ( NULL == pInode )
61      {
62          if(u.u_error)
63              return;
64          /* 创建Inode */
65          pInode = this->MakNode( newACCMODE & (~Inode::ISVTX) );
66          /* 创建失败 */
67          if ( NULL == pInode )
68          {
69              return;
70          }
71
72          /*
73          * 如果所希望的名字不存在, 使用参数trf = 2来调用open1()。
74          * 不需要进行权限检查, 因为刚刚建立的文件的权限和传入参数mode
75          * 所表示的权限内容是一样的。
76          */
77          this->Open1(pInode, File::FWRITE, 2);
78      }
```

4.1.5 问题二

在代码 1 中, 将两个字符串数组的长度都改为 12, 如下所示:

```
char data1[12]="Hello World!";
char data2[12];
```

程序运行将获得如下图所示的输出, 请解释出现这样的输出的原因。

```
QEMU - Press Ctrl+Alt+G to release grab
Machine View
welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#fileText
The file 3 is created.
13 characters have been written into the file 3.
12 characters are read from file 3: Hello World!Hello World!.
[/bin]#
```

解答:

如果这样修改, 字符串 data1 将会没有终止符 (尾 0), 之后从文件读出来写到 data2 后, 也会导致 data2 没有终止符 (尾 0)。此外, 在函数 main1 的栈帧中, 数据 data2、data1 连续存储, 这就导致在最后打印 data2 内容的时候, 会将 data2、data1 的内容全部打出来。

4.2 父子进程共享文件的读写权限和读写指针

将 fileTest 程序的代码修改成如代码 2 所示。代码主要流程如下：

- (1) 父进程首先创建了“/Jessy”文件，创建时，给三类用户分别设置了读写和可执行的权限；（补充：这里我都是按照可读可写权限来创建文件了）
- (2) 创建成功后，父进程将该文件关闭；
- (3) 父进程以可读可写的权限重新打开该文件，此时，建立了该文件的内存打开结构；
- (4) 成功创建子进程后，父进程睡眠等待子进程结束；
- (5) 子进程上台后，通过共享的文件打开结构，向“/Jessy”中写入“Hello World! ”，子进程结束， 唤醒父进程；
- (6) 父进程上台后，从该文件中读出“Hello World! ”，并在屏幕打印。

4.2.1 在 program 文件夹中修改名为 fileTest.c 的文件

```
C fileTest.c U x C FileManager.h G FileManager.cpp
programs > C fileTest.c > ...
1  #include <stdio.h>
2  #include <sys.h>
3  #include <file.h>
4
5  void main1()
6  {
7      char data1[13]="Hello World!";
8      char data2[13];
9      int fd = 0;
10     int count = 0;
11     int i,j;
12
13     fd = creat("Jessy",0666);      //刚创建好的文件，访问方式是可写
14     if (fd>0)
15     {
16         printf("The file %d is created.\n",fd);
17     }
18     else
19     {
20         printf("The file can not be created.\n");
21     }
22     close(fd);
23
24     fd = open("Jessy",3);          //以可读可写的方式打开文件
25
26     if(fork())
27     {
28         i=wait(&j);
29         seek(fd,0,0);
30         count = read(fd, data2, 12);
31         printf("%d characters are read from file %d: %s.", count, fd, data2);
32         printf("\n");
33     }
```

```

33         close(fd);
34     }
35     else
36     {
37         count = write(fd, data1, 13);
38         if (count == 13)
39         {
40             printf("%d characters have been written into the file %d.\n", count, fd);
41         }
42         else
43         {
44             printf("The file can not be written successfully.\n");
45         }
46         close(fd);
47         exit(0);
48     }
49 }

```

4.2.2 重新编译运行 UNIX V6++代码

```

问题 输出 调试控制台 终端 端口
[vesper_center_279@archlinux unix-v6pp-tongji]$ make all

问题 输出 调试控制台 终端 端口
[bin/../etc] > [info 5] 上传成功: v6pp_splash.bmp
[bin/../etc] > [info] 切换路径。
[bin/../etc/..] > [info 5] 上传成功: Shell.exe
[bin/../etc/..] > bye!
cp target/img-workspace/c.img target/
build success (unix-v6pp-tongji).
[vesper_center_279@archlinux unix-v6pp-tongji]$

```

4.2.3 程序运行结果

```

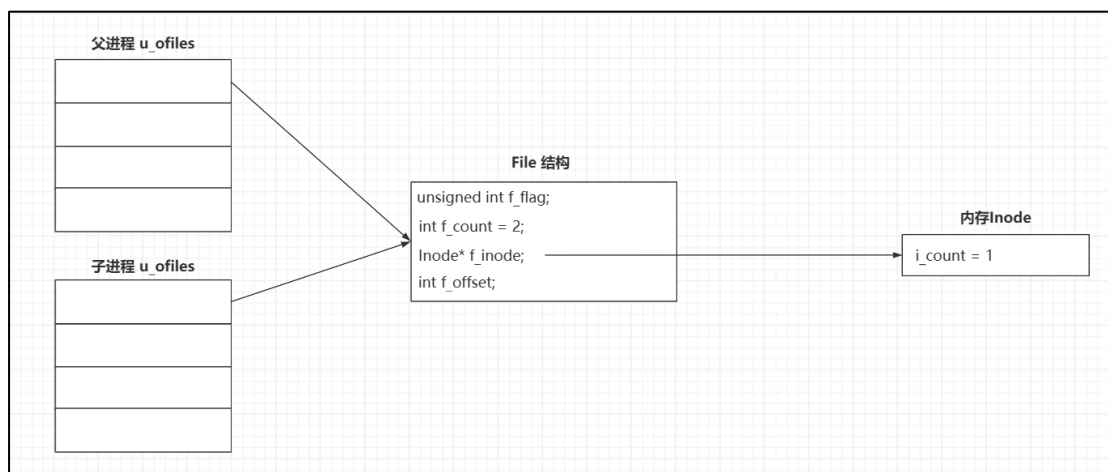
QEMU - Press Ctrl+Alt+G to release grab
Machine View
welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#fileText
The file 3 is created.
13 characters have been written into the file 3.
12 characters are read from file 3: Hello World!
[/bin]#

```

4.2.4 问题一

以文字或绘制的方式说明在代码 2 中，父子进程如何实现对文件的读写权限和读写指针的共享；

解答：代码 2 中父子进程共享了同一个 File 结构以及同一个内存 Inode 节点，因此具有文件读写权限和文件读写指针。



4.2.5 问题二

父进程被唤醒重新上台后，为什么要执行 seek 语句，如果没有这条语句，程序最后的输出是什么样的？为什么？

解答：

如果没有这条语句，最后输出如下图所示：

```

QEMU
Machine View
welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#fileText
The file 3 is created.
13 characters have been written into the file 3.
0 characters are read from file 3:
[/bin]#

```

原因：

父子进程共享文件读写指针，子进程写操作完成后，文件读写指针停留在文件末尾。如果不执行 seek 语句将其放置回文件开头，那么父进程将从文件末尾进行读操作，因此会读不到任何东西。

4.2.6 问题三

代码 2 中，父子进程执行的 close 操作有何不同？

解答：子进程先进行 close 操作，后先释放打开文件描述符 fd，然后让对应 File 结构中 f_count-1,发现 f_count 没有递减到 0，操作结束。之后，父进程执行 close

操作，首先释放打开文件描述符 fd，然后让对应 File 结构中 f_count--，发现 f_count=0，则释放该 File 结构，然后让对应内存 Inode 节点中 i_count--，发现 i_count=0，则释放该内存 Inode 节点。

4.3 父子进程以不同的读写权限打开文件

在本节实验中，要求读者按以下要求编写程序，并得到和下图完全一样的输出：



```
QEMU
Machine View
welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#fileTest
The file 3 is created.
12 characters have been written into the file 3.
12 characters are read from file 3: Hello World!
[/bin]#
```

要求：

- (1) 由父进程创建磁盘文件“/jessy”，创建时为三类用户分别设置可读可写和可执行的权限；（补充：这里我都是按照可读可写权限来创建文件了）
- (2) 父进程创建子进程，并睡眠等待子进程结束；
- (3) 子进程上台后，以可写的方式打开该文件，并向其中写入字符串“Hello World! ”，关闭文件，进程终止，并将写入的字符个数以终止码的方式传递给父进程；
- (4) 父进程被唤醒重新上台后，以只读的方式打开该文件，按照子进程终止码的数量，从该文件中 读取字符，并在屏幕打印，关闭文件。

4.3.1 在 program 文件夹中修改名为 fileTest.c 的文件

```
C fileText.c U X C FileManager.h G+ FileManager.cpp
programs > C fileText.c > main1()
1  #include <stdio.h>
2  #include <sys.h>
3  #include <file.h>
4
5  void main1()
6  {
7      char data1[13]="Hello World!";
8      char data2[13];
9      int fd = 0;
10     int count = 0;
11     int i,j;
12
13     fd = creat("Jessy",0666);    //刚创建好的文件，访问方式是可写
14     if (fd>0)
15     {
16         printf("The file %d is created.\n",fd);
17     }
18     else
19     {
20         printf("The file can not be created.\n");
21     }
22     close(fd);
23
24     if(fork())
25     {
26         i=wait(&j);
27         fd = open("Jessy",1);    //以可读的方式打开文件
28         count = read(fd, data2, j);
29         printf("%d characters are read from file %d: %s.", count, fd, data2);
30         printf("\n");
31         close(fd);
32     }
33     else
34     {
35         fd = open("Jessy",2);    //以可写的方式打开文件
36         count = write(fd, data1, 12);
37         if (count == 12)
```

```
5  void main1()
38  {
39      printf("%d characters have been written into the file %d.\n", count,fd);
40  }
41  else
42  {
43      printf("The file can not be written successfully.\n");
44  }
45  close(fd);
46  exit(count);
47  }
48 }
```

4.3.2 重新编译运行 UNIX V6++代码

```
问题 输出 调试控制台 终端 端口
• [vesper_center_279@archlinux unix-v6pp-tongji]$ make all
```

```
问题 输出 调试控制台 终端 端口
[bin/../../etc] > [info 5] 上传成功: v6pp_splash.bmp
[bin/../../etc] > [info] 切换路径。
[bin/../../etc/..] > [info 5] 上传成功: Shell.exe
[bin/../../etc/..] > bye!
cp target/img-workspace/c.img target/
build success (unix-v6pp-tongji).
```

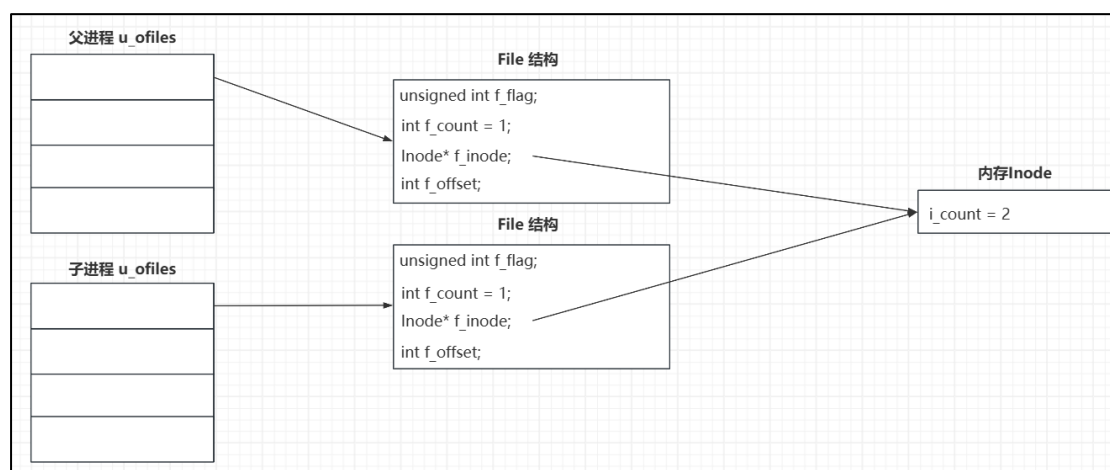
4.3.3 程序运行结果

```
QEMU - Press Ctrl+Alt+G to release grab
Machine View
welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#fileText
The file 3 is created.
12 characters have been written into the file 3.
12 characters are read from file 3: Hello World!
[/bin]#
```

4.3.4 问题一

以文字或绘制的方式说明在你的代码中，父子进程对 Jessy 文件的共享方式；

解答：父子进程分别使用一个 File 结构来共享一个内存 Inode 节点进而以不同的权限使用 Jessy 文件



4.3.5 问题二

在这样的共享方式中，父进程被唤醒重新上台后，是否还需要执行 seek 函数，为什么？

解答：不需要，因为父子进程有各自的 File 结构，所以有不同的文件读写指针，即子进程文件读写操作不会影响父进程的读写操作。

4.3.6 问题三

此处父子进程关闭文件的操作有何不同？

解答：

子进程先进行关闭文件，即执行 close 操作，首先释放打开文件描述符 fd，然后让对应的 File 结构中的 f_count-1,发现 f_count=0,则释放该 File 结构，然后让对应的内存 Inode 结构中的 i_count-1,发现 i_count 没有减到 0，则关闭文件操作结束。

之后，父进程进行关闭文件操作，即执行 close 操作，首先释放文件打开描述符 fd，然后让对应的 File 结构中的 f_count-1,发现 f_count=0,则释放该 File 结构，然后让对应的内存 Inode 结构中的 i_count-1,发现 i_count=0，则释放该内存 Inode 节点。