同济大学计算机系

操作系统实验报告

实验内容 UNIX V6++文件系统

学　　号 2251745

姓　　名 张宇

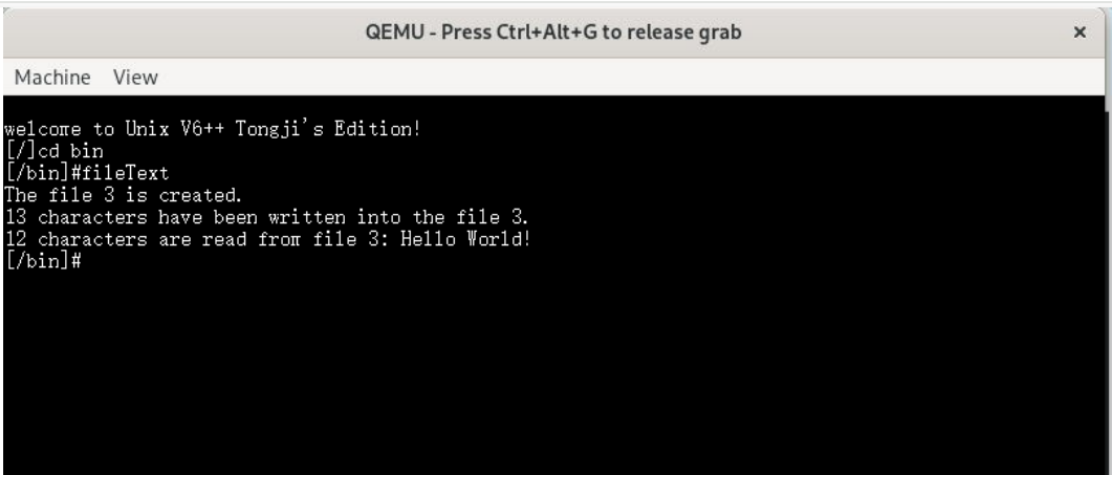
专　　业 计算机科学与技术

授课老师 方钰

一、完成实验 4.1

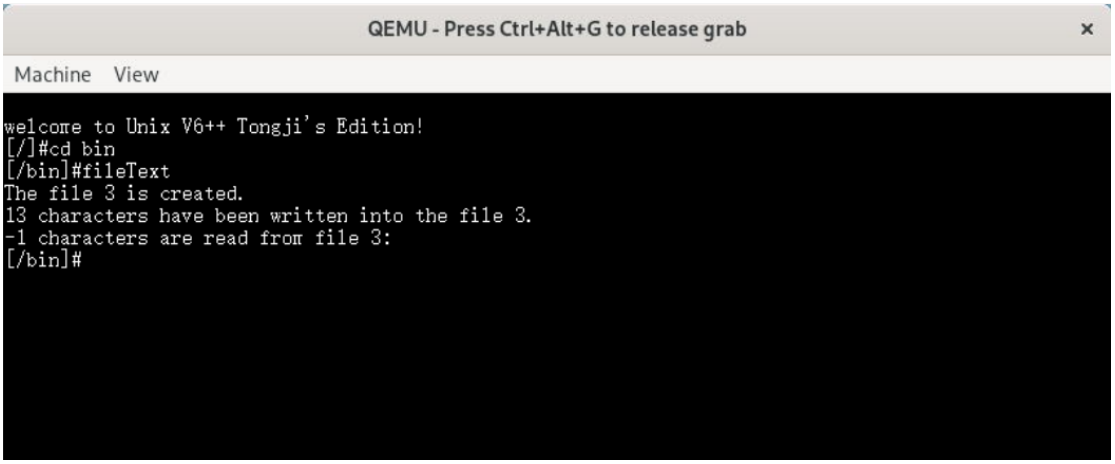
1.1 问题 1

运行 fileText 后结果如下：



文件创建成功之后，默认以可写的方式打开文件，所以这时候，继续对文件进行写操作是没有问题的，但是进行读操作会因为没有相应的权限而无法进行。

将关闭打开文件代码注释后，得到结果如下：



分析下面这个创建文件的函数可知，在创建文件后，是以写权限打开文件，所以如果想要进行读操作，就需要先关闭文件，再重新打开文件才可以实现。

```
/*
 * 功能：创建一个新的文件
 * 效果：建立打开文件结构，内存 i 节点开锁 、i_count 为正数（应该是 1）
 * */
void FileManager::Creat()
{
    Inode* pInode;
    User& u = Kernel::Instance().GetUser();
    unsigned int newACCMode = u.u_arg[1] &
(Inode::IRWXU|Inode::IRWXG|Inode::IRWXO);
    /* 搜索目录的模式为 1，表示创建；若父目录不可写，出错返回 */
    pInode = this->NameI(NextChar, FileManager::CREATE);
```

```
    /* 没有找到相应的 Inode，或 NameI 出错 */
    if ( NULL == pInode )
    {
        if(u.u_error)
            return;
        /* 创建 Inode */
        pInode = this->MakNode( newACCMode & (~Inode::ISVTX) );
        /* 创建失败 */
        if ( NULL == pInode )
        {
            return;
        }
        /*
         * 如果所希望的名字不存在，使用参数 trf = 2 来调用 open1()。
         * 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
         * 所表示的权限内容是一样的。
         */
        this->Open1(pInode, File::FWRITE, 2);
    }
    else
    {
        /* 如果 NameI()搜索到已经存在要创建的文件，则清空该文件（用算法
ITrunc()）。UID 没有改变
         * 原来 UNIX 的设计是这样：文件看上去就像新建的文件一样。然而，新文件所
有者和许可权方式没变。
         * 也就是说 creat 指定的 RWX 比特无效。
         * 邓蓉认为这是不合理的，应该改变。
         * 现在的实现：creat 指定的 RWX 比特有效 */
        this->Open1(pInode, File::FWRITE, 1);
        pInode->i_mode |= newACCMode;
    }
}
```

1.2 问题 2

将两个字符串数组的长度都改为 12 后，结果如下所示：

将两个字符串数组的长度都改为 12 后，字符串没有结束符，而在 main1 函数的栈帧中，data1 紧跟 data2 存储，因而在最后一个 printf 语句处，将 data2 和 data1 连续打印出来。

二、完成实验 4.2

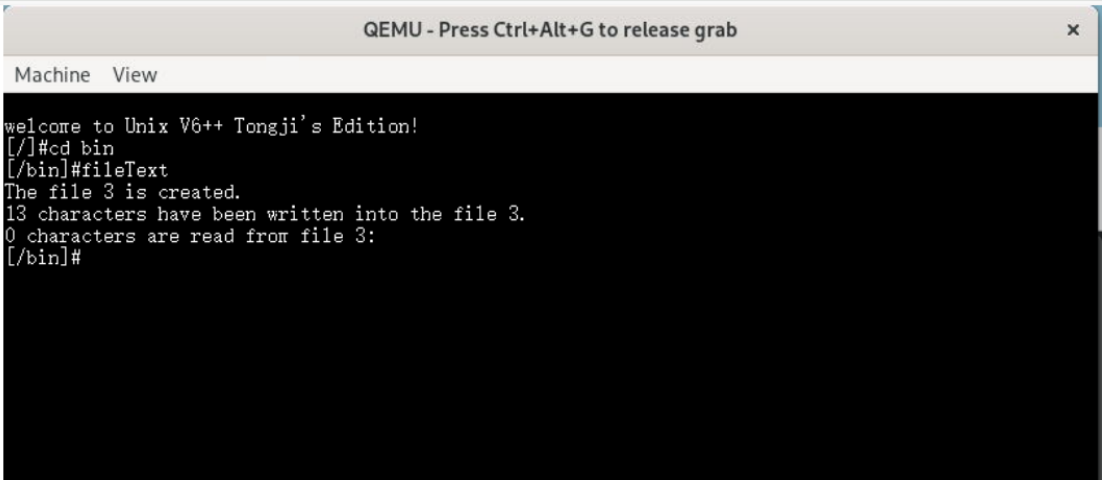
2.1 问题 1

父子进程共享了同一个 File 结构和同一个内存 iNode 节点，因为具有相同的文件读写权限和文件读写指针。

2.2 问题 2

因为父子进程共享同一个 File 结构，使用同一个文件读写指针。在子进程完成文件写操作后，文件读写指针指向文件尾。父进程被唤醒重新上台后，如果不执行 seek 语句，则文件读写指针仍然在文件尾，读操作无法读回“Hello World!”。

删除 seek 语句后程序输出如下图所示：



2.3 问题 3

子进程先执行 close 操作，因为此时内存 iNode 和 File 结构还有父进程在使用，所以只是释放打开文件描述符 fd，递减 File 结构引用计数 f_count，发现未

减到 0，则操作结束。

父进程后执行 close 操作，释放打开文件描述符 fd，递减 File 结构引用计数 f_count，发现减到 0，则释放该 File 结构，递减 iNode 节点引用计数 i_count，发现减到 0，则释放该 iNode 节点。

三、完成实验 4.3

代码 3 与程序运行结果如下：

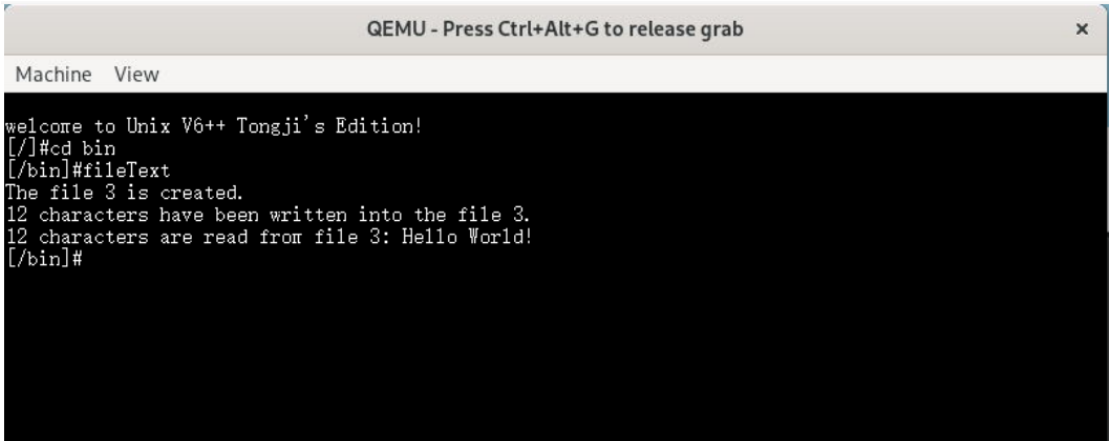
```
#include <stdio.h>
#include <sys.h>
#include <file.h>

void main1()
{
    char data1[13]="Hello World!";
    char data2[13];
    int fd = 0;
    int count = 0;
    int i,j;

    fd = creat("Jessy",0666);      //刚创建好的文件，访问方式是可写
    if (fd>0)
    {
        printf("The file %d is created.\n",fd);
    }
    else
    {
        printf("The file can not be created.\n");
    }
    close(fd);

    if(fork())
    {
        i=wait(&j);
        fd = open("Jessy",1);           //以可读的方式打开文件
        count = read(fd, data2, j);
        printf("%d characters are read from file %d: %s.", count, fd, data2);
        printf("\n");
        close(fd);
    }
    else
    {
        fd = open("Jessy",2);           //以可写的方式打开文件
        count = write(fd, data1, 12);
```

```
        if (count == 12)
        {
            printf("%d characters have been written into the file %d.\n",
count,fd);
        }
        else
        {
            printf("The file can not be written successfully.\n");
        }
        close(fd);
        exit(count);
    }
}
```



3.1 问题 1

父子进程共享同一个 iNode 节点，但是分别有自己的 File 结构，因而父子进程有不同的读写权限（父进程为读权限，子进程为写权限）和不同的读写指针

3.2 问题 2

在这样的共享方式中，父进程被唤醒重新上台后，不再需要执行 seek 函数。因为父子进程有不同的读写指针，子进程对文件的写操作不会影响父进程的读写指针位置。

3.3 问题 3

子进程先执行 close 操作，释放打开文件描述符 fd，递减 File 结构引用计数 f_count，发现减到 0，则释放该 File 结构，递减 iNode 节点引用计数 i_count，发现未减到 0（父进程的 File 结构还在使用），则操作结束。

父进程后执行 close 操作，释放打开文件描述符 fd，递减 File 结构引用计数 f_count，发现减到 0，则释放该 File 结构，递减 iNode 节点引用计数 i_count，发现减到 0，则释放该 iNode 节点，