

# 《操作系统》实验五报告

201220069 周心同

## 一、实验目的

在这一部分，我们将复习一下理论课知识，讨论一下什么是文件系统。

## 二、实验内容

- 本次实验共有如下任务点需要完成（编号都是从1开始计数）：
  - 3个exercise
  - 2个task
  - 1个challenge
- 完成内容：
  - 3个exercise
  - 2个task
  - 1个challenge

## 三、实验过程和结果（按照手册顺序）

**exercise1：**想想为什么我们不使用文件名而使用文件描述符作为文件标识。

```
int read(const char *filename, void *buffer, int size);  
int write(const char *filename, void *buffer, int size);
```

文件描述符表，记录了一个进程打开了哪些文件，每一个有效的文件描述符都对应一个文件的信息。

如果用文件名作为文件标识，不便于知道打开了哪些文件。

用文件描述符进行寻找文件，int类型在进行比较时比文件名更快速。

文件名可能存在同名问题，需要提供路径，比较麻烦。

**exercise2：**为什么内核在处理exec的时候，不需要对进程描述符表和系统文件打开表进行任何修改。（可以先往下看看再回答，或者阅读一下Xv6的shell）因为exec是以新的进程去代替原来的进程，但进程的PID保持不变，可以认为exec出来的进程仍是原来的父进程的子进程，可以使用在原进程中的文件描述符，不必关闭进程原来打开的文件或者对文件使用作修改。

**challenge1: system函数（自行搜索）通过创建一个子进程来执行命令。但一般情况下, system的结果都是输出到屏幕上，有时候我们需要在程序中对这些输出结果进行处理。一种解决方法是定义一个字符数组，并让 system 输出到字符数组中。如何使用重定向和管道来实现这样的效果？**

**Hint: 可以用pipe函数（自行搜索）、read函数（你们都会） .....  
（在Linux系统下自由实现，不要受约束）**

```
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
#include<string.h>

#define Size 100

int main()
{
    char buffer[Size]={0};
    int fd[2];
    pipe(fd);
    int ret = fork();
    if(ret)
    {
        read(fd[0],buffer,Size);
        printf("the result is:\n %s\n",buffer);
    }
    else
    {
        dup2(fd[1],STDOUT_FILENO);
        system("ls");
    }
    close(fd[0]);
    close(fd[1]);
    return 0;
}
```

输出:

```
oslab@oslab-VirtualBox:~/lab5$ g++ challenge1.c
oslab@oslab-VirtualBox:~/lab5$ ./a.out
the result is:
a.out
challenge1.c
lab5
xv6-riscv
```

**exercise3:** 我们可以通过which指令来找到一个程序在哪里，比如 which ls，就输出ls程序的绝对路径（看下面，绝对路径是/usr/bin/ls）。那我在/home/yxz这个目录执行ls的时候，为什么输出/home/yxz/路径下的文件列表，而不是/usr/bin/路径下的文件列表呢？（请根据上面的介绍解释。）

每个进程都有自己的工作目录，执行ls不带任何参数的的时候，就会输出自己工作目录（即当前目录）下的文件列表。

**task1:** 完成irqHandle.c里面有关文件的系统调用的内容，都用TODO标好了，并且添加了提示。

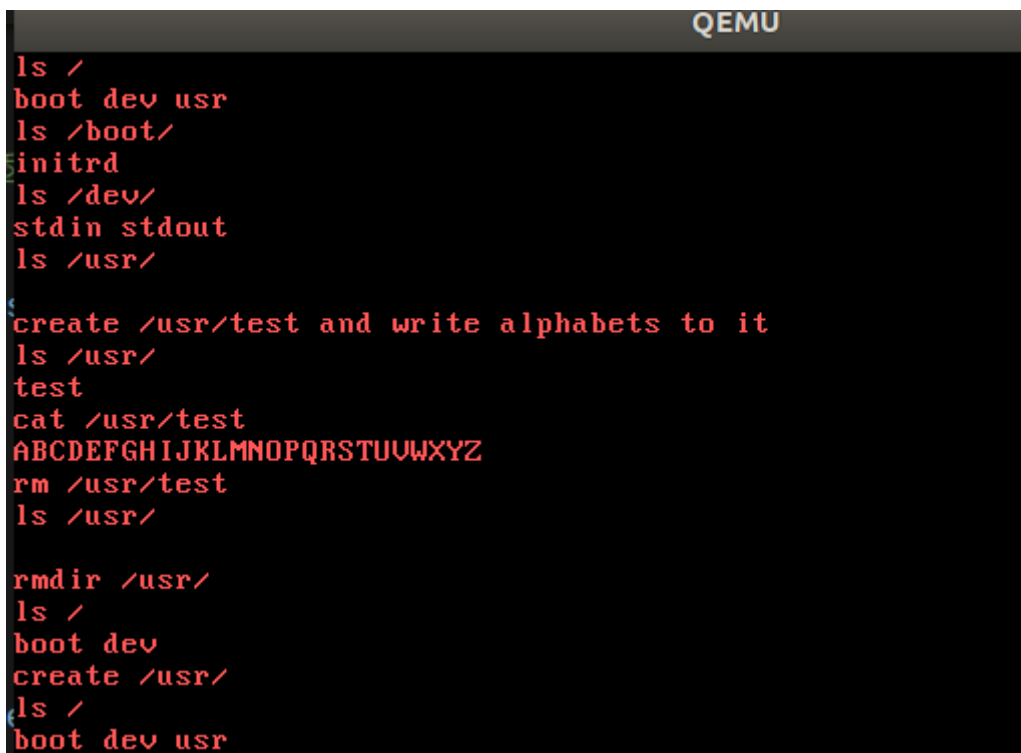
open和remove差不多，是对路径名的截取和函数的应用。

read和write是关于block的理解。

lseek和close填几句代码就行了

**task2:** 在app里面完善简易的ls和cat函数。

ls: 借用DirEntry结构把每次读入的8 ( $8*128=512*2$ ) 个DirEntry的name



```
QEMU
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
```

## 四、实验的启示/意见和建议

这次实验总体比较简单，但是由于提示比较少所以也卡了比较久。。。。

