OSLAB 5

学号	191830064
姓名	姜纪文
院系	地球科学与工程学院

一、exercise

$exercise1$：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。

首先如果使用文件名作为文件标识，不同目录下的文件名是可以一样的，会产生冲突。那么我可以采用文件的绝对路径即创建文件系统调用会返回创建文件的绝对路径，这套实现会让文件系统更加复杂，数据结构也会复杂。不如返回一个内核结构的索引值

$exercise2$：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

因为在exec前已经进行了fork操作，即子进程已经继承了父进程的文件索引表，并且相应地改动了内核的引用和计数参数，所以exec就不需要了。

$exercise3$：我们可以通过$which$指令来找到一个程序在哪里，比如$which$里，就输出ls程序的绝对路径。

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？

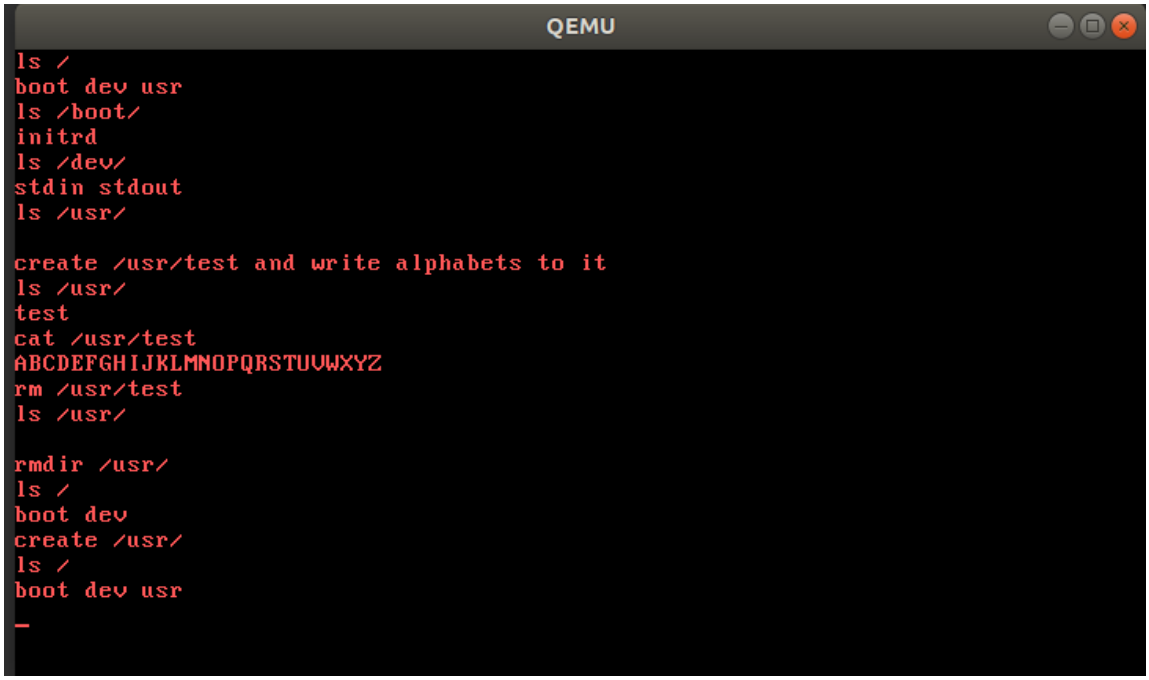
因为我们切换目录时即cd，会调用系统调用来更改工作目录，而执行ls则会默认以工作路径为参数，所以会输出工作路径下的文件列表。

二、库函数的实现以及ls,cat的实现

详情请看实验代码

1. **write**
2. **read**
3. **close**
4. **lseek**
5. **open**
6. **remove**
7. **open**
8. **ls**
9. **cat**

运行截图



三、Chanllenge

*challenge*1：*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？

利用`dup(),pipe(),read(),write()`函数可以达到目的，pipe提供进程通信，dup用于system的重定向。

```
// we do the fork and make the child process do system function,
//then the output of system() will go through pipe into the father's buffer which has already been prepared, atlast father process write the buffer into stdin.

int main()
{
    pid_t pid;
    char buf[1024];
    int fd[2]; //fd[0] read   fd[0]write
    if( pipe(fd) == -1)
        printf("pipe failed\n");//1 stdout 2 stderr 3 stdin 4 fd[0] 5 fd[1]
    pid = fork();
    if( pid > 0 )//father
    {
        //1 stdout 2 stderr 3 stdin 4 fd[0] 5 fd[1]
        close(fd[1]);
        read(fd[0],buf,1024);
        write(1,buf,1024);//write 1024bytes which are fetched from child process into stdin(terminal)
        wait(NULL);
        close(4);
        close(5);
    }
    else if( pid == 0)//child
    {
        int child_stdout = dup(1); // 1 stdout 2 stderr 3 stdin 4 fd[0] 5 fd[1] 6 stdout
        close(1);                  // 1 None 2 stderr 3 stdin 4 fd[0] 5 fd[1] 6 stdout
        int child_write_fd = dup(fd[1]);//1 fd[1] 2 stderr 3 stdin 4 fd[0] 5 fd[1] 6 stdout
        close(fd[0]);//1 fd[1] 2 stderr 3 stdin 4 None 5 fd[1] 6 stdout
        system("cat system.c");//1 fd[1] 2 stderr 3 stdin 4 None 5 fd[1] 6 stdout
        close(1);//1 None 2 stderr 3 stdin 4 None 5 fd[1] 6 stdout
        int resume_stdout = dup(child_stdout); //1 stdout 2 stderr 3 stdin 4 None 5 fd[1] 6 stdout
        close(5);
        close(6);
        sleep(2);//wait the father do write, wait 2 seconds
```

```
        printf("all is resume \n");
        exit(1);
    }
    return 0;
}
```