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);

答：

文件名是一个字符串，而文件描述符是一个整数，使用文件描述符作为标识时，可以更快速的找到对应的 FCB，从而更加高效的管理被打开的文件。

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**)

答：当内核载入第一个用户进程的时候，已经手动在文件描述符和系统打开文件表中填写了 STDIN\STDOUT\STDERR 这三个默认文件描述符的表项。之后，文件描述符表和系统打开文件表将通过系统调用来维护，内核不必额外处理。

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。）

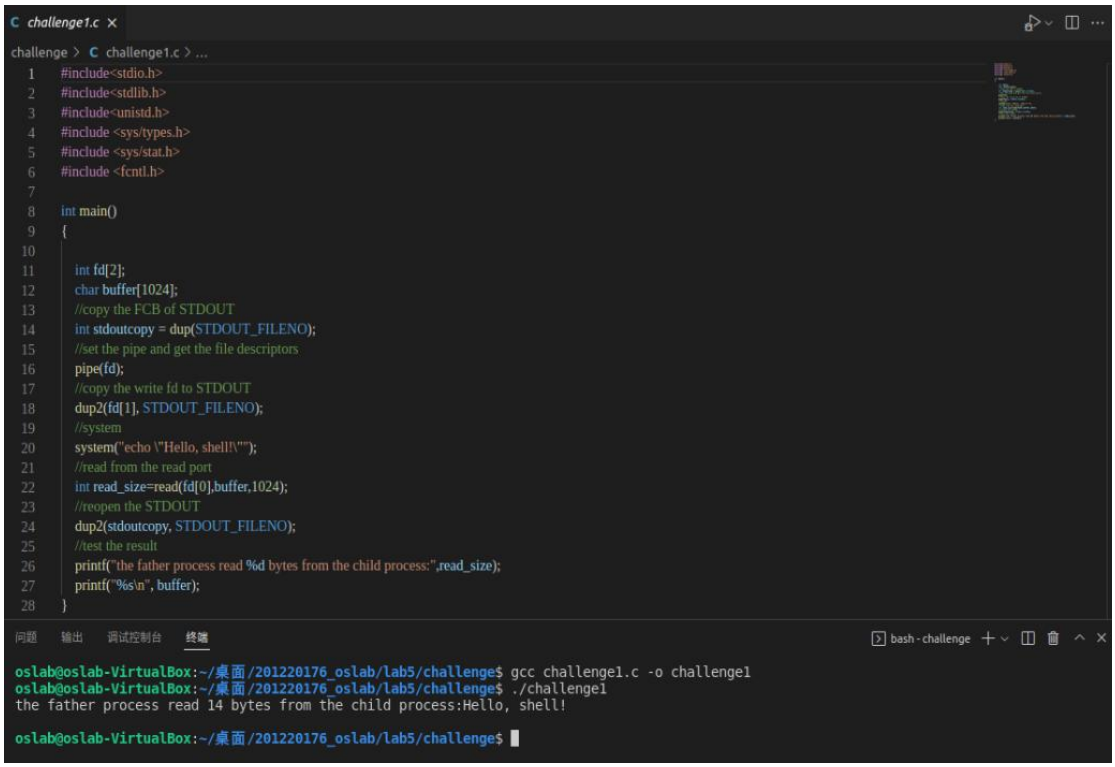
答：因为工作目录不是全局的，一个进程拥有一个工作目录，且当加载一个新的程序时，他会继承原来进程所在的目录，所以通过 which ls 执行 ls 时，他继承了当前目录即/home/yxz/，所以会输出当前路径下的文件列表

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）

实现如下：


challenge1.c
challenge > challenge1.c > ...
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int main()
{
int fd[2];
char buffer[1024];
//copy the FCB of STDOUT
int stdoutcopy = dup(STDOUT_FILENO);
//set the pipe and get the file descriptors
pipe(fd);
//copy the write fd to STDOUT
dup2(fd[1], STDOUT_FILENO);
//system
system("echo \"Hello, shell!\"");
//read from the read port
int read_size=read(fd[0],buffer,1024);
//reopen the STDOUT
dup2(stdoutcopy, STDOUT_FILENO);
//test the result
printf("the father process read %d bytes from the child process:",read_size);
printf("%s\n", buffer);
}
问题
输出
调试控制台
终端
bash - challenge
oslab@oslab-VirtualBox:~/桌面/201220176_oslab/lab5/challenge$ gcc challenge1.c -o challenge1
oslab@oslab-VirtualBox:~/桌面/201220176_oslab/lab5/challenge$ ./challenge1
the father process read 14 bytes from the child process:Hello, shell!
oslab@oslab-VirtualBox:~/桌面/201220176_oslab/lab5/challenge$