计算机系统安全实验报告

实验名称：文件权限管理及搭建虚拟环境

班级：1703101

学号：1170300520

姓名：郭子阳

计算机学院

1.1 Linux 系统文件和目录权限设置与辨识 setuid 程序 uid 差异

1. 设计并实现不同用户对不同类文件的 r、w、x 权限:

(1) 查看系统文件的权限设置

a) 查看/etc/passwd 文件和/etc/bin/passwd 文件的权限设置，并分析其权限为什么这么设置；

分别运行 ls -l /etc/passwd 和 ls -l /bin/passwd，结果如下：

```
[ziyang@ziyang-pc ~]$ ls -l /etc/passwd
-rw-r--r-- 1 root root 1867 11月  29 18:29 /etc/passwd
[ziyang@ziyang-pc ~]$ ls -l /bin/passwd
-rwsr-xr-x 1 root root 63624  8月   1 03:12 /bin/passwd
[ziyang@ziyang-pc ~]$
```

所有用户的信息都保存在/etc/passwd 中，该文件由 root 创建，每个用户对/etc/passwd 都有读权限，但是只有 root 对其有写权限，其他用户都没有写权限，保证了普通用户无法修改用户信息，保证了系统安全。/bin/passwd 用于修改用户的密码，任何用户都可以调用，该文件由 root 创建，密码保存在/etc/shadow 文件中，由于/etc/shadow 文件仅仅允许 root 进行读写，所以/bin/passwd 设置密码必须以 root 身份执行。所以/bin/passwd 设置了 setuid 位，允许普通用户以 root 身份运行该文件。

b) 找到 2 个设置了 setuid 位的可执行程序，该程序的功能，该程序如果不设置 setuid 位是否能够达到相应的功能，

mount 和 ping 程序，查看权限如下：

```
[ziyang@ziyang-pc ~]$ ls -l /bin/mount
-rwsr-xr-x 1 root root 51264  6月  27 18:04 /bin/mount
[ziyang@ziyang-pc ~]$ ls -l /bin/ping
-rwxr-xr-x 1 root root 68368  9月  14 03:49 /bin/ping
[ziyang@ziyang-pc ~]$
```

mount 程序用于将一个分区或者设备挂载在某空文件夹下，ping 命令用于测试网络连接量，由于 mount 挂载设备，ping 建立套接字，这两个行为都需要 root 权限，普通用户无法执行，所以需要设置 setuid 位，使运行这两个程序的 euid 是 root。

(2) 设置文件或目录权限

a) 用户 A 具有文本文件”流星雨.txt”，该用户允许别人下载；

首先创建一个测试用户 exp，与当前用户不同组，并设置密码：

```
[ziyang@ziyang-pc ~]$ sudo useradd exp
[sudo] ziyang 的密码：
[ziyang@ziyang-pc ~]$ sudo passwd exp
新的 密码：
重新输入新的 密码：
passwd：已成功更新密码
[ziyang@ziyang-pc ~]$
```

允许下载，即需要读权限即可，权限位可设置为 644：

```
[ziyang@ziyang-pc CSS_Lab1]$ vim 流星雨.txt
[ziyang@ziyang-pc CSS_Lab1]$ chmod 644 流星雨.txt
[ziyang@ziyang-pc CSS_Lab1]$ ls -l 流星雨.txt
-rw-r--r-- 1 ziyang ziyang 22 11月  29 19:40 流星雨.txt
[ziyang@ziyang-pc CSS_Lab1]$
[ziyang@ziyang-pc CSS_Lab1]$
```

当切换到其他用户时，仍然可以读取该文件：

```
[ziyang@ziyang-pc CSS_Lab1]$ su exp
密码：
[exp@ziyang-pc CSS_Lab1]$ cat 流星雨.txt
流星雨测试文件
```

b) 用户 A 编译了一个可执行文件”cal.exe”，该用户想在系统启动时运行；

首先创建 c 文件并编译为 cal.exe：

```
[ziyang@ziyang-pc CSS_Lab1]$ vim cal.c
[ziyang@ziyang-pc CSS_Lab1]$ gcc cal.c -o cal.exe
[ziyang@ziyang-pc CSS_Lab1]$
```

在系统启动时，无法确认是哪个用户，所以需要给所有用户该文件的执行权限：

```
[ziyang@ziyang-pc CSS_Lab1]$ chmod a+x cal.exe
[ziyang@ziyang-pc CSS_Lab1]$ ls -l cal.exe
-rwxr-xr-x 1 ziyang ziyang 16536 11月  29 19:46 cal.exe
[ziyang@ziyang-pc CSS_Lab1]$
```

在高版本 Linux 中，添加开机启动脚本需要使用 systemd，不赘述。

c) 用户 A 又起草了文件”demo.txt”，想让同组的用户帮其修改文件；

将文件权限设置为 664 即可，同组的用户即拥有读写权限：

```
[ziyang@ziyang-pc CSS_Lab1]$ vim demo.txt
[ziyang@ziyang-pc CSS_Lab1]$ chmod 664 demo.txt
[ziyang@ziyang-pc CSS_Lab1]$ ls -l demo.txt
-rw-rw-r-- 1 ziyang ziyang 0 11月  29 20:05 demo.txt
[ziyang@ziyang-pc CSS_Lab1]$
```

d) 一个 root 用户拥有的网络服务程序”netmonitor.exe”，需要设置 setuid 位才能完成其功能。

创建 netmonitor.exe 后，将权限设置为 4711 即可：

```
[ziyang@ziyang-pc CSS_Lab1]$ chmod 4711 netmonitor.exe
[ziyang@ziyang-pc CSS_Lab1]$ ls -l netmonitor.exe
-rws--x--x 1 ziyang ziyang 16536 11月  29 20:10 netmonitor.exe
[ziyang@ziyang-pc CSS_Lab1]$
```

2. 一些可执行程序运行时需要系统管理员权限，在 UNIX 中可以利用 setuid 位实现其功能，但 setuid 了的程序运行过程中拥有了 root 权限，因此在完成管理操作后需要切换到普通用户的身份执行后续操作。

(1) 设想一种场景，比如提供 http 网络服务，需要设置 setuid 位，并为该场景编制相应的代码；
(2) 如果用户 fork 进程后，父进程和子进程中 euid、ruid、suid 的差别；
(3) 利用 execl 执行 setuid 程序后，euid、ruid、suid 是否有变化；
(4) 程序何时需要临时性放弃 root 权限，何时需要永久性放弃 root 权限，并在程序中分别实现两种放弃权限方法；
(5) execl 函数族中有多个函数，比较有环境变量和无环境变量的函数使用的差异。

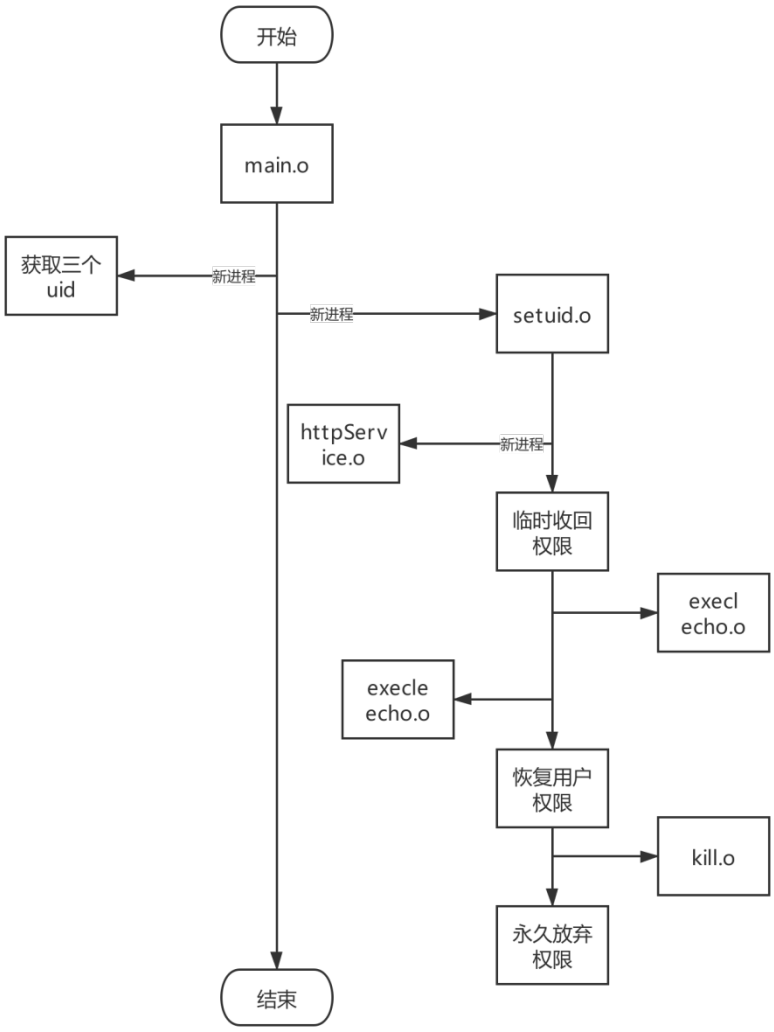
编写一系列代码，用以实现验证实验要求。
C 文件包括 main1.c、setuid.c、httpService.c、echo.c 和 kill.c，编译后为一系列同名的、后缀为.o 的可执行文件，可执行文件的权限如下：

```
[ziyang@ziyang-pc CSS_Lab1]$ ls -l *.o
-rwxr-xr-x 1 ziyang ziyang 16656 11月  30 22:21 echo.o
-rwx------ 1 root   root   16752 11月  30 22:21 httpService.o
-rwx------ 1 root   root   16808 11月  30 22:21 kill.o
-rwxr-xr-x 1 ziyang ziyang 17024 11月  30 22:21 main1.o
-rws--x--x 1 root   root   17216 11月  30 22:21 setuid.o
[ziyang@ziyang-pc CSS_Lab1]$
```

可以看出，echo.o、main1.o 和 setuid.o 任何用户都可以执行，其中 setuid.o 设置了 setuid 位，且属于 root 用户，httpService.o 和 kill.o 只允许 root 用户执行。
运行 main1.o 后，将按照一定顺序调用各个程序，以实现实验要求。执行结果如下（较长，截取部分）：

```
[ziyang@ziyang-pc CSS_Lab1]$ ./main1.o
Process Main, pid=6336
ruid=1000
euid=1000
suid=1000
Process fork, pid=6337
ruid=1000
euid=1000
suid=1000
Process setuid, pid=6338
ruid=1000
euid=0
suid=0
Revoke permission temporarily
ruid=1000
euid=1000
suid=0
Process http, pid=6339
ruid=1000
euid=0
suid=0
a root-only http service...
```

执行过程流程图如下：



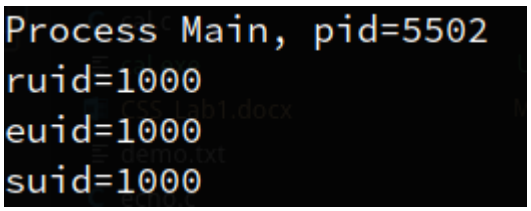
执行过程解释：

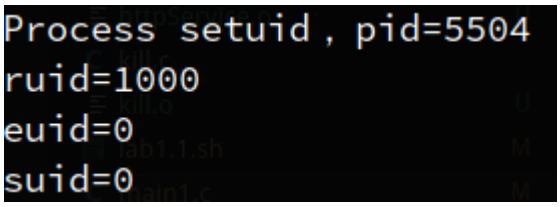
首先以用户身份执行 main1 程序，main1 进程 fork 一个子进程，在子进程中获取 ruid、euid 和 suid。随后，main1 进程又 fork 了一个子进程，并在子进程里调用了 setuid 程序，由于 setuid 被设置了 setuid 位，于是子进程将以 root 的身份运行，setuid 进程首先创建一个子进程并调用执行 httpService，该程序仅允许 root 用户执行。setuid 进程之后又临时放弃权限，并创建了两个子进程，分别以 execl 和 execle 的方式调用执行了 echo 程序，然后恢复 root 权限，调用仅有 root 可以执行的 kill 程序，关闭了一直运行的 httpService 进程，最后，setuid 进程永久放弃 root 权限。各父进程与子进程间都已使用 waitpid 同步（httpService）除外。

实验要求解释：

1）设想一种场景，比如提供 http 网络服务，需要设置 setuid 位，并为该场景编制相应的代码；

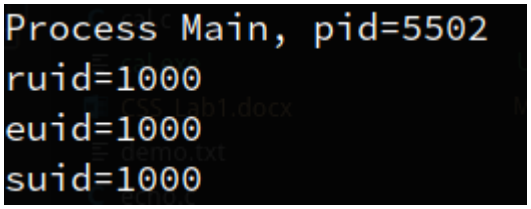
main1 进程在子进程中调用 setuid 时，由于 setuid 设置了 setuid 位，用户以 root 身份执行，main1 进程和 setuid 进程的 ruid、euid 和 suid 如下:

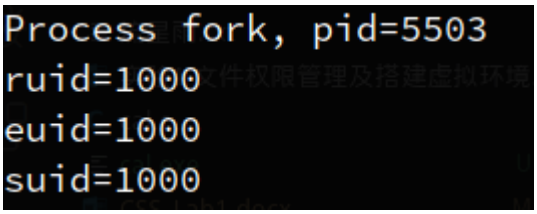




由于 setuid 的 euid 为 0,以 root 身份执行，所以 setuid 才可以调用只有 root 才可以执行的 httpService 程序

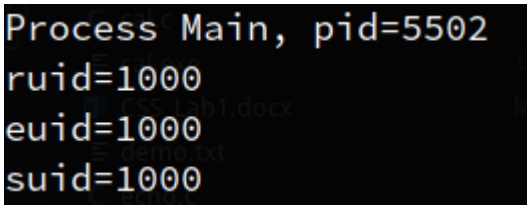
2）如果用户 fork 进程后，父进程和子进程中 euid、ruid、suid 的差别；
main1 进程 fork 一个子进程后，在子进程中查看 euid、ruid 和 suid 和父进程 main1 进程相同：

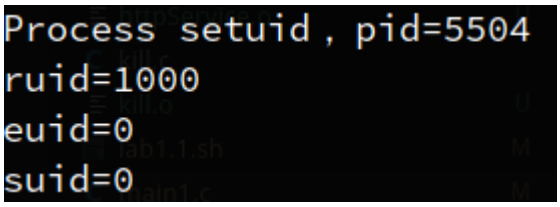




3）利用 execl 执行 setuid 程序后，euid、ruid、suid 是否有变化；

使用 execl 执行 setuid 程序后，ruid 不变，euid 和 suid 变为 0,即以 root 身份执行





4）程序何时需要临时性放弃 root 权限，何时需要永久性放弃 root 权限，并在程序中分别实现两种放弃权限方法；

执行完 root 权限才可以执行的操作后，如果以后还可能需要 root 权限，就临时性放弃权限，否则，就永久性放弃 root 权限。
需要临时放弃权限时，将当前的 euid 保存在 suid 处，并将 euid 设置为当前的 ruid，就实现了临时性放弃权限

```
//临时放弃权限
setresuid(ruid, ruid, euid);
```

需要永久性放弃权限时，将 euid 和 suid 都设置为当前的 ruid 即可：

```
// 永久放弃权限
setresuid(ruid, ruid, ruid);
```

5）execl 函数族中有多个函数，比较有环境变量和无环境变量的函数使用的差异。

当使用 execl 函数时，函数将会将进程的环境变量作为新执行程序的环境变量，而使用 exetle 函数时，需要传递一个指向环境变量字符串的指针，即可做到自定义新执行程序的环境变量。

实验 1.1 创建了自动执行的脚本，运行 lab1.1.sh 时即可完成实验的所有编译和授权工作。由于 git 无法读取仅 root 可读取的可执行文件，于是创建了 run_before_commit.sh，用于在 git 提交前删除所有的可执行文件。

1.2 chroot 的配置

1、准备基本的 chroot 环境

chroot 目录建立在～/chroot 下。

可以使用 whereis ftpd 命令查找到本机上的 ftpd 可执行文件：

```
[ziyang@ziyang-pc ~]$ whereis chroot
chroot: /usr/bin/chroot /usr/share/man/man2/chroot.2.gz /usr/share/man/man1/chroot.1.gz
[ziyang@ziyang-pc ~]$
```

本机中，ftpd 位于/usr/bin/chroot 下，可以直接使用 cp 或者 install 命令复制到～/chroot 文件夹中对应目录下。

ftpd 需要的动态链接库可以使用 ldd 命令查看：

```
[ziyang@ziyang-pc ~]$ ldd /usr/bin/chroot
        linux-vdso.so.1 (0x00007ffe651e5000)
        libc.so.6 => /usr/lib/libc.so.6 (0x00007f3ac1077000)
        /lib64/ld-linux-x86-64.so.2 => /usr/lib64/ld-linux-x86-64.so.2 (0x00007f3ac127c000)
[ziyang@ziyang-pc ~]$
```

在～/chroot 下建立对应的目录，将对应的动态链接库复制过去即可。

2、配置 chroot 环境

在/usr/bin 下建立文件 ftpd.sh，以保证可以带参数运行 ftpd，ftpd 将以守护进程的方式运行：

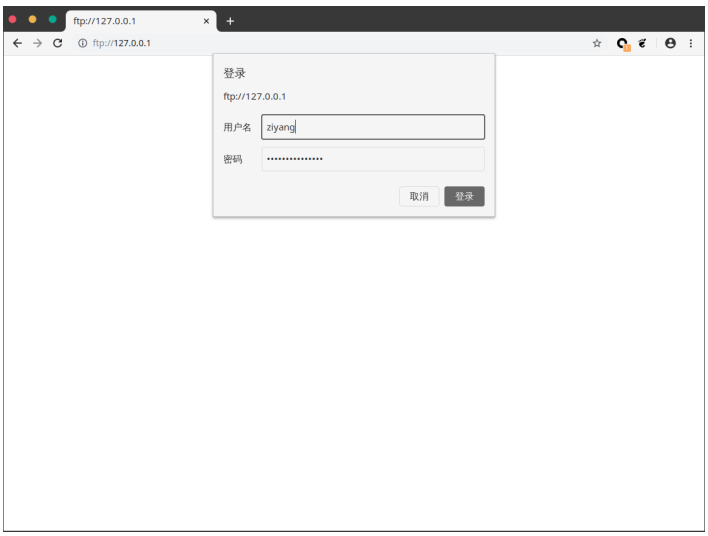
```
#!/bin/bash
/usr/bin/ftpd -D -4
```

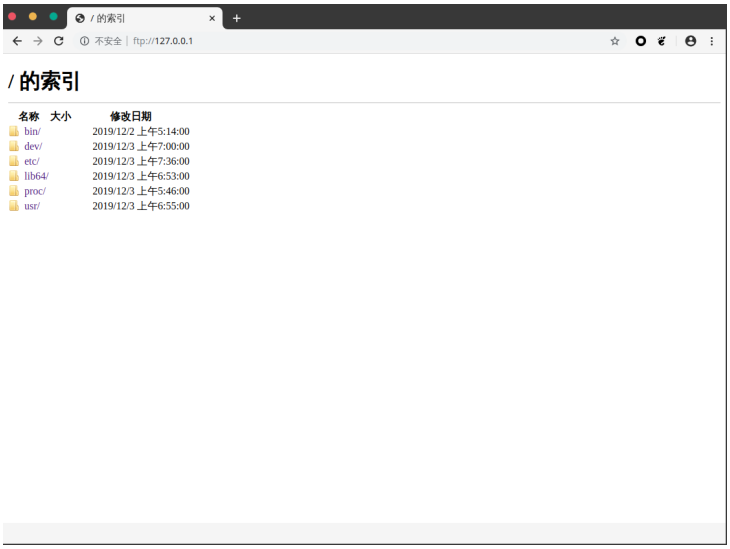
此时就可以通过 sudo chroot ~/chroot /usr/bin/ftpd.sh 运行 ftp 服务，运行后可以通过 ftp 127.0.0.1 命令或者浏览器进入 ftp://127.0.0.1 进入 ftp，但是由于 chroot 环境没有任何用户信息，就无法进行验证。

此时可以通过 ps -A 命令看到 ftpd 进程信息，可以 kill 掉。

之后需要将系统的用户和分组信息复制到 chroot 系统，主要是/etc/passwd、/etc/shadow、/etc/group 等文件。

然而，当根据实验报告要求将这些类似的文件粘贴到 chroot 系统后，仍然无法验证。于是将 passwd（重置用户密码）和 su（切换用户）复制到虚拟环境，试图运行这两个命令。（直接进入 chroot 需要 bash，按照 1 中步骤添加即可）
结果提示密码验证服务不完整，大概是这个原因导致的 ftp 无法登录验证服务。猜测可能是高版本 linux 修改了身份鉴定方式，或者不同的 linux 发行版对于身份认证的实现方式不同。
于是在虚拟环境中安装了 strace（主机安装后复制），使用 strace 来运行 su。命令：strace -o su.log su ziyang，使用这个命令后相当于执行 su ziyang 命令，但是会将日志保存在 su.log 中。错误出现后，可以在 su.log 中查看，以确定错误原因。大概可能是 pam 导致的身份鉴定模块不完善，经过一番 debug 后，su 可以顺利实现输入密码切换用户。重新运行 ftpd 的命令后，即可顺利登录 ftp 网页端。



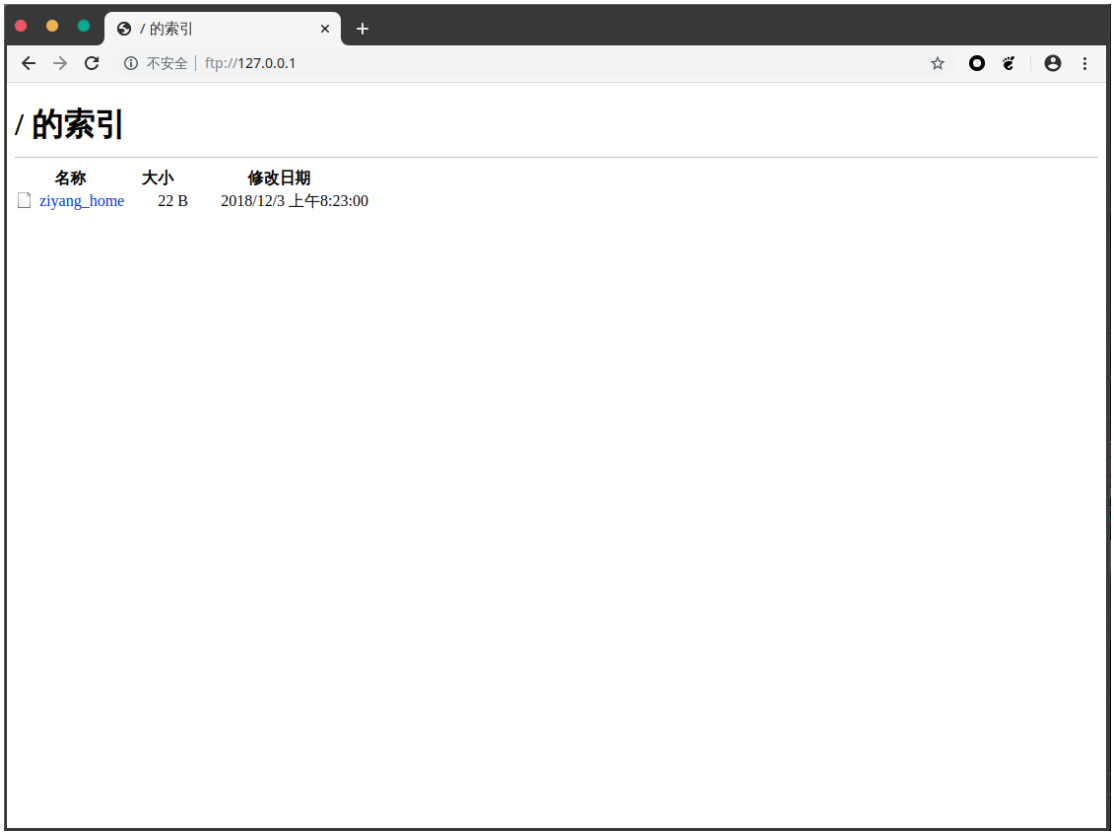


3、结尾工作
在～/chroot/etc 中创建 ftpusers 文件，将用户 exp 添加进去，再登录后，就会提示登录错误：

```
[ziyang@ziyang-pc ~]$ ftp 127.0.0.1
Connected to 127.0.0.1.
220 ziyang-pc FTP server (GNU inetutils 1.9.4) ready.
Name (127.0.0.1:ziyang): exp
331 Password required for exp.
Password:
530 Login incorrect.
ftp: Login failed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

在~/chroot/etc 中创建 ftpchroot 文件，以限制用户只能访问自己的 home 文件夹，

在 ftpchroot 文件中添加用户 ziyang，并且建立～/chroot/home/ziyang 文件夹作为用户 ziyang 的 home 目录，在 ziyang 文件夹下创建文件 ziyang_home 以标识。重启 ftpd 服务后，再以 ziyang 的身份登录后，ftp 服务的根即从～/chroot/home/ziyang 开始：



在～/chroot/etc 下创建 ftpwelcome 文件，在里面写入“Welcome to ziyang’s ftp server”作为连接 ftp 服务器时的欢迎信息，在～/chroot/etc 下创建 motd 文件，在里面写入“Successfully login in ziyang’s ftpd server!”作为用户登录后的提示信息，效果如下：

```
[ziyang@ziyang-pc ~]$ ftp 127.0.0.1
Connected to 127.0.0.1.
220- Welcome to ziyang's ftp server
220 ziyang-pc FTP server (GNU inetutils 1.9.4) ready.
Name (127.0.0.1:ziyang): ziyang
331 Password required for ziyang.
Password:
230- Successfully login in ziyang's ftpd server!
230 User ziyang logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

登录 ftp 时可以看到，如果是指定了只能访问 home 目录的用户，则 ftp 服务器呈现的根是从家目录开始的，如果是未指定的用户，则 ftp 服务器呈现的根是 chroot 环境的根，无法进入上级目录，即实现了虚拟化隔离。

chroot 后降低权限以及 cd 目录的验证，编写程序 mychroot.c 实现，如下：

```
#define _GNU_SOURCE
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

int main()
{
    uid_t ruid, euid, suid;
    getresuid(&ruid, &euid, &suid);
    printf("Before chroot:\nruid=%d\neuid=%d\nsuid=%d\n", ruid, euid, suid);
    printf("change dir\n");
    chdir("/home/ziyang/chroot");
    printf("Change root\n");
    if(chroot("/home/ziyang/chroot") == 0) {
        printf("change root succeed!\n");
    } else
    {
        printf("Change root error!\n");
        return 1;
    }
    printf("After chroot:\nruid=%d\neuid=%d\nsuid=%d\n", ruid, euid, suid);
    setresuid(ruid, ruid, ruid);
    getresuid(&ruid, &euid, &suid);
    printf("After cancel permission:\nruid=%d\neuid=%d\nsuid=%d\n", ruid, euid,
suid);
    execlp("ls", "ls", (char*)0);
    return 0;
}
```

该程序在 chroot 后永久地放弃 root 权限。

在 chroot 时应当在 chroot 环境的根下执行，否则可能会导致权限泄露的问题。验证时只要将上述程序中的 chdir 语句注释掉即可。当注释掉 chdir 语句后，程序最后执行的 ls 命令显示的结果是在该程序目录下，如果该程序目录不在 chroot 环境，那么就导致了权限泄露问题，在 chroot 后依旧可以对 chroot 环境外的文件进行修改。