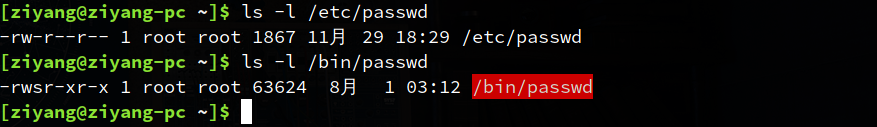
1.1 Linux系统文件和目录权限设置与辨识setuid程序uid差异

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；

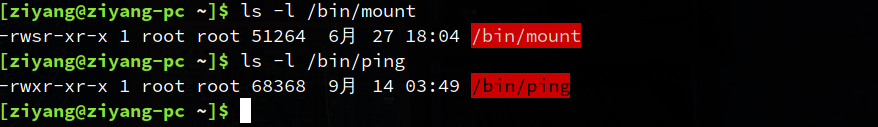
分别运行ls -l /etc/passwd和ls -l /bin/passwd，结果如下：



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，

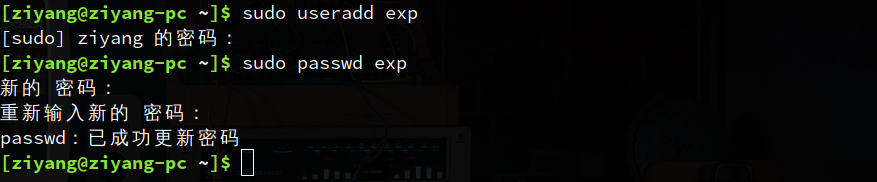
mount和ping程序，查看权限如下：



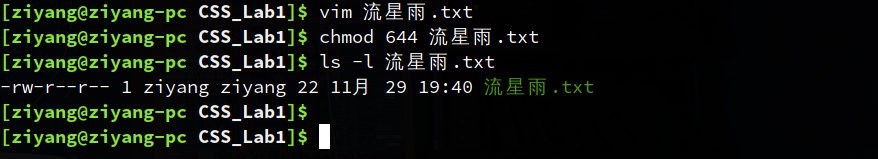
mount程序用于将一个分区或者设备挂载在某空文件夹下，ping命令用于测试网络连接量，由于mount挂载设备，ping建立套接字，这两个行为都需要root权限，普通用户无法执行，所以需要设置setuid位，使运行这两个程序的euid是root。

1. 设置文件或目录权限
2. 用户A具有文本文件”流星雨.txt”，该用户允许别人下载；

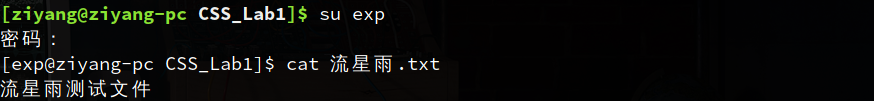
首先创建一个测试用户exp，与当前用户不同组，并设置密码：



允许下载，即需要读权限即可，权限位可设置为644：

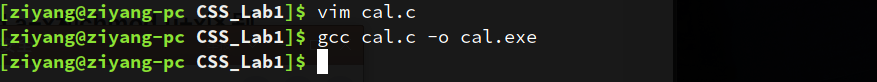


当切换到其他用户时，仍然可以读取该文件：

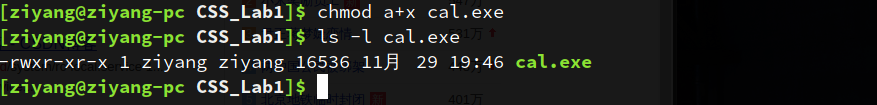


1. 用户A编译了一个可执行文件”cal.exe”，该用户想在系统启动时运行；

首先创建c文件并编译为cal.exe：



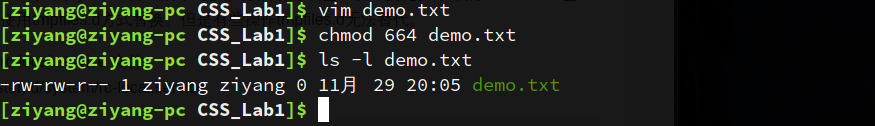
在系统启动时，无法确认是哪个用户，所以需要给所有用户该文件的执行权限：



在高版本Linux中，添加开机启动脚本需要使用systemd，不赘述。

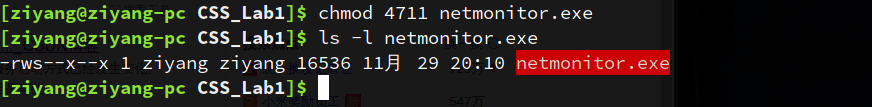
1. 用户A又起草了文件”demo.txt”，想让同组的用户帮其修改文件；

将文件权限设置为664即可，同组的用户即拥有读写权限：



1. 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。

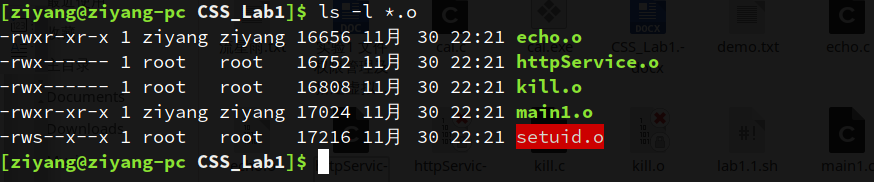
创建netmonitor.exe后，将权限设置为4711即可：



1. 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。
2. 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；
3. 如果用户fork进程后，父进程和子进程中euid、ruid、suid的差别；
4. 利用execl执行setuid程序后，euid、ruid、suid是否有变化；
5. 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；
6. 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。

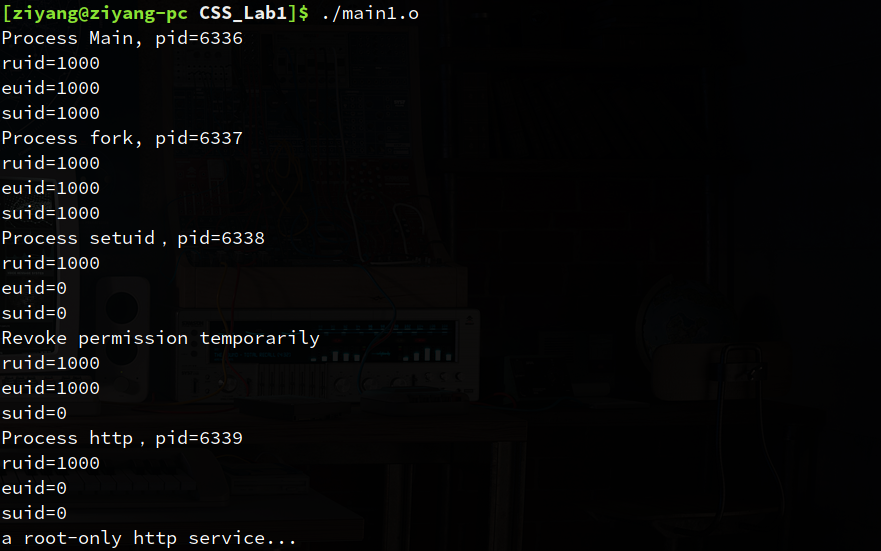
编写一系列代码，用以实现验证实验要求。

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：

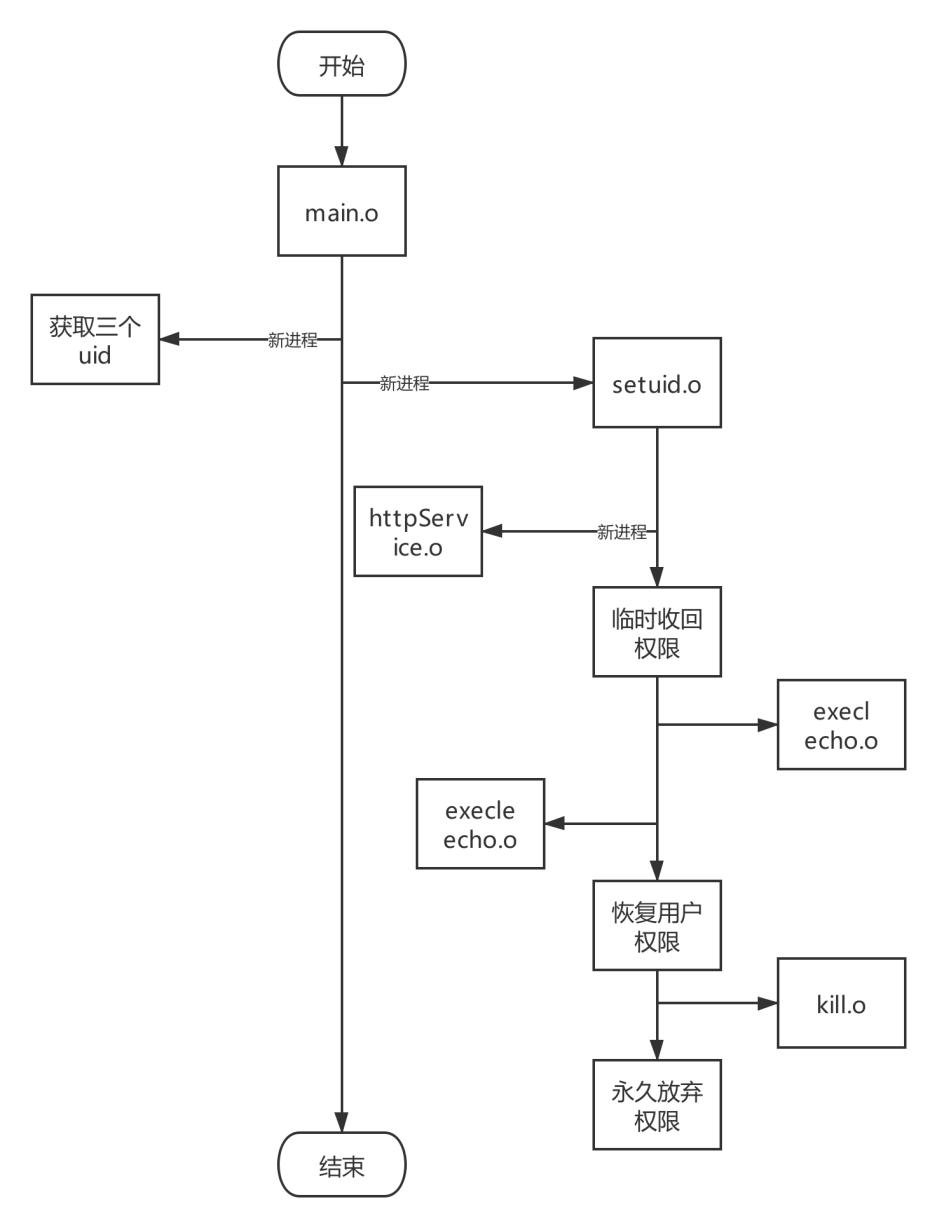


可以看出，echo.o、main1.o和setuid.o任何用户都可以执行，其中setuid.o设置了setuid位，且属于root用户，httpService.o和kill.o只允许root用户执行。

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）：



执行过程流程图如下：



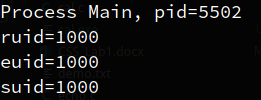
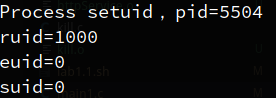
执行过程解释：

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。

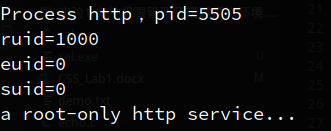
实验要求解释：

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:

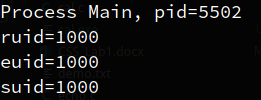
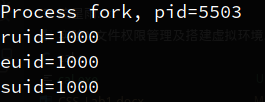
 

由于setuid的euid为0,以root身份执行，所以setuid才可以调用只有root才可以执行的httpService程序



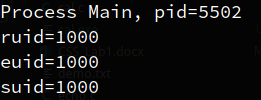
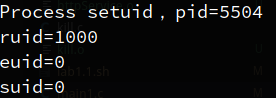
2）如果用户fork进程后，父进程和子进程中euid、ruid、suid的差别；

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：

1. 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

1. 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);

1. 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。

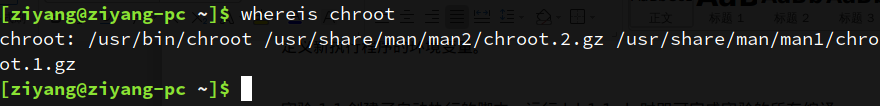
实验1.1创建了自动执行的脚本，运行lab1.1.sh时即可完成实验的所有编译和授权工作。由于git无法读取仅root可读取的可执行文件，于是创建了run\_before\_commit.sh，用于在git提交前删除所有的可执行文件。

1.2 chroot的配置

1、准备基本的chroot环境

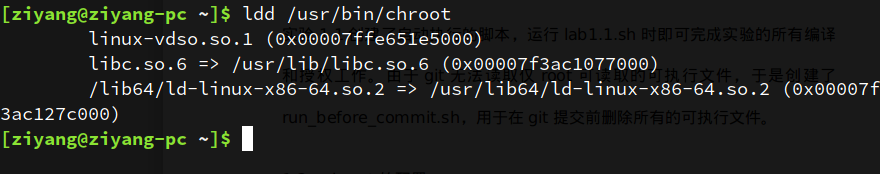
chroot目录建立在～/chroot下。

可以使用whereis ftpd命令查找到本机上的ftpd可执行文件：



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：



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。

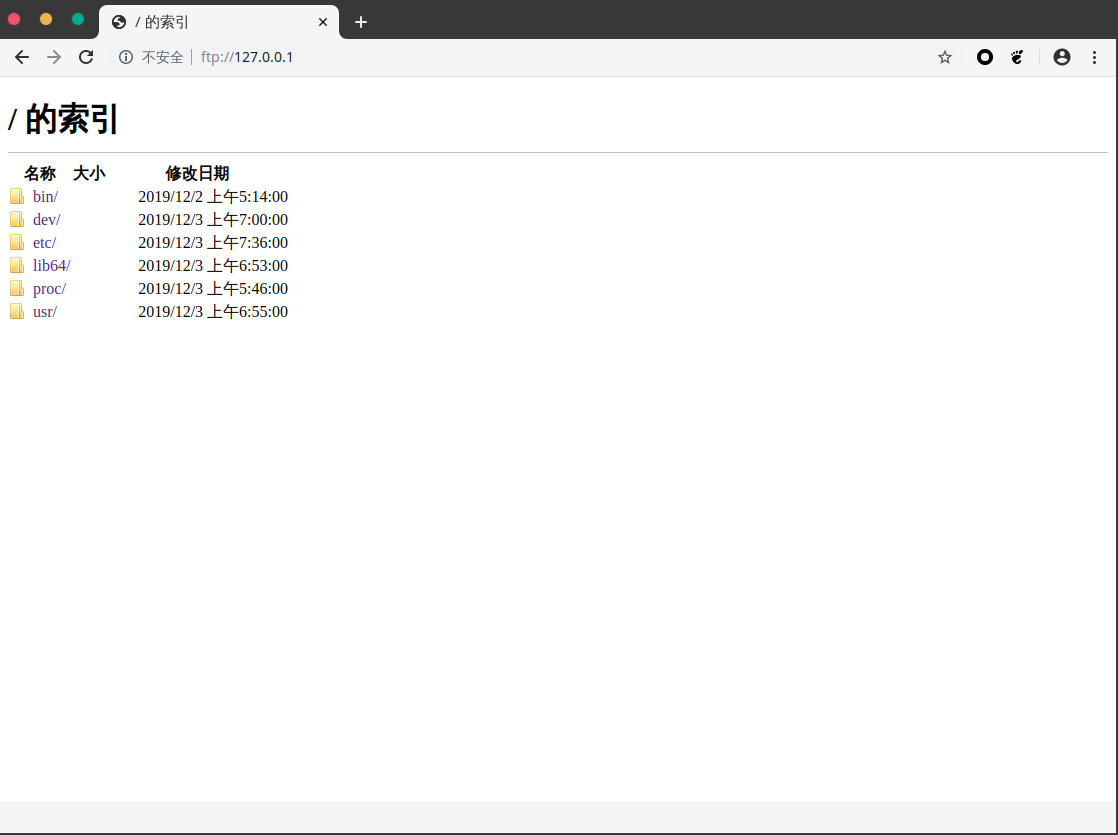
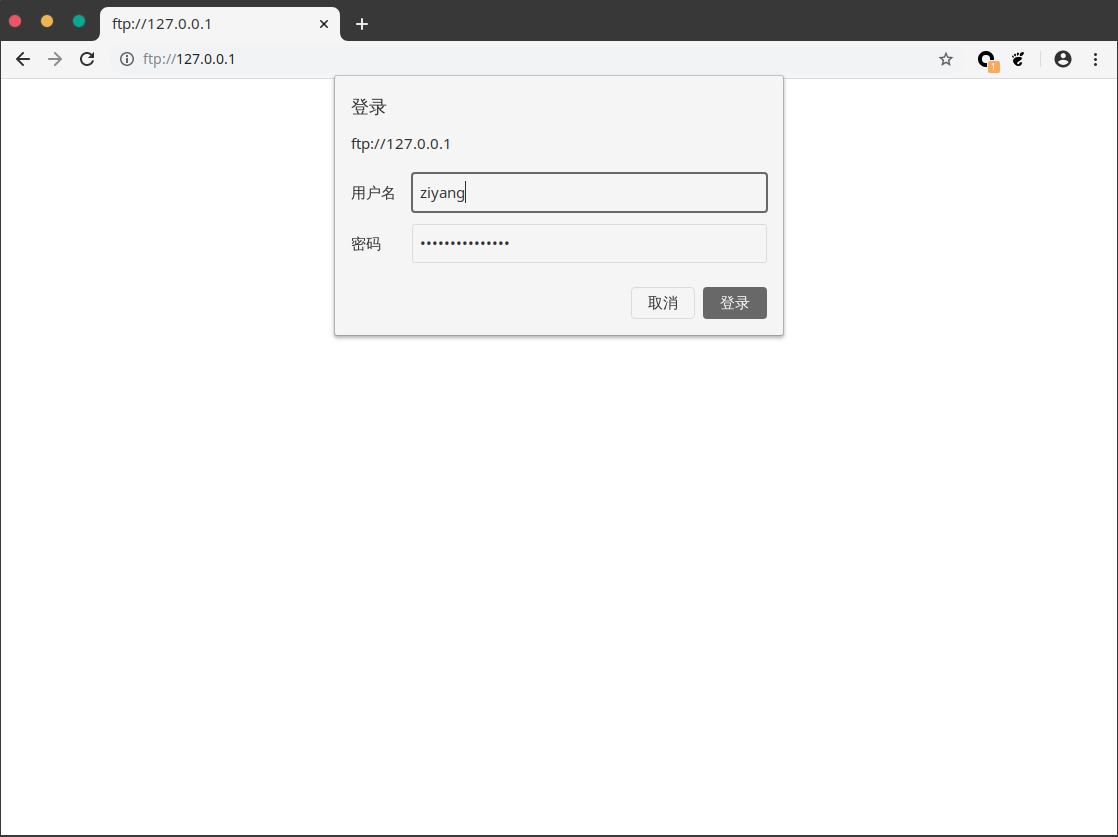
之后需要将系统的用户和分组信息复制到chroot系统，主要是/etc/passwd、/etc/shadow、/etc/group等文件。

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。

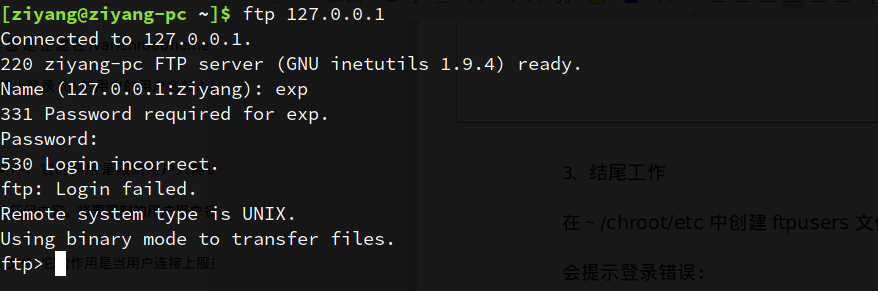
于是在虚拟环境中安装了strace（主机安装后复制），使用strace来运行su。命令：strace -o su.log su ziyang，使用这个命令后相当于执行su ziyang命令，但是会将日志保存在su.log中。错误出现后，可以在su.log中查看，以确定错误原因。

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。

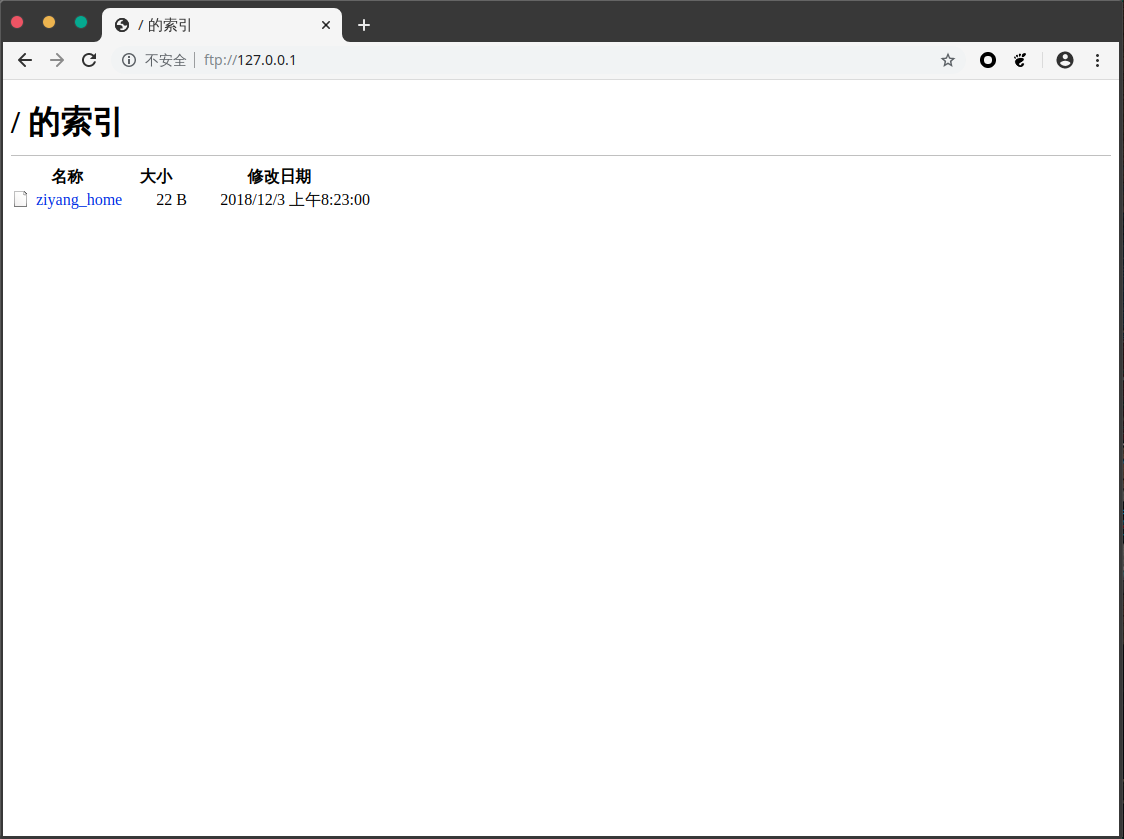


1. 结尾工作

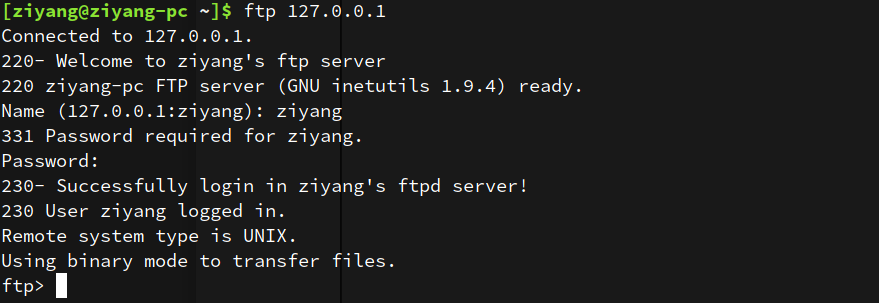
在～/chroot/etc中创建ftpusers文件，将用户exp添加进去，再登录后，就会提示登录错误：



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：



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

#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。