**实验一报告**

姓名：黄禧敏

学号：57119109

时间：2021.7.6

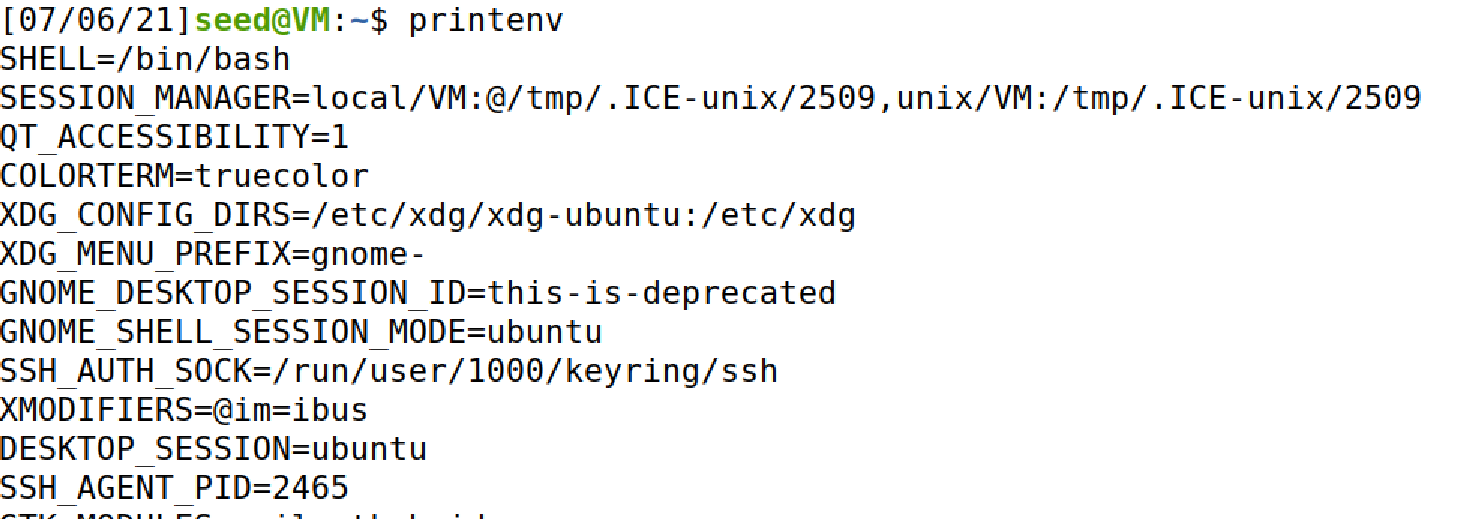
**2.1**

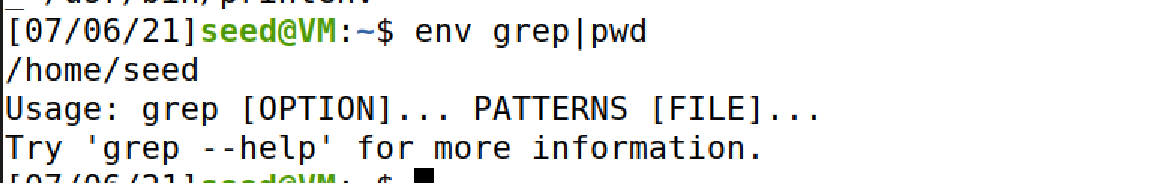
Task:

Use printenv or env command to print out the environment variables. If you are interested in some particular environment variables, such as PWD, you can use "printenv PWD" or "env |grep PWD".

Use export and unset to set or unset environment variables. It should be noted that these two commands are not seperate programs; they are two of the Bash’s internal commands (you will not be able to find them outside of Bash)

Result:





打印环境变量。

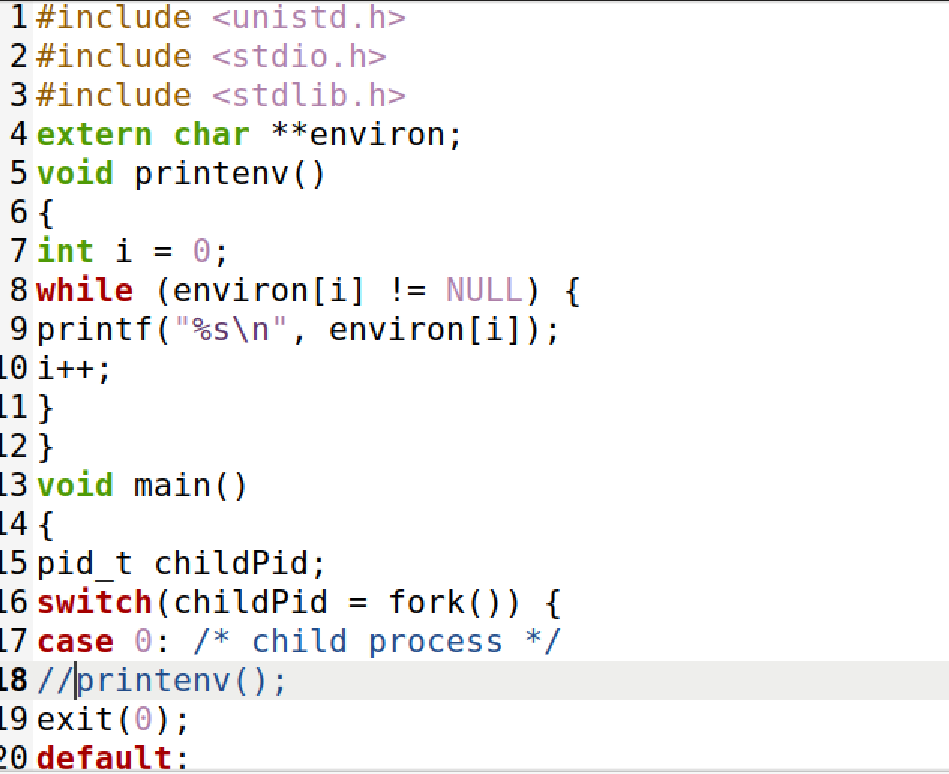
**2.2**

Task:

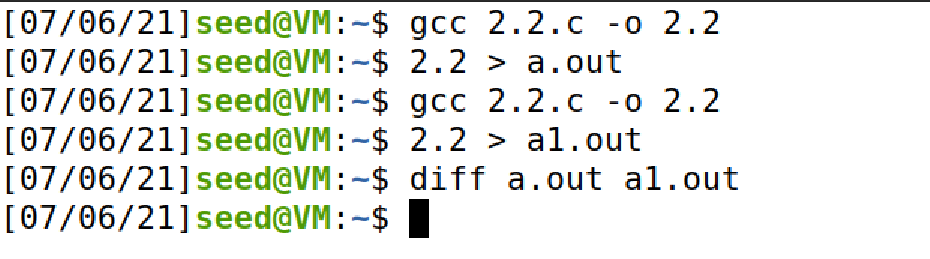
Passing Environment Variables from Parent Process to Child Process

In this task, we study how a child process gets its environment variables from its parent. In Unix, fork() creates a new process by duplicating the calling process. The new process, referred to as the child, is an exact duplicate of the calling process, referred to as the parent; however, several things are not inherited by the child (please see the manual of fork() by typing the following command: man fork). In this task, we would like to know whether the parent’s environment variables are inherited by the child process or not.

Program:



Result:



父进程与子进程打印环境变量完全一致。

**2.3**

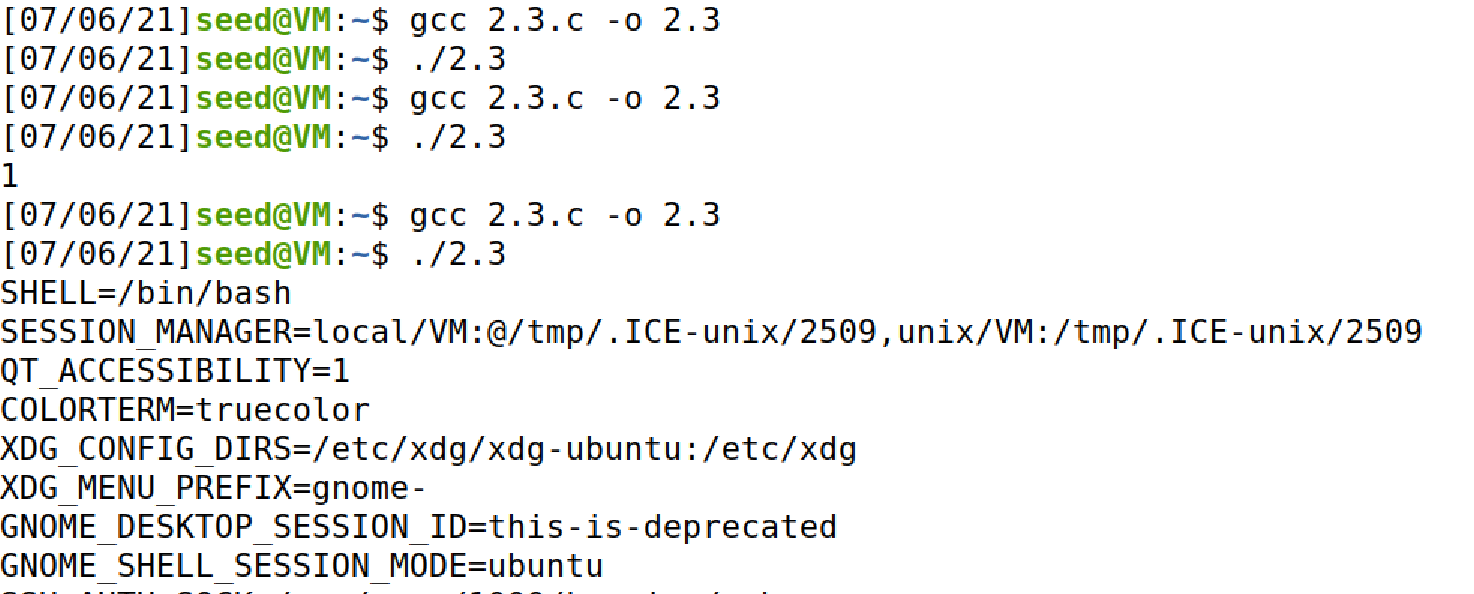
Task:

In this task, we study how environment variables are affected when a new program is executed via execve().The function execve() calls a system call to load a new command and execute it; this function never returns. No new process is created; instead, the calling process’s text, data, bss, and stack are overwritten by that of the program loaded. Essentially, execve() runs the new program inside the calling process. We are interested in what happens to the environment variables; are they automatically inherited by the new program?

Program and result:

execve()的函数原型为：int execve(const char \*filename, char \*const argv[ ], char \*const envp[ ]);

第一个参数为执行程序的文件路径，第二个参数argv[]数组用于向可执行程序传参，第三个参数envp[]用于传递新环境变量数组。



environ为环境参数，env命令打印当前环境变量。

可以看出打印的环境变量完全由传递的参数envp[]决定。

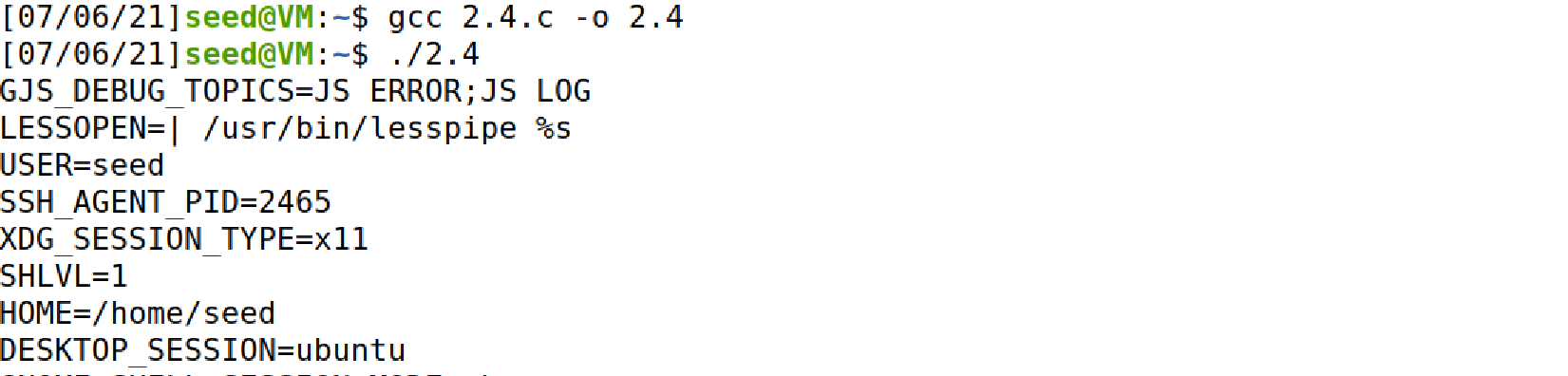
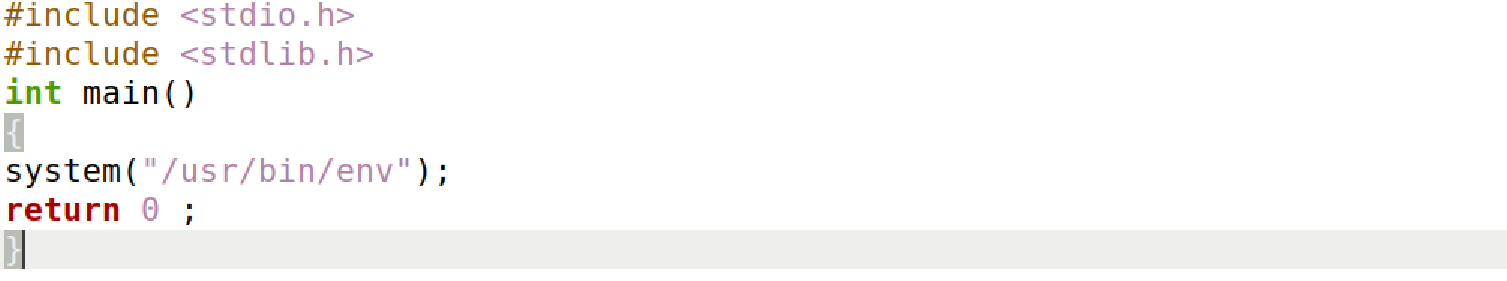
**2.4**

Task:

In this task, we study how environment variables are affected when a new program is executed via the system() function. This function is used to execute a command, but unlike execve(), which directly executes a command, system() actually executes "/bin/sh -c command", i.e., it executes /bin/sh, and asks the shell to execute the command.

If you look at the implementation of the system() function, you will see that it uses execl() to execute /bin/sh; execl() calls execve(), passing to it the environment variables array. Therefore,using system(), the environment variables of the calling process is passed to the new program /bin/sh.Please compile and run the following program to verify this.

Program and result:



系统调用函数，执行system()路径中的可执行程序。

**2.5**

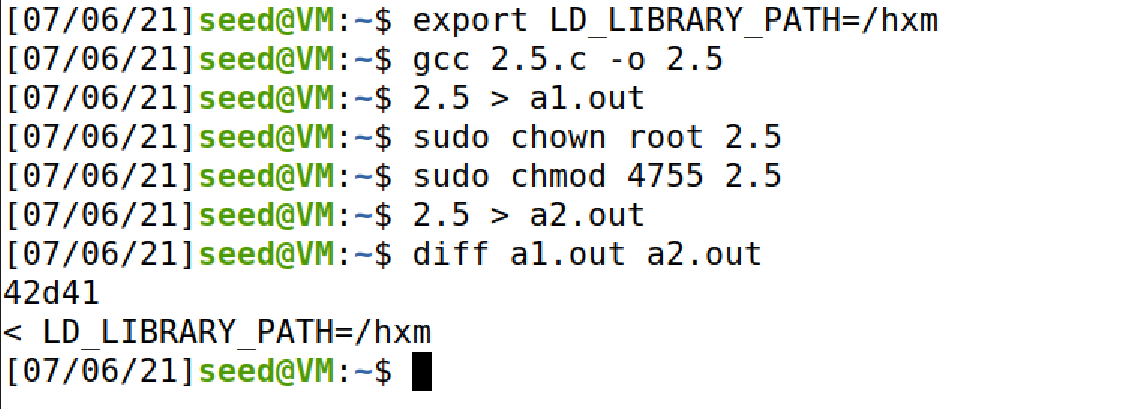
Task:

Set-UID is an important security mechanism in Unix operating systems. When a Set-UID program runs, it assumes the owner’s privileges. For example, if the program’s owner is root, then when anyone runs this program, the program gains the root’s privileges during its execution. Set-UID allows us to do many interesting things, but it escalates the user’s privilege when executed, making it quite risky. Although the behaviors of Set-UID programs are decided by their program logic, not by users, users can indeed affect the behaviors via environment variables. To understand how Set-UID programs are affected, let us first figure out whether environment variables are inherited by the Set-UID program’s process from the user’s

process.

Program and result:

打印环境变量。SET-UID，Compare parent process and child process



Child process 与 parent process相比，少了LD\_LIBRARY中新加的环境变量/hxm，只有将setuid程序还原后，才能使用设置的LD\_LIBRARY。

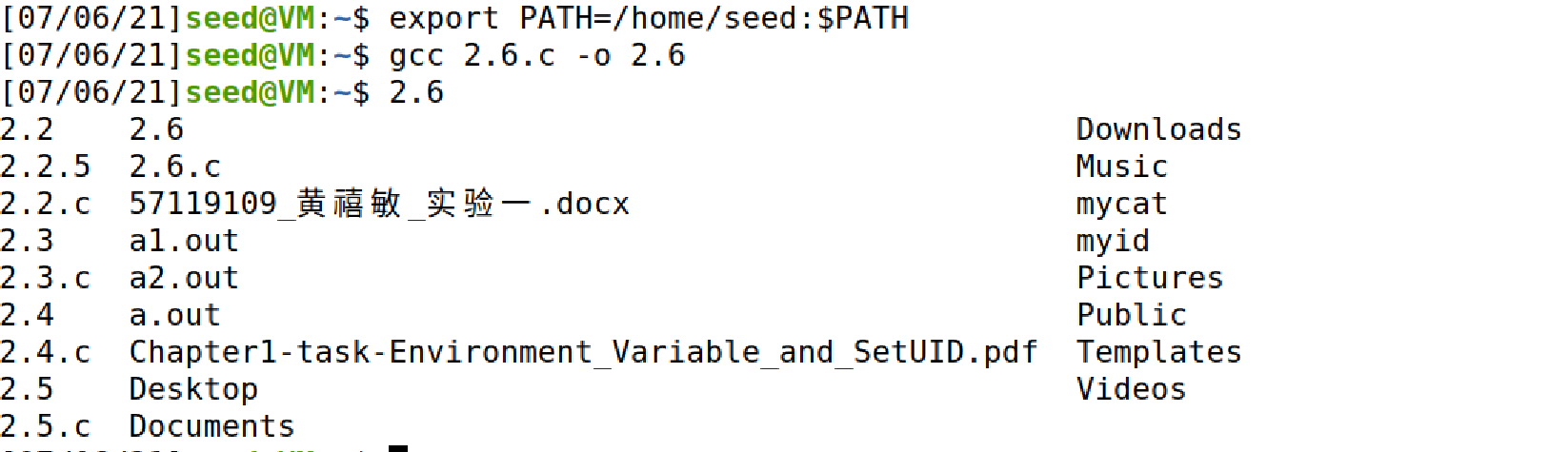
**2.6**

Task:

The PATH Environment Variable and Set-UID Programs

Program and result:

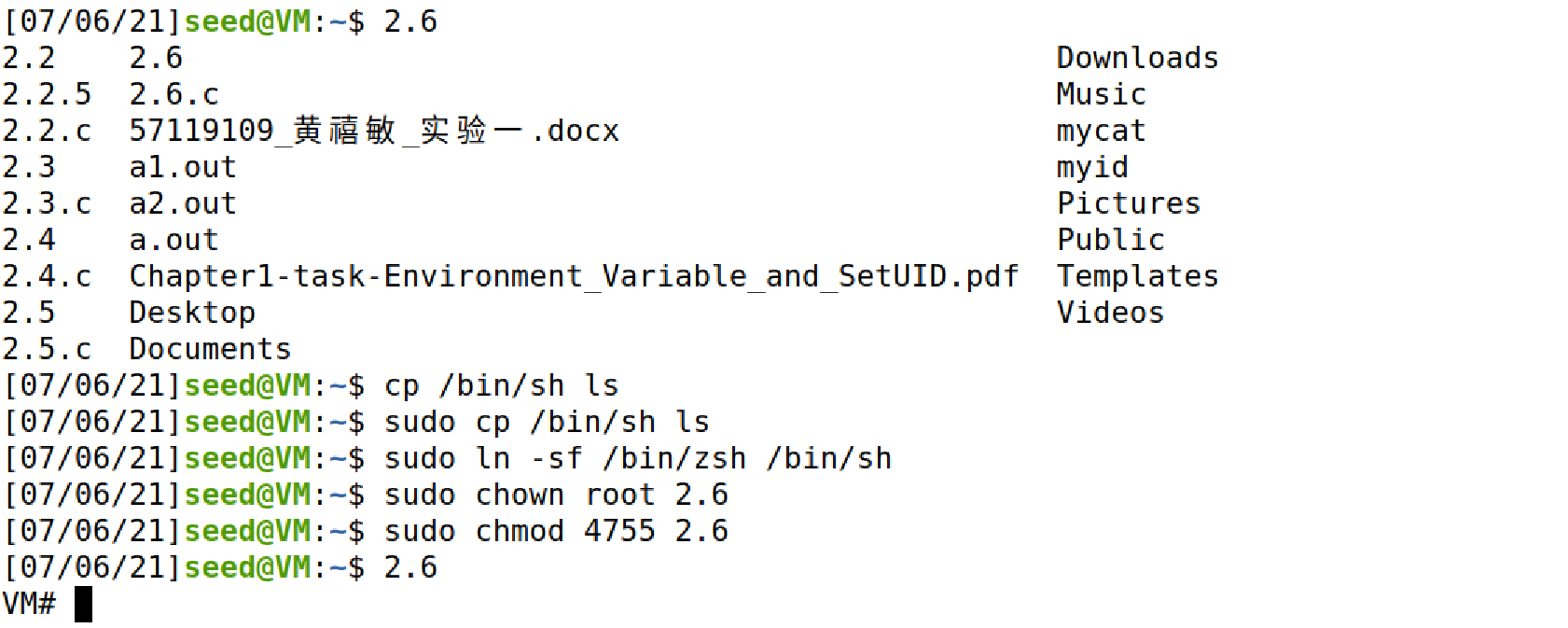
直接编译运行源程序，显示当前目录（为相对路径）



可以运行程序使其拥有root权限。

将/bin/sh 复制到源程序所在目录,Dash对SET-UID程序有保护机制，将sh链接到zsh:

更改路径为当前路径,将2.6更改为SET-UID 程序,运行即可获得root权限。



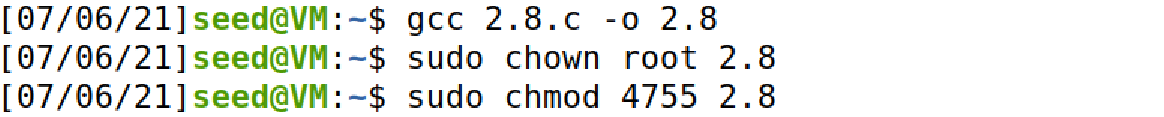
换回dash则无法获得root权限。

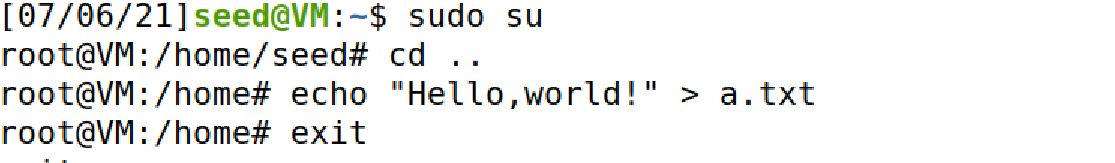
**2.8**

Task:

Invoking External Programs Using system() versus execve().

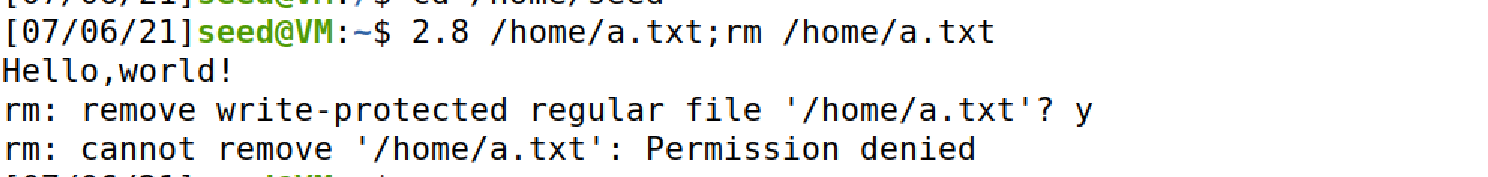
Program and result:



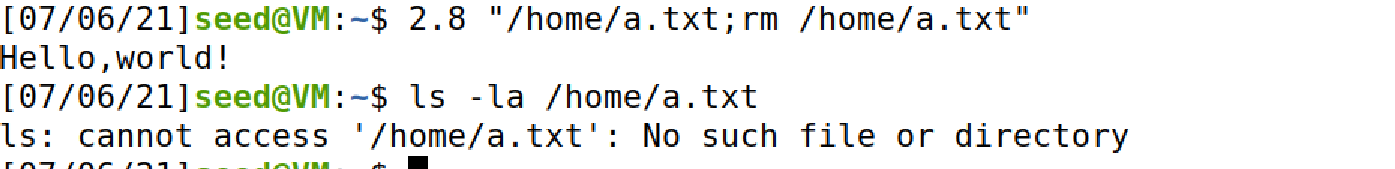


将2.8设置为SETUID程序，将a.txt设置为root文件

Step1:



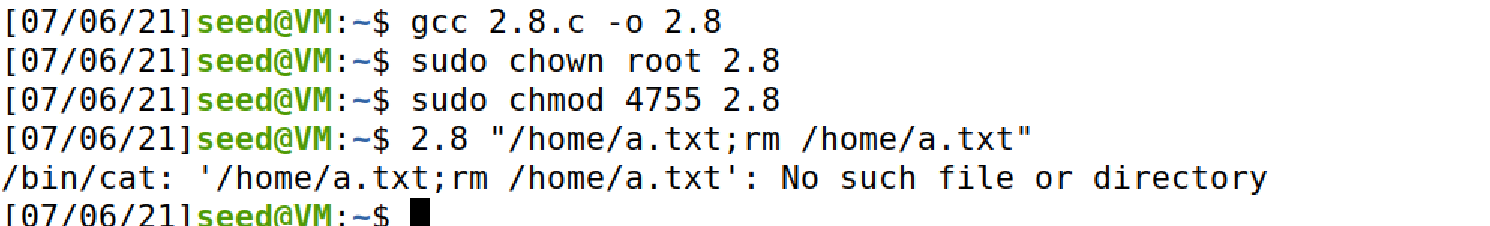
不加引号时，可以用system读取文件内容，不可以删除文件，命令解析第二条指令仍为seed用户下。



加引号后，引号内绑定为同一个参数传入system中，system解析为两个命令，可以在setuid下删除。

文件安全性无法保证，Bob可以通过rm命令remove原文件。

Step2:



无法通过输入rm命令remove文件。

在system()中，执行完一条命令后会继续执行下一条命令，而execve()则将”2.7.1.txt;rm 2.7.1.txt”看作一整个文件名，因而会显示不存在此文件，文件安全性可以得到保证。

**Summary:**

本次实验主要练习chown提权，setuid程序的使用，利用特权程序漏洞获取特殊权限。在seed权限的文件夹下的所有文件都可以通过shell直接完成删除等操作，但在其它权限文件夹下无权限文件则需程序提权操作。