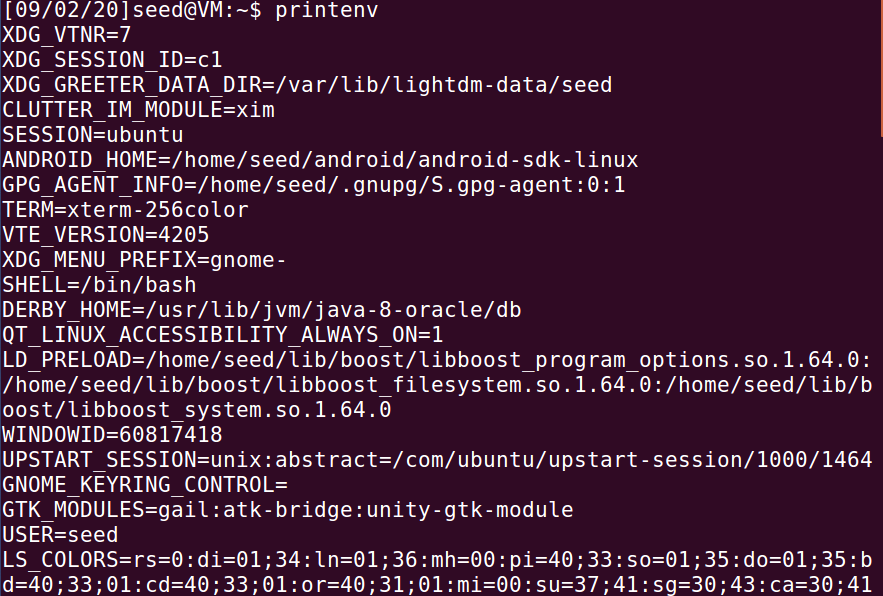
Lab1-report

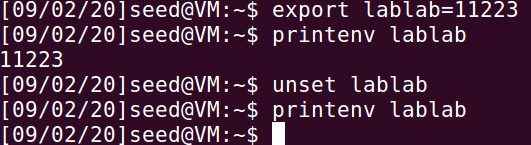
57118113 蔡义涵

Task1：

1. Use printenv or env command to print out the environment variables

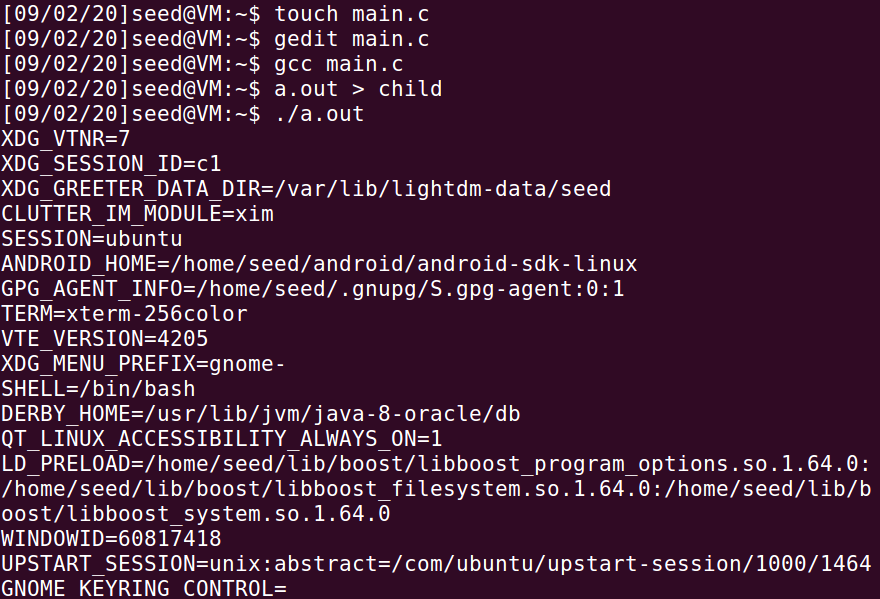


1. Use export and unset to set or unset environment variables

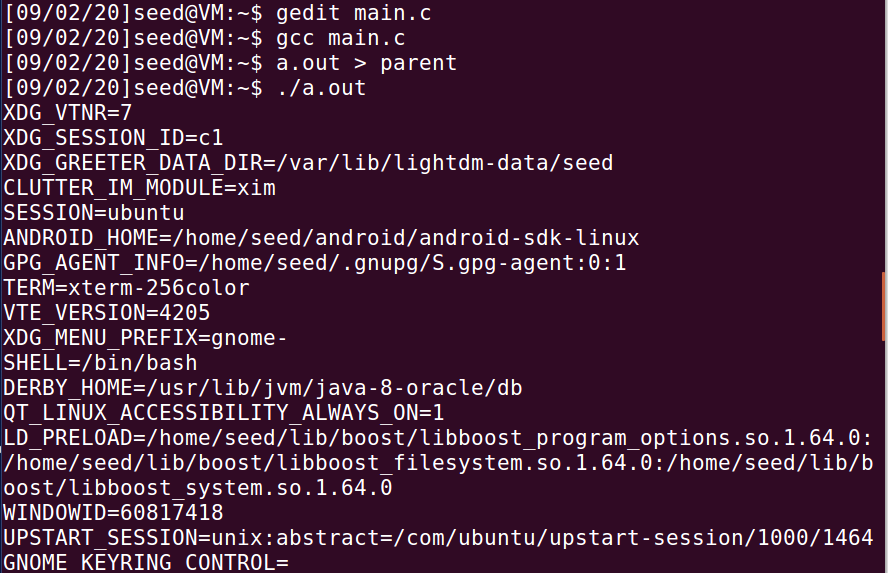


Task2：

1. Please compile and run the following program, and describe your observation



1. Now comment out the printenv() statement in the child process case(Line 1), and uncomment the printenv() statement in the parent process case (Line 2). Compile and run the code again, and describe your observation. Save the output in another file.



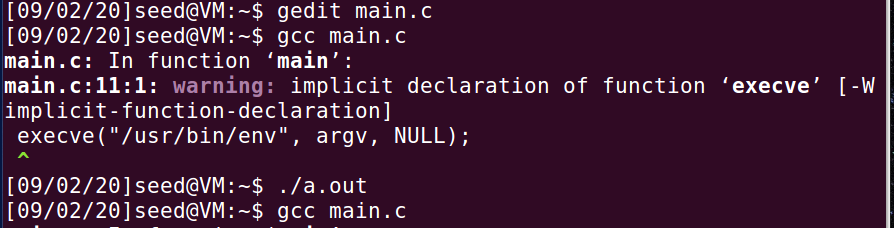
1. Compare the difference of these two files using the diff command. Please draw your conclusion.



Diff指令无输出，说明两文件内容一致，说明子进程完全继承了父进程的环境变量。

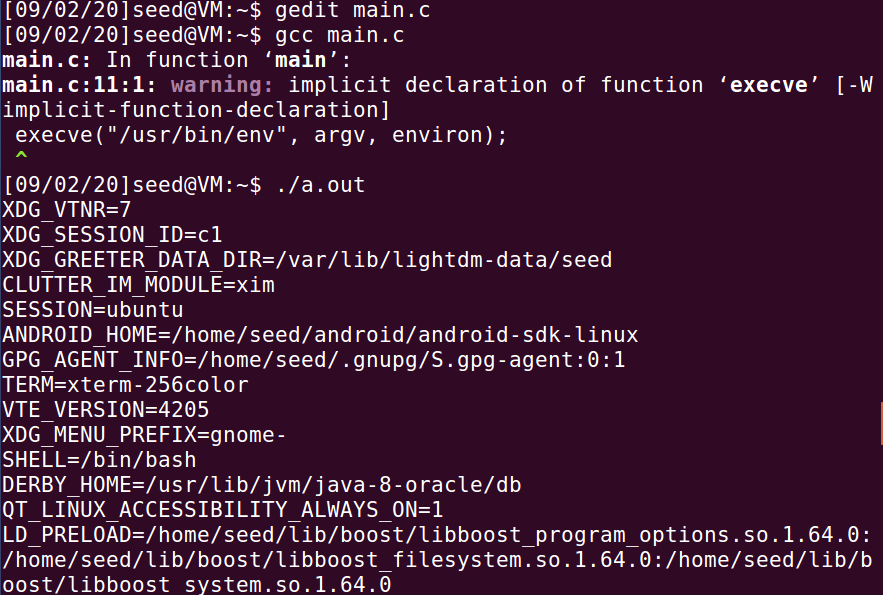
Task3：

1. Please compile and run the following program, and describe your observation. This program simply executes a program called /usr/bin/env, which prints out the environment variables of the current process.



无输出

1. Change the invocation of execve() in Line1 to the following; describe your observation.



成功输出环境变量

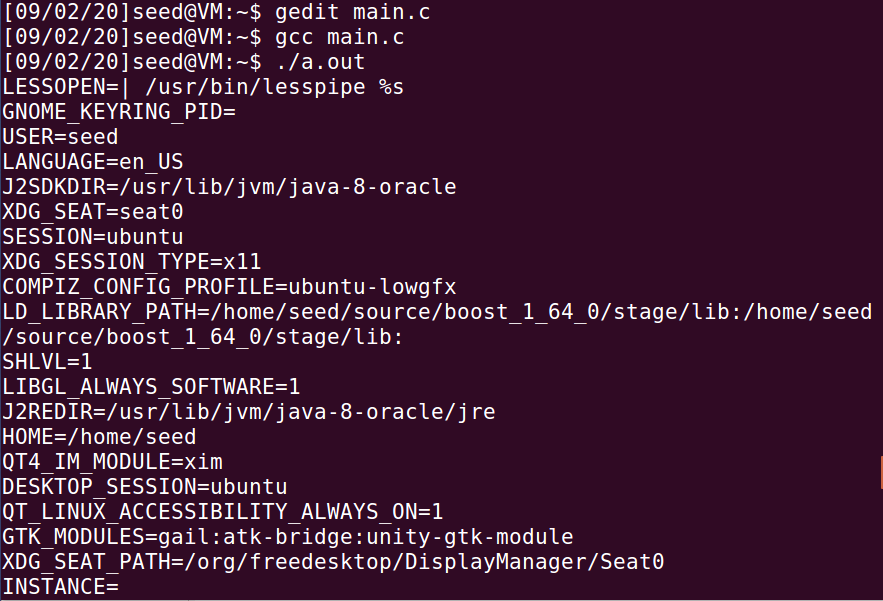
1. Please draw your conclusion regarding how the new program gets its environment variables.

execve()函数创建进程时，进程的环境变量由函数的第三个参数决定。只有当第三个参数为environ时，进程产生输出，表明进程环境变量为全局的环境变量。

Task4：

1. 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.

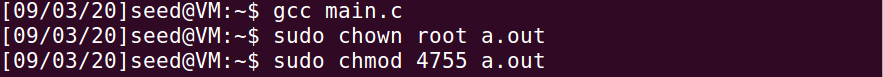
程序运行如下：



成功输出环境变量，证明system函数运行过程中通过execl调用了execve()函数。

Task5：

1. Write the following program that can print out all the environment variables in the current process.
2. Compile the above program, change its ownership to root, and make it a Set-UID program.

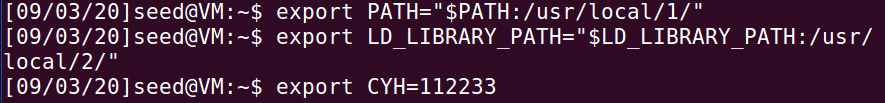


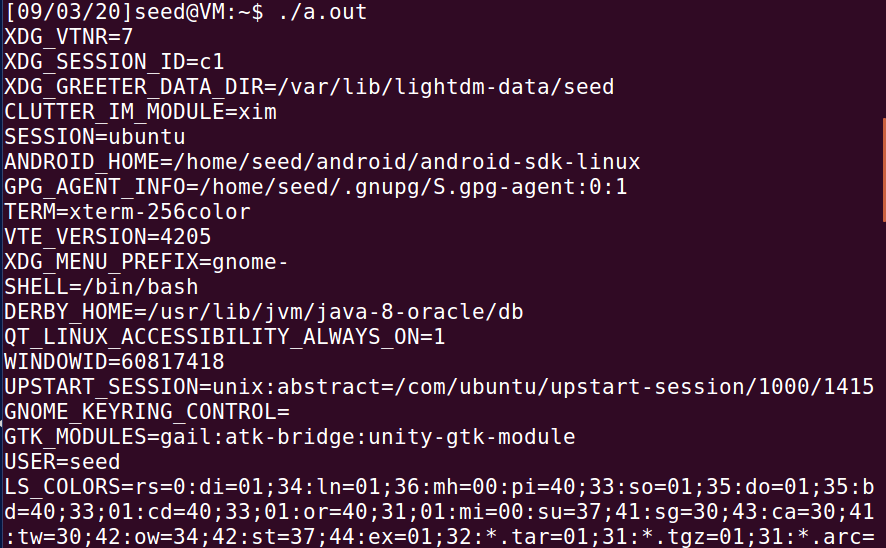
1. In your shell (you need to be in a normal user account, not the root account), use the export command to set the following environment variables (they may have already exist):

• PATH

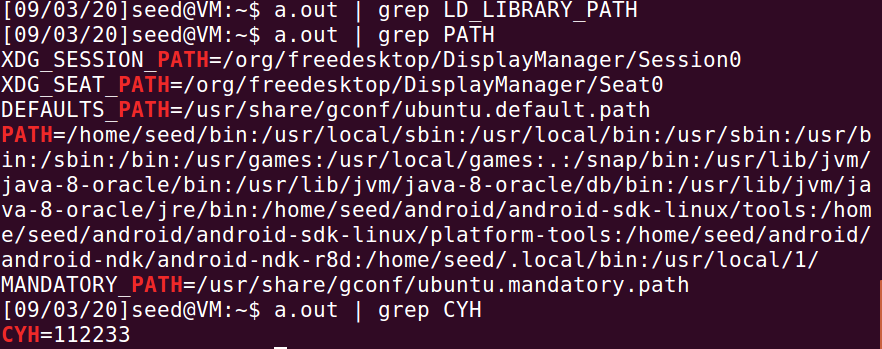
• LD LIBRARY PATH

• ANY NAME (this is an environment variable defined by you, so pick whatever name you want).





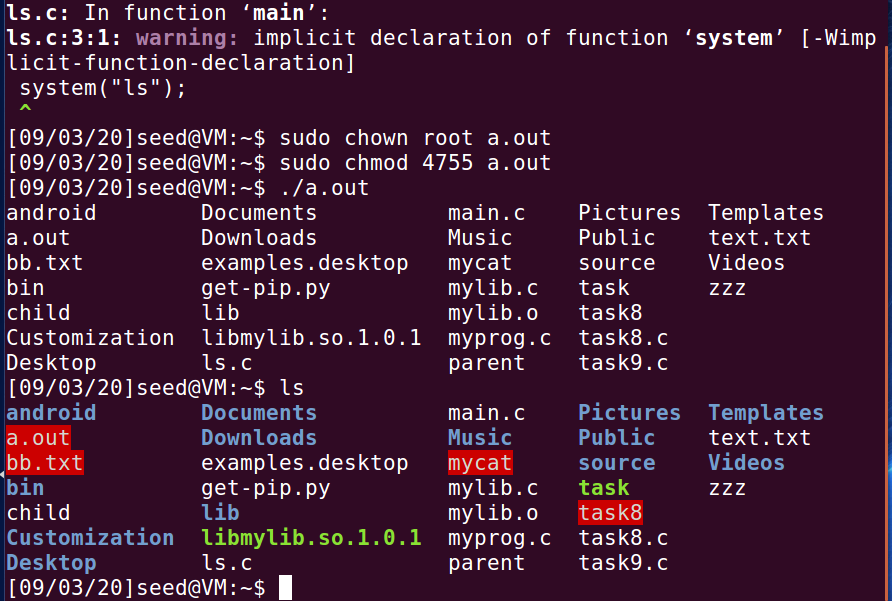
修改后：



结论：setuid子进程可以继承用户进程的PATH环境变量和自定义环境变量CYH，但是不能继承LD\_LIBRARY\_PATH环境变量。

Task6：

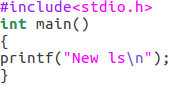
1. Please compile the above program, and change its owner to root, and make it a Set-UID program. Can you let this Set-UID program run your code instead of /bin/ls? If you can, is your code running with the root privilege? Describe and explain your observations.

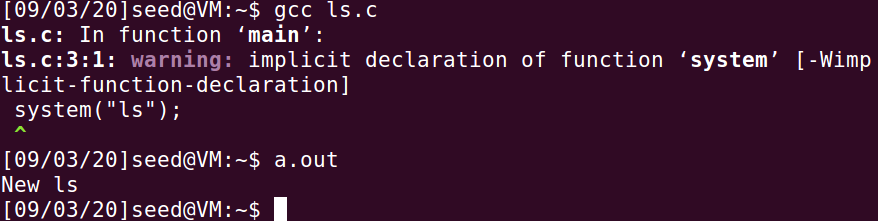


程序运行结果与命令行运行结果一致，证明程序运行的是bin/ls

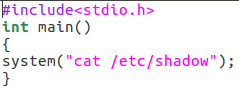
通过以下方式，我们可以运行用户所写的ls代码。

将如下的ls.ci添加到到/bin目录下，再次运行以上程序：





若使用root权限再改写/bin目录下ls程序为



再次运行原程序，得到

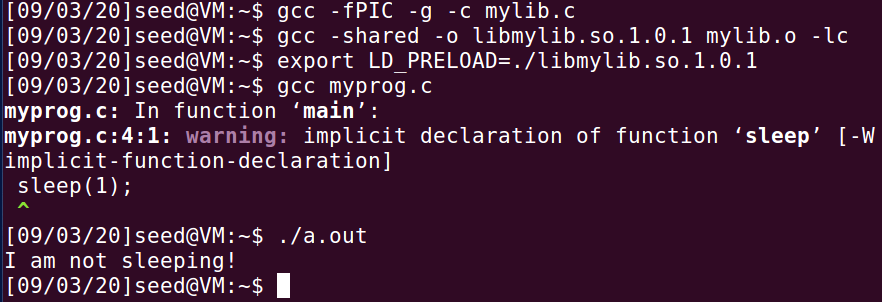


原因：Ubuntu 16.04中的dash程序会阻止自己执行一个Set-UID进程，如果检测到正在执行一个Set-UID进程，dash会马上将EUID降级成RUID。

Task7：

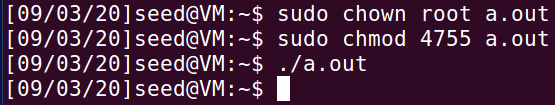
1. First, we will see how these environment variables influence the behavior of dynamic loader/linker when running a normal program.
2. After you have done the above, please run myprog under the following conditions, and observe what happens.

• Make myprog a regular program, and run it as a normal user.



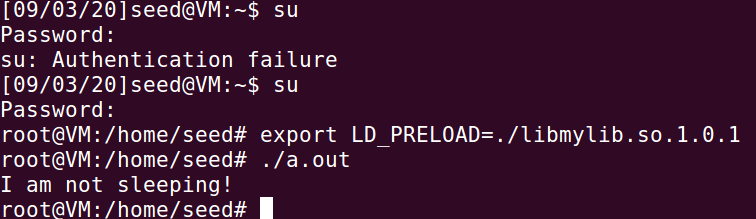
运行的是动态链接库定义的sleep函数

• Make myprog a Set-UID root program, and run it as a normal user.



运行的是系统自带的sleep函数

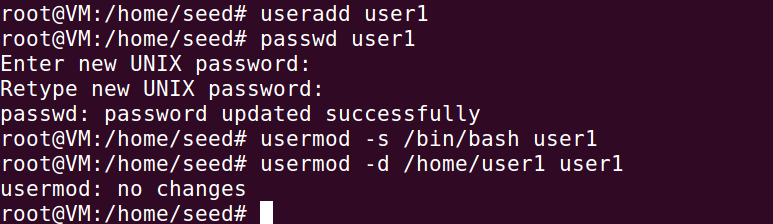
• Make myprog a Set-UID root program, export the LD PRELOAD environment variable again in the root account and run it.



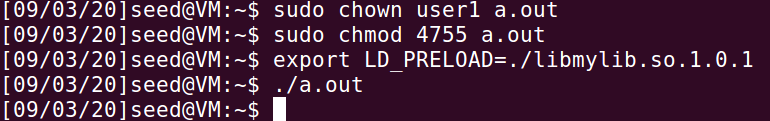
运行的是动态链接库定义的sleep函数

• Make myprog a Set-UID user1 program (i.e., the owner is user1, which is another user account), export the LD PRELOAD environment variable again in a different user’s account (not-root user) and run it.

创建一个新用户user1：



回到seed用户，并修改程序的owner和环境变量LD\_PRELOAD，并运行



运行的是系统自带的sleep函数

1. You should be able to observe different behaviors in the scenarios described above, even though you are running the same program. You need to figure out what causes the difference. Environment variables play a role here. Please design an experiment to figure out the main causes, and explain why the behaviors in Step 2 are different. (Hint: the child process may not inherit the LD \* environment variables).

在运行普通程序时，myprog继承了shell中定义的环境变量LD\_PRELOAD。

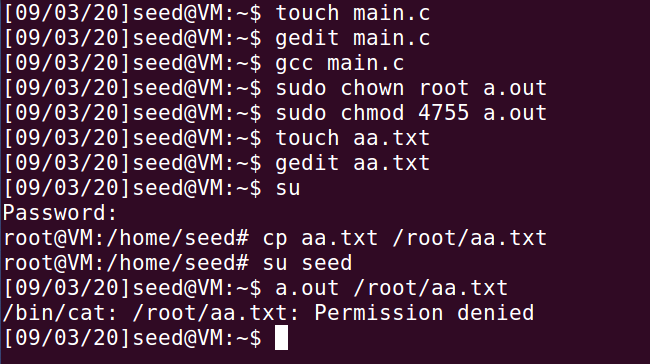
当以普通用户身份运行set-UID程序时，程序的子进程不会继承父进程动态链接库LD\_PRELOAD环境变量。

当以root的身份运行程序时，shell向myprog传递了LD\_PRELOAD环境变量。

最后，当以seed普通用户的身份运行user1的程序时，不会向user1传递在seed中修改的环境变量LD\_PRELOAD。

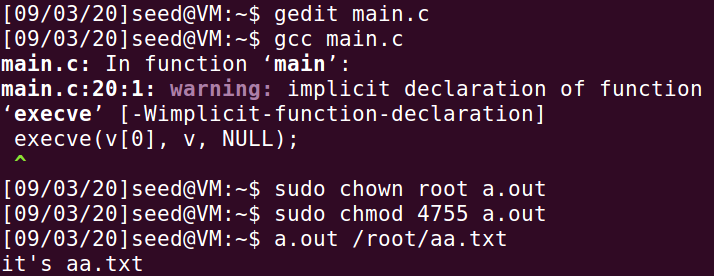
Task8：

1. Compile the above program, make it a root-owned Set-UID program. The program will use system() to invoke the command. If you were Bob, can you compromise the integrity of the system? For example, can you remove a file that is not writable to you?



seed用户无法查看root目录下的aa.txt

1. Comment out the system(command) statement, and uncomment the execve() statement; the program will use execve() to invoke the command. Compile the program, and make it a root-owned Set-UID. Do your attacks in Step 1 still work? Please describe and explain your observations.



修改main.c后，seed账户可以查看/root/aa.txt文件

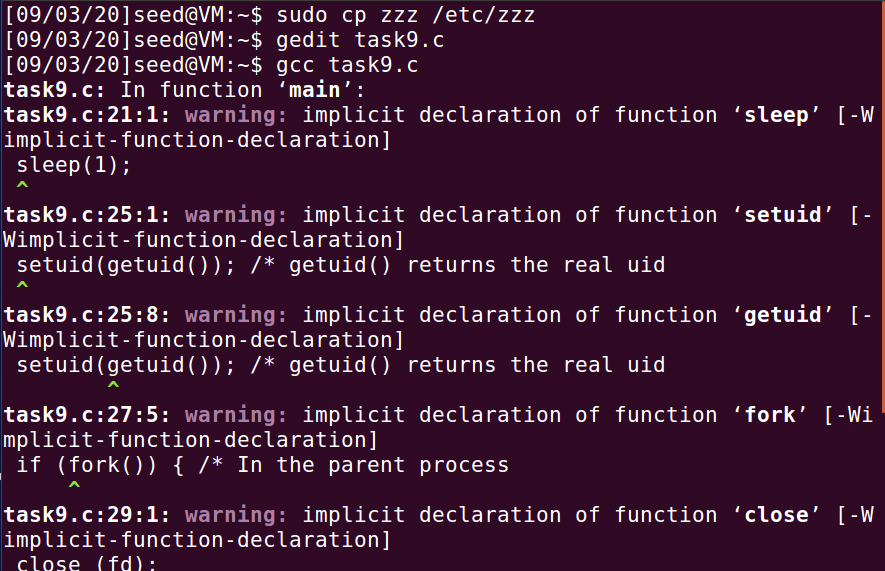


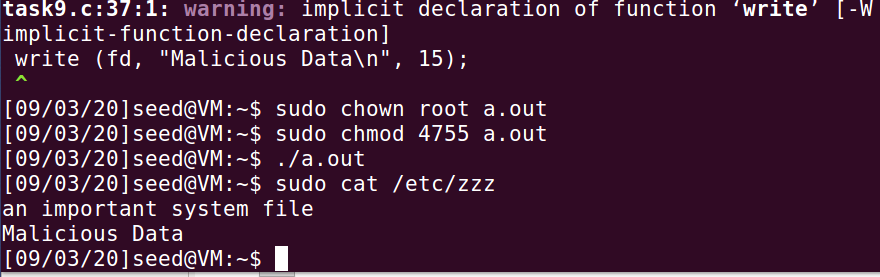
然而，普通用户无法删除root权限的aa.txt文件

Task9：

1. Compile the following program, change its owner to root, and make it a Set-UID program. Run the program as a normal user, and describe what you have observed. Will the file /etc/zzz be modified? Please explain your observation.

先创建zzz文件，写入“an important file”，再将zzz文件复制到etc目录下，运行a.out，再查看/etc/zzz，发现/etc/zzz文件被修改。





task9.c在运行时，通过setuid(getuid())抛弃了权限，但句柄fd并未关闭，因此程序能通过fd来修改文件。