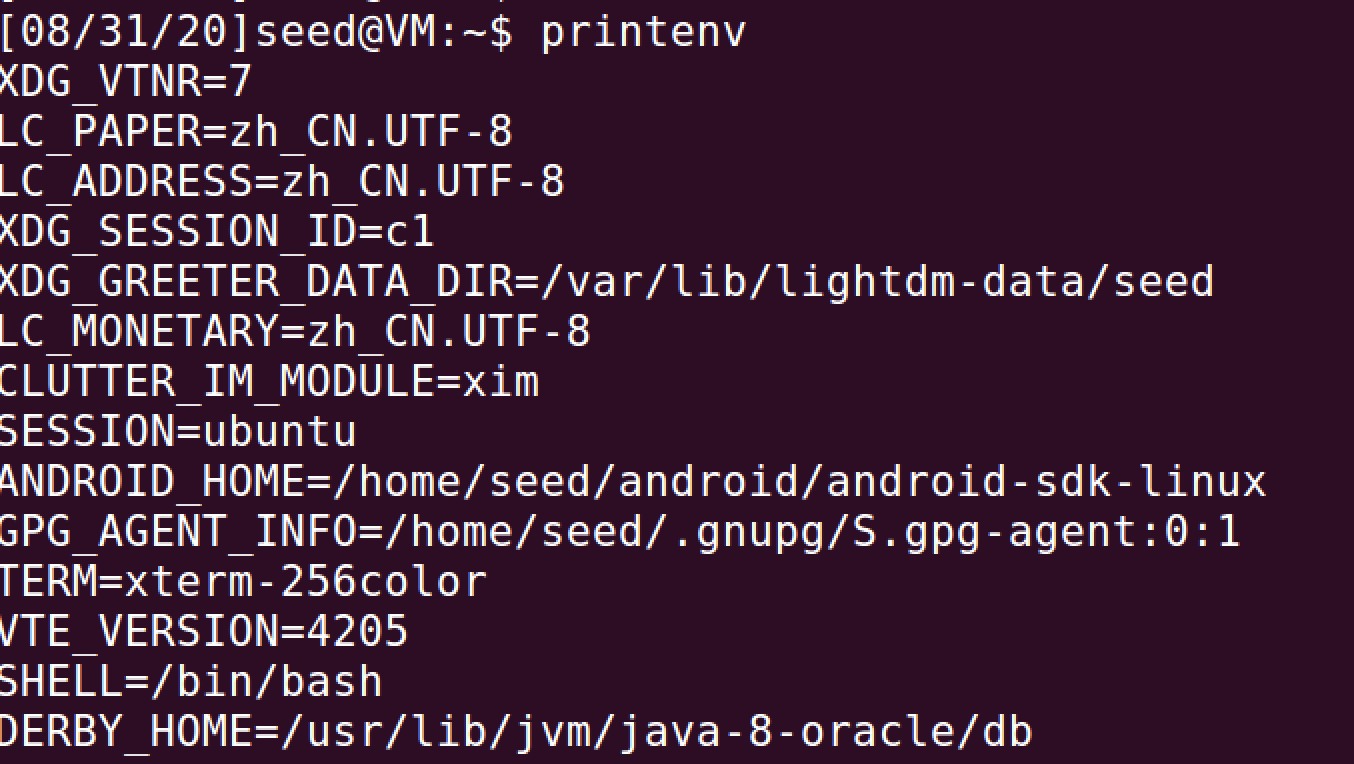
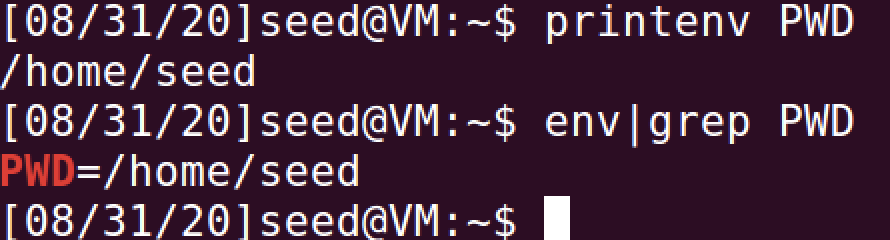
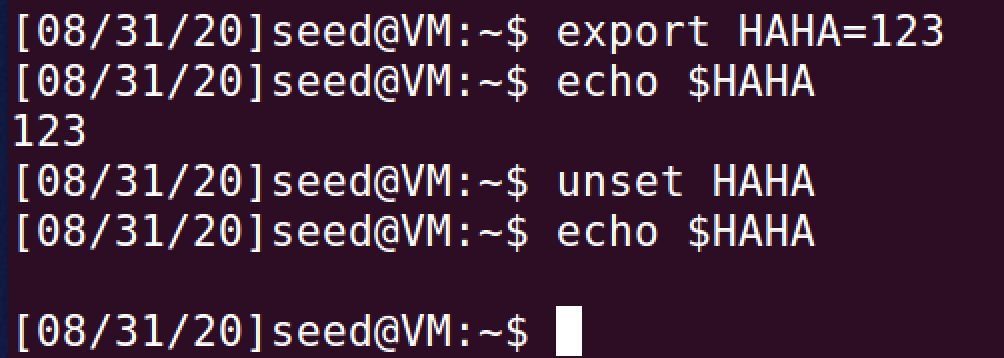
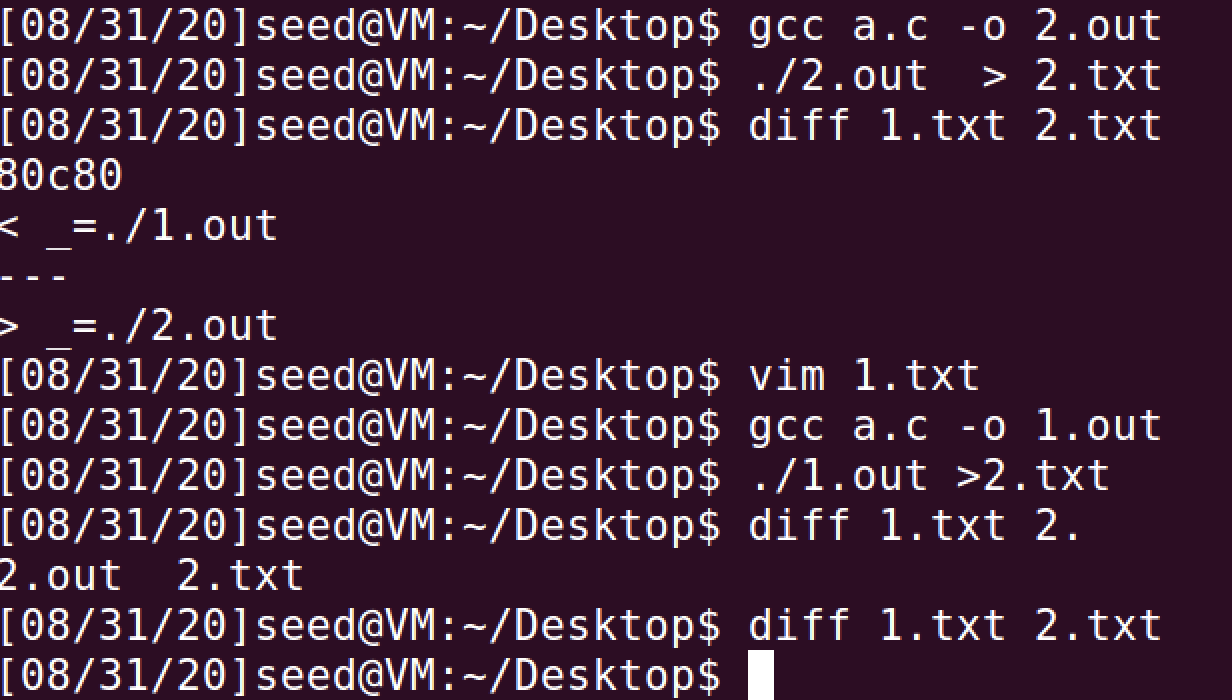
# lab1-report

57117227邵长捷

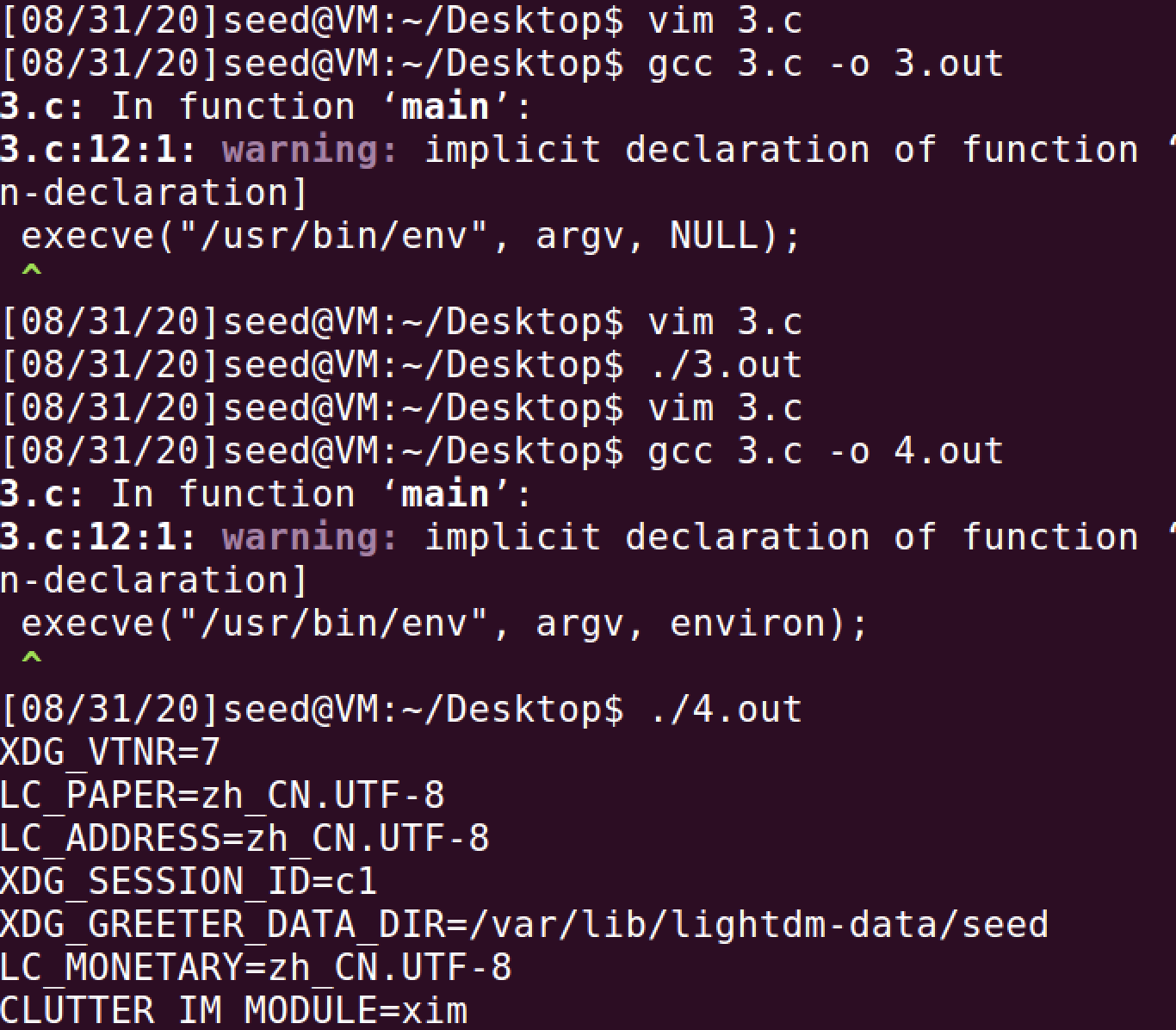
1. Manipulating Environment Variables
2. Try env and printenv
3. Use export and unset
4. Passing Environment Variables from Parent Process to Child Process

Conclusion

由图可见，fork出的子进程的环境变量由父进程的环境变量复制而来。

当两个编译生成的程序文件名分别为1.out和2.out时，在两个输出的txt文件的第80行会有差别，因为该行和程序文件名相关。

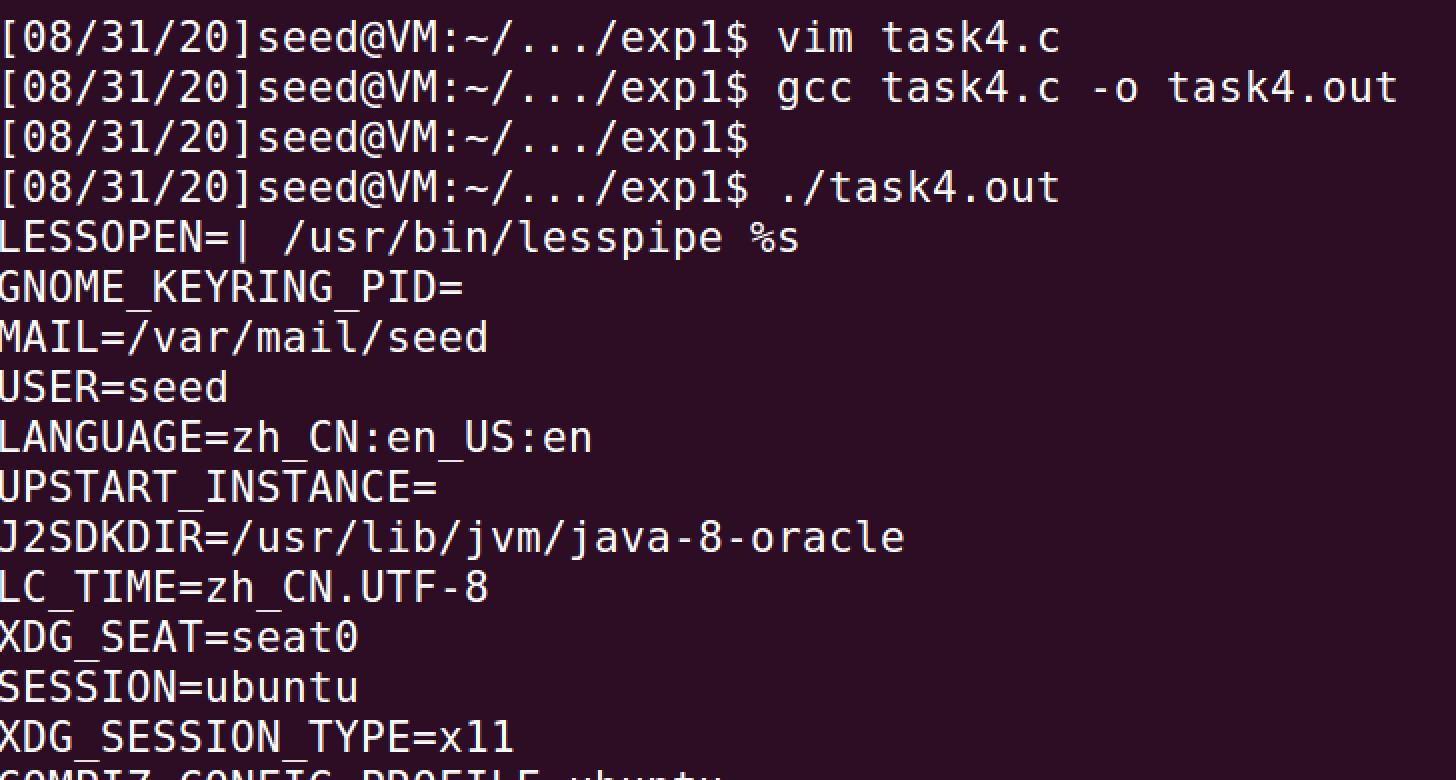
如果编译生成的程序名字一致时，二者的环境变量没有差别。

1. Environment Variables and execve()
2. execve()第三个参数为NULL
3. execve()第三个参数为environ

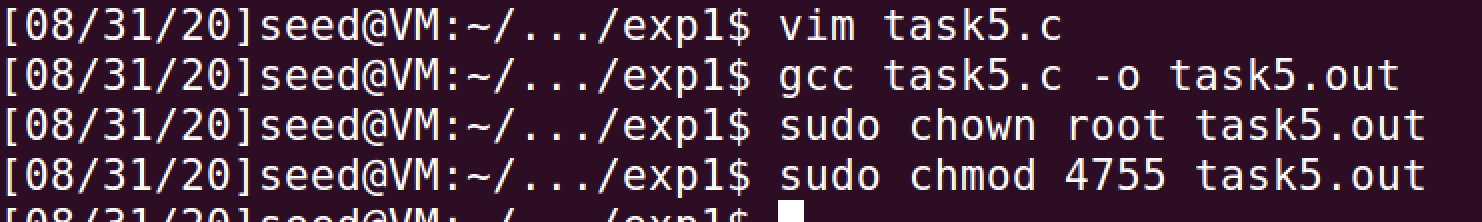
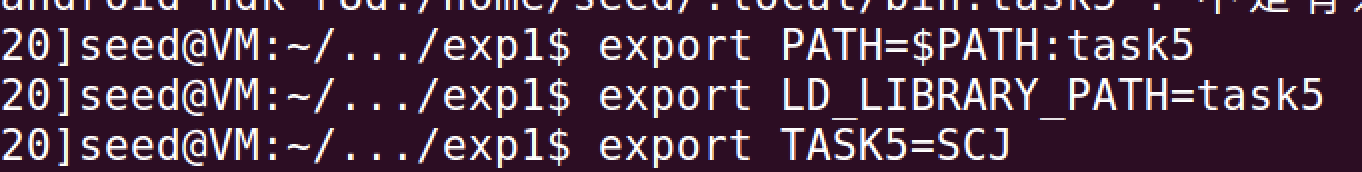
Conclusion

在使用execve()函数时，若不指定第三个参数，则运行的新程序不会继承该进程的环境变量，当传入全局变量environ时，运行的新程序则会继承原先的环境变量。

1. Environment Variables and system()

Conclusion

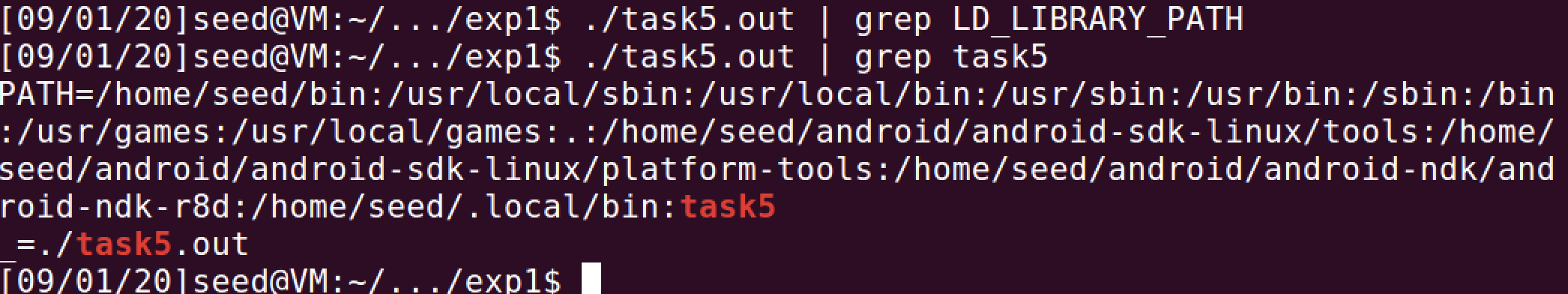
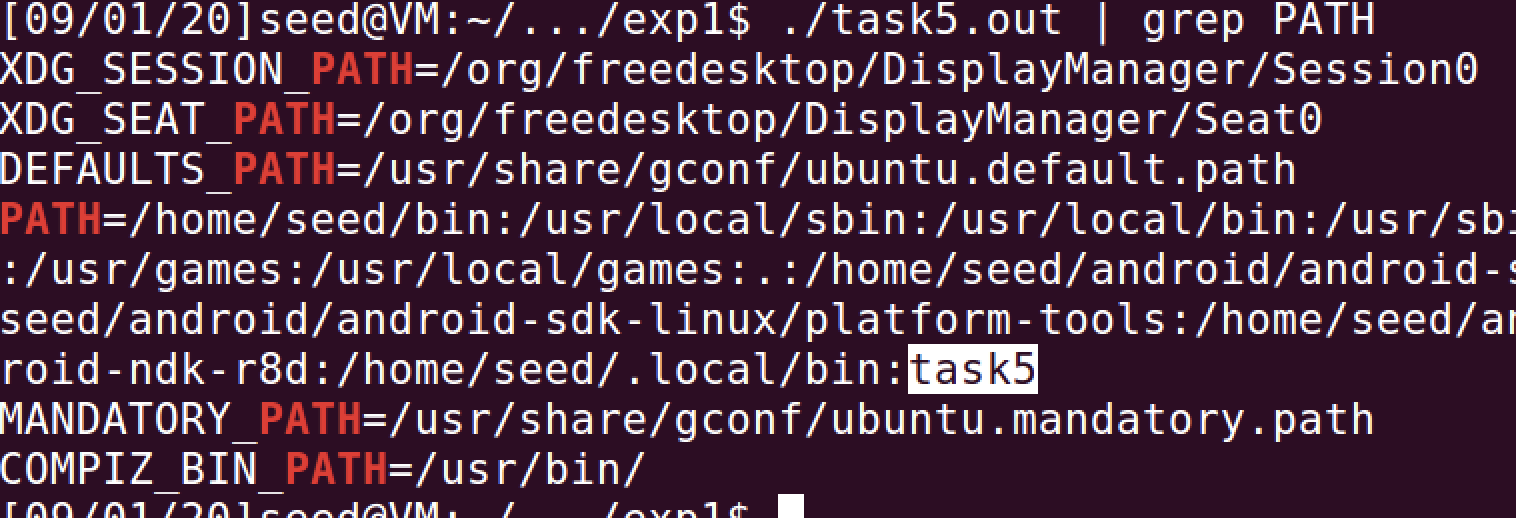
由图可见，编译并运行task4.out后，程序运行了env命令的功能，因而证明了system()函数会新建shell并运行指令的功能。

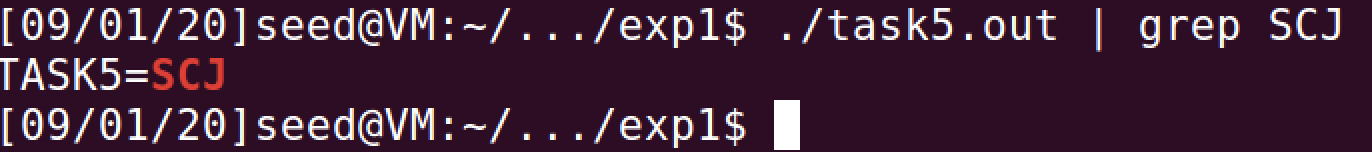
1. Environment Variable and Set-UID Programs
2. 编译task5.c
3. 更改程序拥有者并指定为Set-UID程序
4. 设置环境变量并检查

根据要求设置三个环境变量。

运行并查看相应的环境变量是否更新或生成。

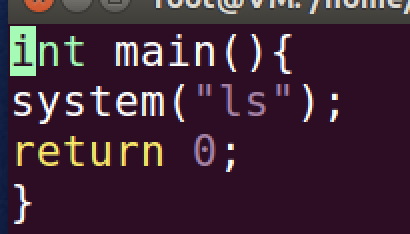
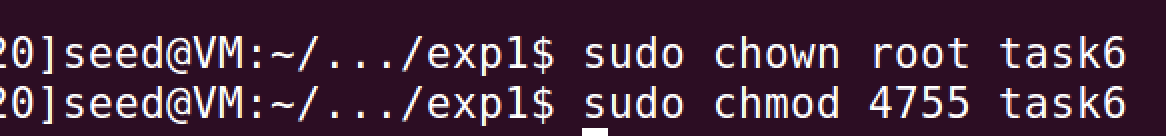
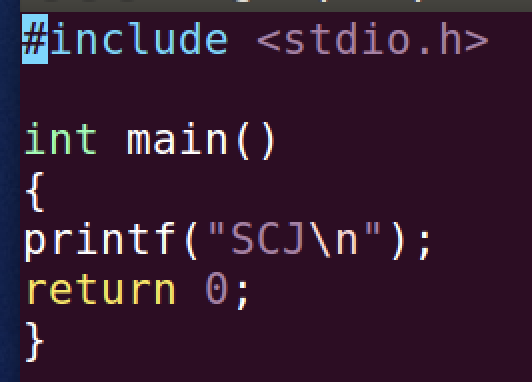
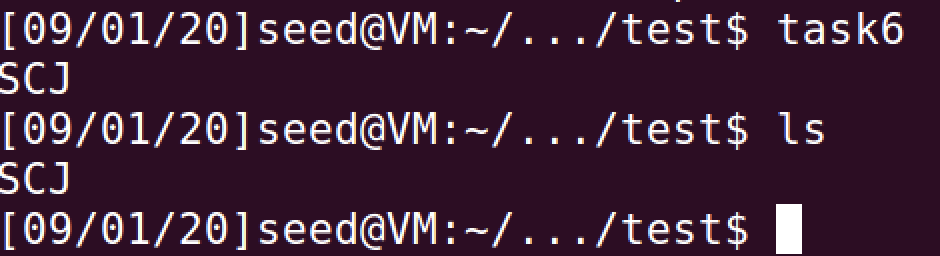
PATH成功更新：

LD\_LIBRARY\_PATH并没有更新：

自定义的变量TASK5成功生成：

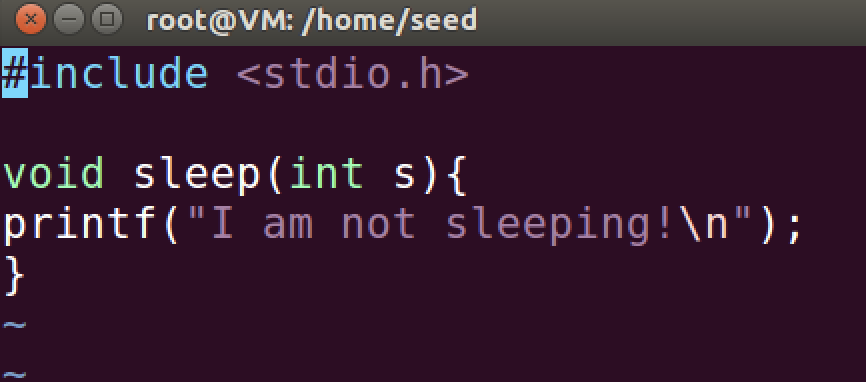
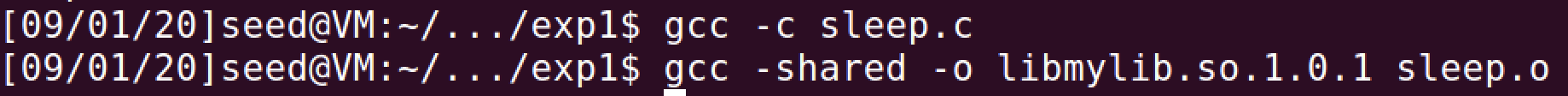
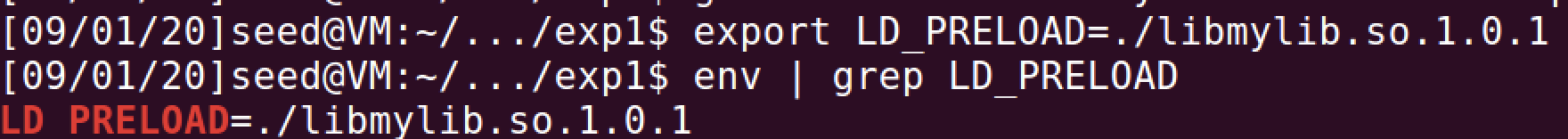
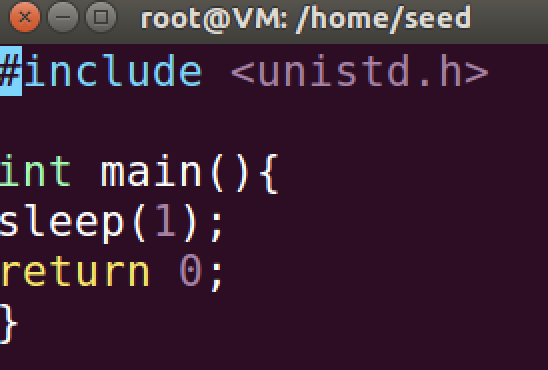
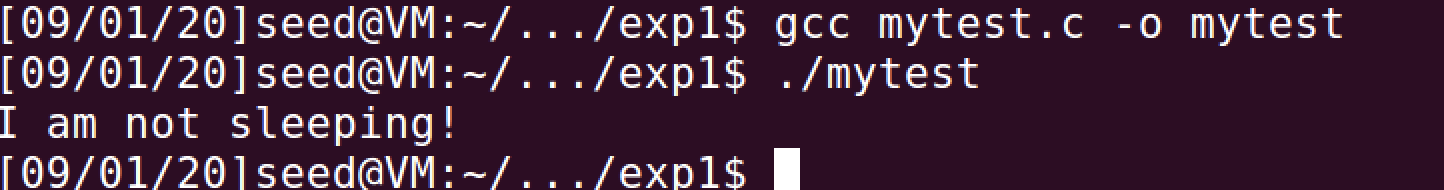
Conclusion

为了防止特殊环境变量对特权程序的动态链接产生任何影响，在运行特权程序task5.out的进程中并不包含LD\_LIBRARY\_PATH。

1. The PATH Environment Variable and Set-UID Programs
2. 编译上图程序为task6
3. 更改程序拥有者并指定为Set-UID程序
4. 在当前目录自定义ls()函数并编译
5. 更改PATH环境变量
6. 运行task6

Conclusion

实验结果表明，在PATH中加入当前目录时，将优先搜索当前目录中的可执行文件ls，从而实现对自定义ls的调用。

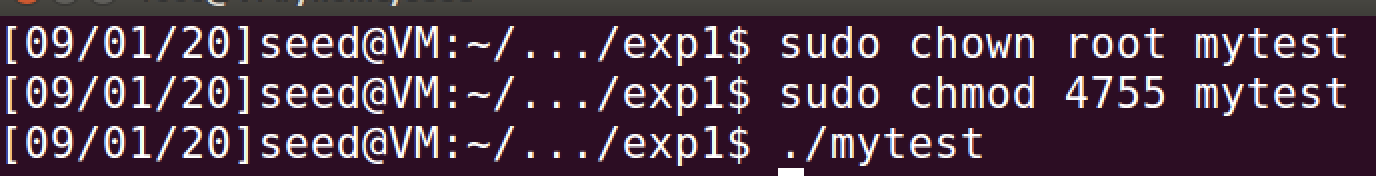
1. The LD PRELOAD Environment Variable and Set-UID Programs
2. 重写sleep()函数
3. 编译并创建新的共享库
4. 设置LD\_PRELOAD环境变量
5. 定义调用函数mytest()
6. 编译并运行mytest

成功调用了自定义的sleep()函数。

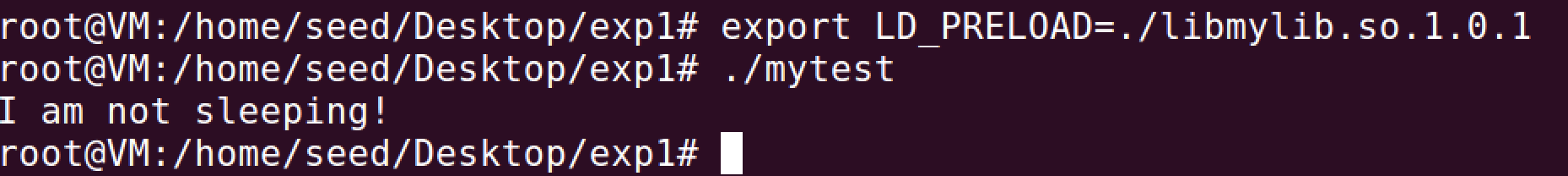
1. Make mytest a regular program, and run it as a normal user.

上述1-5步骤即为普通用户运行非特权程序mytest的情况。

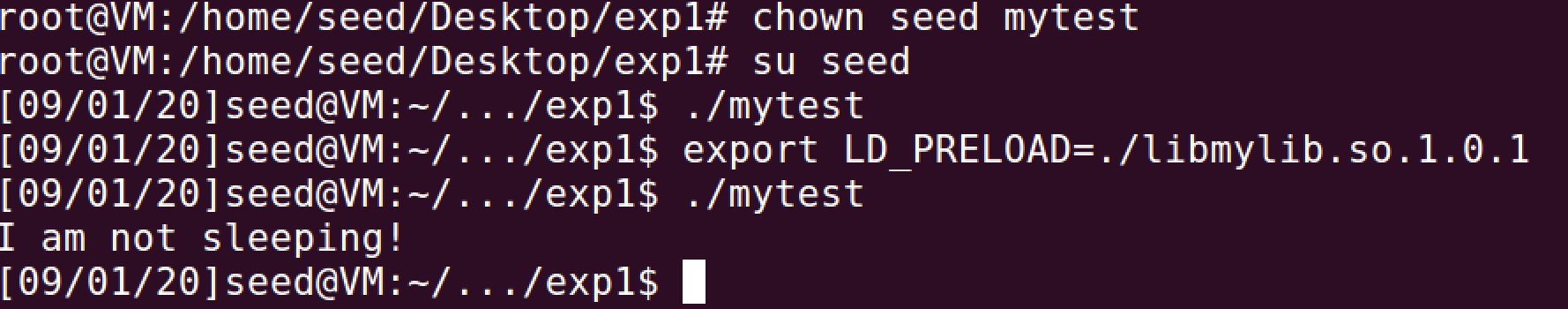
1. Make mytest a Set-UID root program, and run it as a normal user.

将mytest更改为特权程序后，运行时调用系统函数sleep()。

1. Make mytest a Set-UID root program, export the LD\_PRELOAD environment variable again in the root account and run it.

此时可以调用自定义的sleep()函数。

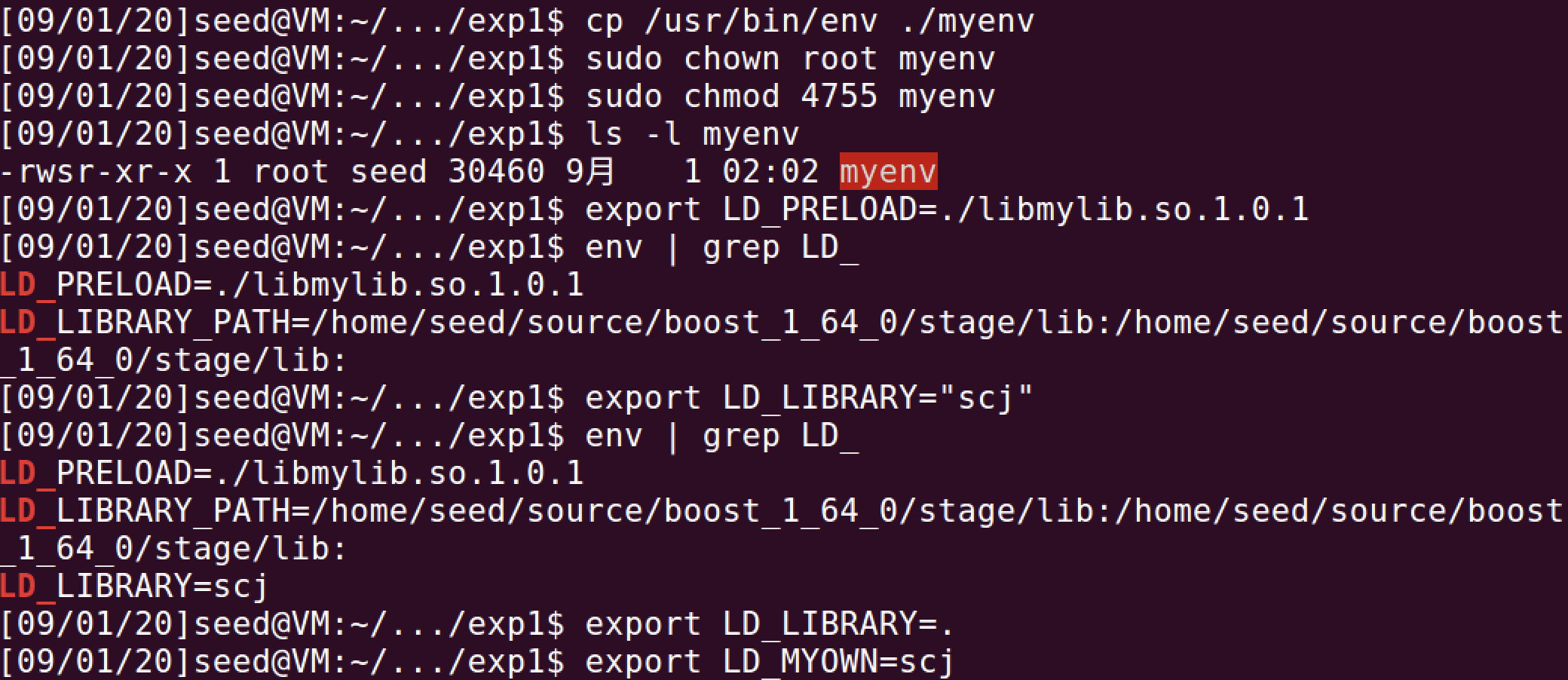
1. Make mytest a Set-UID user1 program (i.e., the owner is user1, which is another user account), export the LD PRELOAD environment variable again in a different user’s account (not-root user) and run

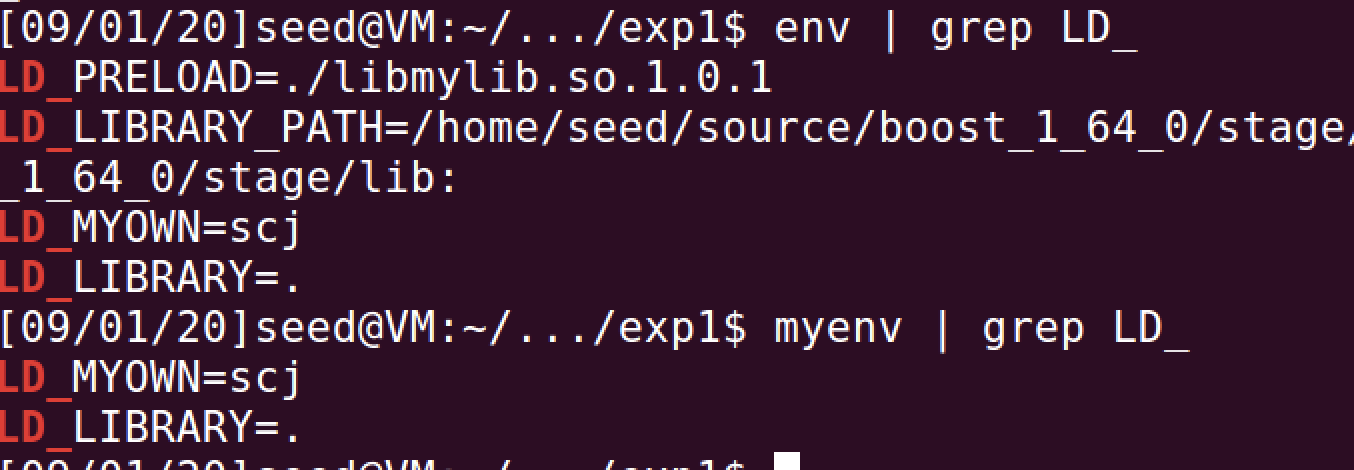
此时可以调用自定义的sleep()函数。

Conclusion

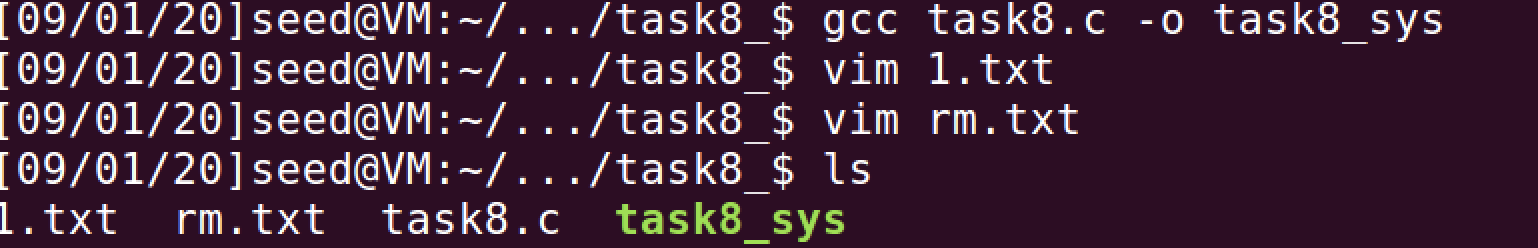
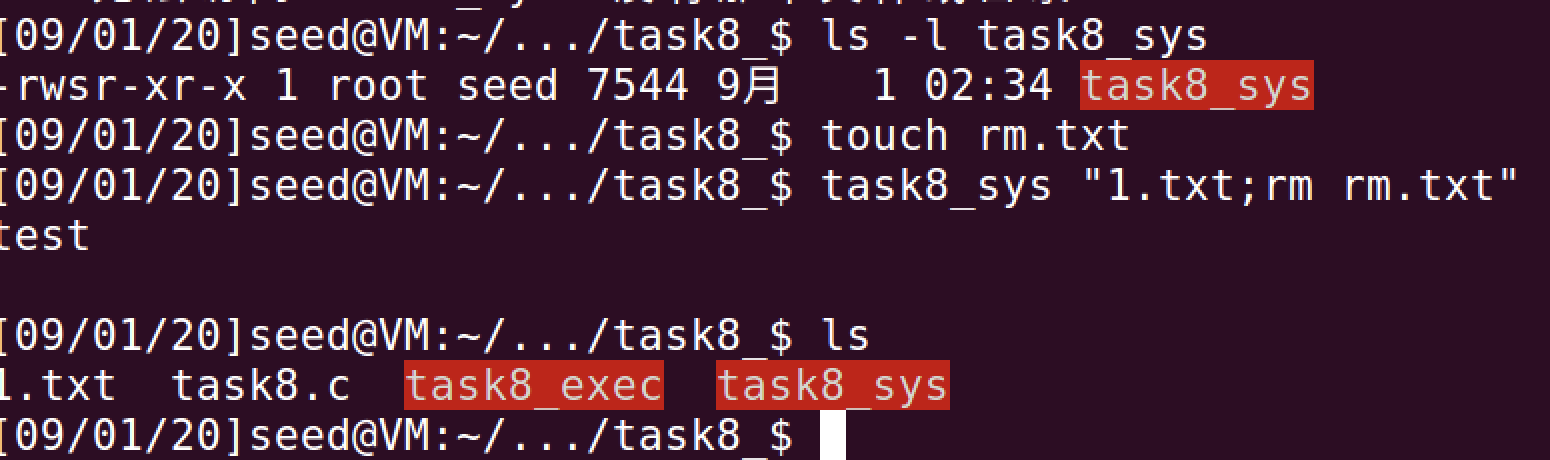
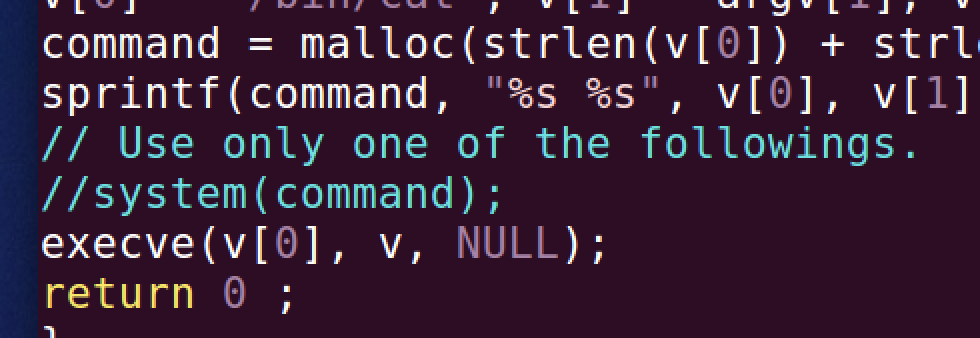
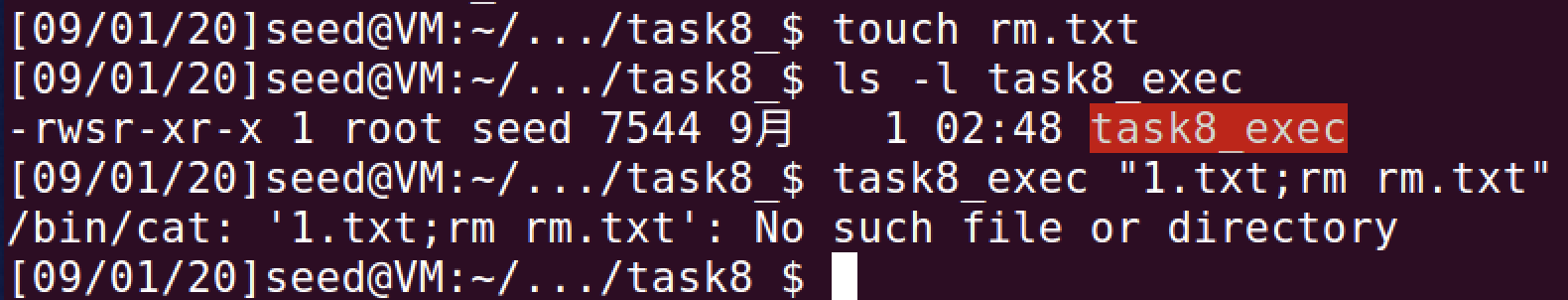
实验结果表明，当进程的真实用户ID和有效用户ID不一样时，进程将忽略LD\_PRELOAD环境变量。当设置Set-UID为后，有效用户等于文件的所有者，因而Step8和Step9可以成功运行自定义sleep()函数。

验证实验

复制一份env程序并设置为特权程序，更改LD\_PRELOAD和LD\_LIBRARY\_PATH环境变量，新增自定义的环境变量LD\_MYOWN和LD\_LIBRARY。

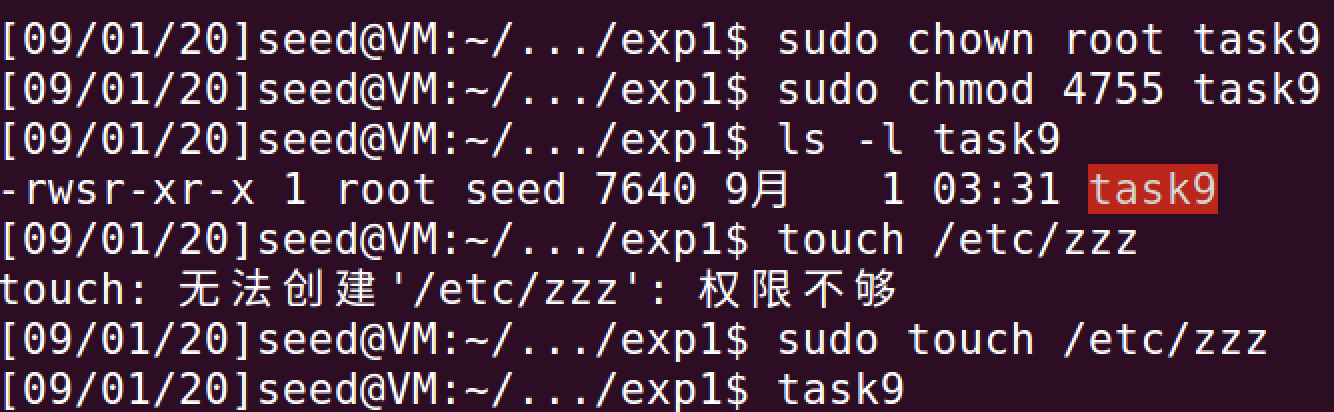
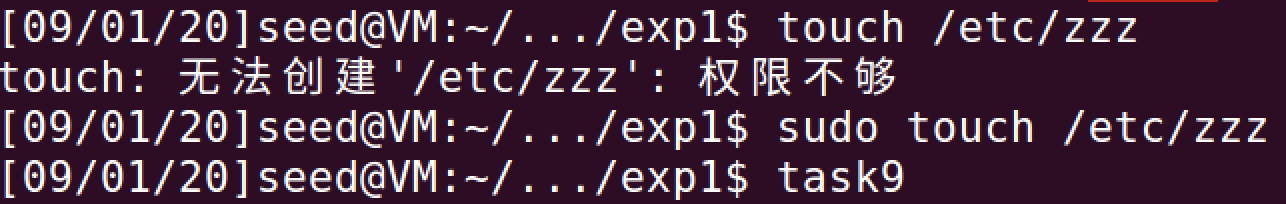
可见，非特权程序env不会屏蔽LD\_PRELOAD和LD\_LIBRARY\_PATH环境变量，而特权程序则会屏蔽二者，只显示了自定义的环境变量。

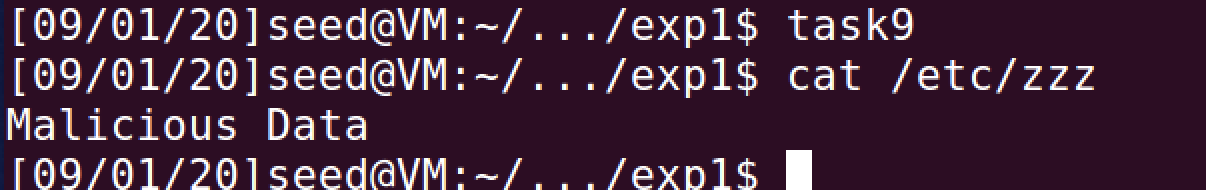
上述实验验证了Set\_UID进程会屏蔽LD\_PRELOAD和LD\_LIBRARY\_PATH环境变量，以防其对特权程序的动态链接产生任何影响。

1. Invoking External Programs Using system() versus
2. 编译并设置task8\_sys为特权程序
3. 利用system()进行多命令执行
4. 换用execve()函数

Conclusion

由于system()函数调用shell，传入的参数可以用分号分割达到执行多条指令的效果；execve()函数会将参数作为一个整体运行，即将其作为一个文件名，找不到文件从而报错，因而不会产生调用system()函数时的效果。

1. Capability Leaking
2. 编译task9并设置为特权程序
3. 创建/etc/zzz文件并运行task9

Conclusion

fork出的子进程仍然具有/etc/zzz读写权限的文件描述符fd，因而可以实现在/etc/zzz文件中写入数据，是一种典型的权限泄露，应当在fork之前关闭文件描述符fd，防止权限的恶意利用。