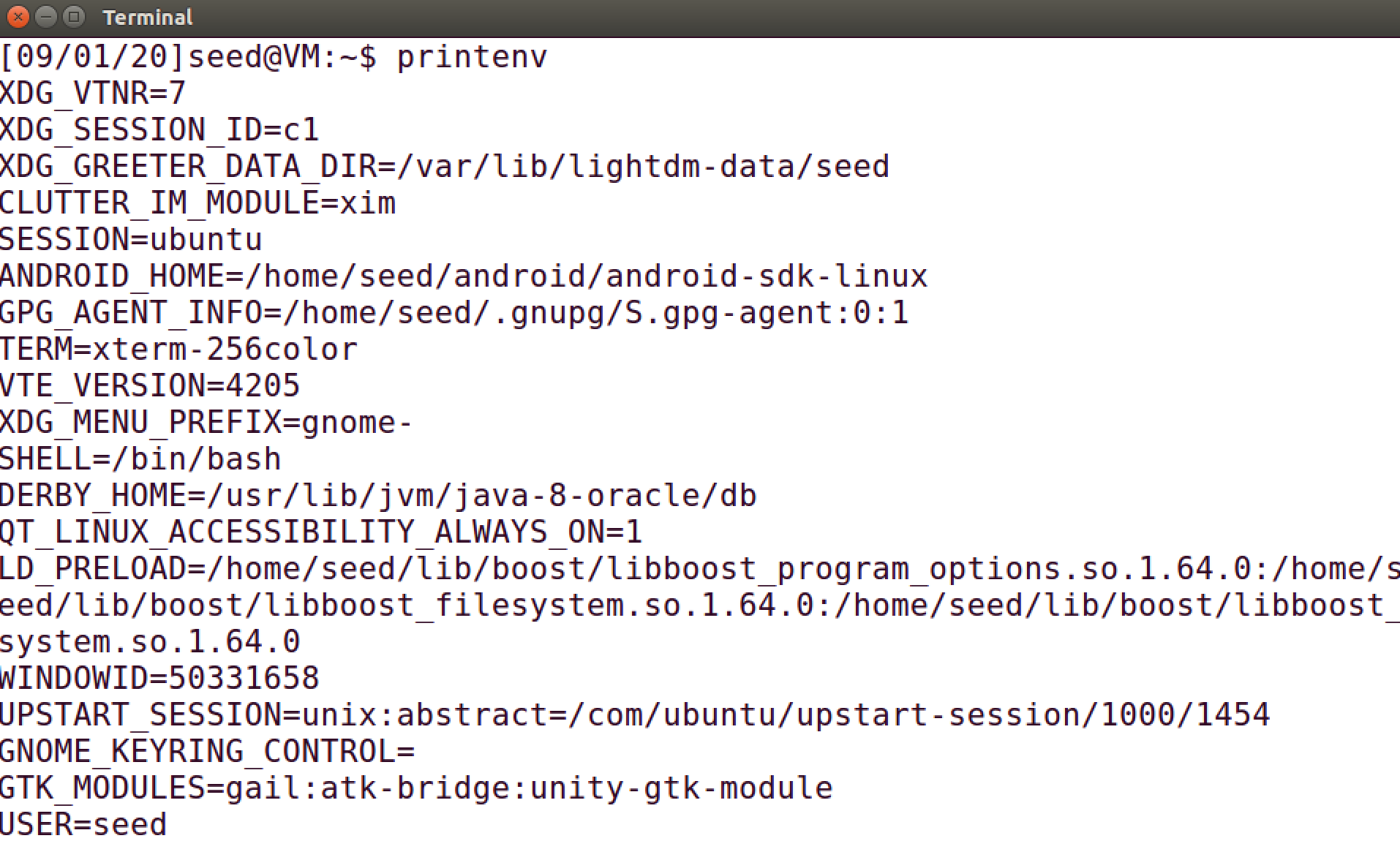
**Lab 1**

**Environment Variable and Set-UID Program**

57117132 费越

# Task 1: Manipulating Environment Variables

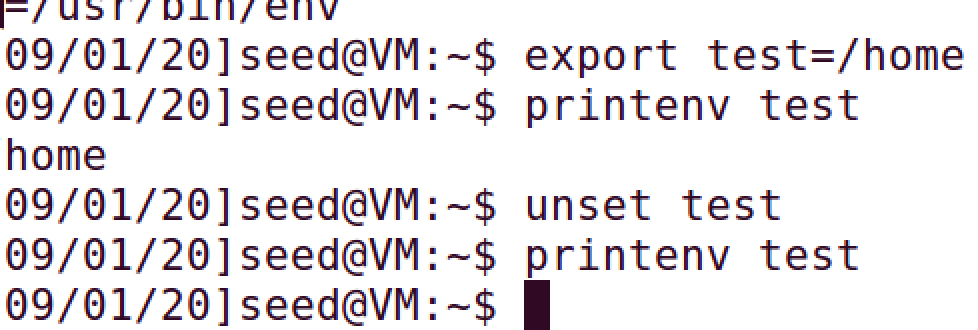
使用printenv命令和env命令打印环境变量：



使用env命令的结果基本与上面相同，仅在末尾部分由于输入的命令不同，显示的命令也有所区别（分别为/usr/bin/printenv(env)）。

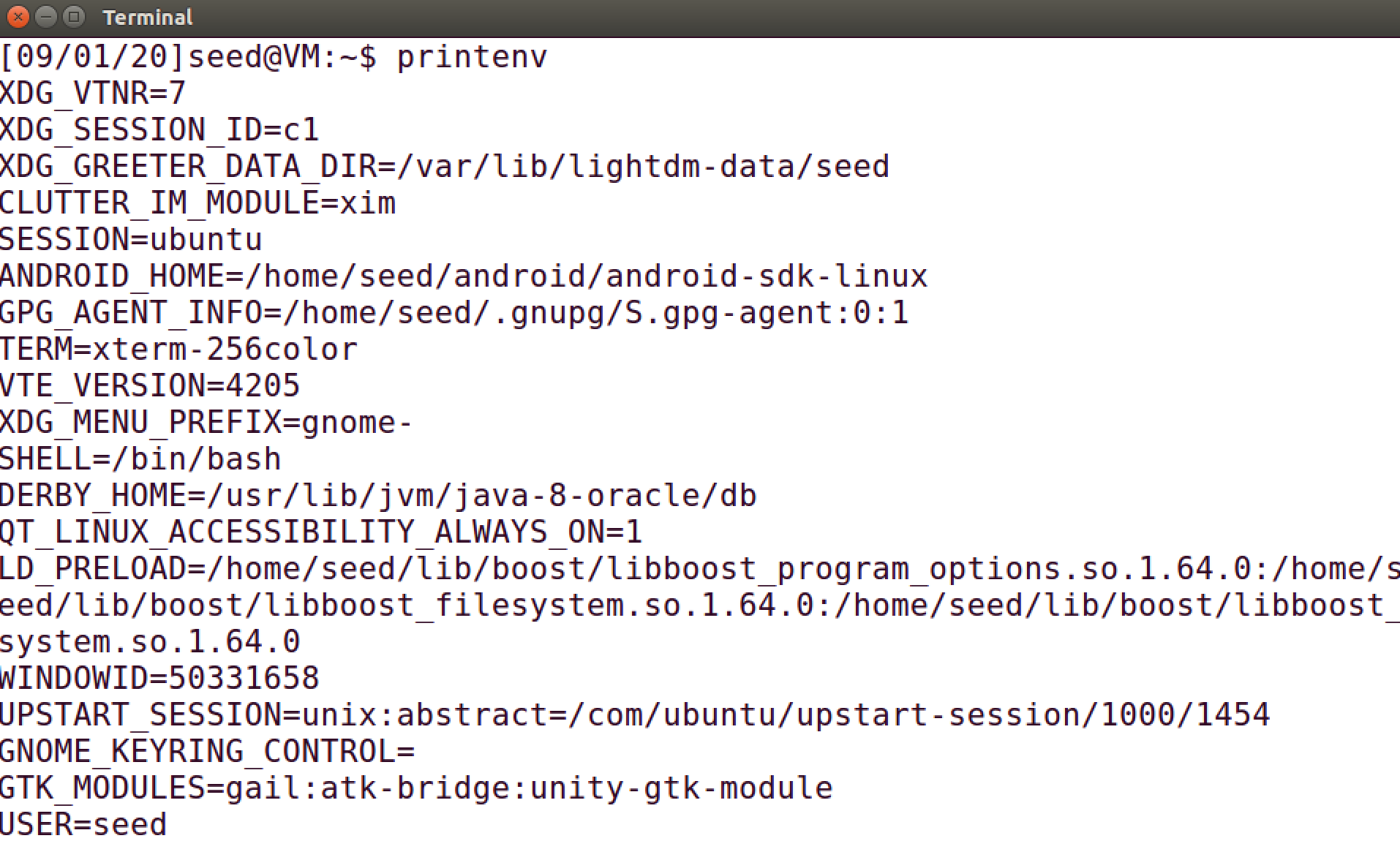
使用export和unset设置或删除环境变量

使用export设置环境变量，使用printenv显示，使用unset删除环境变量。

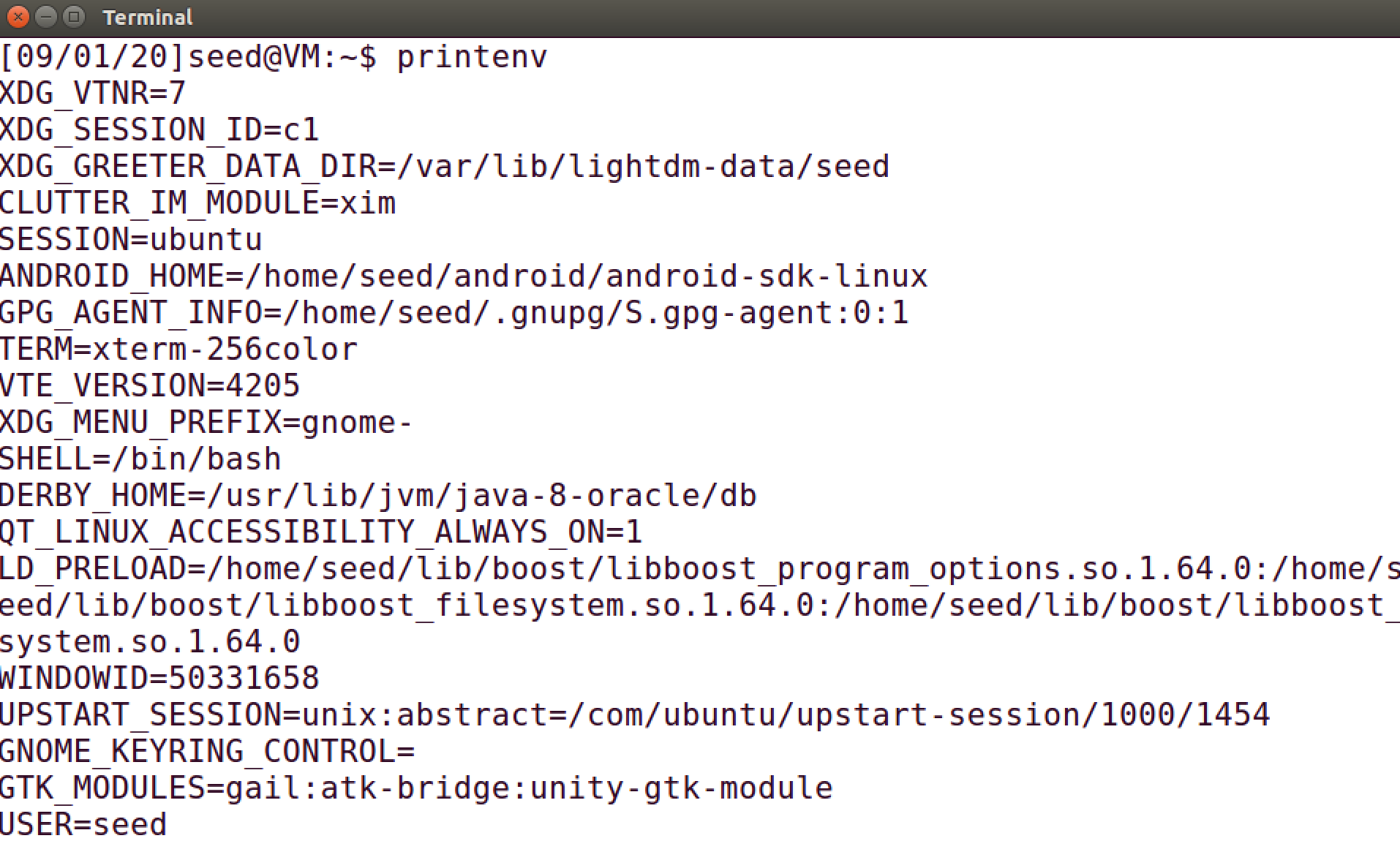


# Task 2: Passing Environment Variables from Parent Process to Child Process

把手册所给程序编译并执行，将其内容保存到文本文件中去，其内容为环境变量的值。



按实验手册要求将程序中child process下的printenv()注释，将parent process下的printenv()取消注释，重新编译并执行，将结果保存为另一个文本文件。内容同样是环境变量。

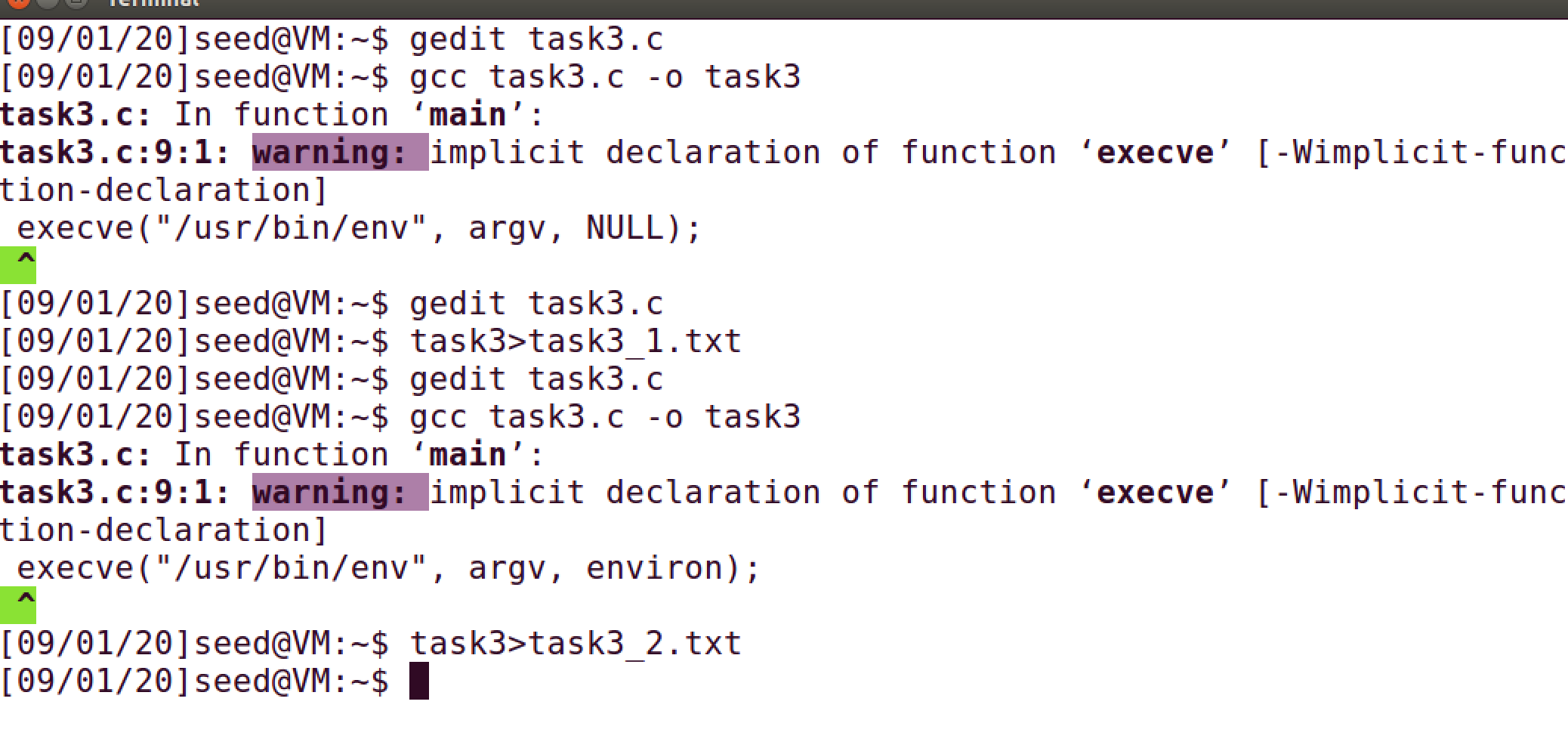


使用diff命令比较，无任何输出，表明两者的内容完全相同。

实验结果表明在Unix环境下使用fork函数，子进程环境变量会继承父进程环境变量。

# Task 3: Environment Variables and execve()

把手册所给程序编译并执行，分别输出到两个文本文件中，结果发现第一个执行结果为空，第二个执行结果为当前的环境变量。



结论：由于execve函数的第三个参数为传递给进程的环境变量，所以第一步赋予新进程的环境变量NULL，打印出环境变量结果就为空白，在修改参数后，结果就变为当前环境变量，其与task1相似。

# Task 4: Environment Variables and system()

编译并运行所给程序，执行结果：

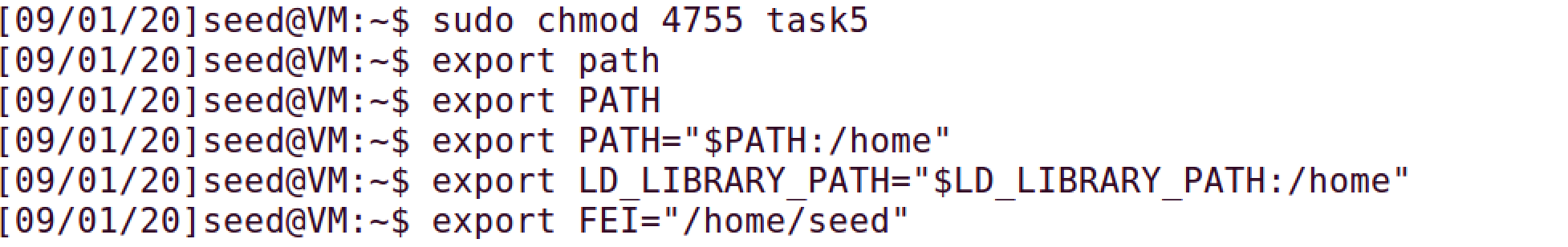


通过printenv命令比较发现二者相同。

结论：通过system命令执行的新程序的环境变量与调用程序相同。

# Task 5: Environment Variable and Set-UID Programs

按实验手册要求操作

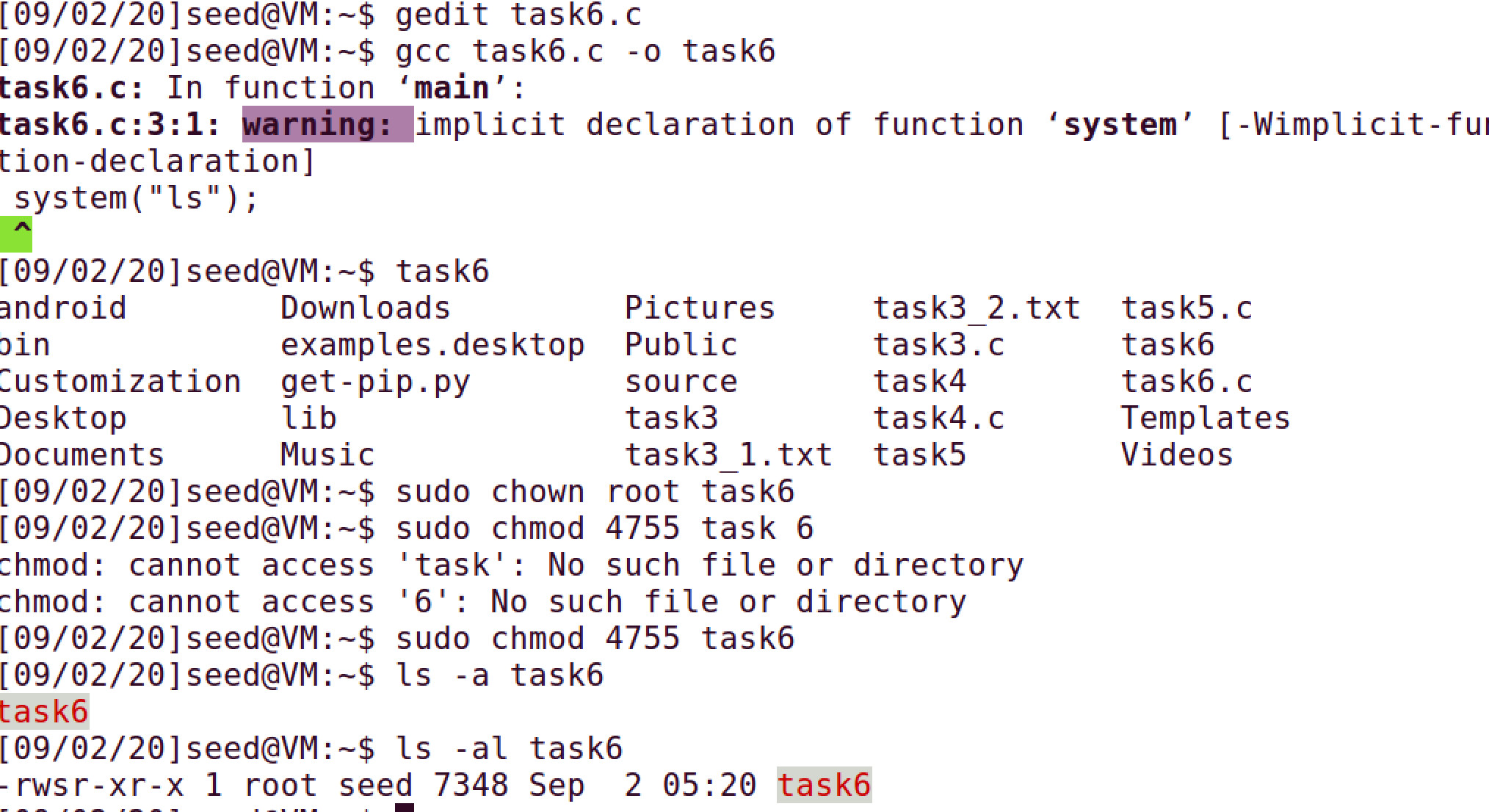


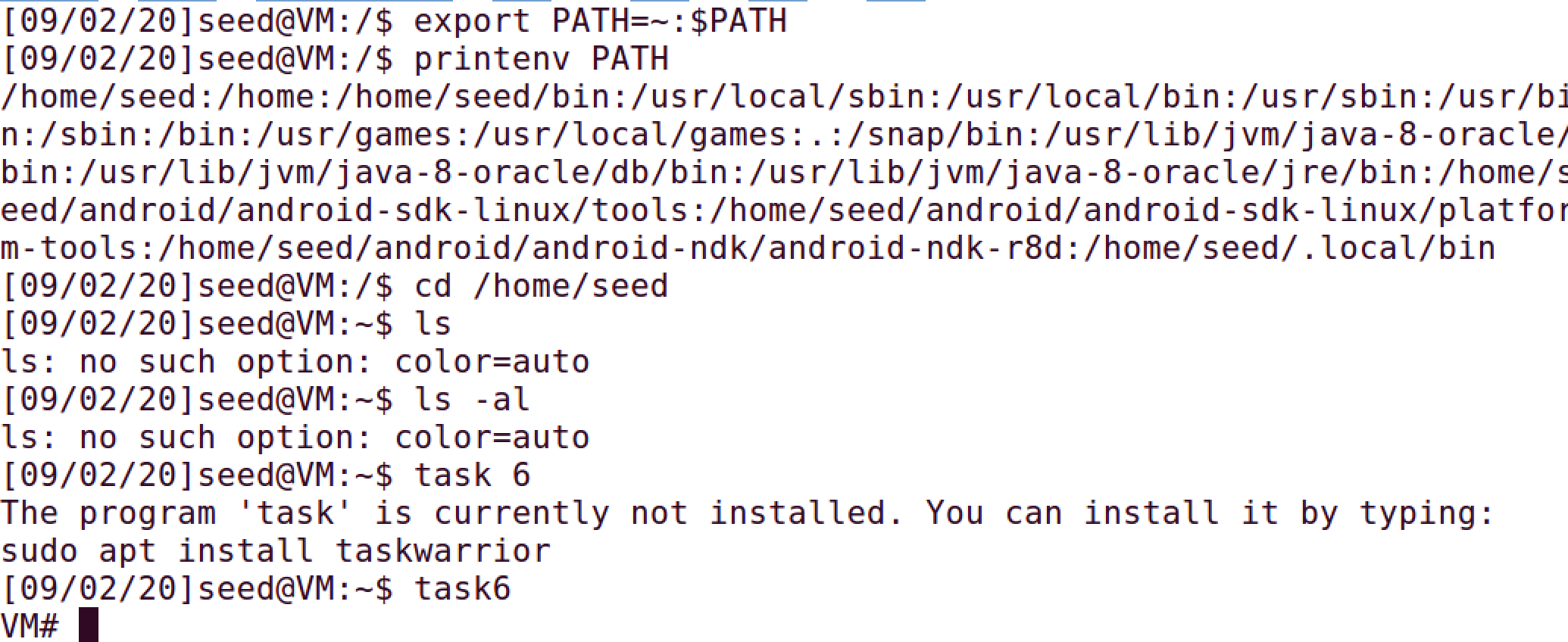


由结果可知，PATH变量中增加了新输入的环境变量，LD\_LIBRARY\_变量没有包含在结果中，自定义的FEI变量出现在结果中。

# Task 6: The PATH Environment Variable and Set-UID Programs

将实验手册所给程序编译，将可执行文件设置为set-UID文件

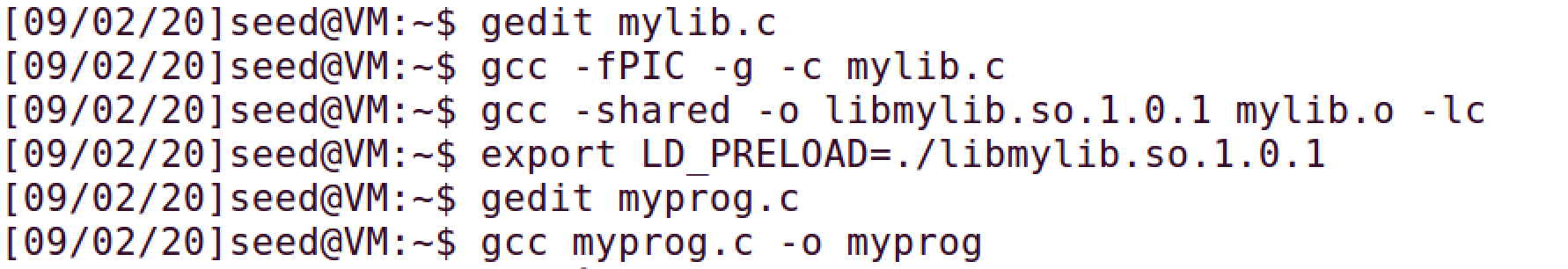




根据实验手册，由于dash的反制措施，所以先把shell改为zsh，然后更改环境变量，将shell复制到当前目录并设其名称为ls，然后重新运行task6，从图中最后的#得知，已经运行了具有root权限的shell，所以set-uid程序所运行的子程序具有root权限。

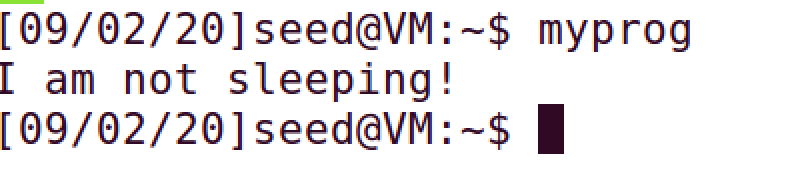
# Task 7: The LD PRELOAD Environment Variable and Set-UID Programs

按照手册所给命令与代码编译mylib.c，然后将LD\_PRELOAD设置为编译的动态链接库。



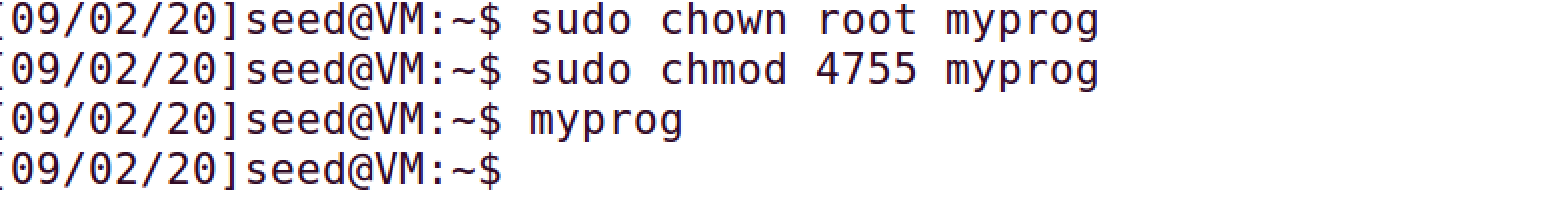
在链接库的相同目录下编译myprog程序，并在不同的情形下运行myprog程序：

（1）普通用户运行程序



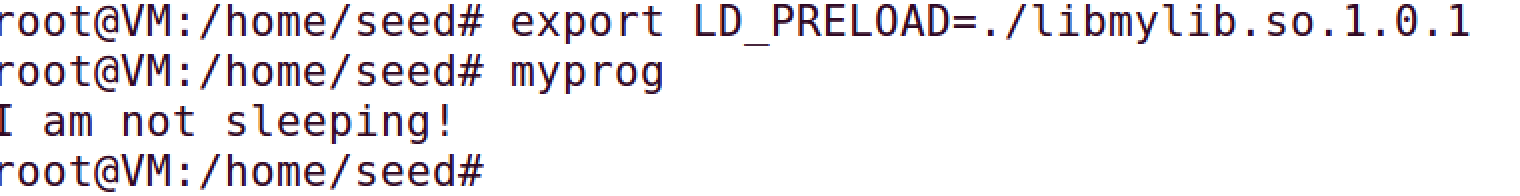
输出I am not sleeping! sleep函数为链接库的

（2）普通用户运行set-UID的root程序



无输出，sleep函数为系统库的

（3）root用户运行程序



输出I am not sleeping! sleep函数为链接库的

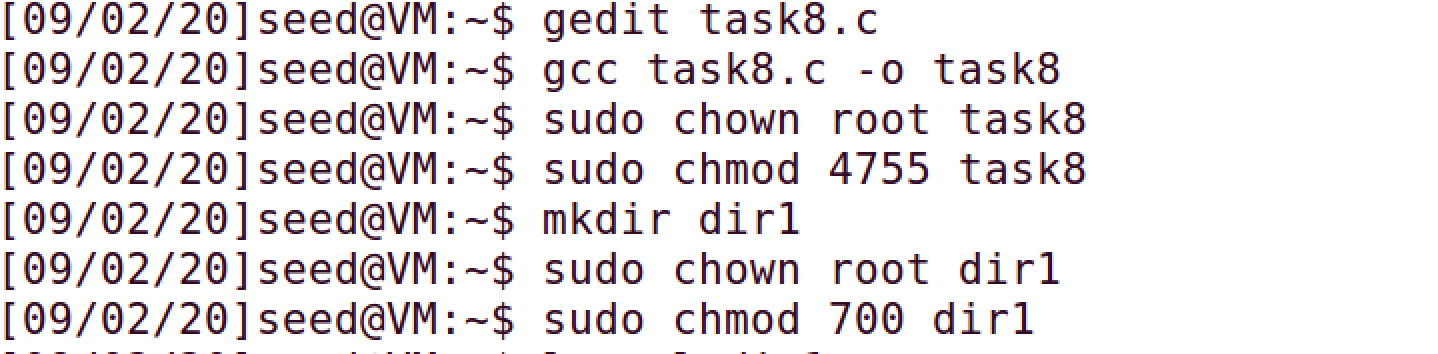
（4）用户运行其他用户的程序

无输出，sleep函数为系统库的

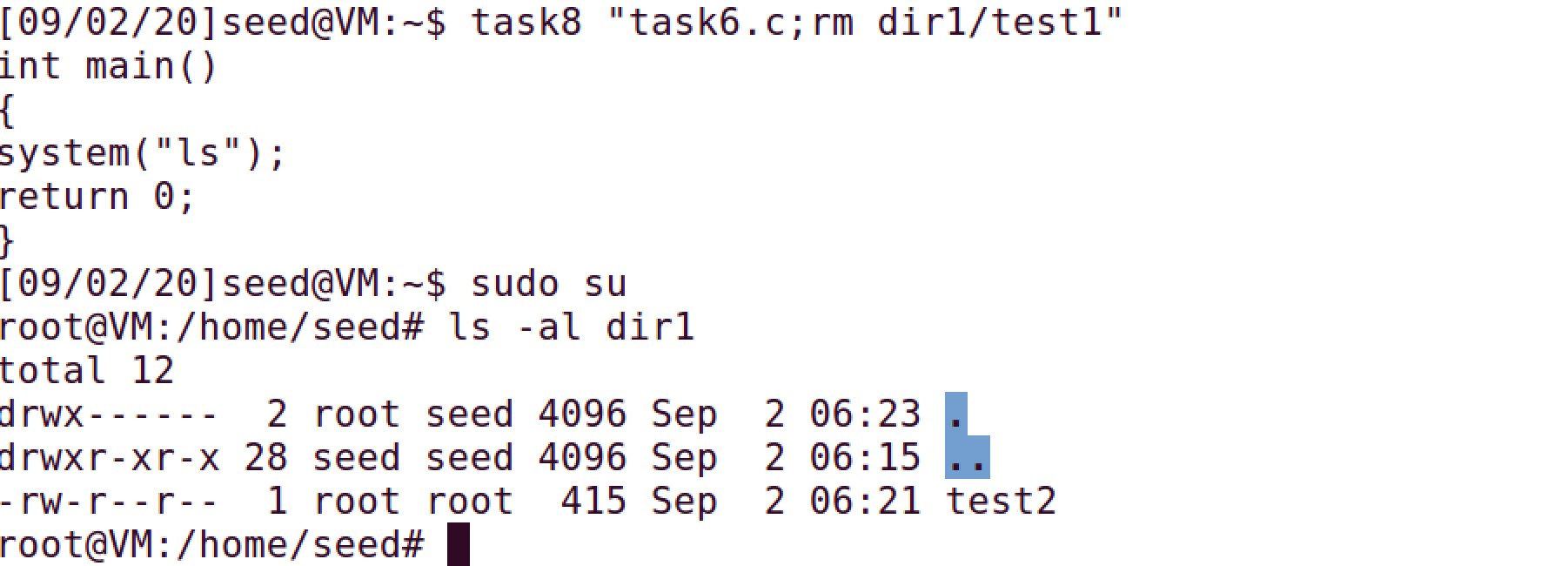
结论：只有用户RUID和EUID相同时LD\_PRELOAD环境变量才会生效。

# Task 8: Invoking External Programs Using system() versus execve()

将所给程序编译，设置为set-uid程序，新建一个只有root有权限的文件夹dir1



在dir1中写入两个文件：test1和test2，在普通用户权限下使用task8删除test1

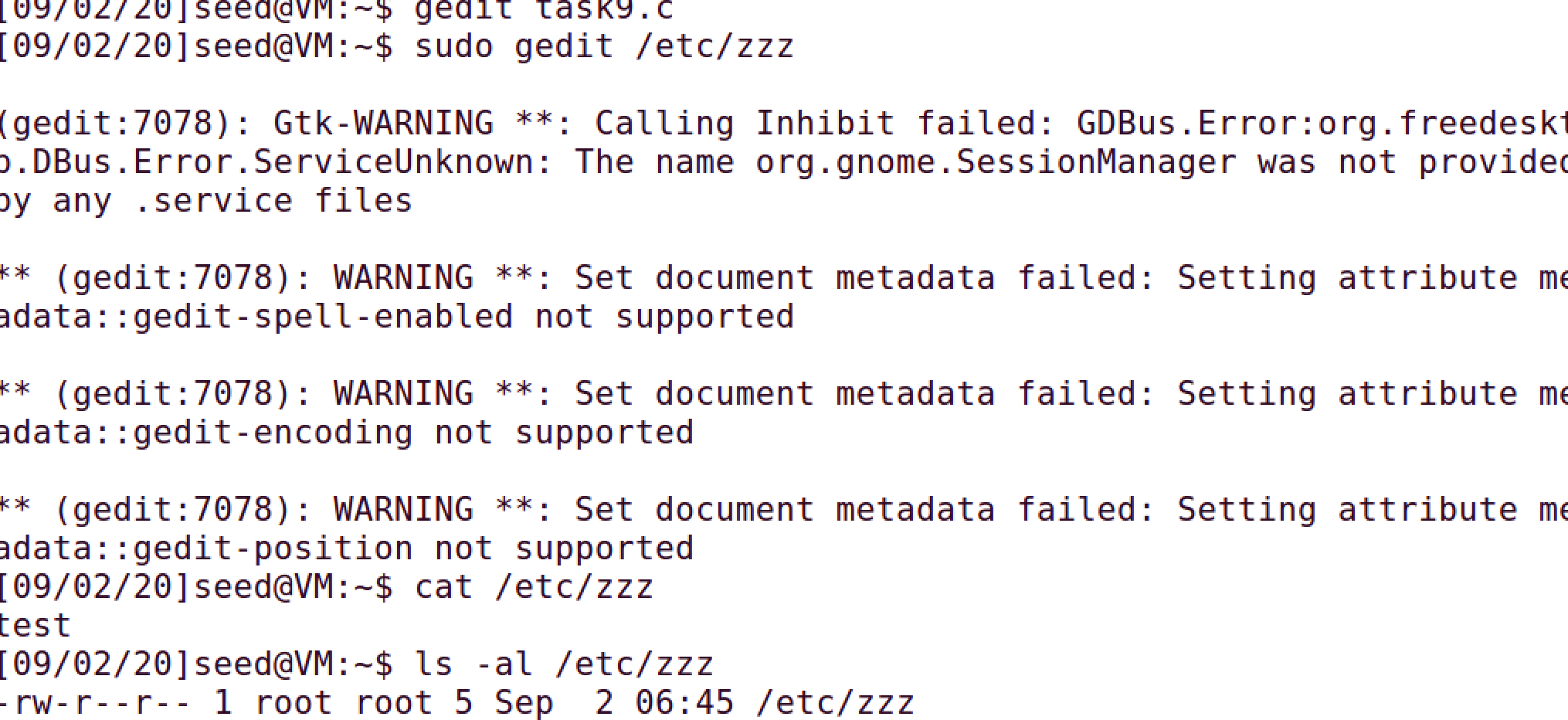


在没有root权限的情况下删除成功。接下来注释掉system(command)语句，使用execve（）来调用该命令。然后尝试按照同样的方法删除test2，发现不能删除，原来的攻击方法已经失效。

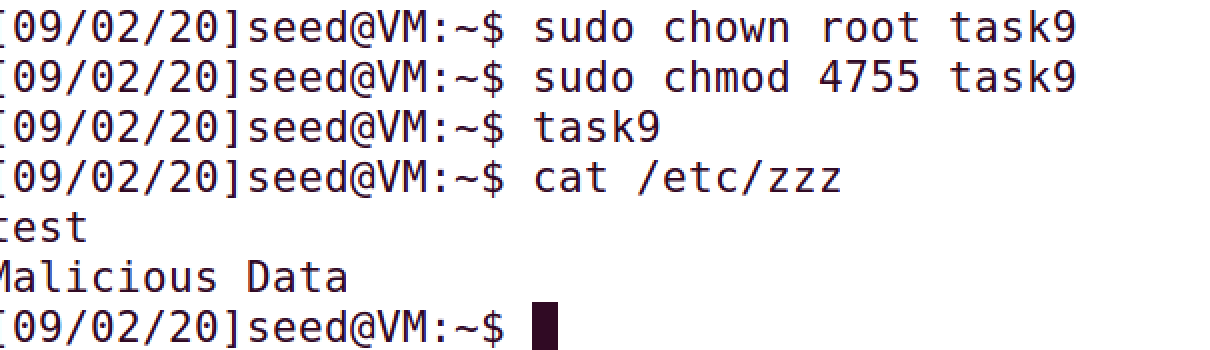
结论：system函数会重新开启一个shell来执行用户的命令，在程序为set-uid程序时会造成巨大安全隐患；而execve函数则是把用户的输入作为参数，其具有更高的安全性。

# Task 9: Capability Leaking

创建/etc/zzz文件



查看权限，只有root用户有写权限。



运行set-uid程序task9，发现zzz文件中的内容已经被改变。

结论：该程序在取消权限前并没有关闭打开的文件，导致seed用户仍可以进行root用户才可以执行的写入操作，造成安全隐患。