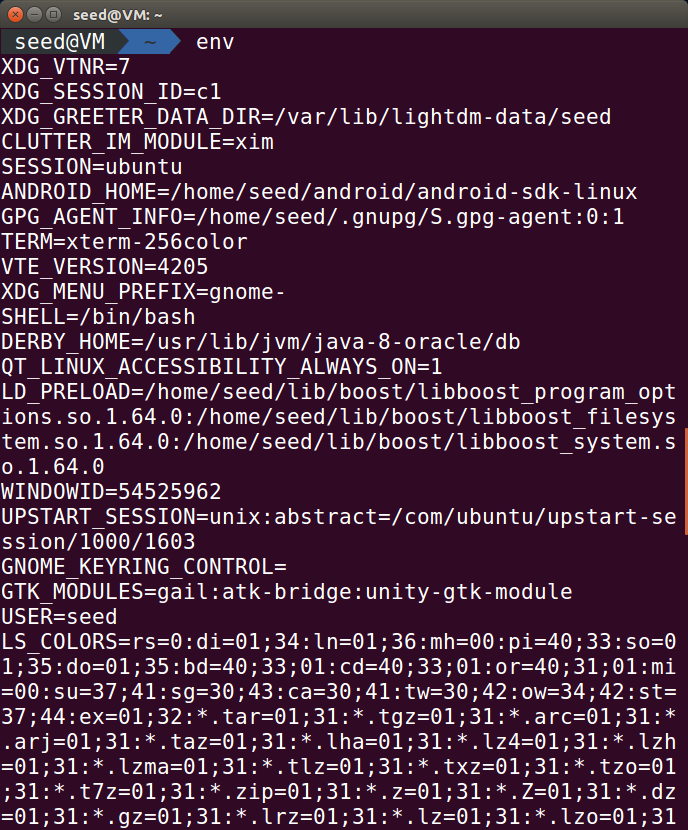
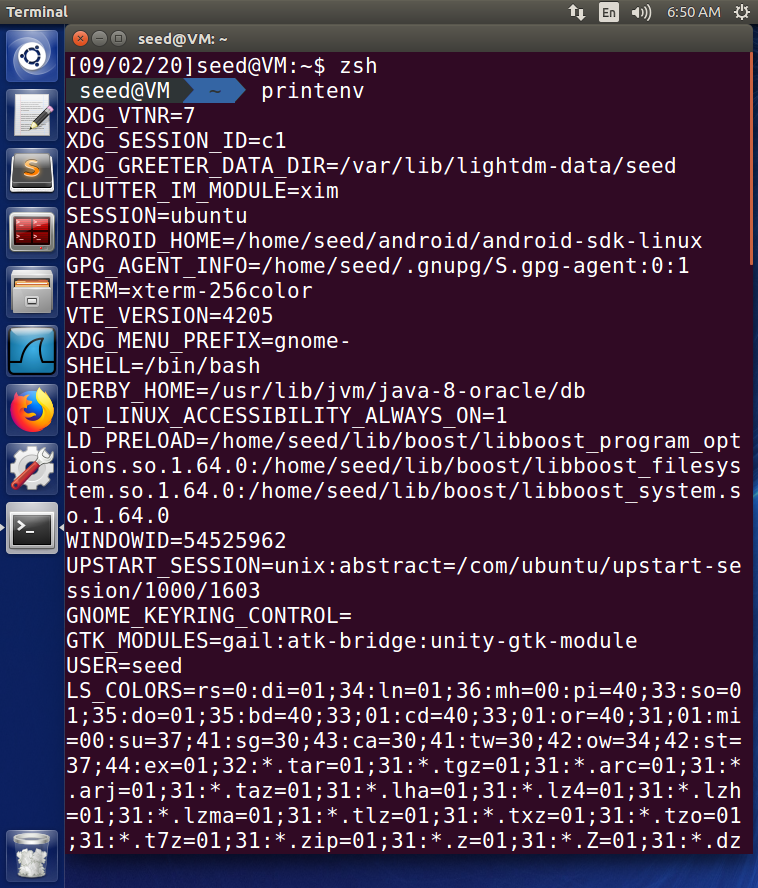
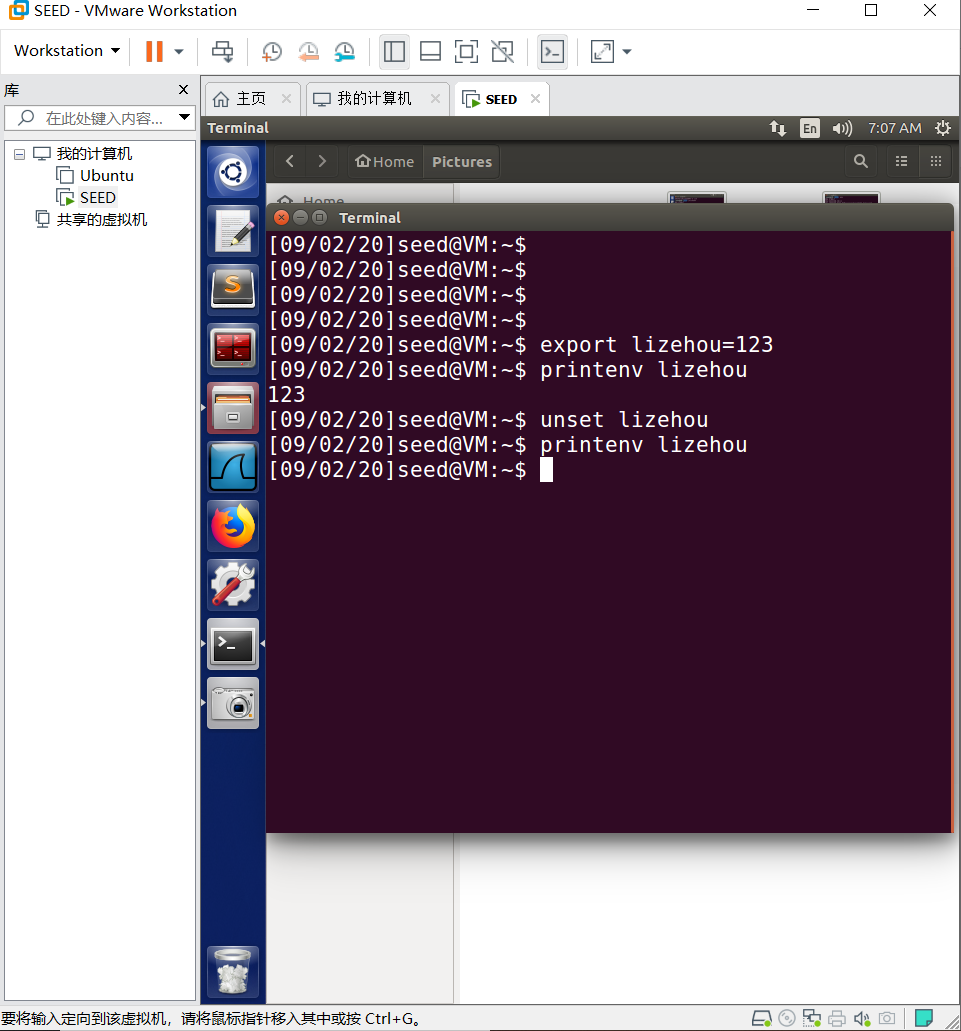
**Task 1**

输出环境变量env



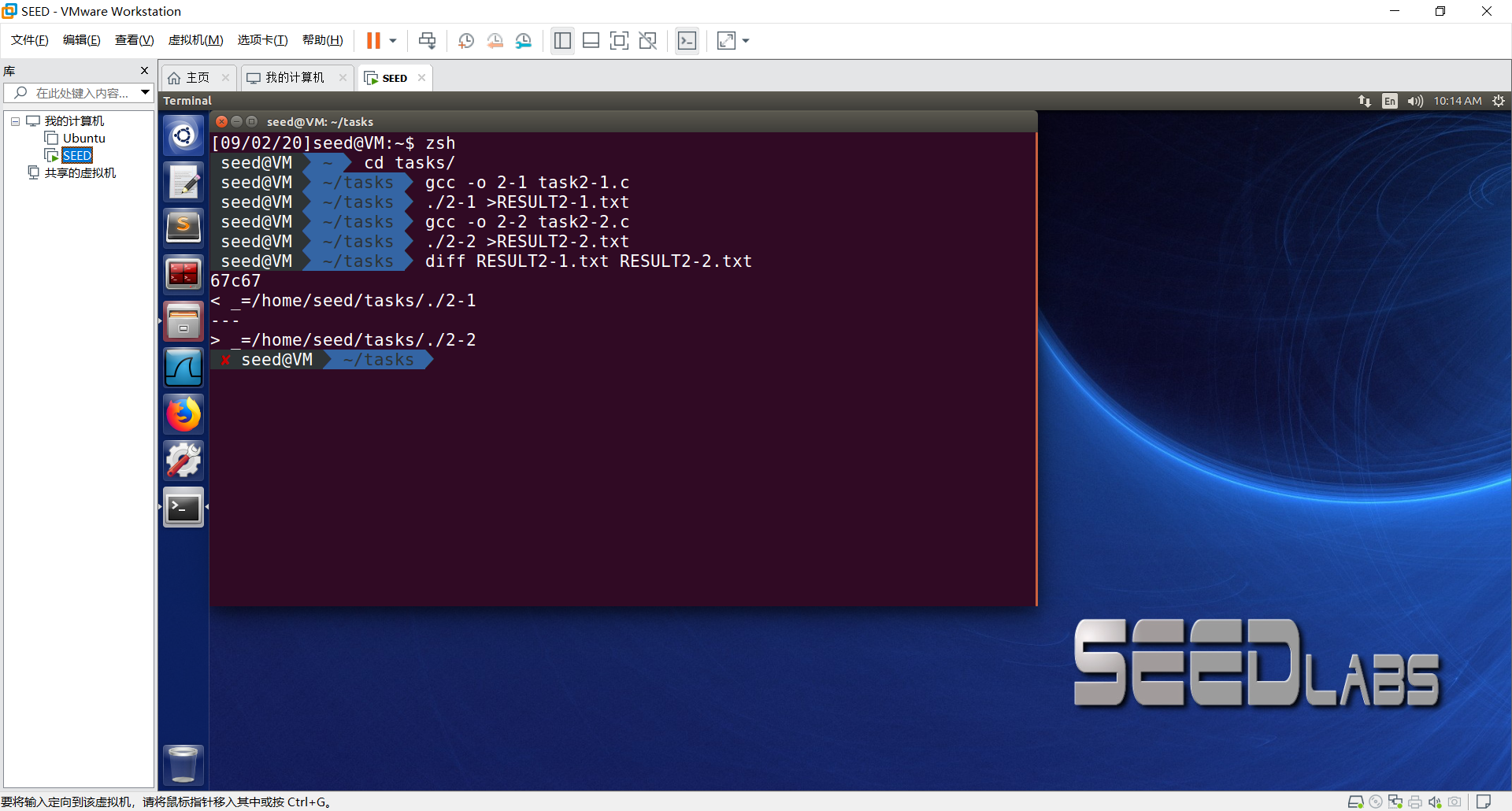


设置变量export并打印

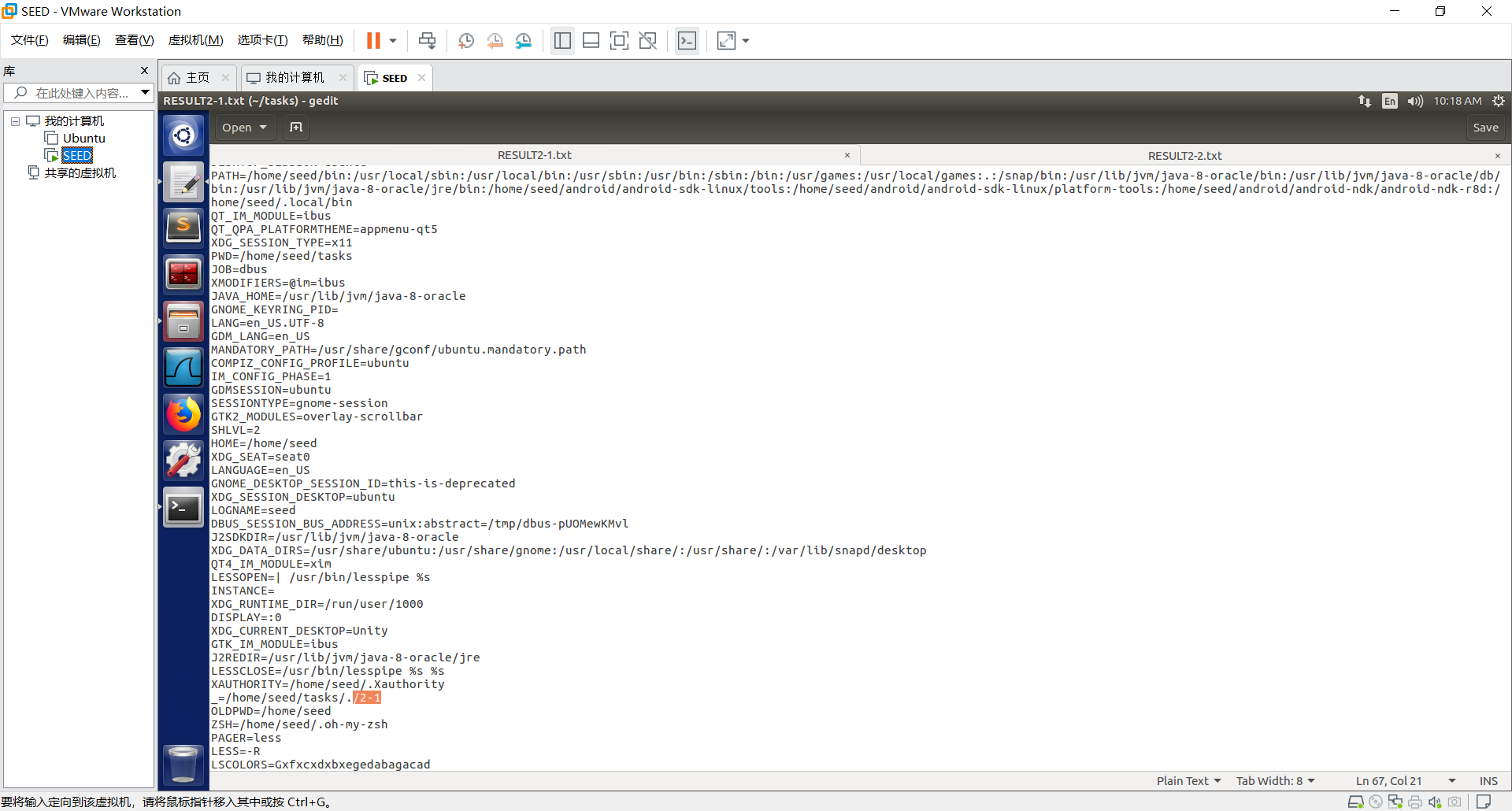


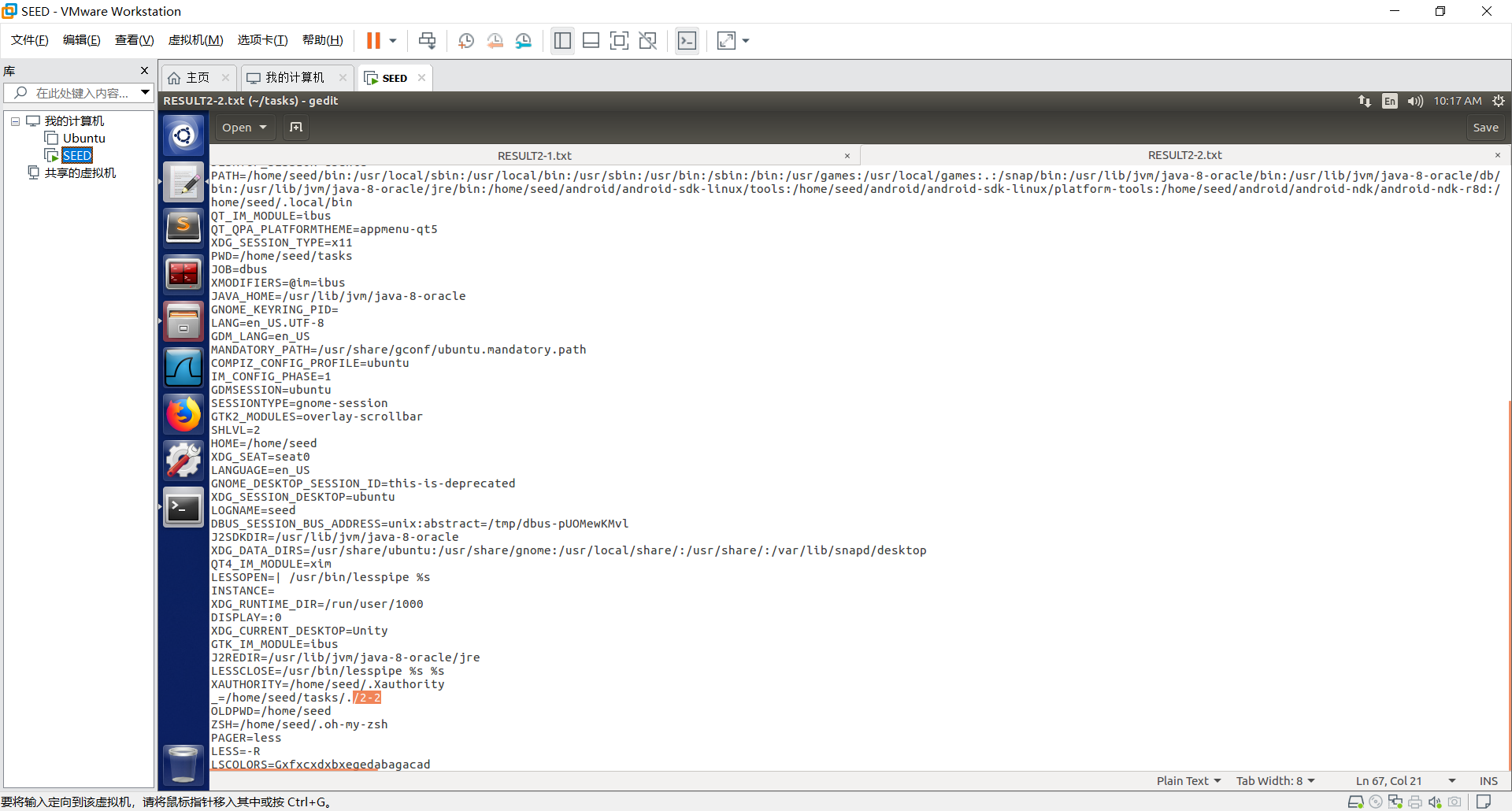
**Task2**

对两个注释位置不同的代码经行编译，并写入不同的文件中。使用diff命令进行比较，报告两个文件的第67行存在不同。



打开两个文件，对比其第67行，发现只是可执行程序的名称不一样，其余之处两个文件都完全一致。

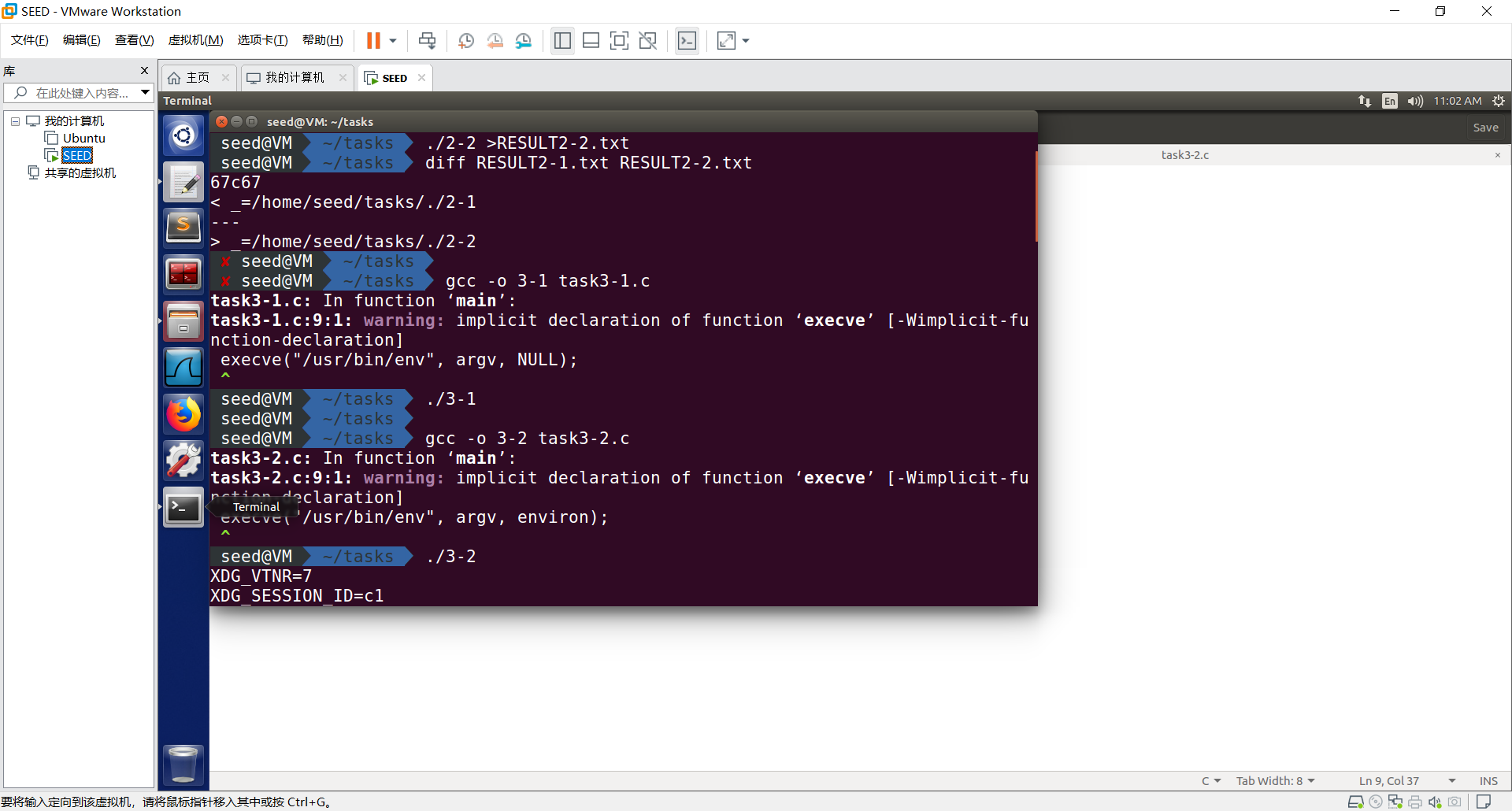




**Task3**

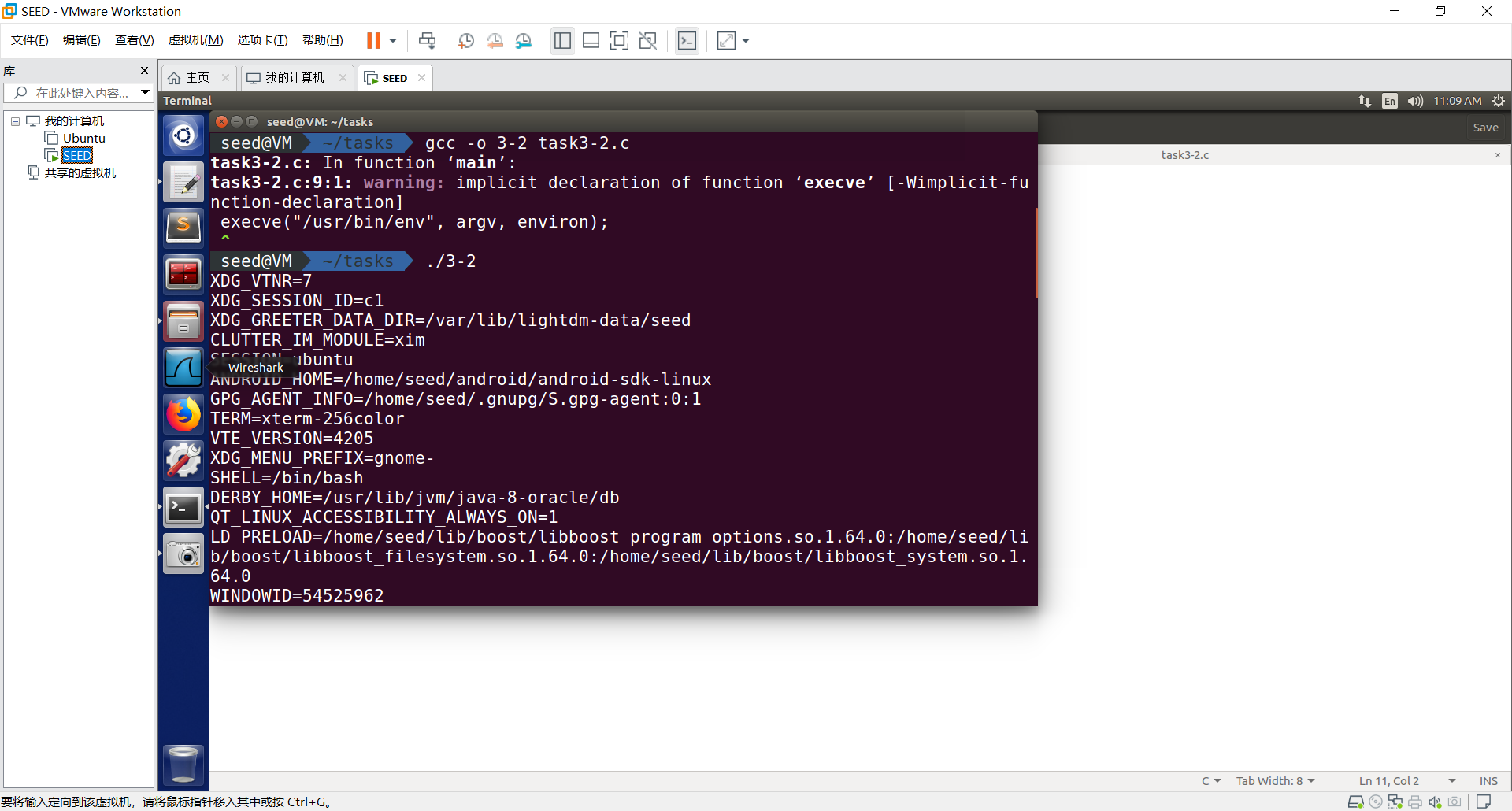
Step-1

未输出环境变量



Step-2

成功输出环境变量

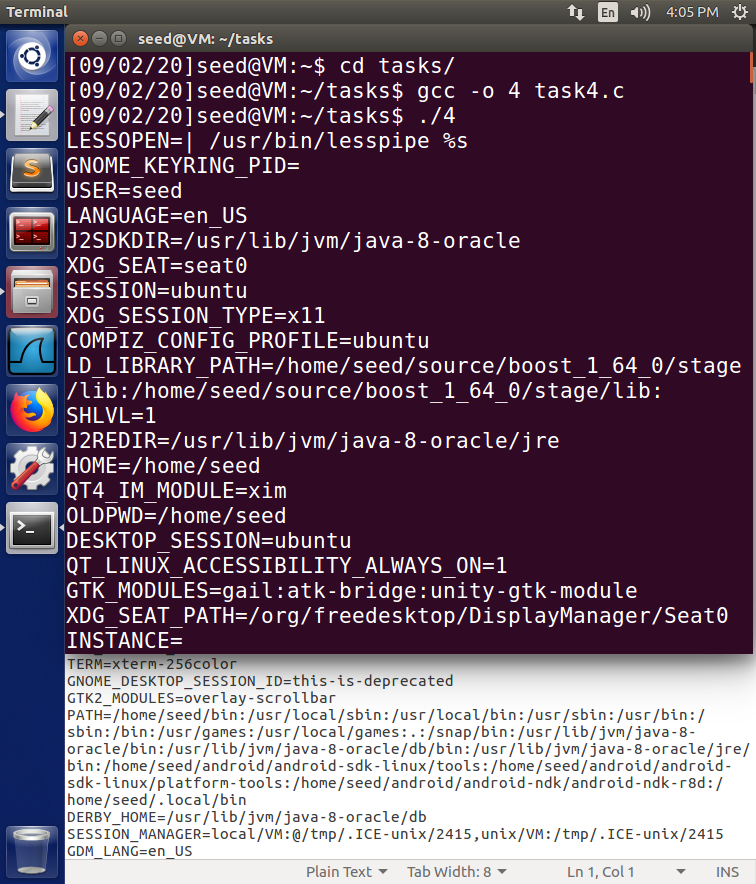


Step-3

前两步区别是由于execve（）函数的参数不同。Step-1中，第三个参数是NULL，指针指向的需要传递的环境变量是一个空指针，故不能输出环境变量；Step-2中，第三个参数是environ，故能输出环境变量。

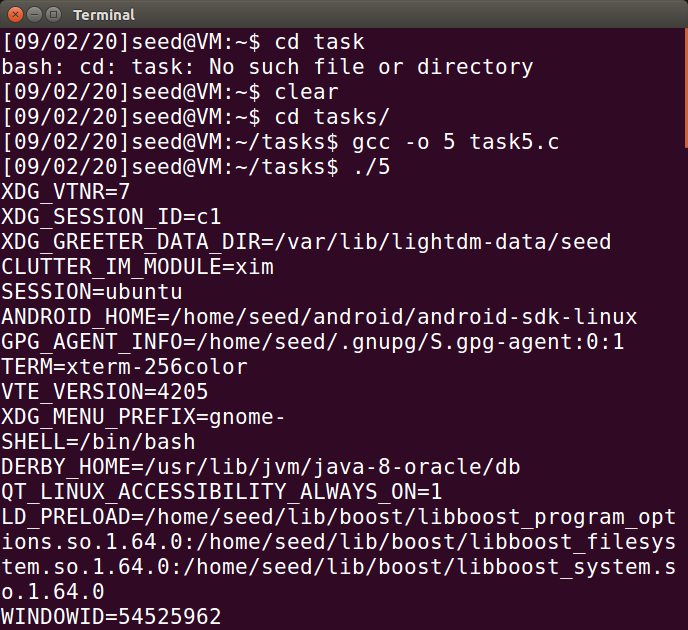
**Task4**

程序运行结果如下图



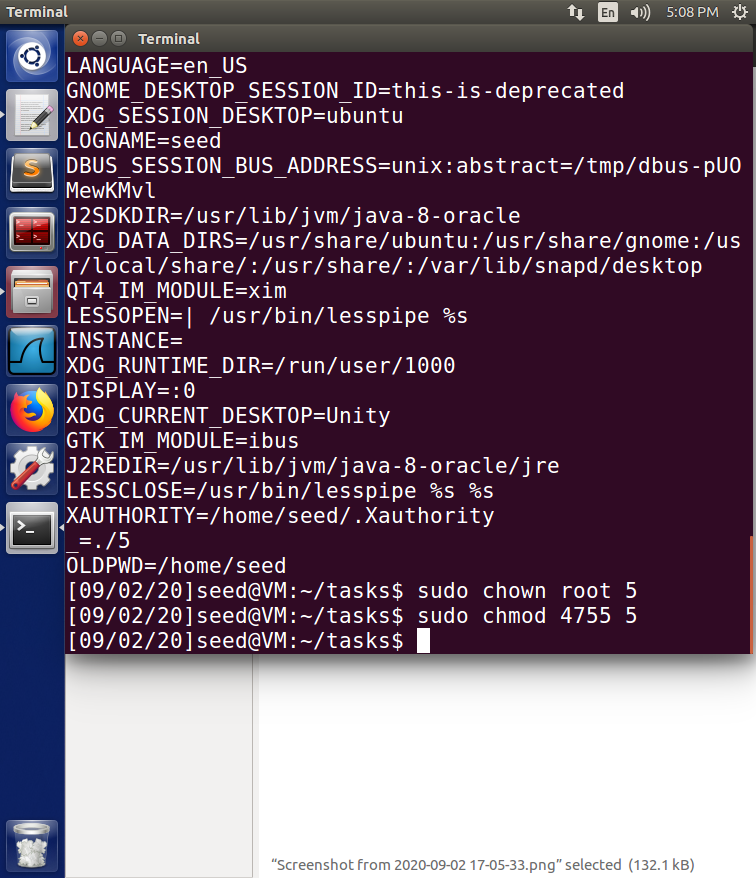
**Task5**

Step-1：打印当前环境变量

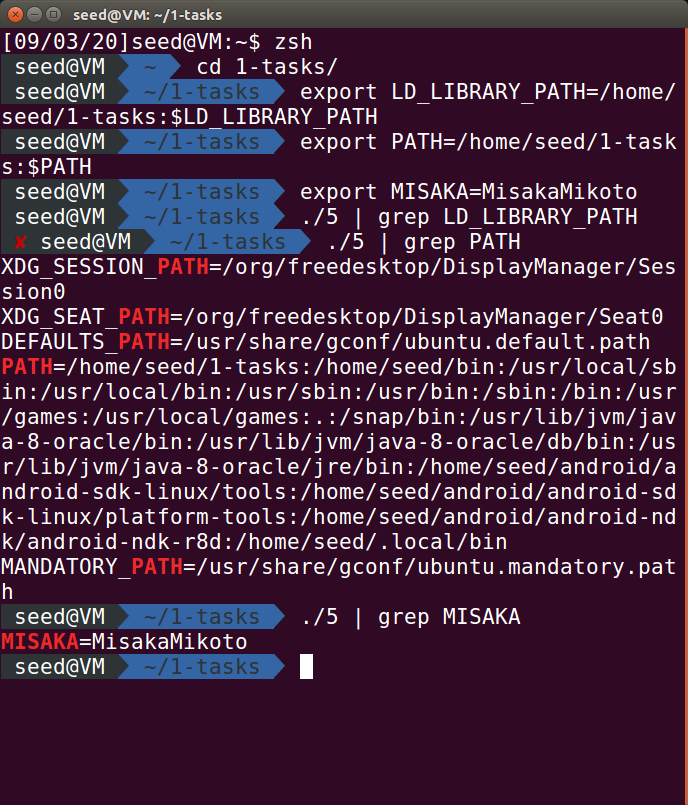


Step-2

将其所有权改为root，并使其成为Set-UID程序。



Step-3

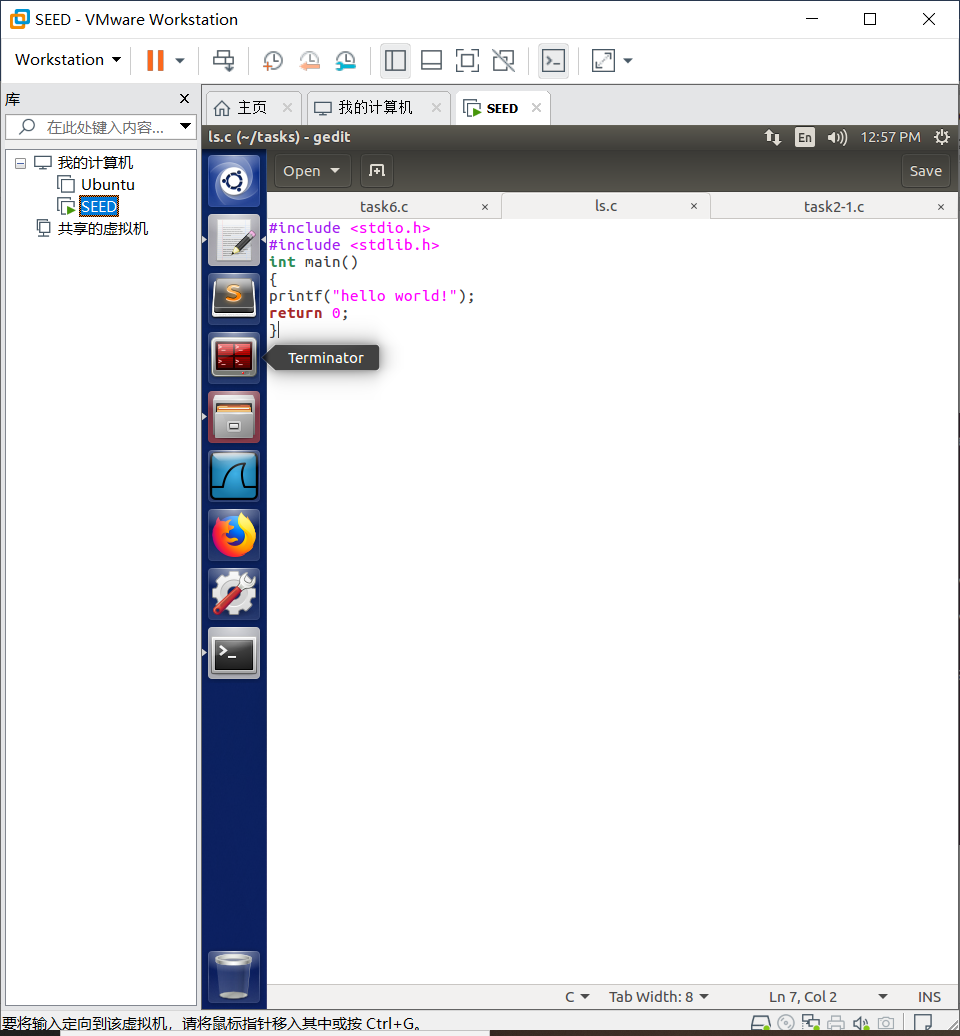


对PATH，MISAKA(自己定义的一个环境变量)，以及LD\_LIBRARY\_PATH进行路径修改，可以发现PATH和FOO两个环境变量正常，而LD\_LIBRARY\_PATH变量不存在，可知Set-UID程序会防止用户劫持动态链接库，造成任意命令执行。

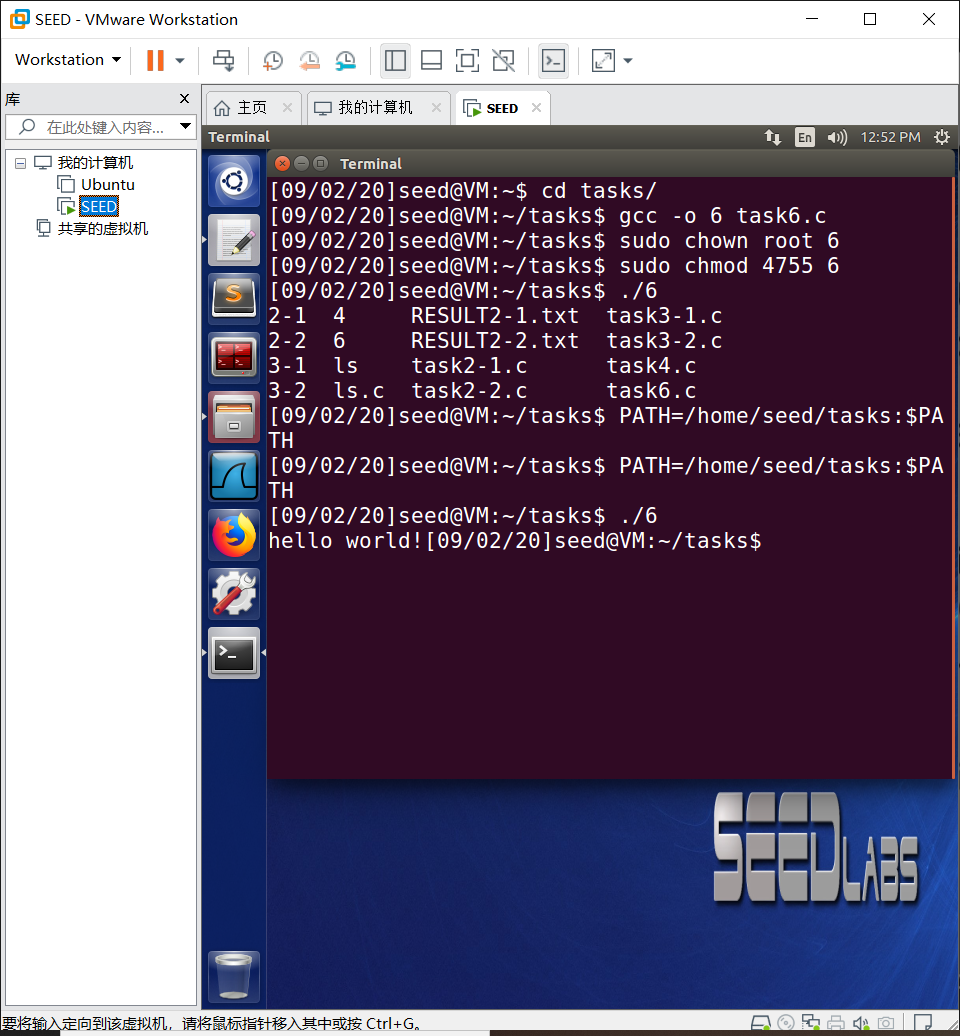
**Task6**

在home/seed/tasks 路径下创建同名文件ls并编译，产生可执行文件ls。

ls如下：



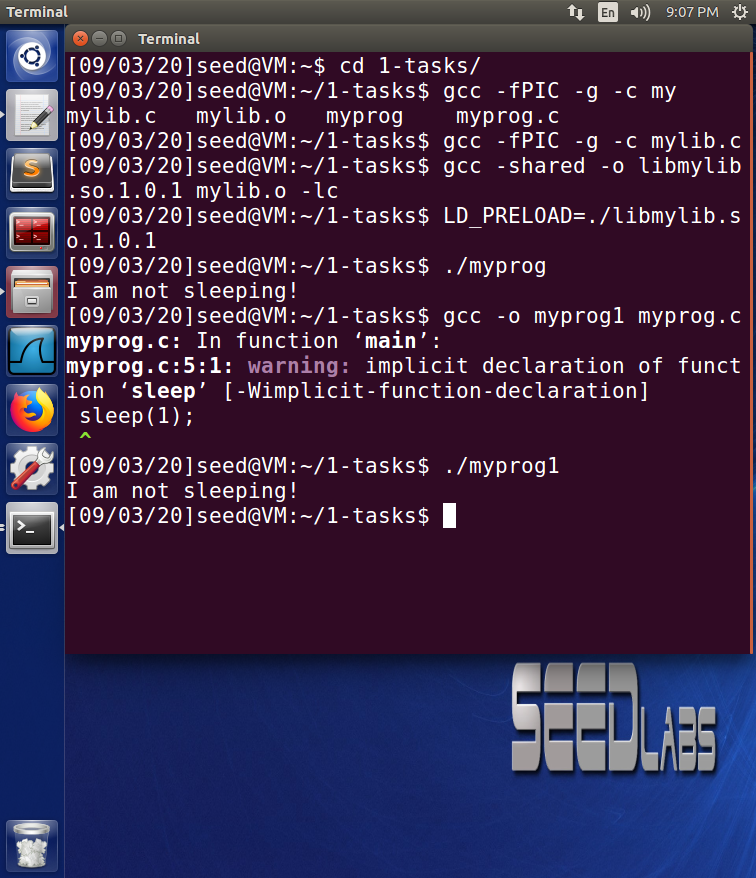
将home/seed/tasks放到PATH的开头，再次执行程序，不再执行bin/ls，而是输出“hello world！”



**Task7**

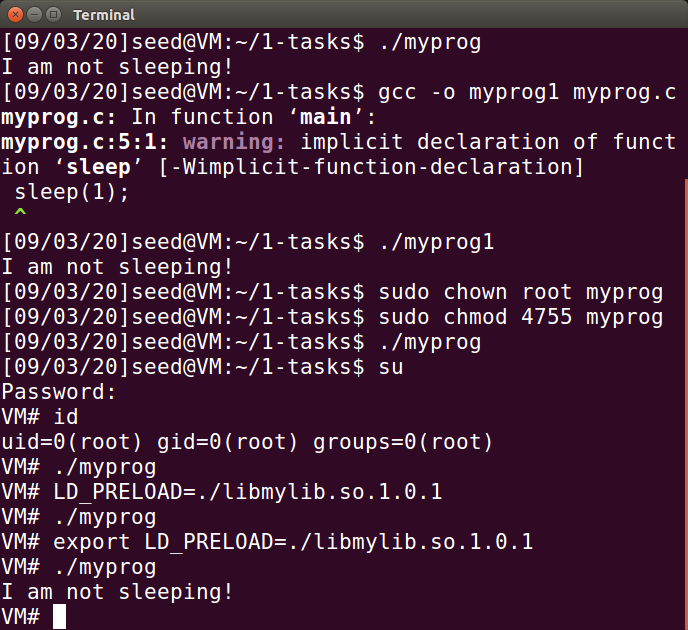
Step-1

按照指示执行操作，以普通用户身份运行myprog，发现劫持成功。

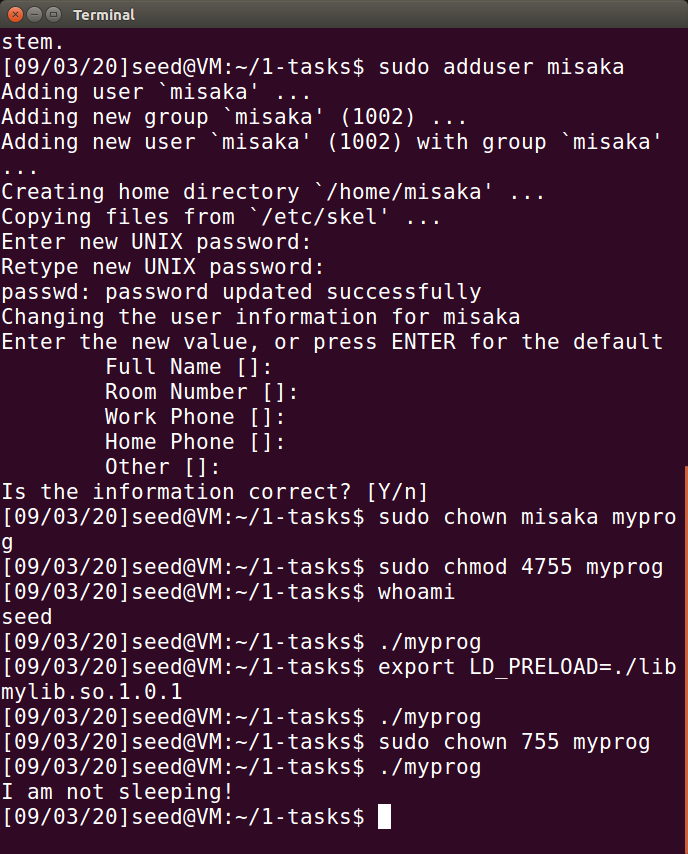


Step-2

将其设置为Set-UID程序，重新设置LD\_PRELOAD变量，调用myprog，sleep再次被劫持

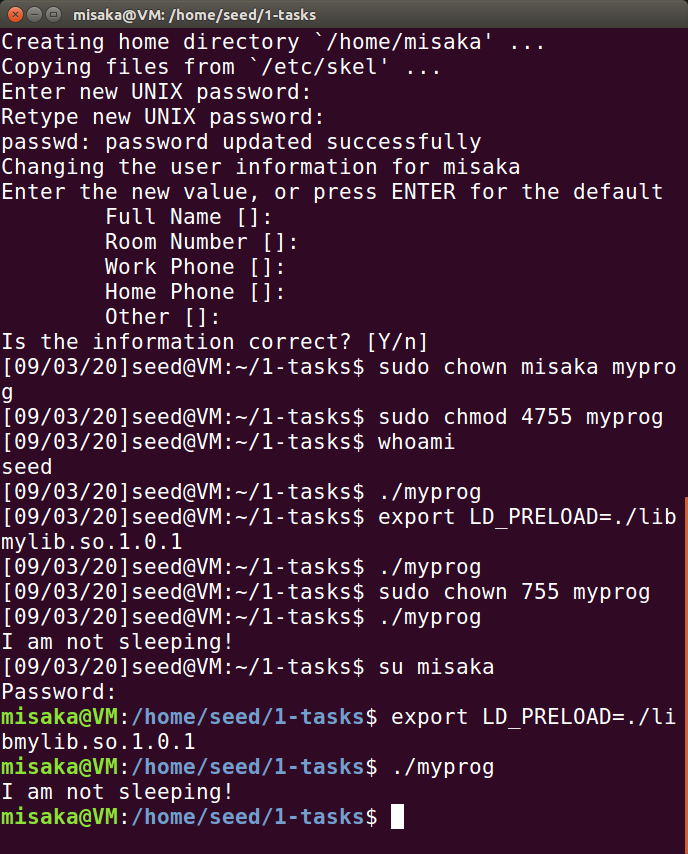


创建一个新用户misaka，将myprog的权限转给他，并设置为setUID程序，切换回非root普通用户，运行myprog，执行正常的sleep函数，再将myprog变为普通可执行文件，使用普通账户执行，sleep函数被劫持。



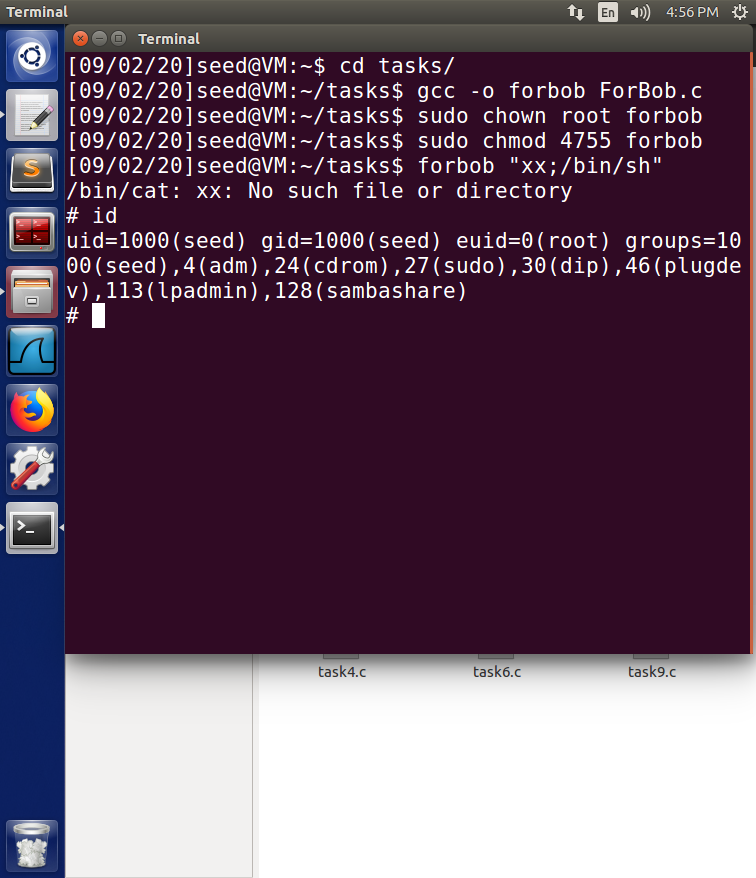
Step-3

最后切换到用户test，重新导出LD\_PRELOAD，运行setUID的myprog，sleep被劫持

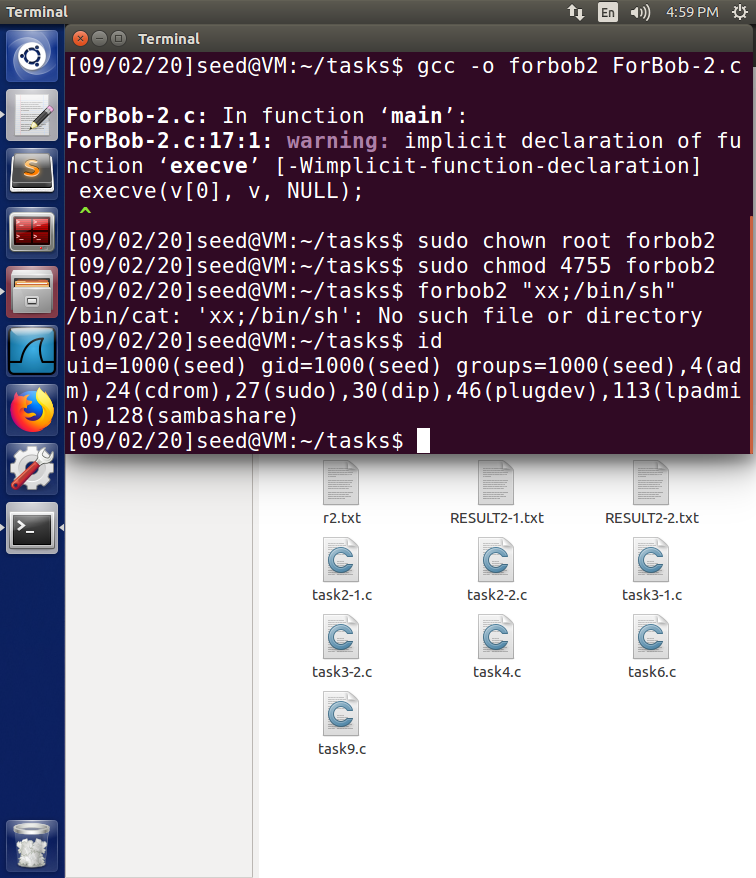


综上所述，setUID程序在进行动态链接时，会对EUID和RUID进行比对，若不一致，则忽略LD\_PRELOAD，防止被劫持

Task8



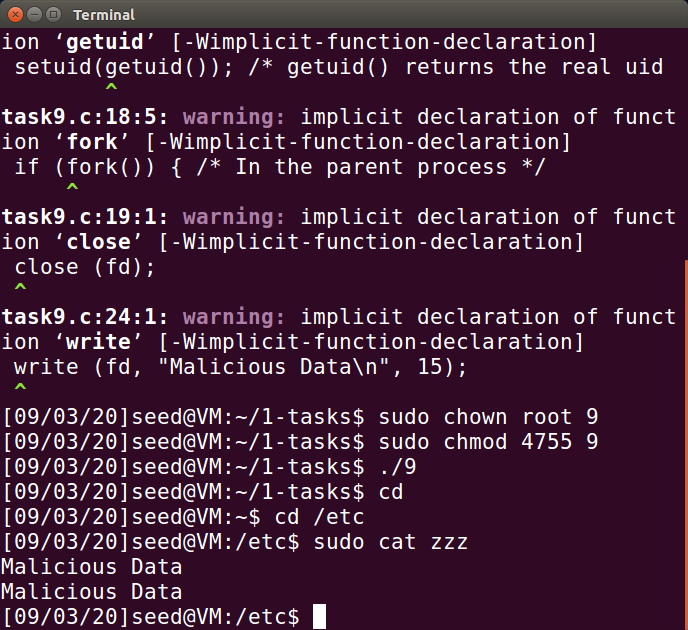
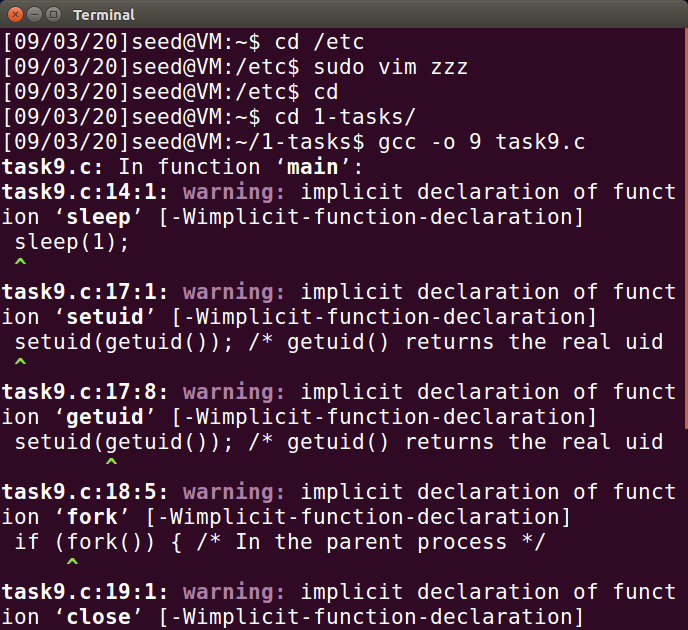
编译所给的程序，使它成为根拥有的Set-UID程序。使用system()调用该命令，此时是有危险的，Bob可能做出有害系统的事情，因为会出现命令解析不到位把用户的输入当作是一个命令的错误，当输入内容存在分号时，用户的输入在分号后的部分就会被当做一个命令。



注释掉system(command)语句，取消execve()语句的注释;让程序使用execve()调用该命令。此时Bob无法做出有害系统的事，因为用户的输入和命令被彻底隔开，不会出现命令解析不到位把用户的输入当作是一个命令的错误。

Task 9

执行运行结果如下图依次所示：



可看出，zzz文件的确遭到了修改。原因在于设置uid前，zzz文件就已经被打开了。只要将语句setuid(getuid())移至调用open函数之前，就能避免这个问题。