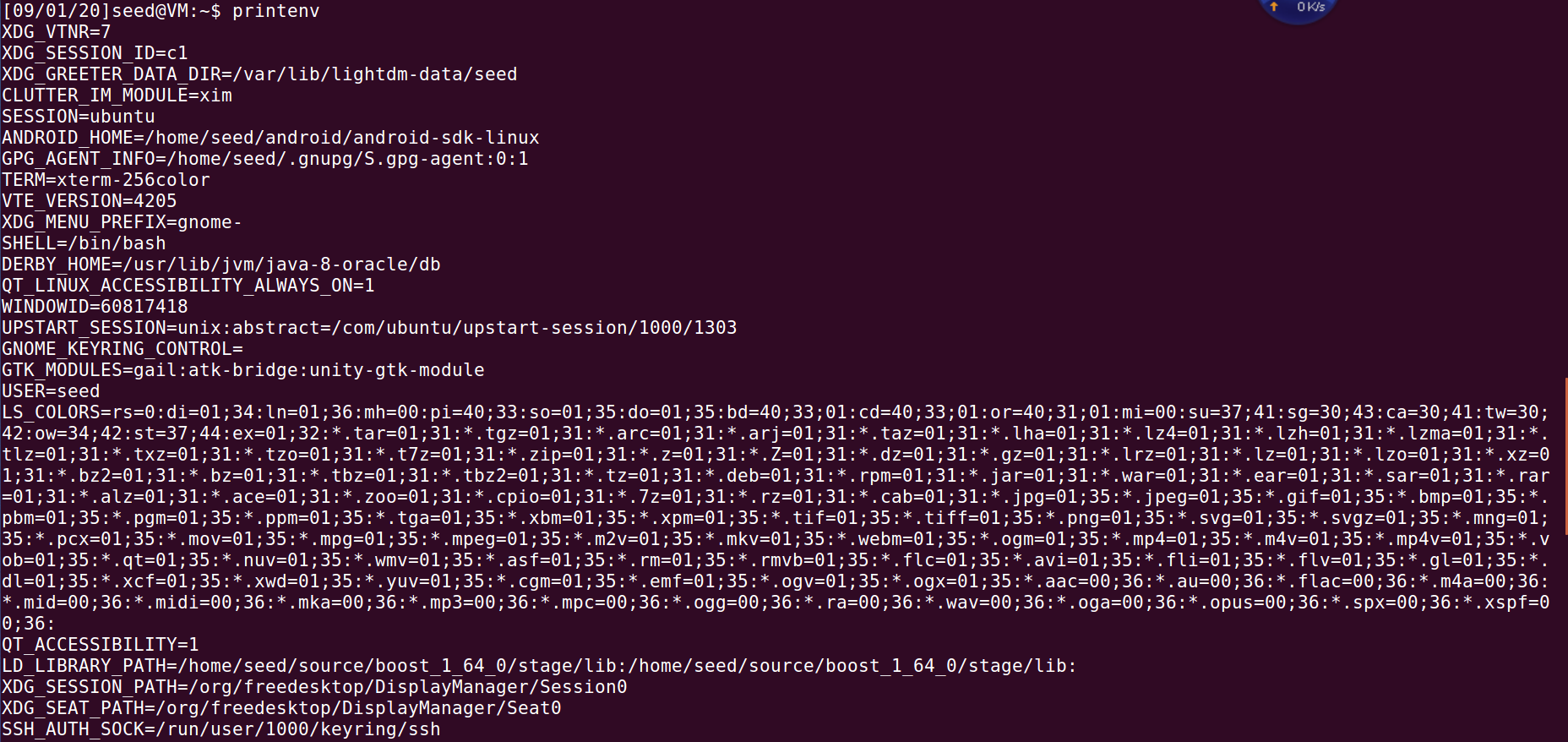
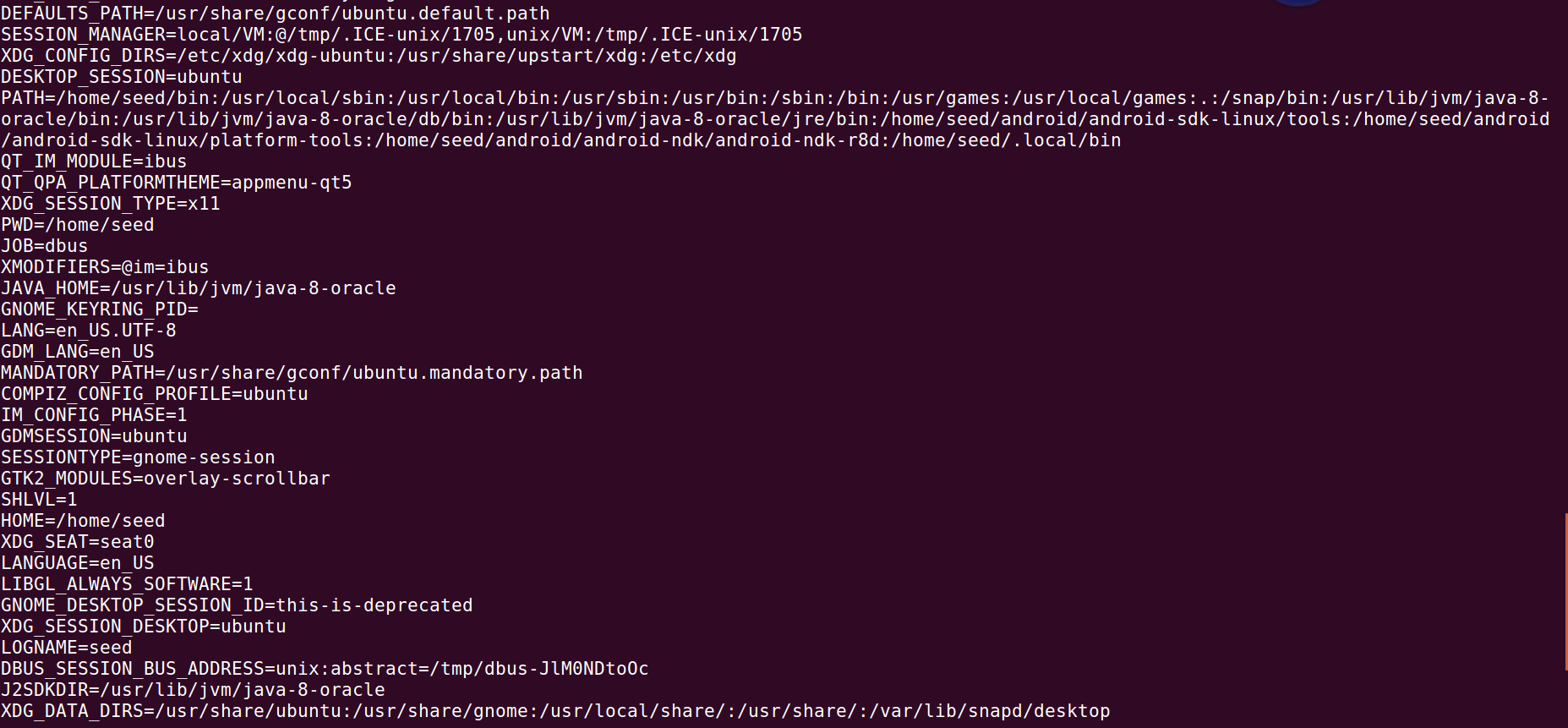
**Environment Variable and Set-UID Program Lab**

**57117230刘玉洁**

**Task 1: Manipulating Environment Variables**

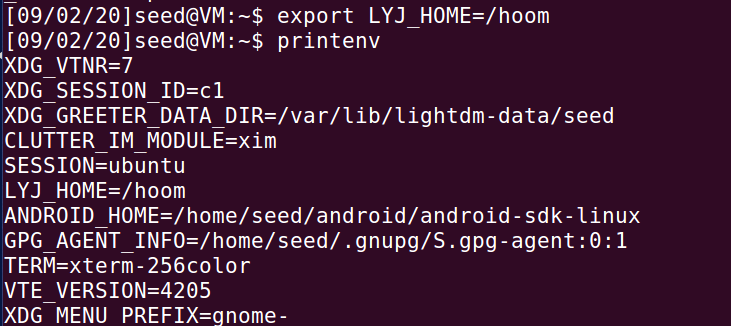
用printenv或env打印出环境变量：







使用export命令设置环境变量：



使用unset命令去掉环境变量：



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

子进程情况：



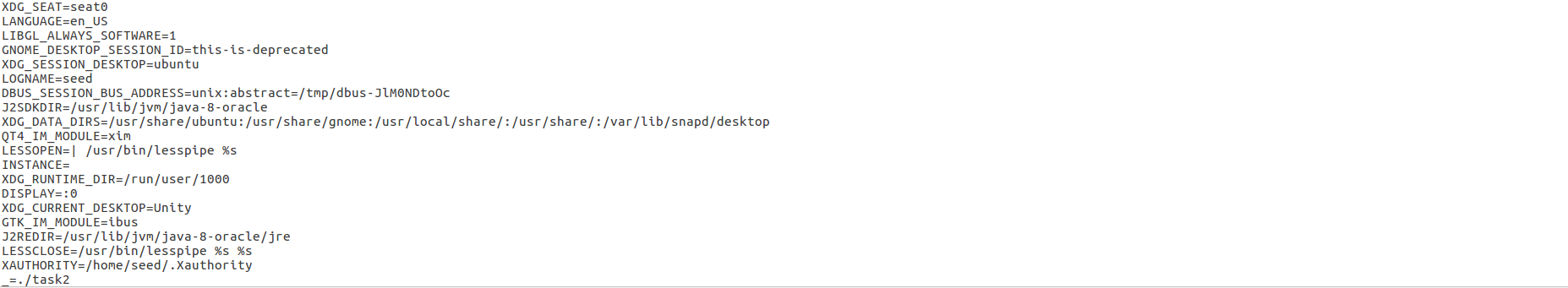




父进程情况：





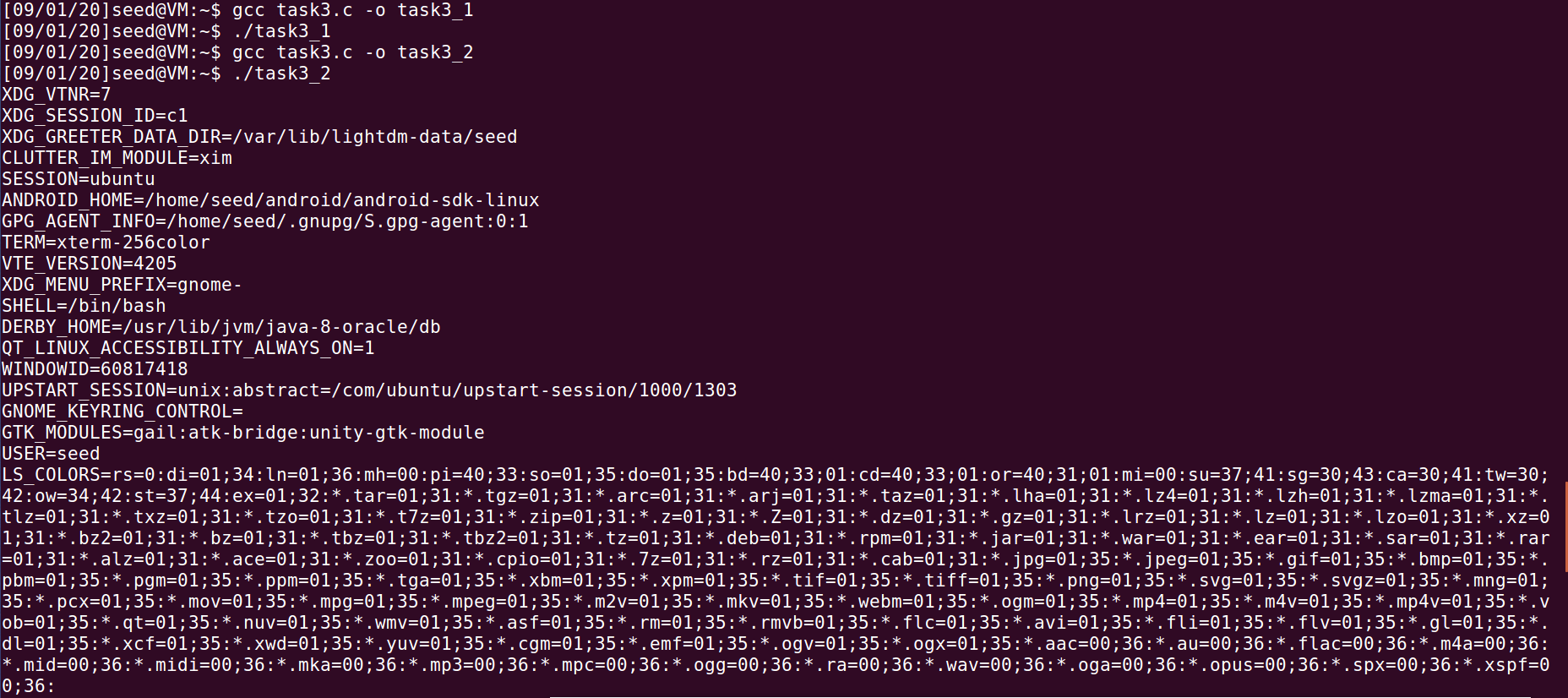


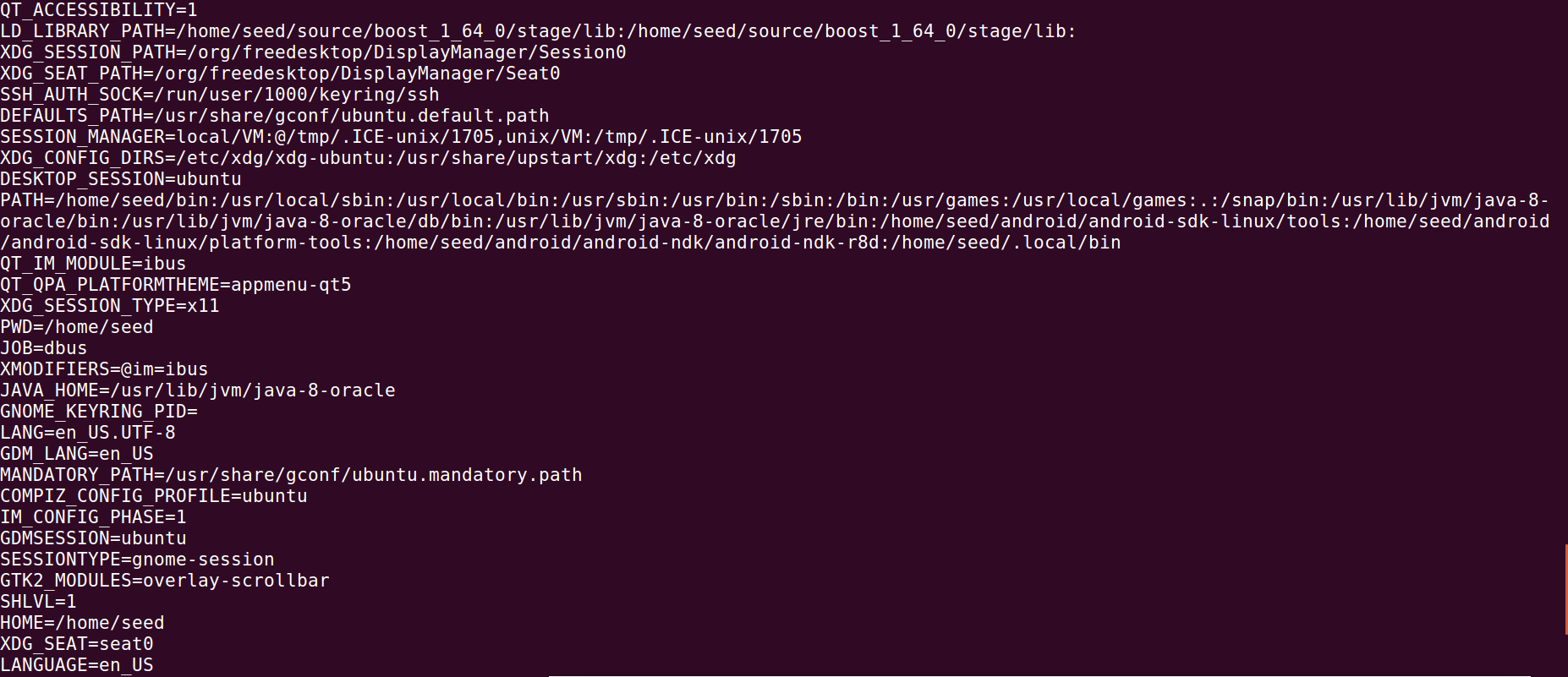
比较：

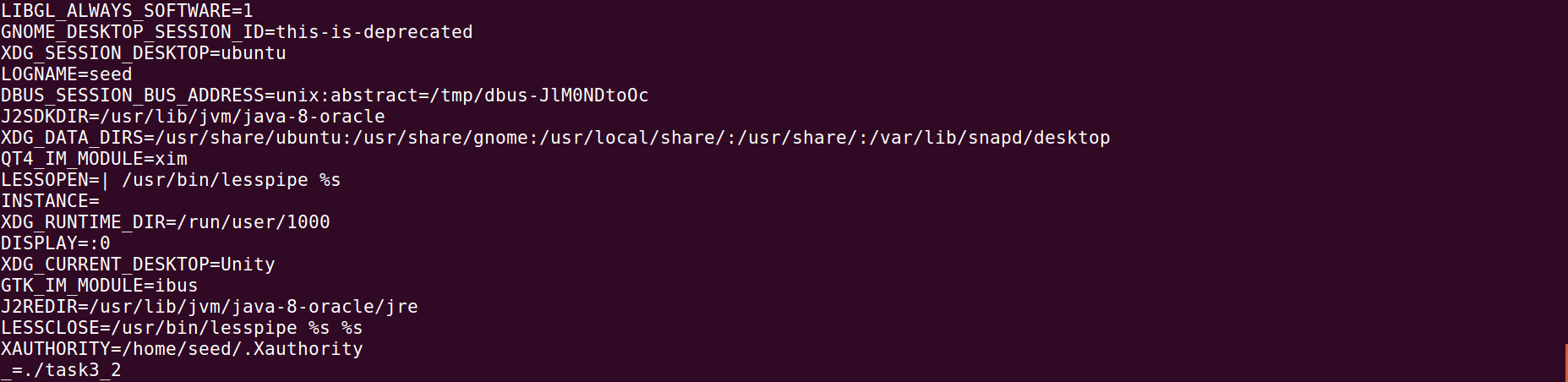


通过比较，可以发现，这两个文件输出的环境变量完全相同，说明原环境变量被子进程完全继承。

**Task 3: Environment Variables and execve()**







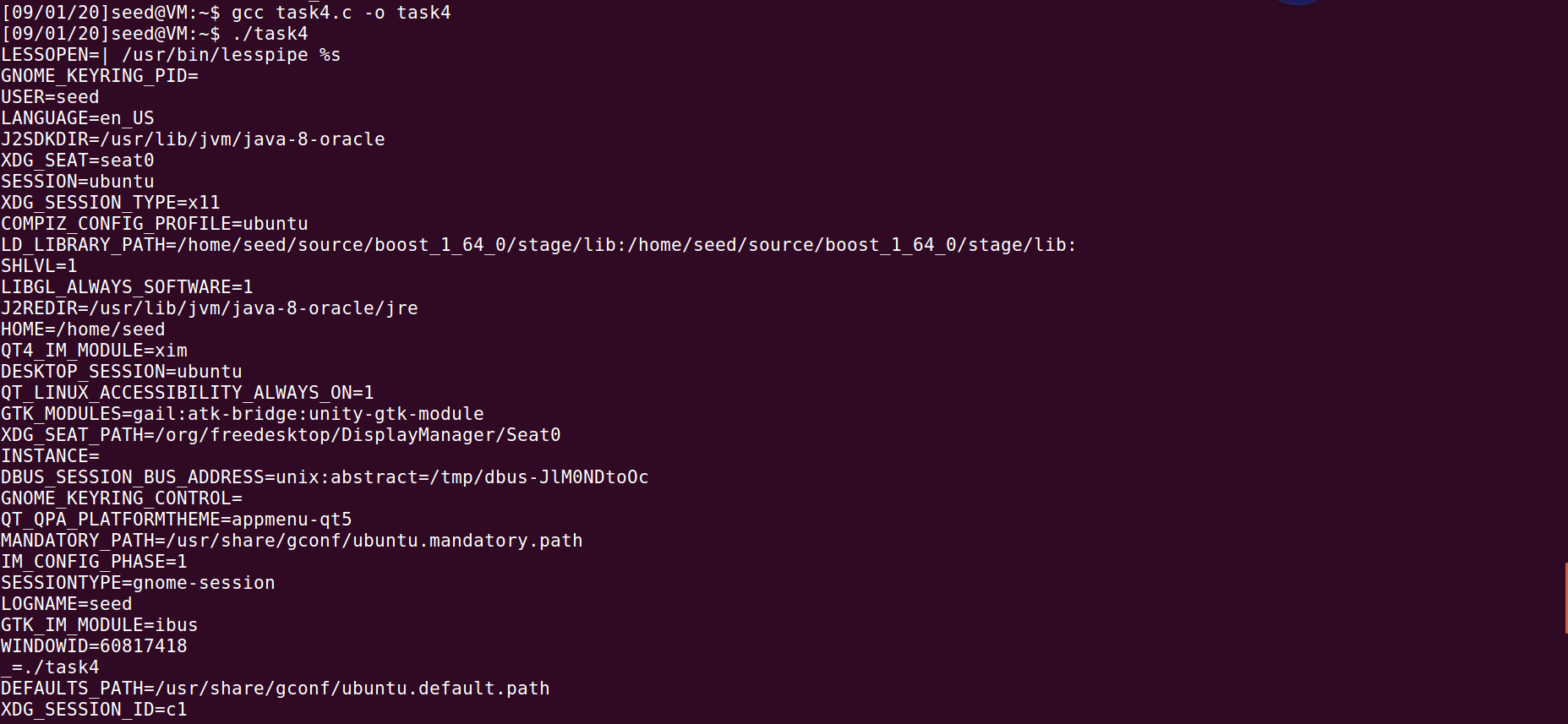
分析：

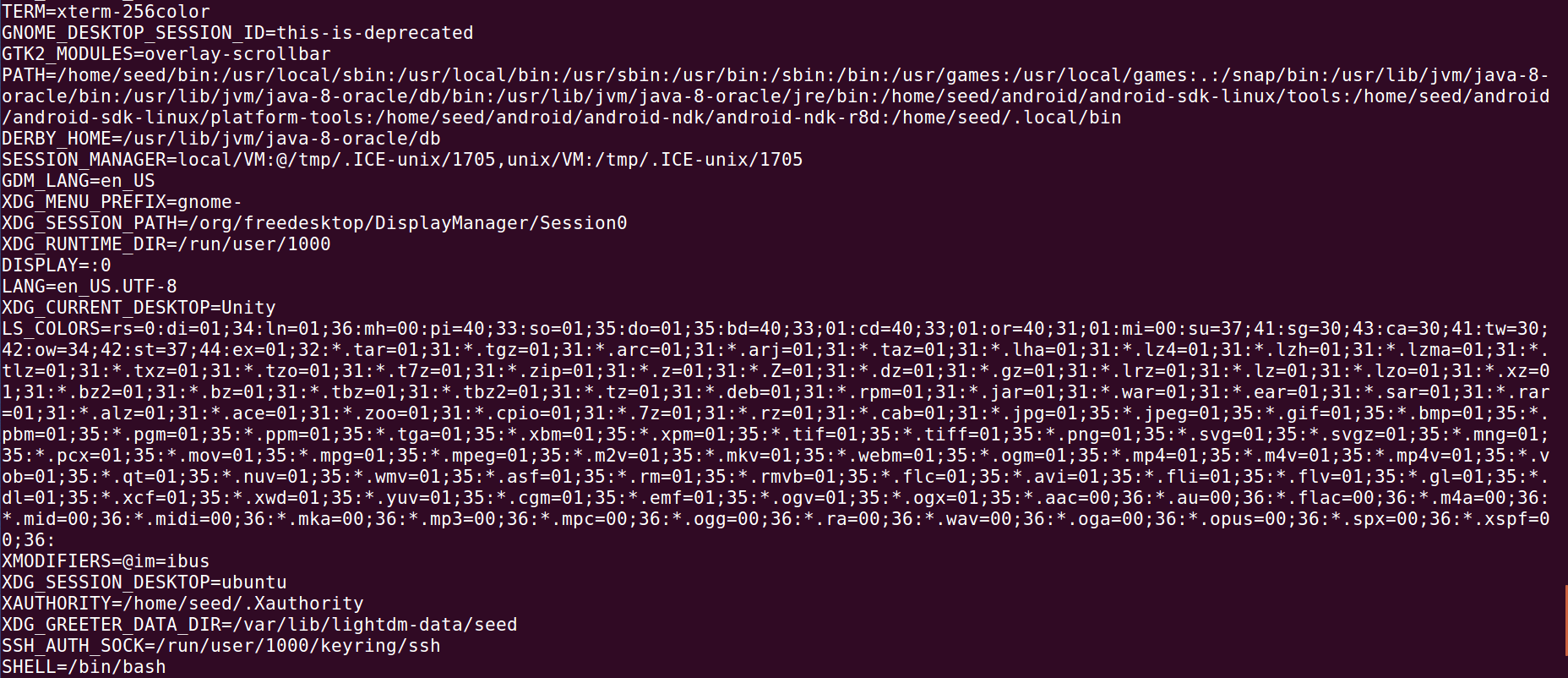
执行第一个execve() 函数，没有输出环境变量。

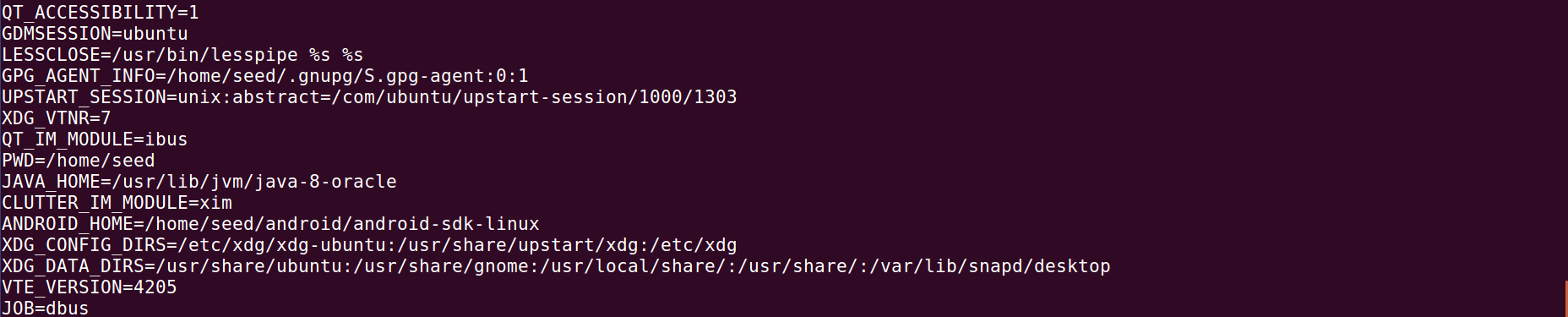
执行第二个execve() 函数，输出当前进程的环境变量。

新进程通过指针数组environ获取环境变量。

**Task 4: Environment Variables and system()**





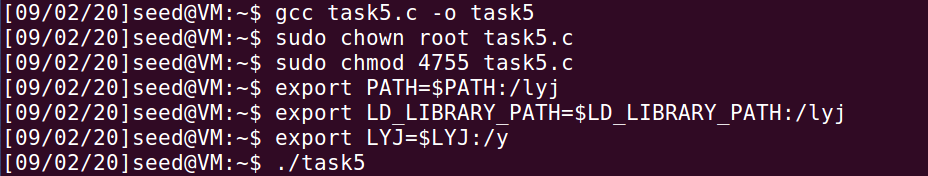


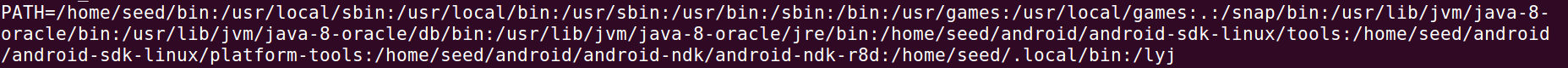
分析：

system函数定义为int system(const char \* string)，该函数调用/bin/sh来执行参数指定的命令，/bin/sh一般是一个软连接，指向某个具体的shell，比如bash,-c 选项是告诉shell从字符串command中读取命令，在该command执行期间，SIGCHLD信号会被暂时搁置，SIGINT和SIGQUIT则会被忽略，意思是进程收到这两个信号后没有任何动作。

System()函数执行了三步操作：①fork一个子进程；②在子进程中调用exec函数去执行command；③在父进程中调用wait去等待子进程结束。若fork失败，system()函数返回-1。如果exec执行成功，也即command顺利执行完毕，则返回command通过exit或return返回的值。如果exec执行失败，也即command没有顺利执行，比如信号被中断，或者command命令根本不存在，system()函数返回127，如果command为NULL，则system()函数返回值非0，一般为1。

**Task 5: Environment Variable and Set-UID Programs**



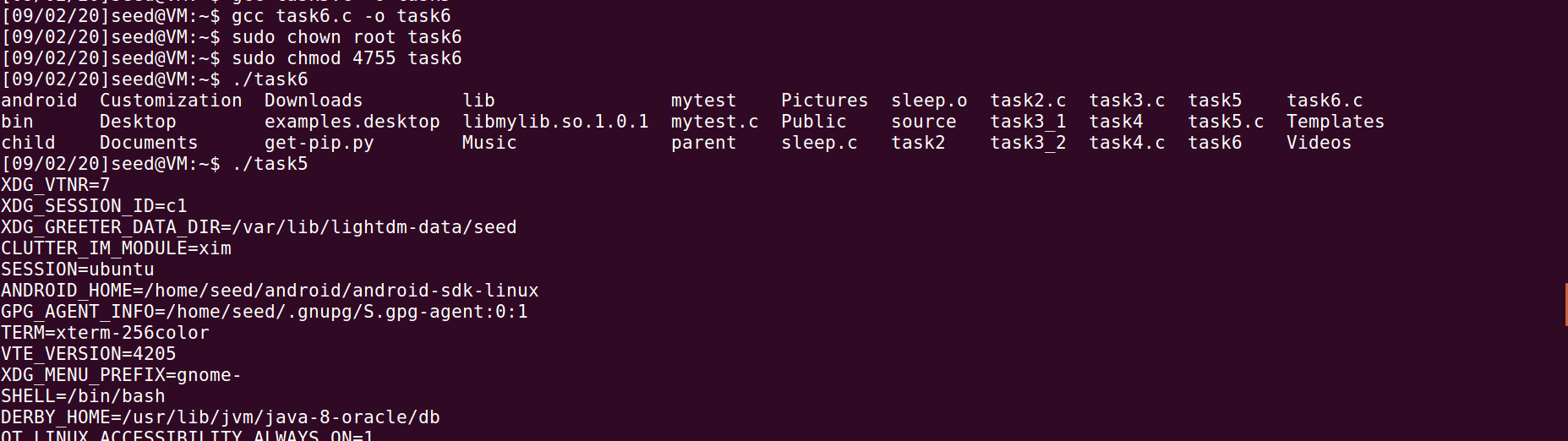






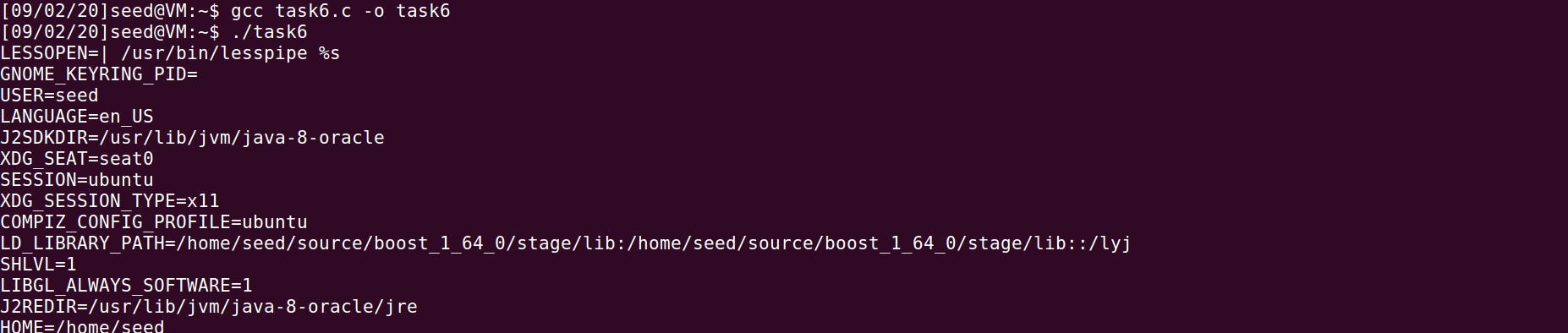
可以看到以上三个被定义的环境变量全部被包括在shell中。

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



运行task6，发现该程序执行的是ls命令。由于system()函数是调用了shell环境变量，运行task5，发现SHELL=/bin/bash。于是将自己的可执行文件夹所在的目录加在了SHELL环境变量的开头：

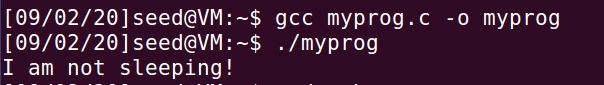




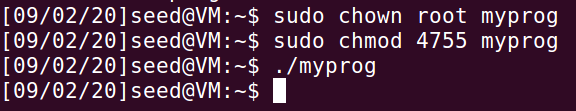
又将task6.c中的system()函数的参数改为task5，重新编译运行./task6，在终端中输出了所有的环境变量，此时，代码不是以root权限运行的。

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

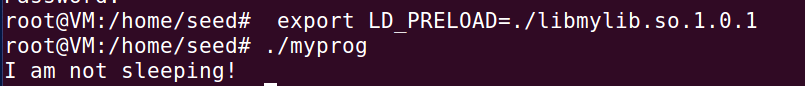
以普通用户的身份运行myprog程序：



以普通用户运行拥有root权限的myprog程序：

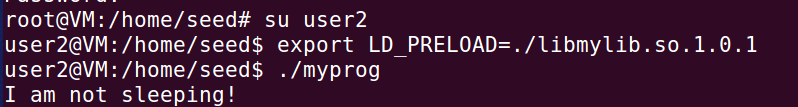


在root下设置环境变量，以root身份运行拥有root权限的myprog程序：



在user2用户中设置LD\_PRELOAD环境变量并运行user1下的myprog程序：

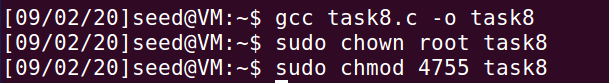




导致他们不同的原因就在于LD\_PRELOAD环境变量。LD\_PRELOAD环境变量是Unix动态链接库中的一个环境变量，它可以影响程序的运行时的链接，允许定义在程序运行前优先加载的动态链接库。这个功能主要是用来有选择性的载入不同动态链接库中的相同函数。在该实验中，mylib.c通过sleep函数，生成了一个libmylib.so.1.0.1链接库，然后将该链接库添加到LD\_PRELOAD环境变量上。比较这四次实验，第一次、第三次和第四次实验在LD\_PRELOAD环境变量中也添加了该链接库。

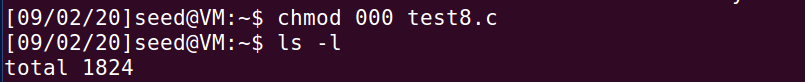
**Task 8: Invoking External Programs Using system() versus execve()**

编译程序，赋予其root用户权限，并将其变为SET-UID程序：



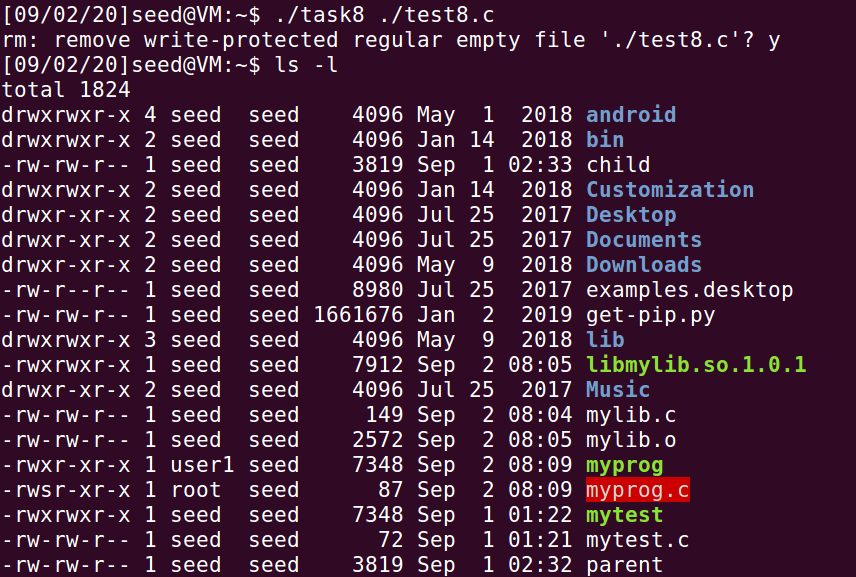
若将代码中的v[0] = "/bin/cat"改为v[0] = "rm"，即删除命令。

新建一个test8.c文件，将其权限改为000：

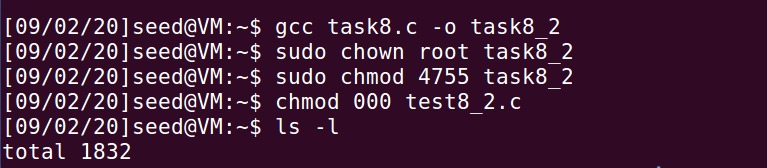




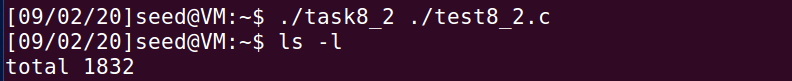
执行以下命令：./task8 ./test8.c，可发现test8.c文件被删除：



注释掉system()语句，并取消注释execve()语句，程序将使用execve()来调用命令：







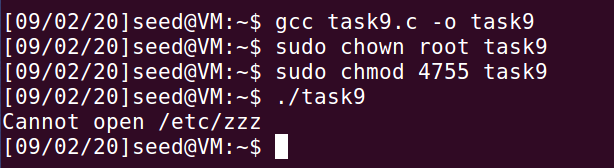


重复之前的方法，文件将无法被删除。

由此可见，由于execve()没有调用shell，在特权程序中使用时没有问题。

**Task 9: Capability Leaking**

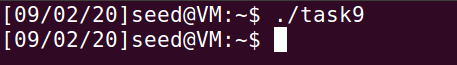
编译程序，赋予其root用户权限，并将其变为SET-UID程序：

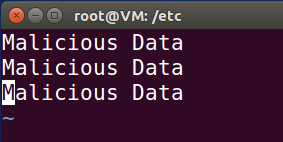


创建/etc/zzz文件：



运行程序：





观察到程序先执行了sleep()函数，随后分别执行父进程和子进程命令，发现/etc/zzz文件可以被修改。

分析：在fork之前打开文件，父进程和子进程可以共享打开的文件描述符，但关闭时可以分别关闭，互相之间不产生影响。