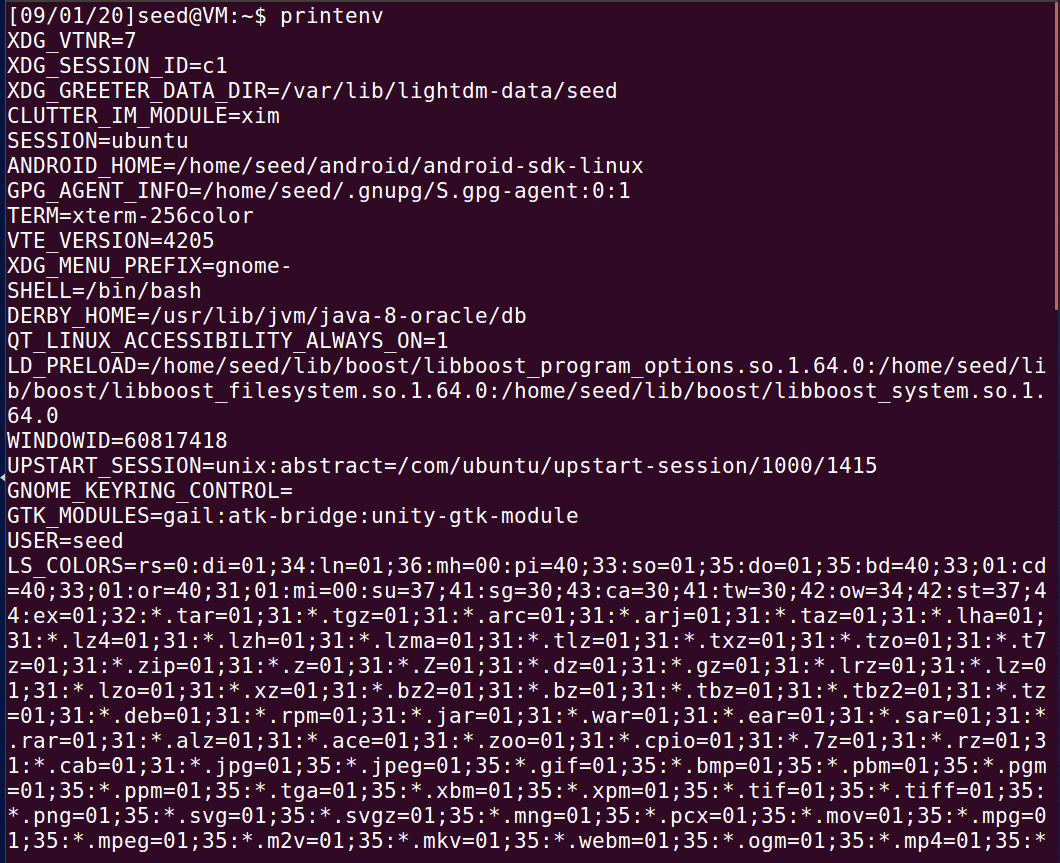
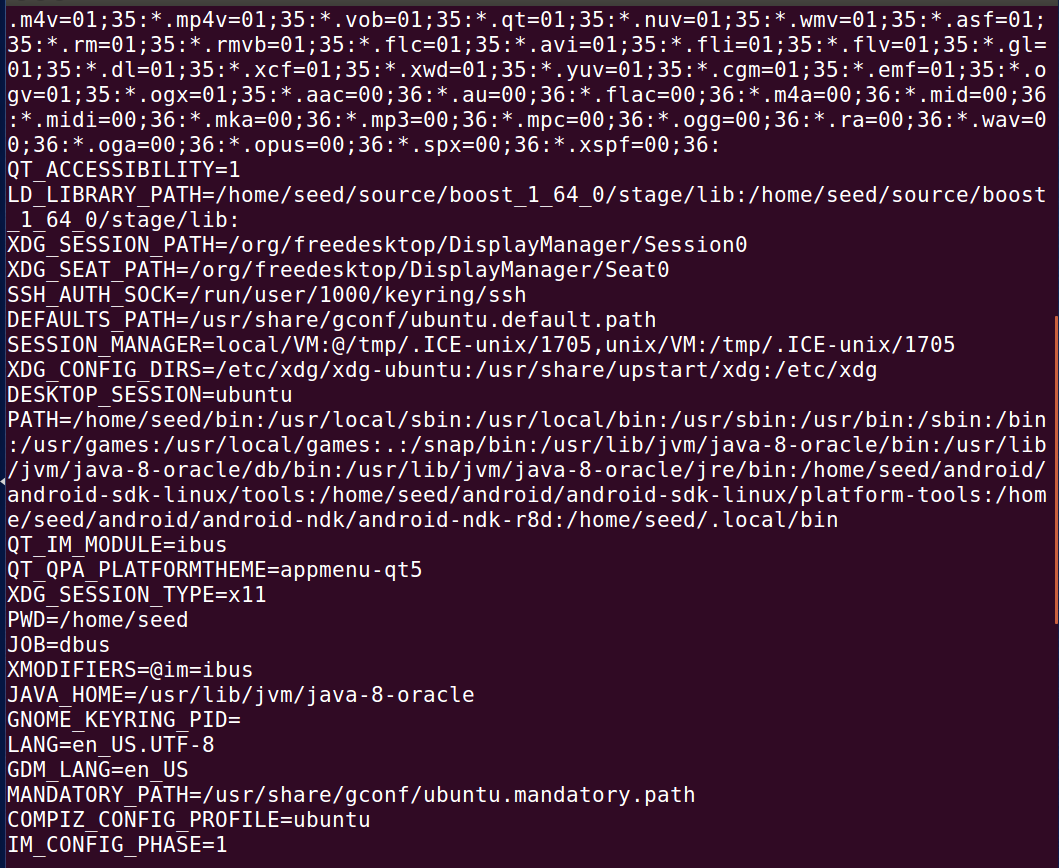
**Lab1-Report**

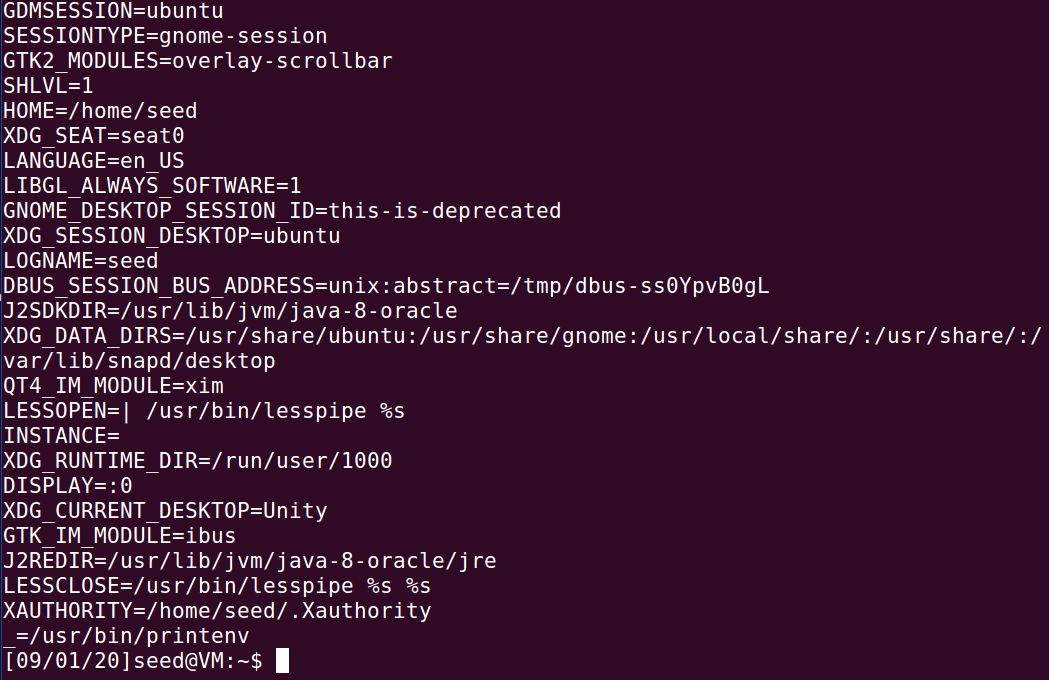
57118116 陈煜

Task 1: Manipulating Environment Variables

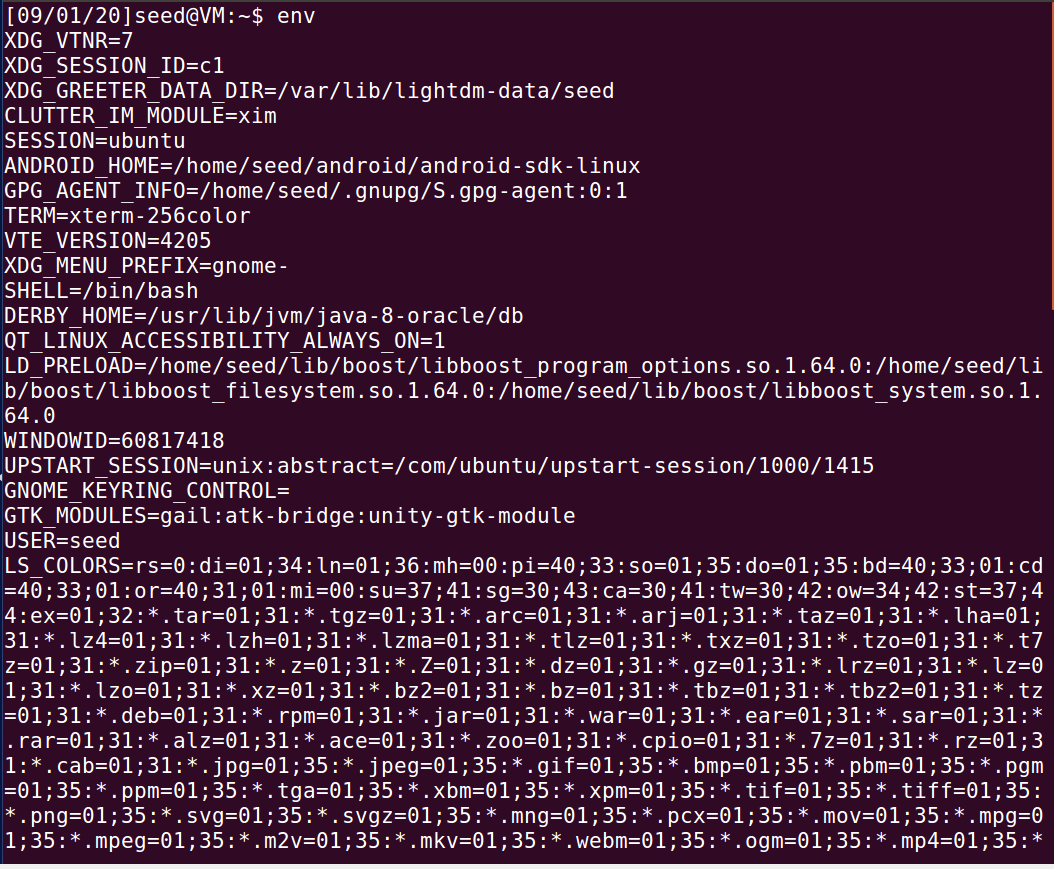
（1）使用printenv查看环境变量：

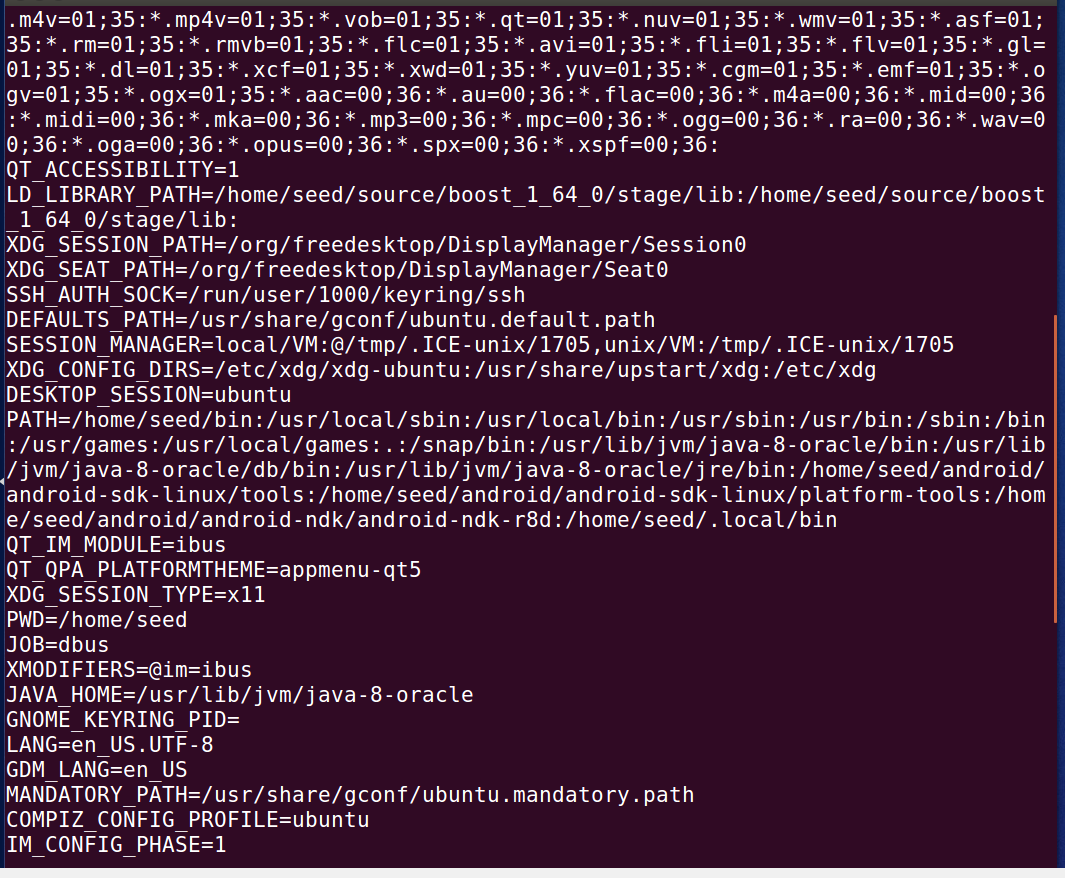


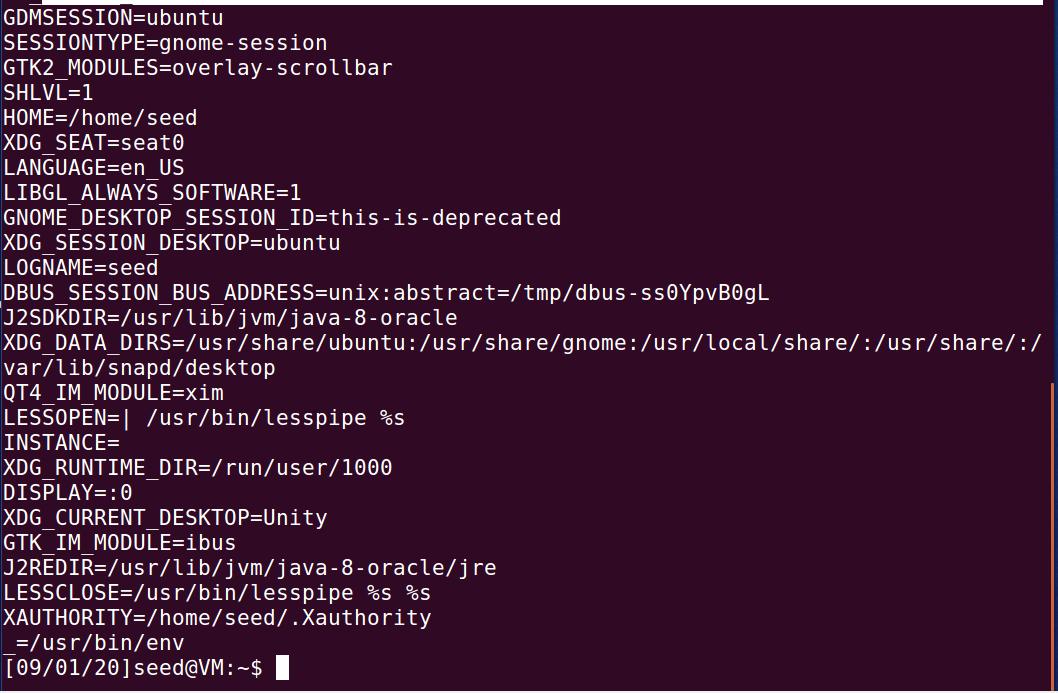




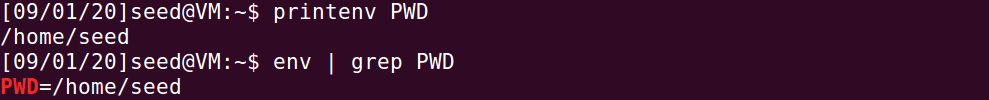
（2）使用env查看环境变量：



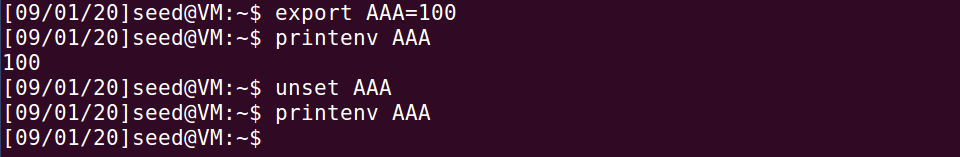




（3）查看环境变量PWD：

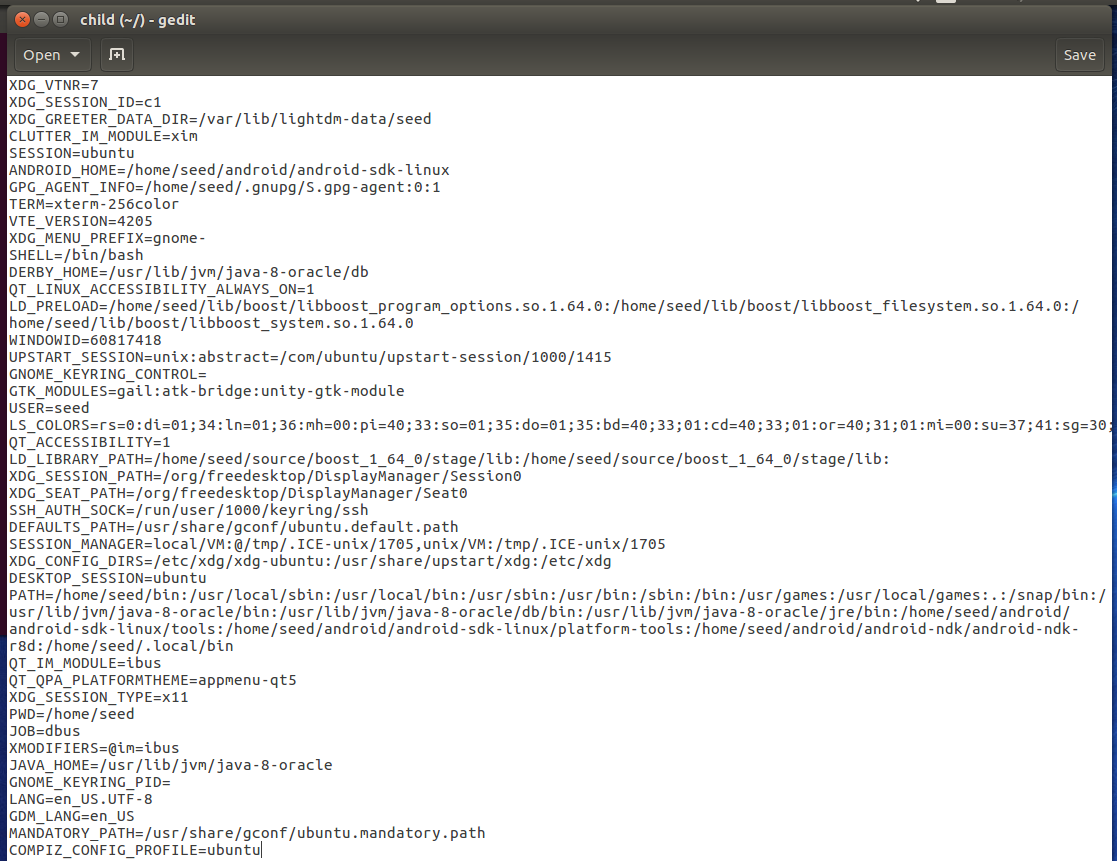


（4）使用export、unset增加、删除环境变量



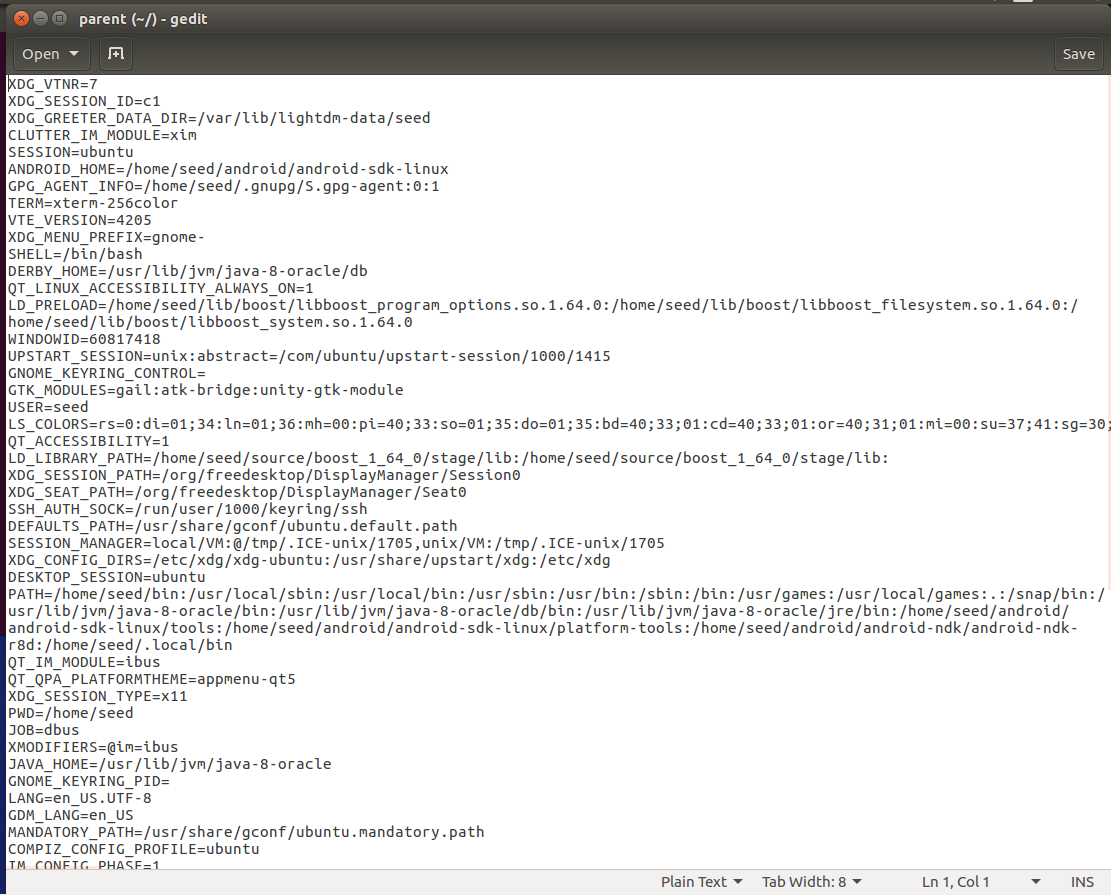
Task 2: Passing Environment Variables from Parent Process to Child Process

Step1: child中的文件：





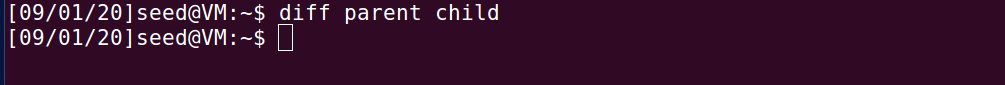
Step2: parent中的文件：





通过观察，两次输出的内容没有差异。

Step3: 比较parent和child两个文件的差别。



通过diff命令，发现parent和child两个文件中的内容相同。说明在默认情况下，子进程会继承父进程的所有环境变量。

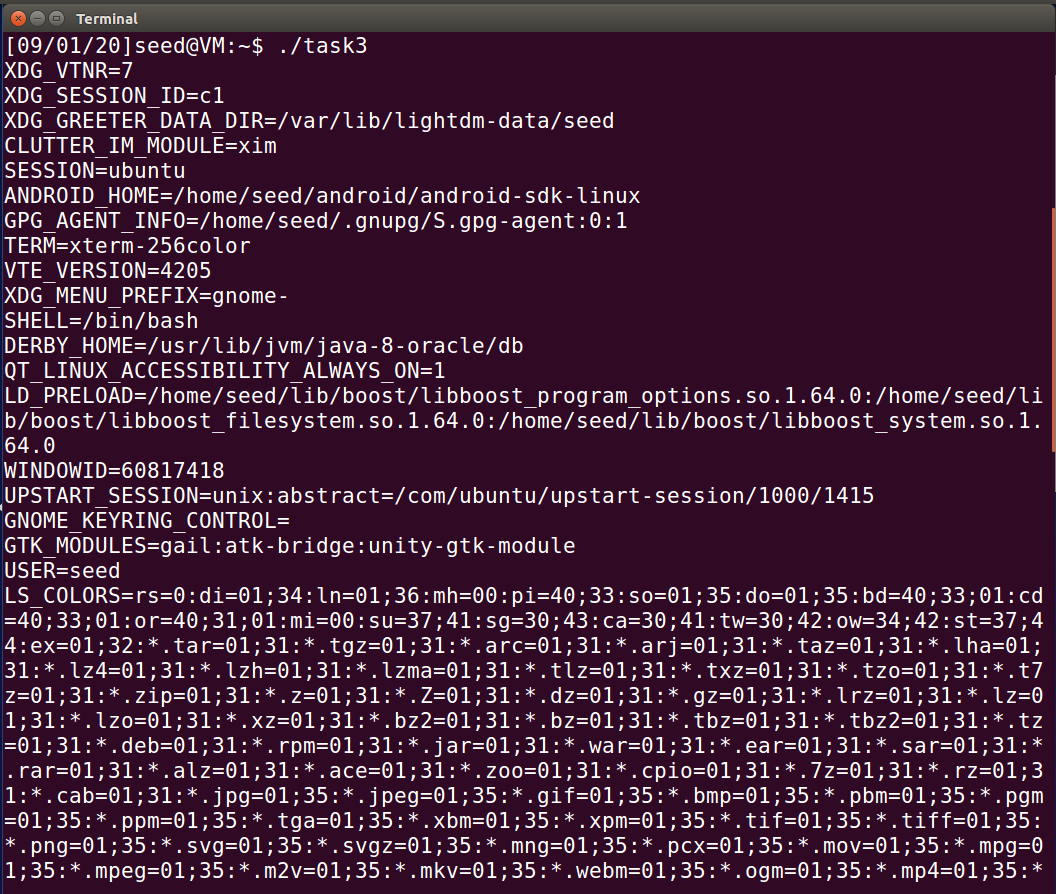
Task 3: Environment Variables and execve()

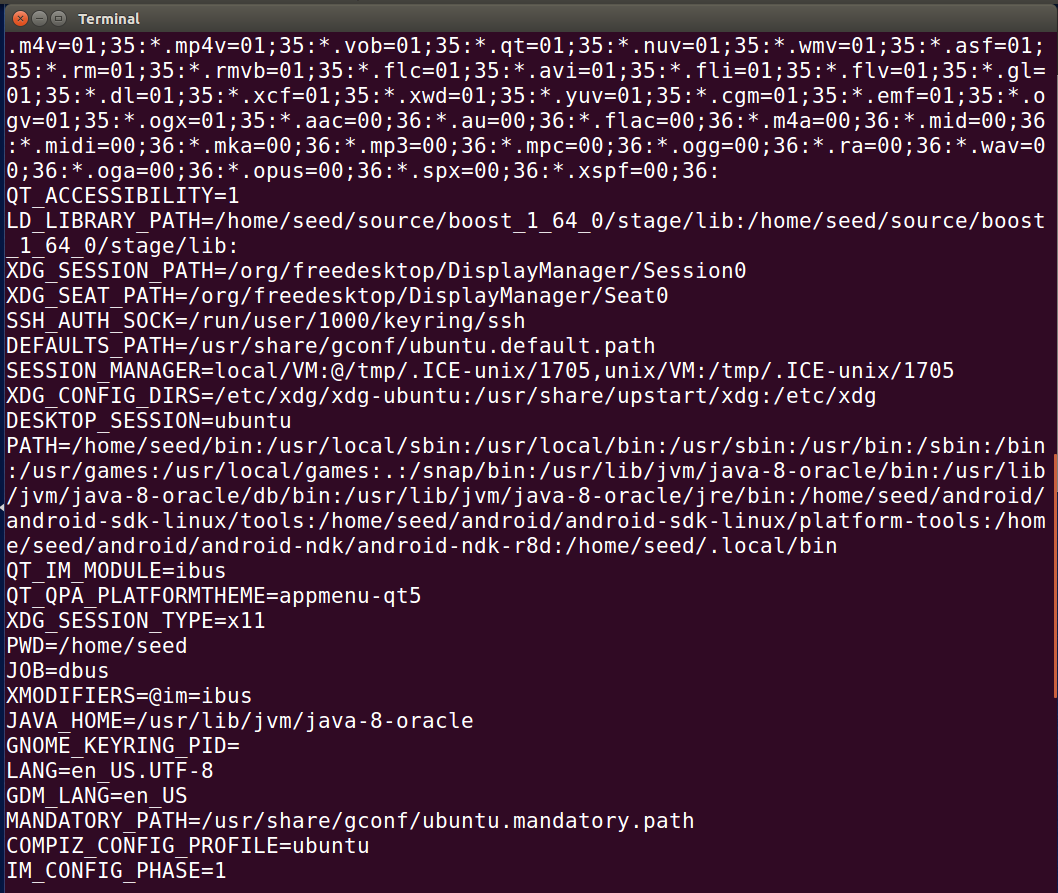
Step1: 运行结果如下：

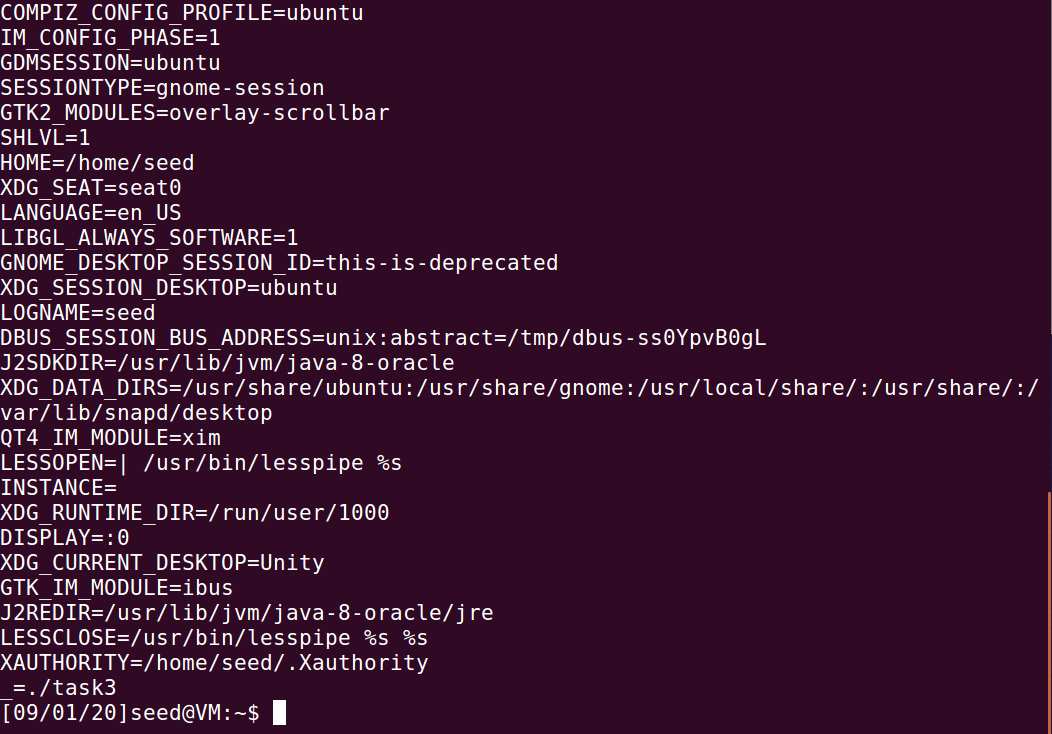


程序运行没有输出.

Step2: 运行结果如下：





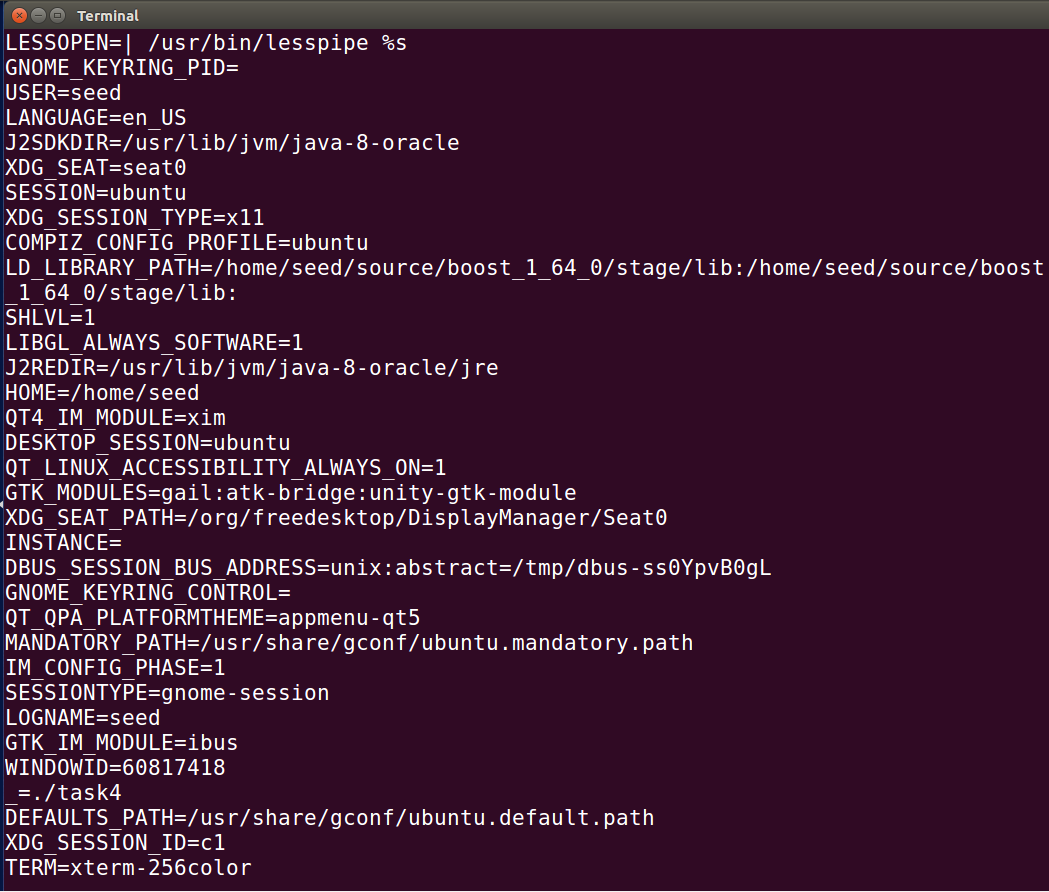


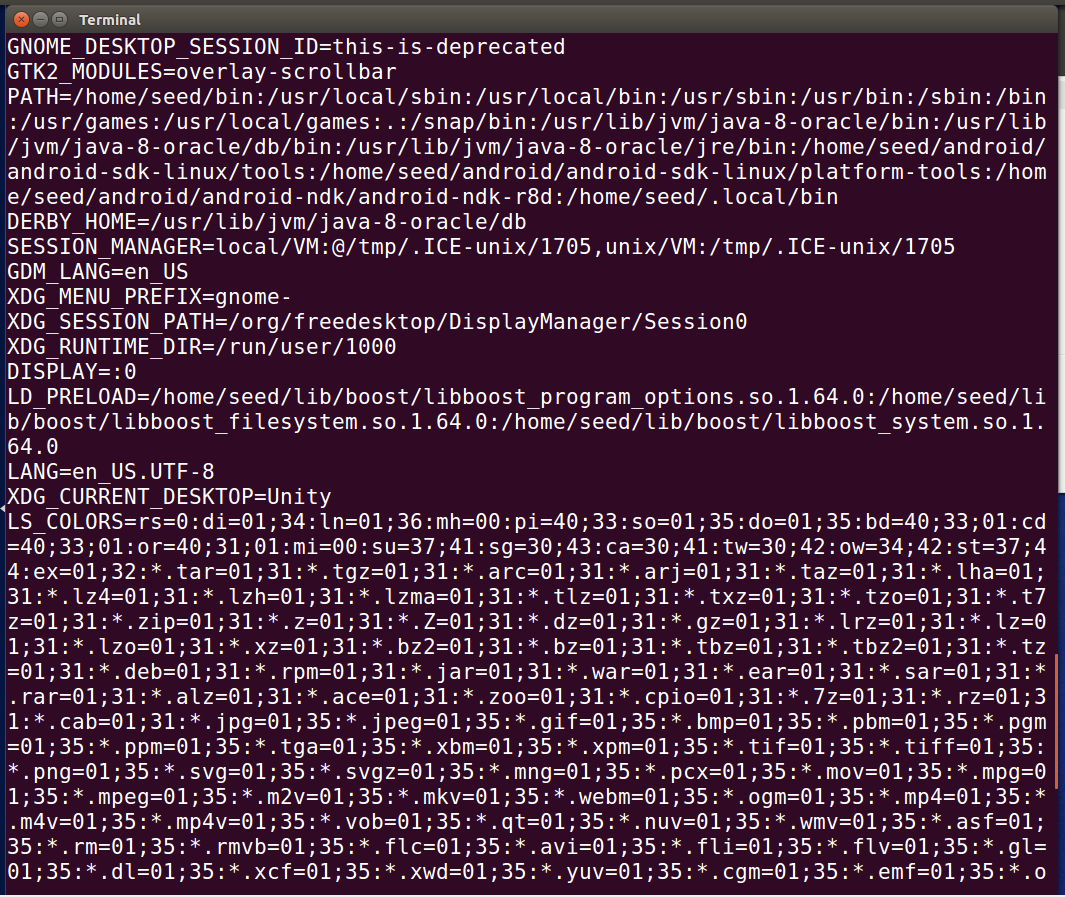
输出结果与命令printenv的输出结果相同。

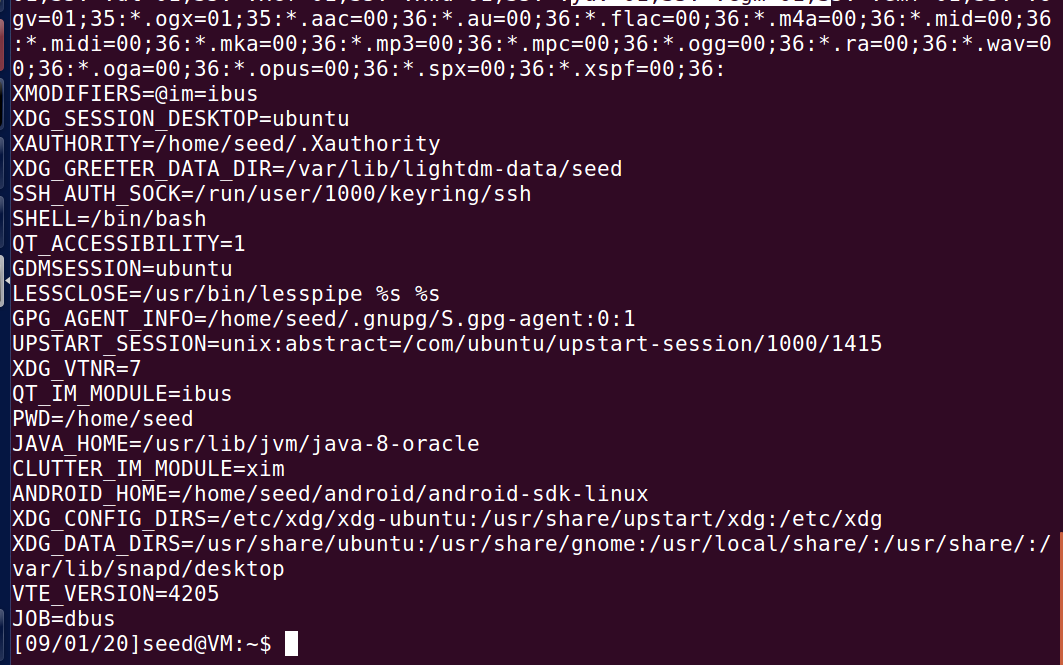
Step3: 结论：使用execve()创建进程时，进程的环境变量由函数的第三个参数决定。当第3个参数是NULL时，进程不继承环境变量；当第3个参数是environ时，进程的环境变量为全局的环境变量。

Task 4: Environment Variables and system()

编译后，程序运行结果如下：



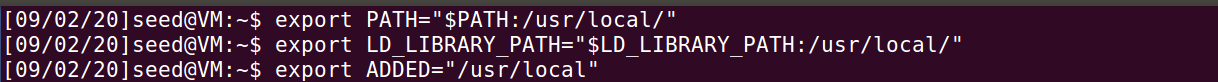




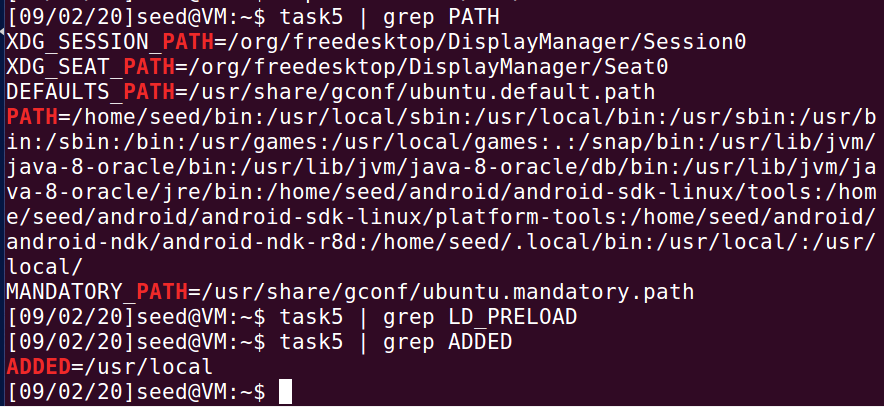
System通过execl调用execve进而调用shell，自动传递环境变量。

Task 5: Environment Variable and Set-UID Programs

程序命名为task5, 添加的两个变量为



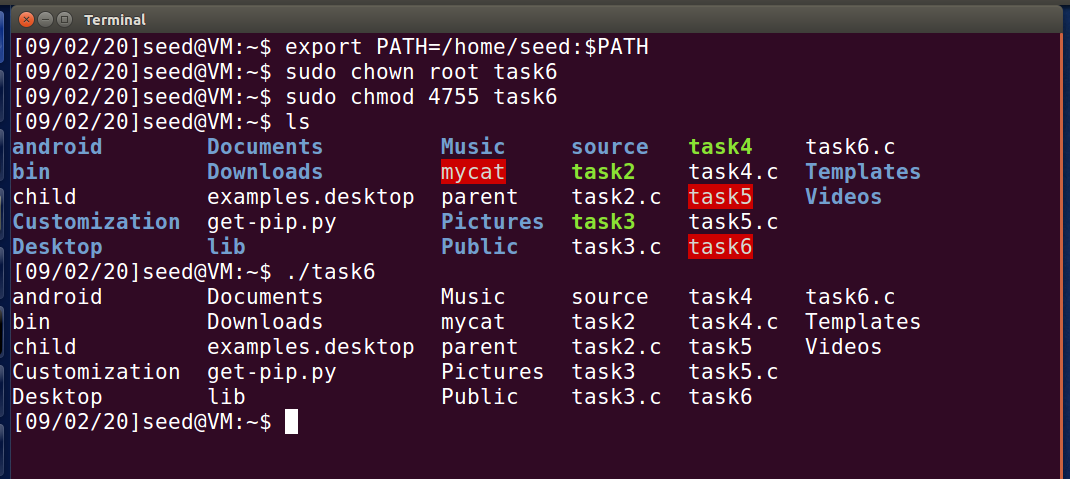
运行task5之后，输入如下命令：



可以发现，子进程没有继承父进程的LD\_LIBRARY\_PATH环境变量，但是继承了自定义环境变量ADDED，也继承了PATH。

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

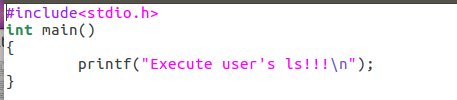
运行结果如下：



运行task6输出的结果和通过命令行运行ls的结果相同，可见程序运行的是/bin/ls。

可以通过如下方式，让task6运行用户写的ls代码，而不是/bin/ls。

编写新的ls.c程序：

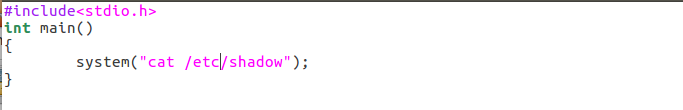


编译后将其拷贝到/bin/目录下。再次运行task6程序，结果如下：

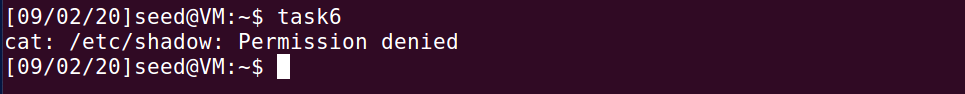


运行的是用户定义的ls程序。

如果使用root权限，改写ls程序：



将其拷贝到/bin/目录下，运行task6的结果：

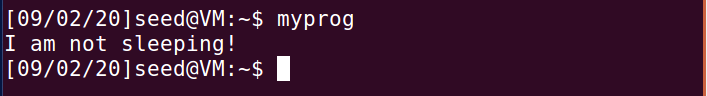


被阻止。

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。

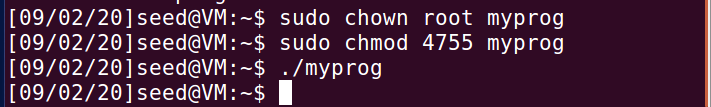
Task 7: The LD PRELOAD Environment Variable and Set-UID Programs

（1）Make myprog a regular program, and run it as a normal user.结果：



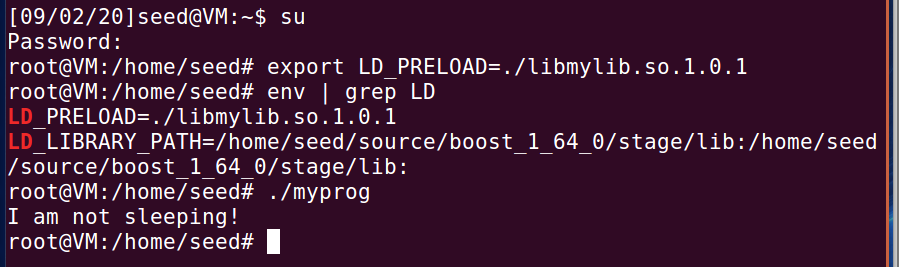
运行了动态链接库中定义的sleep。

（2）Make myprog a Set-UID root program, and run it as a normal user.结果：



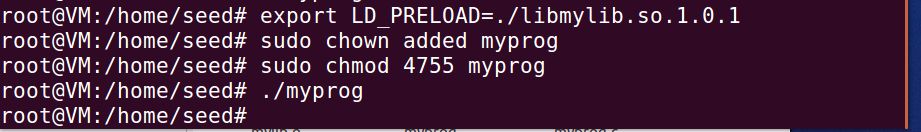
运行的是系统自带的sleep。

（3）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中定义的sleep。

（4）创建一个新用户added。运行结果如下：



运行的是系统自带的sleep。

Step3:

在（1）中，程序为普通程序，myprog继承了shell中定义的环境变量LD\_PRELOAD。

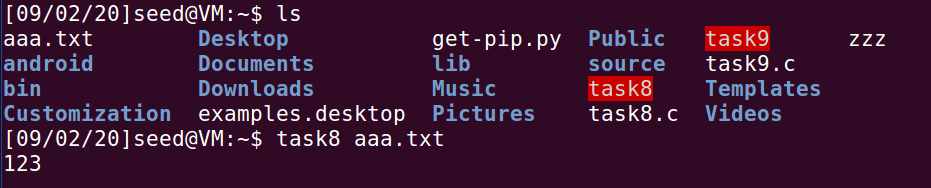
在（2）中。以普通用户的身份执行Set-UID程序，程序的子进程不会继承父进程动态链接库LD\_PRELOAD环境变量。

在（3）中，以root的身份执行程序，shell向myprog传递了LD\_PRELOAD环境变量。

在（4）中，以seed的身份执行added的程序myprog，不会向added传递在seed中添加的环境变量LD\_PRELOAD。

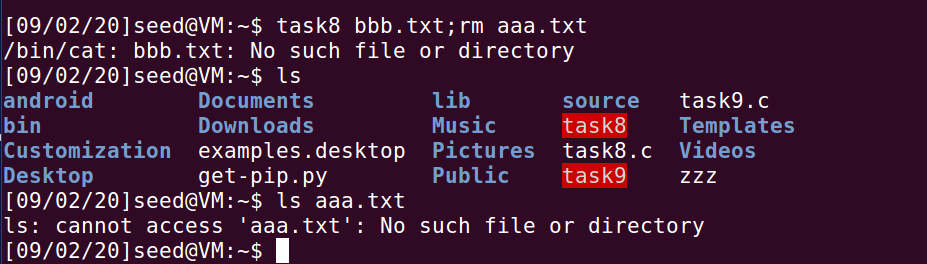
Task 8: Invoking External Programs Using system() versus execve()

Step1: 编译task8后，在当前目录下创建“aaa.txt”，使用task8，运行结果如下：



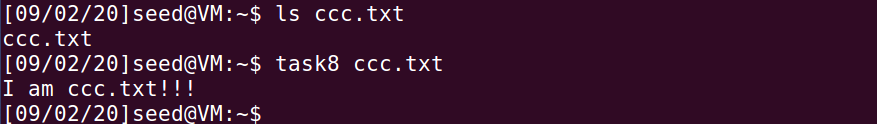
能查看文件“aaa.txt”。

“bbb.txt”文件不存在。运行如下命令：



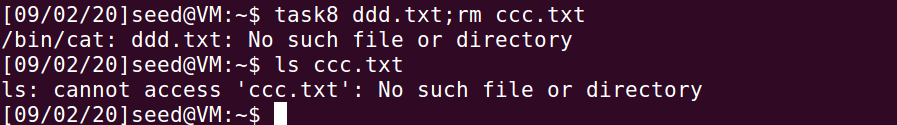
可以发现“aaa.txt”文件被删除。

Step2:修改代码后重新编译task8。在当前目录下创建“ccc.txt”，使用task8，运行结果如下：



能查看文件“ccc.txt”。

“ddd.txt”文件不存在。运行如下命令：

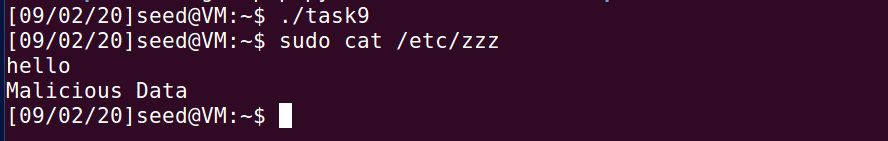


可以发现“ccc.txt”文件被删除。

Task 9: Capability Leaking

创建zzz文件，向文件中写入“hello”，并拷贝到/etc/zzz中。

运行task9，再查看/etc/zzz：



/etc/zzz文件被修改。

原因：尽管task9执行过程中，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。