SEED Labs

Environment Variable and Set-UID Program Lab

Author 崔英杰

Env

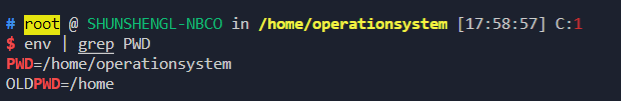
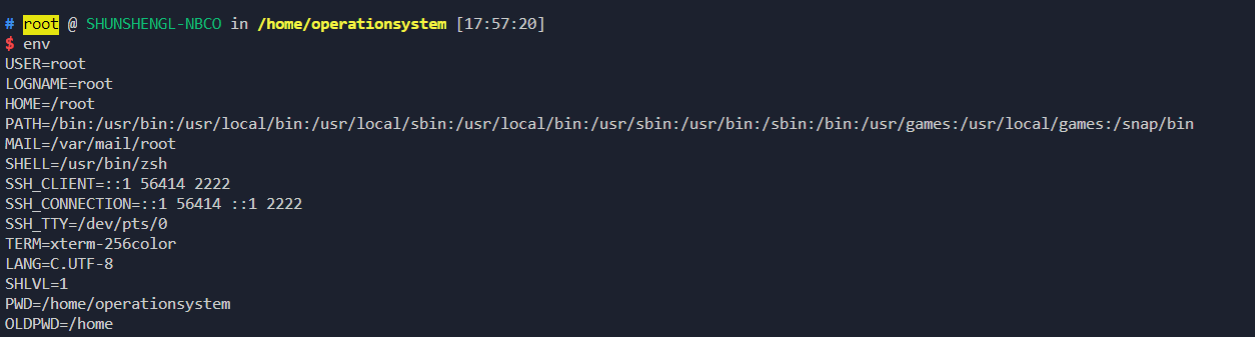
System: Ubuntu 18.0.4 LTS In Windows10(WSL2)

Editor: Visual Studio Code

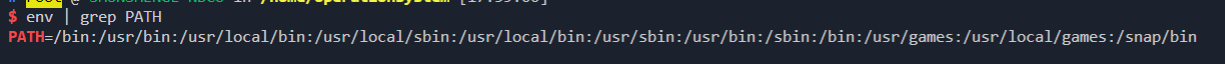
User: seed & root

Lab Tasks

Task 1: Manipulating Environment Variables

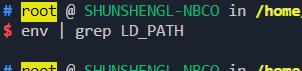
Sample：用export 和unset 增减环境变量

1.PATH添加/home/seed

改变前：改变语句：

改变后：2.移除环境变量LD\_PATH

改变前：

改变后：该变量已被删除

Task 2: Passing Environment Variables from Parent Process to Child

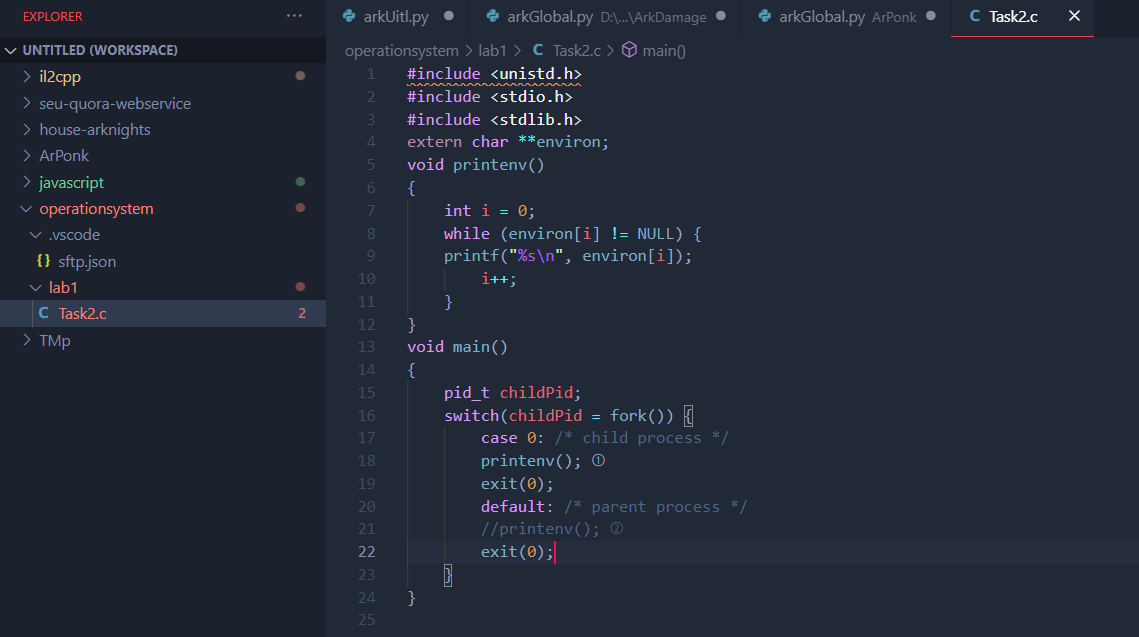
Process

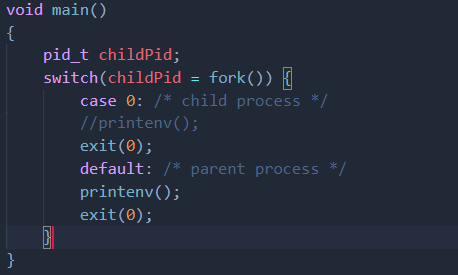
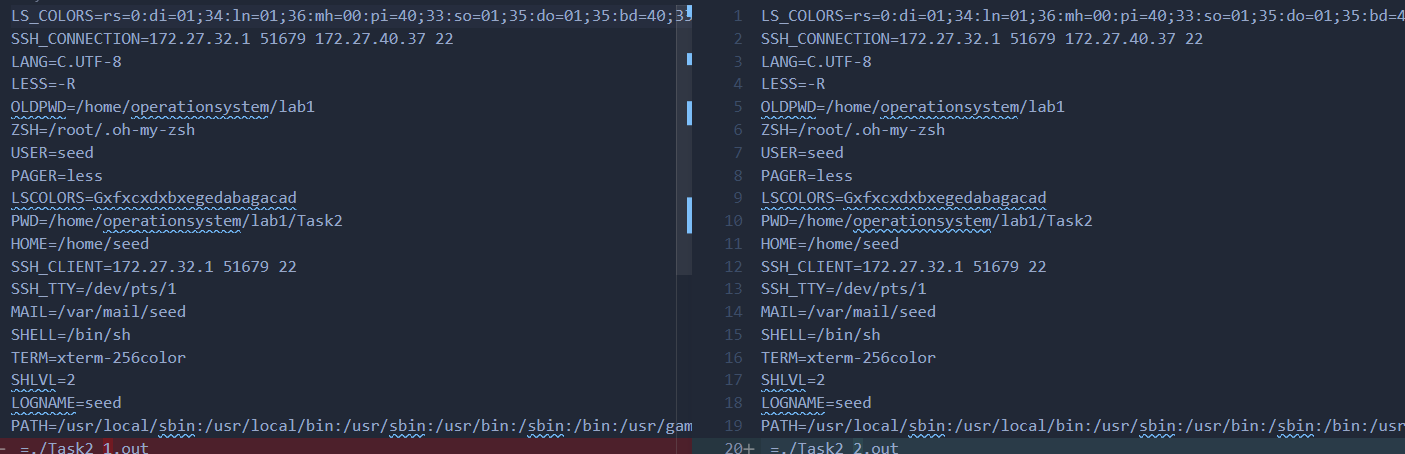
Useage

切换到seed用户

保存文件 通过sftp自动同步

编译运行Task2.c

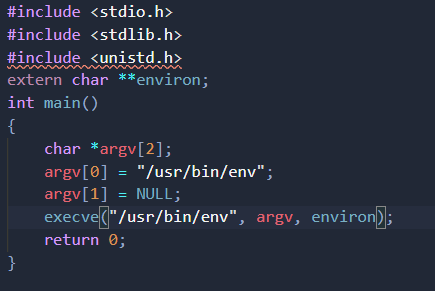
su seed保存文件 通过sftp自动同步

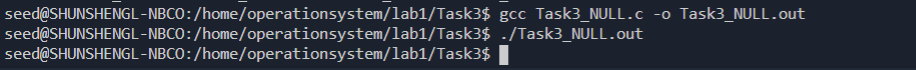
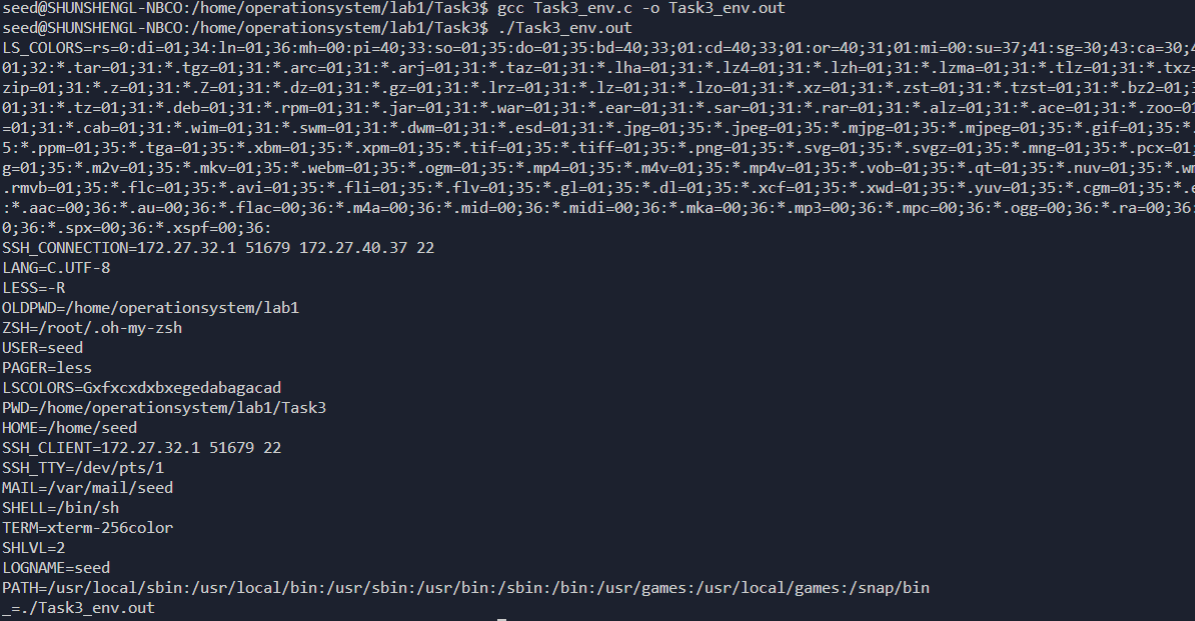
编译运行Task2.c将Line 1版本的运行结果保存在文件 child\_1 中修改注释，（保存在同一文件目录下）编译运行 Task2\_2.out 的运行结果保存在 child\_2 之中同步在vscode之中进行比较，可以很明显地看到环境变量是被继承了的（不同之处仅是运行程序的文件名）

Task 3: Environment Variables and execve()

Useage

使用seed用户

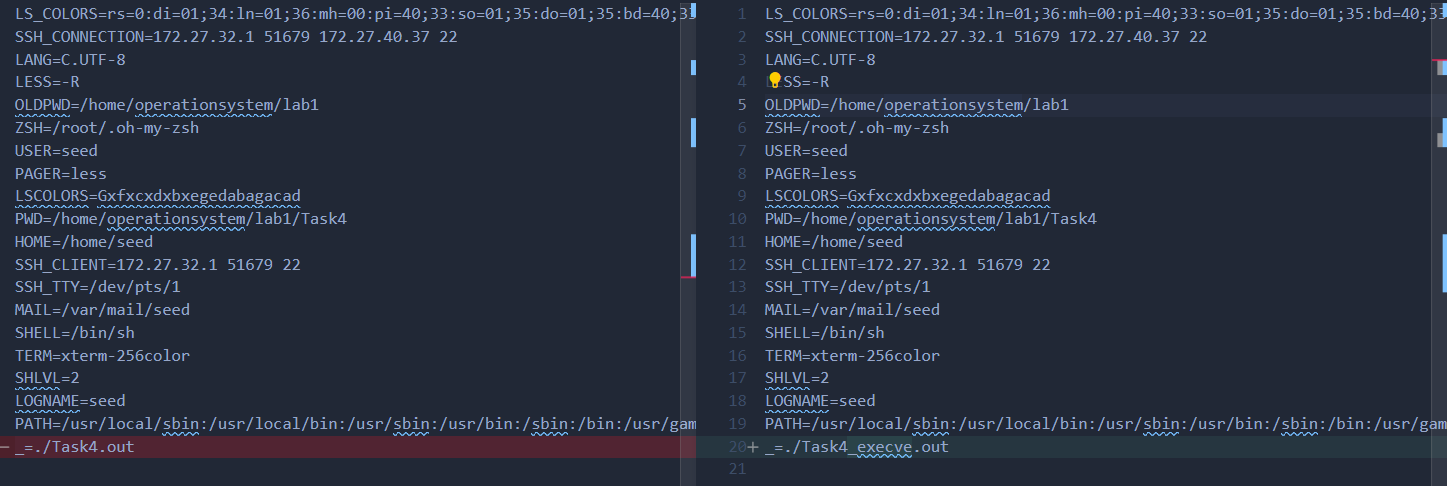
Task3\_NULL.cTask3\_env.c编译运行后

execve第三参数为NULL版本没有输出execve第三参数为environ版本有输出可以发现后者成功输出了当前进程的环境变量，而前者失败了。

Task 4: Environment Variables and system()

Useage

使用seed用户

直接编译运行，将结果保留在 4.txt同时生成一份利用 execve 函数调用得到的结果，保存在 4.execve.txt比较两个文件：除开运行的文件命以外，环境变量一样，即输出的环境变量为当前程序的环境变量

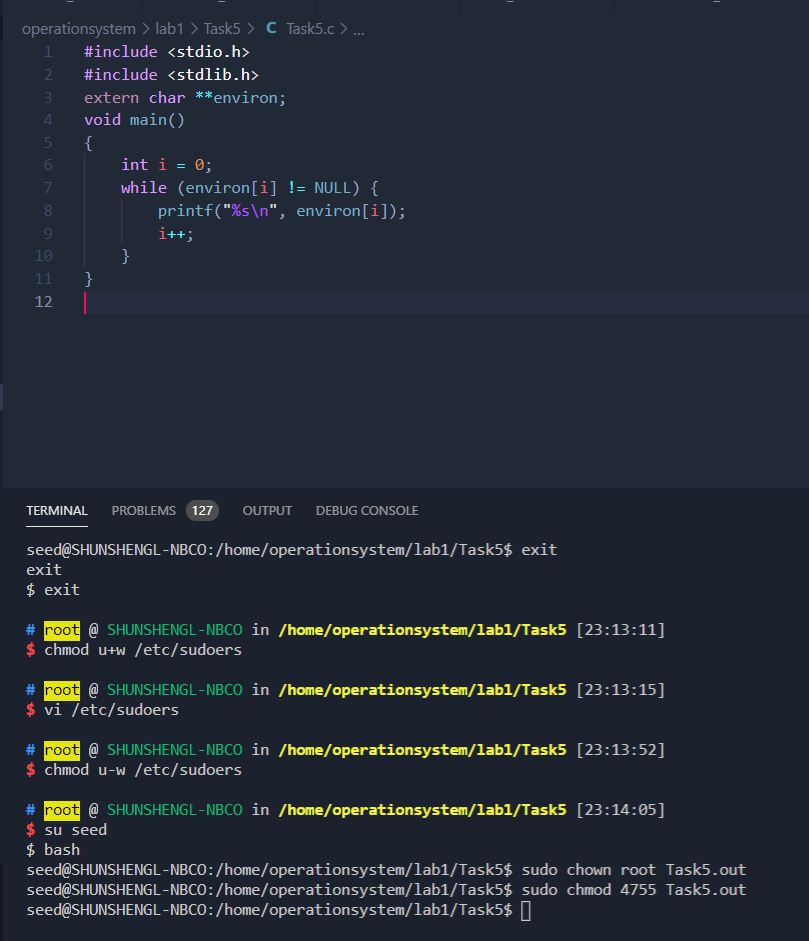
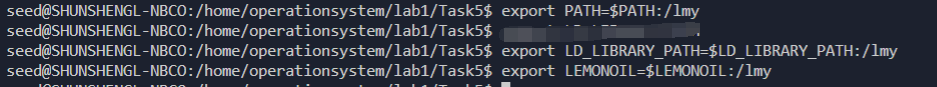
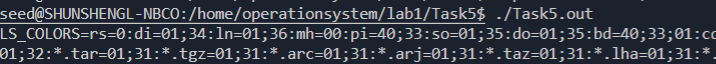
Task 5: Environment Variable and Set-UID Programs

Useage

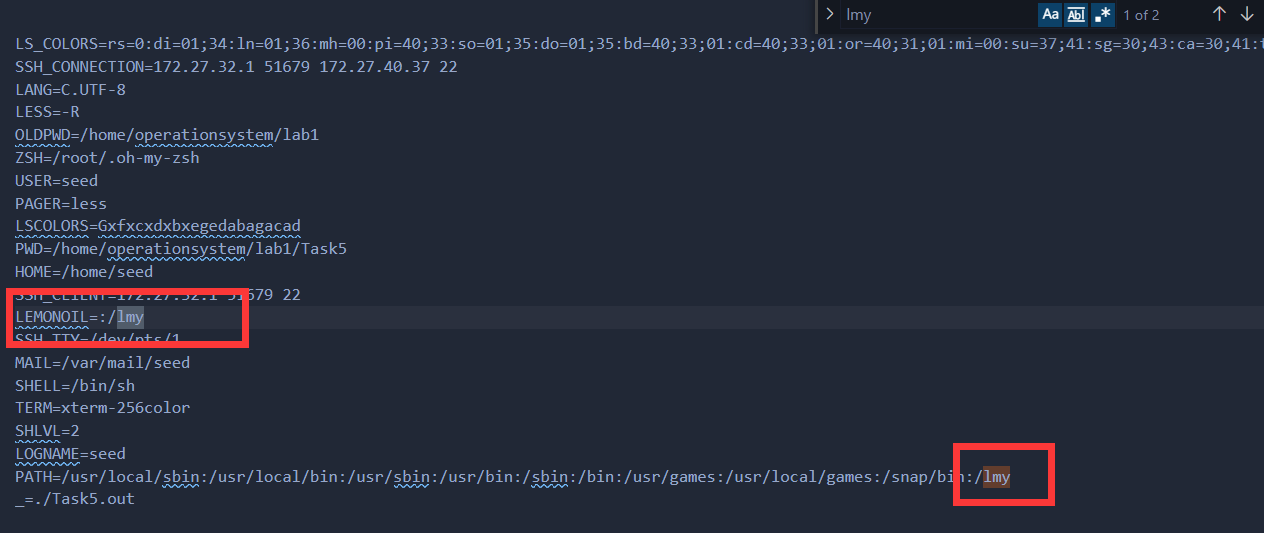
此Task需要切换用户，但通过sudo可以避免

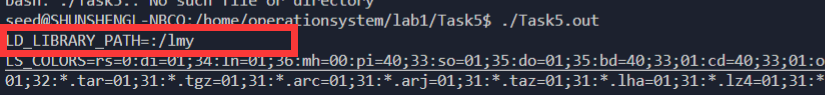
使用seed用户（已通过root给seed添加sudo权限）

编译得到可执行程序Task5.out

并完成Step2,给程序手动提权export step3中所要求的环境变量运行程序，得到结果：可以很明显得看到

PATH

ANY\_NAME(LEMONOIL) 当然LD\_LIBRARY\_PATH并没有显示出来，原因在于 LD\_LIBRARY\_PATH 是linux自带的用于指定查找动态

链接库的环境变量，我们通过给之前的 SET\_UID 程序“降级”，再运行即可观察到 LD\_LIBRARY\_PATH造成这样的原因在于程序运行中Loader会set-uid程序所存储的 LD\_LIBRARY\_PATH ,转而再全局中查找要

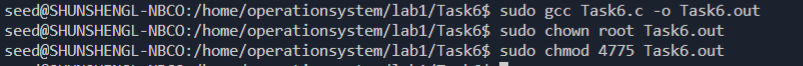
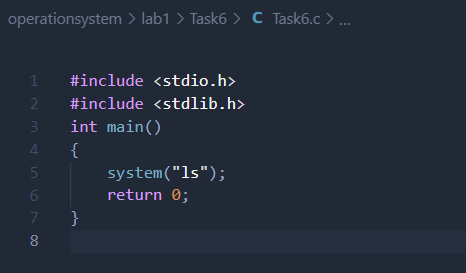
用的函数地址，以防止恶意程序修改 LD\_LIBRARY\_PATH 使程序链接并执行恶意代码。 Set-UID 程序继承

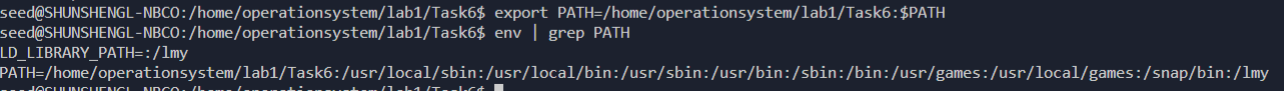
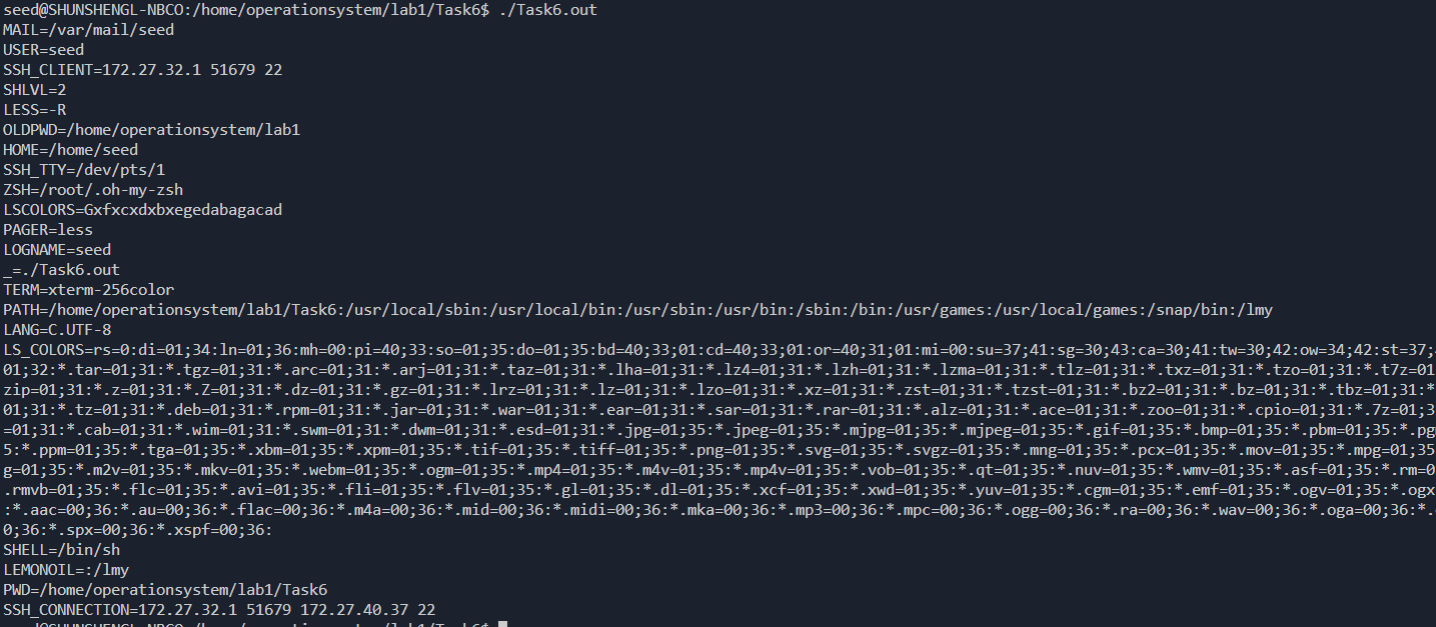
了shell的 PATH 与 CHIALE .

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

Useage

依然使用seed用户

通过sudo，编译且对程序进行提权此时运行Task6.out，程序实际上调用了 /bin/ls ，显示正常接下来我们构造 Trick 程序

首先将当前目录添加至PATH之首编写trick程序并编译，且将可执行程序名字命名为 ls现在让我们再次运行之前的Task6.out此时程序并没有像之前一样运行 /bin/ls ,而是转而运行当前目录下所构造的 ls ,该程序被定义为调用

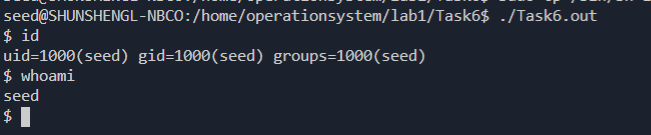
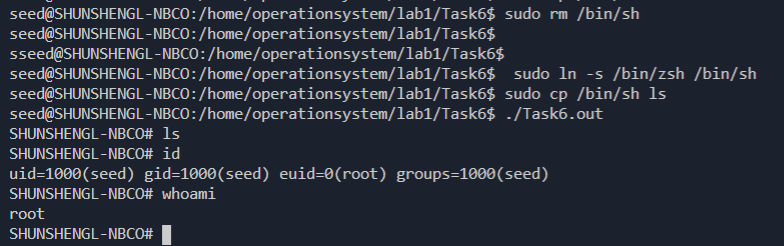
execve 执行 /usr/bin/env ,导致输出的结果不同。

这个trick分为两步，首先Task6.out中 system 函数运行的命令没有提供绝对路径，此时链接器会在环境

变量中逐个寻找含有 ls 程序的目录，找到第一个后就会链接并运行。所以第二步我们通过在环境变量

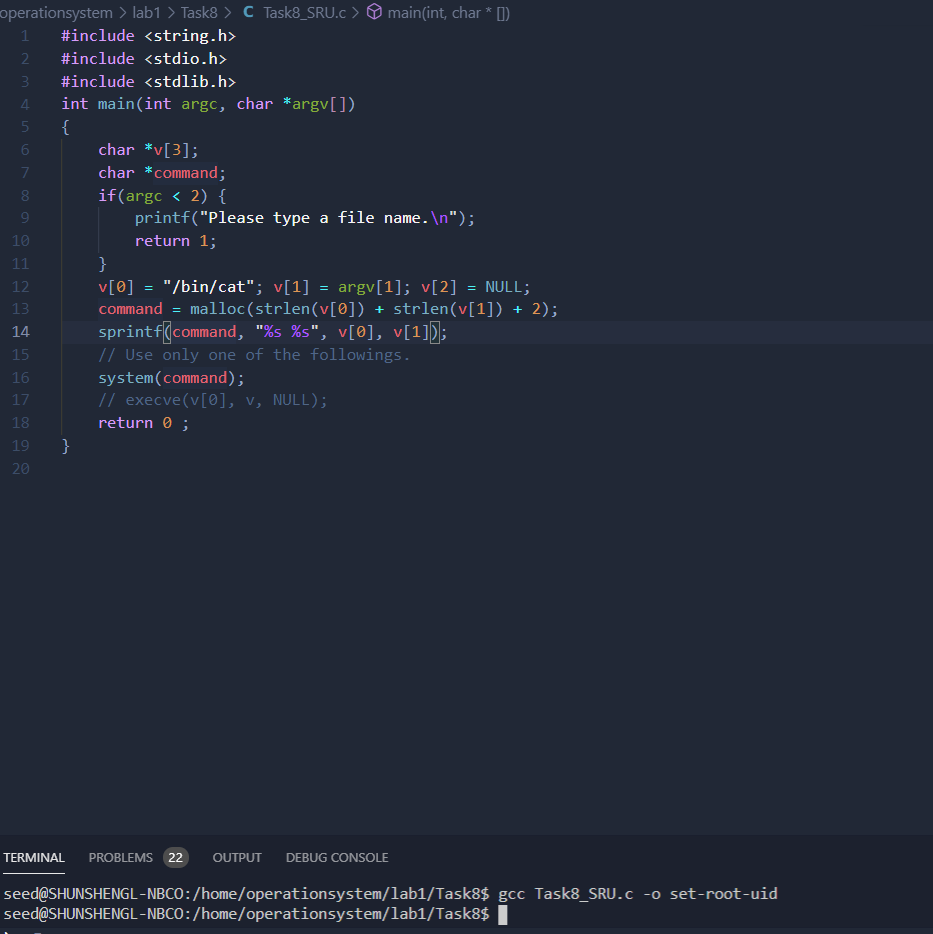
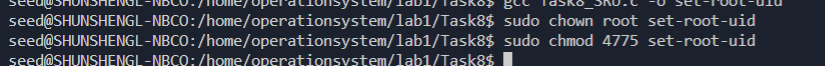
PATH之前插队，加入当前目录，使得当前目录下的 ls 程序在 PATH 中运行优先级比在其之后

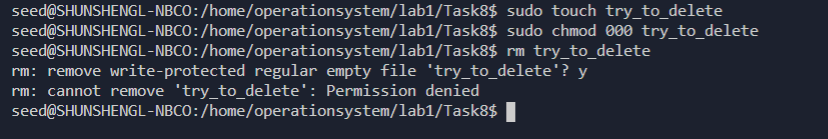
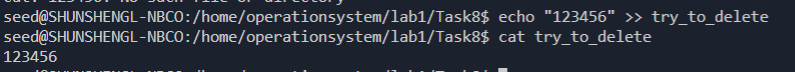
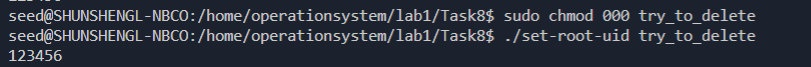
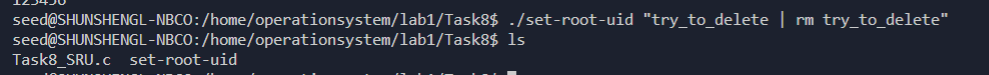
的 /bin/ls 更高，使得trick成立。

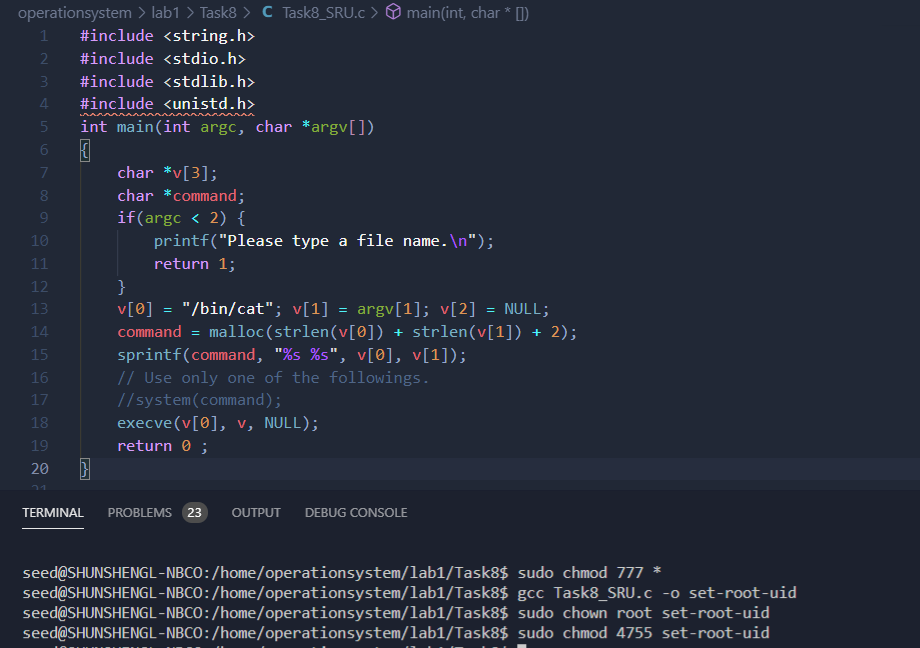
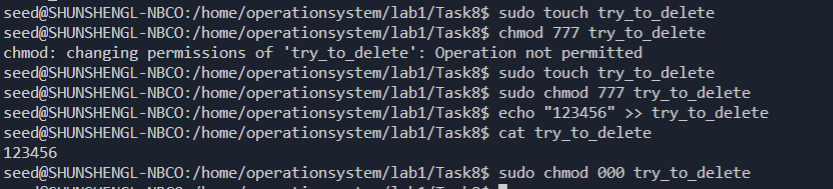
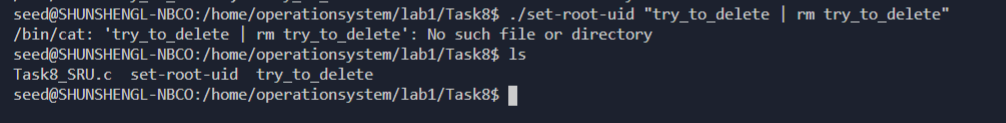
对/bin/sh的进一步尝试，当前是Ubuntu18.0.4（依然生效）拷贝 /bin/sh 到当前目录并替换掉生成的ls，运行Task6.out，成功运行 sh 程序，进入shell页面查看权限，并非root权限，然后我们更换 /bin/sh 的link，链接到zsh上此时，用户权限已由seed变更为root

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

Useage

使用seed用户，编写编译提供的 program提供权限接下来新建一个文件 try\_to\_delete ,修改其权限，此时我们发现我们没有权限可以删除这样一个我们

自己创建的文件暂时换回 777 权限，向文件中加入内容再将文件权限重置为 000 ,利用刚刚的set-uid权限读取文件内容接下来利用Task7中的知识，我们来尝试删除该文件没错，利用子shell的方式，通过管道运行了新的命令 rm try\_to\_delete ，可以观察到文件已被删除

接下来我们修改注释，重新编译授权并且生成新的 try\_to\_delete让我们利用之前的方式进行尝试很明显，这一次失败了