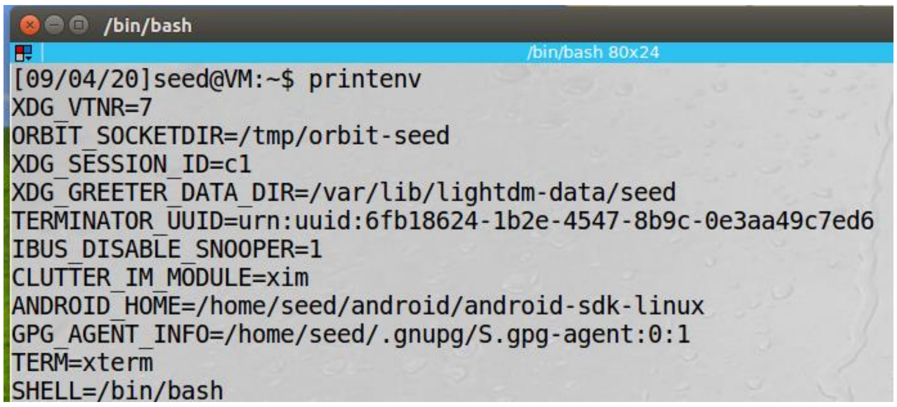
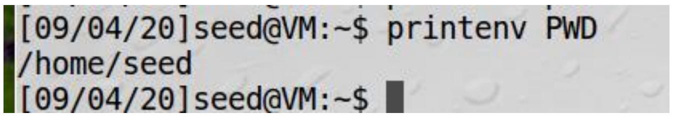
57117216 殷广成

Task1



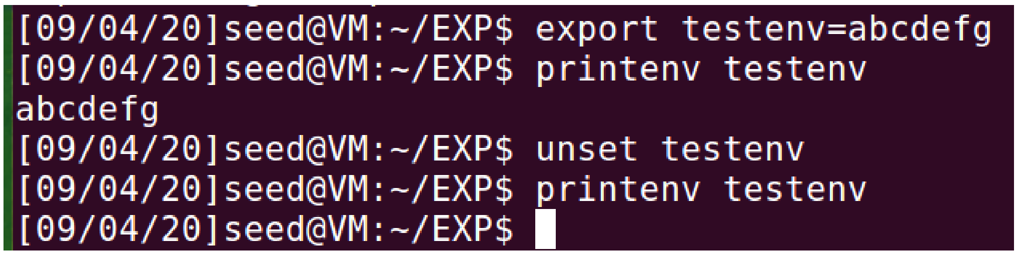
```
[09/04/20]seed@VM:~$ printenv
XDG_VTNR=7
ORBIT_SOCKETDIR=/tmp/orbit-seed
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
TERMINATOR_UUID=urn:uuid:6fb18624-1b2e-4547-8b9c-0e3aa49c7ed6
IBUS_DISABLE_SNOOPER=1
CLUTTER_IM_MODULE=xim
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm
SHELL=/bin/bash
```

利用 printenv 关键词



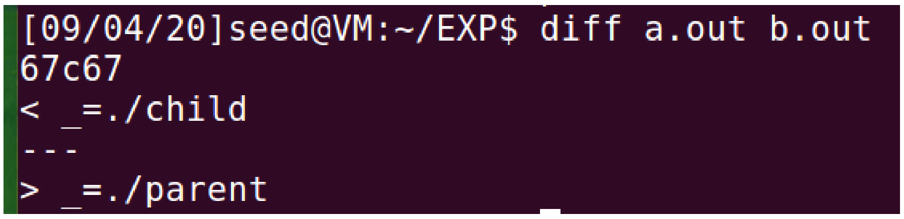
```
[09/04/20]seed@VM:~$ printenv PWD
/home/seed
[09/04/20]seed@VM:~$ 
```

新建、删除环境变量:



```
[09/04/20]seed@VM:~/EXP$ export testenv=abcdefg
[09/04/20]seed@VM:~/EXP$ printenv testenv
abcdefg
[09/04/20]seed@VM:~/EXP$ unset testenv
[09/04/20]seed@VM:~/EXP$ printenv testenv
[09/04/20]seed@VM:~/EXP$ 
```

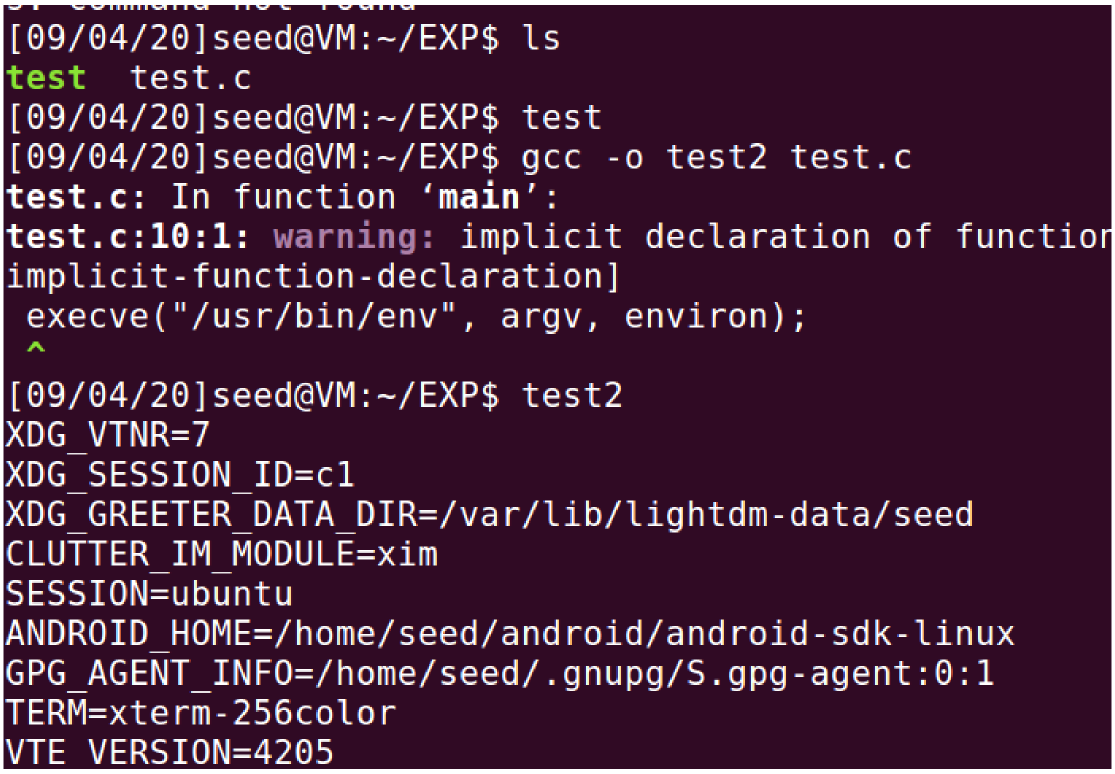
task2.



```
[09/04/20]seed@VM:~/EXP$ diff a.out b.out
67c67
< _=./child
---
> _=./parent
```

可见子进程与父进程输出基本相同，环境变量是继承的

Task3

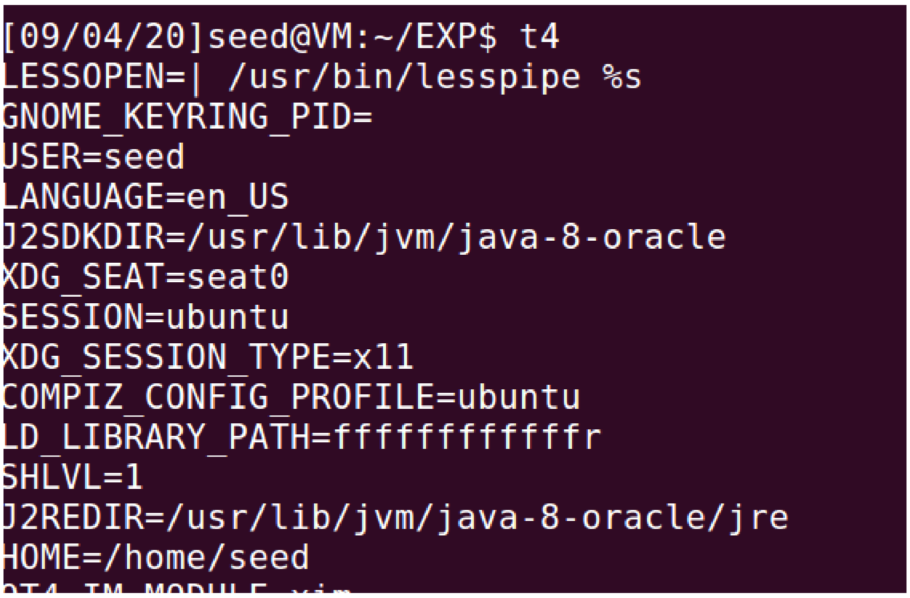


```
[09/04/20]seed@VM:~/EXP$ ls
test  test.c
[09/04/20]seed@VM:~/EXP$ test
[09/04/20]seed@VM:~/EXP$ gcc -o test2 test.c
test.c: In function ‘main’:
test.c:10:1: warning: implicit declaration of functior
implicit-function-declaration]
 execve("/usr/bin/env", argv, environ);
 ^
[09/04/20]seed@VM:~/EXP$ test2
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
```

参数 3 为 NULL 时无输出

而换成 environ 的时候有输出，这里验证了 execve 的功能以及从外部继承环境变量的功能。如果 execve 不指定环境变量，则不能继承外部环境变量。

Task4



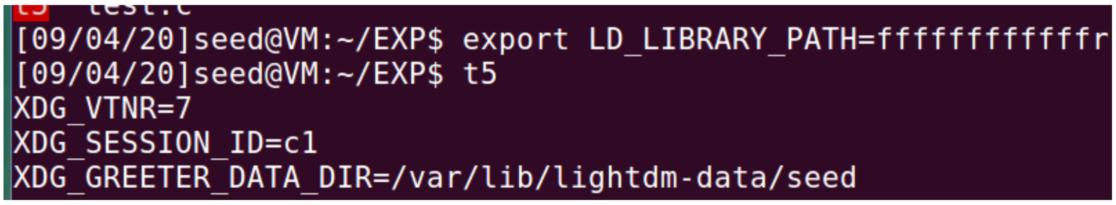
```
[09/04/20]seed@VM:~/EXP$ t4
LESSOPEN=| /usr/bin/lesspipe %s
GNOME_KEYRING_PID=
USER=seed
LANGUAGE=en_US
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_SEAT=seat0
SESSION=ubuntu
XDG_SESSION_TYPE=x11
COMPIZ_CONFIG_PROFILE=ubuntu
LD_LIBRARY_PATH=fffffffffffr
SHLVL=1
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
HOME=/home/seed
QT4_IM_MODULE=xim
```

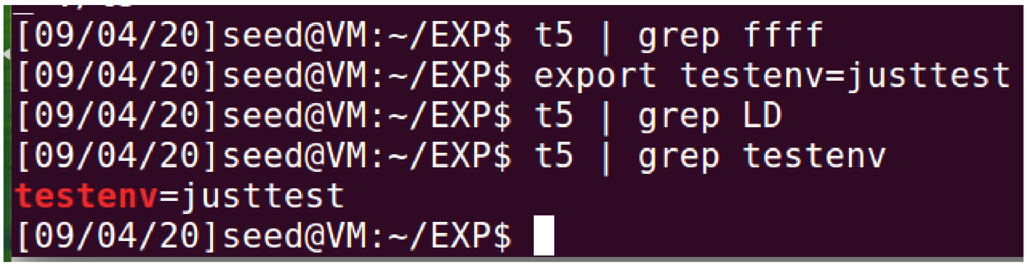
System()会创建一个子进程，继承原来的环境变量，

函数原型：

```
execl("/bin/sh", "sh", "-c", cmdstring, (char *)0);
```

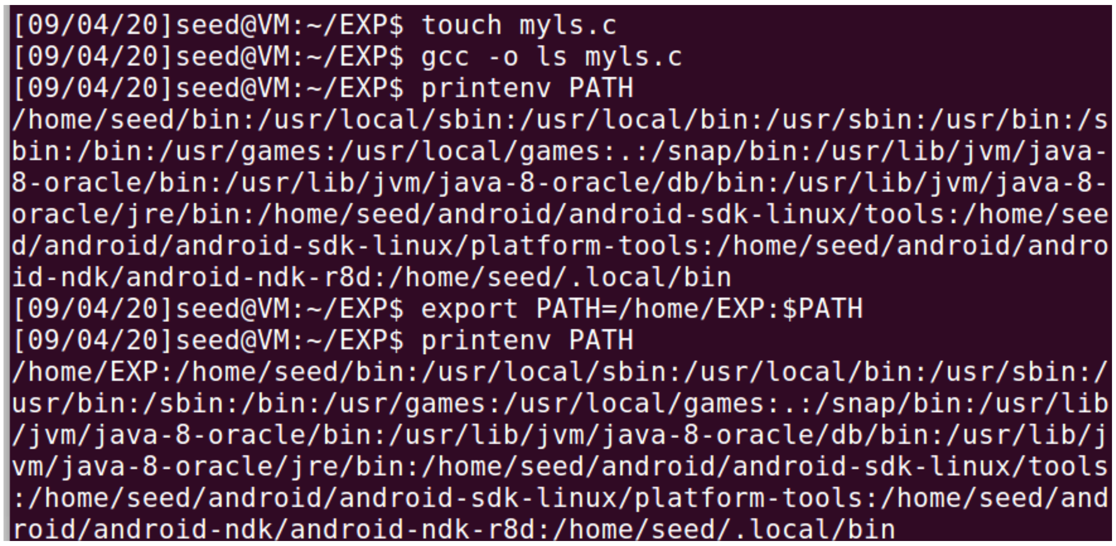
Task5

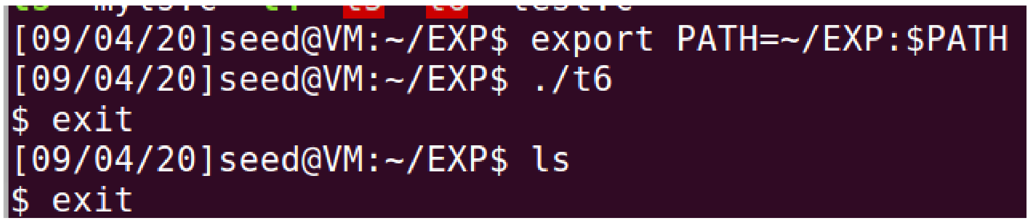

```
[09/04/20]seed@VM:~/EXP$ export LD_LIBRARY_PATH=fffffffffffr
[09/04/20]seed@VM:~/EXP$ t5
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
```


```
[09/04/20]seed@VM:~/EXP$ t5 | grep ffff
[09/04/20]seed@VM:~/EXP$ export testenv=justtest
[09/04/20]seed@VM:~/EXP$ t5 | grep LD
[09/04/20]seed@VM:~/EXP$ t5 | grep testenv
testenv=justtest
[09/04/20]seed@VM:~/EXP$
```

环境变量中有 testenv 没有 LD_LIBRARY_PATH
说明系统监测到该进程为 setuid 进程，因此屏蔽了该环境变量

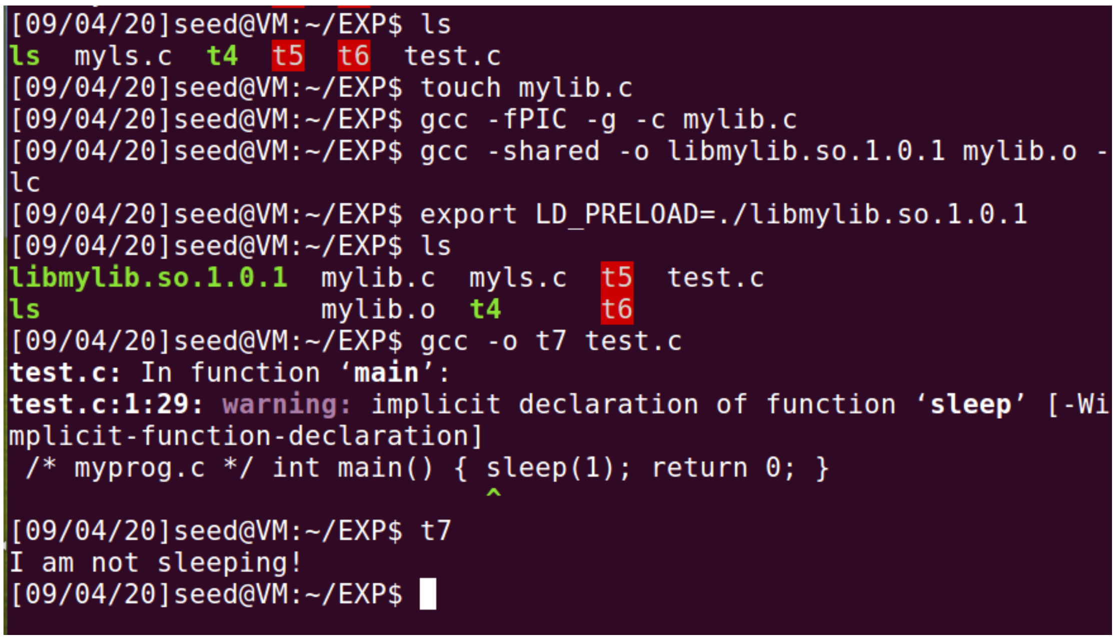
Task6


```
[09/04/20]seed@VM:~/EXP$ touch myls.c
[09/04/20]seed@VM:~/EXP$ gcc -o ls myls.c
[09/04/20]seed@VM:~/EXP$ printenv PATH
/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/s
bin:/bin:/usr/games:/usr/local/games:.:/snap/bin:/usr/lib/jvm/java-
8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-
oracle/jre/bin:/home/seed/android/android-sdk-linux/tools:/home/see
d/android/android-sdk-linux/platform-tools:/home/seed/android/andro
id-ndk/android-ndk-r8d:/home/seed/.local/bin
[09/04/20]seed@VM:~/EXP$ export PATH=/home/EXP:$PATH
[09/04/20]seed@VM:~/EXP$ printenv PATH
/home/EXP:/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/
usr/bin:/sbin:/bin:/usr/games:/usr/local/games:.:/snap/bin:/usr/lib
/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/j
vm/java-8-oracle/jre/bin:/home/seed/android/android-sdk-linux/tools
:/home/seed/android/android-sdk-linux/platform-tools:/home/seed/and
roid/android-ndk/android-ndk-r8d:/home/seed/.local/bin
```
写一个自己的 ls，但实际上通过 system 开了一个 shell
再设置环境变量的路径


```
[09/04/20]seed@VM:~/EXP$ export PATH=~/EXP:$PATH
[09/04/20]seed@VM:~/EXP$ ./t6
$ exit
[09/04/20]seed@VM:~/EXP$ ls
$ exit
```

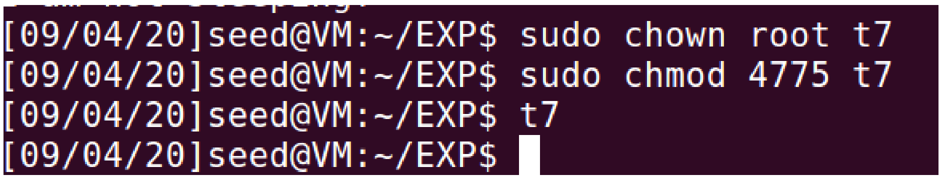
这里重新设置一下 path，成功进入 shell，ls 命令也进入 shell
原理十分简单，这里不再说明

Task7



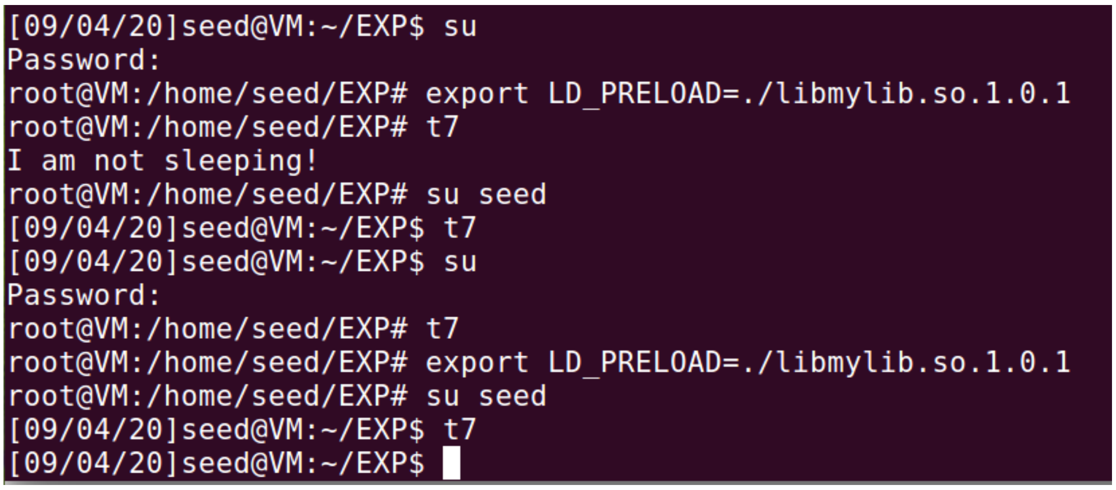
```
[09/04/20]seed@VM:~/EXP$ ls
ls  myls.c  t4  t5  t6  test.c
[09/04/20]seed@VM:~/EXP$ touch mylib.c
[09/04/20]seed@VM:~/EXP$ gcc -fPIC -g -c mylib.c
[09/04/20]seed@VM:~/EXP$ gcc -shared -o libmylib.so.1.0.1 mylib.o -
lc
[09/04/20]seed@VM:~/EXP$ export LD_PRELOAD=./libmylib.so.1.0.1
[09/04/20]seed@VM:~/EXP$ ls
libmylib.so.1.0.1  mylib.c  myls.c  t5  test.c
ls                 mylib.o  t4      t6
[09/04/20]seed@VM:~/EXP$ gcc -o t7 test.c
test.c: In function ‘main’:
test.c:1:29: warning: implicit declaration of function ‘sleep’ [-Wi
mplicit-function-declaration]
 /* myprog.c */ int main() { sleep(1); return 0; }
                             ^
[09/04/20]seed@VM:~/EXP$ t7
I am not sleeping!
[09/04/20]seed@VM:~/EXP$ 
```

普通用户运行程序，成功连接 mylib



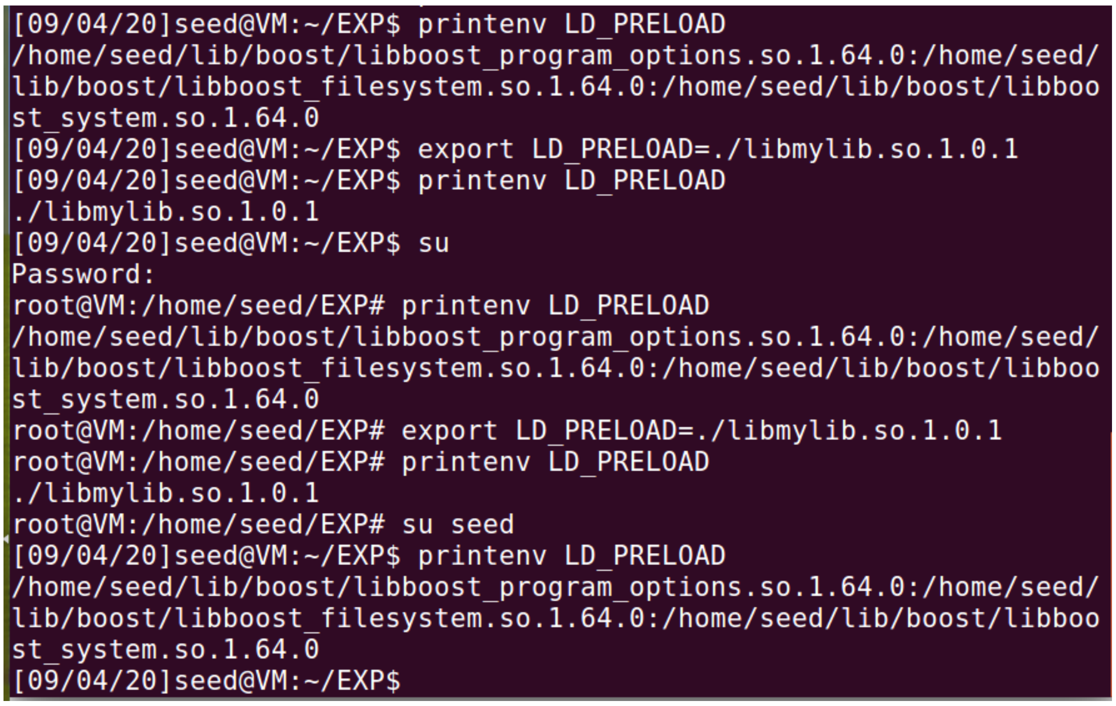
```
[09/04/20]seed@VM:~/EXP$ sudo chown root t7
[09/04/20]seed@VM:~/EXP$ sudo chmod 4775 t7
[09/04/20]seed@VM:~/EXP$ t7
[09/04/20]seed@VM:~/EXP$ 
```

切换至 setuid 程序, 这时运行后发现无法通过 LD*链接到 mylib, 说明动态链接器会屏蔽 LD*环境变量，同上也是一种防御机制。



```
[09/04/20]seed@VM:~/EXP$ su
Password:
root@VM:/home/seed/EXP# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/EXP# t7
I am not sleeping!
root@VM:/home/seed/EXP# su seed
[09/04/20]seed@VM:~/EXP$ t7
[09/04/20]seed@VM:~/EXP$ su
Password:
root@VM:/home/seed/EXP# t7
root@VM:/home/seed/EXP# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/EXP# su seed
[09/04/20]seed@VM:~/EXP$ t7
[09/04/20]seed@VM:~/EXP$ 
```

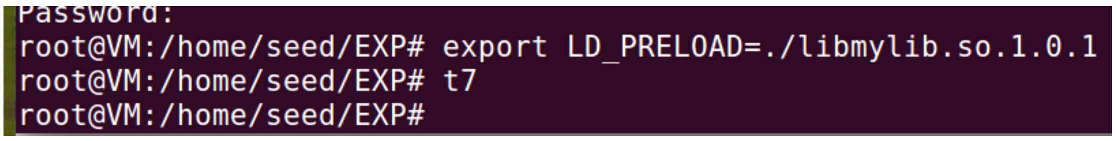
可以看到，利用 root 用户设置 LD_PRELOAD 之后在 root 用户下运行 t7 可以重新链接，而中间如果进行切换用户，则无法链接，说明切换用户后会重置 LDenv。下面证明:



```
[09/04/20]seed@VM:~/EXP$ printenv LD_PRELOAD
/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed/
lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboo
st_system.so.1.64.0
[09/04/20]seed@VM:~/EXP$ export LD_PRELOAD=./libmylib.so.1.0.1
[09/04/20]seed@VM:~/EXP$ printenv LD_PRELOAD
./libmylib.so.1.0.1
[09/04/20]seed@VM:~/EXP$ su
Password:
root@VM:/home/seed/EXP# printenv LD_PRELOAD
/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed/
lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboo
st_system.so.1.64.0
root@VM:/home/seed/EXP# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/EXP# printenv LD_PRELOAD
./libmylib.so.1.0.1
root@VM:/home/seed/EXP# su seed
[09/04/20]seed@VM:~/EXP$ printenv LD_PRELOAD
/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed/
lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboo
st_system.so.1.64.0
[09/04/20]seed@VM:~/EXP$
```

经过多次对照实验发现推论正确

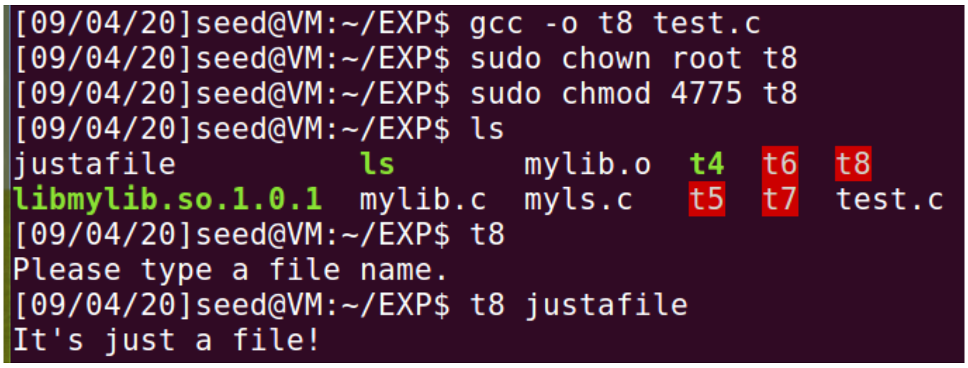
将 t7 修改为 seed 的 setuid 程序，再在 root 中重设 LD_PRELOAD 并运行:



```
Password:
root@VM:/home/seed/EXP# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/EXP# t7
root@VM:/home/seed/EXP#
```

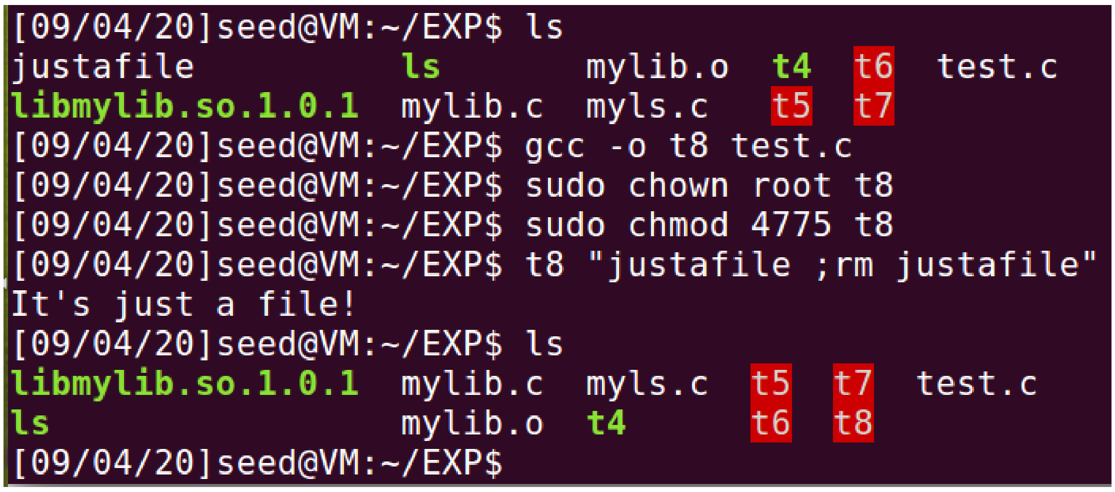
无效，说明 LD*成功链接的前提是同一个用户的程序+env

Task8



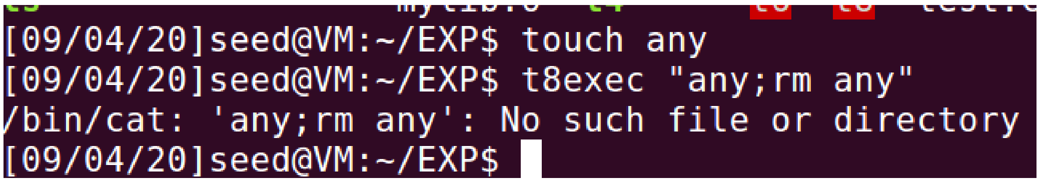
```
[09/04/20]seed@VM:~/EXP$ gcc -o t8 test.c
[09/04/20]seed@VM:~/EXP$ sudo chown root t8
[09/04/20]seed@VM:~/EXP$ sudo chmod 4775 t8
[09/04/20]seed@VM:~/EXP$ ls
justafile          ls       mylib.o  t4  t6  t8
libmylib.so.1.0.1  mylib.c  myls.c   t5  t7  test.c
[09/04/20]seed@VM:~/EXP$ t8
Please type a file name.
[09/04/20]seed@VM:~/EXP$ t8 justafile
It's just a file!
```

正常运行



```
[09/04/20]seed@VM:~/EXP$ ls
justafile          ls        mylib.o  t4   t6   test.c
libmylib.so.1.0.1  mylib.c   myls.c   t5   t7
[09/04/20]seed@VM:~/EXP$ gcc -o t8 test.c
[09/04/20]seed@VM:~/EXP$ sudo chown root t8
[09/04/20]seed@VM:~/EXP$ sudo chmod 4775 t8
[09/04/20]seed@VM:~/EXP$ t8 "justafile ;rm justafile"
It's just a file!
[09/04/20]seed@VM:~/EXP$ ls
libmylib.so.1.0.1  mylib.c  myls.c  t5  t7  test.c
ls                 mylib.o  t4      t6  t8
[09/04/20]seed@VM:~/EXP$
```

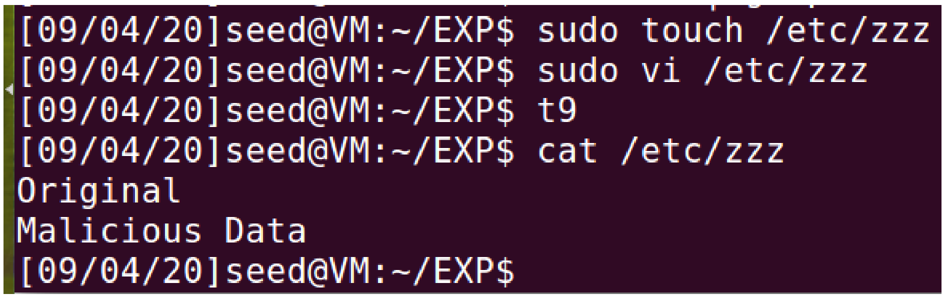
因为 system 可以将字符串作为指令执行，所以可以通过这个特性使其执行多个指令。



```
[09/04/20]seed@VM:~/EXP$ touch any
[09/04/20]seed@VM:~/EXP$ t8exec "any;rm any"
/bin/cat: 'any;rm any': No such file or directory
[09/04/20]seed@VM:~/EXP$
```

execve()指定第一个参数为命令，第二个参数为命令的参数，这样就保证了只有指定的命令会运行。

Task9



```
[09/04/20]seed@VM:~/EXP$ sudo touch /etc/zzz
[09/04/20]seed@VM:~/EXP$ sudo vi /etc/zzz
[09/04/20]seed@VM:~/EXP$ t9
[09/04/20]seed@VM:~/EXP$ cat /etc/zzz
Original
Malicious Data
[09/04/20]seed@VM:~/EXP$
```

文件句柄没有处理，导致残留，被子进程利用写入数据