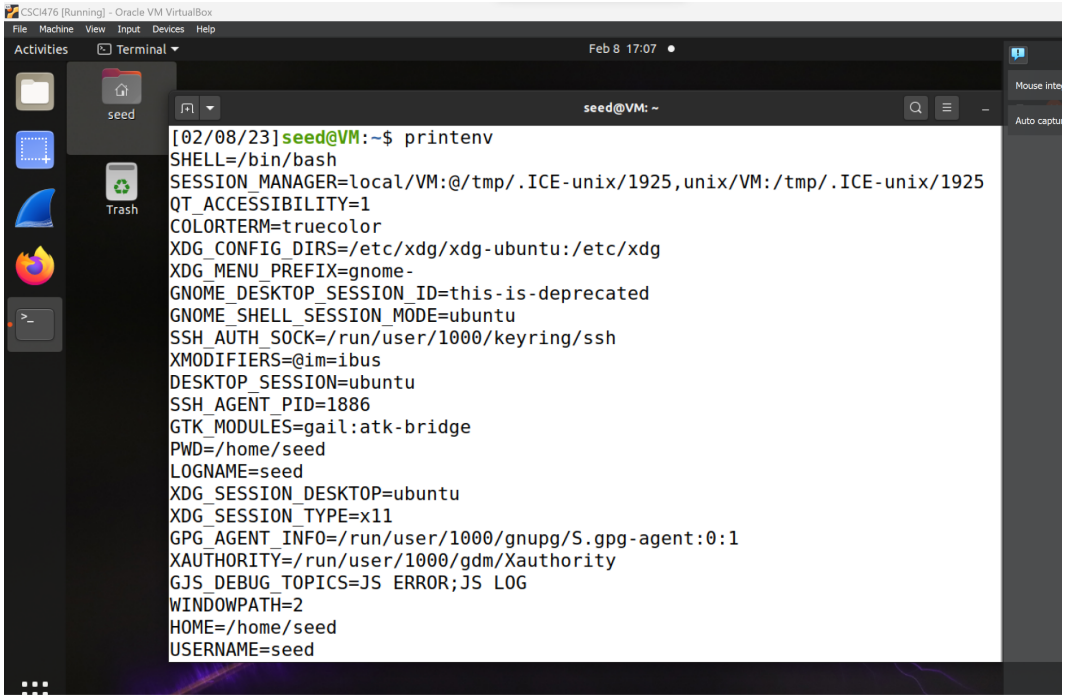
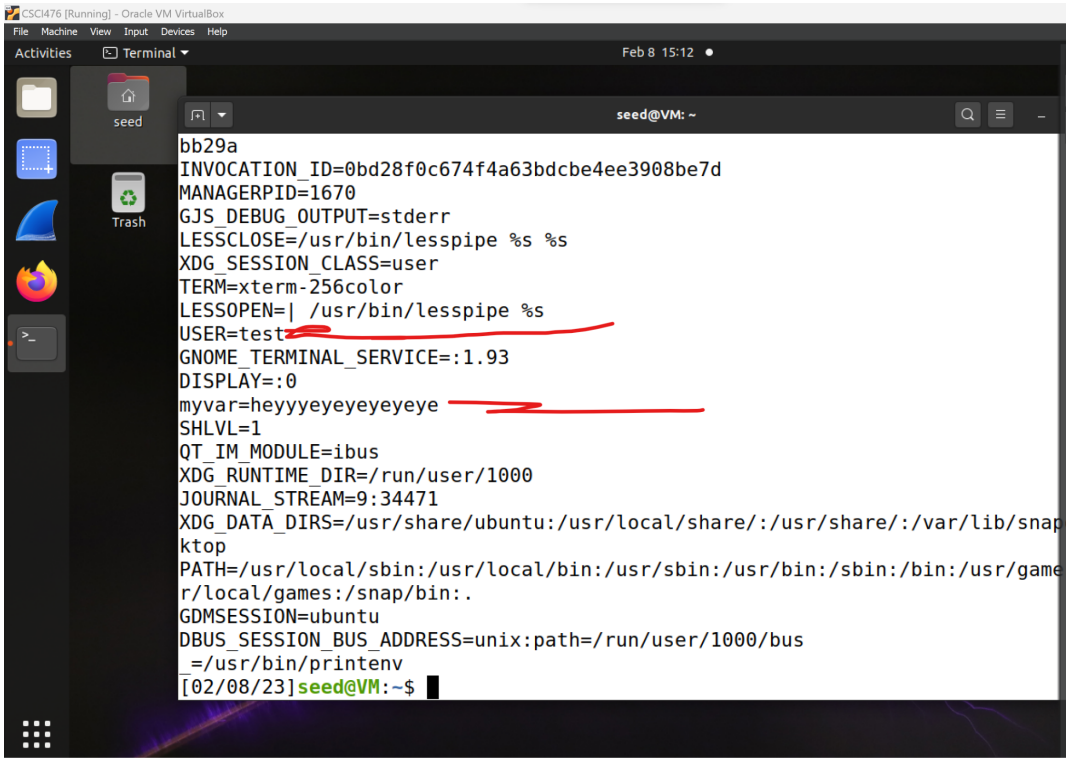
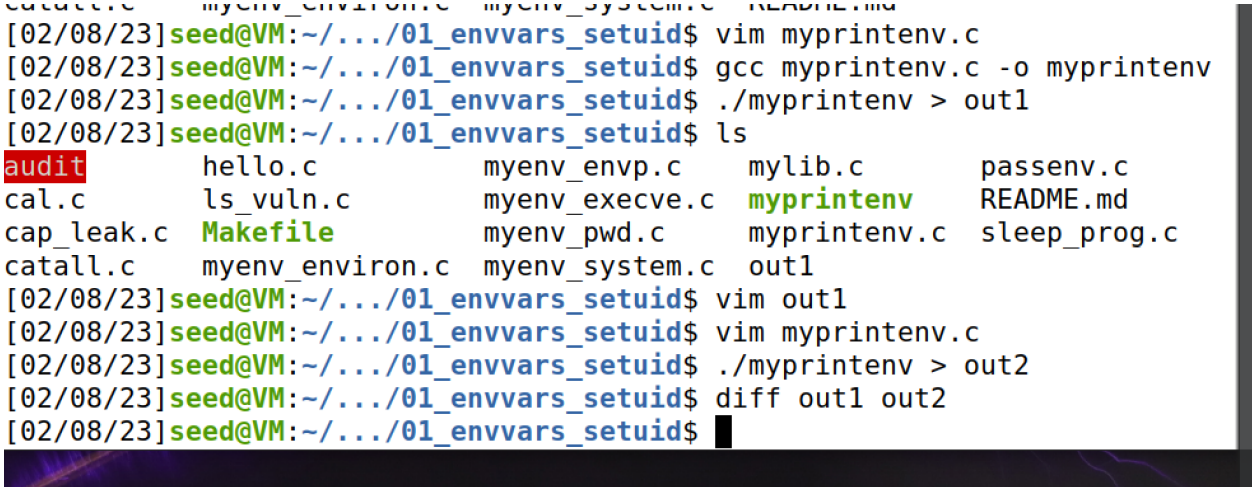
Task1: use printenv



Task 1.2: environment variables

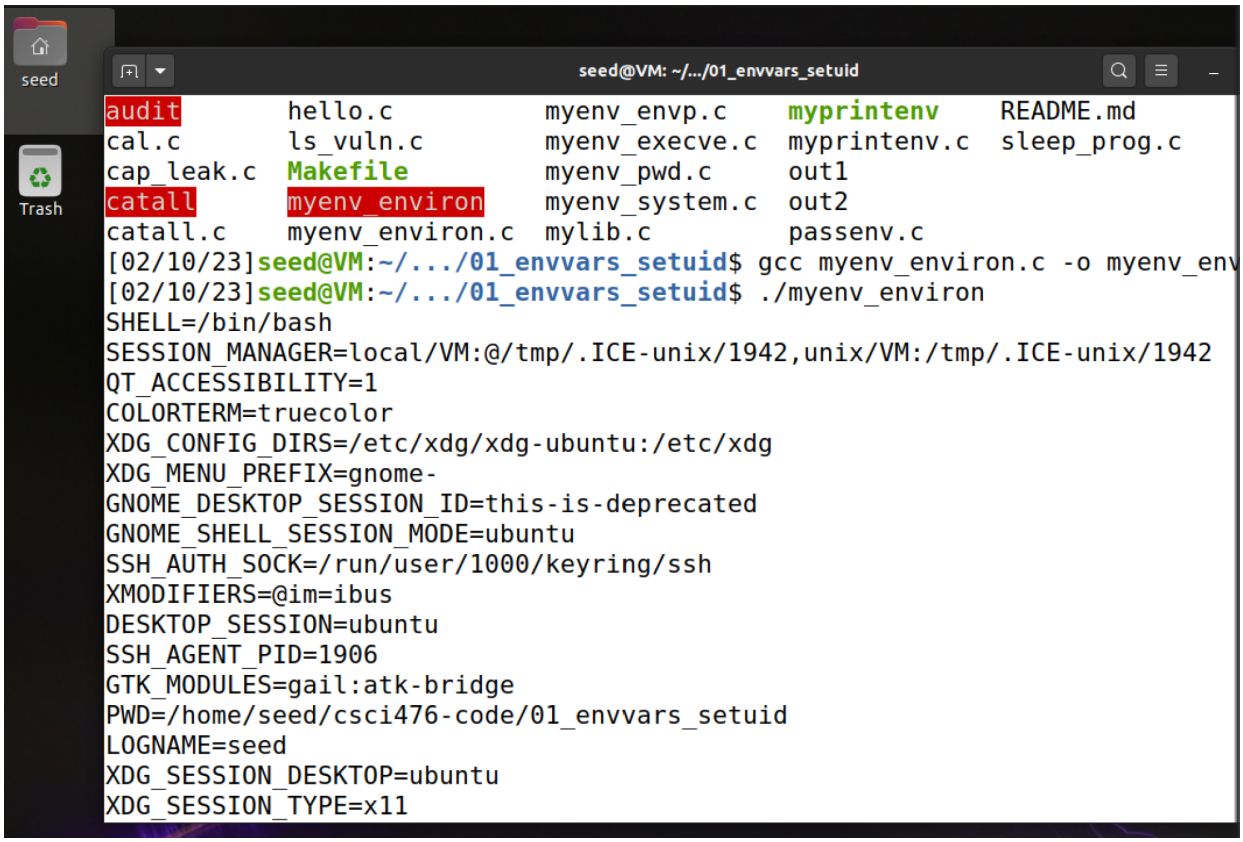


Tasks 2.1- 2.3:



There is no difference between the environment variables between the parent and the child forks, therefore, the child and the parent are inheriting the same environment variables.

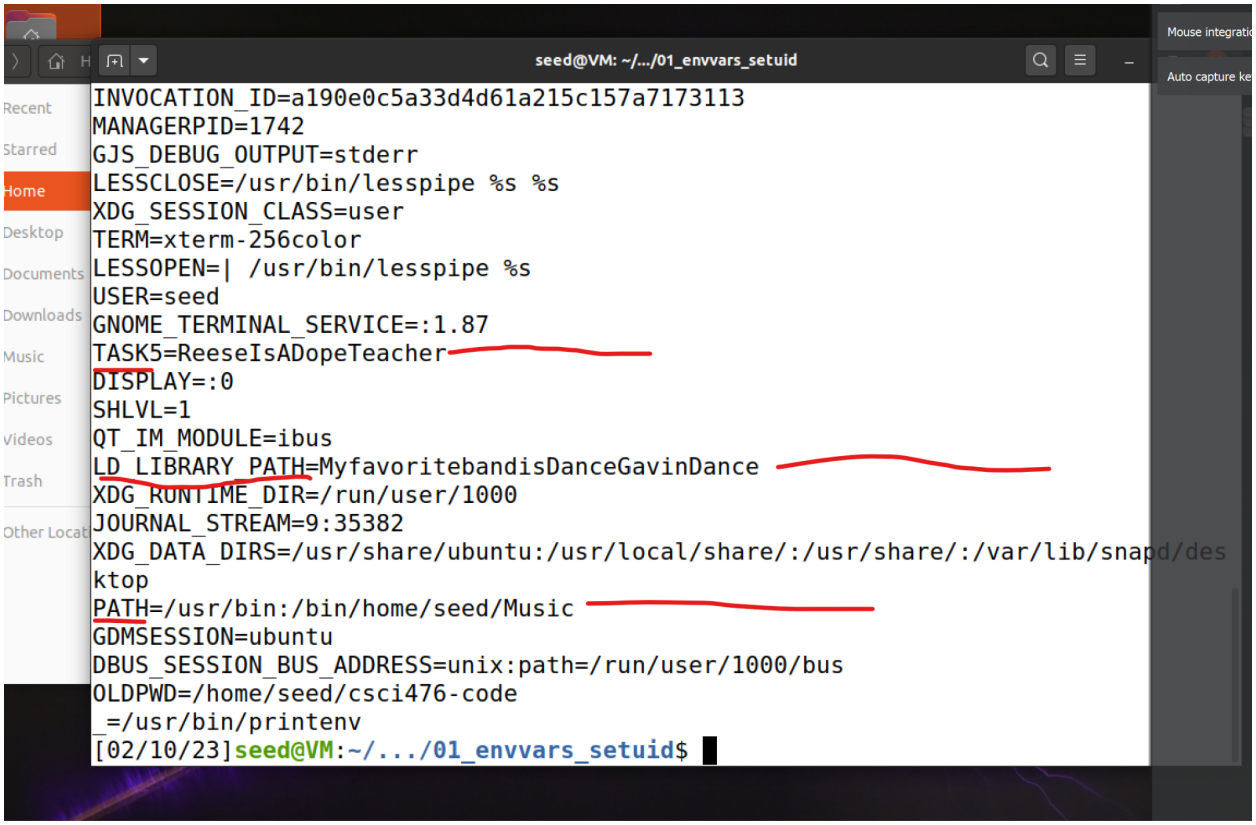
Tasks 3.1: Verify that your implementation correctly prints the environment variables.



3.2: Compile the above program, change its ownership to root, and make it a Set-UID program.

Task 3.3: In your shell (you need to be in a normal user account, not the root account), use the export command to set the following environment variables

Here is proof that I was able to set the specified environment variables to what I'd like. (This is not the output from running the program.)

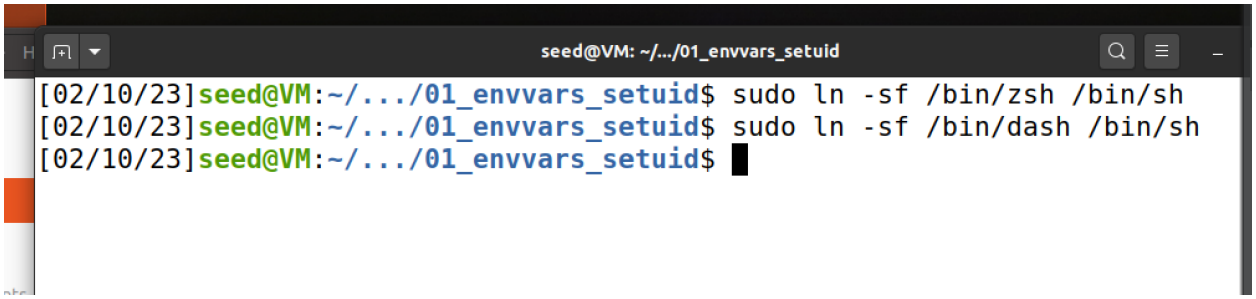


After running the program:

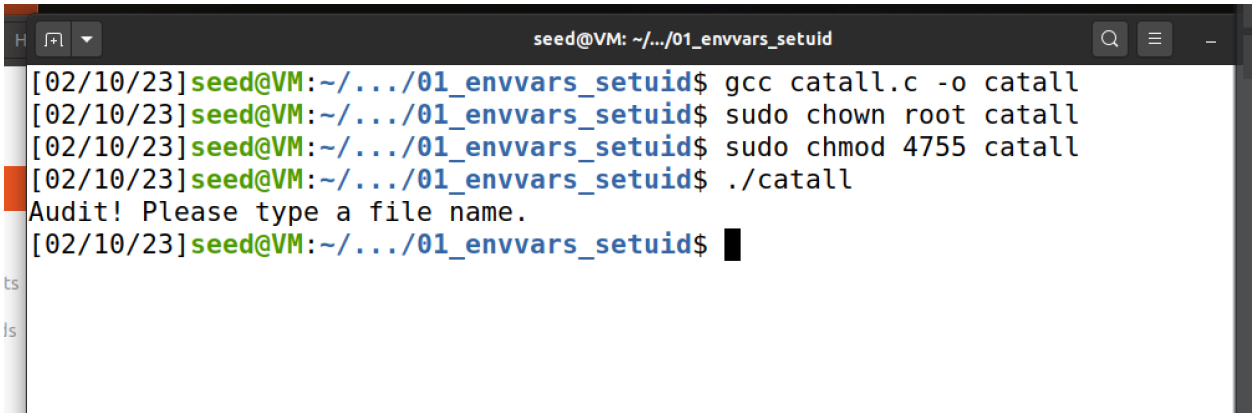
```
TASK5=ReeseIsADopeTeacher
DISPLAY=:0
SHLVL=1
QT_IM_MODULE=ibus
LD_LIBRARY_PATH=MyfavoritebandisDanceGavinDance
XDG_RUNTIME_DIR=/run/user/1000
JOURNAL_STREAM=9:35382
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share/:/var/lib/snapd/d
ktop
PATH=/usr/bin:/bin/home/seed/Music
GDMSESSION=ubuntu
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
OLDPWD=/home/seed/csci476-code
_=./myenv_environ
[02/10/23]seed@VM:~/.../01 envvars setuid$
```

The variables **PATH, LD_LIBARARY_PATH,** and **TASK5** are present in the child fork process; I don't think there is anything too surprising about this.

4.0 : disabling counter measures



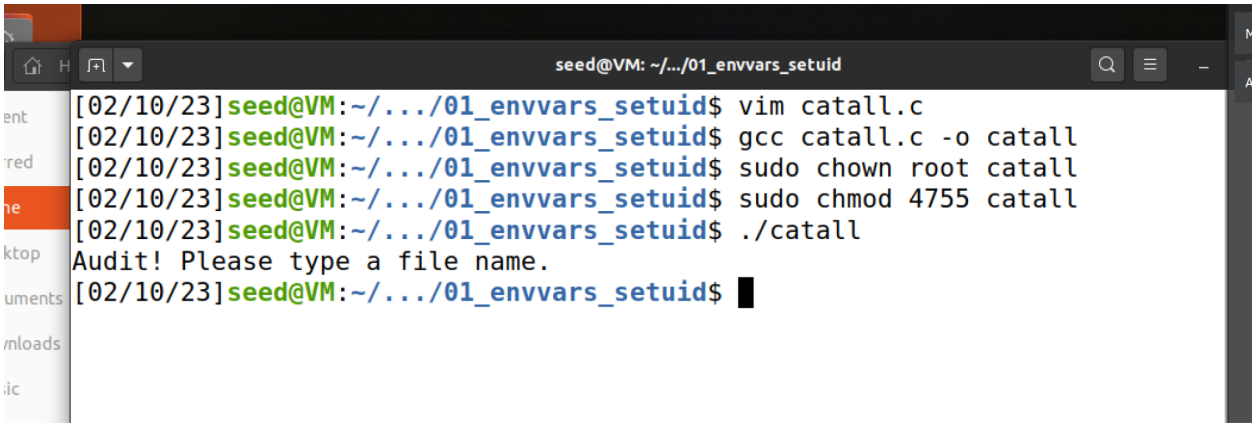
Task 4.1 :



Yes, since the program uses **System(command)**, Bob can abuse this by running malicious lines of code as appended arguments when he tries to read a file. Since the program above has higher privileges as a SET-UID program during execution, and

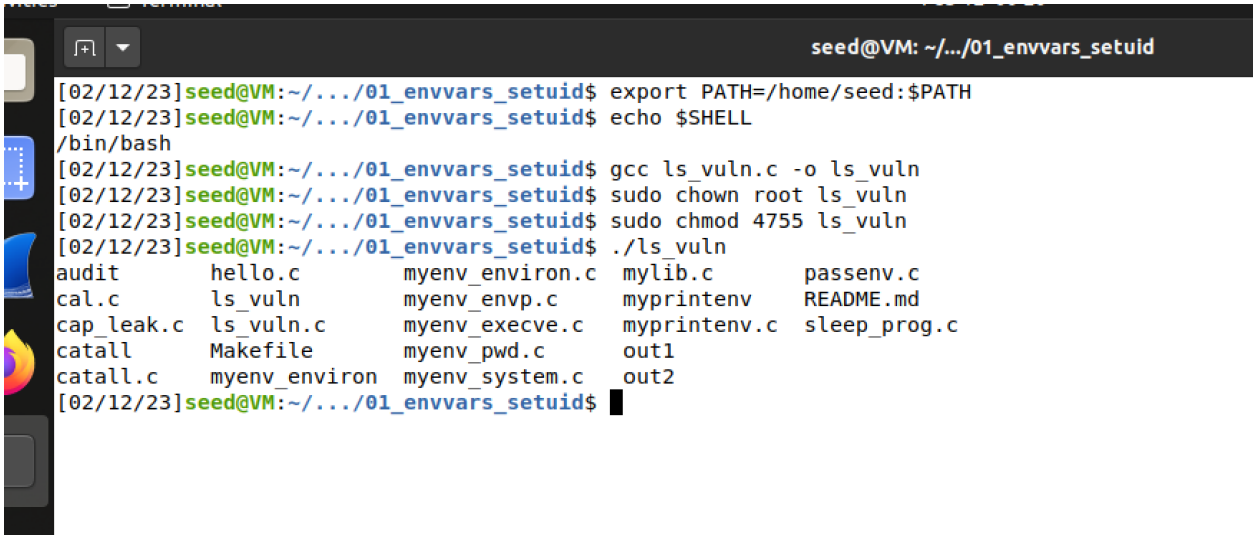
System() will run with these higher privileges, Bob can trick System() to run these malicious lines of code with its higher privileges.

Task 4.2: Comment out the system(command) statement, and uncomment the execve() statement; the program will use execve() to invoke the command. Compile the program, and make it a root-owned Set-UID.



Running the execve() command will not let Bob's attack from task 4.1 pass. This is because execve() replaces the contents of the process and a whole new executable is loaded.

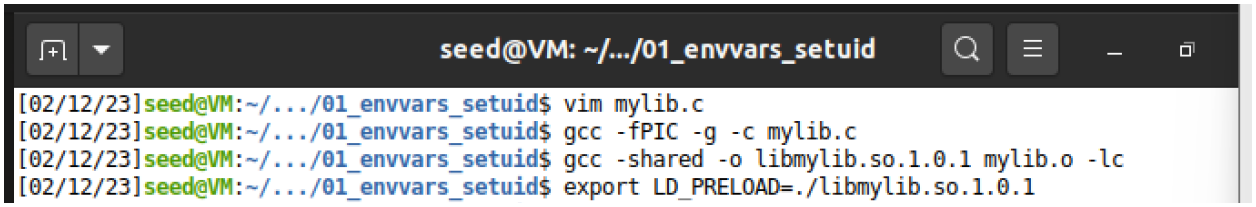
Task 5: PATH and Set-UID Programs

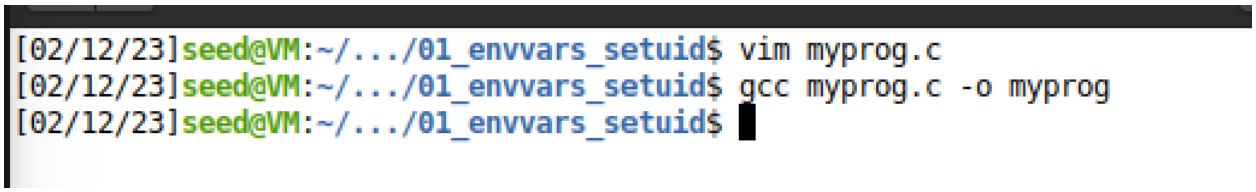


Yes, we can make this Set-UID program run our code by changing the PATH environment variable to locate a directory we choose, and then naming our own code the same name as ls_vuln. The computer will think our *potentially malicious* code is the safe code it's expecting.

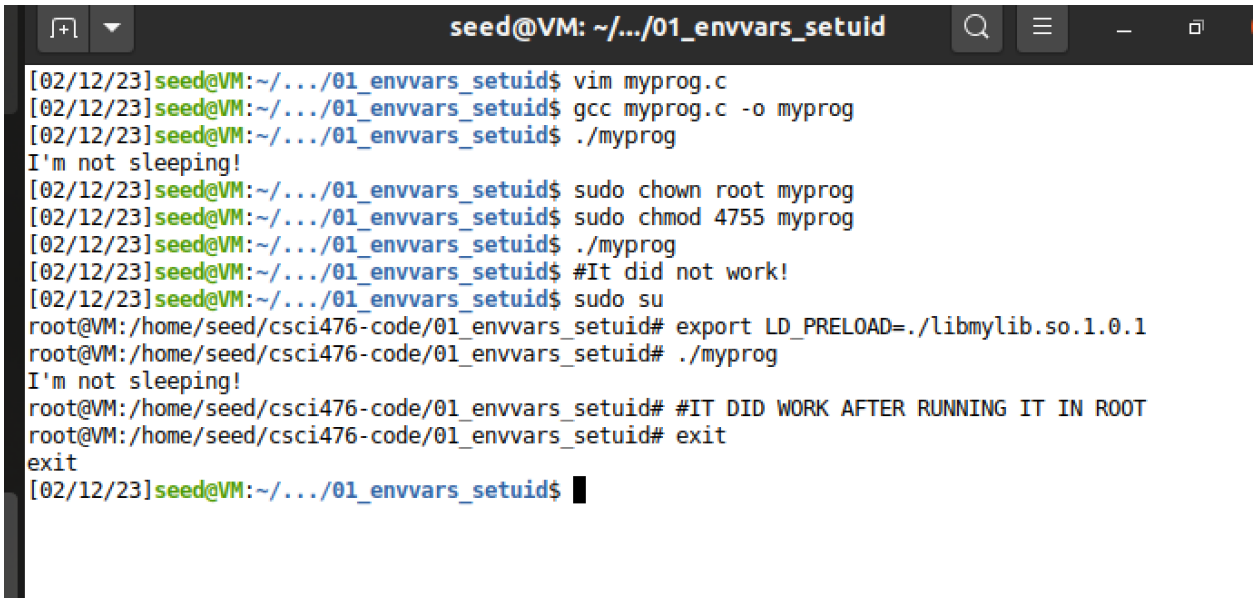
This will also run our code with root privileges as this Set-UID program runs System(), which will give our program temporarily-high privileges.

Task 6.1 : LS_PRELOAD and Set-UID Programs





Tasks 6.2: LS_PRELOAD cont'



Task 6.3:

When we ran the **myprog** program, it worked initially. However, when we chained the owner to root and made it a Set-UID program, it no longer worked! Finally, when we entered a higher-privileged shell and *then* ran it again, our **myprog** program worked again.

We can refer back to this slide with similar-yielding results:

Process Owner	Set-UID program?	Success?
seed	❌	✅
seed	✅	❌
root	✅	✅

We can say our failure in the seed-process, Set-UID program was due to the child no longer inheriting the LD_PRELOAD environment variable that we previously edited.