

Homework 1

Environment Variable and Set-UID Program Lab

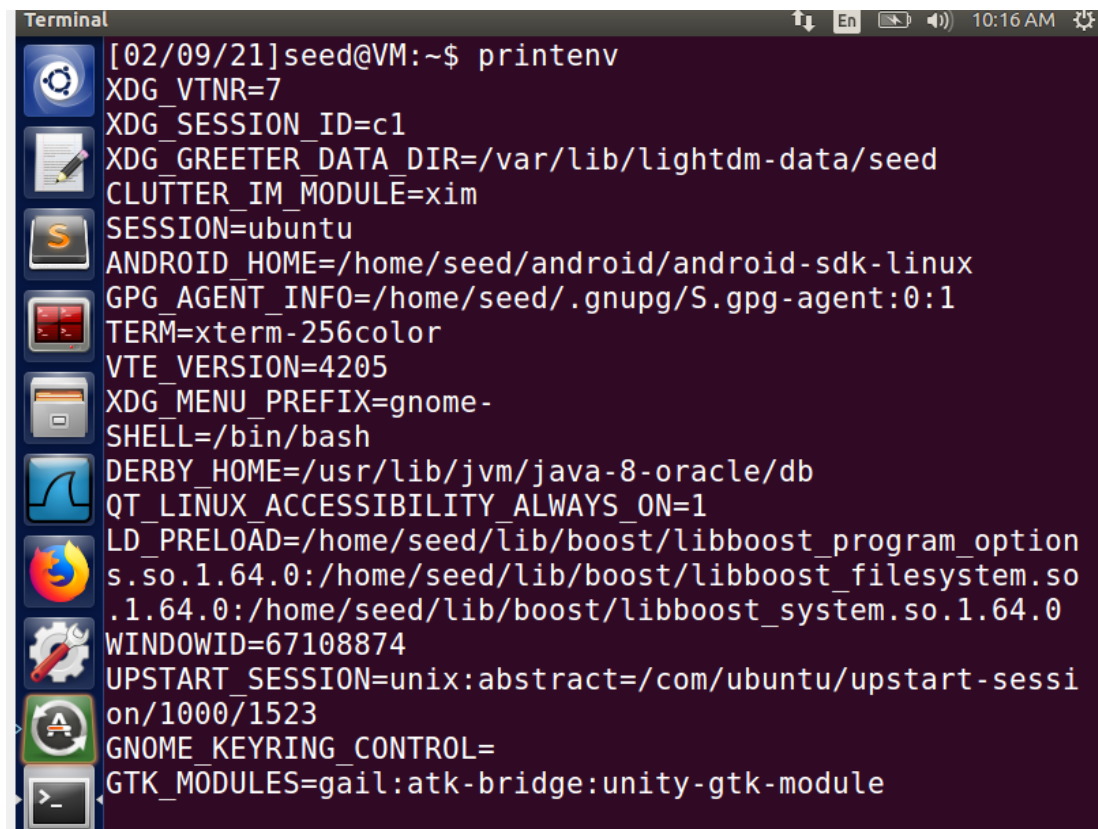
Name:- Aparna Krishna Bhat

UTA ID:- 1001255079

Task 1: Manipulating Environment Variables

The aim of this task is to study the commands that can be used to set and unset environment variables.

In any shell there are a good number of environment variables, set either by the system, or by own shell scripts and configuration. All the environment variables can be printed on the terminal using the *printenv* or *env* command.

A terminal window titled "Terminal" with a dark background and light text. The prompt is "[02/09/21]seed@VM:~\$". The command "printenv" has been executed, and the output lists various environment variables. The variables shown include XDG_VTNR, XDG_SESSION_ID, XDG_GREETER_DATA_DIR, CLUTTER_IM_MODULE, SESSION, ANDROID_HOME, GPG_AGENT_INFO, TERM, VTE_VERSION, XDG_MENU_PREFIX, SHELL, DERBY_HOME, QT_LINUX_ACCESSIBILITY_ALWAYS_ON, LD_PRELOAD, WINDOWID, UPSTART_SESSION, GNOME_KEYRING_CONTROL, and GTK_MODULES. The terminal window has a sidebar on the left with icons for various applications like a file manager, terminal, and web browser. The top status bar shows the time as 10:16 AM and some system icons.

```
[02/09/21]seed@VM:~$ 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_option
s.so.1.64.0:/home/seed/lib/boost/libboost_filesystem.so
.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.0
WINDOWID=67108874
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-sessi
on/1000/1523
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
```

Fig 1:- Output for printenv command with a few more lines(not shown)

We can append a variable name as a parameter, to only show that variable value. Fig 2 shows the output for *env | grep PWD* and few more

```

[02/09/21]seed@VM:~$ printenv PWD
/home/seed
[02/09/21]seed@VM:~$ env | grep PWD
PWD=/home/seed
[02/09/21]seed@VM:~$ env | grep LOGNAME
LOGNAME=seed
[02/09/21]seed@VM:~$ env | grep JOB
JOB=unity-settings-daemon
UPSTART_JOB=unity7
[02/09/21]seed@VM:~$

```

Fig 2

Using **export**, the environment variable will be set for the current shell session. Therefore, if we try to open it another shell or if we restart the VM, the environment variable will not be accessible anymore. As seen in the output environment variable FOO has been set using export with a value 'test string value'.

```

[02/09/21]seed@VM:~$ export FOO = "test string value"
bash: export: `=': not a valid identifier
bash: export: `test string value': not a valid identifier
[02/09/21]seed@VM:~$ export FOO = 'test string value'
bash: export: `=': not a valid identifier
bash: export: `test string value': not a valid identifier
[02/09/21]seed@VM:~$ export FOO='test string value'
[02/09/21]seed@VM:~$ print FOO
Unescaped left brace in regex is deprecated, passed through
in regex; marked by <-- HERE in m/%{ <-- HERE (.*)}/ at /usr/
bin/print line 528.
Error: no such file "FOO"
[02/09/21]seed@VM:~$ printenv FOO
test string value
[02/09/21]seed@VM:~$

```

Fig 3:- Output for command **export**

unset command is used to unset any local environment variable temporarily. Once we unset FOO and then try to find it using env command, it returns nothing because there is no variable FOO.

```

SHELL=/bin/bash
FOO=test string value
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:/h
ome/seed/lib/boost/libboost_system.so.1.64.0
WINDOWID=67108874

```

Fig 4

```
[02/09/21]seed@VM:~$ unset F00
[02/09/21]seed@VM:~$ printenv F00
[02/09/21]seed@VM:~$
```

Fig 5

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

The aim of this task is to study how a child process gets its environment variables from its parent.

The content of the output of task2.c containing child process with printenv is stored in file named task2.out. It displays all the environment variables of the child process. The parent process with printenv command gives the similar output.

```
[02/09/21]seed@VM:~$ vi task2.c
[02/09/21]seed@VM:~$ g++ -o task2.out task2.c -std=c++11
[02/09/21]seed@VM:~$ ./task2.out > child
[02/09/21]seed@VM:~$ vi task2.c
[02/09/21]seed@VM:~$ g++ -o task2_1.out task2.c -std=c++11
[02/09/21]seed@VM:~$ ./task2_1.out > child1
[02/09/21]seed@VM:~$ diff child child1
72c72
< _=./task2.out
---
> _=./task2_1.out
[02/09/21]seed@VM:~$
```

Fig 6

```
[02/11/21]seed@VM:~$ ./task2.out
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.so.1.64.0
WINDOWID=67108874
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1523
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=seed
```

Fig:- Environment Variables of the child process

The following are created in this task:

- 1) Task2.c:- The file containing the code snippet of the task
- 2) Task2.out:- The executable file with child process `printenv()`
- 3) child:- Output from the file Task2.out file
- 4) Task2_1.out:- The executable file with parent process `printenv()`
- 5) child1:- Output from the Task2_1.out file

The output for **diff** command is interpreted as follows:

72c72 means that in the 72nd line (left) in left file is changed to the 72nd line (right) in the right file, where c stands for changing and the left and right numbers indicate the line number. The < denotes lines in the left file and > indicates in the right file showing the changed content. If both the programs were compiled into a file with the same name, there would not be any difference between the output of the parent and child process.

```
[02/11/21]seed@VM:~$ vi task2.c
[02/11/21]seed@VM:~$ g++ -o task2_nodiff task2.c -std=c++11
[02/11/21]seed@VM:~$ ./task2_nodiff > child
[02/11/21]seed@VM:~$ ./task2_nodiff > child1
[02/11/21]seed@VM:~$ diff child child1
[02/11/21]seed@VM:~$
```

Fig:- No difference scenario

Task 3:- Environment Variables and `execve()`

The aim of this task is to study how environment variables are affected when a new program is executed via `execve()`

Created a program task3.c where the 3rd argument of the `execve` command i.e., environment variable is set to **NULL**. The output of this program is nothing as only the shell is returned (as shown in Fig 7) . We then modify the 3rd argument of the `execve` command and set the environment variable to **environ**. The output of this program is that the environment variables are printed (as shown in Fig 8).

```
[02/09/21]seed@VM:~$ ./task3.out
[02/09/21]seed@VM:~$
```

Fig: 7 :- program output with the NULL parameter

```

[02/09/21]seed@VM:~$ ./task3.out
[02/09/21]seed@VM:~$ ./task3_1.out
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-

```

Fig 8:- program output with environ as 3rd parameter

The program does not have any environment variable when we pass **NULL** to `execve()`. By passing **environ** to `execve`, all the environment variables of the current process are passed to the program. Thus, we can conclude that the parameters for `execve` decides the environment variables of the process.

Task 4:- Environment Variables and system()

The aim of this task is to study how environment variables are affected when a new program is executed via the `system()` function.

The program is compiled and executed and as seen in Fig 9, even though we do not explicitly send any environment variables in the program, the output shows the environment variable of the current process. This happens because the `system` function implicitly passes the environment variables to the called function `/bin/sh`.

```

[02/09/21]seed@VM:~$ ./task4.out
LESSOPEN=| /usr/bin/lesspipe %s
GNOME_KEYRING_PID=
USER=seed
LANGUAGE=en_US
UPSTART_INSTANCE=
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_SEAT=seat0
SESSION=ubuntu
XDG_SESSION_TYPE=x11
COMPIZ_CONFIG_PROFILE=ubuntu-lowgfx
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
SHLVL=1
LIBGL_ALWAYS_SOFTWARE=1
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
HOME=/home/seed
QT4_IM_MODULE=xim
DESKTOP_SESSION=ubuntu
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
GTK_MODULES=gail:atk-bridge:unity-gtk-module
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
INSTANCE=

```

Fig 9

When the system function executes, it does not execute the command directly. It calls the shell instead and the shell executes the command. The shell internally calls the `execve` command, and the environment variables of the calling process are passed to the shell and the shell passes it to the `execve` command.

Task 5: Environment Variable and Set-UID Programs

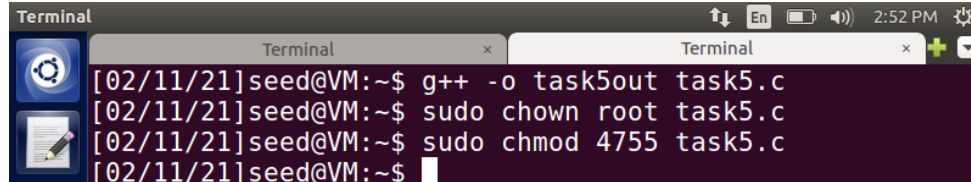
The aim of this task is to study about Set-UID which is an important security mechanism in Unix operating systems.

After compiling the given program, we change the ownership and permission of the file using the following commands:

sudo chown root filename (making the root as the owner of filename)

sudo chmod 4755 filename (making the program a SET-UID program by setting set-uid bit)

This makes the program a SET-UID root program. I initialized a ***new variable with name AB and value 'seedLabs'***, ***Set PATH value as /home/seed:\$PATH***, ***set LD_LIBRARY_PATH value as 'foo bar'*** using ***export*** command. The following screenshots shows the performed steps:



```
Terminal
[02/11/21]seed@VM:~$ g++ -o task5out task5.c
[02/11/21]seed@VM:~$ sudo chown root task5.c
[02/11/21]seed@VM:~$ sudo chmod 4755 task5.c
[02/11/21]seed@VM:~$
```

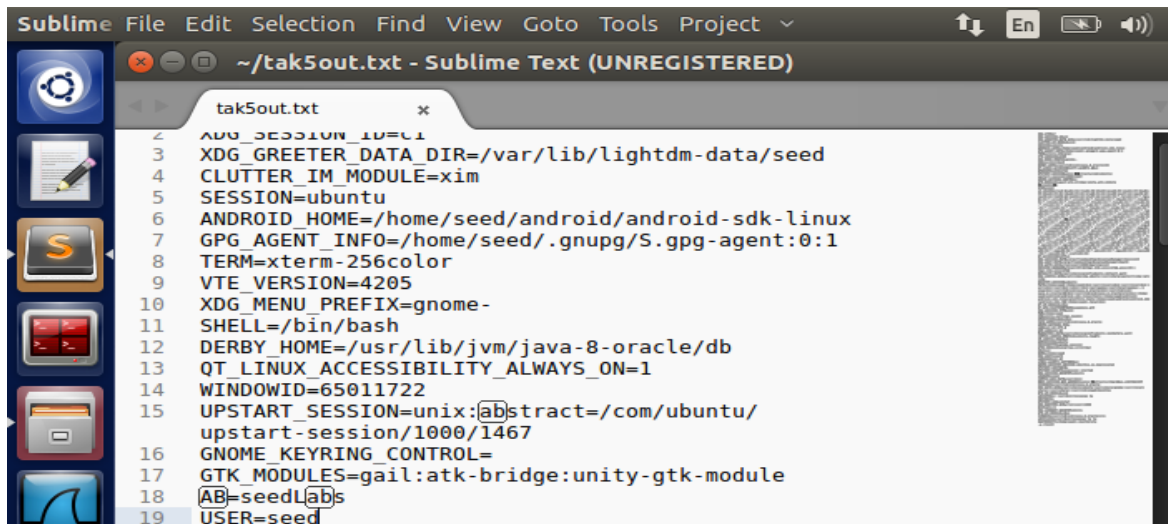
Fig 10:- Compiling and making it a set-UID program



```
[02/11/21]seed@VM:~$ export PATH=/home/seed:$PATH
[02/11/21]seed@VM:~$ export LD_LIBRARY_PATH='foo bar'
[02/11/21]seed@VM:~$ env | grep PATH
LD_LIBRARY_PATH=foo bar
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
PATH=/home/seed:/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/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
```

Fig 11:- Setting values to environment variables in the user shell

On running the above compiled program and storing the output in a text file named task5out.txt, it can be noted that the child process inherits the PATH and custom AB environment variable but there is no LD_LIBRARY_PATH environment variable.

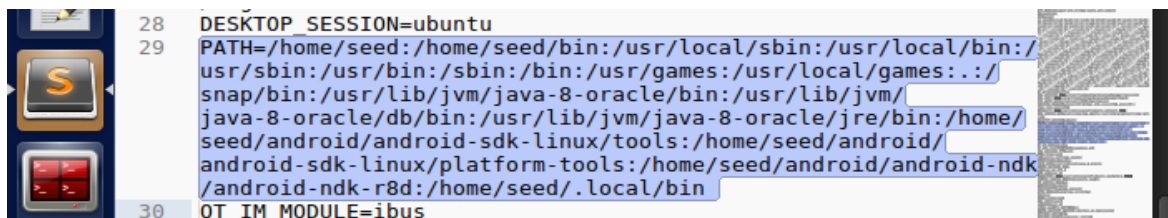


```

1  ~/task5out.txt - Sublime Text (UNREGISTERED)
2  tak5out.txt
3  2  ANDROID_SESSION_ID=01
4  3  XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
5  4  CLUTTER_IM_MODULE=xim
6  5  SESSION=ubuntu
7  6  ANDROID_HOME=/home/seed/android/android-sdk-linux
8  7  GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
9  8  TERM=xterm-256color
10 9  VTE_VERSION=4205
1110 XDG_MENU_PREFIX=gnome-
1211 SHELL=/bin/bash
1312 DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
1413 QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
1514 WINDOWID=65011722
1615 UPSTART_SESSION=unix:abstract=/com/ubuntu/
17upstart-session/1000/1467
1816 GNOME_KEYRING_CONTROL=
1917 GTK_MODULES=gail:atk-bridge:unity-gtk-module
2018 AB=seedlabs
2119 USER=seed

```

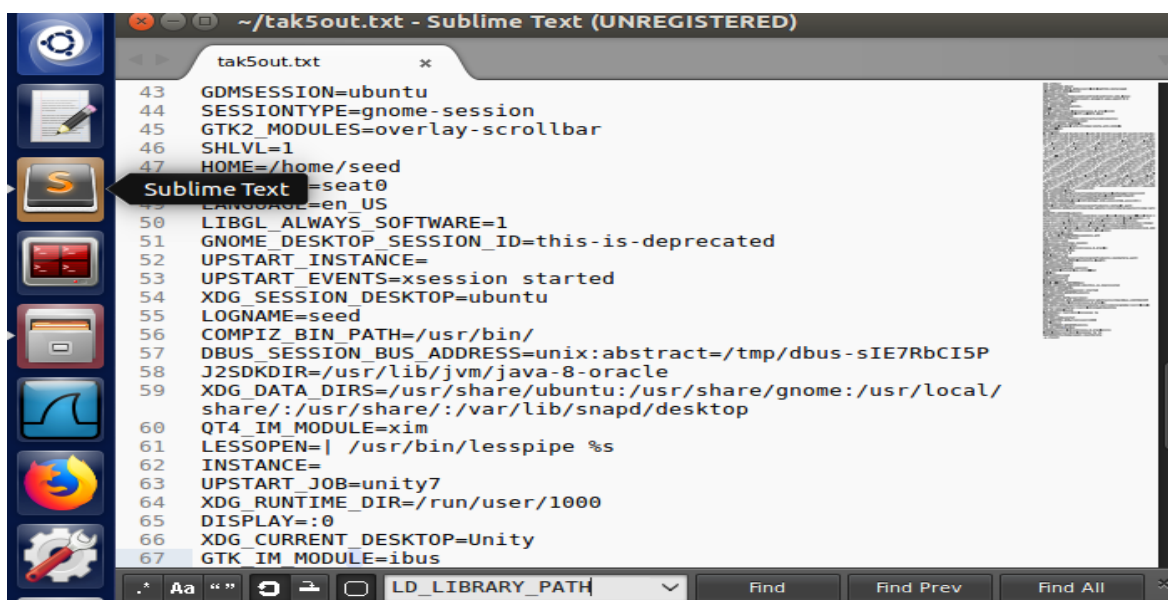
Fig 12:- task5out.txt file



```

28 28  DESKTOP_SESSION=ubuntu
29 29  PATH=/home/seed:/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/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-r8d:/home/seed/.local/bin
30 30  QT_IM_MODULE=ibus

```



```

43 43  GDMSESSION=ubuntu
44 44  SESSIONTYPE=gnome-session
45 45  GTK2_MODULES=overlay-scrollbar
46 46  SHLVL=1
47 47  HOME=/home/seed
48 48  XDG_SESSION_DESKTOP=ubuntu
49 49  LOGNAME=seed
50 50  COMPIZ_BIN_PATH=/usr/bin/
51 51  DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-sIE7RbCI5P
52 52  J2SDKDIR=/usr/lib/jvm/java-8-oracle
53 53  XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share:/usr/share:/var/lib/snapd/desktop
54 54  QT4_IM_MODULE=xim
55 55  LESSOPEN=| /usr/bin/lesspipe %s
56 56  INSTANCE=
57 57  UPSTART_JOB=unity7
58 58  XDG_RUNTIME_DIR=/run/user/1000
59 59  DISPLAY=:0
60 60  XDG_CURRENT_DESKTOP=Unity
61 61  GTK_IM_MODULE=ibus

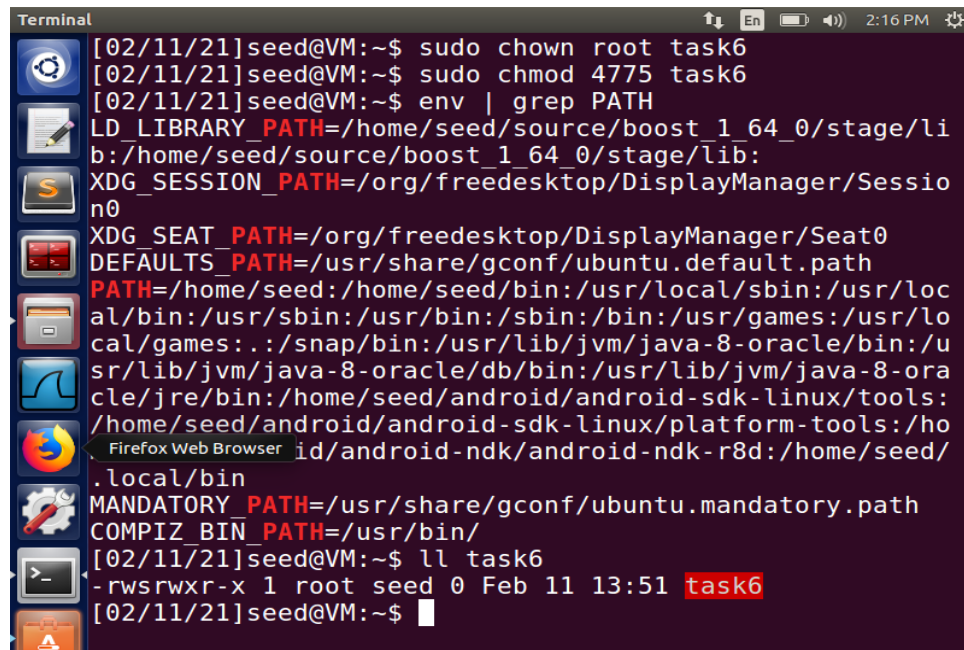
```

This shows that the SET-UID program's child process may not inherit all the environment variables of the parent process, LD_LIBRARY_PATH being one of them over here. It is a protection mechanism against malicious files being placed into shared libraries. The LD_LIBRARY_PATH is ignored here because the RUID and EUID are different. That is why only the other two environment variables are seen in the output.

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

The aim of this task is to study the effect of shell program invoking, system() within a Set-UID program which can be quite dangerous.

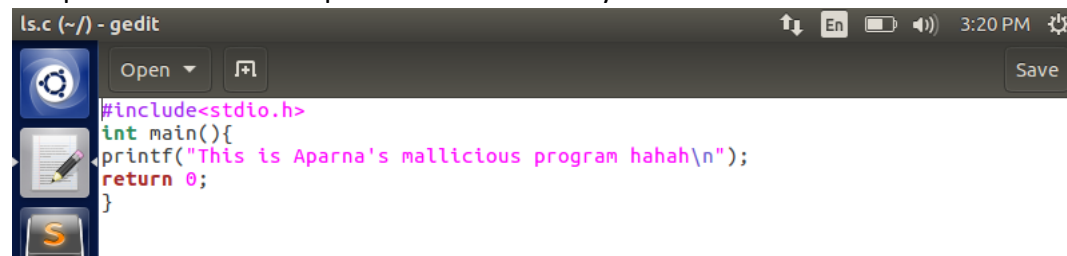
The given program is written in the file named task6.c and compiled into task6 file. Then the compiled program's owner is changed into root and its converted into a SET-UID program. Also, confirming that task6 is a SET-UID program with root as the owner



```
Terminal
[02/11/21]seed@VM:~$ sudo chown root task6
[02/11/21]seed@VM:~$ sudo chmod 4775 task6
[02/11/21]seed@VM:~$ env | grep PATH
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
PATH=/home/seed:/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/jvm/java-8-oracle/jre/bin:/home/seed/android/android-sdk-linux/tools:/home/seed/android/android-sdk-linux/platform-tools:/home/seed/android-ndk/android-ndk-r8d:/home/seed/.local/bin
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_BIN_PATH=/usr/bin/
[02/11/21]seed@VM:~$ ll task6
-rwsrwxr-x 1 root seed 0 Feb 11 13:51 task6
[02/11/21]seed@VM:~$
```

Fig 13

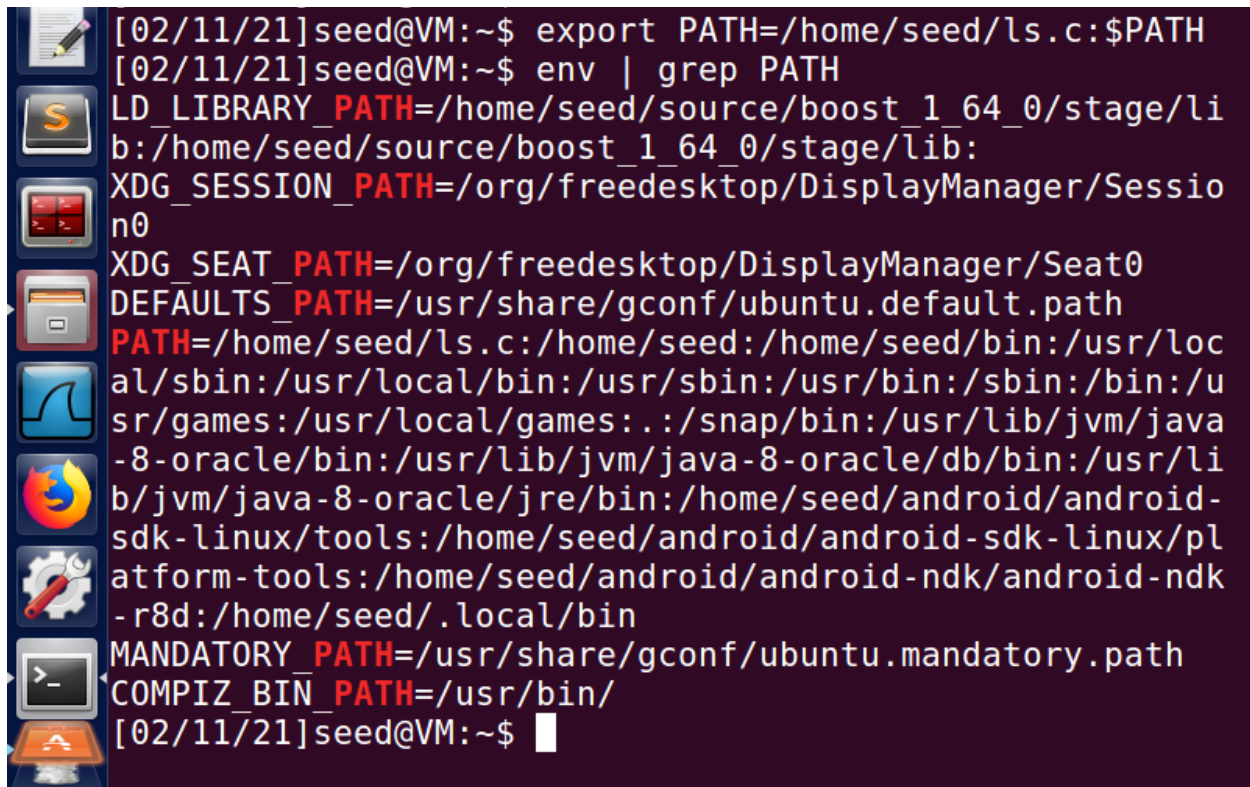
I created a new program ls.c and compiled it. This is the malicious file that I will be placing in the path. This file will replace the functionality of the ls command



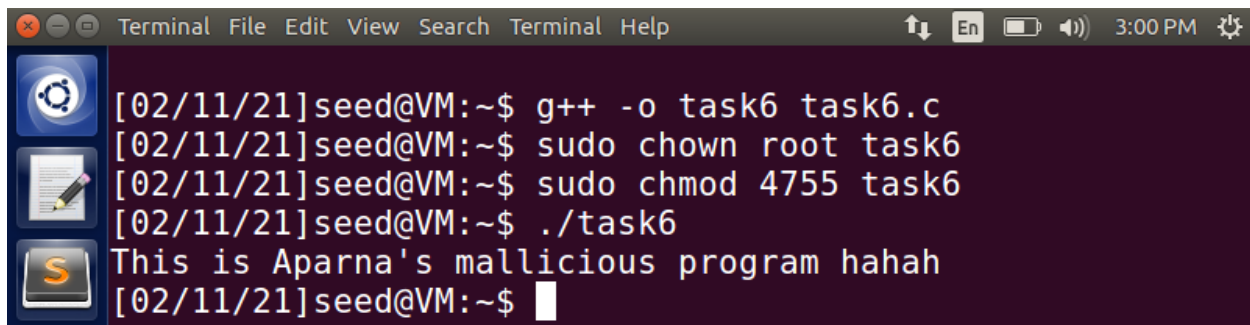
```
ls.c (~/) - gedit
#include<stdio.h>
int main(){
printf("This is Aparna's malicious program hahah\n");
return 0;
}
```

Fig 14:- My malicious ls.c program

To run my program instead of the standard ls program, I changed the value of environment variable PATH and provided the path to my file as the first value of the variable. This makes the program to search for the file in my directory first before any other directory and since I have the file with the same name as ls, the current program will execute my program.

A terminal window with a dark background and light-colored text. On the left side, there is a vertical dock with several application icons: a notepad, a terminal, a file manager, a web browser, a music player, a settings gear, and a terminal icon. The terminal text shows the user 'seed' at 'VM' in the home directory. They run 'export PATH=/home/seed/ls.c:\$PATH' and then 'env | grep PATH'. The output shows various system paths, with the new path '/home/seed/ls.c' added at the beginning of the PATH variable.

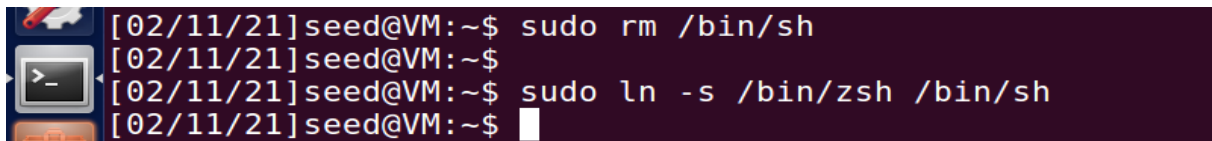
```
[02/11/21]seed@VM:~$ export PATH=/home/seed/ls.c:$PATH
[02/11/21]seed@VM:~$ env | grep PATH
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
PATH=/home/seed/ls.c:/home/seed:/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/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
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_BIN_PATH=/usr/bin/
[02/11/21]seed@VM:~$
```

A terminal window showing the execution of a program. The user 'seed' at 'VM' runs 'g++ -o task6 task6.c', 'sudo chown root task6', and 'sudo chmod 4755 task6'. Then they run './task6', which outputs 'This is Aparna's malicious program hahah'. The terminal window has a title bar with 'Terminal' and various icons. The dock on the left is the same as in the previous image.

```
[02/11/21]seed@VM:~$ g++ -o task6 task6.c
[02/11/21]seed@VM:~$ sudo chown root task6
[02/11/21]seed@VM:~$ sudo chmod 4755 task6
[02/11/21]seed@VM:~$ ./task6
This is Aparna's malicious program hahah
[02/11/21]seed@VM:~$
```

This illustrates how it is possible to modify the PATH environment variable to point to a desired file and run user-defined program(s) that may be malicious. It is potentially risky because of the inclusion of shell and environment variables because we use system(). Also, we have specified the relative path of the program instead of specifying the absolute path. Due to this, the system() will spawn a shell which will look for a ls program in the location specified by the PATH environment variable. Thus, the attacker can run a malicious code with root privileges by changing the PATH value to a malicious file with the same name as defined in the program since it is a root-owned SET-UID program. Hence using relative path and system function in a SET-UID program could lead to severe attacks.

To overcome the dash shell security mechanism of dropping SET-UID program's privileges on being called from one, link the /bin/sh to another shell

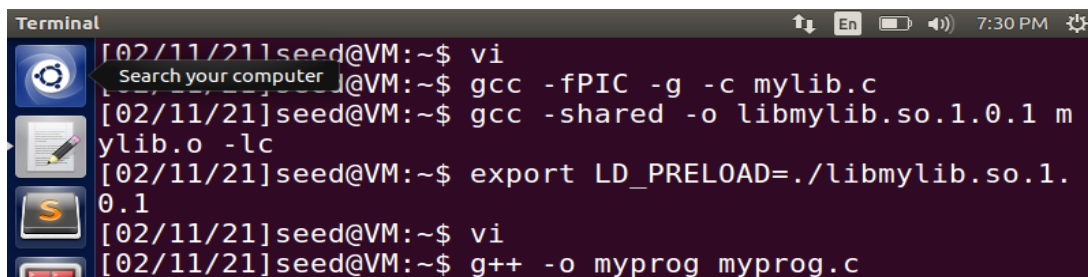


```
[02/11/21]seed@VM:~$ sudo rm /bin/sh
[02/11/21]seed@VM:~$
[02/11/21]seed@VM:~$ sudo ln -s /bin/zsh /bin/sh
[02/11/21]seed@VM:~$
```

Task 7: The LD_PRELOAD Environment Variable and Set-UID Programs

The aim of this task is to study how Set-UID programs deal with some of the environment variables.

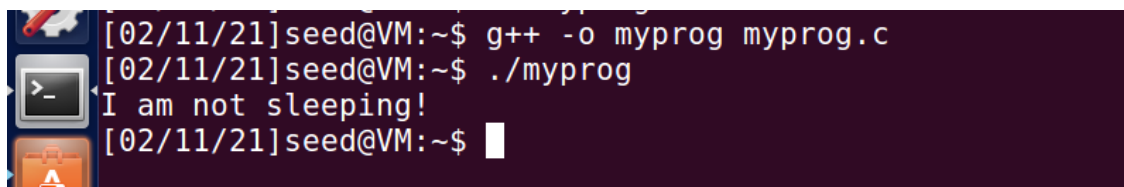
We create a program **mylib.c**, compile and make it a dynamic link library. We set the **LD_PRELOAD** environment variable using the export command to point to the dynamic library link (DLL) we just created. We then create a program **myprog.c** and compile it. Below Fig 15 shows these operations.



```
Terminal
[02/11/21]seed@VM:~$ vi
[02/11/21]seed@VM:~$ gcc -fPIC -g -c mylib.c
[02/11/21]seed@VM:~$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
[02/11/21]seed@VM:~$ export LD_PRELOAD=./libmylib.so.1.0.1
[02/11/21]seed@VM:~$ vi
[02/11/21]seed@VM:~$ g++ -o myprog myprog.c
```

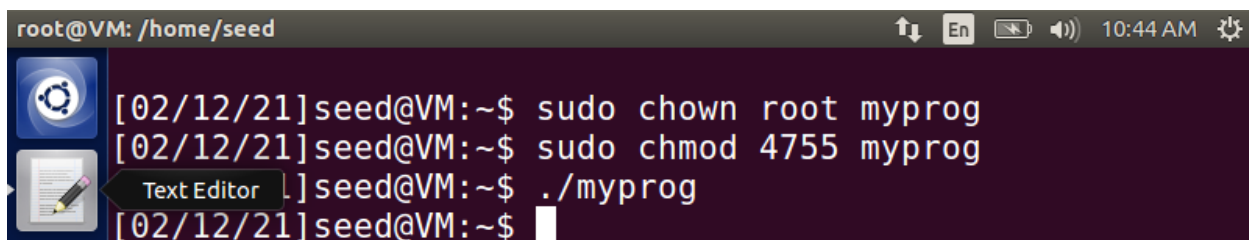
Fig 15

Running it as a normal user program our own library function gets invoked here. It means that this program calls the mylib DLL that we just created instead of the mylib.c DLL.



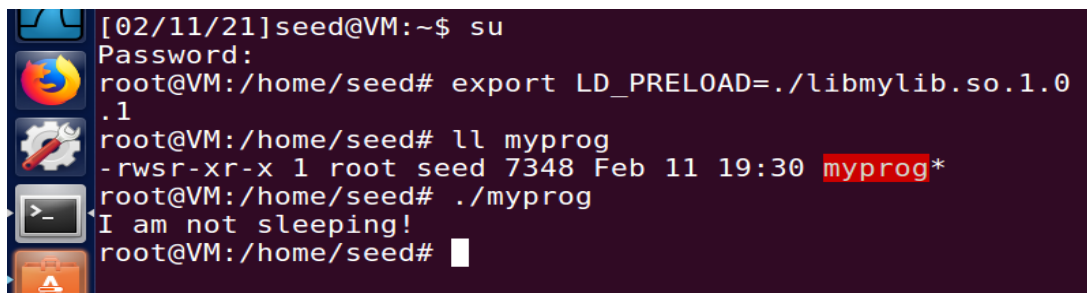
```
[02/11/21]seed@VM:~$ g++ -o myprog myprog.c
[02/11/21]seed@VM:~$ ./myprog
I am not sleeping!
[02/11/21]seed@VM:~$
```

Running it as a set-UID root program as a normal user



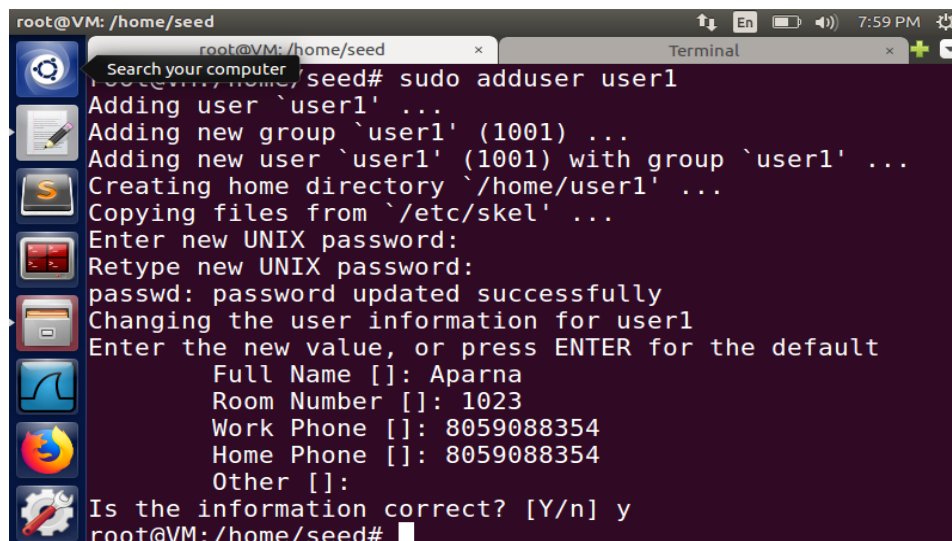
```
root@VM: /home/seed
[02/12/21]seed@VM:~$ sudo chown root myprog
[02/12/21]seed@VM:~$ sudo chmod 4755 myprog
[02/12/21]seed@VM:~$ ./myprog
I am not sleeping!
[02/12/21]seed@VM:~$
```

Running it as a set-UID program as a root user



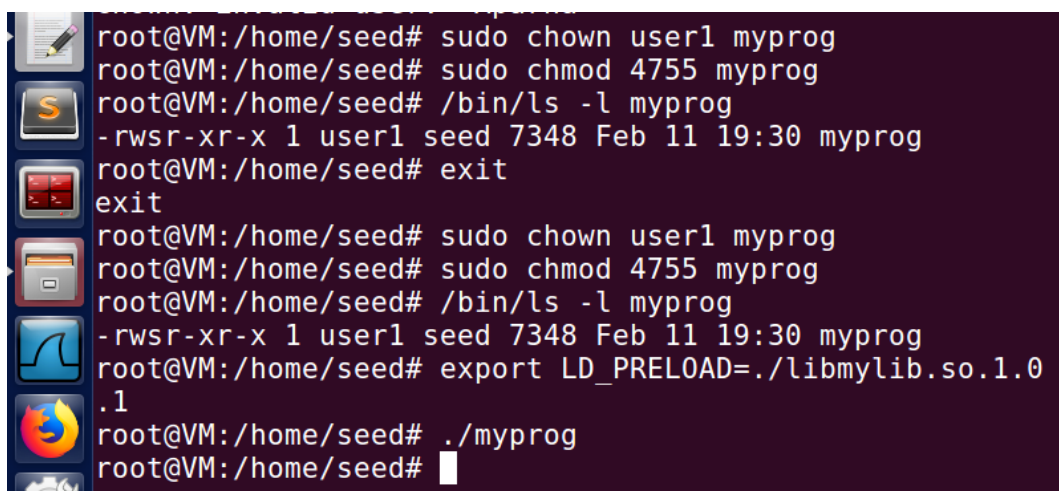
```
[02/11/21]seed@VM:~$ su
Password:
root@VM:/home/seed# export LD_PRELOAD=./libmylib.so.1.0
root@VM:/home/seed# ll myprog
-rwsr-xr-x 1 root seed 7348 Feb 11 19:30 myprog*
root@VM:/home/seed# ./myprog
I am not sleeping!
root@VM:/home/seed#
```

The above screenshot shows that we are executing the myprog.c program from root account. It is a set-UID program owned by the root. We set the LD_PRELOAD environment variable pointing to the DLL we created. When we execute the program, the program calls the mylib.c DLL that we have created.



```
root@VM:/home/seed# sudo adduser user1
Adding user `user1' ...
Adding new group `user1' (1001) ...
Adding new user `user1' (1001) with group `user1' ...
Creating home directory `/home/user1' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for user1
Enter the new value, or press ENTER for the default
  Full Name []: Aparna
   Room Number []: 1023
   Work Phone []: 8059088354
   Home Phone []: 8059088354
    Other []:
Is the information correct? [Y/n] y
root@VM:/home/seed#
```

Fig 16: Creating a new User



```
root@VM:/home/seed# sudo chown user1 myprog
root@VM:/home/seed# sudo chmod 4755 myprog
root@VM:/home/seed# /bin/ls -l myprog
-rwsr-xr-x 1 user1 seed 7348 Feb 11 19:30 myprog
root@VM:/home/seed# exit
exit
root@VM:/home/seed# sudo chown user1 myprog
root@VM:/home/seed# sudo chmod 4755 myprog
root@VM:/home/seed# /bin/ls -l myprog
-rwsr-xr-x 1 user1 seed 7348 Feb 11 19:30 myprog
root@VM:/home/seed# export LD_PRELOAD=./libmylib.so.1.0
root@VM:/home/seed# ./myprog
root@VM:/home/seed#
```

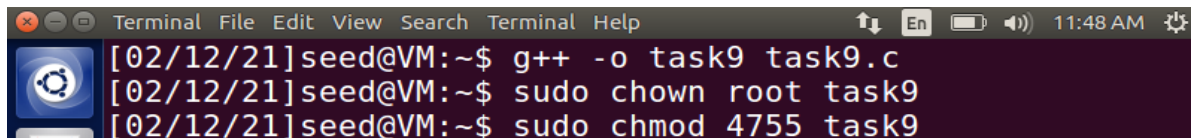
Fig 17

Fig 16 shows how we have created a new user user1. In Fig 17 we can see how to make the myprog.c program a set-UID program owned by user1. We then set the LD_PRELOAD environment variable pointing to the DLL we created. When we execute the program from another user account like seed, the program sleeps for some time. This means that the program does not invoke the DLL we created.

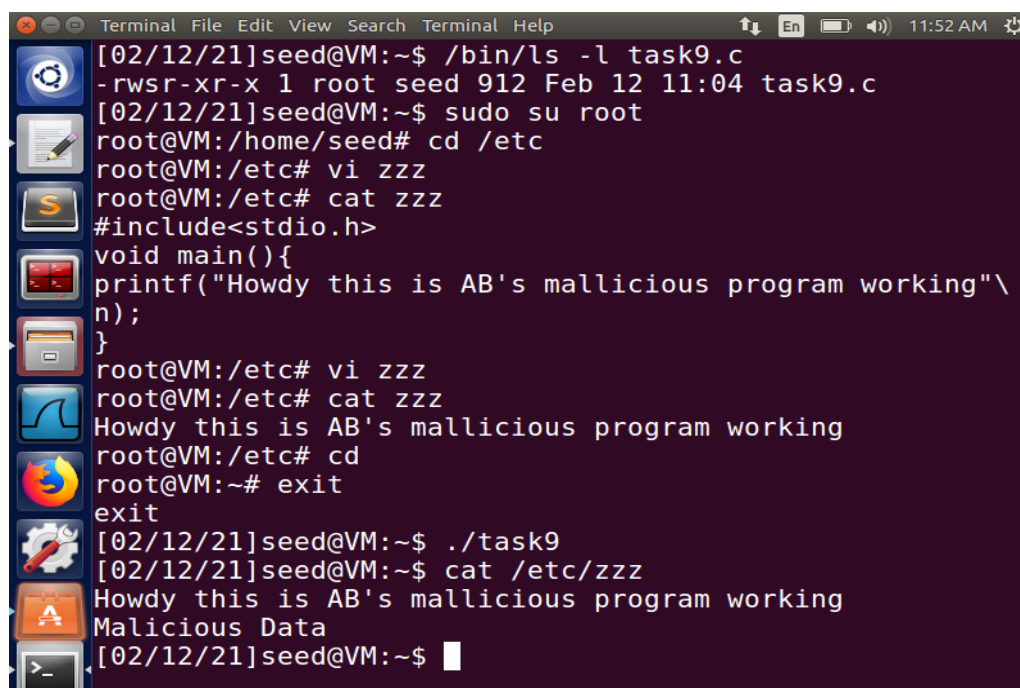
This behavior indicates that the LD_PRELOAD variable is present if the effective and real ID are the same and is ignored if they are different. It is basically a protection mechanism of the SET-UID program. In the first and third scenario, since the owner and the account executing the file were the same, the LD_PRELOAD variable was present and user-defined library was preloaded. Whereas, in the second case, the effective ID was of root and real ID was of seed, the LD_PRELOAD variable was ignored, and system-defined sleep function was called instead. In the fourth scenario, myprog is a Set-UID program owned by a user and run by another user. Hence LD_PRELOAD is ignored again because it is a Set-UID program.

Task 9: Capability Leaking

The aim of this task is to study the principle of Least Privilege, Set-UID programs that often permanently relinquish their root privileges if such privileges are not needed anymore.



```
Terminal File Edit View Search Terminal Help
[02/12/21]seed@VM:~$ g++ -o task9 task9.c
[02/12/21]seed@VM:~$ sudo chown root task9
[02/12/21]seed@VM:~$ sudo chmod 4755 task9
```



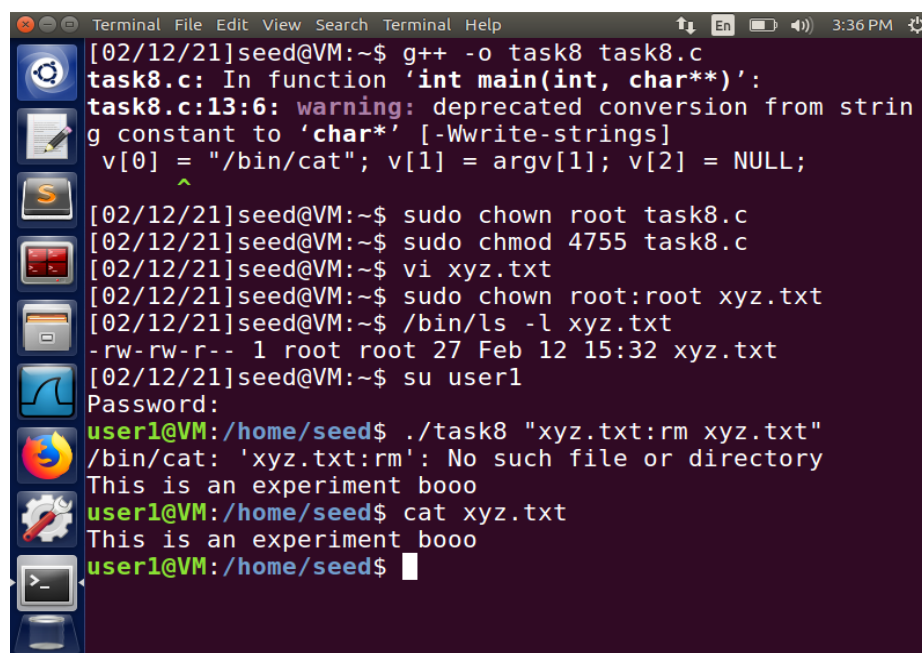
```
Terminal File Edit View Search Terminal Help
[02/12/21]seed@VM:~$ /bin/ls -l task9.c
-rwsr-xr-x 1 root seed 912 Feb 12 11:04 task9.c
[02/12/21]seed@VM:~$ sudo su root
root@VM:/home/seed# cd /etc
root@VM:/etc# vi zzz
root@VM:/etc# cat zzz
#include<stdio.h>
void main(){
printf("Howdy this is AB's mallicious program working"\n);
}
root@VM:/etc# vi zzz
root@VM:/etc# cat zzz
Howdy this is AB's mallicious program working
root@VM:/etc# cd
root@VM:~# exit
exit
[02/12/21]seed@VM:~$ ./task9
[02/12/21]seed@VM:~$ cat /etc/zzz
Howdy this is AB's mallicious program working
Mallicious Data
[02/12/21]seed@VM:~$
```

Created a program task9.c, compiled it and made it a Set-UID program owned by root. I then Logged in as root and created a program **zzz** in the **/etc** directory. Exited from the root account and executed the program from seed account. We can notice that the file **/etc/zzz** is modified by appending the content of the child process into the file. This is capability leaking. The above screen shots show the mentioned operations.

This happens because even though in the program, we dropped the privileges, we did not close the file at the right time and hence the file was still running with privileged permissions that allowed the data in the file to be modified, even without the right permissions. The child inherits copies of the parent's set of open file descriptors during fork call. Hence the malicious user can successfully modify the content of a privileged file. This shows that it is important to close the file descriptor after dropping privileges, for it to have the appropriate permissions. To avoid such attacks, the file descriptor must be closed before the fork call.

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

Here, first I compile the program provided into a file named task8.c. Next, this file is converted into a root-owned SET-UID program with executable permission to other users. I then created a xyz.txt file that is owned by root with no write privileges to other users. Now I login into user1 account and execute the program. The program displays the contents of the xyz.txt file and deletes the file on which it did not have the write permission by using the rm command after the : . This shows that even though user1 did not have any permission to write, it could remove a file easily by assuming the privileges of the root user.



```
Terminal File Edit View Search Terminal Help 3:36 PM
[02/12/21]seed@VM:~$ g++ -o task8 task8.c
task8.c: In function 'int main(int, char**)':
task8.c:13:6: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
  v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;
  ^
[02/12/21]seed@VM:~$ sudo chown root task8.c
[02/12/21]seed@VM:~$ sudo chmod 4755 task8.c
[02/12/21]seed@VM:~$ vi xyz.txt
[02/12/21]seed@VM:~$ sudo chown root:root xyz.txt
[02/12/21]seed@VM:~$ /bin/ls -l xyz.txt
-rw-rw-r-- 1 root root 27 Feb 12 15:32 xyz.txt
[02/12/21]seed@VM:~$ su user1
Password:
user1@VM:/home/seed$ ./task8 "xyz.txt:rm xyz.txt"
/bin/cat: 'xyz.txt:rm': No such file or directory
This is an experiment booo
user1@VM:/home/seed$ cat xyz.txt
This is an experiment booo
user1@VM:/home/seed$
```

Fig 18:- with system()


```
user1@VM: /home/seed
[02/12/21]seed@VM:~$ vi xyz.txt
[02/12/21]seed@VM:~$ vi task8.c
[02/12/21]seed@VM:~$ vi task8.c
[02/12/21]seed@VM:~$ g++ -o task8 task8.c
task8.c: In function 'int main(int, char**)':
task8.c:13:6: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
  v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;
      ^
[02/12/21]seed@VM:~$ sudo chown root task8.c
[02/12/21]seed@VM:~$ sudo chmod 4755 task8.c
[02/12/21]seed@VM:~$ sudo chown root:root xyz.txt
[02/12/21]seed@VM:~$ /bin/ls -l xyz.txt
-rw-rw-r-- 1 root root 27 Feb 12 15:37 xyz.txt
[02/12/21]seed@VM:~$ su user1
Password:
```

Fig 19:- With execve()

```
user1@VM: /home/seed
user1@VM:/home/seed$ ./task8 xyz.txt:rm xyz.txt
/bin/cat: 'xyz.txt:rm': No such file or directory
user1@VM:/home/seed$ ./task8 "xyz.txt:rm xyz.txt"
/bin/cat: 'xyz.txt:rm xyz.txt': No such file or directory
user1@VM:/home/seed$ ./task8 "xyz.txt;rm xyz.txt"
/bin/cat: 'xyz.txt;rm xyz.txt': No such file or directory
user1@VM:/home/seed$ ./task8 xyz.txt;rm xyz.txt
This is an experiment booo
rm: remove write-protected regular file 'xyz.txt'? y
rm: cannot remove 'xyz.txt': Permission denied
user1@VM:/home/seed$
```

I modified the code to have the `execve()` command and comment the `system()` command. When I execute the program without `""`, the program executes only till the `;` and displays the contents of the file. The remaining part of the argument consisting of the `rm` command is not executed since it does not have permissions. When I try to execute the program with `""`, the program searches for the exact match. So, a file by that name would not exist.

When `system()` executes, it does not execute the command directly. It calls the shell instead and executes the command. So, if the program is a Set-UID program, the user will have temporary root privileges and can remove any file they wish to with root privileges. Multiple commands can

be passed together using the `" "`. There is no input validation while using `system()`, but there is some when we use `execve()`. When we use `execve()` command it replaces the program with the called program and passes the argument strings exactly as specified and does not interpret quotes. Thus, when we pass something after the `' ; '` it is treated as a new command and root privileges would have been lost. This avoids this kind of attack.