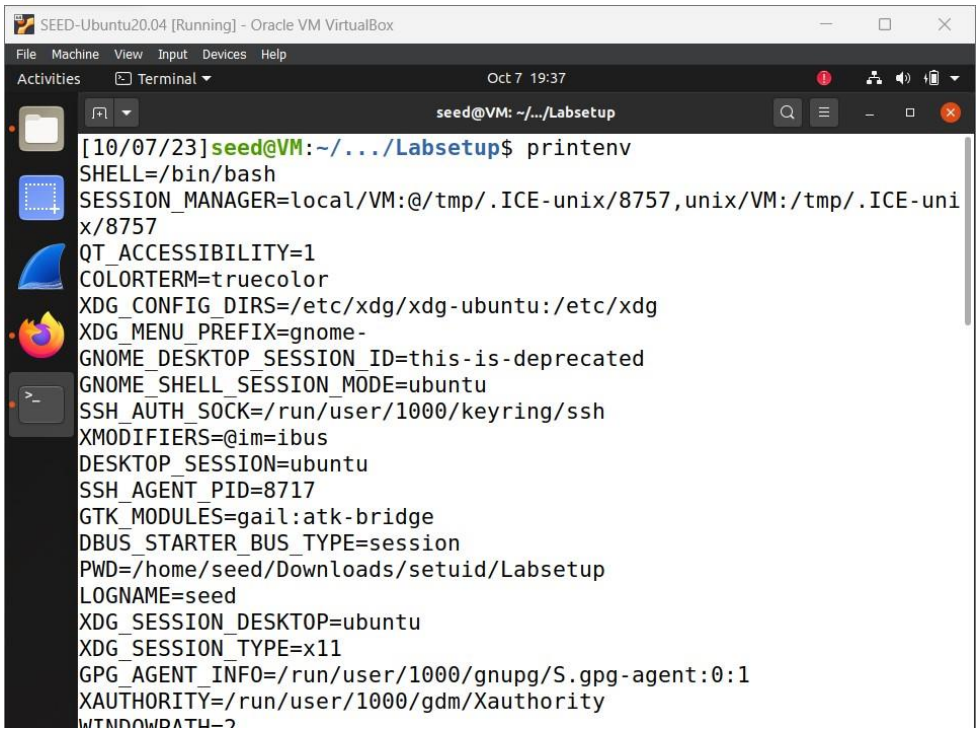
NAME: YASH SNEHAL SHETIYA

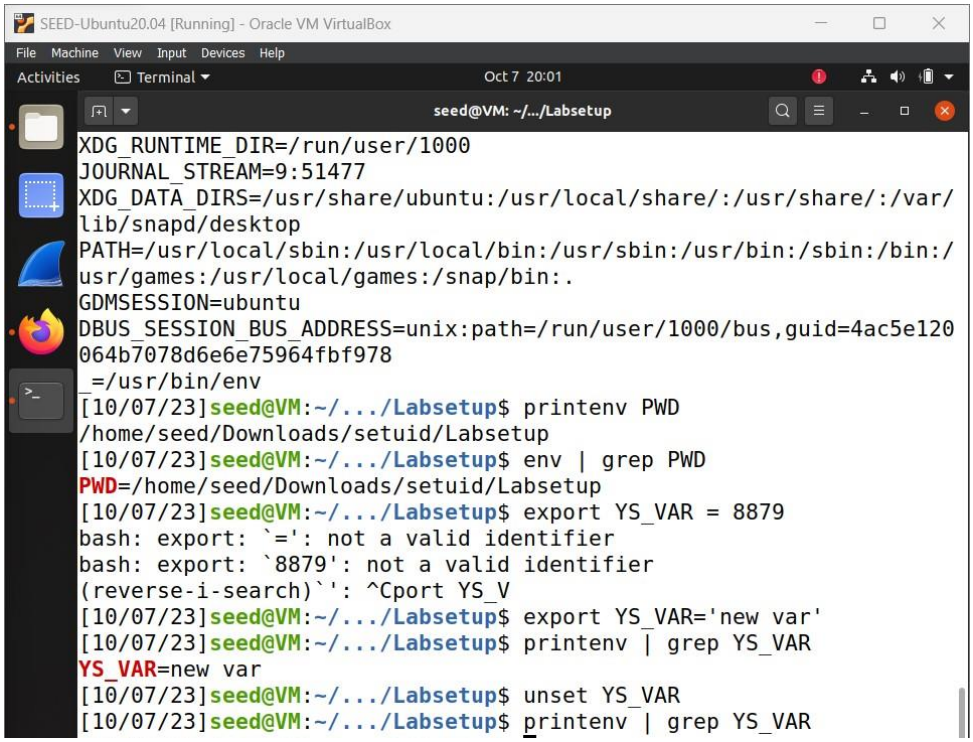
SUID: 9276568741

LAB: Environment Variable and Set-UID Program Lab

TASK 1:

We use the printenv and env command to print out the environment variables





TASK 2:

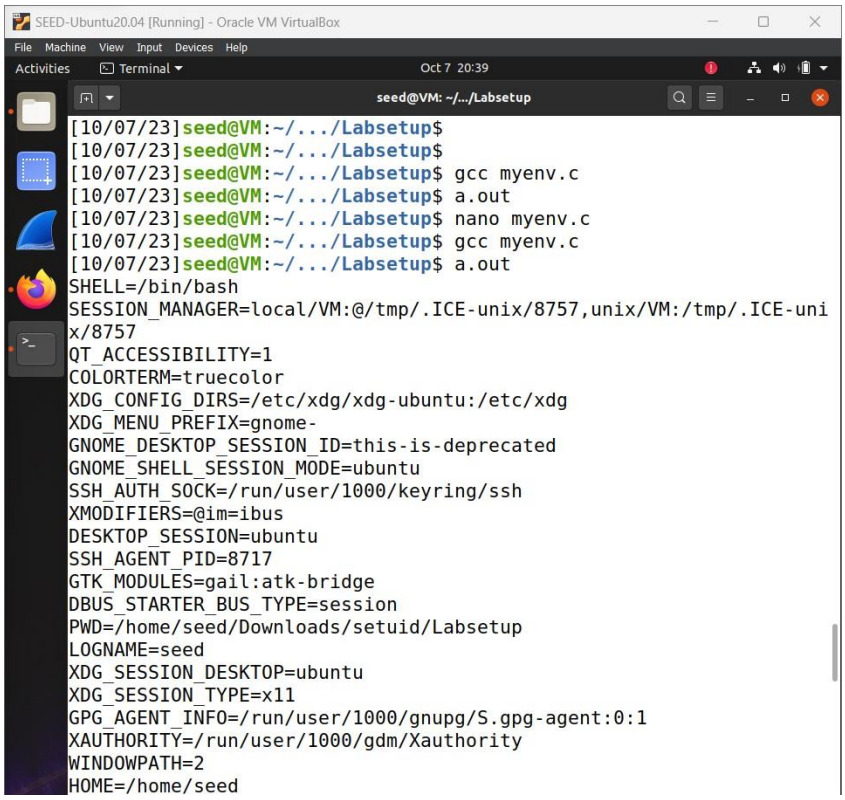
```
[10/07/23]seed@VM:~/.../Labsetup$ ls
cap_leak.c  catall.c  myenv.c  myprintenv.c
[10/07/23]seed@VM:~/.../Labsetup$ cat myprintenv.c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

extern char **environ;

void printenv()
{
  int i = 0;
  while (environ[i] != NULL) {
     printf("%s\n", environ[i]);
     i++;
  }
}

void main()
{
  pid_t childPid;
  switch(childPid = fork()) {
    case 0:  /* child process */
      printenv();
      exit(0);
    default:  /* parent process */
      // printenv();
      exit(0);
  }
```

I modified as per given terms and the executable was named file1. When checked difference in file and file 1 there was no difference.

```
      exit(0);
    default:  /* parent process */
       printenv();
      exit(0);
  }
}
[10/11/23]seed@VM:~/.../Labsetup$ ls
a.out       file              mylib.o       task5
cap_leak    file1             myprintenv.c  task5.c
cap_leak.c  libmylib.so.1.0.1 myprog        task6
catall      ls                myprog.c      task6.c
catall2     myenv.c           task4         testing2.txt
catall.c    mylib.c           task4.c       testing.txt
[10/11/23]seed@VM:~/.../Labsetup$ diff file file1
[10/11/23]seed@VM:~/.../Labsetup$ cat file
SHELL=/bin/bash
```

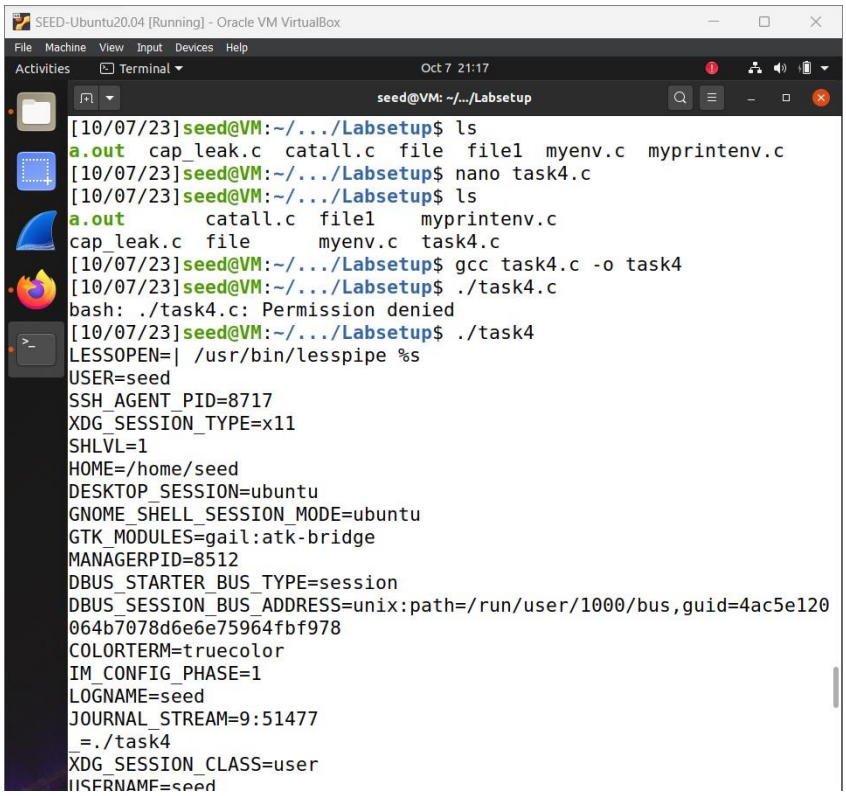
TASK 3:

It can be concluded that environment variable passed in execve function is NULL and hence there is no output. When we make the required changes and pass environ the global environment variable, it can be successfully compiled and the variables will be printed out.



TASK 4:

System() invokes the fork () function to create a new child process which uses execl to generate a process to call /bin/sh and executes in this shell.



TASK 5:

We make a new file named task5.c and compile it. Then we change the effective user of the file to root and set the UID bit, we even check if it set correctly or not.

```
[10/10/23]seed@VM:~/.../Labsetup$ nano task5.c
[10/10/23]seed@VM:~/.../Labsetup$ ls
a.out       catall.c  file1    myprintenv.c  task4.c
cap_leak.c  file      myenv.c  task4         task5.c
[10/10/23]seed@VM:~/.../Labsetup$ gcc task5.c -o task5
[10/10/23]seed@VM:~/.../Labsetup$ sudo chown root task5
[10/10/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 task5
[10/10/23]seed@VM:~/.../Labsetup$ ls -l task5
-rwsr-xr-x 1 root seed 16768 Oct 10 22:00 task5
[10/10/23]seed@VM:~/.../Labsetup$ ls
a.out       catall.c  file1    myprintenv.c  task4.c  task5.c
cap_leak.c  file      myenv.c  task4         task5
[10/10/23]seed@VM:~/.../Labsetup$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/g
ames:/usr/local/games:/snap/bin:.
[10/10/23]seed@VM:~/.../Labsetup$
[10/10/23]seed@VM:~/.../Labsetup$
[10/10/23]seed@VM:~/.../Labsetup$ export LD_LIBRARY_PATH='88797'
[10/10/23]seed@VM:~/.../Labsetup$ export yash='1'
[10/10/23]seed@VM:~/.../Labsetup$ ./task5
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/8757,unix/VM:/tmp/.ICE-uni
x/8757
QT_ACCESSIBILITY=1
```

Now we use the export command to set the following environment variables and then execute the program to view the environment variables.

```
USER=seed
GNOME_TERMINAL_SERVICE=:1.383
DISPLAY=:0
SHLVL=1
yash=1
QT_IM_MODULE=ibus
DBUS_STARTER_ADDRESS=unix:path=/run/user/1000/bus,guid=4ac5e120064b
7078d6e6e75964fbf978
XDG_RUNTIME_DIR=/run/user/1000
JOURNAL_STREAM=9:51477
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share/:/var/
lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/games:/usr/local/games:/snap/bin:.
GDMSESSION=ubuntu
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus,guid=4ac5e120
064b7078d6e6e75964fbf978
_=./task5
[10/10/23]seed@VM:~/.../Labsetup$ ./task5 | grep LD_LIB
[10/10/23]seed@VM:~/.../Labsetup$ ./task5 | grep yash
yash=1
[10/10/23]seed@VM:~/.../Labsetup$ printenv | grep LD_LIB
LD_LIBRARY_PATH=88797
```

We could see that our own variable yash was visible but the LD_LIBRARY_PATH variable is not printed by the program. When we don't use the program and just want to view the environment variables by the command printenv we can view the LD_LIBRARY_PATH variable.

It is not visible to us because of a protection mechanism, the LD_LIBRARY_PATH variable is setting the path of the dynamic library which will not be in environment variables for the child process.

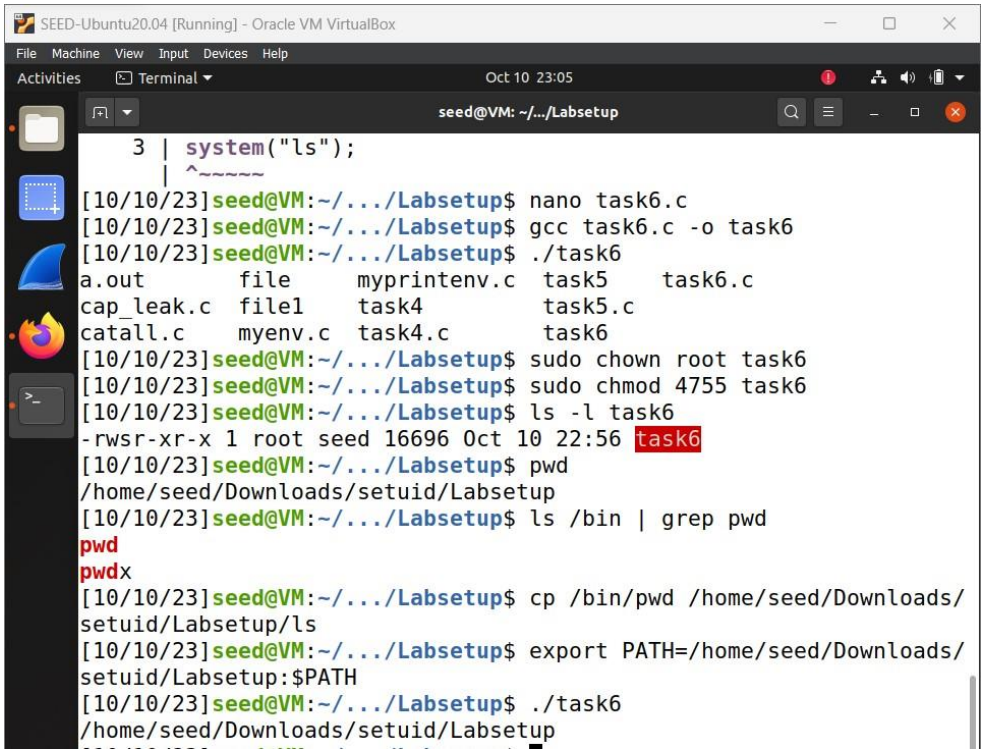
TASK 6:

We create a new file task6.c with the given code to execute the ls command and compile it. After running it we can see that it performs the (ls) operation and lists the file in the directory.

Before doing this task execute the command (sudo ln -s /bin/zsh /bin/sh)

We copy the /bin/pwd command to this directory and name it (ls) and also modify the path variable to this directory so that priority will be given to this directory when searching.

Now when we execute the task6 program, the system("ls") command will search in the /home/seed/Downloads/setuid/Labsetup/ls in which we have copied the pwd program so when the program executes it will be equivalent to pwd operation. Hence we achieve the use of a setUID program to run our own code.

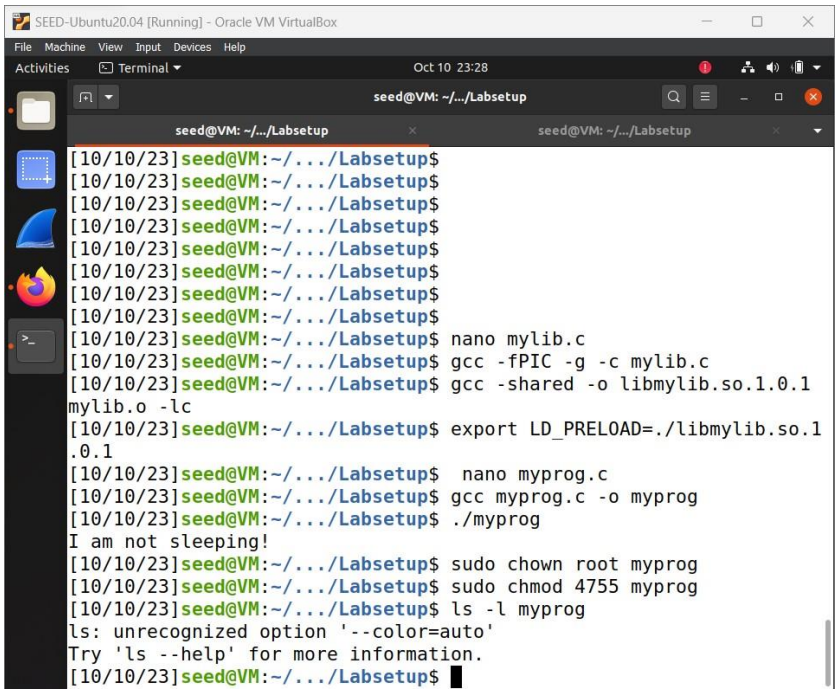


TASK 7:

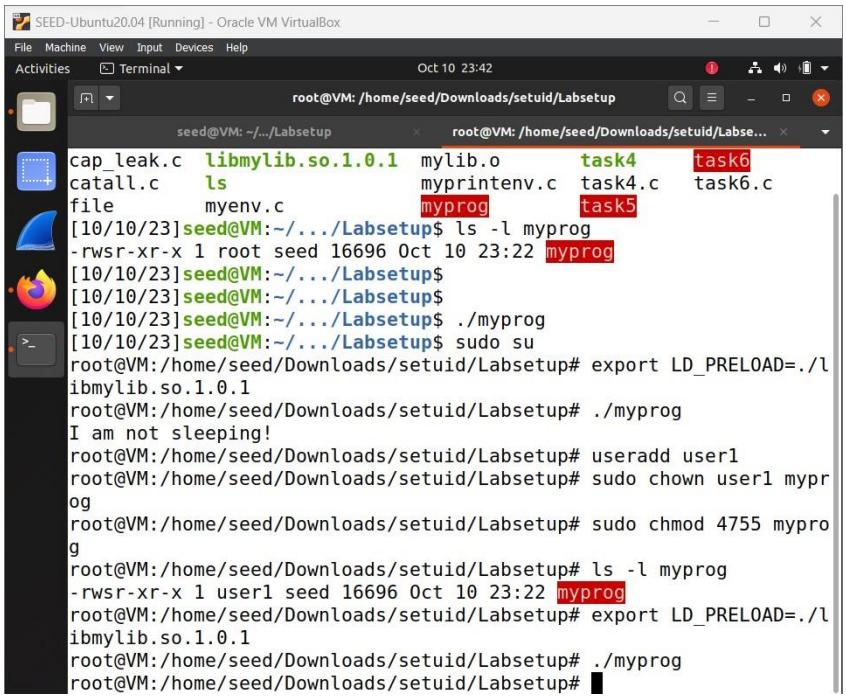
First we make a new file named mylib.c and compile it into libmylib.so.1.0.1 library and set the LD_PRELOAD environment variable as given. Now, we make and compile myprog.c in the same directory.

Then we run myprog as a regular program and we can see the output indicating that sleep function is linked to the custom dynamic link library.

We change the effective user of the myprog program to root and also set it as a SETUID program.



We can see the changes:



Now that we have set it as a SETUID program, we run it and can see that after running for about 1 second there is no output.

Then we need to make the SETUID program root executable, when we run the program by root the program shows the output that it first showed.

Checking the output when myprog is a SETUID user11 program i.e now user1 is the owner of myprog. We export the LD_PRELOAD again and run the program. the program runs for about 1 second and there is no output.

We can conclude that defence mechanism for LD_PRELOAD is through links. First when the effective user and the real user is both seed LD_PRELOAD is not ignored and the result is printed.

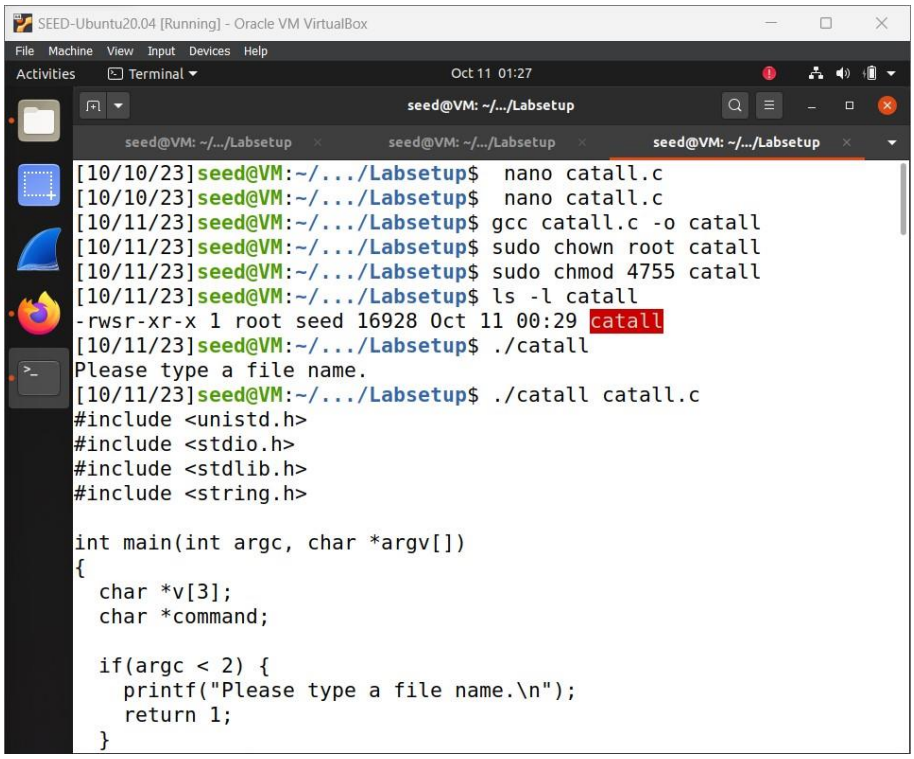
When we change the effective user to root and real user is seed, the LD_PRELOAD is blocked and hence there is no output and sleep() is executed.

Now when the effective and real user is root the LD_PRELOAD is linked to the dynamic library and we can see the output.

When we change to user1 the effective user is user1 and the real user is seed, the LD_PRELOAD is blocked and sleep() is executed so there is no output.

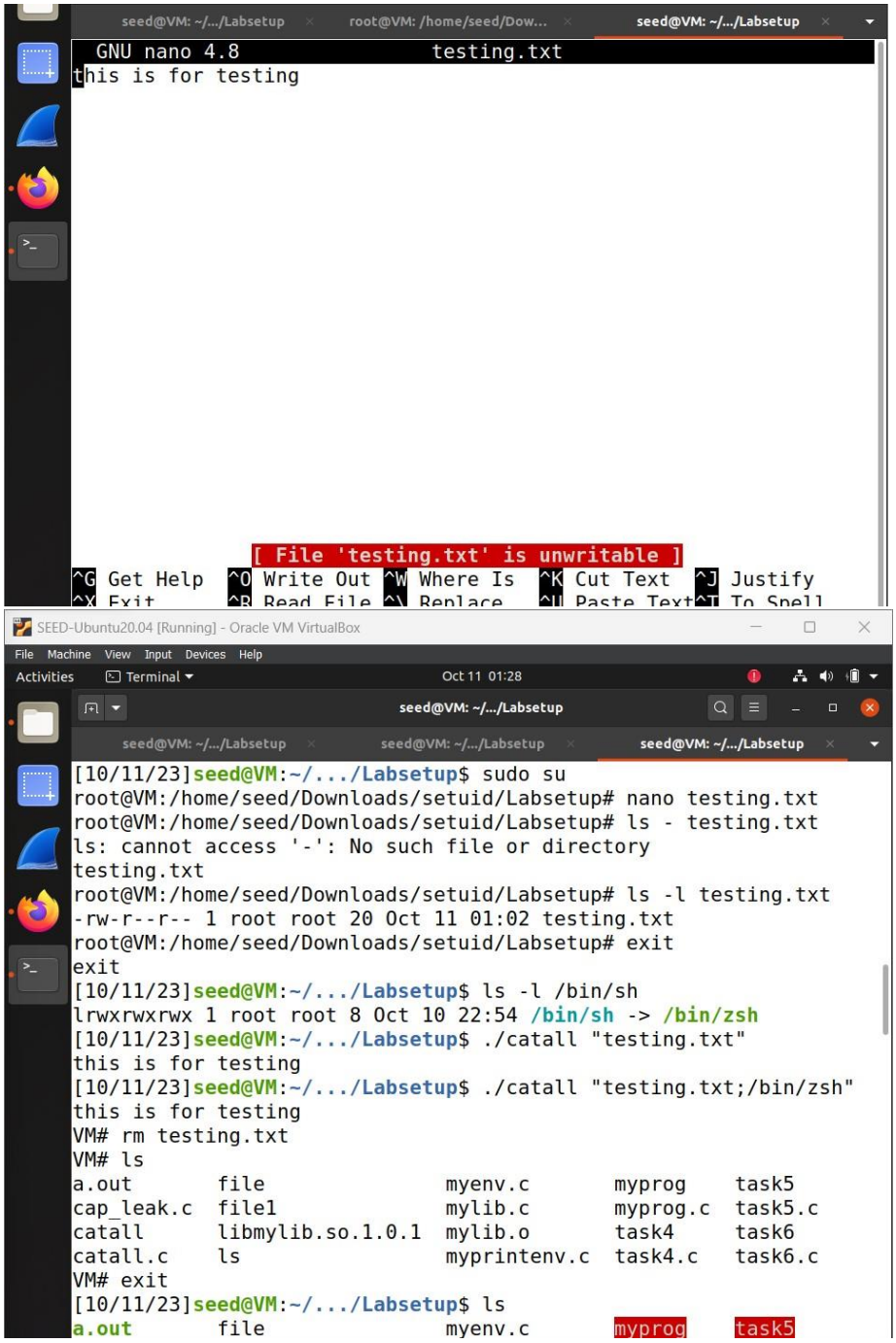
TASK 8:

As given we have catall.c file, we compile it and set effective user to root and set it as SETUID program. We check the execution of the file and it does execute and prints out the content properly.



Now, we create a new file testing.txt using root user so that the owner of the file will be root and only root has permission to modify and delete the file.

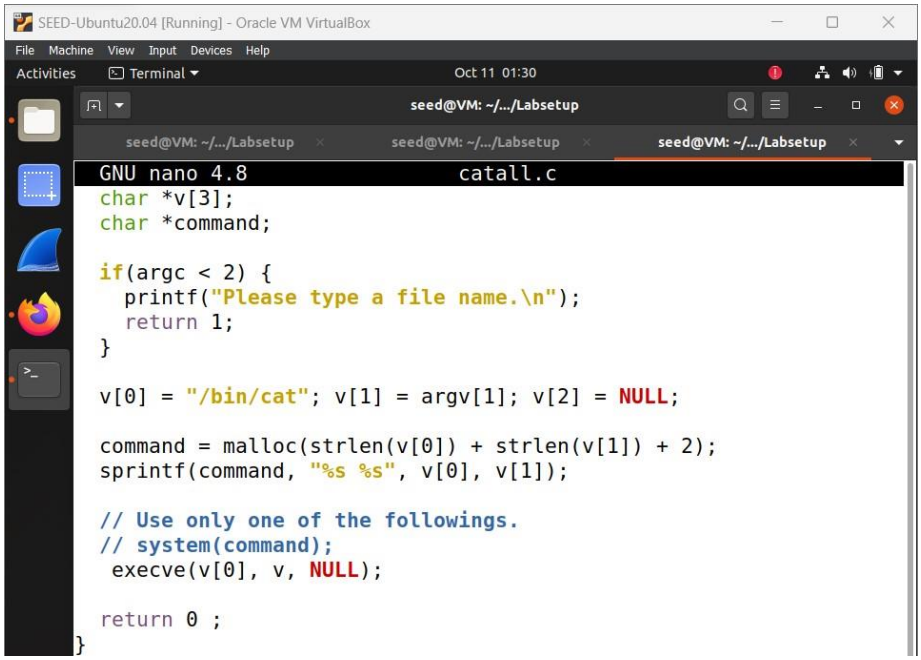
We can see below that this file isnt writeable to us.

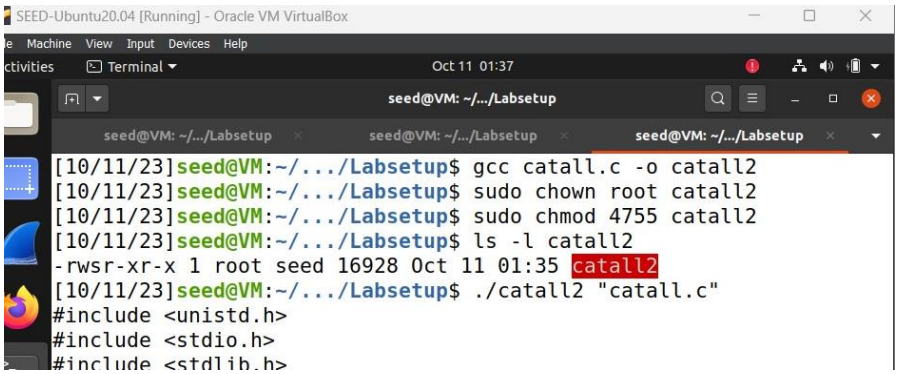


The program uses system() to call the command, I obtained a shell with root privileges through catall successfully delete the testing.txt file as seen above.

//

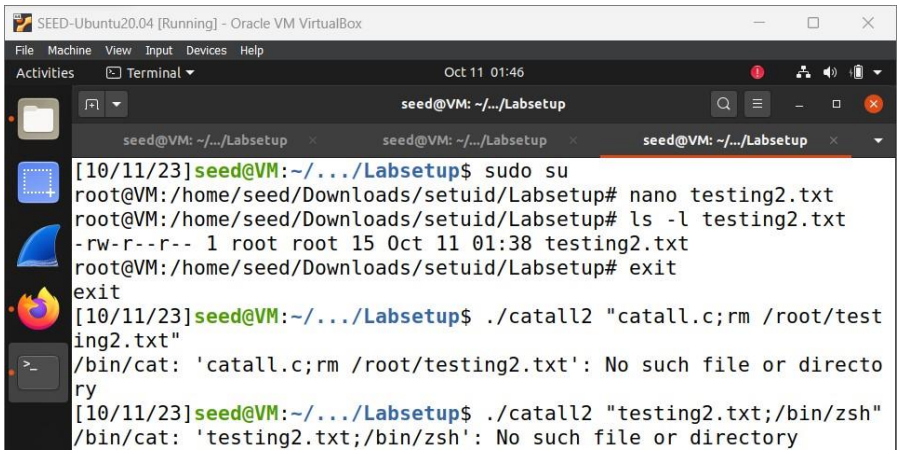
Commenting out the system() command and now the program will use execve() to call the command.





Similarly creating one more testing2.txt file and placing it in root directory.

Just like above I tried to execute the program to get shell with root privilege but it does not work.



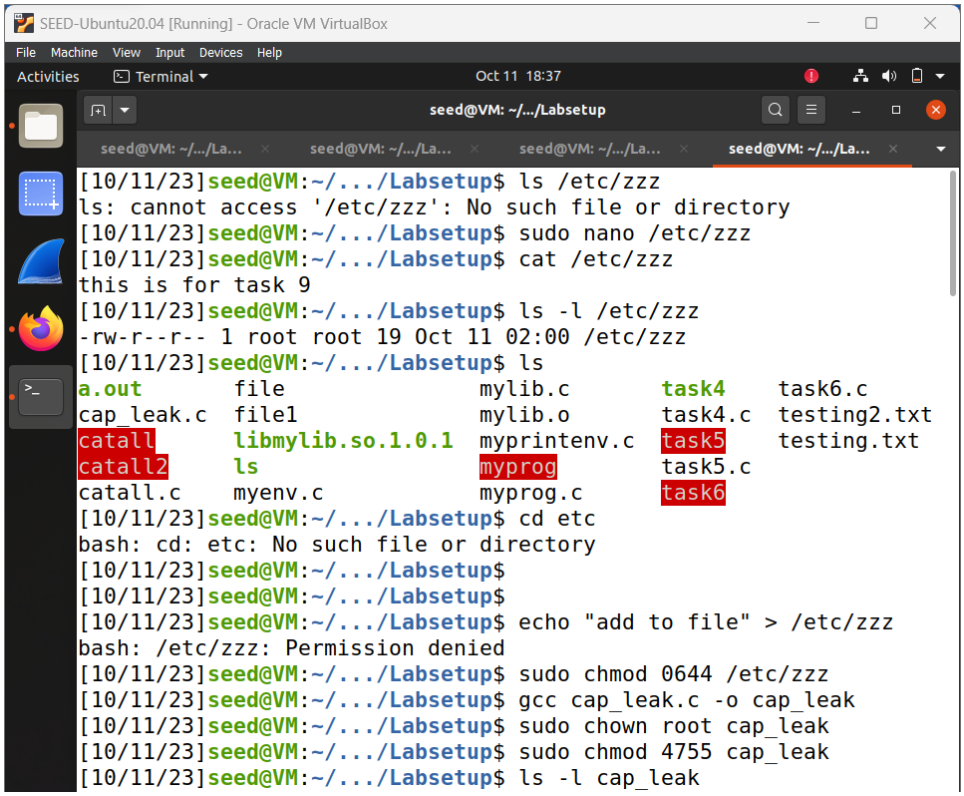
In system() the shell is called to execute the command, the execve() only executes a command so at this time only one process can be executed as a command. Basically there is separation of code and data.

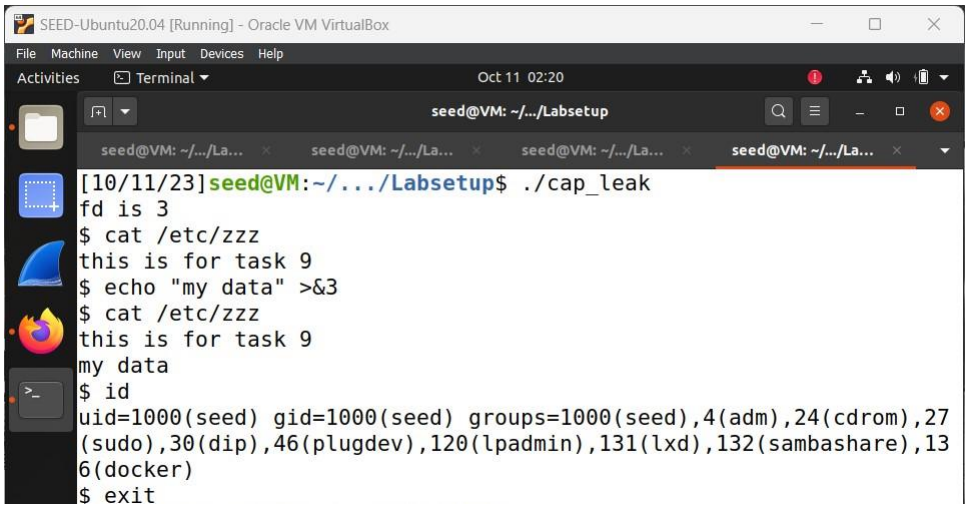
TASK 9:

First we create /etc/zzz as there is no such file

We compile the cap_leak.c file and set the effective user as root and also set it as SETUID program.

When tried to add to file permission was denied.





The program is a root user and is a SETUID program and can write etc/zzz file. After executing setuid() there is permission leak and it still has root permissions.