Alec Broughal
September 29, 2023
CIS 518

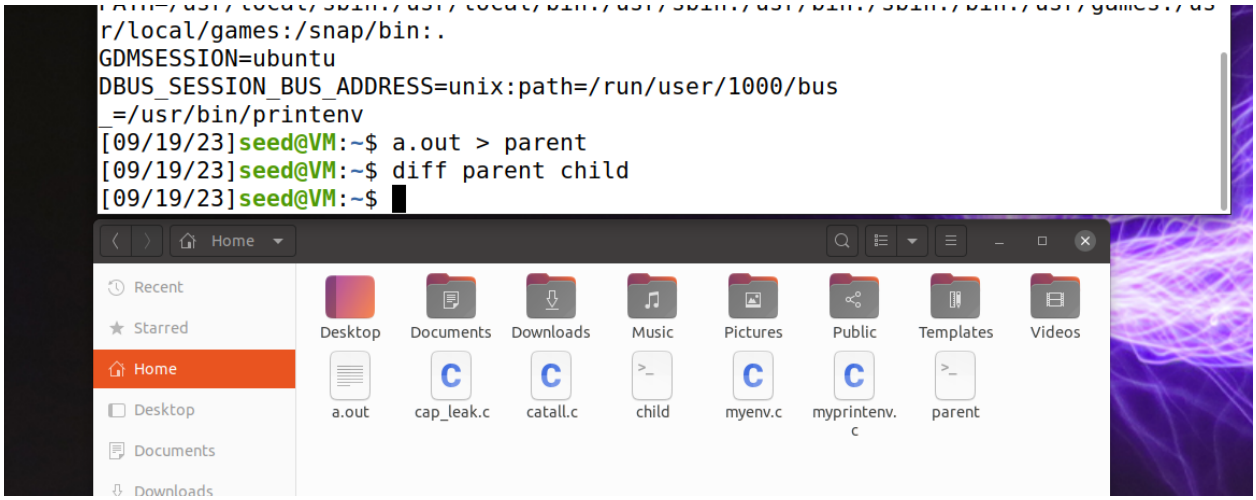
Lab 1 Report

Task 1 - We found that using printenv prints out the environment variables, and we were able to use ‘export’ to create an environment variable called ‘foo’ and then we were able to use ‘unset’ to delete the ‘foo’ environment variable. The first image shows what ‘printenv’ shows us, and the second image shows us using ‘export’ and ‘unset’ successfully.

```
[09/19/23]seed@VM:~$ printenv
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2087,unix/VM:/tmp/.ICE-unix/2087
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=2050
GTK_MODULES=gail:atk-bridge
PWD=/home/seed
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
```

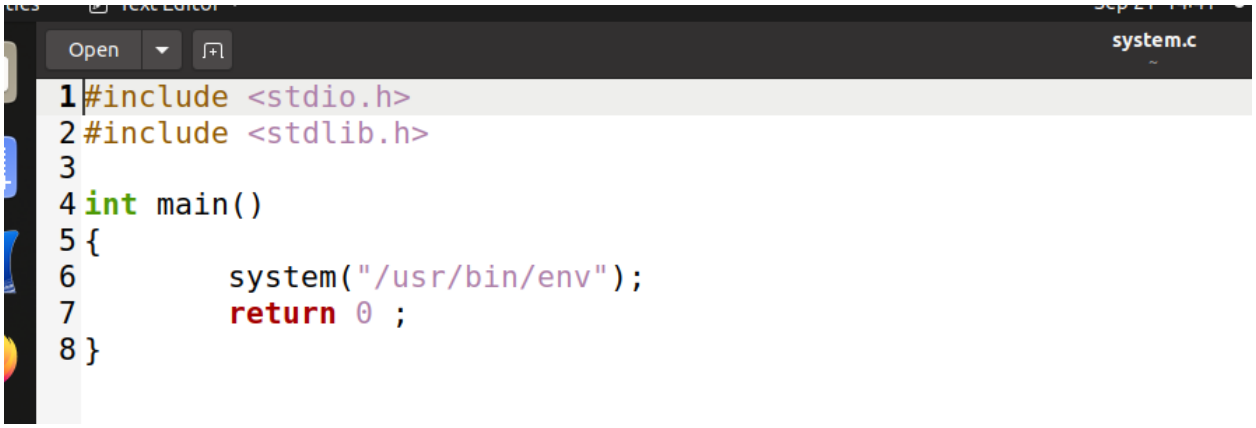
```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
r/local/games:/snap/bin:.
GDMSESSION=ubuntu
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
_=/usr/bin/printenv
[09/19/23]seed@VM:~$ printenv pwd
[09/19/23]seed@VM:~$ printenv PWD
/home/seed
[09/19/23]seed@VM:~$ export foo='test string'
[09/19/23]seed@VM:~$ printenv foo
test string
[09/19/23]seed@VM:~$ unset foo
[09/19/23]seed@VM:~$ printenv foo
[09/19/23]seed@VM:~$
```

Task 2 - We found that fork created a child process and inherits all environment variables from the existing parent process. It branches out or “forks.”The image below shows us proving that by using fork, all environment variables are inherited, which we did in the terminal with the ‘diff’ command, confirming that the child inherited the environment variables from the parent process. We can see the child and parent files underneath the Terminal.

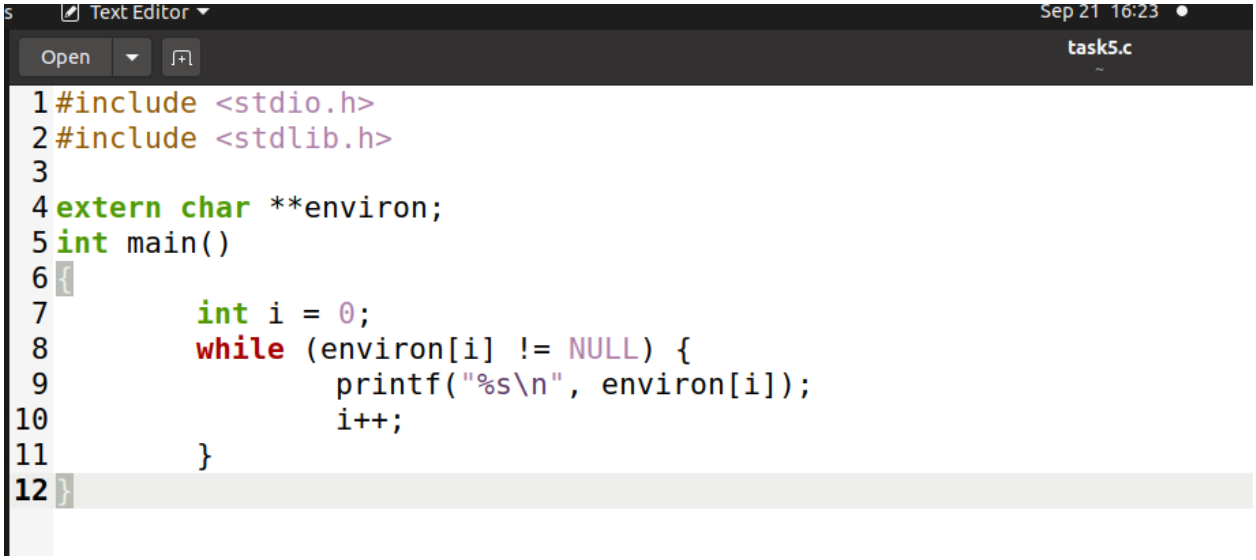


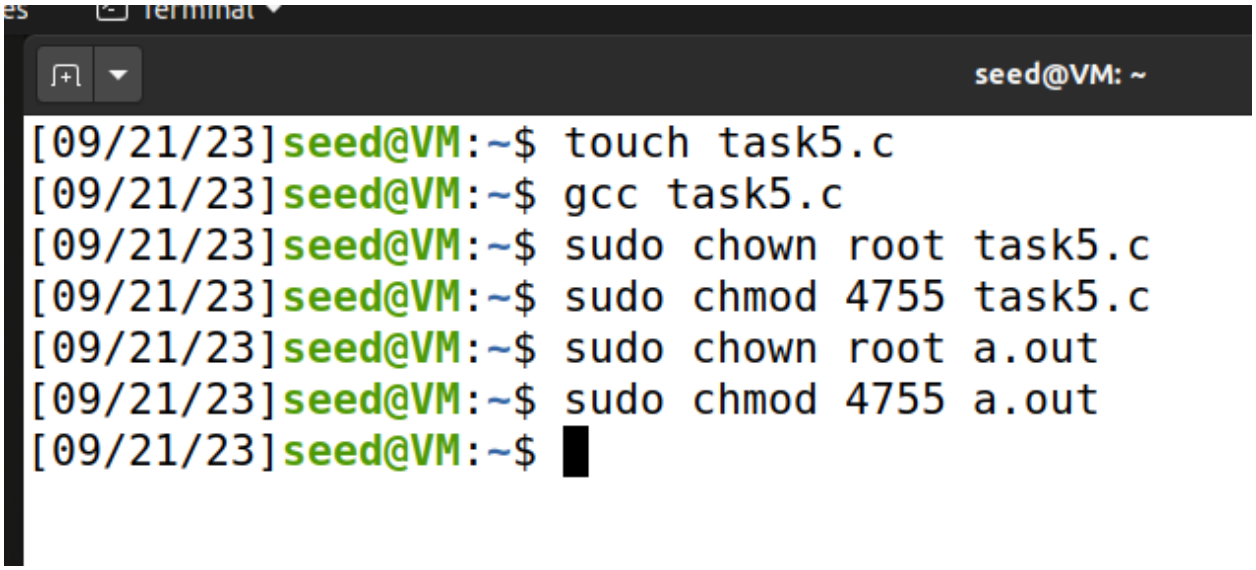
Task 3 - The new program gets environment variables because in myenv.c we replaced ‘NULL’ with ‘environ.’ NULL skips over inheriting environment variables and by changing it with environ, we can get whatever environment variables are available. With execve, we have the option to inherit environment variables or not, whereas fork does not give you the option. (image below is of myenv.c after we changed NULL to environ on line 12)

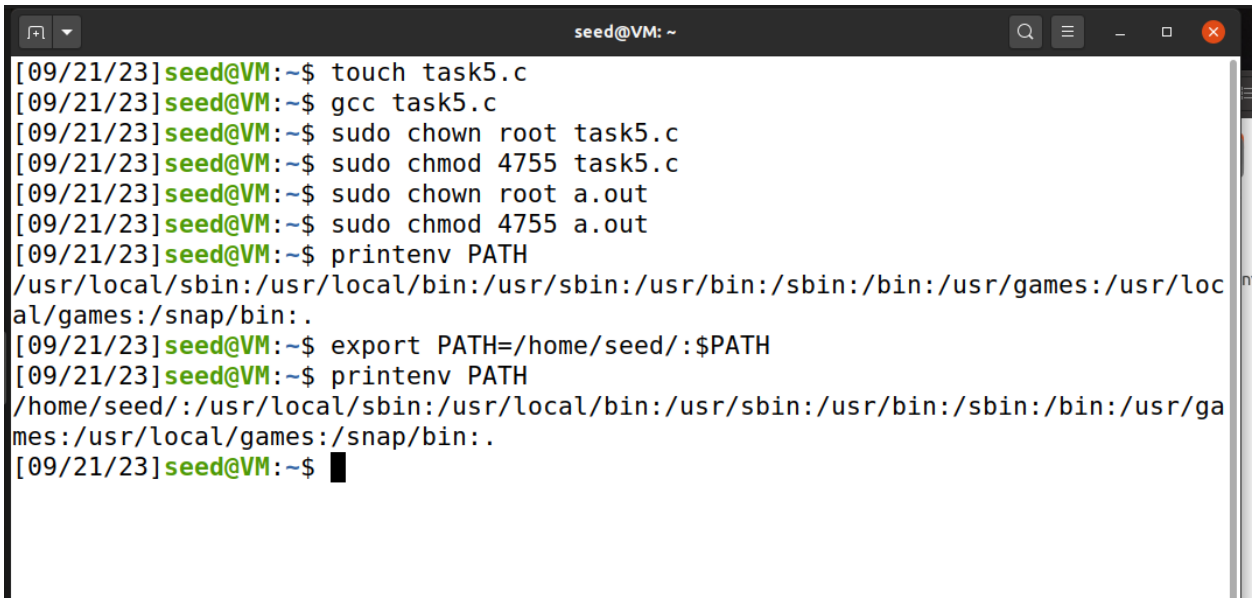
Task 4 - In the terminal we entered the command ‘touch system.c’ which then created a file called system.c and then we edited the file and added the requested 8 lines of code for it to become an executable, with the first image below showing system.c with the newly added code. Calling system () opens a new terminal and the new process takes place inside of that terminal.

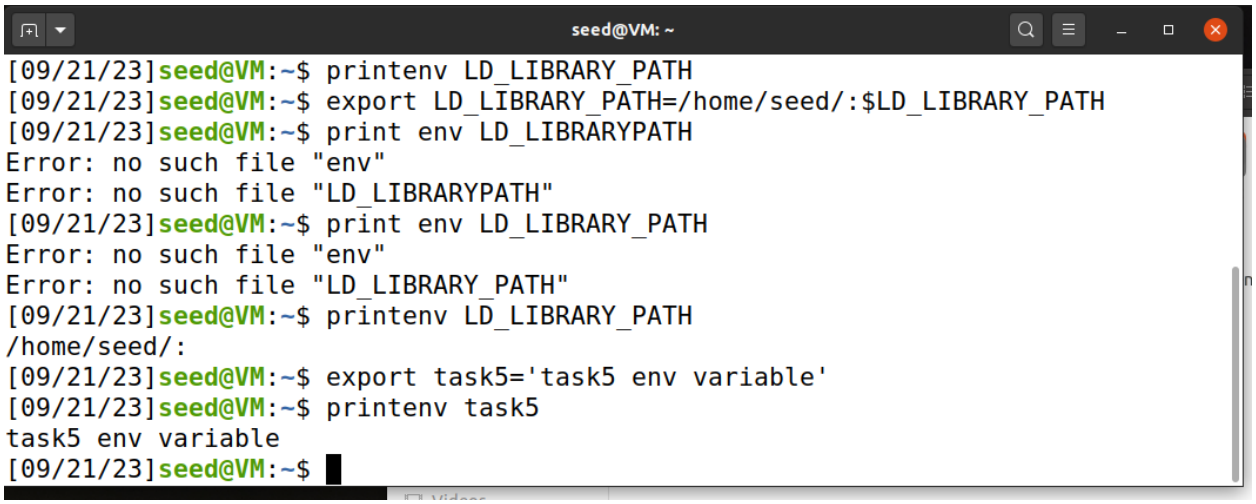


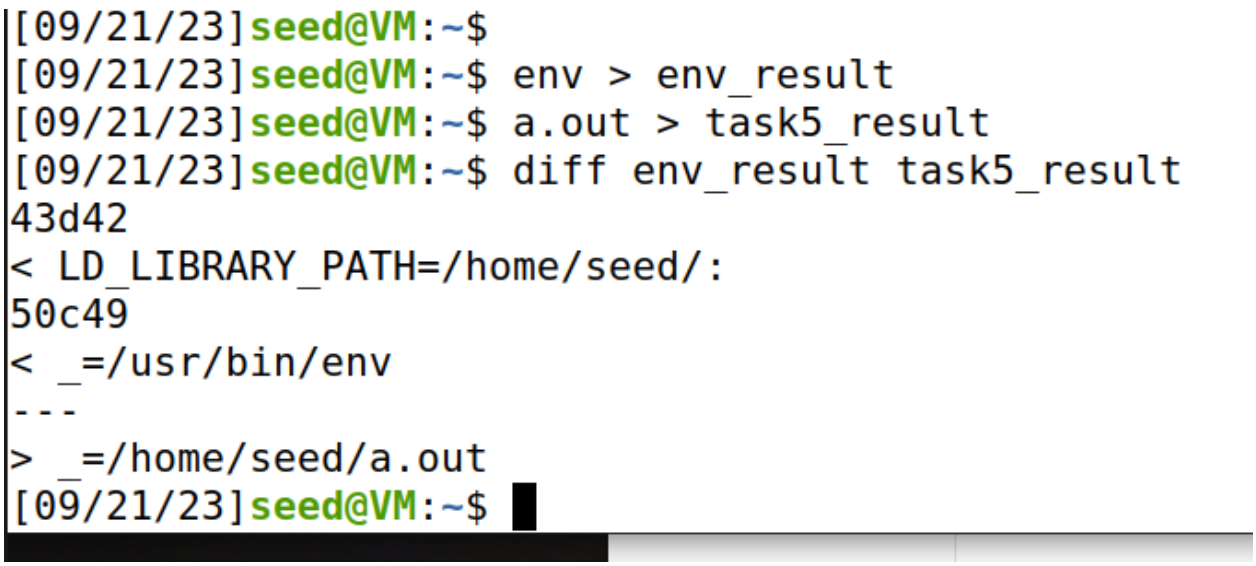
Task 5 - We created a program called task5.c, which is the first image below. We then compiled it with the command ‘gcc task5.c’ and then ‘sudo chown root task5.c’ and then ‘sudo chmod 4755 task5.c’ and then ’sudo chown root a.out’ and then ‘sudo chmod root a.out’ which is seen in the second screenshot below. In the next line of codes we set the environment variable path, and we’ll have home seed in front of it, and doing a second ‘printenv’ shows us that addition, which can be seen in the third screenshot below. Ultimately there is a patch that has not allowed the “printenv LD_LIBRARY_PATH” to display anything. Then we exported the library path to a new location as seen in the fourth image below, and doing “printenv LD_LIBRARY_PATH” now shows us the home/seed. We then added ‘task5 env variable’ to task5.c and did printenv to view that, also seen in the fourth image. We store all environment variables in env_result, which we create in the fifth screenshot below. We then save the environment variables of a.out to task5_result, and we can see that when comparing env_result and task5_result, we can see that not everything has been inherited(also in fifth image below). When opening each file to compare, we can see that the LD_LIBRARY_PATH environment variable was not inherited, because the dynamic link loader doesn’t inherit, but it protects some environment variables.


Text Editor
Sep 21 16:23
Open
task5.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 extern char **environ;
5 int main()
6 {
7 int i = 0;
8 while (environ[i] != NULL) {
9 printf("%s\n", environ[i]);
10 i++;
11 }
12 }


Terminal
seed@VM: ~
[09/21/23]seed@VM:~$ touch task5.c
[09/21/23]seed@VM:~$ gcc task5.c
[09/21/23]seed@VM:~$ sudo chown root task5.c
[09/21/23]seed@VM:~$ sudo chmod 4755 task5.c
[09/21/23]seed@VM:~$ sudo chown root a.out
[09/21/23]seed@VM:~$ sudo chmod 4755 a.out
[09/21/23]seed@VM:~$

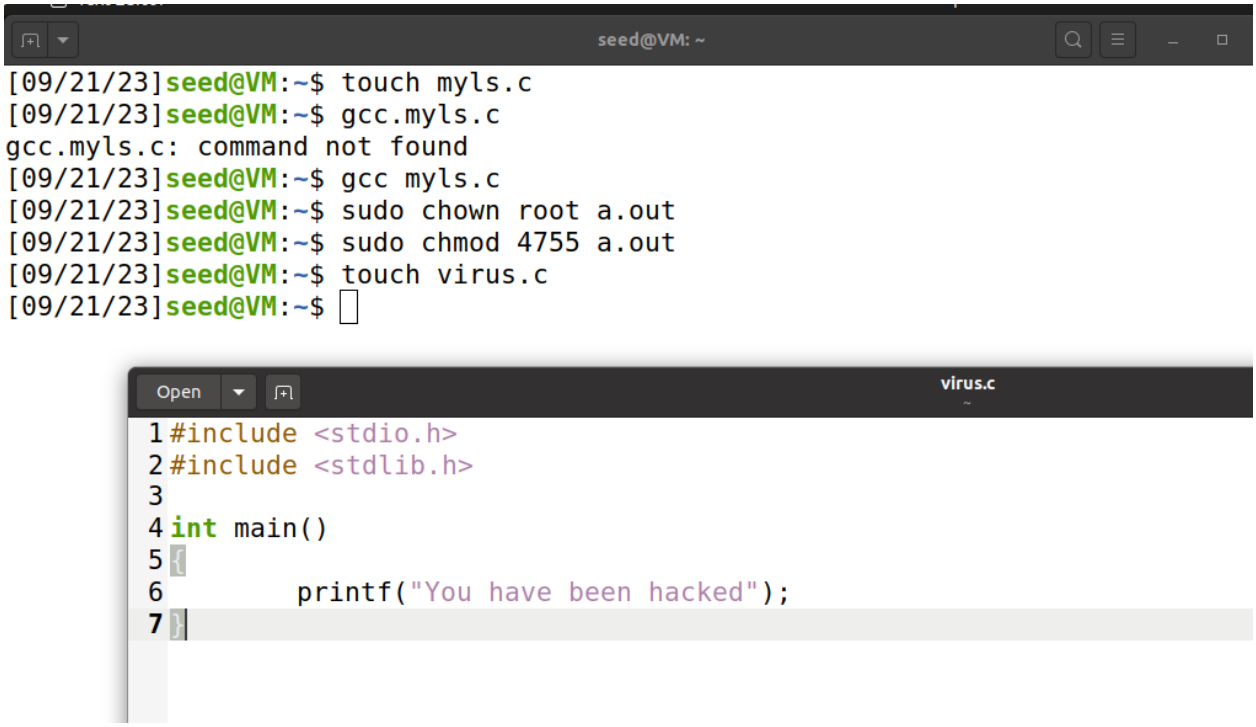

seed@VM: ~
[09/21/23]seed@VM:~$ touch task5.c
[09/21/23]seed@VM:~$ gcc task5.c
[09/21/23]seed@VM:~$ sudo chown root task5.c
[09/21/23]seed@VM:~$ sudo chmod 4755 task5.c
[09/21/23]seed@VM:~$ sudo chown root a.out
[09/21/23]seed@VM:~$ sudo chmod 4755 a.out
[09/21/23]seed@VM:~$ printenv PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
[09/21/23]seed@VM:~$ export PATH=/home/seed/:$PATH
[09/21/23]seed@VM:~$ printenv PATH
/home/seed/:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
[09/21/23]seed@VM:~$


seed@VM: ~
[09/21/23]seed@VM:~$ printenv LD_LIBRARY_PATH
[09/21/23]seed@VM:~$ export LD_LIBRARY_PATH=/home/seed/:$LD_LIBRARY_PATH
[09/21/23]seed@VM:~$ print env LD_LIBRARYPATH
Error: no such file "env"
Error: no such file "LD_LIBRARYPATH"
[09/21/23]seed@VM:~$ print env LD_LIBRARY_PATH
Error: no such file "env"
Error: no such file "LD_LIBRARY_PATH"
[09/21/23]seed@VM:~$ printenv LD_LIBRARY_PATH
/home/seed/:
[09/21/23]seed@VM:~$ export task5='task5 env variable'
[09/21/23]seed@VM:~$ printenv task5
task5 env variable
[09/21/23]seed@VM:~$

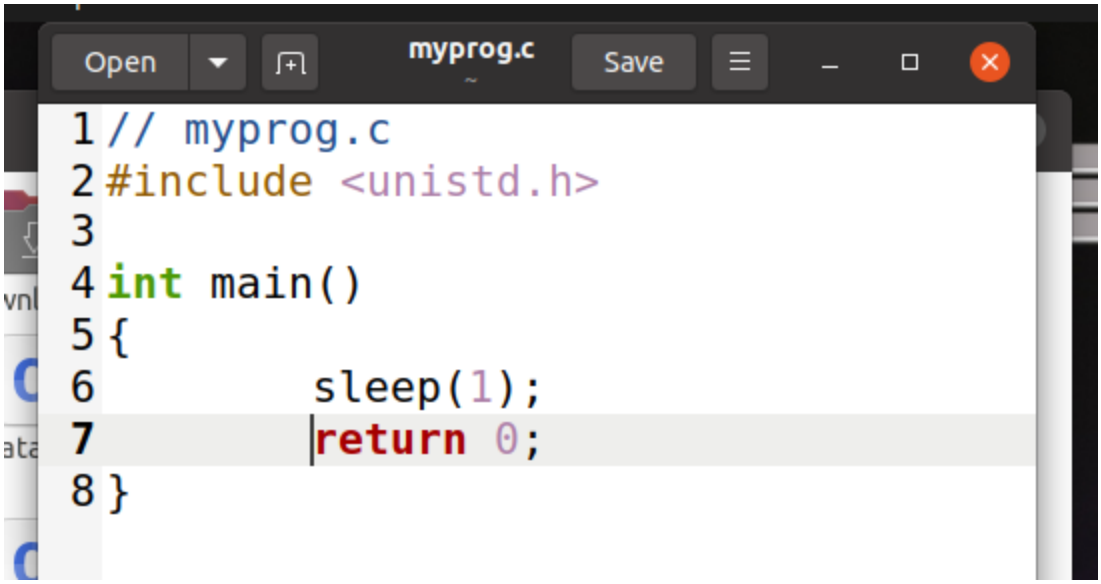

[09/21/23]seed@VM:~$
[09/21/23]seed@VM:~$ env > env_result
[09/21/23]seed@VM:~$ a.out > task5_result
[09/21/23]seed@VM:~$ diff env_result task5_result
43d42
< LD_LIBRARY_PATH=/home/seed/:
50c49
< _=/usr/bin/env

> _=/home/seed/a.out
[09/21/23]seed@VM:~$

Task 6 - We created a program called myls.c and ran ‘sudo chown root a.out’ and ‘sudo chmod 4755 a.out’ and also created a program called virus.c, as seen in the first screenshot below. We can then rename virus.c with ‘gcc virus.c > ls’ to try and get our malicious code to run instead, and if you type ‘a.out’ it displays our hack message from virus.c. The SetUID program myls.c calls ls, and we have successfully hacked it to run our virus.c instead of the actual ls. We changed where it points so that a.out doesn’t actually display the ls for myls.c, it displays virus.c, as seen in the second image below.



Task 7 - The screenshot below shows myprog.c as instructed to compile.



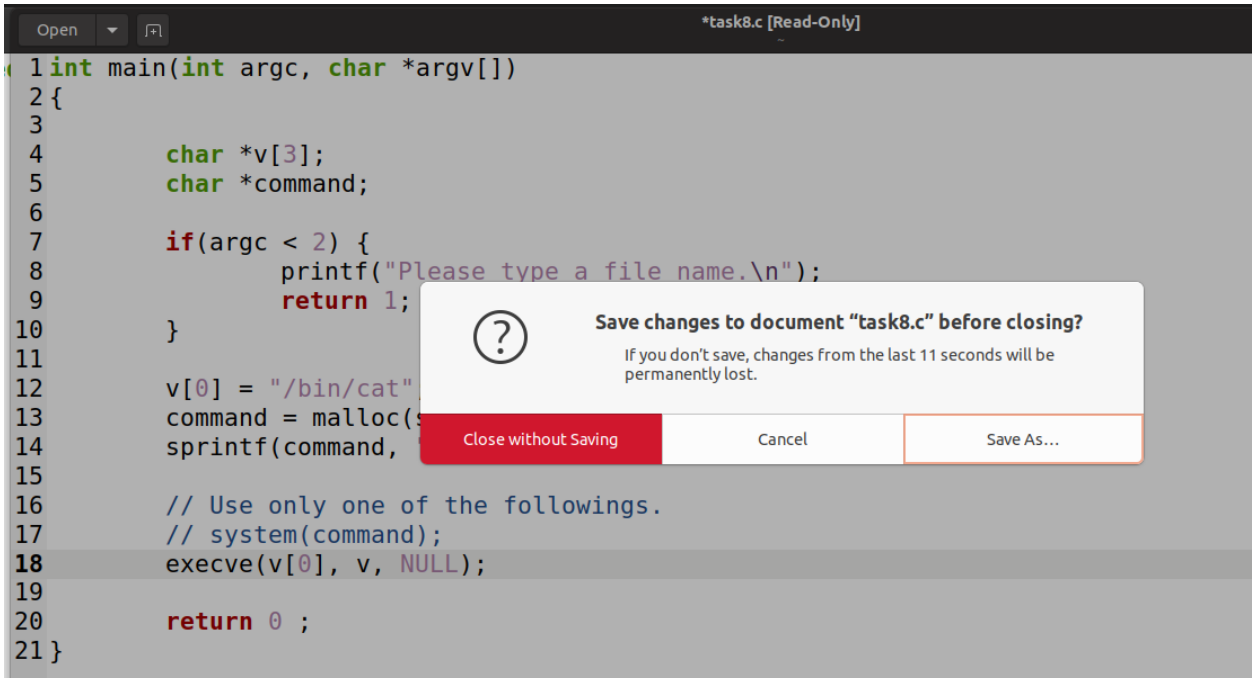
Task 8 - The first screenshot below is of task8.c, the program that it had us compile. If you ran the program with elevated privileges, then that could give Bob the ability to write, rather than it being read-only, which would yes compromise the integrity of the system. The second screenshot shows us setting the Set-UID bit to allow the program to run with root permissions. The third image below is from Step 2 of this task, and I ran into the error seen in the image when trying to comment out the // system (command); line.

Open task8.c [Read-Only]

```c
int main(int argc, char *argv[])
{

        char *v[3];
        char *command;

        if(argc < 2) {
                printf("Please type a file name.\n");
                return 1;
        }

        v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;
        command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
        sprintf(command, "%s %s", v[0], v[1]);

        // Use only one of the followings.
        system(command);
        // execve(v[0], v, NULL);

        return 0 ;
}
```

seed@VM: ~

```
3]seed@VM:~$ sudo chown root:root task8.c
3]seed@VM:~$ sudo chmod u+s task8.c
3]seed@VM:~$
```



Task 9 - In the first screenshot below you can see task9.c, which is what the Lab had us compile. Once following the requested steps, we got to the second screenshot where we were unable to run the program as normal user after making it a Set-UID program. So no, I could not exploit the capability leaking vulnerability in this program.

Open

task9.c

```c
void main()
{
        int fd;
        char *v[2];
        /* Assume that /etc/zzz is an important system file,
        * and it is owned by root with permission 0644.
        * Before running this program, you should create
        * the file /etc/zzz first. */
        fd = open("/etc/zzz", O_RDWR | O_APPEND);
        if (fd == -1) {
        printf("Cannot open /etc/zzz\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);
}

```

seed@VM: ~

```
[09/25/23]seed@VM:~$ sudo chown root task9.c
[09/25/23]seed@VM:~$ sudo chmod u+s task9.c
[09/25/23]seed@VM:~$ .\task9.c
.task9.c: command not found
[09/25/23]seed@VM:~$ .\task9.c
.task9.c: command not found
[09/25/23]seed@VM:~$
```