Jared Ren
1/28/2020
Lab report 1

2.2. Step 3

My conclusion is that the environment variables were successfully passed from parent to child.

2.2 What I learned: I learned that environment variables can be passed from parent processes to child processes.

2.3 step 1

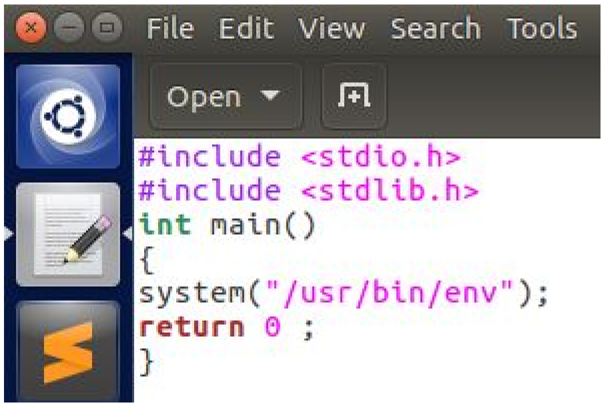
I received an implicit declaration warning on at compilation time on the C file. When ran nothing was shown in the terminal.

Step 2: NULL was changed to environ and this carried out the /usr/bin/env program that displays all environment variables of the process.

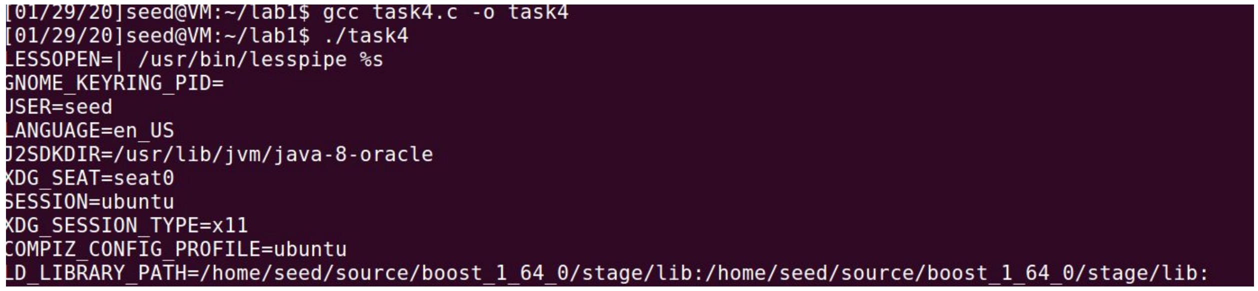
Step 3: the new program gets its environment variables by de-referencing the pointers in the passed array of pointers to environment variables.

2.3 What I learned: That environment variables can be accessed by programs not just processes. That execve passes an array of pointers to environment variables to the new program.

2.4 task4.c created



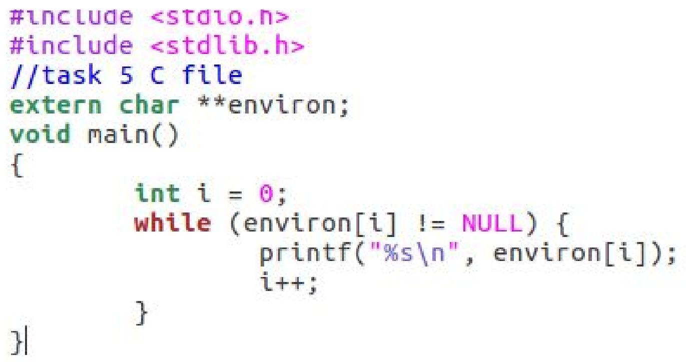
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
system("/usr/bin/env");
return 0 ;
}
```


```
[01/29/20]seed@VM:~/lab1$ gcc task4.c -o task4
[01/29/20]seed@VM:~/lab1$ ./task4
LESSOPEN=| /usr/bin/lesspipe %s
GNOME_KEYRING_PID=
USER=seed
LANGUAGE=en_US
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_SEAT=seat0
SESSION=ubuntu
XDG_SESSION_TYPE=x11
COMPIZ_CONFIG_PROFILE=ubuntu
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
```
Compiling and running the task4.c file

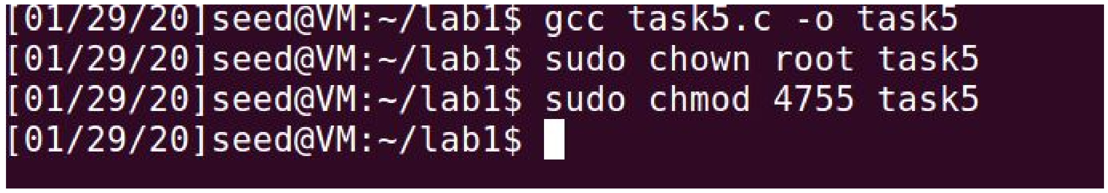
2.4 What I learned: I learned that using system() the current process executes the command and prompts the shell to execute it. System() indirectly calls execve which passes to the new program the array of pointers to environment variables.

2.5

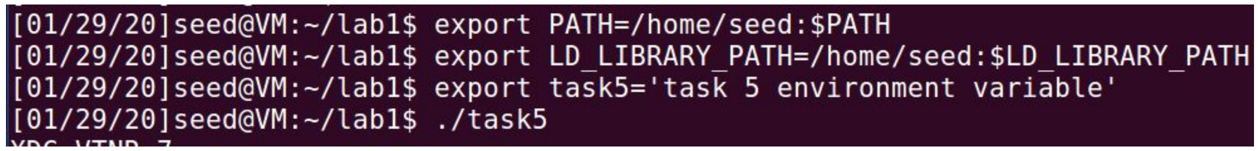
Step 1 create C file that prints out all the environment variables in the current process.


```c
#include <stdio.h>
#include <stdlib.h>
//task 5 C file
extern char **environ;
void main()
{
        int i = 0;
        while (environ[i] != NULL) {
                printf("%s\n", environ[i]);
                i++;
        }
}
```

Step 2: compile, change ownership and make set UID program.

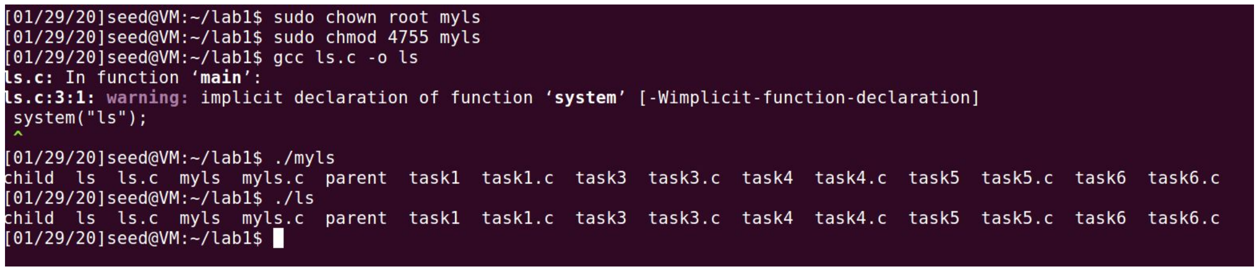

```
[01/29/20]seed@VM:~/lab1$ gcc task5.c -o task5
[01/29/20]seed@VM:~/lab1$ sudo chown root task5
[01/29/20]seed@VM:~/lab1$ sudo chmod 4755 task5
[01/29/20]seed@VM:~/lab1$ 
```

Step 3:


```
[01/29/20]seed@VM:~/lab1$ export PATH=/home/seed:$PATH
[01/29/20]seed@VM:~/lab1$ export LD_LIBRARY_PATH=/home/seed:$LD_LIBRARY_PATH
[01/29/20]seed@VM:~/lab1$ export task5='task 5 environment variable'
[01/29/20]seed@VM:~/lab1$ ./task5
```
Observation: the LD_LIBRARY_PATH environment variable was not listed in the environment variables. This means this environment variable wasn't passed from the parent shell process to the child set-UID process. My defined environment variable was included in the listing so it passed as well as the PATH environment variable.

2.5 What I learned: That certain environment variables don’t get passed from parent processes to child processes depending on the type of process.

2.6
My code is running with root privilege because it is a set-UID program



```
[01/29/20]seed@VM:~/lab1$ sudo chown root myls
[01/29/20]seed@VM:~/lab1$ sudo chmod 4755 myls
[01/29/20]seed@VM:~/lab1$ gcc ls.c -o ls
ls.c: In function ‘main’:
ls.c:3:1: warning: implicit declaration of function ‘system’ [-Wimplicit-function-declaration]
 system("ls");
 ^
[01/29/20]seed@VM:~/lab1$ ./myls
child  ls  ls.c  myls  myls.c  parent  task1  task1.c  task3  task3.c  task4  task4.c  task5  task5.c  task6  task6.c
[01/29/20]seed@VM:~/lab1$ ./ls
child  ls  ls.c  myls  myls.c  parent  task1  task1.c  task3  task3.c  task4  task4.c  task5  task5.c  task6  task6.c
[01/29/20]seed@VM:~/lab1$ 
```

What I learned: an environment variable can be changed by users and affect the behavior of a program, executing different code.

2.7
What I’ve learned: That there are certain situations when a user can use an environment variable to override a system function.

2.8

Step 1: Bob can compromise the integrity of the file system. When executing the task8 file after making it a root owned set-UID program I was able to delete a file from the lab directory. When I exited the terminal and went back in to the directory the file was removed from the listing.

Step 2: The attacks in step 1 do not work. Commenting out system in the program made it unable for Bob to do the things he did in the first step.

What I learned: when making a set-UID make one that doesn’t call system in it. The execve function doesn’t invoke the shell and doesn’t open the door for Bob to delete a file.