

Project 1. Environment Variable and Set-UID Program Lab

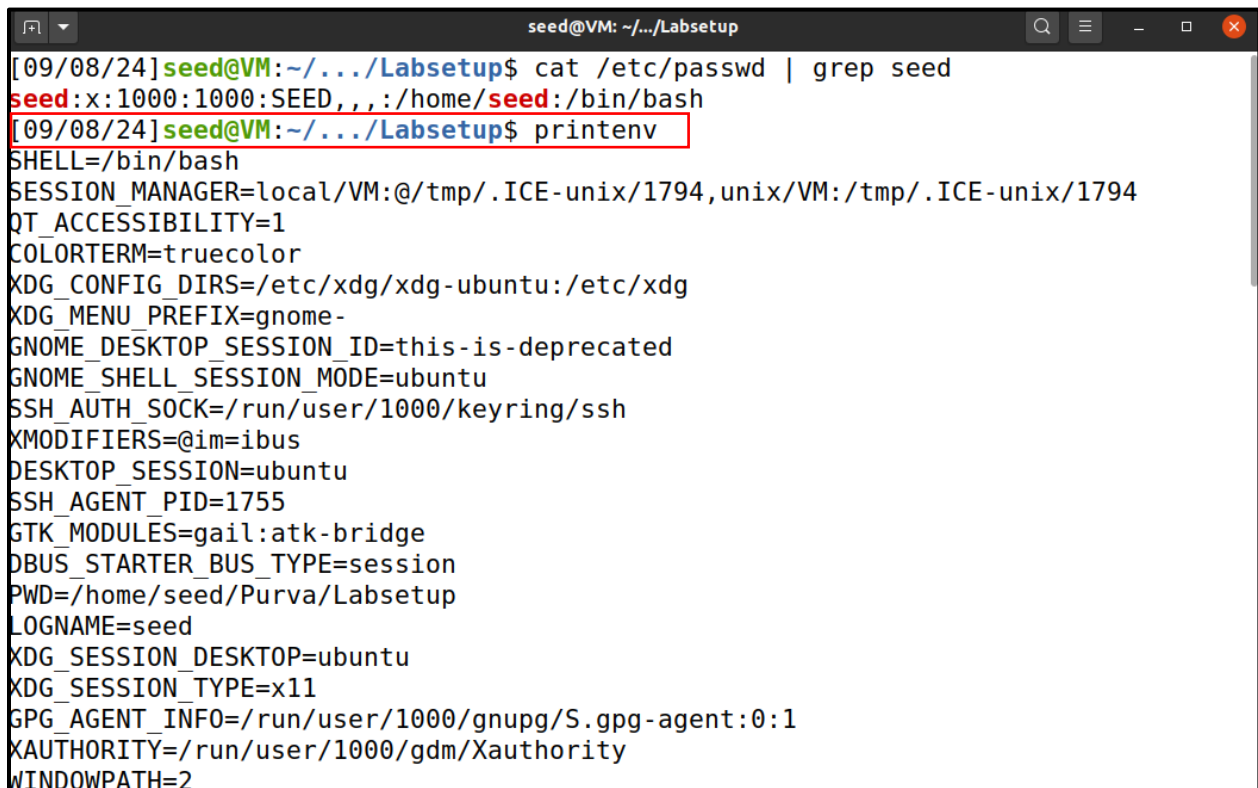
Purva Naresh Rumde

pr23b

Task 1: Manipulating Environment Variables

In this task we tried the `printenv`, `env`, `printenv PWD`, `env | grep PWD`, `export MYVAR` and `unset MYVAR`. This involves setting and unsetting environment variables in the Bash shell and observing how these commands behave.

`printenv`: This command prints all the environment variables and their values that are currently set in your shell session. Each environment variable is printed on a new line.

A terminal window titled 'seed@VM: ~/.../Labsetup' showing the output of the 'printenv' command. The output lists various environment variables and their values, such as SHELL=/bin/bash, SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1794,unix/VM:/tmp/.ICE-unix/1794, QT_ACCESSIBILITY=1, and PWD=/home/seed/Purva/Labsetup. The command prompt and the 'printenv' command itself are highlighted with a red box.

```
[09/08/24]seed@VM:~/.../Labsetup$ cat /etc/passwd | grep seed
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
[09/08/24]seed@VM:~/.../Labsetup$ printenv
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1794,unix/VM:/tmp/.ICE-unix/1794
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=1755
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Purva/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
WINDOWPATH=2
```

```
seed@VM: ~/.../Labsetup
INVOCATION_ID=be0c5ffb91c3453baebcedbdc0170168
MANAGERPID=1562
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
GNOME_TERMINAL_SERVICE=:1.94
DISPLAY=:0
SHLVL=1
QT_IM_MODULE=ibus
DBUS_STARTER_ADDRESS=unix:path=/run/user/1000/bus,guid=c4ae91f5cbf8488a81da649566de1a5f
XDG_RUNTIME_DIR=/run/user/1000
JOURNAL_STREAM=9:32224
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=c4ae91f5cbf8488a81da649566de1a5f
=/usr/bin/printenv
[09/08/24] seed@VM: ~/.../Labsetup$
```

env: Similar to printenv, the env command displays the environment variables and their values. However, env also allows you to run a command with modified environment variables, while printenv is primarily for printing them.

```
seed@VM: ~/.../Labsetup
~/usr/bin/printenv
[09/08/24] seed@VM: ~/.../Labsetup$ env
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1794,unix/VM:/tmp/.ICE-unix/1794
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=1755
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Purva/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
WINDOWPATH=2
HOME=/home/seed
USERNAME=seed
```

```
seed@VM: ~/.../Labsetup
INVOCATION_ID=be0c5ffb91c3453baebcedbdc0170168
MANAGERPID=1562
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
GNOME_TERMINAL_SERVICE=:1.94
DISPLAY=:0
SHLVL=1
QT_IM_MODULE=ibus
DBUS_STARTER_ADDRESS=unix:path=/run/user/1000/bus,guid=c4ae91f5cbf8488a81da649566de1a5f
XDG_RUNTIME_DIR=/run/user/1000
JOURNAL_STREAM=9:32224
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=c4ae91f5cbf8488a81da649566de1a5f
=/usr/bin/env
[09/08/24] seed@VM: ~/.../Labsetup$
```

printenv PWD: This specific usage of printenv only prints the value of the PWD environment variable, which represents the "Present Working Directory" of the shell session.

env | grep PWD: This command uses a **pipe** (|) to pass the output of the env command to the grep command. env lists all the environment variables. grep PWD filters this list to only show lines containing the string "PWD". Essentially, it finds and displays the PWD variable.

```
seed@VM: ~/.../Labsetup
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
GNOME_TERMINAL_SERVICE=:1.94
DISPLAY=:0
SHLVL=1
QT_IM_MODULE=ibus
DBUS_STARTER_ADDRESS=unix:path=/run/user/1000/bus,guid=c4ae91f5cbf8488a81da649566de1a5f
XDG_RUNTIME_DIR=/run/user/1000
JOURNAL_STREAM=9:32224
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=c4ae91f5cbf8488a81da649566de1a5f
=/usr/bin/env
[09/08/24] seed@VM: ~/.../Labsetup$ printenv PWD
/home/seed/Purva/Labsetup
[09/08/24] seed@VM: ~/.../Labsetup$ env | grep PWD
PWD=/home/seed/Purva/Labsetup
[09/08/24] seed@VM: ~/.../Labsetup$
```

`export MYVAR`: This command sets a new environment variable called `MYVAR`. The variable will persist in the current shell session and can be accessed by any subprocesses started from that session. However, `MYVAR` needs a value, so you'd usually write:

- `export MYVAR=purva cs`

`unset MYVAR`: This command removes the environment variable `MYVAR` from the environment. After running `unset MYVAR`, the variable will no longer exist or be accessible in the shell session.

```
[09/08/24] seed@VM:~/.../Labsetup$ export MYVAR='purva cs'
[09/08/24] seed@VM:~/.../Labsetup$ printenv MYVAR
purva cs
[09/08/24] seed@VM:~/.../Labsetup$ env | grep MYVAR
MYVAR=purva cs
[09/08/24] seed@VM:~/.../Labsetup$ unset MYVAR
[09/08/24] seed@VM:~/.../Labsetup$ env | grep MYVAR
[09/08/24] seed@VM:~/.../Labsetup$
```

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

This task involves examining whether a parent process passes its environment variables to its child process.

Step 1:

First Run:

This program will:

- Fork into a child process.
- The child will execute `printenv` and display all its environment variables.
- The parent will wait for the child to complete and then also run `printenv`.

Expected Output: Both the child and parent processes should have identical environment variables, as the child inherits the environment from the parent by default in most Unix-based systems. Therefore, both `printenv` calls will print the same set of environment variables.

```
seed@VM: ~/.../Labsetup
[09/08/24]seed@VM:~/.../Labsetup$ gcc myprintenv.c -o child
[09/08/24]seed@VM:~/.../Labsetup$ ./child
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1794,unix/VM:/tmp/.ICE-unix/1794
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=1755
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Purva/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
WINDOWPATH=2
```

Step 2:

Commenting out the `printenv()` call in the child process and execute it in the parent process instead.

Second Run:

In this modified version, only the parent process runs `printenv`, so the environment variables are printed once.

Expected Output: Since the child process does not print its environment, you will only see the environment variables once, from the parent. The output should still be the same as before because the child inherits the same environment, even if it doesn't print it.

```
Open  myprintenv.c
~/Purva/Labsetup

1 #include <unistd.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 extern char **environ;
6
7 void printenv()
8 {
9     int i = 0;
10    while (environ[i] != NULL) {
11        printf("%s\n", environ[i]);
12        i++;
13    }
14 }
15
16 void main()
17 {
18     pid_t childPid;
19     switch(childPid = fork()) {
20         case 0: /* child process */
21             //printenv();
22             exit(0);
23         default: /* parent process */
24             printenv();
25             exit(0);
26     }
27 }
```

```
seed@VM: ~/.../Labsetup

= ./child
[09/08/24] seed@VM: ~/.../Labsetup$ gcc myprintenv.c -o parent
[09/08/24] seed@VM: ~/.../Labsetup$ ./parent

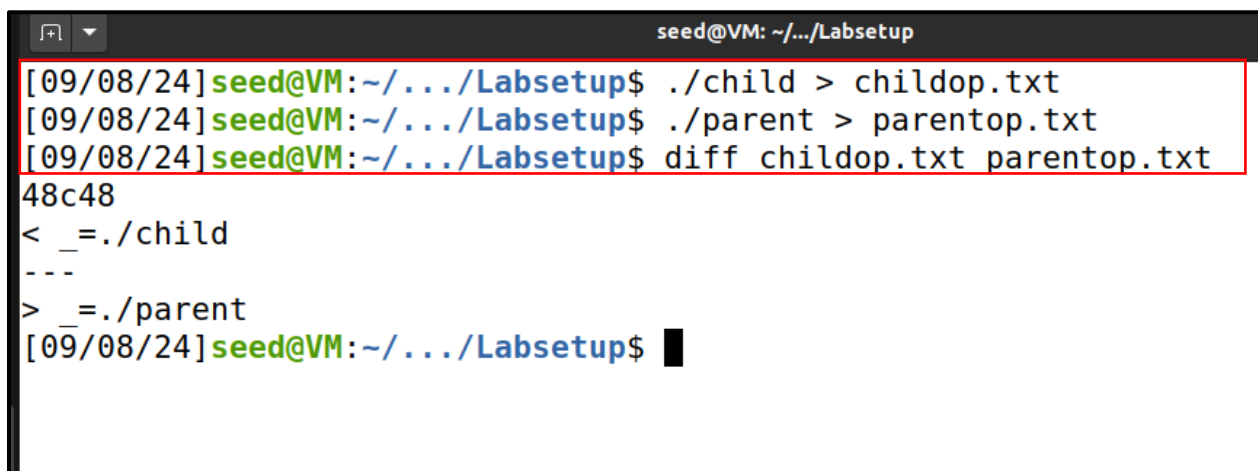
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1794,unix/VM:/tmp/.ICE-unix/1794
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=1755
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Purva/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
WINDOWPATH=2
```

Step 3: Comparison

Comparison:

- **First Run:** Both the parent and child print the same environment variables.
- **Second Run:** Only the parent prints the environment variables, but they are identical to what the child would have printed in the first run because the child process inherits the environment from the parent.

The key point is that in a typical Unix-like operating system, child processes inherit the environment of the parent process unless explicitly modified. Therefore, the outputs would be identical in both cases.

A terminal window titled 'seed@VM: ~/.../Labsetup' showing a series of commands and their outputs. The first three lines are highlighted with a red box: '[09/08/24] seed@VM:~/.../Labsetup\$./child > childop.txt', '[09/08/24] seed@VM:~/.../Labsetup\$./parent > parentop.txt', and '[09/08/24] seed@VM:~/.../Labsetup\$ diff childop.txt parentop.txt'. The output of the diff command is shown below: '48c48', '< _=./child', '---', '> _=./parent', and '[09/08/24] seed@VM:~/.../Labsetup\$' followed by a cursor.

```
seed@VM: ~/.../Labsetup
[09/08/24] seed@VM:~/.../Labsetup$ ./child > childop.txt
[09/08/24] seed@VM:~/.../Labsetup$ ./parent > parentop.txt
[09/08/24] seed@VM:~/.../Labsetup$ diff childop.txt parentop.txt
48c48
< _=./child
---
> _=./parent
[09/08/24] seed@VM:~/.../Labsetup$
```

Task 3: Environment Variables and execve()

This task looks at how environment variables are affected when a program is executed using the `execve()` function.

Step 1: When I ran the `./nullenv`, since `NULL` is passed for the environment, the `env` command will print **nothing** (i.e., no environment variables will be set). This happens because `execve()` replaces the current process image with a new one, and passing `NULL` means the new process doesn't get any environment variables from the parent.

Step 2: `./environ`

In this case, the `execve()` function passes the parent's environment (`environ`) to the new program. As a result, when the `env` program is executed, it will print all the environment variables inherited from the parent.

Step 3: Comparison

Comparison:

- **With NULL:** The environment of the new process is empty, so the env program prints nothing.
- **With environ:** The environment of the new process is inherited from the parent, and the env program prints all the inherited environment variables.

```
seed@VM: ~/.../Labsetup
[09/08/24]seed@VM:~/.../Labsetup$ gcc myenv.c -o nullenv
[09/08/24]seed@VM:~/.../Labsetup$ ./nullenv
[09/08/24]seed@VM:~/.../Labsetup$ gcc myenv.c -o environ
[09/08/24]seed@VM:~/.../Labsetup$ ./environ
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1794,unix/VM:/tmp/.ICE-unix/1794
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=1755
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Purva/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
```

```
myenv.c
~/Purva/Labsetup
myprintenv.c
1#include <unistd.h>
2
3extern char **environ;
4
5int main()
6{
7    char *argv[2];
8
9    argv[0] = "/usr/bin/env";
10   argv[1] = NULL;
11
12   //execve("/usr/bin/env", argv, NULL);
13   execve("/usr/bin/env", argv, environ);
14
15   return 0 ;
16}
17
```


Task 4: Environment Variables and system()

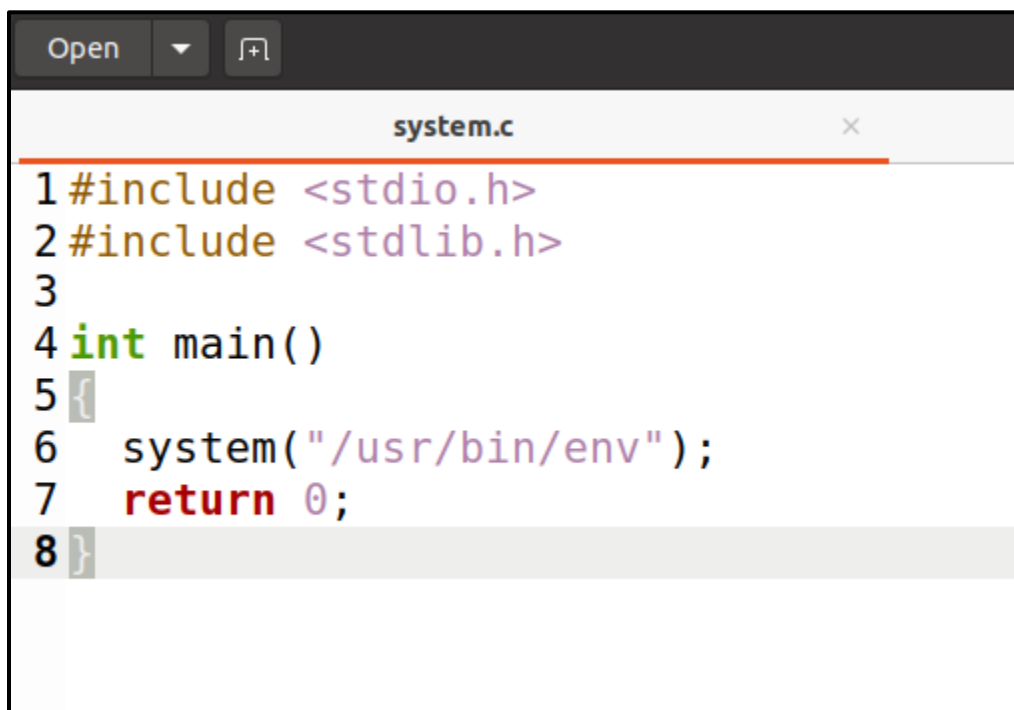
The goal is to investigate how environment variables are passed when using the `system()` function to execute a command.

Overview of the Task

1. **system() Function:** The `system()` function is a library function in C that invokes a shell to execute a command. This means that when you use `system("/usr/bin/env")`, the program calls the `/usr/bin/env` command through the shell.
2. **Purpose of the Task:** By running `/usr/bin/env`, we observe the environment variables that are passed to the shell invoked by the `system()` call. This task helps to understand what environment variables are available in the environment of the command that gets executed by `system()`.

You should see a list of all environment variables such as `PATH`, `HOME`, `USER`. The variables listed are those that are passed to the process started by `system()`, which are inherited from the current process. This helps confirm that the `system()` call inherits the environment from the parent process unless it is explicitly modified.

The code provided to check the execution of `system()`.

A screenshot of a code editor window titled 'system.c'. The editor contains C code that includes `<stdio.h>` and `<stdlib.h>`, and defines a `main()` function that calls `system("/usr/bin/env");` before returning 0. The code is syntax-highlighted: keywords like `int`, `return`, and `main` are in green; string literals are in purple; and control flow keywords like `if` and `else` are in red. The code is as follows:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     system("/usr/bin/env");
7     return 0;
8 }
```

```
seed@VM: ~/.../Labsetup
[09/08/24] seed@VM: ~/.../Labsetup$ ls
cap_leak.c  child  environ  myprintenv.c  parent  system.c
catall.c   childop.txt  myenv.c  nullenv  parentop.txt
[09/08/24] seed@VM: ~/.../Labsetup$ gcc system.c -o sys
[09/08/24] seed@VM: ~/.../Labsetup$ ./sys
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SSH_AGENT_PID=1755
XDG_SESSION_TYPE=x11
SHLVL=1
HOME=/home/seed
DESKTOP_SESSION=ubuntu
GNOME_SHELL_SESSION_MODE=ubuntu
GTK_MODULES=gail:atk-bridge
MANAGERPID=1562
DBUS_STARTER_BUS_TYPE=session
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus,guid=c4ae91f5cbf8488a81da649566de1a5f
COLORTERM=truecolor
IM_CONFIG_PHASE=1
LOGNAME=seed
JOURNAL_STREAM=9:32224
=./sys
XDG_SESSION_CLASS=user
```

```
seed@VM: ~/.../Labsetup
p4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
GNOME_TERMINAL_SERVICE=:1.94
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
SHELL=/bin/bash
QT_ACCESSIBILITY=1
GDMSESSION=ubuntu
LESSCLOSE=/usr/bin/lesspipe %s %s
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
QT_IM_MODULE=ibus
PWD=/home/seed/Purva/Labsetup
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
DBUS_STARTER_ADDRESS=unix:path=/run/user/1000/bus,guid=c4ae91f5cbf8488a81da649566de1a5f
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share/:/var/lib/snapd/desktop
VTE_VERSION=6003
[09/08/24] seed@VM: ~/.../Labsetup$ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 Nov 24 2020 /bin/sh -> dash
[09/08/24] seed@VM: ~/.../Labsetup$
```

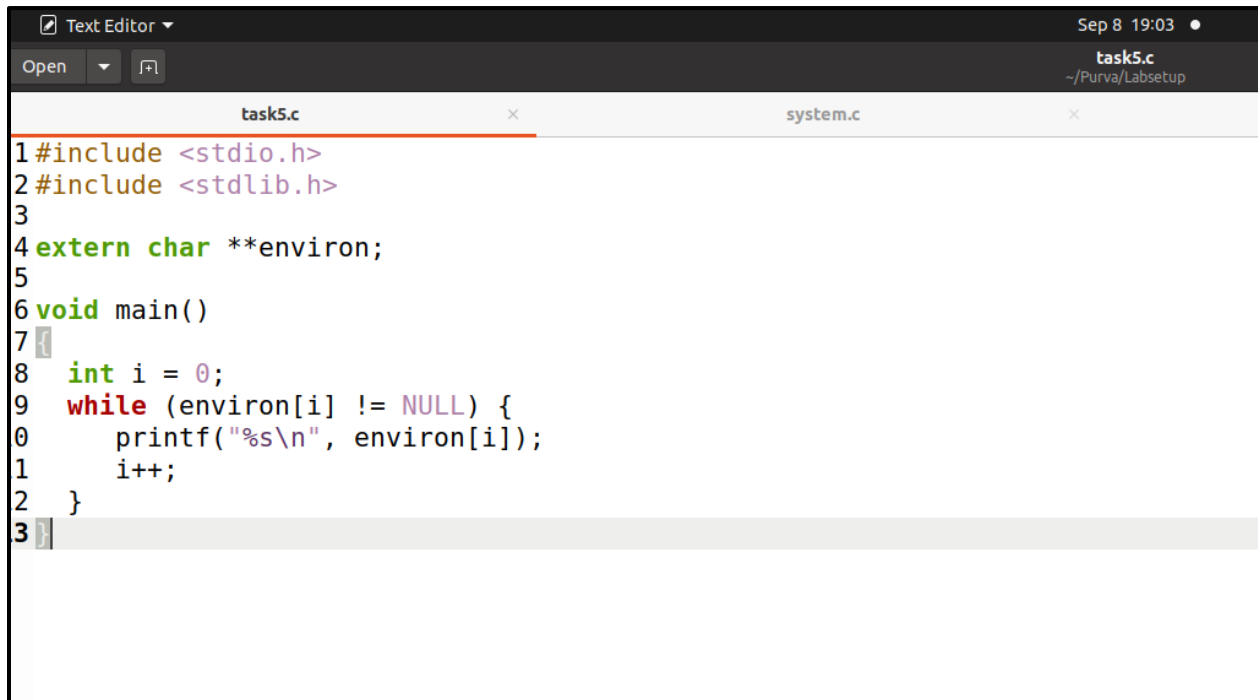
Task 5: Environment Variable and Set-UID Programs

The objective is to explore how environment variables interact with **Set-UID (Set User ID)** programs, which are programs that run with the privileges of the file owner, rather than the user executing the program.

First we write the program provided in the lab file.

Compile and change the permissions of the file and change its owner to root.

Now we execute the file, we see the basic desired output.

A screenshot of a text editor window titled "Text Editor" with a dark theme. The window shows two tabs: "task5.c" (active) and "system.c". The code in "task5.c" is as follows:

```
1#include <stdio.h>
2#include <stdlib.h>
3
4extern char **environ;
5
6void main()
7{
8    int i = 0;
9    while (environ[i] != NULL) {
10        printf("%s\n", environ[i]);
11        i++;
12    }
13}
```

The code is color-coded: preprocessor directives are purple, keywords are green, and identifiers/strings are purple. The window's title bar shows the date and time "Sep 8 19:03" and the file path "~/Purva/Labsetup".

```
seed@VM: ~/.../Labsetup
[09/08/24] seed@VM: ~/.../Labsetup$ gcc task5.c -o printenv
[09/08/24] seed@VM: ~/.../Labsetup$ ./printenv
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1794,unix/VM:/tmp/.ICE-unix/1794
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=1755
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Purva/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
WINDOWPATH=2
HOME=/home/seed
```

Now we execute the 3 commands.

1. PATH :

We execute the echo \$PATH command and the desired path is displayed.

2. LD_LIBRARY_PATH

echo \$LD_LIBRARY_PATH does not produce any output, it's likely because the LD_LIBRARY_PATH environment variable is either **not set** or is **empty**. This environment variable is used to specify additional directories where the dynamic linker should look for libraries.

So we set LD_LIBRARY_PATH = '.'

Now while printing all the environment variables we first export LD_LIBRARY_PATH and then use the command printenv LD_LIBRARY_PATH to print it.

3. MYVAR

We set the MYVAR with the command export MYVAR='purva cs'.

Print it using printenv MYVAR and get the result printed.

```
seed@VM: ~/.../Labsetup
[09/08/24]seed@VM:~/.../Labsetup$ sudo chmod 4755 printenv
[09/08/24]seed@VM:~/.../Labsetup$ ls -l printenv
-rwsr-xr-x 1 root seed 16768 Sep  8 19:02 printenv
[09/08/24]seed@VM:~/.../Labsetup$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
[09/08/24]seed@VM:~/.../Labsetup$ echo $LD_LIBRARY_PATH

[09/08/24]seed@VM:~/.../Labsetup$ export MYVAR='purva cs'
[09/08/24]seed@VM:~/.../Labsetup$ LD_LIBRARY_PATH='.'
[09/08/24]seed@VM:~/.../Labsetup$ printenv PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
[09/08/24]seed@VM:~/.../Labsetup$ printenv LD_LIBRARY_PATH
[09/08/24]seed@VM:~/.../Labsetup$ export LD_LIBRARY_PATH
[09/08/24]seed@VM:~/.../Labsetup$ printenv LD_LIBRARY_PATH
.
[09/08/24]seed@VM:~/.../Labsetup$ printenv MYVAR
purva cs
```

Now we check whether all the environment variables set in the parent process will be executed in the child process as well.

So here,

`./printenv | grep PATH` gets executed and produces the result.

`./printenv | grep MYVAR` get executed and prints the name that was set.

`./printenv | grep LD_LIBRARY_PATH` this command does not get executed as it can be because of Security Restrictions with Set-UID Programs.

```
[09/08/24]seed@VM:~/.../Labsetup$ ./printenv | grep PATH
WINDOWPATH=2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
[09/08/24]seed@VM:~/.../Labsetup$ ./printenv | grep MYVAR
MYVAR=purva cs
[09/08/24]seed@VM:~/.../Labsetup$ ./printenv | grep LD_LIBRARY_PATH
[09/08/24]seed@VM:~/.../Labsetup$
```

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

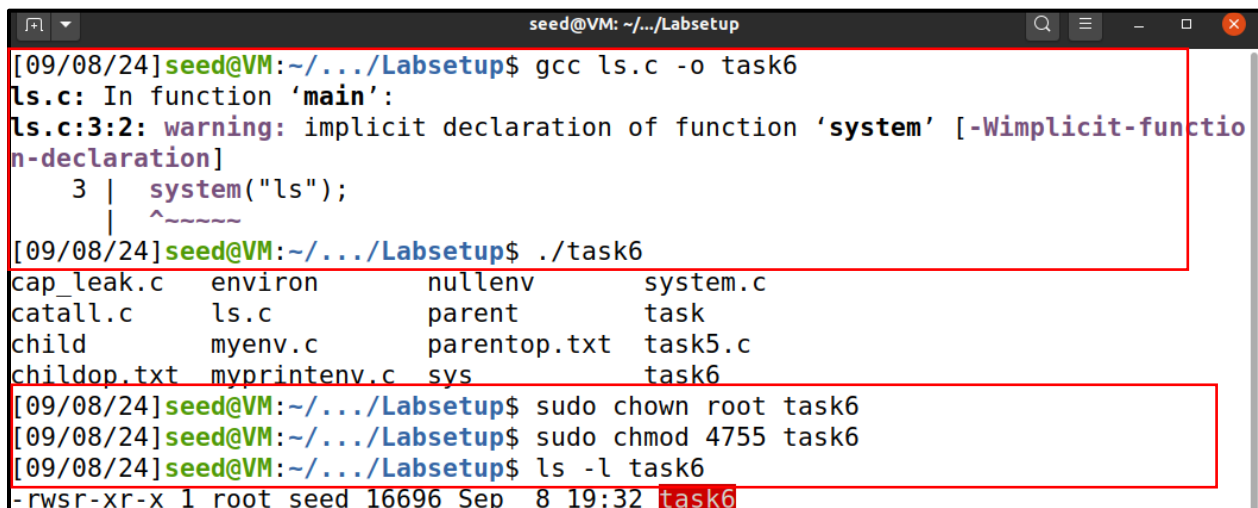
This task explores how the **PATH** environment variable interacts with **Set-UID programs**, and how manipulating the PATH variable can affect the execution of these programs. This is an important concept because the **PATH** variable determines where the system looks for executables when commands are run, and altering it can pose security risks, especially in the context of Set-UID programs.

- We write a program using the `system()` function which will execute the `ls` command.
- We compile and execute the program.
- Now we change the permissions of this `./task6` file to root.



```
ls.c
~/Purva/Labsetup

1 int main()
2 {
3     system("ls");
4     return 0;
5 }
```



```
seed@VM: ~/.../Labsetup
[09/08/24] seed@VM:~/.../Labsetup$ gcc ls.c -o task6
ls.c: In function 'main':
ls.c:3:2: warning: implicit declaration of function 'system' [-Wimplicit-function-declaration]
   3 |     system("ls");
     |     ^~~~~~
[09/08/24] seed@VM:~/.../Labsetup$ ./task6
cap_leak.c  environ      nullenv      system.c
catall.c    ls.c          parent       task
child       myenv.c       parentop.txt task5.c
childop.txt myprintenv.c sys          task6
[09/08/24] seed@VM:~/.../Labsetup$ sudo chown root task6
[09/08/24] seed@VM:~/.../Labsetup$ sudo chmod 4755 task6
[09/08/24] seed@VM:~/.../Labsetup$ ls -l task6
-rwsr-xr-x 1 root seed 16696 Sep  8 19:32 task6
```

- Now we check the address of the current directory with the `PWD` command.
- And print the environment variable that is `PATH`, with the help of `printenv` command
- Now we export our `PATH` of our working directory to the Environment variable `$PATH`.
- Now this does is we can run our executable like the `ls` command. So we can simply type in `task6` and it will work as `ls` command. This is quite dangerous as we are running it in a malicious way as the root permits overdoing the `ls` command.

```
seed@VM: ~/.../Labsetup
[09/08/24]seed@VM:~/.../Labsetup$ ls -l task6
-rwsr-xr-x 1 root seed 16696 Sep  8 19:32 task6
[09/08/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Purva/Labsetup
[09/08/24]seed@VM:~/.../Labsetup$ printenv PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:..
[09/08/24]seed@VM:~/.../Labsetup$ export PATH=/home/seed/Purva/Labsetup:$PATH
[09/08/24]seed@VM:~/.../Labsetup$ task6
cap_leak.c  environ      nullenv      system.c
catall.c    ls.c          parent       task
child       myenv.c       parentop.txt task5.c
childop.txt myprintenv.c sys          task6
[09/08/24]seed@VM:~/.../Labsetup$ ls
cap_leak.c  environ      nullenv      system.c
catall.c    ls.c          parent       task
child       myenv.c       parentop.txt task5.c
childop.txt myprintenv.c sys          task6
[09/08/24]seed@VM:~/.../Labsetup$ ./task6
cap_leak.c  environ      nullenv      system.c
catall.c    ls.c          parent       task
child       myenv.c       parentop.txt task5.c
childop.txt myprintenv.c sys          task6
[09/08/24]seed@VM:~/.../Labsetup$
```

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

The objective of this task is to explore how the LD_PRELOAD environment variable interacts with Set-UID programs. LD_PRELOAD is used to load custom shared libraries before others, potentially allowing users to override functions in standard libraries like libc.

So we have program given in the lab file that is a simple program:

```
#include void sleep (int s)
```

```
{
```

```
/* If this is invoked by a privileged program, you can do damages here! */
```

```
printf("I am not sleeping!\n");
```

```
}
```

After writing and compiling the program we run the command **gcc -shared -o libmylib.so.1.0.1 mylib.o -lc** is used to create a shared library in Linux.

The command **export LD_PRELOAD=./libmylib.so.1.0.1** is used to instruct the dynamic linker/loader to load a custom shared library (libmylib.so.1.0.1) before any other libraries when a program is executed. This can be useful for overriding functions in standard libraries.

Then we compile a new program that is

```
/* myprog.c */
```

```
int main()
```

```
{
```

```
sleep(1);
```

```
return 0;
```

```
}
```

Case 1: Make myprog a regular program, and run it as a normal user

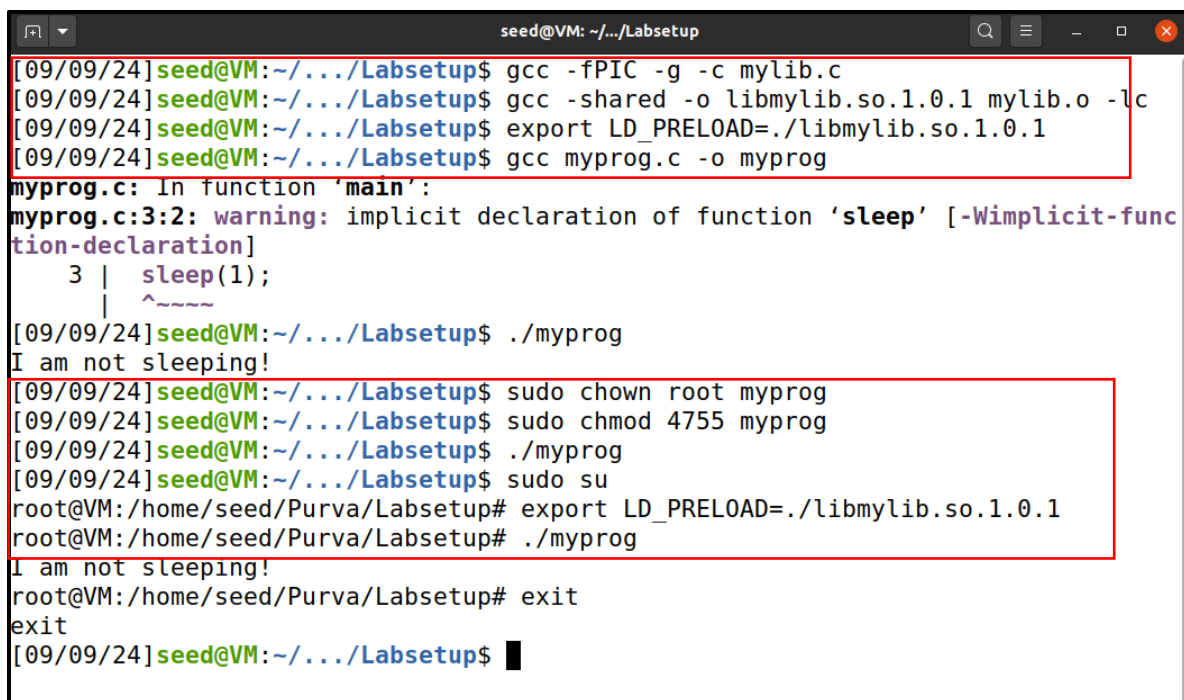
So, we run the program `./myprog` and get the output as “I am not sleeping” after waiting for 1 second.

Case 2: Make myprog a Set-UID root program, and run it as a normal user.

Then we changed the permissions of the myprog program to root and run the program again but this time it doesn't print anything.

Case 3: Make myprog a Set-UID root program, export the LD_PRELOAD environment variable again in the root account and run it.

Now we enter the root mode and then export the LD_PRELOAD and then run the program and this time it runs the sleep function and prints “I am not sleeping” and later exit the root.



```
seed@VM: ~/.../Labsetup
[09/09/24] seed@VM:~/.../Labsetup$ gcc -fPIC -g -c mylib.c
[09/09/24] seed@VM:~/.../Labsetup$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
[09/09/24] seed@VM:~/.../Labsetup$ export LD_PRELOAD=./libmylib.so.1.0.1
[09/09/24] seed@VM:~/.../Labsetup$ gcc myprog.c -o myprog
myprog.c: In function 'main':
myprog.c:3:2: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
   3 |     sleep(1);
     |     ^~~~~
[09/09/24] seed@VM:~/.../Labsetup$ ./myprog
I am not sleeping!
[09/09/24] seed@VM:~/.../Labsetup$ sudo chown root myprog
[09/09/24] seed@VM:~/.../Labsetup$ sudo chmod 4755 myprog
[09/09/24] seed@VM:~/.../Labsetup$ ./myprog
[09/09/24] seed@VM:~/.../Labsetup$ sudo su
root@VM:/home/seed/Purva/Labsetup# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/Purva/Labsetup# ./myprog
I am not sleeping!
root@VM:/home/seed/Purva/Labsetup# exit
exit
[09/09/24] seed@VM:~/.../Labsetup$
```


Case 4: Make myprog a Set-UID user1 program (i.e., the owner is user1, which is another user account), export the LD_PRELOAD environment variable again in a different user's account (not-root user) and run it.

Now we create a new user which we name as user1

Change the permissions of myprog to user1

Run the program and don't get any output.

In the case of a Set-UID program, the behavior may differ due to security restrictions. Set-UID programs typically ignore LD_PRELOAD for security reasons to prevent privilege escalation by malicious users.

```
seed@VM: ~/.../Labsetup
exit
[09/09/24]seed@VM:~/.../Labsetup$ 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' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for user1
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] y
[09/09/24]seed@VM:~/.../Labsetup$ sudo chown user1 myprog
[09/09/24]seed@VM:~/.../Labsetup$ sudo chmod 4755 user1
chmod: cannot access 'user1': No such file or directory
[09/09/24]seed@VM:~/.../Labsetup$ sudo chmod 4755 myprog
[09/09/24]seed@VM:~/.../Labsetup$ ./myprog
[09/09/24]seed@VM:~/.../Labsetup$
```

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

In this task, you examine the differences between invoking external programs using the system() function and the execve() function in a privileged (Set-UID) context.

system(): This function is dangerous when used in a privileged program because it invokes a shell (/bin/sh) to execute the command, making it vulnerable to environment variables like

PATH. Malicious users can modify the PATH variable to run their own programs instead of system commands like /bin/ls.

execve(): This function directly executes the specified program without invoking a shell. As a result, it's not vulnerable to PATH manipulation.

Now we first change the permissions of the program catall that is provided in the lab file and make it root owned.

Compile and run the program with execve() commented out so which means only system() is being executed, with an input as a file.

So basically the program executes the task of cat function.

```
[09/09/24] seed@VM: ~/.../Labsetup$ sudo chown root catall  
[09/09/24] seed@VM: ~/.../Labsetup$ sudo chmod 4755 catall
```

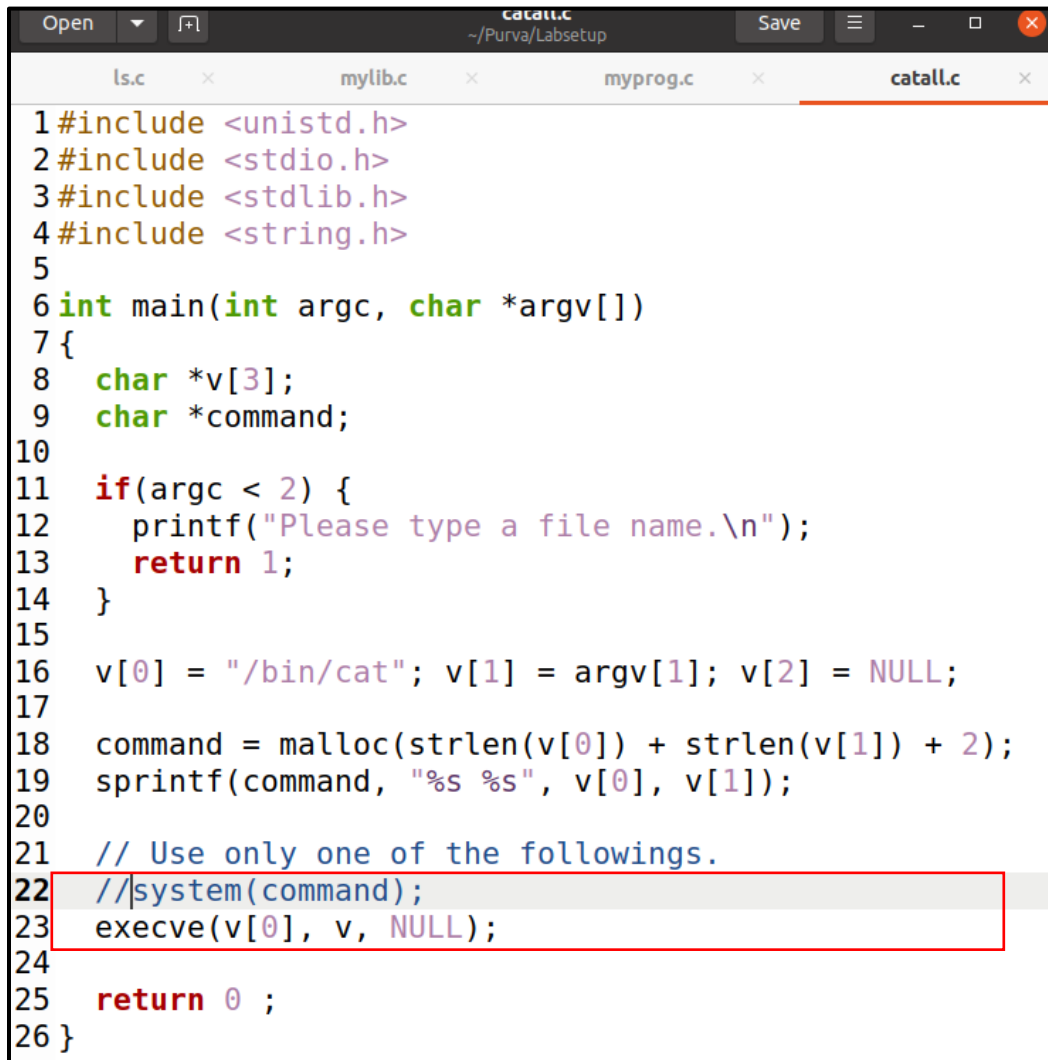
Now we run the program with "mylib.c; rm mylib.c", **rm mylib.c:** This command attempts to delete the file mylib.c (using the rm command).

Now after deleting the file we again run the ./catall mylib.c command and it displays no such file.

So it is clear that system() function allows BOB to delete the file even though he has not written that file.

```
seed@VM: ~/.../Labsetup  
[09/09/24] seed@VM: ~/.../Labsetup$ gcc catall.c -o catall  
[09/09/24] seed@VM: ~/.../Labsetup$ ./catall  
Please type a file name.  
[09/09/24] seed@VM: ~/.../Labsetup$ ./catall mylib.c  
#include <stdio.h>  
void sleep(int s)  
{  
    printf("I am not sleeping!\n");  
}  
[09/09/24] seed@VM: ~/.../Labsetup$ ./catall "mylib.c"  
#include <stdio.h>  
void sleep(int s)  
{  
    printf("I am not sleeping!\n");  
}  
[09/09/24] seed@VM: ~/.../Labsetup$ ./catall "mylib.c;rm mylib.c"  
#include <stdio.h>  
void sleep(int s)  
{  
    printf("I am not sleeping!\n");  
}  
[09/09/24] seed@VM: ~/.../Labsetup$ ./catall "mylib.c"  
/bin/cat: mylib.c: No such file or directory  
[09/09/24] seed@VM: ~/.../Labsetup$
```

Now we comment out `system()` and try to execute the `execve()` function and see if BOB can perform the same execution and still be able to delete the file.



```
1#include <unistd.h>
2#include <stdio.h>
3#include <stdlib.h>
4#include <string.h>
5
6int main(int argc, char *argv[])
7{
8    char *v[3];
9    char *command;
10
11    if(argc < 2) {
12        printf("Please type a file name.\n");
13        return 1;
14    }
15
16    v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;
17
18    command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
19    sprintf(command, "%s %s", v[0], v[1]);
20
21    // Use only one of the followings.
22    //system(command);
23    execve(v[0], v, NULL);
24
25    return 0 ;
26 }
```

So we first change the permissions of the file that we will be executing.

Now run the program with `ls.c` as input file.

Bob is able to read the contents of the file.

Now when tried to use the `rm` command and delete the file he is not able to do so, which means `execve()` blocks the user from deleting the file.

Observations:

- When using `system()`, the program is vulnerable to PATH manipulation, allowing an attacker to execute arbitrary code with root privileges.

- Using `execve()` mitigates this risk because it bypasses the shell, making it much safer for Set-UID programs.

```
seed@VM: ~/.../Labsetup
[09/09/24] seed@VM:~/.../Labsetup$ gcc catall.c -o catall1
[09/09/24] seed@VM:~/.../Labsetup$ sudo chown root catall1
[09/09/24] seed@VM:~/.../Labsetup$ sudo chmod 4755 catall1
[09/09/24] seed@VM:~/.../Labsetup$ ls -l catall1
-rwsr-xr-x 1 root seed 16928 Sep  9 12:49 catall1
[09/09/24] seed@VM:~/.../Labsetup$ ./catall1
Please type a file name.
[09/09/24] seed@VM:~/.../Labsetup$ ./catall1 ls.c
int main()
{
    system("ls");
    return 0;
}
```

```
seed@VM: ~/.../Labsetup
[09/09/24] seed@VM:~/.../Labsetup$ ./catall1 ls.c
int main()
{
    system("ls");
    return 0;
}
[09/09/24] seed@VM:~/.../Labsetup$ ./catall1 "ls.c;rm ls.c"
/bin/cat: 'ls.c;rm ls.c': No such file or directory
[09/09/24] seed@VM:~/.../Labsetup$
```

Task 9: Capability Leaking

This task investigates how capabilities can leak in Set-UID programs when they relinquish root privileges.

- **setuid()**: This system call is used to drop root privileges in Set-UID programs, but it's possible for capabilities (such as the ability to write to files) to leak if not properly handled.

Steps:

We compile the provided program, make it a Set-UID root program, and run it as a normal user.

We try changing the permissions until we get `s` which is a shell permission.

Then we compile and change its permissions and then run the `cap_leak` program and that is how it has allowed us to enter the shell where we can make changes, add content in the `/etc/zzz` file.

The program opens the file `/etc/zzz` (which is a privileged file), drops root privileges using `setuid(getuid())`, and then forks a child process. In a real-world scenario, the child process could be compromised, leading to an attacker injecting malicious code.

Observations:

- Despite dropping root privileges, the child process is still able to modify the privileged file `/etc/zzz` because the file descriptor `fd` was inherited by the child process before privileges were dropped.
- This demonstrates a capability leak, where the process still has access to privileged capabilities (in this case, writing to a root-owned file) even after privileges were relinquished.

These tasks help illustrate the risks of improper handling of environment variables and privileges in Set-UID programs, as well as the importance of understanding how system calls like `execve()` and `setuid()` behave in different contexts.

```
seed@VM: ~/.../Labsetup
[09/09/24]seed@VM:~/.../Labsetup$ cat /etc/zzz
cat: /etc/zzz: No such file or directory
[09/09/24]seed@VM:~/.../Labsetup$ sudo cat > /etc/zzz
bash: /etc/zzz: Permission denied
[09/09/24]seed@VM:~/.../Labsetup$ sudo gedit /etc/zzz
sudo: gedit: command not found
[09/09/24]seed@VM:~/.../Labsetup$ sudo gedit /etc/zzz

(gedit:5053): Tepl-WARNING **: 13:10:15.861: GVfs metadata is not supported. Fall
back to TeplMetadataManager. Either GVfs is not correctly installed or GVfs met
adata are not supported on this platform. In the latter case, you should configu
re Tepl with --disable-gvfs-metadata.
[09/09/24]seed@VM:~/.../Labsetup$ ls -l /etc/zzz
-rw-r--r-- 1 root root 12 Sep  9 13:10 /etc/zzz
[09/09/24]seed@VM:~/.../Labsetup$ cat /etc/zzz
Hi There!!!
[09/09/24]seed@VM:~/.../Labsetup$ echo "Info" > /etc/zzz
bash: /etc/zzz: Permission denied
[09/09/24]seed@VM:~/.../Labsetup$ sudo chmod 0644 /etc/zzz
[09/09/24]seed@VM:~/.../Labsetup$ ls -l /etc/zzz
-rw-r--r-- 1 root root 12 Sep  9 13:10 /etc/zzz
[09/09/24]seed@VM:~/.../Labsetup$
```

```
[09/09/24]seed@VM:~/.../Labsetup$ gcc cap_leak.c -o capleak
[09/09/24]seed@VM:~/.../Labsetup$ sudo chown root:root capleak
[09/09/24]seed@VM:~/.../Labsetup$ sudo chmod 4755 capleak
[09/09/24]seed@VM:~/.../Labsetup$ ls -l capleak
-rwsr-xr-x 1 root root 17008 Sep  9 13:17 capleak
[09/09/24]seed@VM:~/.../Labsetup$ ./capleak
fd is 3
$ cat /etc/zzz
Hi There!!!
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
$ echo "Information" >&3
$ cat /etc/zzz
Hi There!!!
Information
$
```

Conclusion:

This lab provided an in-depth exploration of the interaction between environment variables, system calls, and Set-UID programs, highlighting several key security implications. Through practical tasks, the following important concepts were demonstrated:

1. **Impact of Environment Variables:** The lab showed how environment variables, such as PATH and LD_PRELOAD, can significantly influence the behavior of programs. When improperly handled, these variables can lead to unintended consequences, especially in privileged (Set-UID) programs.

2. **Vulnerabilities in Set-UID Programs:** The tasks illustrated the risks associated with Set-UID programs, which run with elevated privileges. If Set-UID programs fail to properly restrict or sanitize environment variables, attackers can manipulate them to escalate privileges, bypass security, or execute arbitrary code.
3. **Command Injection and Security Risks:** The lab highlighted the dangers of using functions like `system()` that invoke shells. These functions are vulnerable to command injection attacks, which allow attackers to run unauthorized commands by manipulating input, as demonstrated in the task involving `system()` versus `execve()`.
4. **Understanding Privilege Leaks:** Tasks related to the `setuid()` system call emphasized the importance of securely relinquishing privileges. Even after dropping root privileges, improperly handled file descriptors and capabilities can still be exploited by attackers, leading to **capability leaking**.
5. **Importance of Proper Input Handling:** This lab emphasized the necessity of input validation, sanitization, and strict handling of environment variables to prevent exploitation. Programs, especially those with elevated privileges, should never trust external input, and functions that interact with shells should be avoided or replaced with safer alternatives like `execve()`.