

PES UNIVERSITY

INFORMATION SECURITY

LAB REPORT

NAME:AMRUTHA BS

Lab: #1

Environment Variable and Set-UID Program Lab

OBJECTIVE:

To understand

- **How environment variables work**
- **How they are propagated from parent process to child**
- **How they affect system/program behavior**

This lab particularly oriented in how environment variables affect the behavior of Set-UID programs, which are usually privileged programs.

Task 1: Manipulating environment variables

In this task, Study the commands that can be used to set and unset environment variables.

- **Use printenv or env command to print out the environment variables. If you are interested in some particular environment variables, such as PWD, you can use**

Command:

\$ printenv PWD

or

\$ env | grep PWD

It can be clearly seen in the below screenshot that the value of PWD is **/home/seed**.

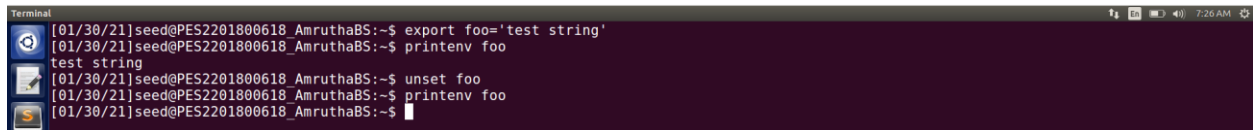


```
Terminal
[01/30/21]seed@PES2201800618_AmruthaBS:~$ printenv PWD
/home/seed
[01/30/21]seed@PES2201800618_AmruthaBS:~$ env|grep PWD
PWD=/home/seed
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

- Use export and unset to set or unset environment variables.

Command:

```
$ export foo='test string'
$ printenv foo
$ unset foo
$ printenv foo
```



```
Terminal
[01/30/21]seed@PES2201800618_AmruthaBS:~$ export foo='test string'
[01/30/21]seed@PES2201800618_AmruthaBS:~$ printenv foo
test string
[01/30/21]seed@PES2201800618_AmruthaBS:~$ unset foo
[01/30/21]seed@PES2201800618_AmruthaBS:~$ printenv foo
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

Task 2: Inheriting environment variables from parents

In this task, Study how environment variables are inherited by child processes from their parents. In Unix, fork() creates a new process by duplicating the calling process. The new process, referred to as the child, is an exact duplicate of the calling process, referred to as the parent; however, several things are not inherited by the child. In this task, Students should know whether the parent's environment variables are inherited by the child process or not.

Step 1: Please compile and run the following program, and describe your observation. Because the output contains many strings, you should save the output into a file, such as using `a.out > child` (assuming that a.out is your executable file name).

```
penv.c (~/) - gedit
/*penv program */
#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);
    }
}
```

Commands:

\$ gcc penv.c

\$ a.out>child

\$ ls -l child

```
Terminal
[01/30/21]seed@PES2201800618_AmruthaBS:~$ gcc penv.c
[01/30/21]seed@PES2201800618_AmruthaBS:~$ a.out>child
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ls -l child
-rw-rw-r-- 1 seed seed 4006 Jan 30 07:33 child
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

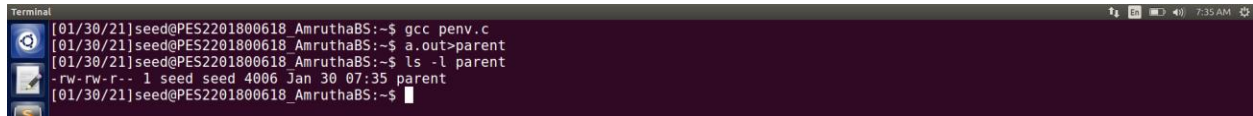
Step 2: Now comment out the printenv() statement in the child process case, and uncomment the printenv() statement in the parent process case. Compile and run the code, and describe your observation. Save the output in another file.

Commands:

\$ gcc penv.c

\$ a.out>parent

\$ ls -l parent

A terminal window with a dark purple background. The prompt is [01/30/21]seed@PES2201800618_AmruthaBS:~\$. The user enters 'gcc penv.c', then 'a.out>parent', and finally 'ls -l parent'. The output shows a file named 'parent' with permissions '-rw-rw-r--', owned by 'seed', size '4006', dated 'Jan 30 07:35', and located in the 'parent' directory.

```
[01/30/21]seed@PES2201800618_AmruthaBS:~$ gcc penv.c
[01/30/21]seed@PES2201800618_AmruthaBS:~$ a.out>parent
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ls -l parent
-rw-rw-r-- 1 seed seed 4006 Jan 30 07:35 parent
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

Step 3: Compare the difference of these two files using the diff command. Please draw your conclusion.

Command:

\$ diff child parent

It can be concluded from the below screenshot that both parent and child files have the same content. The parent's environment variables are inherited by the child process.

A terminal window showing the command 'diff child parent' being executed. The output is empty, indicating that the two files are identical.

```
[01/30/21]seed@PES2201800618_AmruthaBS:~$ diff child parent
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

Task 3: Environment variables and execve()

In this task, Study how environment variables are affected when a new program is executed via `execve()`. The function `execve()` calls a system call to load a new command and execute it; this function never returns. No new process is created; instead, the calling process's text, data, bss, and stack are overwritten by that of the program loaded. Essentially, `execve()` runs the new program inside the calling process. Here our interest is what happens to the environment variables; are they automatically inherited by the new program?

Step 1: Please compile and run the following program, and describe your observation. This program simply execute a program called `/usr/bin/env`, which prints out the environment variables of the current process.



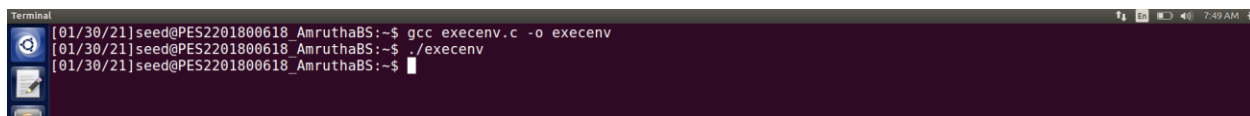
```
*execenv.c (~/) - gedit
/* execenv.c program */
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
extern char **environ;
int main()
{
    char *argv[2];
    argv[0]="/usr/bin/env";
    argv[1]=NULL;
    execve("/usr/bin/env",argv,NULL);
    return 0 ;
}
```

Commands:

```
$ gcc execenv.c -o execenv
```

```
$/execenv
```

Give your observation with screen shot.



```
Terminal
[01/30/21]seed@PES2201800618_AmruthaBS:~$ gcc execenv.c -o execenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ./execenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

Step 2: Now, change the invocation of execve() to the following, and describe your observation. (make changes in the program given above)

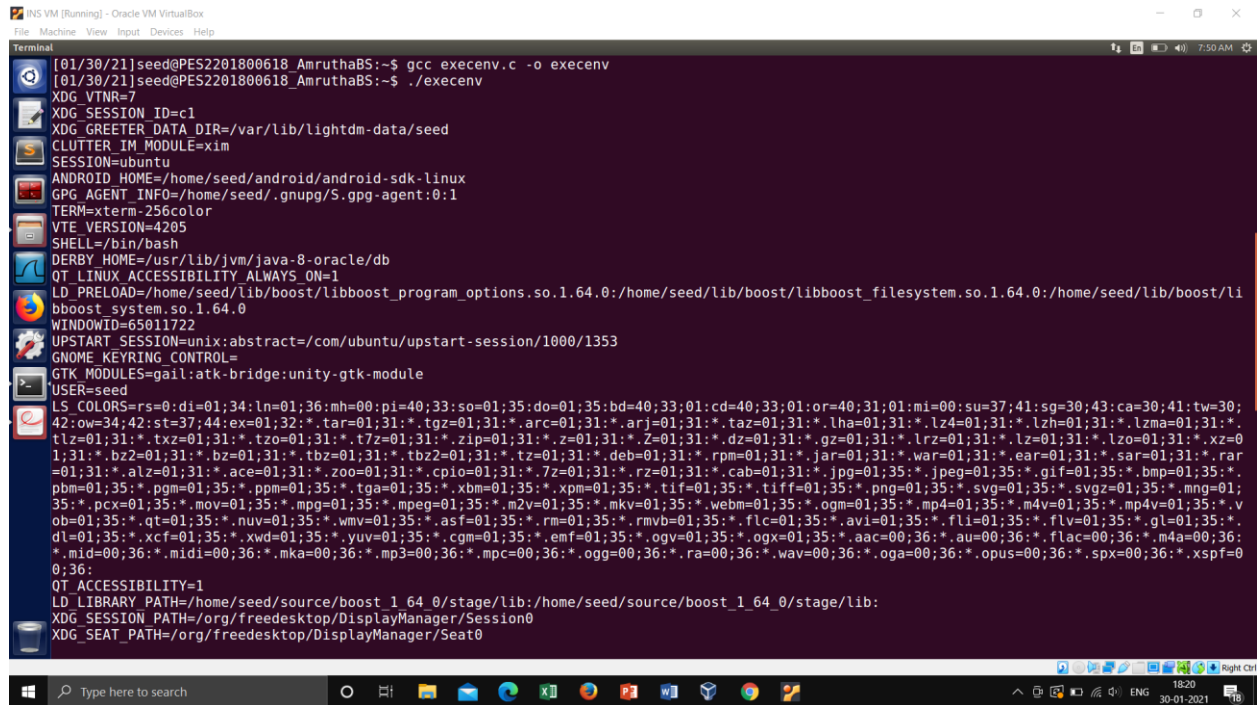
```
execve("/usr/bin/env", argv, environ);
```

Commands:

```
$ gcc execenv.c -o execenv
```

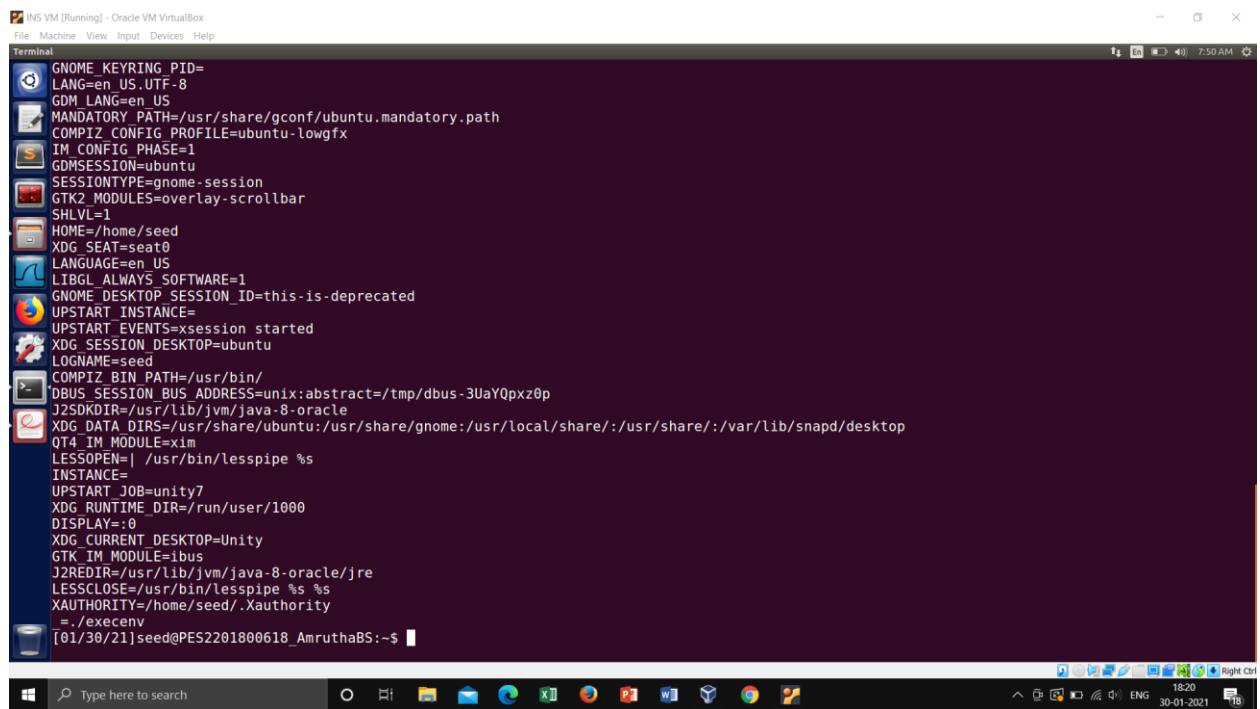
```
$/execenv
```

Give your observation with screen shot.



```
INS VM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Terminal
[01/30/21]seed@PES2201800618_AmruthaBS:~$ gcc execenv.c -o execenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ./execenv
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed/lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.0
WINDOWID=65011722
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1353
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=seed
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33:ol=cd=40;33:or=40;31:01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lzh=01;31:*.lzm=01;31:*.t1z=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.t7z=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=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:
QT_ACCESSIBILITY=1
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
```



```
INS VM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Terminal
GNOME_KEYRING_PID=
LANG=en_US.UTF-8
GDM_LANG=en_US
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_CONFIG_PROFILE=ubuntu-lowgfx
IM_CONFIG_PHASE=1
GDMSESSION=ubuntu
SESSIONTYPE=gnome-session
GTK2_MODULES=overlay-scrollbar
SHLVL=1
HOME=/home/seed
XDG_SEAT=seat0
LANGUAGE=en_US
LIBGL_ALWAYS_SOFTWARE=1
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
UPSTART_INSTANCE=
UPSTART_EVENTS=xsession started
XDG_SESSION_DESKTOP=ubuntu
LOGNAME=seed
COMPIZ_BIN_PATH=/usr/bin/
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-3UaYQpxz0p
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share:/usr/share:/var/lib/napd/desktop
QT4_IM_MODULE=xim
LESSOPEN=| /usr/bin/lesspipe %s
INSTANCE=
UPSTART_JOB=unity7
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
GTK_IM_MODULE=ibus
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
LESSCLOSE=/usr/bin/lesspipe %s %s
XAUTHORITY=/home/seed/.Xauthority
./execenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

From the above screenshots it can be concluded that by running the `execve()` command the new program does not automatically inherit the environment

variables of the old program. They have to be explicitly passed to the new program.

Task 4: Environment variables and system()

In this task, Study how environment variables are affected when a new program is executed via the `system()` function. This function is used to execute a command, but unlike `execve()`, which directly executes a command, `system()` actually executes `"/bin/sh -c command"`, i.e., it executes `/bin/sh`, and asks the shell to execute the command.

If you look at the implementation of the `system()` function, you will see that it uses `execl()` to execute `/bin/sh`; `execl()` calls `execve()`, passing to it the environment variables array.

Therefore using `system()`, the environment variables of the calling process is passed to the new program `/bin/sh`.

Please compile and run the following program to verify this.



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

When the above program is executed the environment variables are displayed as shown on the below screenshot. Hence, it can be concluded that calling process passes the environment variables array to the new program.

```
INS VM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Terminal
GNOME KEYRING PID=
LANG=en_US.UTF-8
GDM LANG=en_US
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_CONFIG_PROFILE=ubuntu-lowgfx
IM_CONFIG_PHASE=1
GDMSESSION=ubuntu
SESSIONTYPE=gnome-session
GTK2_MODULES=overlay-scrollbar
SHLVL=1
HOME=/home/seed
XDG_SEAT=seat0
LANGUAGE=en_US
LIBGL_ALWAYS_SOFTWARE=1
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
UPSTART_INSTANCE=
UPSTART_EVENTS=xsession started
XDG_SESSION_DESKTOP=ubuntu
LOGNAME=seed
COMPIZ_BIN_PATH=/usr/bin/
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-3UaYQpxz0p
J25DKDIR=/usr/lib/jvm/java-8-oracle
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share:/usr/share:/var/lib/flatpak/desktop
QT4_IM_MODULE=xim
LESSOPEN=| /usr/bin/lesspipe %s
INSTANCE=
UPSTART_JOB=unity7
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
GTK_IM_MODULE=ibus
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
LESSCLOSE=/usr/bin/lesspipe %s %s
XAUTHORITY=/home/seed/.Xauthority
./execenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

```
INS VM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

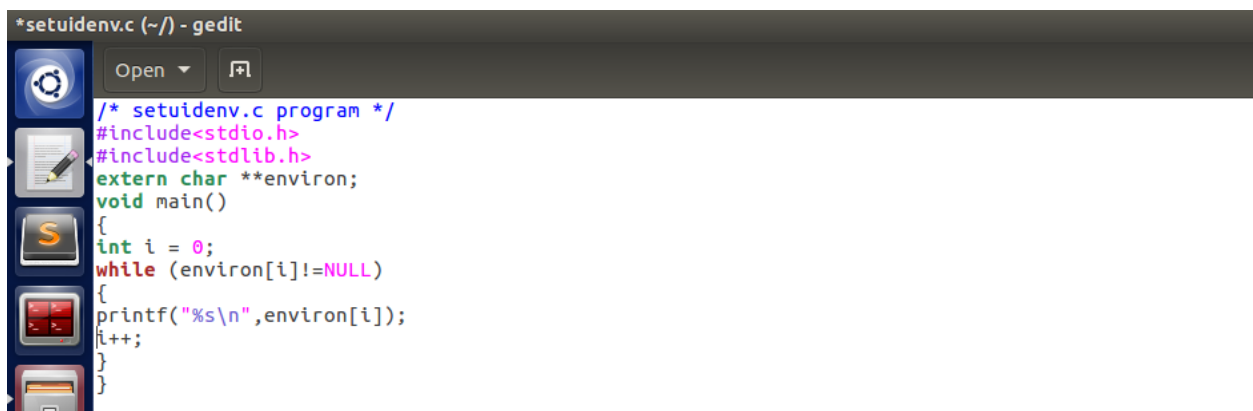
Terminal
LANG=en_US.UTF-8
XDG_CURRENT_DESKTOP=Unity
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33:01:cd=40;33:01:or=40;31:01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.taz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=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:
XMODIFIERS=@im=ibus
XDG_SESSION_DESKTOP=ubuntu
XAUTHORITY=/home/seed/.Xauthority
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
SHELL=/bin/bash
QT_ACCESSIBILITY=1
GDMSESSION=ubuntu
LESSCLOSE=/usr/bin/lesspipe %s %s
UPSTART_EVENTS=xsession started
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1353
XDG_VTNR=7
QT_IM_MODULE=ibus
PWD=/home/seed
JAVA_HOME=/usr/lib/jvm/java-8-oracle
CLUTTER_IM_MODULE=xim
ANDROID_HOME=/home/seed/android/android-sdk-linux
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/etc/xdg
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share:/usr/share:/var/lib/flatpak/desktop
VTE_VERSION=4205
JOB=unity-settings-daemon
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```


Task 5: Environment variable and Set-UID Programs

Set-UID is an important security mechanism in Unix operating systems. When a Set-UID program runs, it assumes the owner's privileges. For example, if the program's owner is root, then when anyone runs this program, the program gains the root's privileges during its execution.

Set-UID allows us to do many interesting things, but it escalates the user's privilege when executed, making it quite risky. Although the behaviors of Set-UID programs are decided by their program logic, not by users, users can indeed affect the behaviors via environment variables. To understand how Set-UID programs are affected, let us first figure out whether environment variables are inherited by the Set-UID program's process from the user's process.

Step 1: We are going to write a program that can print out all the environment variables in the current process.



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

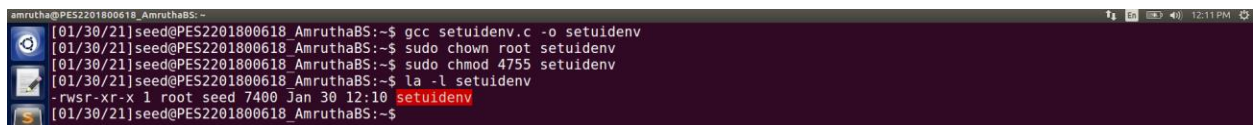
Commands:

\$ gcc setuidenv.c -o setuidenv

\$ sudo chown root setuidenv

\$ sudo chmod 4755 setuidenv

\$ la -l setuidenv



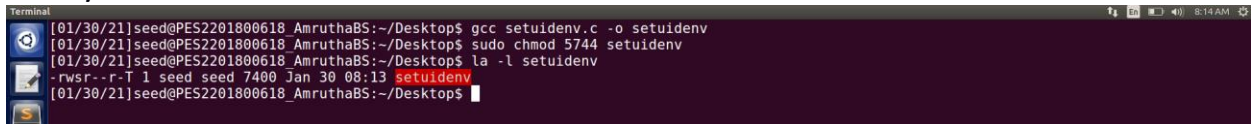
```
amrutha@PES2201800618_AmruthaBS:~$ gcc setuidenv.c -o setuidenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$ sudo chown root setuidenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$ sudo chmod 4755 setuidenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$ la -l setuidenv
-rwsr-xr-x 1 root seed 7400 Jan 30 12:10 setuidenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

Step 2: Compile the above program, change its ownership to root, and make it a Set-UID program.

Commands:

```
$gcc setuidenv.c -o setuidenv
$sudo chmod 5744 setuidenv
$ la -l setuidenv
```

Give your observation with screen shot.

A screenshot of a terminal window with a dark background. The prompt is [01/30/21]seed@PES2201800618_AmruthaBS:~/Desktop\$. The user enters 'gcc setuidenv.c -o setuidenv', then 'sudo chmod 5744 setuidenv', and finally 'la -l setuidenv'. The output shows the file permissions as -rwsr--r-- 1 seed seed 7400 Jan 30 08:13 setuidenv. The terminal window has a title bar with 'Terminal' and system icons on the right.

Step 3: In your Bash shell (you need to be in a normal user account, not the root account),use the export command to set the following environment variables (they may have already exist):

- **PATH**
- **LD_LIBRARY_PATH**
- **ANY NAME (this is an environment variable defined by you, so pick whatever name you want).**

Commands:

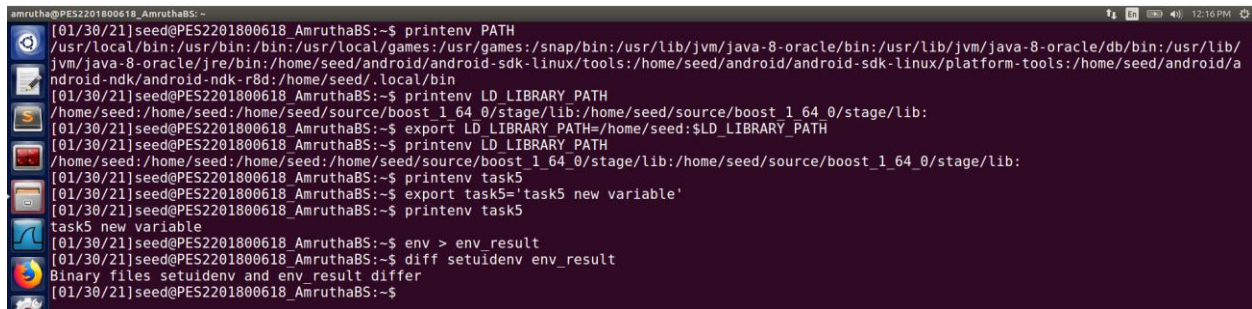
```
printenv PATH
printenv LD_LIBRARY_PATH
export LD_LIBRARY_PATH=/home/seed:$LD_LIBRARY_PATH
printenv LD_LIBRARY_PATH
printenv task5
export task5='task5 new variable'
printenv task5
env > env_result
diff setuidenv env_result
```

These environment variables are set in the user's shell process. Now, run the Set-UID program from Step 2 in your shell. After you type the name of the program in your shell, the shell forks a child process, and uses the child process to run the

program. Please check whether all the environment variables you set in the shell process (parent) get into the Set-UID child process. Describe your observation. If there are surprises to you, describe them.

Give your observation with screen shot.

No, all the environment variables you set in the shell process (parent) do not get into the Set-UID child process since the EUID and RUID of the child process is different.



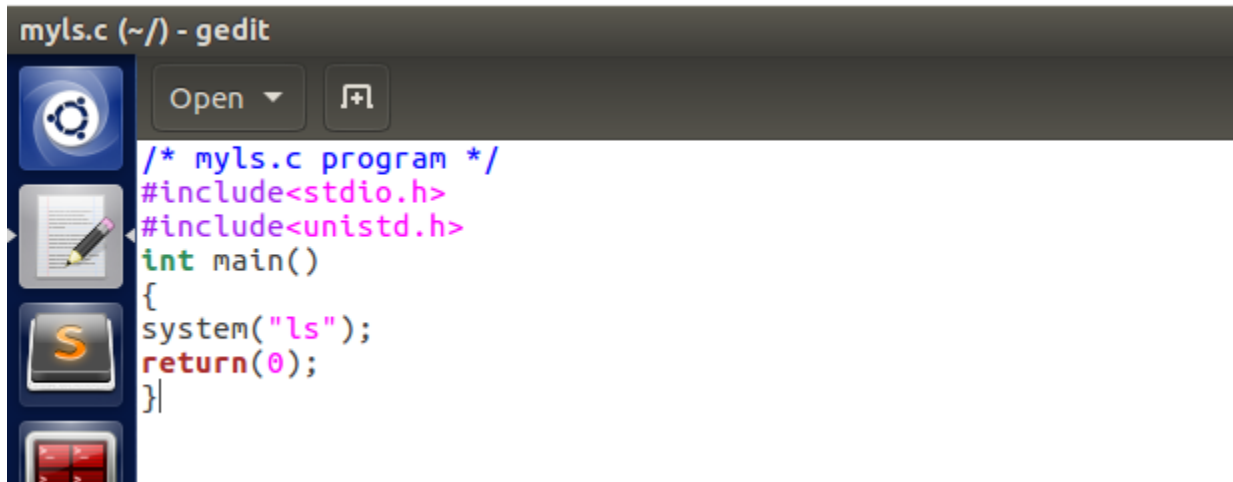
```
amrutha@PES2201800618_AmruthaBS:~  
[01/30/21]seed@PES2201800618_AmruthaBS:~$ printenv PATH  
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/snap/bin:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/  
jvm/java-8-oracle/jre/bin:/home/seed/android/android-sdk-linux/tools:/home/seed/android/android-sdk-linux/platform-tools:/home/seed/android/a  
ndroid-ndk/android-ndk-r8d:/home/seed/.local/bin  
[01/30/21]seed@PES2201800618_AmruthaBS:~$ printenv LD_LIBRARY_PATH  
/home/seed:/home/seed:/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:  
[01/30/21]seed@PES2201800618_AmruthaBS:~$ export LD_LIBRARY_PATH=/home/seed:$LD_LIBRARY_PATH  
[01/30/21]seed@PES2201800618_AmruthaBS:~$ printenv LD_LIBRARY_PATH  
/home/seed:/home/seed:/home/seed:/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:  
[01/30/21]seed@PES2201800618_AmruthaBS:~$ printenv task5  
[01/30/21]seed@PES2201800618_AmruthaBS:~$ export task5='task5 new variable'  
[01/30/21]seed@PES2201800618_AmruthaBS:~$ printenv task5  
task5 new variable  
[01/30/21]seed@PES2201800618_AmruthaBS:~$ env > env_result  
[01/30/21]seed@PES2201800618_AmruthaBS:~$ diff setuidenv env_result  
Binary files setuidenv and env_result differ  
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

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

Because of the shell program invoked, calling system() within a Set-UID program is quite dangerous. This is because the actual behavior of the shell program can be affected by environment variables, such as PATH; these environment variables are provided by the user, who may be malicious. By changing these variables, malicious users can control the behavior of the Set-UID program. In Bash, you can change the PATH environment variable in the following way (this example adds the directory /home/seed to the beginning of the PATH environment variable):

\$ export PATH=/home/seed:\$PATH

The Set-UID program below is supposed to execute the /bin/lis command; however, the programmer only uses the relative path for the lis command, rather than the absolute path:

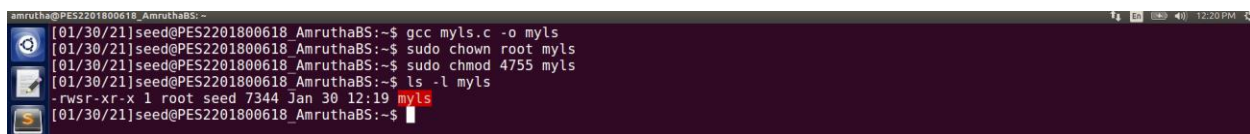


```
mysls.c (~/) - gedit
/* myls.c program */
#include<stdio.h>
#include<unistd.h>
int main()
{
    system("ls");
    return(0);
}
```

Please compile the above program, and change its owner to root, and make it a Set-UID program. Can you let this Set-UID program run your code instead of /bin/ls? If you can, is your code running with the root privilege? Describe and explain your observations.

Commands:

```
$ gcc myls.c -o myls
$ sudo chown root myls
$ sudo chmod 4755 myls
$ ls -l myls
```



```
amrutha@PES2201800618_AmruthaBS:~$ gcc myls.c -o myls
[01/30/21]seed@PES2201800618_AmruthaBS:~$ sudo chown root myls
[01/30/21]seed@PES2201800618_AmruthaBS:~$ sudo chmod 4755 myls
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ls -l myls
-rwxr-xr-x 1 root seed 7344 Jan 30 12:19 myls
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```



```
mysls.c (~/) - gedit
/* myls.c program */
#include<stdio.h>
#include<unistd.h>
int main()
{
    printf("\n This is my ls Program\n");
    printf("\n my real Uid is :%d\n My Effective uid is:%d\n",getuid(),geteuid());
    return(0);
}
```

Commands:

```
$ gcc ls.c -o ls
```

```
$ rm /bin/sh
```

```
$ ln -s /bin/zsh /bin/sh
```

```
$ export PATH=/home/seed/Desktop/Environ_set_uid:$PATH
```

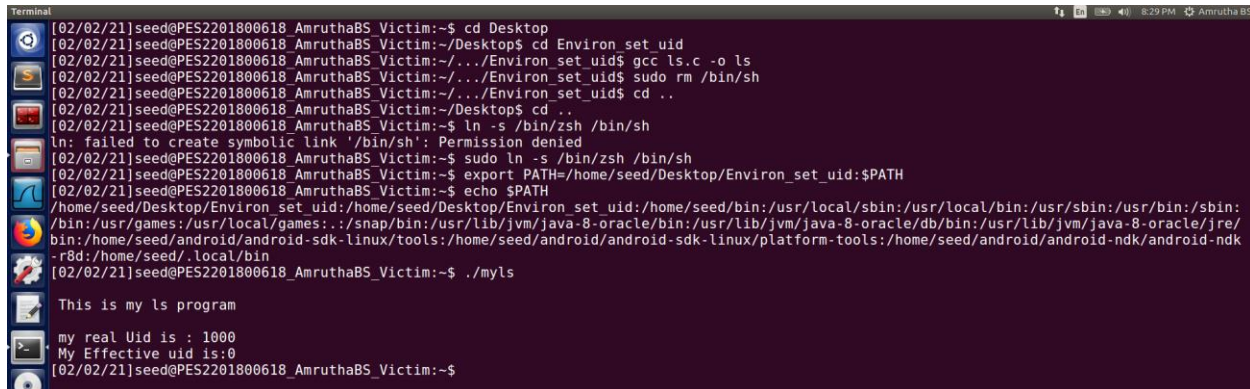
```
$ echo $PATH
```

```
$ ./mys
```

Give your observation with screen shot.

It can be seen from the below screen shot that when mys.c program is run now ls program defined by us gets executed.

And RUID and EUID is different.

A terminal window showing a series of commands and their outputs. The user is in a directory /home/seed/Desktop. They run 'cd Desktop', then 'cd Environ_set_uid', then 'gcc ls.c -o ls', then 'sudo rm /bin/sh', then 'cd ..', then 'ln -s /bin/zsh /bin/sh', then 'sudo ln -s /bin/zsh /bin/sh', then 'export PATH=/home/seed/Desktop/Environ_set_uid:\$PATH', then 'echo \$PATH', then './mys'. The output of './mys' is 'This is my ls program' followed by 'my real Uid is : 1000' and 'My Effective uid is:0'. The terminal title bar shows 'Terminal' and the user is 'AmruthaBS'.

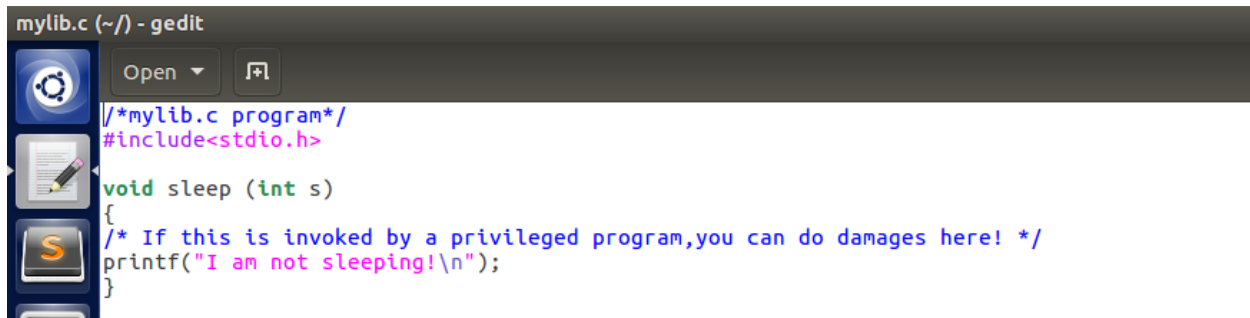
Task 7: The LD PRELOAD environment variable and Set-UID Programs

In this task, study how Set-UID programs deal with some of the environment variables. Several environment variables, including LD PRELOAD, LD LIBRARY PATH, and other LD influence the behavior of dynamic loader/linker. A dynamic loader/linker is the part of an operating system (OS) that loads (from persistent storage to RAM) and links the shared libraries needed by an executable at run time. In Linux, ld.so or ld-linux.so, are the dynamic loader/linker (each for different types of binary).

Among the environment variables that affect their behaviors, LD LIBRARY PATH and LD PRELOAD are the two that we are concerned in this lab. In Linux, LD LIBRARY PATH is a colon separated set of directories where libraries should be searched for first, before the standard set of directories. LD PRELOAD specifies a list of additional, user-specified, shared libraries to be loaded before all others. In this task, students will only study LD PRELOAD.

Step 1: First, see how these environment variables influence the behavior of dynamic loader/linker when running a normal program. Please follow these steps:

1. Build a dynamic link library. Create the following program, and name it mylib.c. It basically overrides the sleep() function in libc:



```
mylib.c (~/) - gedit
/*mylib.c program*/
#include<stdio.h>

void sleep (int s)
{
    /* If this is invoked by a privileged program,you can do damages here! */
    printf("I am not sleeping!\n");
}
```

2. Compile the above program using the following commands (in the -lc argument, the second character is `):

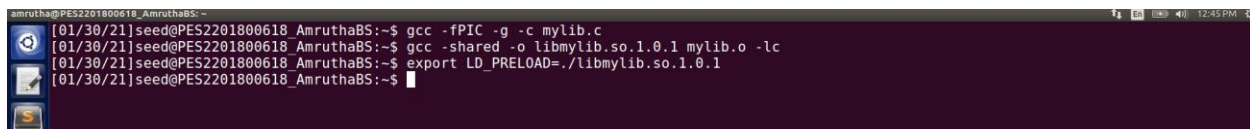
Command:

```
$ gcc -fPIC -g -c mylib.c
```

```
$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
```

3. Now, set the LD PRELOAD environment variable: **Command:**

```
$ export LD_PRELOAD=./libmylib.so.1.0.1
```



```
amrutha@PES2201800618_AmruthaBS:~$ gcc -fPIC -g -c mylib.c
[01/30/21]seed@PES2201800618_AmruthaBS:~$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
[01/30/21]seed@PES2201800618_AmruthaBS:~$ export LD_PRELOAD=./libmylib.so.1.0.1
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

4. Finally, compile the following program myprog, and it in the same directory as the above dynamic link library libmylib.so.1.0.1:



```
myprog.c (~/) - gedit
/*myprog.c program */
#include<stdio.h>

int main()
{
    sleep(1);
    return 0;
}
```


Give your observation with screen shot.

Step 2: After you have done the above, please run myprog under the following conditions, and observe what happens.

- Make myprog a regular program, and run it as a normal user.

```
Terminal
[02/02/21]seed@PES2201800618_AmruthaBS_Victim:~$ gcc myprog.c -o myprog
[02/02/21]seed@PES2201800618_AmruthaBS_Victim:~$ ls -l myprog
-rwxrwxr-x 1 seed seed 7348 Feb  2 21:09 myprog
[02/02/21]seed@PES2201800618_AmruthaBS_Victim:~$ ./myprog
I am not sleeping!
[02/02/21]seed@PES2201800618_AmruthaBS_Victim:~$
```

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

```
Terminal
[02/02/21]seed@PES2201800618_AmruthaBS_Victim:~$ sudo -s
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# gcc myprog.c -o myprog
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# chown root myprog
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# chmod 4755 myprog
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# exit
exit
[02/02/21]seed@PES2201800618_AmruthaBS_Victim:~$ ls -l myprog
-rwsr-xr-x 1 root root 7348 Feb  2 21:10 myprog
[02/02/21]seed@PES2201800618_AmruthaBS_Victim:~$ ./myprog
[02/02/21]seed@PES2201800618_AmruthaBS_Victim:~$
```

- Make myprog a Set-UID root program, export the LD PRELOAD environment variable again in the root account and run it.

```
Terminal
[02/02/21]seed@PES2201800618_AmruthaBS_Victim:~$ sudo -s
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# gcc myprog.c -o myprog
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# chown root myprog
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# chmod 4755 myprog
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# export LD_PRELOAD=./libmylib.so.1.0.1
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# ./myprog
I am not sleeping!
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~#
```

- 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.

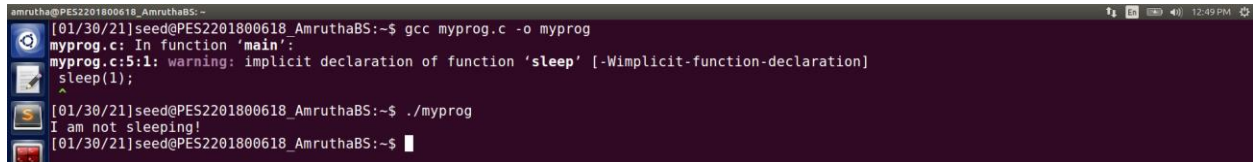
```
user1@PES2201800618_AmruthaBS_Victim:/home/seed
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# sudo -s
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# useradd -d /usr/user1 -m user1
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# chown user1 myprog
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# su user1
user1@PES2201800618_AmruthaBS_Victim:/home/seed$ gcc myprog.c -o myprog
user1@PES2201800618_AmruthaBS_Victim:/home/seed$ export LD_PRELOAD=./libmylib.so.1.0.1
user1@PES2201800618_AmruthaBS_Victim:/home/seed$ exit
exit
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# exit
exit
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~# ./myprog
I am not sleeping!
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~$ ls -l myprog
-rwxr-xr-x 1 user1 root 7348 Feb  2 21:14 myprog
[02/02/21]root@PES2201800618_AmruthaBS_Victim:~$
```

Command:

```
$ gcc myprog.c -o myprog
```

```
$ ./myprog
```

Give your observation with screen shot.



```
amrutha@PES2201800618_AmruthaBS:~$ gcc myprog.c -o myprog
myprog.c: In function 'main':
myprog.c:5:1: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
sleep(1);
^
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ./myprog
I am not sleeping!
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

Step 3: You should be able to observe different behaviors in the scenarios described above, even though you are running the same program. You need to figure out what causes the difference. Environment variables play a role here. Please design an experiment to figure out the main causes, and explain why the behaviors in Step 2 are different. (Hint: the child process may not inherit the LD * environment variables).

In root environment execute the myprog.c program

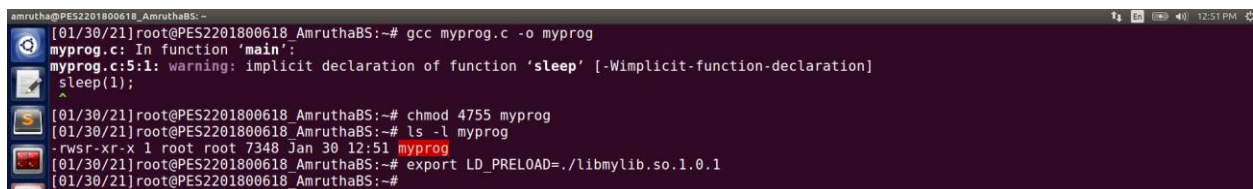
Command:

```
$ gcc myprog.c -o myprog
```

```
$ chmod 4755 myprog
```

```
$ ls -l myprog
```

```
$ export LD_PRELOAD=./libmylib.so.1.0.1
```



```
amrutha@PES2201800618_AmruthaBS:~$ gcc myprog.c -o myprog
myprog.c: In function 'main':
myprog.c:5:1: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
sleep(1);
^
[01/30/21]root@PES2201800618_AmruthaBS:~$ chmod 4755 myprog
[01/30/21]root@PES2201800618_AmruthaBS:~$ ls -l myprog
-rwsr-xr-x 1 root root 7348 Jan 30 12:51 myprog
[01/30/21]root@PES2201800618_AmruthaBS:~$ export LD_PRELOAD=./libmylib.so.1.0.1
[01/30/21]root@PES2201800618_AmruthaBS:~$
```

come out of root and check the behavior

Command:

```
$ ls -l myprog
```

```
$ export LD_PRELOAD=./libmylib.so.1.0.1
```

```
$ whoami
```

```
$ seed
```

```
$ ./myprog
```


Give your observation with screen shot.



```
amrutha@PES2201800618_AmruthaBS:~$ ls -l myprog
-rwsr-xr-x 1 root root 7348 Jan 30 12:51 myprog
amrutha@PES2201800618_AmruthaBS:~$ export LD_PRELOAD=./libmylib.so.1.0.1
amrutha@PES2201800618_AmruthaBS:~$ whoami
seed
amrutha@PES2201800618_AmruthaBS:~$ ./myprog
seed
amrutha@PES2201800618_AmruthaBS:~$
```

Task 8: Invoking external programs using system() versus execve()

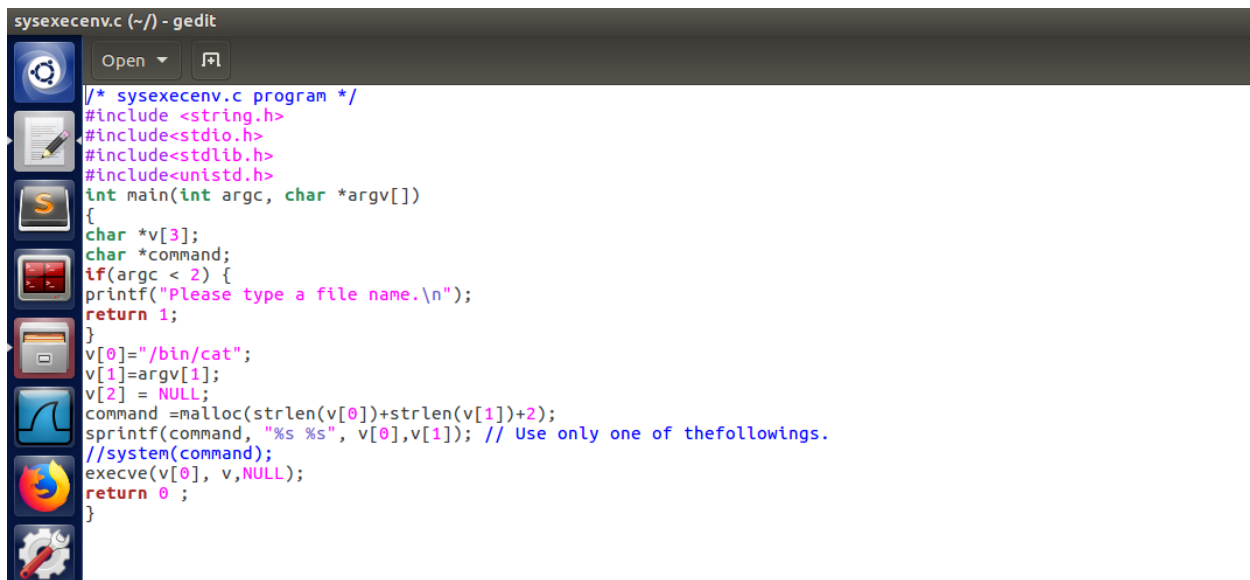
Although system() and execve() can both be used to run new programs, system() is quite dangerous if used in a privileged program, such as Set-UID programs. We have seen how the PATH environment variable affect the behavior of system(), because the variable affects how the shell works. execve() does not have the problem, because it does not invoke shell.

Invoking shell has another dangerous consequence, and this time, it has nothing to do with environment variables.

Look at the following scenario:

Bob works for an auditing agency, and he needs to investigate a company for a suspected fraud. For the investigation purpose, Bob needs to be able to read all the files in the company's Unix system; on the other hand, to protect the integrity of the system, Bob should not be able to modify any file. To achieve this goal, Vince, the superuser of the system, wrote a special set-rootuid program (see below), and then gave the executable permission to Bob.

This program requires Bob to type a file name at the command line, and then it will run /bin/cat to display the specified file. Since the program is running as a root, it can display any file Bob specifies. However, since the program has no write operations, Vince is very sure that Bob cannot use this special program to modify any file.



```
sysexecenv.c (~/) - gedit
/* sysexecenv.c program */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
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 thefollowings.
    //system(command);
    execve(v[0], v,NULL);
    return 0 ;
}
```

Step 1: Compile the above program, make root its owner, and change it to a Set-UID program. The program will use system() to invoke the command. If you were Bob, can you compromise the integrity of the system? For example, can you remove a file that is not writable to you?

(create two files “myfile”(owner is seed) and “rootfile”(owner is root))

Command:

```
$ gcc sysexecenv.c -o sysexecenv
$ sudo chown root sysexecenv
$ sudo chmod 4755 sysexecenv
$ ls -l rootfile myfile sysexecenv
$ ./sysexecenv "myfile;rm rootfile"
$ ls -l rootfile
```

Give your observation with screen shot.

Yes, Bob can compromise the integrity of the system.He can modify the files if the program uses system() to invoke the command.

It can be seen from the below screenshot that we are able to remove the rootfile when system() is used to invoke the command.

```
amrutha@PES2201800618_AmruthaBS:~$ gcc sysexecenv.c -o sysexecenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$ sudo chown root sysexecenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$ sudo chmod 4755 sysexecenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ls -l rootfile myfile sysexecenv
-rw-rw-r-- 1 seed seed 6 Jan 30 13:05 myfile
-rw-r--r-- 1 root root 18 Jan 30 13:06 rootfile
-rwsr-xr-x 1 root seed 7552 Jan 30 13:07 sysexecenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ./sysexecenv "myfile;rm rootfile"
Hello
rm: remove write-protected regular file 'rootfile'? y
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ls -l rootfile
ls: cannot access 'rootfile': No such file or directory
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

Step 2: Comment out the system(command) statement, and uncomment the execve() statement; the program will use execve() to invoke the command. Compile the program, and make it SetUID (owned by root). Do your attacks in Step 1 still work? Please describe and explain your observations.

Command:

```
$ gcc sysexecenv.c -o eysexecenv
$ sudo chown root sysexecenv
$ sudo chmod 4755 sysexecenv
$ ls -l rootfile myfile sysexecenv
$ ./sysexecenv "myfile;rm rootfile"
$ ls -l rootfile
```

Give your observation with screen shot.

The attacks in step 1 do not work now as the command is invoked using execve().

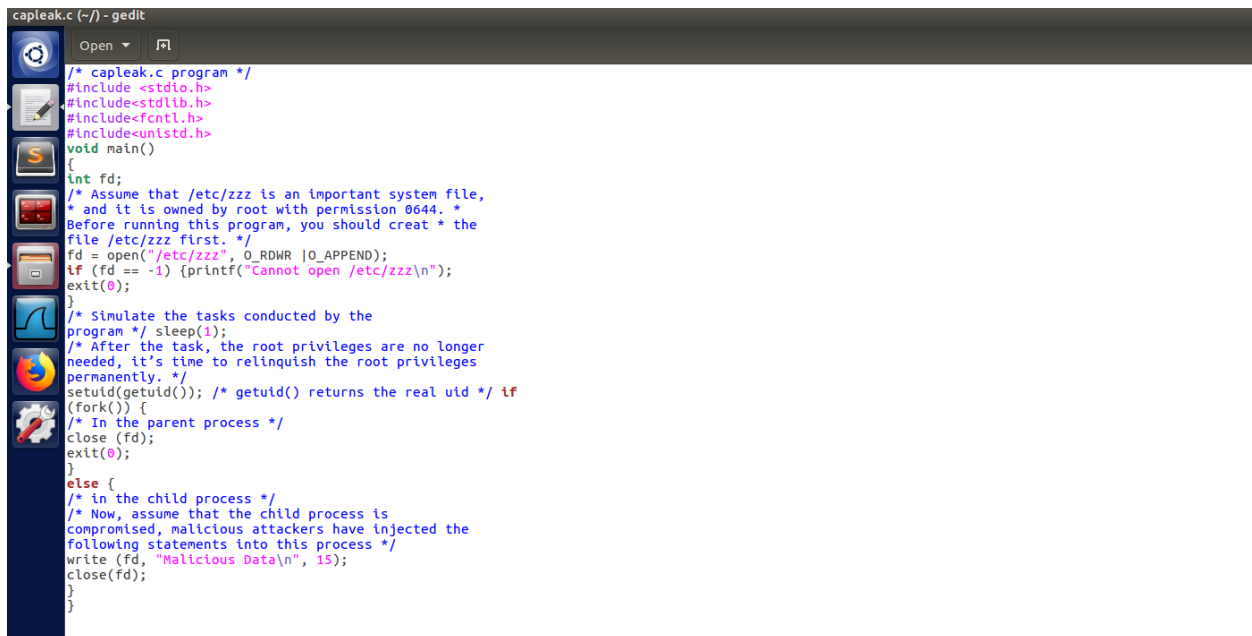
It can be seen form the below screenshot that we are unable to remove the rootfile. Here we are using execve() instead of system().

```
amrutha@PES2201800618_AmruthaBS:~$ gcc sysexecenv.c -o sysexecenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$ sudo chown root sysexecenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$ sudo chmod 4755 sysexecenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ls -l rootfile myfile sysexecenv
-rw-rw-r-- 1 seed seed 26 Jan 30 13:09 myfile
-rw-r--r-- 1 root root 26 Jan 30 13:13 rootfile
-rwsr-xr-x 1 root seed 7552 Jan 30 13:15 sysexecenv
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ./sysexecenv "myfile;rm rootfile"
/bin/cat: 'myfile;rm rootfile': No such file or directory
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ls -l rootfile
-rw-r--r-- 1 root root 26 Jan 30 13:13 rootfile
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```

Task 9: Capability Leaking

To follow the Principle of Least Privilege, Set-UID programs often permanently relinquish their root privileges if such privileges are not needed anymore. Moreover, sometimes, the program needs to hand over its control to the user; in

this case, root privileges must be revoked. The `setuid()` system call can be used to revoke the privileges. According to the manual, “`setuid()` sets the effective user ID of the calling process. If the effective UID of the caller is root, the real UID and saved set-user-ID are also set”. Therefore, if a Set-UID program with effective UID 0 calls `setuid(n)`, the process will become a normal process, with all its UIDs being set to n. When revoking the privilege, one of the common mistakes is capability leaking. The process may have gained some privileged capabilities when it was still privileged; when the privileged is downgraded, if the program does not clean up those capabilities, they may still be accessible by the non-privileged process. In other words, although the effective user ID of the process becomes non-privileged, the process is still privileged because it possesses privileged capabilities. Compile the following program, change its owner to root, and make it a Set-UID program. Run the program as a normal user, and describe what you have observed. Will the file `/etc/zxx` be modified? Please explain your observation.



```
capleak.c (~/) - gedit
/* capleak.c program */
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

void main()
{
    int fd;
    /* Assume that /etc/zxx is an important system file,
     * and it is owned by root with permission 0644. *
     * Before running this program, you should create * the
     * file /etc/zxx first. */
    fd = open("/etc/zxx", O_RDWR | O_APPEND);
    if (fd == -1) {printf("Cannot open /etc/zxx\n");
    exit(0);
    }
    /* Simulate the tasks conducted by the
    program */ sleep(1);
    /* After the task, the root privileges are no longer
    needed, it's time to relinquish the root privileges
    permanently. */
    setuid(getuid()); /* getuid() returns the real uid */ if
    (fork()) {
        /* In the parent process */
        close (fd);
        exit(0);
    }
    else {
        /* in the child process */
        /* Now, assume that the child process is
        compromised, malicious attackers have injected the
        following statements into this process */
        write (fd, "Malicious Data\n", 15);
        close(fd);
    }
}
```

Command:

```
$ gcc capleak.c -o capleak
$ sudo chown root capleak
$ sudo chmod 4755 capleak
$ ls -l capleak
```

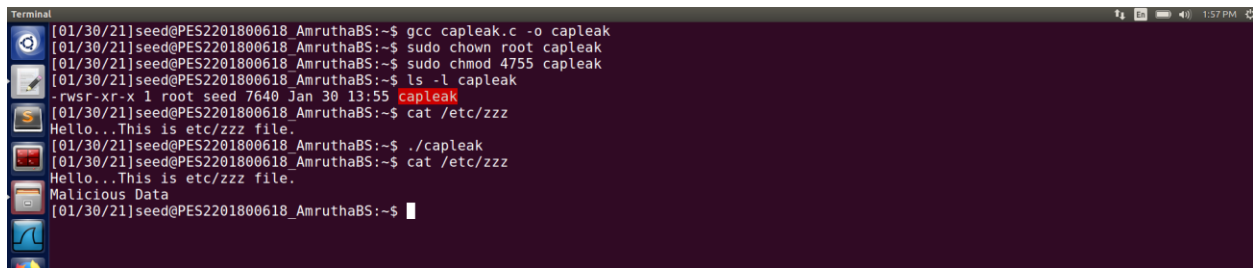
```
$ cat /etc/zzz
```

```
$/capleak
```

```
$ cat /etc/zzz
```

Give your observation with screen shot.

It can be seen from the below screenshot that /etc/zzz file has been modified. The file descriptor fd is closed only in the parent process and not in the child process. Hence, the child process still has the access to the file and can modify it.



```
Terminal
[01/30/21]seed@PES2201800618_AmruthaBS:~$ gcc capleak.c -o capleak
[01/30/21]seed@PES2201800618_AmruthaBS:~$ sudo chown root capleak
[01/30/21]seed@PES2201800618_AmruthaBS:~$ sudo chmod 4755 capleak
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ls -l capleak
-rwsr-xr-x 1 root seed 7640 Jan 30 13:55 capleak
[01/30/21]seed@PES2201800618_AmruthaBS:~$ cat /etc/zzz
Hello...This is etc/zzz file.
[01/30/21]seed@PES2201800618_AmruthaBS:~$ ./capleak
[01/30/21]seed@PES2201800618_AmruthaBS:~$ cat /etc/zzz
Hello...This is etc/zzz file.
Malicious Data
[01/30/21]seed@PES2201800618_AmruthaBS:~$
```