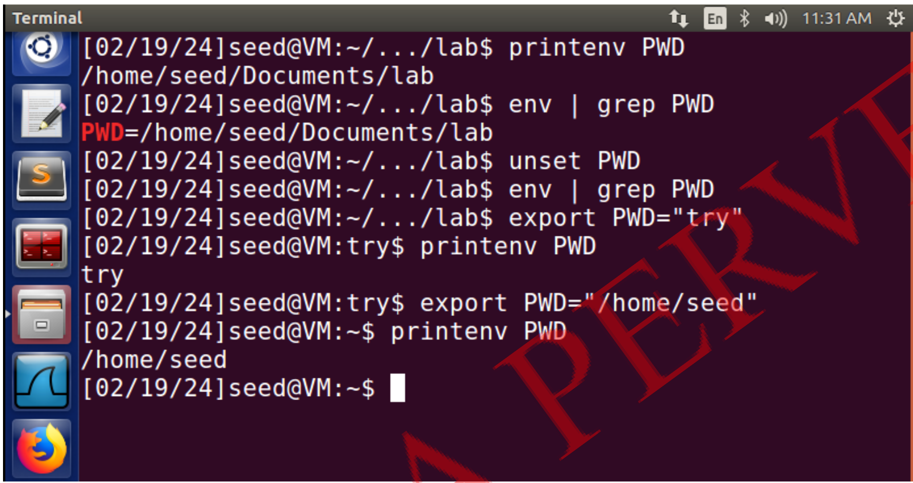
Environment Variable and Set-UID Program Lab

Task 1: Manipulating environment variables

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

Use printenv or env command to print out the environment variables. There are some particular environment variables, such as PWD, we can use "printenv PWD" or "env | grep PWD".

Use export and unset to set or unset environment variables.



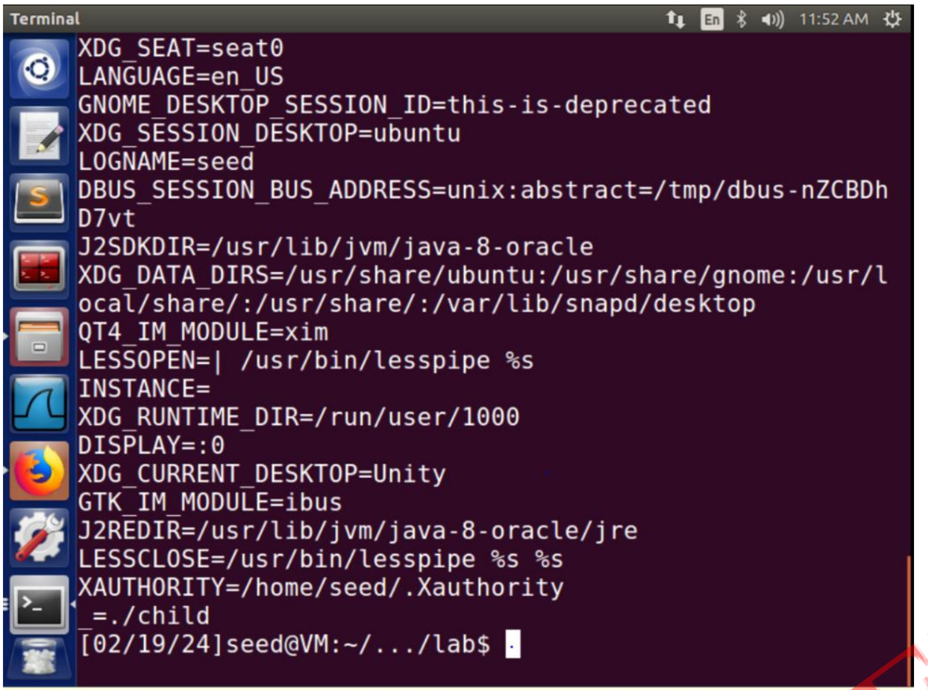
Task 2: Inheriting environment variables from parents

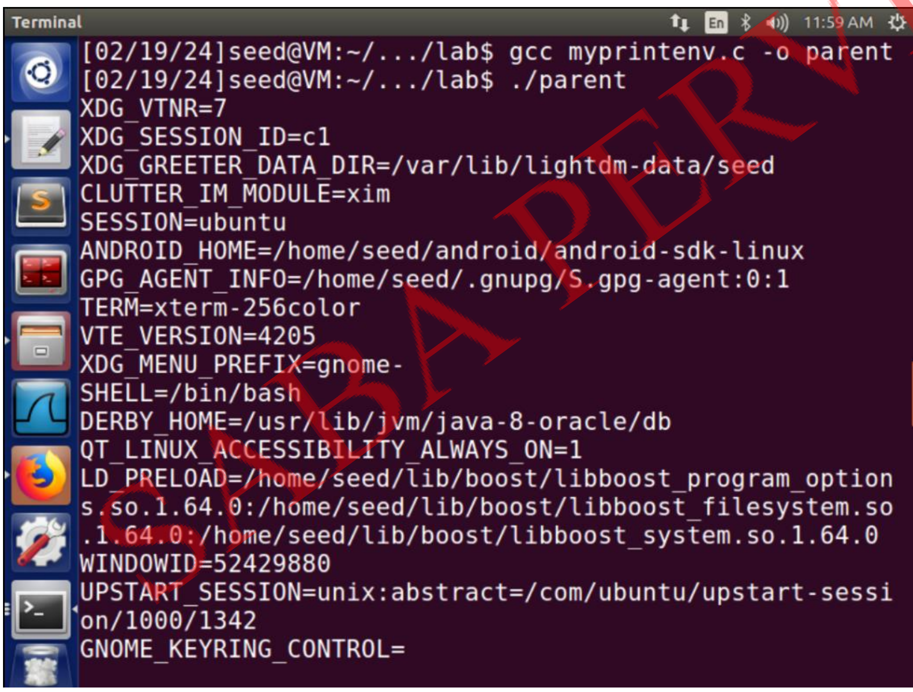
In this task, we study how environment variables are inherited by child processes from their parents. In Unix, fork() creates a new process by duplicating the calling process. The new process, referred to as the child, is an exact duplicate of the calling process, referred to as the parent; however, several things are not inherited by the child (please see the manual of fork() by typing the following command: man fork). In this task, we would like to know whether the parent's environment variables are inherited by the child process or not.

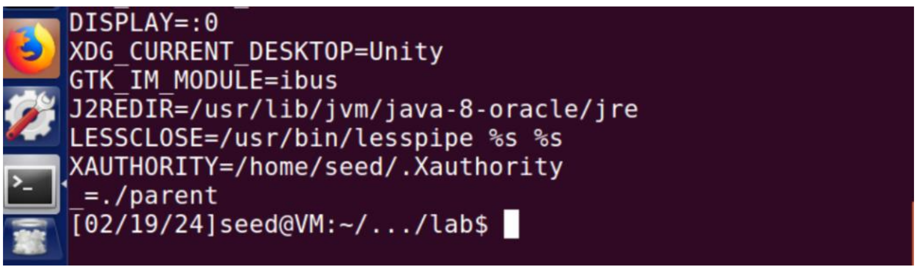
Code used for this task:

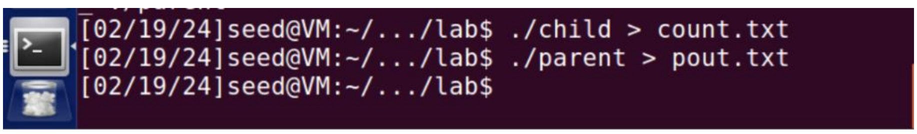
```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

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);
  }
}
```

```
[02/19/24]seed@VM:~/.../lab$ gcc myprintenv.c -o child
[02/19/24]seed@VM:~/.../lab$ ./child
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
XDG_MENU_PREFIX=gnome-
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_option
s.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=52429880
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-sessi
on/1000/1342
GNOME_KEYRING_CONTROL=
```


Terminal
11:52 AM
XDG_SEAT=seat0
LANGUAGE=en_US
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
XDG_SESSION_DESKTOP=ubuntu
LOGNAME=seed
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-nZCBDh
D7vt
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/l
ocal/share/:/usr/share/:/var/lib/snapd/desktop
QT4_IM_MODULE=xim
LESSOPEN=| /usr/bin/lesspipe %s
INSTANCE=
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
_=./child
[02/19/24]seed@VM:~/.../lab$


Terminal
11:59 AM
[02/19/24]seed@VM:~/.../lab$ gcc myprintenv.c -o parent
[02/19/24]seed@VM:~/.../lab$ ./parent
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
XDG_MENU_PREFIX=gnome-
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_option
s.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=52429880
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-sessi
on/1000/1342
GNOME_KEYRING_CONTROL=


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
_=./parent
[02/19/24]seed@VM:~/.../lab$


[02/19/24]seed@VM:~/.../lab$ ./child > count.txt
[02/19/24]seed@VM:~/.../lab$ ./parent > pout.txt
[02/19/24]seed@VM:~/.../lab$

```
[02/19/24]seed@VM:~/.../lab$ diff count.txt pout.txt
67c67
< _=./child
---
> _=./parent
[02/19/24]seed@VM:~/.../lab$
```

The program prints all environment variables in the parent or the child process

From the screenshot above, Parent process and child process have the same environment variables since they have the same output.

Task 3: Environment variables and execve()

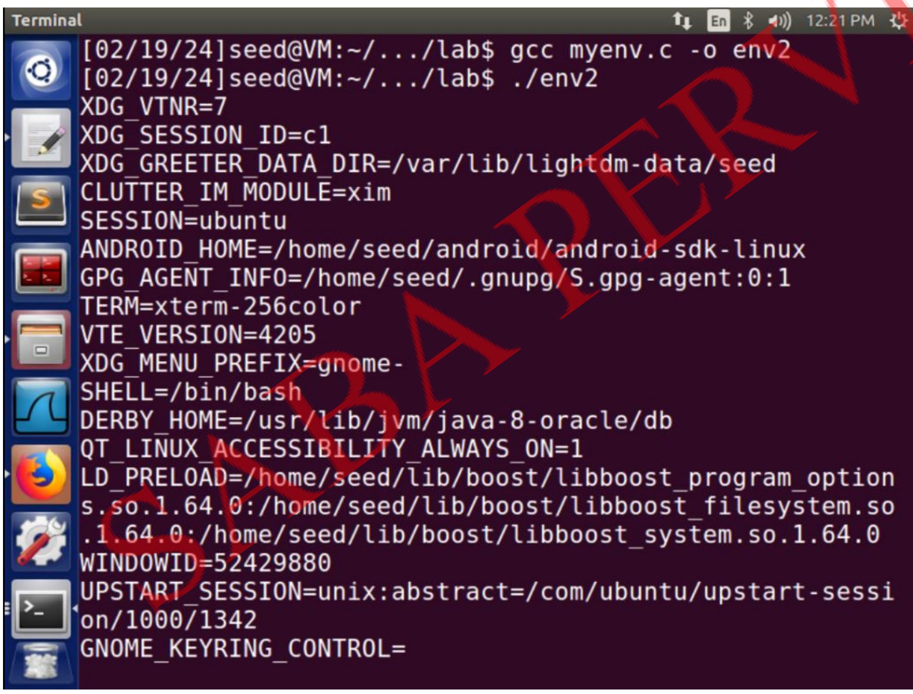
In this task, we study how environment variables are affected when a new program is executed via execve(). The function execve() calls a system call to load a new command and execute it; this function never returns. No new process is created; instead, the calling process's text, data, bss, and stack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

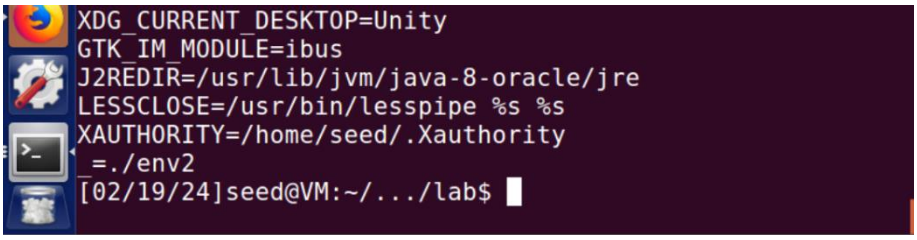
Code used for this task:

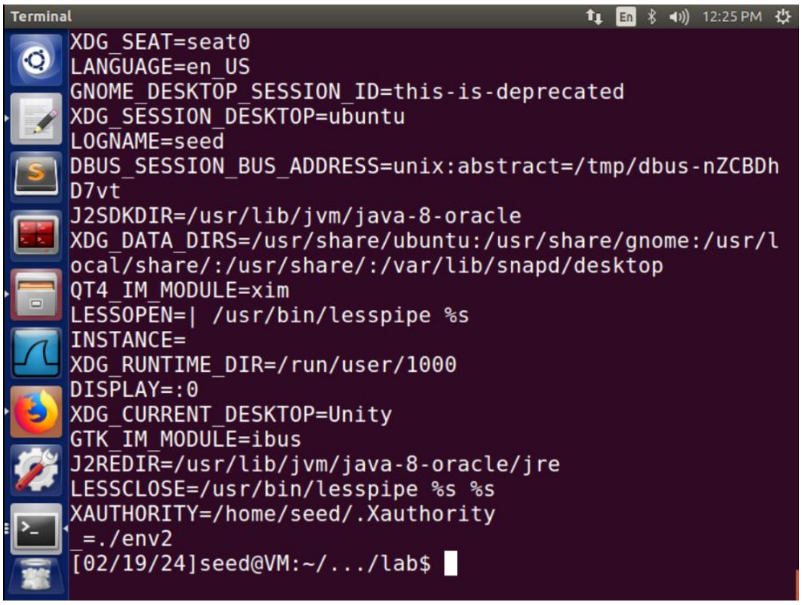
```
myprintenv.c

#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 ;
}
```

```
[02/19/24]seed@VM:~/.../lab$ gcc myenv.c -o env1
[02/19/24]seed@VM:~/.../lab$ ./env1
[02/19/24]seed@VM:~/.../lab$
```


*myenv.c (~/Downloads/lab) - gedit
Open
myprintenv.c
#include <unistd.h>
extern char **environ;
int main()
{
char *argv[2];
argv[0] = "/usr/bin/env";
argv[1] = NULL;
execve("/usr/bin/env", argv, environ);
return 0 ;
}


Terminal
12:21 PM
[02/19/24]seed@VM:~/.../lab$ gcc myenv.c -o env2
[02/19/24]seed@VM:~/.../lab$ ./env2
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
XDG_MENU_PREFIX=gnome-
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_option
s.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=52429880
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-sessi
on/1000/1342
GNOME_KEYRING_CONTROL=


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
_=./env2
[02/19/24]seed@VM:~/.../lab$

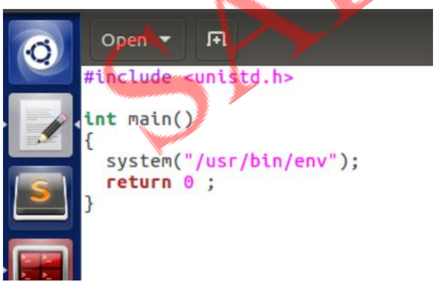


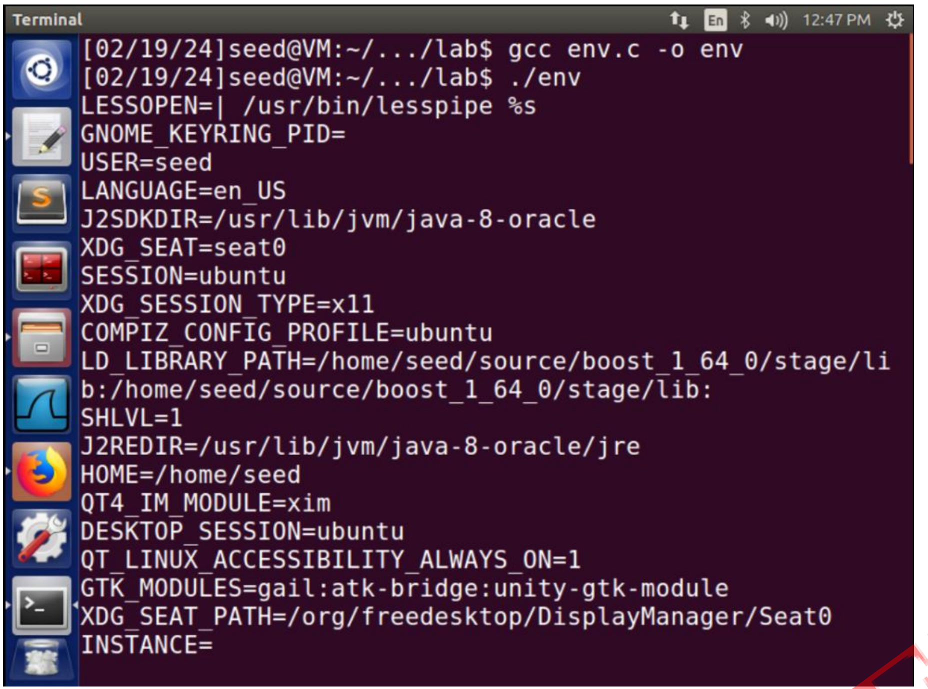
The program compiled in step 1 (env1) prints nothing, whereas the program in step 2 (env2) prints all environment variables. Therefore we know the environment variables are not inherited in the program. Instead, the program get them by external pointer `environ`.

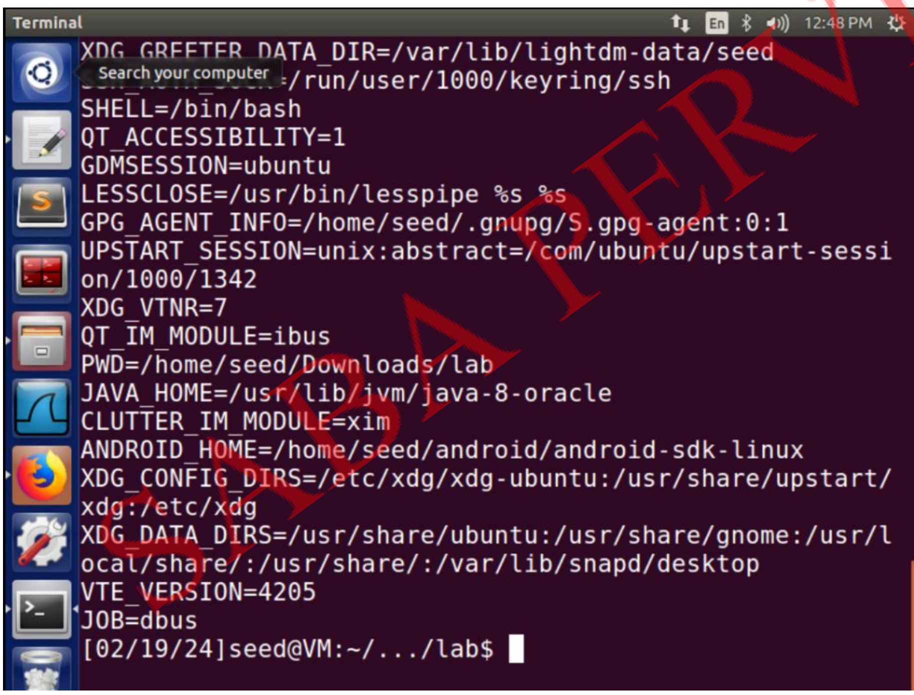
Task 4: Environment variables and system()

In this task, we study how environment variables are affected when a new program is executed via the system() function.

Code used for this task:





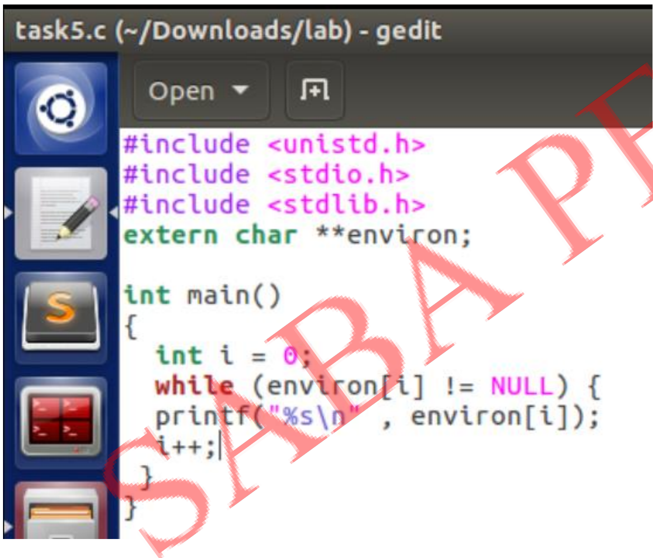


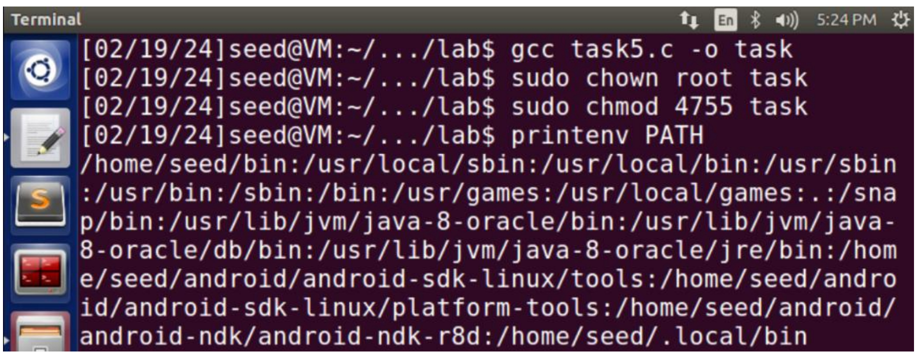
From the screenshot, the program prints all environment variables under `/usr/bin/env`. Therefore the `system()` function has passed environment variables to `/bin/sh`.

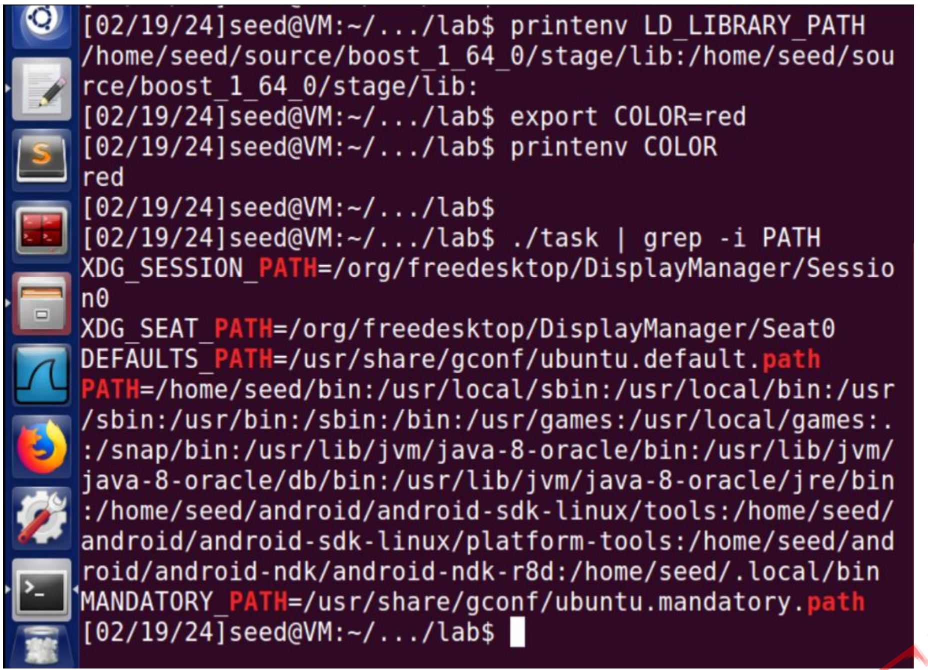
Task 5: Environment variable and Set-UID Programs

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.

Code used for this task:





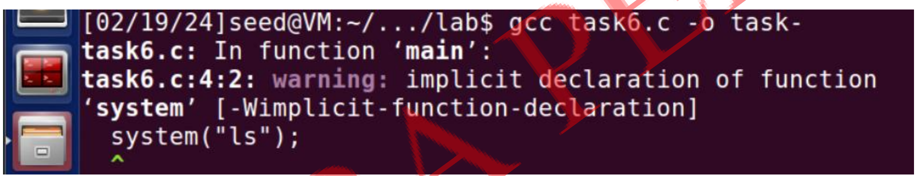


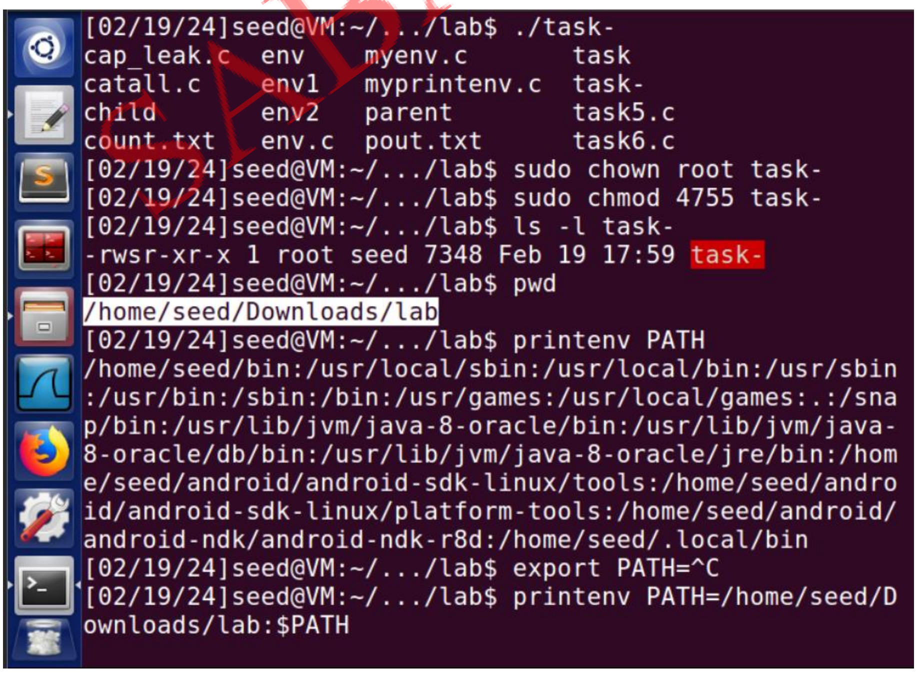
As shown in the screenshot, the `Path` is set to `/bin:/home/bin:/usr/bin`, but `LD_LIBRARY_PATH` is not printed, which means it didn't enter the child process.

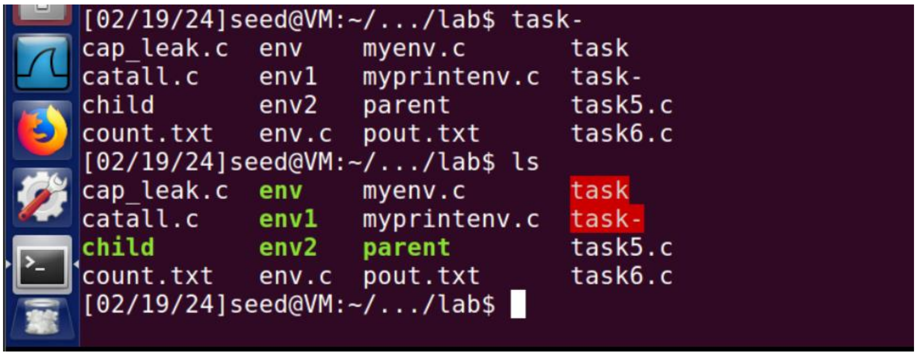
Task 6: The PATH Environment variable and Set-UID Programs

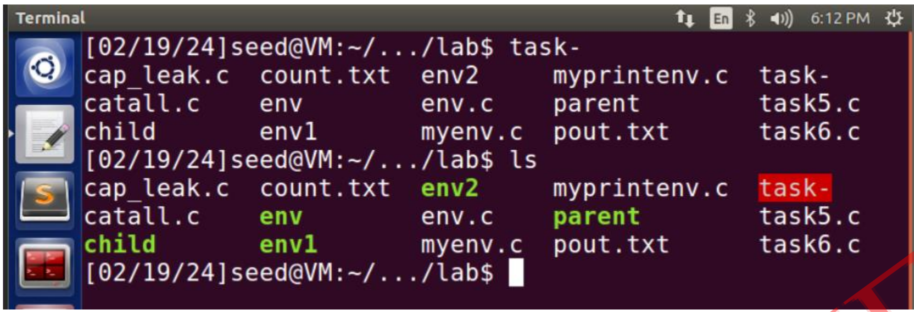
Because of the shell program invoked,
calling system() within a Set-UID program is quite dangerous. This is because the actual behavior of the shell program can be affected by environment variables, such as PATH; these environment variables are provided by the user, who may be malicious. By changing these variables, malicious users can control the behavior of the Set-UID program.

Code used for this task:









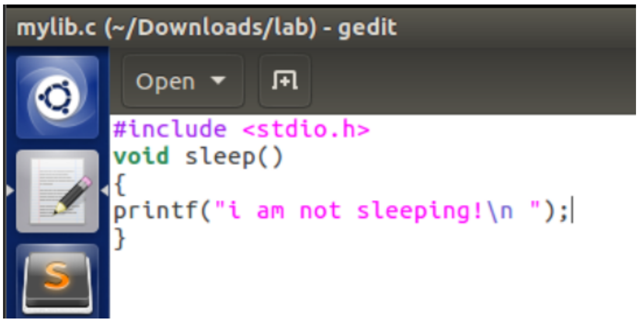
The program can gain root privilege if we copy `\bin\sh` to current directory and add current directory to `PATH`.

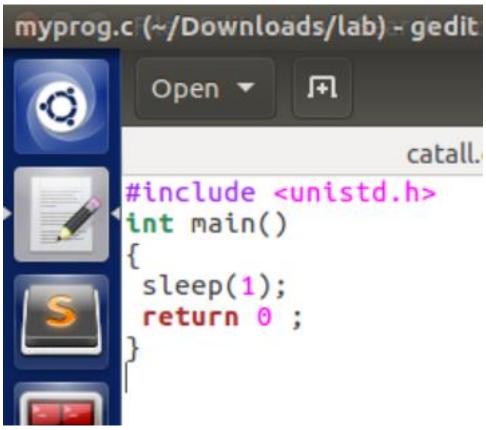
Task 7: The LD_PRELOAD environment variable and Set-UID Programs

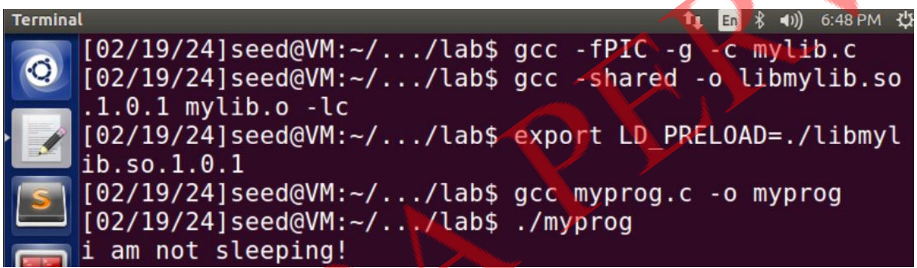
In this task, we study how Set-UID programs deal with some of the environment variables. Several environment variables, including LD_PRELOAD, LD_LIBRARY_PATH, and other LD_* influence the behavior of dynamic loader/linker. A dynamic loader/linker is the part of an operating system (OS) that loads (from persistent storage to RAM) and links the shared libraries needed by an executable at run time.

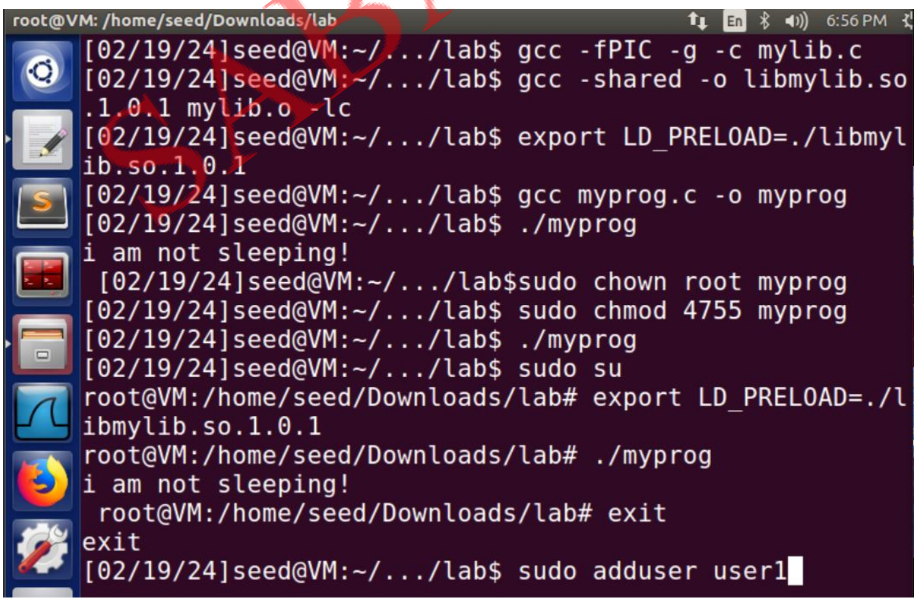
In Linux, ld.so or ld-linux.so, are the dynamic loader/linker. Among the environment variables that affect their behaviors, LD_LIBRARY_PATH and LD_PRELOAD are the two that we are concered in this lab. In Linux, LD_LIBRARY_PATH is a colon-separated set of directories where libraries should be searched 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, we will only study LD_PRELOAD.

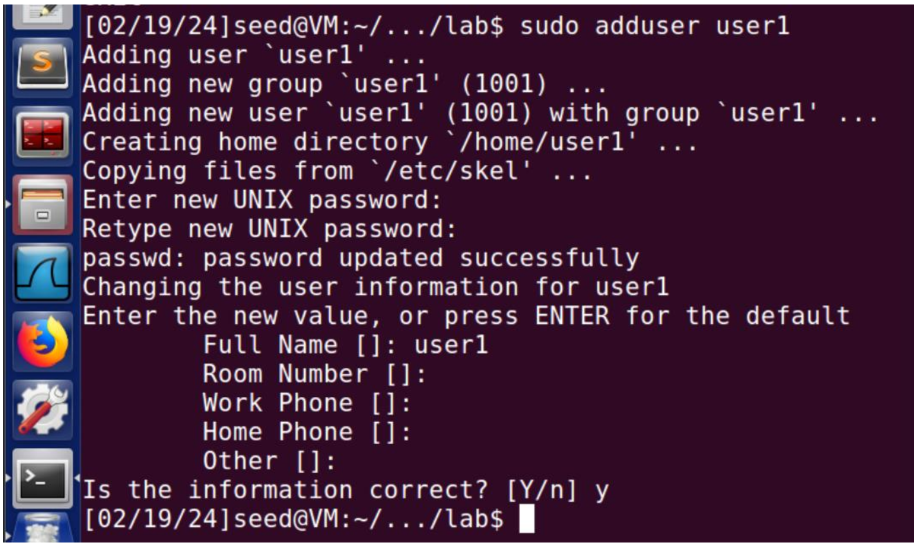
Code used for this task:

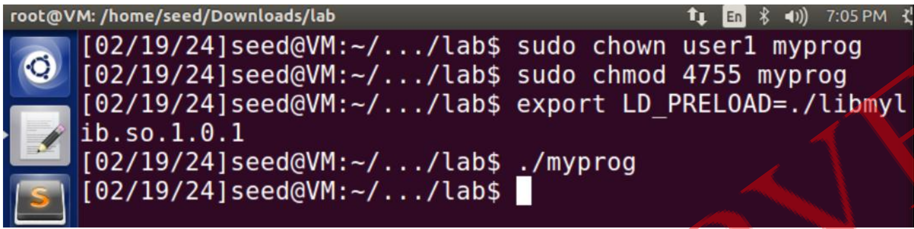












Make `myprog` a regular program, and run it as a normal user.

The program will use the environment variable set by user and call the `sleep()` in `libmylib.so.1.0.1`.

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

In this case, the program will ignore `LD_PRELOAD` set by user and use the default `sleep()` function.

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

In this case, exported `LD_PRELOAD` dominants. The program will use the `sleep()` in `libmylib.so.1.0.1`.

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.

In this case, LD_PRELOAD is not overwritten.

In conclusion, only the program owner can run the program with overwritten environment variables.

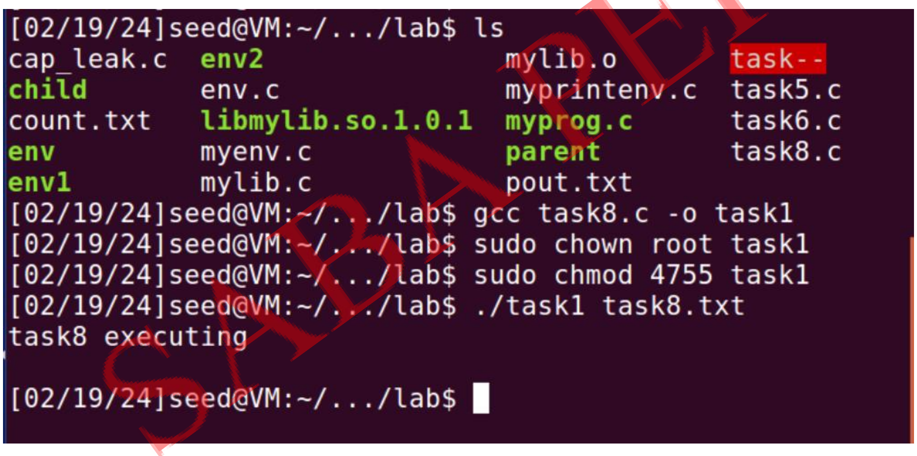
Task 8: Invoking external program using system() versus execve()

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.

Code used for this task:

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.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 the followings.
  //system(command);
   execve(v[0], v, NULL);

  return 0 ;
}
```

```
[02/19/24]seed@VM:~/.../lab$ gcc task8.c -o task--
[02/19/24]seed@VM:~/.../lab$ sudo chown root task--
[02/19/24]seed@VM:~/.../lab$ sudo chmod 4755 task--
[02/19/24]seed@VM:~/.../lab$ ls -l task--
-rwsr-xr-x 1 root seed 7544 Feb 19 19:16 task--
```



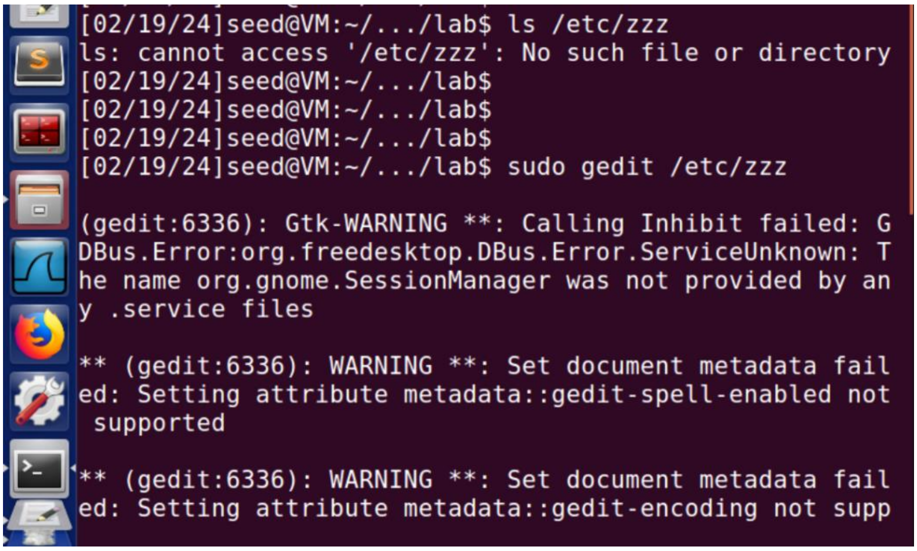
we can insert a command after ; to modify protected le. However, in second scenario, `execve()` sees the argument as a whole name so we cannot make exploit on that.

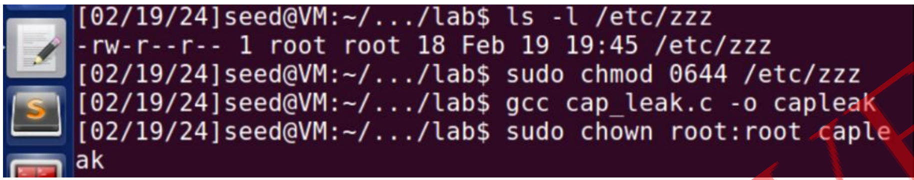
Task 9: Capability Leaking

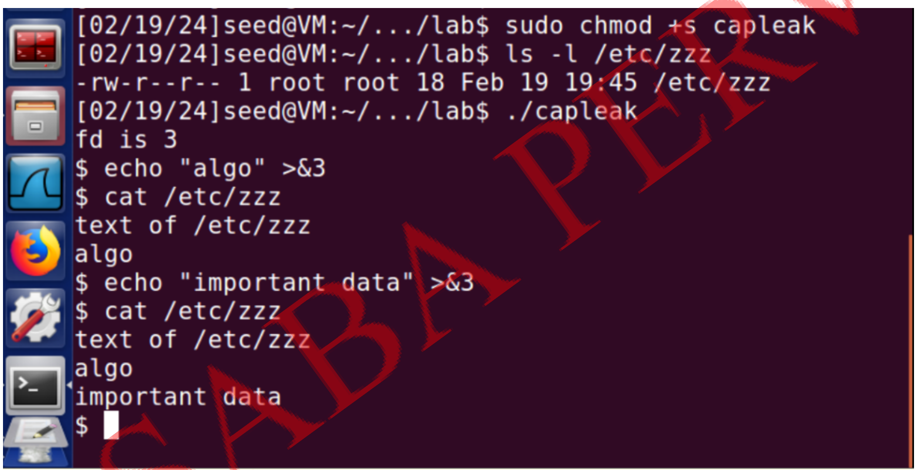
To follow the Principle of Least Privilege, Set-UID programs often permanently relinquish their root privileges if such privileges are not needed anymore. Moreover, sometimes, the program needs to hand over its control to the user; in this case, root privileges must be revoked. The setuid() system call 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 .

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.

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

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);
}
```







From the screenshot above, we can see that the le has been modied. This is because `zzz` is opened before `setuid()` and has root privilege.