```
LAB: Environment Variables and Set-UID Programs
***********************************************

Lab Files:  myprintenv.c, myenv.c, system_env.c, setuid_env.c
            myls.c, ls.c, catall.c, cap_leak.c


Task 1: Manipulating Environment Variables and Shell Variables
==============================================================

#### Printing environment variables

$ printenv

$ printenv PWD

$ printenv | grep PWD

$ env

$ env | grep PWD

#### Printing shell variables

$ set

$ set | grep PWD

#### Accessing value of any shell or environment variable

$ echo $PWD

#### Creating shell variables

$ VAR1='Hello World!'
$ echo $VAR1
$ set | grep VAR1

$ VAR2='Bye World!'
$ echo $VAR2
$ set | grep VAR2

#### Creating environment variables

$ printenv VAR1         # no output displayed; VAR1 is not an environment variable yet

$ export VAR1           # turns it into an environment variable

$ printenv VAR1         # should display the value of VAR1

$ export | grep VAR1    # should display the value of VAR1

#### Demoting and unsetting variables

$ export -n VAR1        # change it back into a shell variable

$ printenv VAR1         # no output displayed

$ export | grep VAR1    # no output displayed

$ unset VAR1            # unsetting shell or environment variables
$ unset VAR2
$ echo $VAR1
$ echo $VAR2


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

file: myprintenv.c
```

```
- Uncomment call to printenv() for "child process code"
- Comment out call to printenv() for "parent process code"
- Compile and run the program using the commands below

$ gcc myprintenv.c -o myprintenv
$ ./myprintenv > child
$ ls -l child       # verify file is created

- Comment out call to printenv() for "child process code"
- Uncomment call to printenv() for "parent process code"
- Compile and run the program using the commands below

$ gcc myprintenv.c -o myprintenv
$ ./myprintenv > parent
$ ls -l parent

Compare the two output files using the "diff" command to check if the
parent's environment variables are inherited by the child process or not.

$ diff parent child     # there should be no difference between the two files


Task 3: Environment Variables and execve()
==========================================

file: myenv.c

First Run:
    Uncomment the first execve() statement and comment out the remaining two execve()
    statements in the code.

    Compile and run the program using the commands below.

    $ gcc myenv.c -o myenv
    $ ./myenv

    The program should not produce any output.

Second Run:
    Uncomment the second execve() statement and comment out the remaining two execve()
    statements in the code.

    Compile and run the program using the commands below.

    $ gcc myenv.c -o myenv
    $ ./myenv

    The program should output a list of environment variables and their values.

Third Run:
    Uncomment the third execve() statement and comment out the remaining two execve()
    statements in the code.

    Compile and run the program using the commands below.

    $ gcc myenv.c -o myenv
    $ ./myenv

    The program should output only two environment variables - AAA and BBB


Task 4: Environment Variables and system()
==========================================

file: system_env.c

Note: system(command) actually executes "/bin/sh -c command"

$ gcc system_env.c -o system_env
```

```
$ ./system_env

The program should output a list of environment variables and their values.


Task 5: Environment Variables and Set-UID Programs
==================================================

file: setuid_env.c

Compile setuidenv.c and make the executable a root owned set-uid program

$ gcc -o setuid_env setuid_env.c
$ sudo chown root setuid_env
$ sudo chmod 4755 setuid_env

Let's modify (and/or create) the following environment variables and export them.

    PATH
    LD_LIBRARY_PATH
    MY_NAME

$ OLD_PATH=$PATH
$ export PATH=$PWD:$PATH
$ echo $PATH

$ OLD_LD_LIBRARY_PATH=$LD_LIBRARY_PATH
$ export LD_LIBRARY_PATH=$PWD:$LD_LIBRARY_PATH
$ echo $LD_LIBRARY_PATH

$ export MY_NAME='John Doe'

$ ./setuid_env > setuid_env_result

$ env > env_result

Note that the program setuid_env.c (a set-uid process) will not inherit the
the environment variable LD_LIBRARY_PATH (as shown below):

$ cat setuid_env_result | grep LD_LIBRARY_PATH      # should be no output

$ cat env_result | grep LD_LIBRARY_PATH             # should output one line

$ cat setuid_env_result | grep MY_NAME              # MY_NAME=John Doe

$ cat env_result | grep MY_NAME                     # MY_NAME=John Doe

$ cat setuid_env_result | grep ^PATH                # PATH=...

$ cat env_result | grep ^PATH                       # PATH=...

IMPORTANT: revert changes to PATH and LD_LIBRARY_PATH environment variables

$ PATH=$OLD_PATH
$ LD_LIBRARY_PATH=$OLD_LD_LIBRARY_PATH


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

files: myls.c and ls.c

Compile myls.c and make the executable a root owned set-uid program

$ gcc -o myls myls.c
$ sudo chown root myls
$ sudo chmod 4755 myls
$ ls -l myls        # myls should appear with red background with "s" bit
$ ./myls            # should see the listing of current directory
```

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

Should produce three line of output as shown below:

```
You are running my ls program!!
My real uid is: 1000
My effective uid is: 1000
```

Let's modify PATH environment variable as follows:

```
$ OLD_PATH=$PATH
$ export PATH=$PWD:$PATH

$ ./myls        // runs ls program in the current directory in non-privileged mode
```

system("ls") call in myls.c executes /bin/sh program first, and then asks this shell program to run the "ls" command. In Ubuntu 20.04 (and several earlier versions), /bin/sh is actually a symbolic link to /bin/dash. The dash shell program has a countermeasure that prevents itself from being executed in a Set-UID process. The shell program /bin/zsh does not have such a countermeasure.

Switch the default shell from /bin/dash to /bin/zsh:

```
$ sudo ln -sf /bin/zsh /bin/sh

$ ./myls        // now runs ls program with root privilege with the following output
                // notice the effective uid is now 0

You are running my ls program!!
My real uid is: 1000
My effective uid is: 0
```

IMPORTANT: Let's revert the changes made using the commands below

```
$ sudo ln -sf /bin/dash /bin/sh

$ PATH=$OLD_PATH
```

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

!!!! SKIPPING THIS TASK !!!!

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

file: catall.c

Uncomment "system(command)" statement and comment out "execve(v[0], v, NULL)" statement.
Compile and make it a root-owned set-uid program

```
$ gcc catall.c -o catall
$ sudo chown root catall
$ sudo chmod 4755 catall
```

Make two new files: "seedfile" owned by seed user and "rootfile" owned by root user

```
$ echo "This file is owned by seed user" > seedfile
$ cat seedfile

$ echo "This file is owned by root user" > rootfile
$ cat rootfile
$ sudo chown root rootfile

$ ./catall "seedfile;rm rootfile"

$ ls        # rootfile is removed
```

```
Comment out "system(command)" statement and uncomment "execve(v[0], v, NULL)" statement.
Compile and make it a root-owned set-uid program

$ gcc catall.c -o catall
$ sudo chown root catall
$ sudo chmod 4755 catall

$ echo "This file is owned by root user" > rootfile
$ sudo chown root rootfile

$ ./catall "seedfile;rm rootfile"   # should generate an error


Task 9: Capability Leaking
==========================

files: cap_leak.c

Create the file /etc/zzz:

$ ls -ld /etc                    # check directory permissions
$ sudo touch /etc/zzz
$ cat /etc/zzz                   # empty file is created
$ echo aaaaaaa > /etc/zzz        # /etc/zzz: permission denied

Compile cap_leak.c and make the executable a root-owned set-uid program

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

$ ./cap_leak
fd is 3
$ echo aaaaaaa > /etc/zzz        ; permission denied
$ echo aaaaaaa >& 3
$ cat /etc/zzz
aaaaaaa
$ exit

Note: To mitigate capability leaking, the file descriptor should also be closed
along with downgrading the privileges.
```