

Environment Variable and Set-UID Program Lab

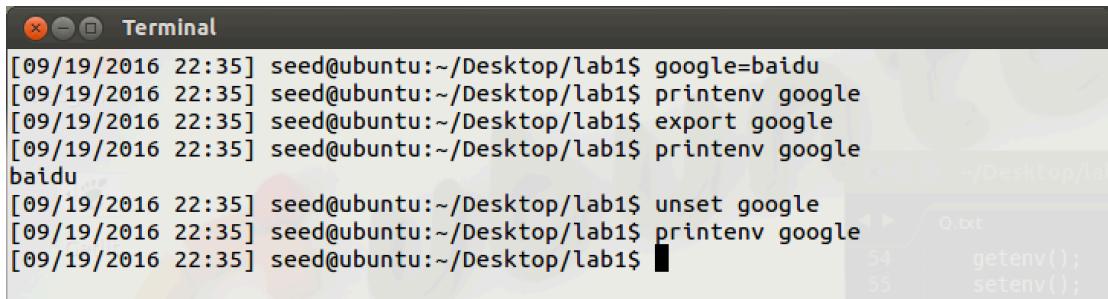
Task 1: manipulating environment variables

```
[09/19/2016 22:12] seed@ubuntu:~/Desktop/lab1$ env
[09/19/2016 22:12] seed@ubuntu:~/Desktop/lab1$ env
SSH_AGENT_PID=2381
GPG_AGENT_INFO=/tmp/keyring-BHBFZ/gpg:0:1
TERM=xterm
SHELL=/bin/bash
XDG_SESSION_COOKIE=6da3e071019f67095bc4c5e900000002-1472674553.339500-1947096018
WINDOWID=62916518
OLDPWD=/home/seed/Desktop
GNOME_KEYRING_CONTROL=/tmp/keyring-BHBFZ
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;01:cd
=40;33;01:or=40;31;01:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=0
1;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31:*.tar.=01;31:*.lzh=01;31:*.lzma=01;31:*
.tlz=01;31:*.txz=01;31:*.lzp=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lz
=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.d
eb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31
*:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.jpg=01;35:*.jpeg=0
1;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgn=01;35:*.ppm=01;35:*.tga=01;35:*.x
bm=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:*.mk
v=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;3
5:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=0
1;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:*.axv=01;35:*.anx=01;35:*.ogg=01;35:*
```

Fig 1. env command print out environment variables

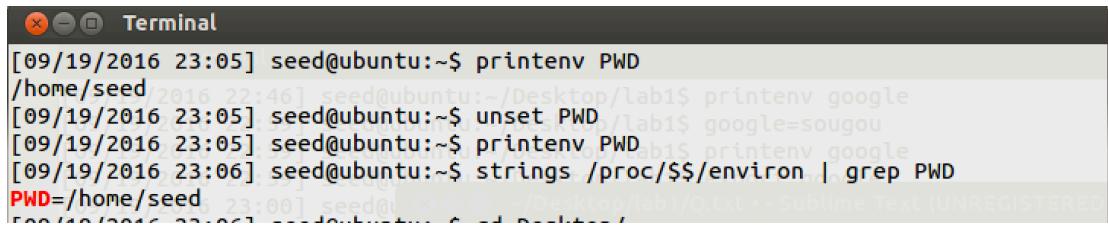
```
[09/19/2016 22:24] seed@ubuntu:~/Desktop/lab1$ env | grep g=F
g=F
[09/19/2016 22:24] seed@ubuntu:~/Desktop/lab1$ printenv g
F libcap2.22
[09/19/2016 22:24] seed@ubuntu:~/Desktop/lab1$
```

Fig 2. Specify the environment variable and print out



```
[09/19/2016 22:35] seed@ubuntu:~/Desktop/lab1$ google=baidu
[09/19/2016 22:35] seed@ubuntu:~/Desktop/lab1$ printenv google
[09/19/2016 22:35] seed@ubuntu:~/Desktop/lab1$ export google
[09/19/2016 22:35] seed@ubuntu:~/Desktop/lab1$ printenv google
baidu
[09/19/2016 22:35] seed@ubuntu:~/Desktop/lab1$ unset google
[09/19/2016 22:35] seed@ubuntu:~/Desktop/lab1$ printenv google
[09/19/2016 22:35] seed@ubuntu:~/Desktop/lab1$
```

Fig 3. export and unset environment variables



```
[09/19/2016 23:05] seed@ubuntu:~$ printenv PWD
/home/seed
[09/19/2016 23:05] seed@ubuntu:~$ unset PWD
[09/19/2016 23:05] seed@ubuntu:~$ printenv PWD
[09/19/2016 23:05] seed@ubuntu:~$ strings /proc/$$/environ | grep PWD
PWD=/home/seed
[09/19/2016 23:05] seed@ubuntu:~$
```

Fig 4. Unset variable effects shell variable

Observation:

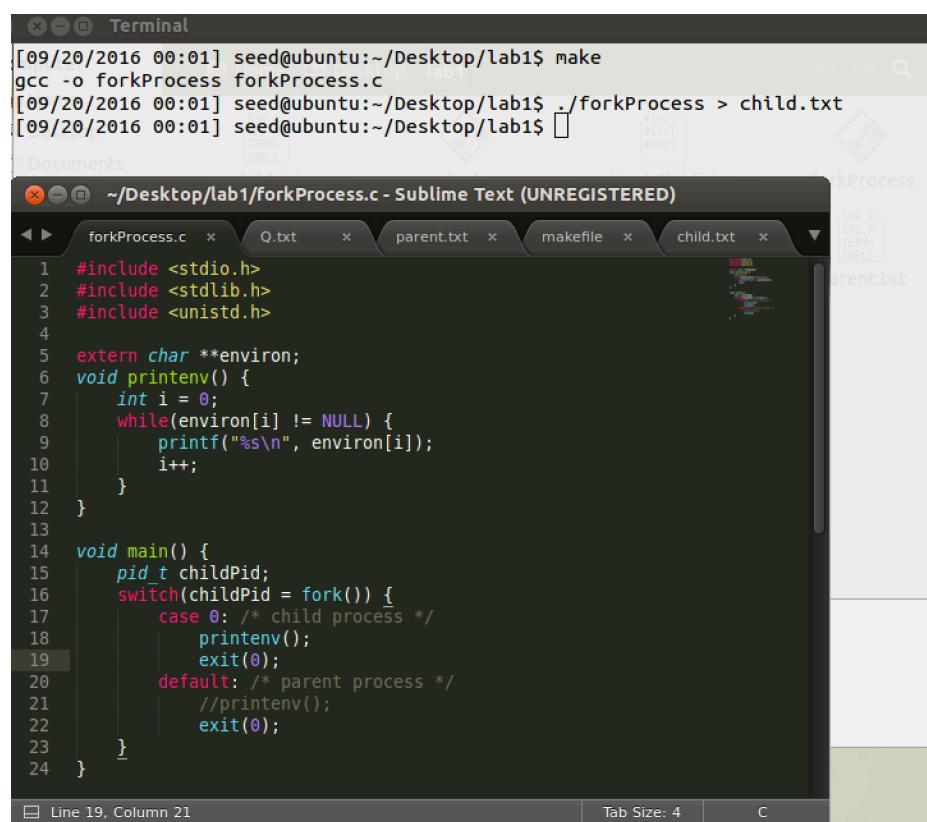
- 1.1 From Fig 1, we can know ‘printenv’ and ‘env’ command they all print out all the environment variables, actually these printing out are shell variables. And from fig 2, we know that ‘printenv [variablename]’ and ‘env | grep [variablename]’ can print out specify environment variable.
- 1.2 From fig 3, we can use printenv [variablename] print out variable after we export the variable. As google, at first I did not export it. I cannot search it using printenv google, but after I export google, I can print it out using printenv google. The idea for this result is that shell gets deep copy from environment variables, and then I created google=baidu as shell variable, and then use export to make sure child process inherits this variable as environment variable.
- 1.3 As fig 3, we can use unset [variablename] to remove this variable from shell variables. Like fig 4, unset can remove shell variable but it cannot remove the environment variable.

Explanation:

1.4 For Fig 3, as google, at first I did not export it. I cannot search it using printenv google, but after I export google, I can print it out using printenv google. The idea for this result is that shell gets deep copy from environment variables, and then I created google=baidu as shell variable, and then use export to make sure child process inherits this variable as environment variable.

1.5 Shell can deep copy environment variables called shell variables. Then child process can access these variables after they were exported. Command printenv is kind of child process, so export should be used.

Task2: Inheriting environment variables from parents



```
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ make
gcc -o forkProcess forkProcess.c
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ ./forkProcess > child.txt
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ 

~/Documents/forkProcess
~/Desktop/lab1/forkProcess.c - Sublime Text (UNREGISTERED)
forkProcess.c * Q.txt * parent.txt * makefile * child.txt *
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 extern char **environ;
6 void printenv() {
7     int i = 0;
8     while(environ[i] != NULL) {
9         printf("%s\n", environ[i]);
10        i++;
11    }
12 }
13
14 void main() {
15     pid_t childPid;
16     switch(childPid = fork()) {
17         case 0: /* child process */
18             printenv();
19             exit(0);
20         default: /* parent process */
21             //printenv();
22             exit(0);
23     }
24 }
```

Line 19, Column 21 Tab Size: 4 C

Fig 2.1 fork child process print environment variables

The terminal window shows the following command sequence:

```
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ make
gcc -o forkProcess forkProcess.c
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ ./forkProcess > child.txt
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$
```

The Sublime Text editor window titled "child.txt" displays the following environment variables:

```
1 SSH_AGENT_PID=2381
2 GPG_AGENT_INFO=/tmp/keyring-BHBFz/gpg:0:1
3 TERM=xterm
4 SHELL=/bin/bash
5 XDG_SESSION_COOKIE=6da3e071019f67095bc4c5e900000002-1472674553.
339500-1947096018
6 WINDOWID=62916518
7 OLDPWD=/home/seed/Desktop
8 GNOME_KEYRING_CONTROL=/tmp/keyring-BHBFz
9 USER=seed
10 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: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:
*:arj=01;31:*.tarz=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*
txz=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31
*:lz=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2
=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01
*:ear=01;31:*.sar=01;31:*.rar=01;31:*.ace=01;31:*.zoo=01;31
*:cpio=01;31:*.7z=01;31:*.rz=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=0
1;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;3
5:*.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:*.rmv
```

Fig 2.2 result of fork child process print environment variables

The terminal window shows the following command sequence:

```
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ make
gcc -o forkProcess forkProcess.c
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ ./forkProcess > child.txt
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ make clean
rm forkProcess
[09/20/2016 00:06] seed@ubuntu:~/Desktop/lab1$ make
gcc -o forkProcess forkProcess.c
[09/20/2016 00:06] seed@ubuntu:~/Desktop/lab1$ ./forkProcess > parent.txt
[09/20/2016 00:06] seed@ubuntu:~/Desktop/lab1$
```

The Sublime Text editor window titled "forkProcess.c" displays the following C code:

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

Fig 2.3 parent process print environment variables

```
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ make
gcc -o forkProcess forkProcess.c
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ ./forkProcess > child.txt
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ make clean
rm forkProcess
[09/20/2016 00:06] seed@ubuntu:~/Desktop/lab1$ make
gcc -o forkProcess forkProcess.c
[09/20/2016 00:06] seed@ubuntu:~/Desktop/lab1$ ./forkProcess > parent.txt
[09/20/2016 00:06] seed@ubuntu:~/Desktop/lab1$
```

Sublime Text window content:

```
1 SSH_AGENT_PID=2381
2 GPG_AGENT_INFO=/tmp/keyring-BHBFZ/gpg:0:1
3 TERM=xterm
4 SHELL=/bin/bash
5 XDG_SESSION_COOKIE=6da3e071019f67095bc4c5e900000002-1472674553.
339500-1947096018
6 WINDOWID=62916518
7 OLDPWD=/home/seed/Desktop
8 GNOME_KEYRING_CONTROL=/tmp/keyring-BHBFZ
9 USER=seed
10 LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;
35:bd=40;33:cd=40;33:or=40;31: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:
*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*
txz=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31
*:*.lz=01;31:*.xz=01;31:*.bz=01;31:*.bz=01;31:*.tbz=01;31:*.tbz
=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01
;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.ace=01;31:*.zoo=01;31
*:*.cpio=01;31:*.7z=01;31:*.rz=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=0
1;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;3
5:*.ogg=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:
```

Fig 2.4 result of parent process print environment variables

```
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ make
gcc -o forkProcess forkProcess.c
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ ./forkProcess > child.txt
[09/20/2016 00:01] seed@ubuntu:~/Desktop/lab1$ make clean
rm forkProcess
[09/20/2016 00:06] seed@ubuntu:~/Desktop/lab1$ make
gcc -o forkProcess forkProcess.c
[09/20/2016 00:06] seed@ubuntu:~/Desktop/lab1$ ./forkProcess > parent.txt
[09/20/2016 00:06] seed@ubuntu:~/Desktop/lab1$ diff parent.txt child.txt
[09/20/2016 00:10] seed@ubuntu:~/Desktop/lab1$
```

Fig 2.5 compare child.txt and parent.txt

Observation:

2.1 From Fig 2.1, and Fig 2.2 we can get information that we use fork function to make child process out put some environment variables and wrote them in child.txt.

2.2 From Fig 2.3, and Fig 2.4 we can get information that we use parent process out put some environment variables wrote them in parent.txt.

2.3 As Fig 2.5, we found that contents of child.txt and parent.txt have no difference.

Explanation:

2.4 No difference between child and parent process variables means that fork() can produce child process, and child process could inherit environment variables from parent process.

Task3: Environment variables and execve()

The screenshot shows a terminal window and a Sublime Text editor side-by-side.

In the terminal window (top), the command `make` is run to build `execve.c` into `execve`. Then, the program is executed with `./execve`. The output shows the program running successfully.

```
[09/20/2016 00:45] seed@ubuntu:~/Desktop/lab1$ make
gcc -o execve execve.c
[09/20/2016 00:45] seed@ubuntu:~/Desktop/lab1$ ./execve
[09/20/2016 00:45] seed@ubuntu:~/Desktop/lab1$ 
```

In the Sublime Text editor (bottom), the file `execve.c` is open. The code defines a main function that calls `execve` with the argument `"/usr/bin/env"`. The third argument is explicitly set to `NULL`.

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 extern char **environ;
5
6 int main() {
7     char *argv[2];
8     argv[0] = "/usr/bin/env";
9     argv[1] = NULL;
10
11     execve(argv[0], argv, NULL);
12
13     return 0;
14 }
```

Fig 3.1 execve 3rd argument equals NULL

The screenshot shows a desktop environment with two windows. The top window is a terminal window titled 'Terminal' containing command-line output. The bottom window is a Sublime Text editor window titled '~/Desktop/lab1/execve.c - Sublime Text (UNREGISTERED)' displaying C code.

```
[09/20/2016 00:45] seed@ubuntu:~/Desktop/lab1$ make
gcc -o execve execve.c
[09/20/2016 00:45] seed@ubuntu:~/Desktop/lab1$ ./execve
[09/20/2016 00:45] seed@ubuntu:~/Desktop/lab1$ make
gcc -o execve execve.c
[09/20/2016 00:48] seed@ubuntu:~/Desktop/lab1$ ./execve
SSH_AGENT_PID=2381
GPG_AGENT_INFO=/tmp/keyring-BHBPFz/gpg:0:1
TERM=xterm
SHELL=/bin/bash
XDG_SESSION_COOKIE=6da3e071019f67095bc4c5e900000002-1472674553.:.
WINDOWID=62916518

```

```

1 #include <stdlib.h>
2 #include <stdio.h>
3
4 extern char **environ;
5
6 int main() {
7     char *argv[2];
8     argv[0] = "/usr/bin/env";
9     argv[1] = NULL;
10
11     execve(argv[0], argv, environ);
12
13     return 0;
14 }

```

Fig 3.2 execve 3rd argument with environ**Observation:**

3.1 From Fig 3.1, the 3rd argument with NULL, execve function did not print out environment variables.

3.2 From Fig 3.2, the 3rd argument with environ, execve function printed out environment variables.

Explanation:

3.3 3rd argument of execve is used for passing environment variables. We can specify single or several variables or the whole environ. The process which runs execve could get the environment variables according to 3rd argument. So execve could not get environment variables automatically, they must specify as argument.

Task4: Environment variables and system()

The screenshot shows a desktop environment with a terminal window and a code editor.

Terminal:

```
[09/20/2016 01:17] seed@ubuntu:~/Desktop/lab1$ make
gcc -o system system.c
[09/20/2016 01:17] seed@ubuntu:~/Desktop/lab1$ ./system
LESSOPEN=| /usr/bin/lesspipe %s
GNOME_KEYRING_PID=2294
USER=seed
SSH_AGENT_PID=2381
SHLVL=1
OLDPWD=/home/seed/Desktop
HOME=/home/seed
XDG_SESSION_COOKIE=6da3e071019f67095bc4c5e900000002-1472674553.339500-1
DESKTOP_SESSION=ubuntu-2d
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
```

Sublime Text Editor:

File: ~/Desktop/lab1/system.c - Sublime Text (UNREGISTERED)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char* argv[]) {
5     system("/usr/bin/env");
6     return 0;
7 }
```

Fig 4.1 run system.c file

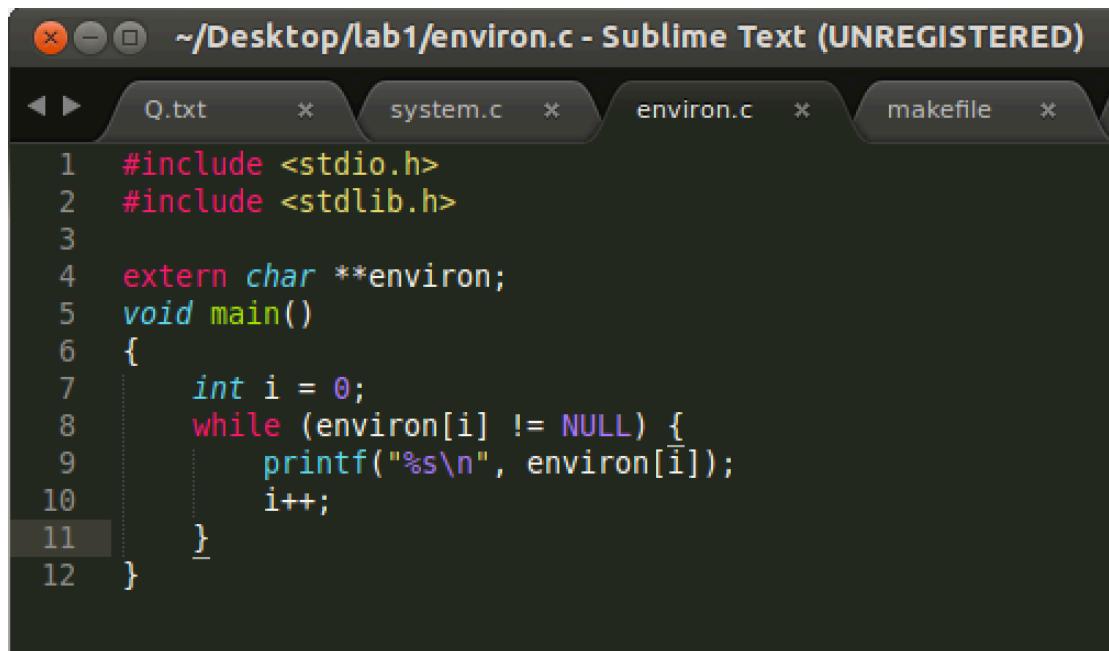
Observation:

4.1 From Fig 4.1, system did not specify any environment variable to this process, but this process could print out all the environment variables.

Explanation:

4.2 Function system call execl() to execute /bin/sh and execve(). Moreover, it pass current environ to execve(). Function system could have environment variables automatically. So it is quit dangerous if used in a privileged program.

Task5: Environment variables and Set-UID Programs

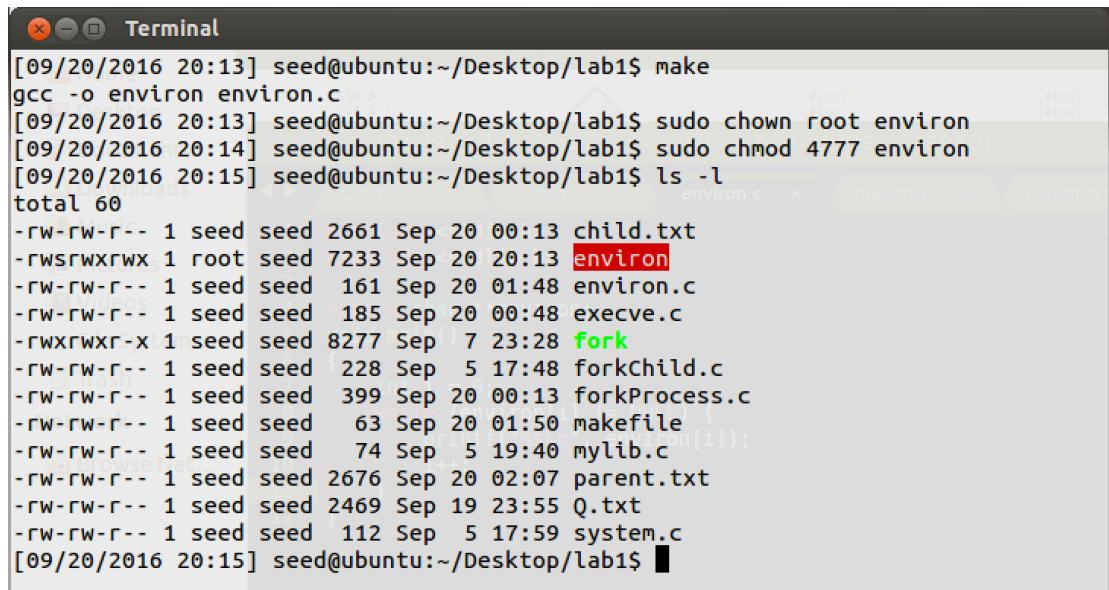


```

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

```

Fig 5.1 print environment variables program



```

[09/20/2016 20:13] seed@ubuntu:~/Desktop/lab1$ make
gcc -o environ environ.c
[09/20/2016 20:13] seed@ubuntu:~/Desktop/lab1$ sudo chown root environ
[09/20/2016 20:14] seed@ubuntu:~/Desktop/lab1$ sudo chmod 4777 environ
[09/20/2016 20:15] seed@ubuntu:~/Desktop/lab1$ ls -l
total 60
-rw-rw-r-- 1 seed seed 2661 Sep 20 00:13 child.txt
-rwSrwxrwx 1 root seed 7233 Sep 20 20:13 environ
-rw-rw-r-- 1 seed seed 161 Sep 20 01:48 environ.c
-rw-rw-r-- 1 seed seed 185 Sep 20 00:48 execve.c
-rwxrwxr-x 1 seed seed 8277 Sep 7 23:28 fork
-rw-rw-r-- 1 seed seed 228 Sep 5 17:48 forkChild.c
-rw-rw-r-- 1 seed seed 399 Sep 20 00:13 forkProcess.c
-rw-rw-r-- 1 seed seed 63 Sep 20 01:50 makefile
-rw-rw-r-- 1 seed seed 74 Sep 5 19:40 mylib.c
-rw-rw-r-- 1 seed seed 2676 Sep 20 02:07 parent.txt
-rw-rw-r-- 1 seed seed 2469 Sep 19 23:55 Q.txt
-rw-rw-r-- 1 seed seed 112 Sep 5 17:59 system.c
[09/20/2016 20:15] seed@ubuntu:~/Desktop/lab1$ █

```

Fig 5.2 change environ owner to root and make it set-uid program

```
[09/20/2016 20:23] seed@ubuntu:~/Desktop/lab1$ ./environ > env.txt
[09/20/2016 20:23] seed@ubuntu:~/Desktop/lab1$ cat env.txt | grep google=
google=baidu
[09/20/2016 20:24] seed@ubuntu:~/Desktop/lab1$ cat env.txt | grep PATH=
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
DEFAULTS_PATH=/usr/share/gconf/ubuntu-2d.default.path
PATH=/seed/Wenbin:.: /usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr
/sbin:/usr/bin:/sbin:/bin:/usr/games/environt
MANDATORY_PATH=/usr/share/gconf/ubuntu-2d.mandatory.path
[09/20/2016 20:24] seed@ubuntu:~/Desktop/lab1$ cat env.txt | grep LD_LIBRARY_PATH
[09/20/2016 20:25] seed@ubuntu:~/Desktop/lab1$
```

Fig 5.3 run ./environ program and make it out write in env.txt and then search variables

Observation:

5.1 Fig 5.1, this is the program to print out environment variables.

5.2 As Fig 5.2, I compiled the environ.c file, and changed owner as root and set this program as a Set-UID program.

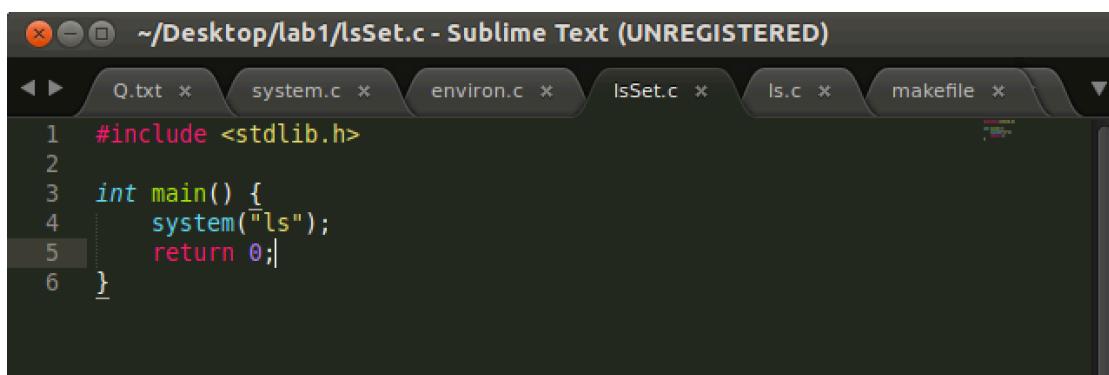
5.3 Run this program. From Fig 5.3, I printed out all the environment variables in env.txt.

Then I searched google, PATH, LD_LIBRARY_PATH variable in env.txt. The result is that we found google and PATH. But, I did not find LD_LIBRARY_PATH variable.

Explanation:

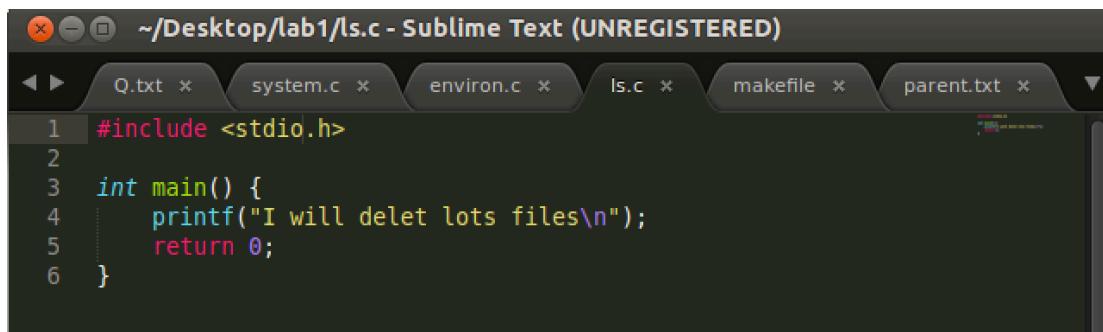
5.4 The reason of why LD_LIBRARY_PATH not be displayed is kind of security feature. This is one. Set-UID program will ignore LD_LIBRARY_PATH. Otherwise, this is a security hole. For example, you can make a self-made program referenced by path of Set-UID program. So that self-made program could run as root privilege.

Task6: The PATH Environment variable and Set-UID Programs



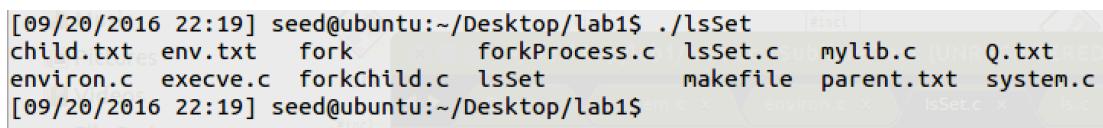
```
~/Desktop/lab1/lsSet.c - Sublime Text (UNREGISTERED)
Q.txt * system.c * environ.c * lsSet.c * ls.c * makefile *
1 #include <stdlib.h>
2
3 int main() {
4     system("ls");
5     return 0;
6 }
```

Fig 6.1 a program call system().



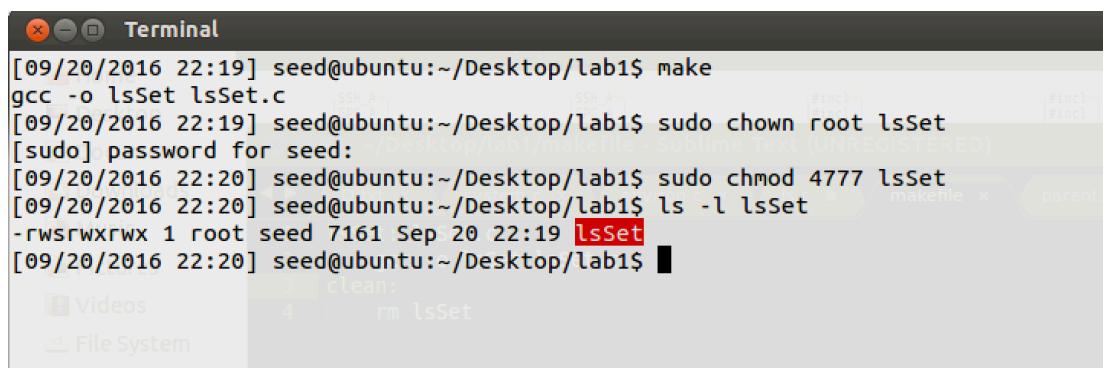
```
#include <stdio.h>
int main() {
    printf("I will delet lots files\n");
    return 0;
}
```

Fig 6.2 a program file named ls.c



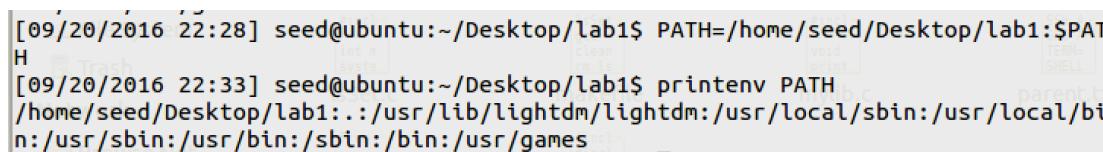
```
[09/20/2016 22:19] seed@ubuntu:~/Desktop/lab1$ ./lsSet
child.txt env.txt fork forkProcess.c lsSet.c mylib.c Q.txt
environ.c execve.c forkChild.c lsSet makefile parent.txt system.c
[09/20/2016 22:19] seed@ubuntu:~/Desktop/lab1$
```

Fig 6.3 run ./lsSet after compile this file



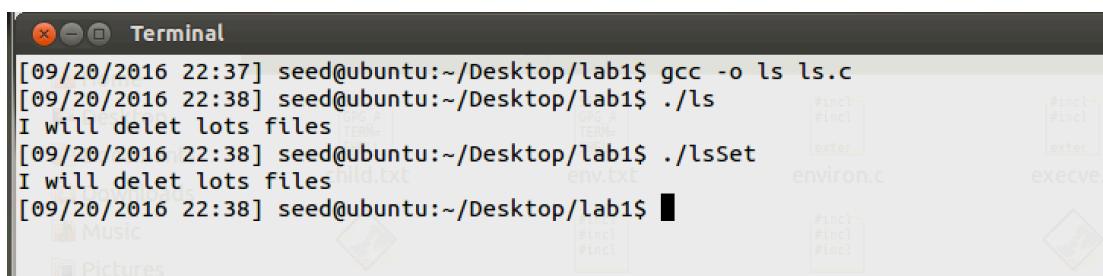
```
Terminal
[09/20/2016 22:19] seed@ubuntu:~/Desktop/lab1$ make
gcc -o lsSet lsSet.c
[09/20/2016 22:19] seed@ubuntu:~/Desktop/lab1$ sudo chown root lsSet
[sudo] password for seed:
[09/20/2016 22:20] seed@ubuntu:~/Desktop/lab1$ sudo chmod 4777 lsSet
[09/20/2016 22:20] seed@ubuntu:~/Desktop/lab1$ ls -l lsSet
-rwsrwxrwx 1 root seed 7161 Sep 20 22:19 lsSet
[09/20/2016 22:20] seed@ubuntu:~/Desktop/lab1$
```

Fig 6.4 compile lsSet.c and change owner to root, and make it Set-UID program



```
[09/20/2016 22:28] seed@ubuntu:~/Desktop/lab1$ PATH=/home/seed/Desktop/lab1:$PATH
[09/20/2016 22:33] seed@ubuntu:~/Desktop/lab1$ printenv PATH
/home/seed/Desktop/lab1:::/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

Fig 6.5 change PATH variable.



```
Terminal
[09/20/2016 22:37] seed@ubuntu:~/Desktop/lab1$ gcc -o ls ls.c
[09/20/2016 22:38] seed@ubuntu:~/Desktop/lab1$ ./ls
I will delet lots files
[09/20/2016 22:38] seed@ubuntu:~/Desktop/lab1$ ./lsSet
I will delet lots files
[09/20/2016 22:38] seed@ubuntu:~/Desktop/lab1$
```

Fig 6.6 compile ls.c file and run ls and lsSet.

```

1 #include <stdio.h>
2
3 int main() {
4     printf("I will delet lots files\n");
5     printf("uid = %d\n", getuid());
6     printf("euid = %d\n", geteuid());
7     return 0;
8 }

```

Fig 6.7 change ls.c file and print uid and euid

```

[09/20/2016 23:31] seed@ubuntu:~/Desktop/lab1$ ./lsSet
I will delet lots files
uid = 1000
euid = 0
[09/20/2016 23:32] seed@ubuntu:~/Desktop/lab1$ 

```

Fig 6.8 call ./lsSet to make sure ls has root privilege.

Observation:

6.1 Fig 6.1 and Fig 6.2 are two program used in this task. As Fig6.4, compile lsSet.c file and change owner to root and then make it Set-UID program.

6.2 As Fig6.5, change PATH variable: PATH=/home/seed/Desktop/lab1:\$PATH

6.3 From Fig6.6, compile ls.c file, and then run this file. Later, run lsSet. We can find it print "I will delet lots files" rather than list files, compare to Fig 6.3.

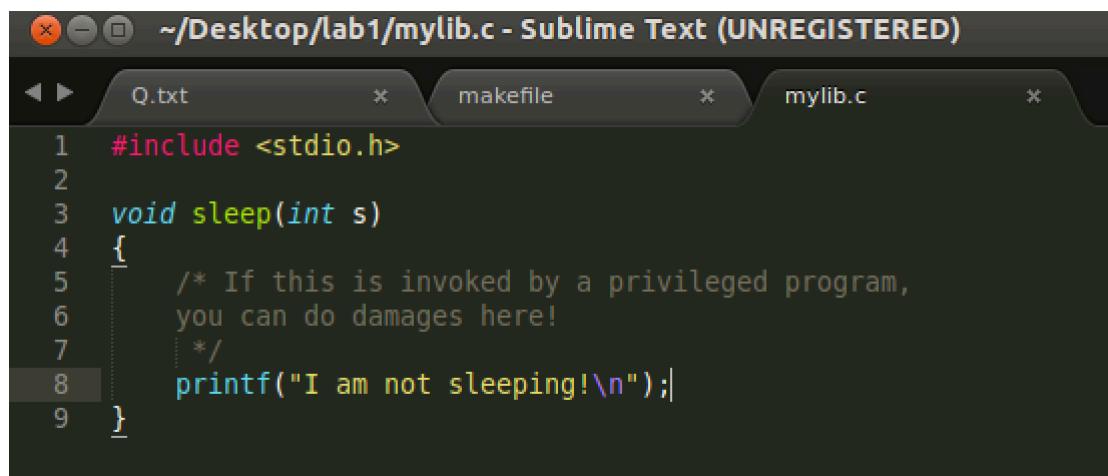
6.4 From Fig 6.8, we can get information uid = 1000, euid = 0. So that ./ls has root privilege.

Explanation:

6.5 system() will get environment variables from shell variables. Because I changed PATH variable, I redirected /bin/ls to /home/seed/Desktop/lab1/ls. So system("ls") will call shell to invoke /home/seed/Desktop/lab1/ls.

6.6 When I run ./IsSet, I get root privilege, because it is a Set-UID program. So root privilege process' child process still has root privilege.

Task7: The LD_PRELOAD environment variable and Set-UID Programs



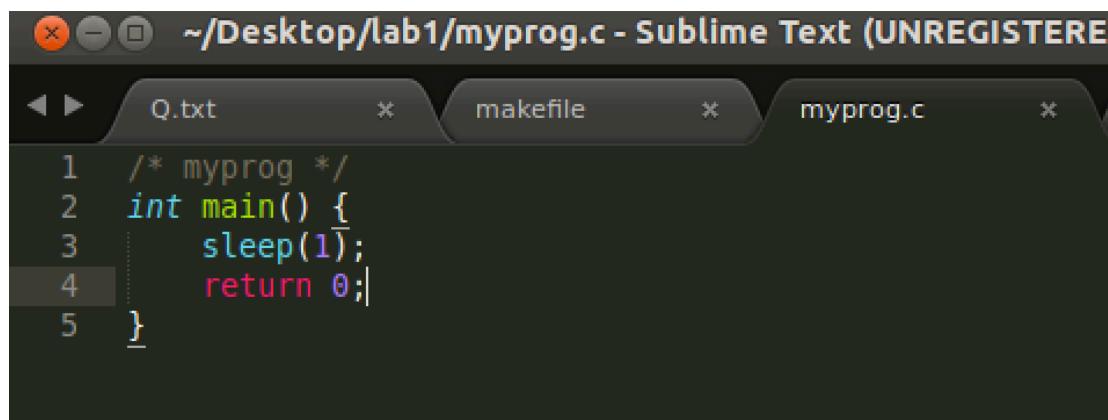
```

~/Desktop/lab1/mylib.c - Sublime Text (UNREGISTERED)

Q.txt      * makefile      * mylib.c      *
1 #include <stdio.h>
2
3 void sleep(int s)
4 {
5     /* If this is invoked by a privileged program,
6     you can do damages here!
7     */
8     printf("I am not sleeping!\n");
9 }

```

Fig 7.1 mylib.c



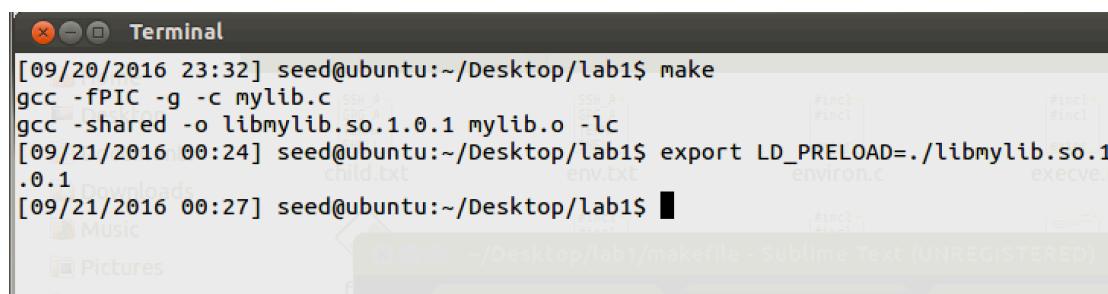
```

~/Desktop/lab1/myprog.c - Sublime Text (UNREGISTERED)

Q.txt      * makefile      * myprog.c      *
1 /* myprog */
2 int main() {
3     sleep(1);
4     return 0;
5 }

```

Fig 7.2 myprog.c



```

Terminal
[09/20/2016 23:32] seed@ubuntu:~/Desktop/lab1$ make
gcc -fPIC -g -c mylib.c
gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
[09/21/2016 00:24] seed@ubuntu:~/Desktop/lab1$ export LD_PRELOAD=./libmylib.so.1.0.1
[09/21/2016 00:27] seed@ubuntu:~/Desktop/lab1$ 

```

Fig 7.3 create libmylib.c and export LD_PRELOAD

```
[09/21/2016 00:41] seed@ubuntu:~/Desktop/lab1$ make
gcc -o myprog myprog.c
[09/21/2016 00:41] seed@ubuntu:~/Desktop/lab1$ ls -l myprog
-rwxrwxr-x 1 seed seed 7161 Sep 21 00:41 myprog
[09/21/2016 00:41] seed@ubuntu:~/Desktop/lab1$ ./myprog
[09/21/2016 00:42] seed@ubuntu:~/Desktop/lab1$ sudo chown root myprog
[sudo] password for seed:
[09/21/2016 00:45] seed@ubuntu:~/Desktop/lab1$ sudo chmod 4777 myprog
[09/21/2016 00:46] seed@ubuntu:~/Desktop/lab1$ ./myprog
[09/21/2016 00:46] seed@ubuntu:~/Desktop/lab1$ ls -l myprog
-rwsrwxrwx 1 root seed 7161 Sep 21 00:41 myprog
[09/21/2016 00:47] seed@ubuntu:~/Desktop/lab1$ sudo su root
[09/21/2016 00:47] root@ubuntu:/home/seed/Desktop/lab1# export LD_PRELOAD=./libmylib.so.1.0.1
[09/21/2016 00:49] root@ubuntu:/home/seed/Desktop/lab1# ./myprog
I am not sleeping!
[09/21/2016 00:50] root@ubuntu:/home/seed/Desktop/lab1#
```

Fig 7.4 three process for step 2

```
[09/21/2016 01:04] seed1@ubuntu:/home/seed/Desktop/lab1$ make
[09/21/2016 01:04] seed1@ubuntu:/home/seed/Desktop/lab1$ sudo chown seed1 myprog
[09/21/2016 01:04] seed1@ubuntu:/home/seed/Desktop/lab1$ sudo chmod 4777 myprog
[09/21/2016 01:05] seed1@ubuntu:/home/seed/Desktop/lab1$ ls -l myprog
-rwsrwxrwx 1 seed1 seed 7161 Sep 21 00:41 myprog
[09/21/2016 01:05] seed1@ubuntu:/home/seed/Desktop/lab1$ ./myprog
[09/21/2016 01:06] seed1@ubuntu:/home/seed/Desktop/lab1$
```

Fig 7.4 last process for step 2

Observation:

7.1 seed user run regular program, sleep() be called normally.

7.2 seed user run Set-UID root program, sleep() be called normally.

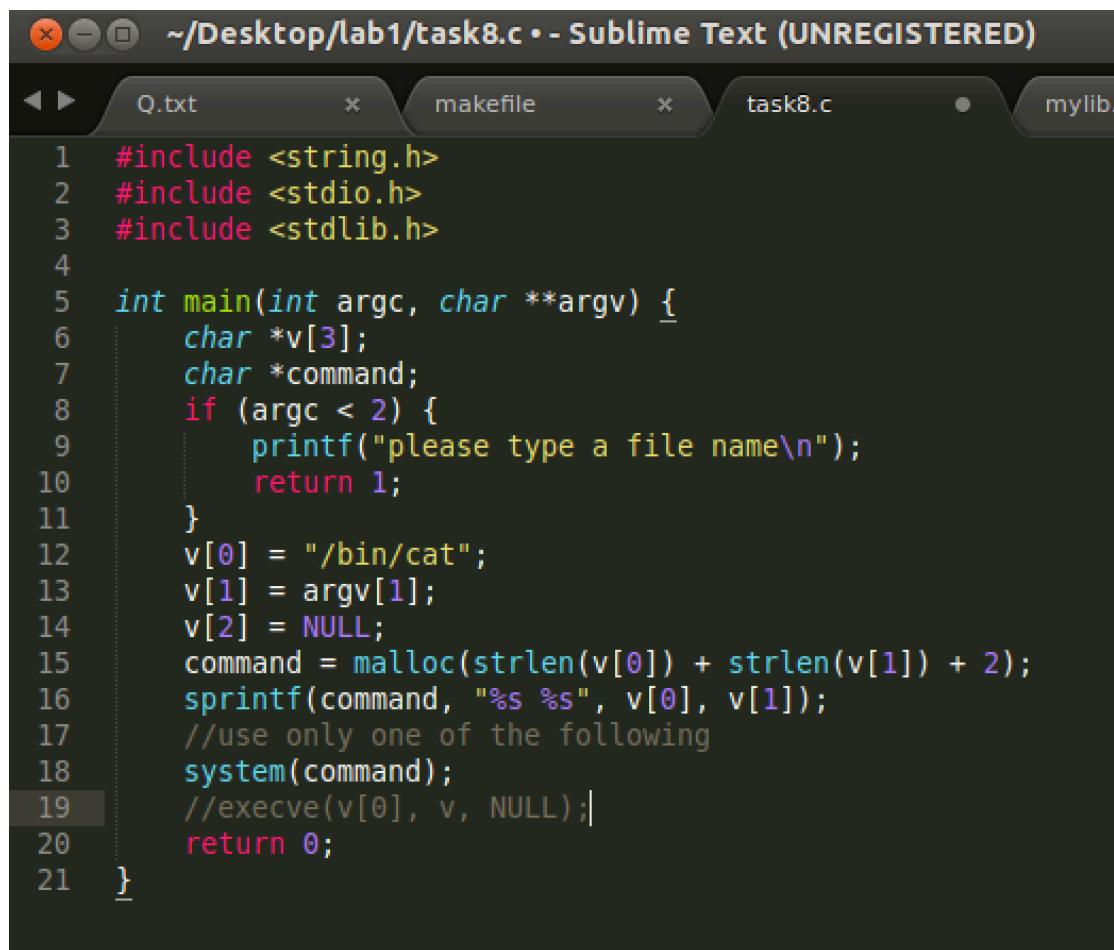
7.3 Root account run Set-UID root program, sleep() defined in libmylib.so.1.0.1 be called.

7.4 seed user run Set-UID root program, sleep() be called normally.

Explanation:

7.5 from the result of this experiment we can get conclusion that when normal user run Set-UID program LD_* variables will be ignored.

7.6 If root account, export variable and run Set-UID LD_* will not be ignored. So only root account can run export LD_* effected program.

Task8: Invoking external programs using system() versus execve()

The screenshot shows a Sublime Text window with multiple tabs. The active tab is 'task8.c'. The code in 'task8.c' is as follows:

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

Fig 8.1 call system()

```

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

```

Fig 8.2 call execve()

```

seed1@ubuntu:~/Desktop/lab1$ echo "execv" > test_exe.txt
seed1@ubuntu:~/Desktop/lab1$ echo "system" > test_sys.txt
seed1@ubuntu:~/Desktop/lab1$ sudo chown root:root test_exe.txt
seed1@ubuntu:~/Desktop/lab1$ sudo chown root:root test_sys.txt
seed1@ubuntu:~/Desktop/lab1$ sudo chmod 755 test_exe.txt
seed1@ubuntu:~/Desktop/lab1$ sudo chmod 755 test_sys.txt
seed1@ubuntu:~/Desktop/lab1$ ls -l test_exe.txt test_sys.txt
-rwxr-xr-x 1 root root 6 Sep 21 04:36 test_exe.txt
-rwxr-xr-x 1 root root 7 Sep 21 04:36 test_sys.txt
seed1@ubuntu:~/Desktop/lab1$ 

```

Fig 8.3 create two not writeable root owner file

```

seed1@ubuntu: /home/seed/Desktop/lab1
[09/21/2016 04:41] seed@ubuntu:~/Desktop/lab1$ make
gcc -o task8 task8.c
[09/21/2016 04:41] seed@ubuntu:~/Desktop/lab1$ sudo chown root task8
[09/21/2016 04:41] seed@ubuntu:~/Desktop/lab1$ sudo chmod 4755 task8
[09/21/2016 04:42] seed@ubuntu:~/Desktop/lab1$ ./task8 "test_sys.txt"
system
[09/21/2016 04:43] seed@ubuntu:~/Desktop/lab1$ ./task8 "test_sys.txt;rm test_sys
.txt"
system
[09/21/2016 04:43] seed@ubuntu:~/Desktop/lab1$ ls -l test_sys.txt
ls: cannot access test_sys.txt: No such file or directory
[09/21/2016 04:43] seed@ubuntu:~/Desktop/lab1$ make clean
rm task8
[09/21/2016 04:45] seed@ubuntu:~/Desktop/lab1$ ./task8 "test_exe.txt"
execv
[09/21/2016 04:46] seed@ubuntu:~/Desktop/lab1$ ./task8 "test_exe.txt;rm test_exe
.txt"
/bin/cat: test_exe.txt;rm test_exe.txt: No such file or directory
[09/21/2016 04:47] seed@ubuntu:~/Desktop/lab1$ █

```

Fig 8.4 procedure for lab

Observation:

8.1 From Fig 8.4, when system() is called, input “./task8 test.txt” will get contents of test.txt.

“./task8 test.txt;rm test.txt” will remove not writeable test_sys.txt

8.2 From Fig 8.4, when execve() is called, input “./task8 test.txt” will get contents of test.txt.

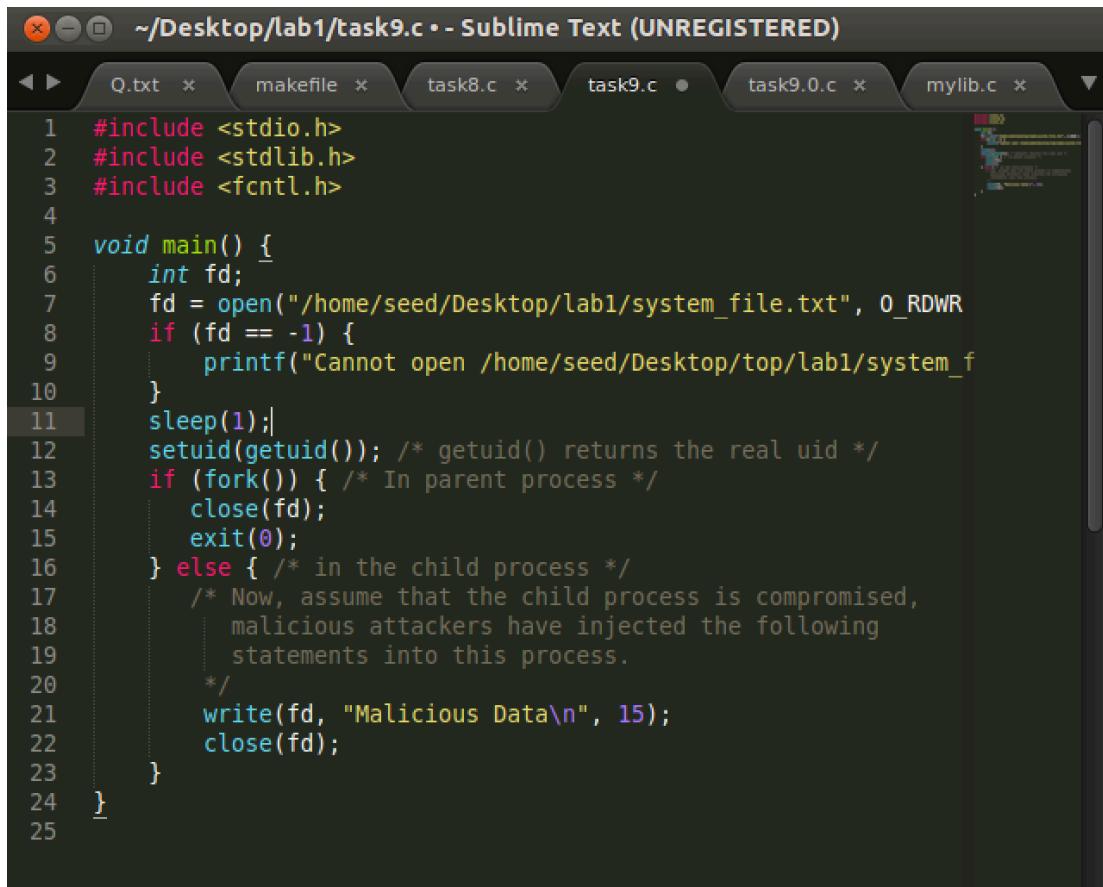
“./task8 test.txt;rm test.txt” will print error message.

Explanation:

8.3 Set-UID program can help user get file owner's privilege. As file owner is root, so this program could get root privilege. System() will invoke shell to run command. If “file;command2” is input for this program, command2 will be executed. So the file can be removed.

8.4 Execve is not invoked from shell. If we use “file;command2” as input, this string will be pass to execv(file, args[], enrp) as second argument. This function will not understand input arguments.

Task9: Capability Leaking

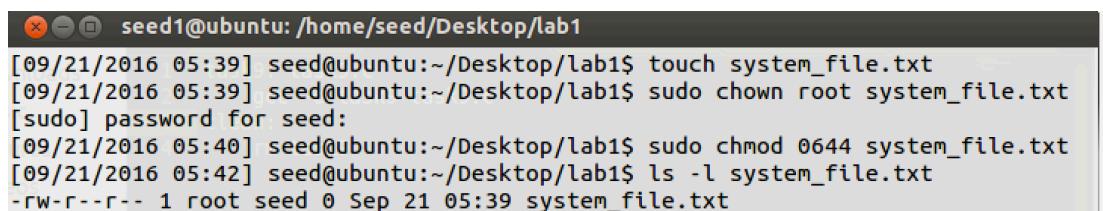


```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <fcntl.h>
4
5 void main() {
6     int fd;
7     fd = open("/home/seed/Desktop/lab1/system_file.txt", O_RDWR);
8     if (fd == -1) {
9         printf("Cannot open /home/seed/Desktop/top/lab1/system_");
10    }
11    sleep(1);
12    setuid(getuid()); /* getuid() returns the real uid */
13    if (fork()) { /* In parent process */
14        close(fd);
15        exit(0);
16    } else { /* in the child process */
17        /* Now, assume that the child process is compromised,
18           malicious attackers have injected the following
19           statements into this process.
20        */
21        write(fd, "Malicious Data\n", 15);
22        close(fd);
23    }
24 }
25

```

Fig 9.1 task9 Set-UID program code

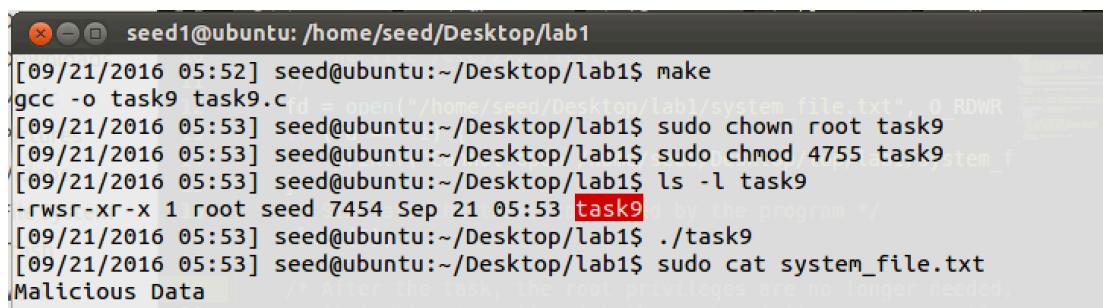


```

seed1@ubuntu: /home/seed/Desktop/lab1
[09/21/2016 05:39] seed@ubuntu:~/Desktop/lab1$ touch system_file.txt
[09/21/2016 05:39] seed@ubuntu:~/Desktop/lab1$ sudo chown root system_file.txt
[sudo] password for seed:
[09/21/2016 05:40] seed@ubuntu:~/Desktop/lab1$ sudo chmod 0644 system_file.txt
[09/21/2016 05:42] seed@ubuntu:~/Desktop/lab1$ ls -l system_file.txt
-rw-r--r-- 1 root seed 0 Sep 21 05:39 system_file.txt

```

Fig 9.2 make a test file owned by root and permission with 0644

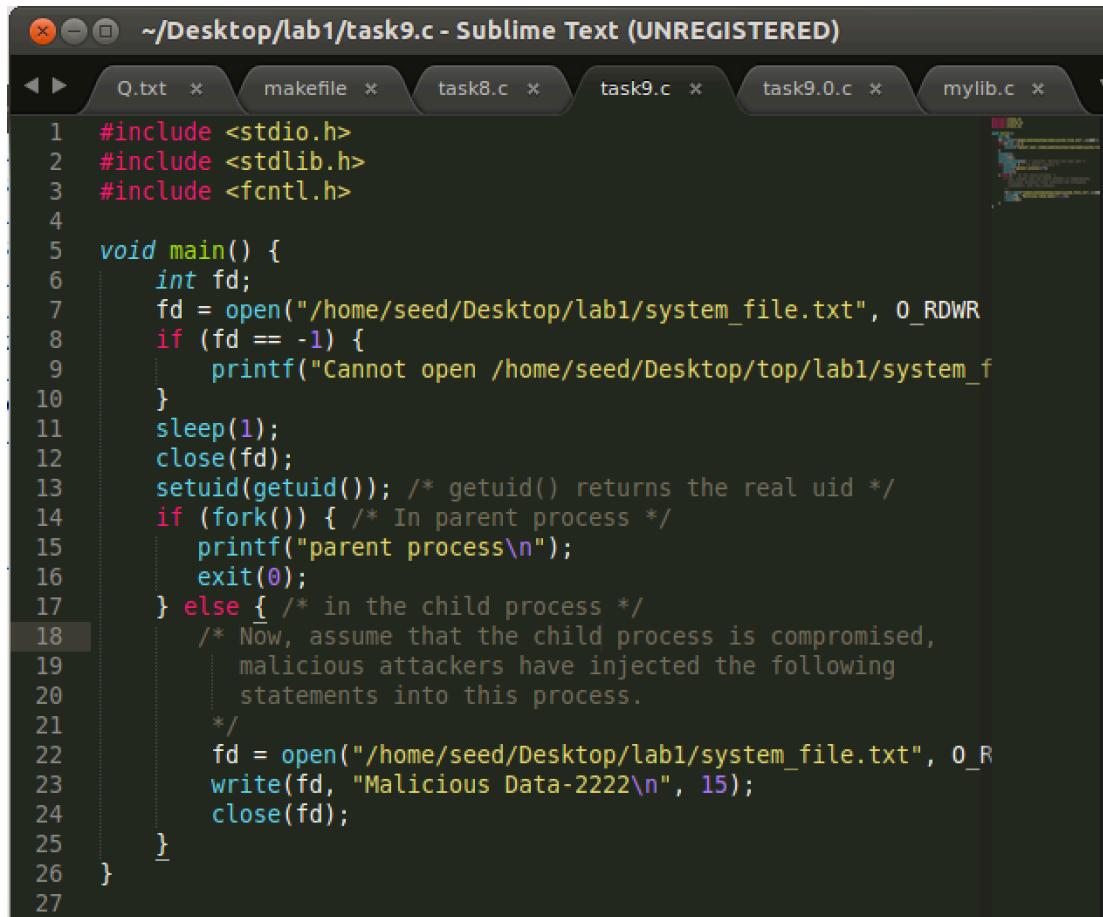


```

seed1@ubuntu: /home/seed/Desktop/lab1
[09/21/2016 05:52] seed@ubuntu:~/Desktop/lab1$ make
gcc -o task9 task9.c
[09/21/2016 05:53] seed@ubuntu:~/Desktop/lab1$ sudo chown root task9
[09/21/2016 05:53] seed@ubuntu:~/Desktop/lab1$ sudo chmod 4755 task9
[09/21/2016 05:53] seed@ubuntu:~/Desktop/lab1$ ls -l task9
-rwsr-xr-x 1 root seed 7454 Sep 21 05:53 task9
[09/21/2016 05:53] seed@ubuntu:~/Desktop/lab1$ ./task9
[09/21/2016 05:53] seed@ubuntu:~/Desktop/lab1$ sudo cat system_file.txt
Malicious Data

```

Fig 9.3 make task9.c owned by root and Set-UID, compiele and run task 9 code



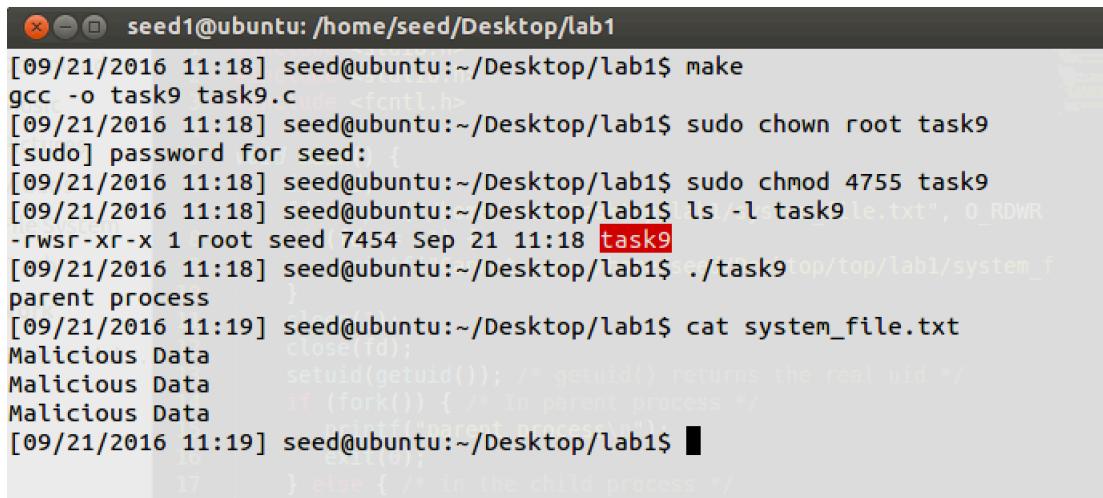
```

~/Desktop/lab1/task9.c - Sublime Text (UNREGISTERED)

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <fcntl.h>
4
5 void main() {
6     int fd;
7     fd = open("/home/seed/Desktop/lab1/system_file.txt", O_RDWR
8     if (fd == -1) {
9         printf("Cannot open /home/seed/Desktop/top/lab1/system_
10    }
11    sleep(1);
12    close(fd);
13    setuid(getuid()); /* getuid() returns the real uid */
14    if (fork()) { /* In parent process */
15        printf("parent process\n");
16        exit(0);
17    } else { /* in the child process */
18        /* Now, assume that the child process is compromised,
19           malicious attackers have injected the following
20           statements into this process.
21        */
22        fd = open("/home/seed/Desktop/lab1/system_file.txt", O_R
23        write(fd, "Malicious Data-2222\n", 15);
24        close(fd);
25    }
26 }
27

```

Fig 9.4 change code to test child process privilege



```

seed1@ubuntu:~/Desktop/lab1
[09/21/2016 11:18] seed@ubuntu:~/Desktop/lab1$ make
gcc -o task9 task9.c
[09/21/2016 11:18] seed@ubuntu:~/Desktop/lab1$ sudo chown root task9
[sudo] password for seed: 
[09/21/2016 11:18] seed@ubuntu:~/Desktop/lab1$ sudo chmod 4755 task9
[09/21/2016 11:18] seed@ubuntu:~/Desktop/lab1$ ls -l task9
-rwsr-xr-x 1 root seed 7454 Sep 21 11:18 task9
[09/21/2016 11:18] seed@ubuntu:~/Desktop/lab1$ ./task9
parent process
[09/21/2016 11:19] seed@ubuntu:~/Desktop/lab1$ cat system_file.txt
Malicious Data
Malicious Data
Malicious Data
[09/21/2016 11:19] seed@ubuntu:~/Desktop/lab1$ 

```

Fig 9.5 new text is not be written in test file.

Observation:

9.1 From Fig 9.3, after I run ./task9, I can cat contents of “Malicious Data”.

9.2 As Fig 9.4, 9.5, new text “Malicious Data-2222” cannot be written in test file.

Explanation:

9.3 The test file system_file.txt is root owned and permission 0644, so that only user with root privilege could write something in it. So child process forked by ./task9 had root privilege.

9.4 Because task9 is owned by root and Set-UID program, when it is run, user could get root privilege.

9.5 getuid() returns real uid number, so setuid(getuid()) can set euid back and equal to real uid. After euid back to real uid, its privilege downgraded. But, the process is still privileged because it possesses privilege capabilities.

9.6 In this program, file descriptor was opened with root privilege and then child process was forked. Child process had a fd with root privilege opened and child process can run it. So child process can write its thing in this file.

9.7 As Fig 9.4, I closed fd, before fork. Then child process will not have root privilege to open this file when they open it in child process with normal privilege.