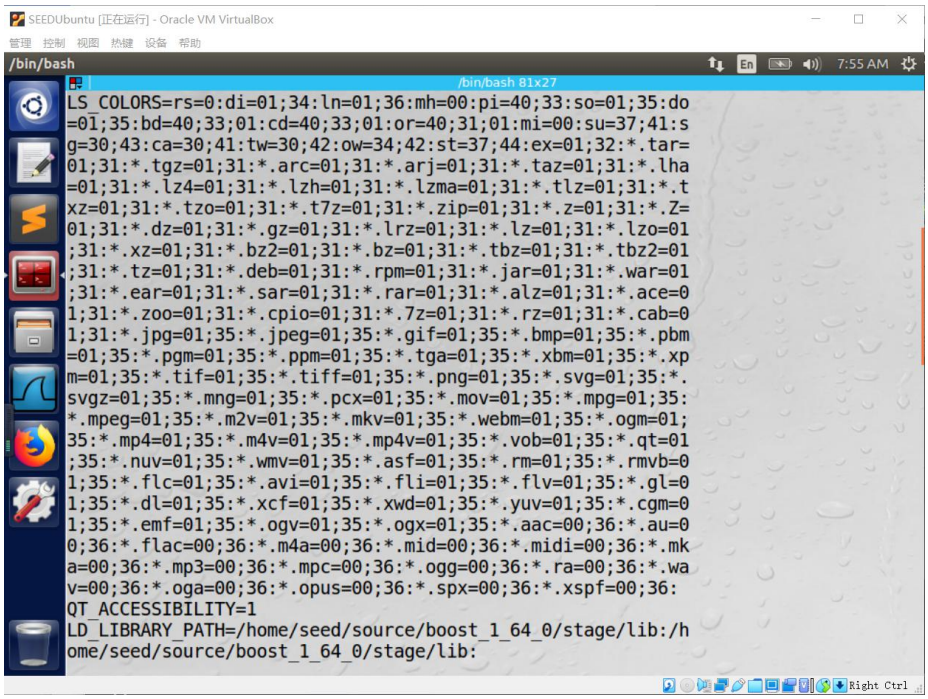
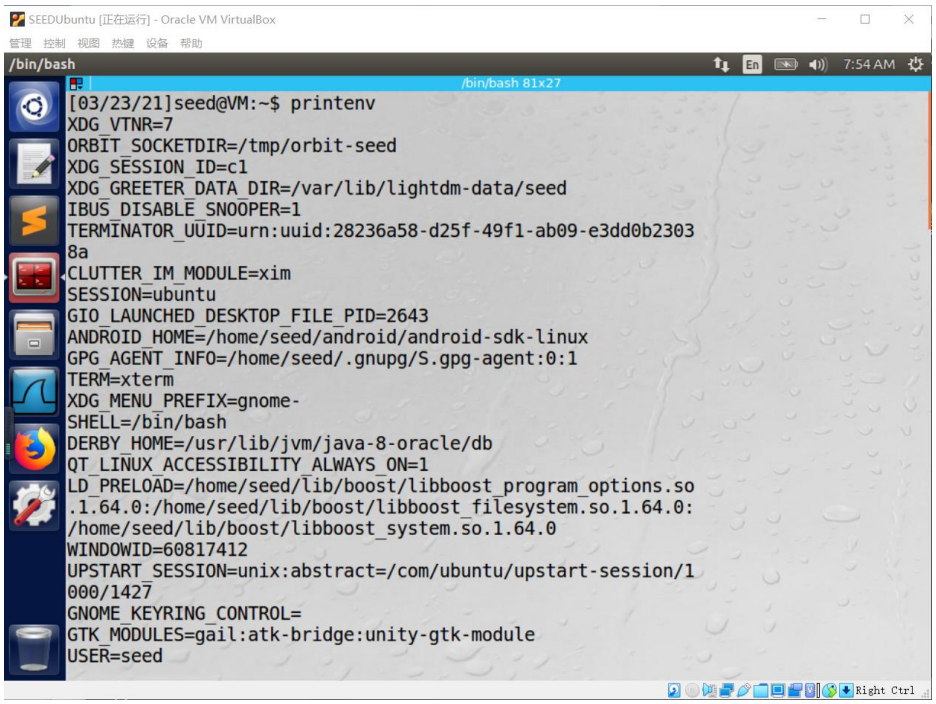
LAB1 实验报告

57119136 李政君

Task 1：Manipulating Environment Variables

打印环境变量





设置和取消设置环境变量

```
[03/23/21]seed@VM:~$ export LZJ="57119136"
[03/23/21]seed@VM:~$ echo LZJ
LZJ
[03/23/21]seed@VM:~$ printenv LZJ
57119136
[03/23/21]seed@VM:~$ unset LZJ
[03/23/21]seed@VM:~$ printenv LZJ
[03/23/21]seed@VM:~$
```

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

比较两次编译结果的输出文档

```
[03/23/21]seed@VM:~/.../task2$ diff a.txt b.txt
```

结果：子进程与父进程运行的结果一样，两次输出的环境变量除文档名称外完全相同

结论：使用 fork（）调用后父进程的环境变量确实能被子进程继承。

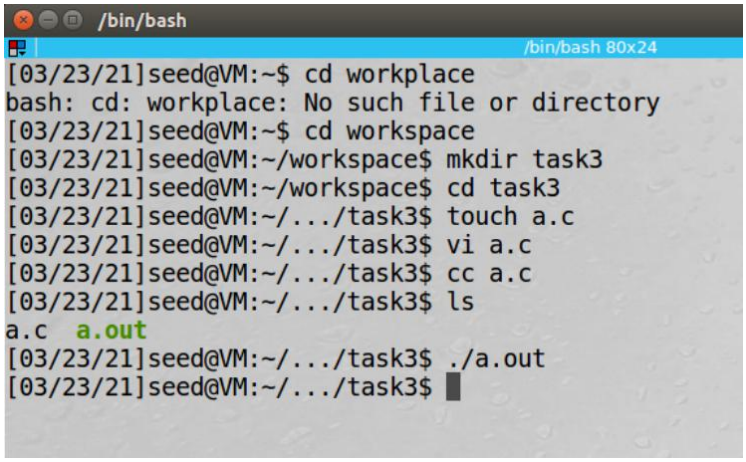
Task 3： Environment Variables and execve()

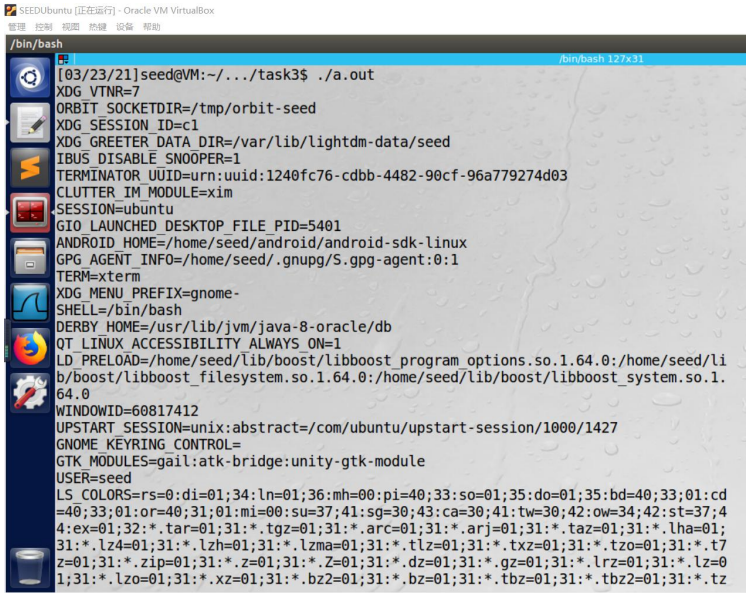
根据实验要求分别进行两次编译执行

```
/bin/bash
#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 ;
}
"a.c" 13L, 202C
```

```
/bin/bash
#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, environ);
return 0 ;
}
"a.c" 13L, 205C
```

下图分别为第一次执行结果和第二次执行结果，第一次编译输出为空，第二次编译输出当前环境变量

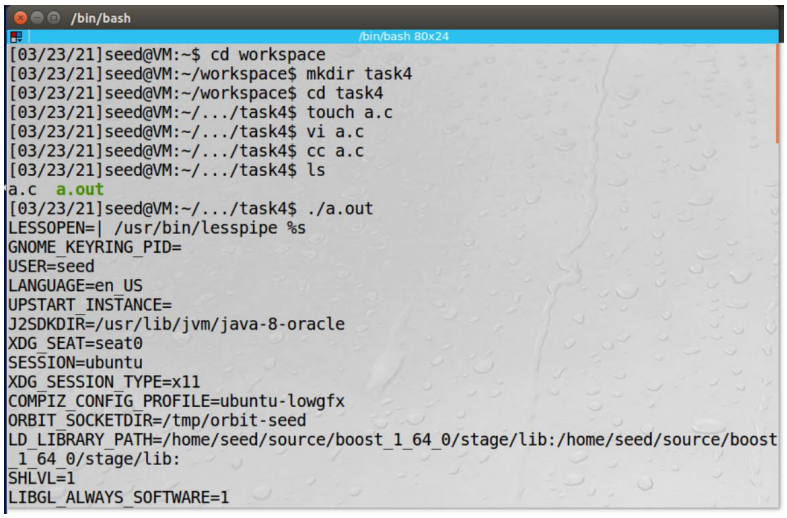




结论：父进程通过 environ 传递参数时新进程获得了原来的环境变量输出

Task 4：Environment Variables and system()

根据实验要求编译程序，输出结果为当前环境变量



结论：system()会调用 fork()产生子进程，由子进程来调用/bin/sh 执行参数 string 字符串所代表的命令，此命令执行完后随即返回原调用的进程。

Task 5：Environment Variable and Set-UID Programs

Step 1：程序编译执行后输出为当前环境变量

```
[03/23/21]seed@VM:~$ cd workspace
[03/23/21]seed@VM:~/workspace$ mkdir task5
[03/23/21]seed@VM:~/workspace$ cd task5
[03/23/21]seed@VM:~/.../task5$ touch a.c
[03/23/21]seed@VM:~/.../task5$ vi a,c
[03/23/21]seed@VM:~/.../task5$ vi a.c
[03/23/21]seed@VM:~/.../task5$ cc a.c
[03/23/21]seed@VM:~/.../task5$ ls
a,c  a.c  a.out
[03/23/21]seed@VM:~/.../task5$ ./a.out
XDG_VTNR=7
ORBIT_SOCKETDIR=/tmp/orbit-seed
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
IBUS_DISABLE_SNOOPER=1
TERMINATOR_UUID=urn:uuid:7015269c-8d44-420d-9996-3633689c907b
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
XDG_MENU_PREFIX=gnome-
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
```

Step 2：编译上述程序，将其所有权更改为 root，并使其成为 Set-UID 程序

```
[03/23/21]seed@VM:~/.../task5$ sudo chown root a.c
[03/23/21]seed@VM:~/.../task5$ sudo chmod u+s a.c
```

Step 3：在 shell 中使用导出命令设置以下三个环境变量

```
[03/23/21]seed@VM:~/.../task5$ export PATH="$PATH:/usr/local/"
[03/23/21]seed@VM:~/.../task5$ export LD_LIBRARY_PATH="$LD_LIBRARY-PATH:/usr/loc
al/"
[03/23/21]seed@VM:~/.../task5$ export LZJ="/usr/local"
[03/23/21]seed@VM:~/.../task5$ cc a.c
[03/23/21]seed@VM:~/.../task5$ ./a.out
```

运行后只能输出 PATH 和 LZJ 两个环境变量，不可输出 LD_LIBRARY 环境变量

```
LZJ=/usr/local
```

```
PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
:/usr/games:/usr/local/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:/hom
e/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin:/usr/local/
```

将此程序设置为 seed 试验后再次执行，可以将三个环境变量全部输出

```
LD_LIBRARY_PATH=-PATH:/usr/local/
```

实验总结：Set-UID 程序申请的 shell 是 root 权限的，Linux 因为这个权限太高太危险了，就不会显示 LD-PRELOAD；而恢复普通程序后运行，就可以输出 LD-PRELOAD。

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

根据实验要求将程序进行编译，修改环境变量，并设置程序为 Set-UID 程序取消保护机制后编译运行

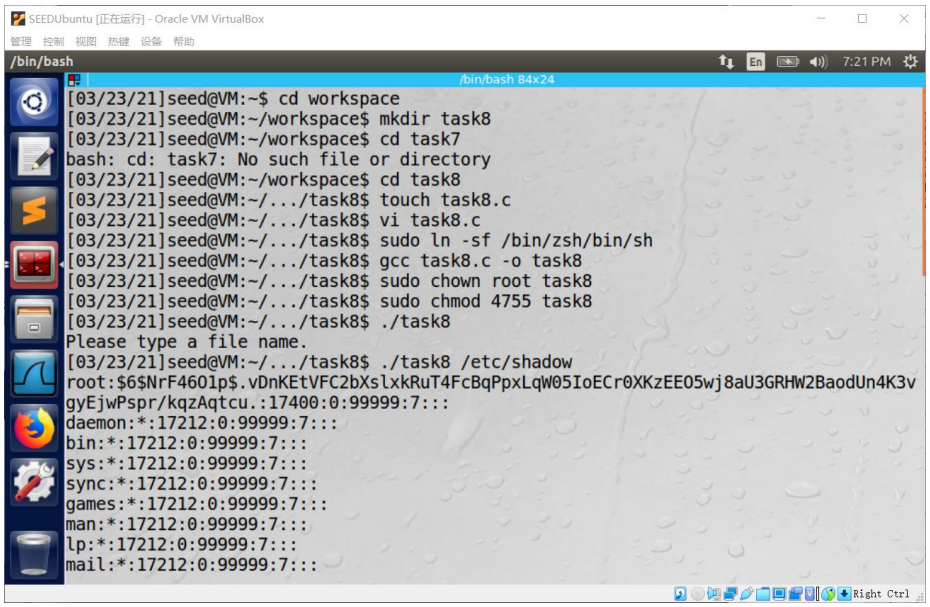
```
[03/22/21]seed@VM:~$ sudo rm /bin/sh
[03/22/21]seed@VM:~$ sudo ln -s /bin/zsh /bin/sh
[03/22/21]seed@VM:~$
```

```
[03/22/21]seed@VM:~$ task6
android             mylib.c         task4_out.txt
bin                 mylib.o         task5
CopyTo.txt          myprog.c        task5.c
Customization       Pictures        task5_out.txt
Desktop             Public          task6
docker-admin        source          task6.c
Documents           task2.c         task8
Downloads           task2_out2.txt  task8.c
examples.desktop    task2_out.txt   task9
get-pip.py          task3.c         task9.c
hello.c             task3_out2.txt  Templates
lib                 task3_out.txt   Videos
libmylib.so.1.0.1   task4
Music               task4.c
```

实验总结：由本次实验可知，可以通过 root 权限直接调用打开 shell。

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

根据实验要求编译输出影子文件




```
[03/23/21]seed@VM:~/.../task8$ ./task8 "aa;/bin/sh"
/bin/cat: aa: No such file or directory
# whoami
root
#
```

注释 system()，取消注释 execve()后，提权失败

```
# vi task8.c
# gcc task8.c -o task8
task8.c: In function ‘main’:
task8.c:17:1: warning: implicit declaration of function ‘execve’ [-Wimplicit-functio
n-declaration]
 execve(v[0], v, NULL);
 ^
# ./task8 "aa;/bin/sh"
/bin/cat: 'aa;/bin/sh': No such file or directory
#
```

实验总结：在 Linux 系统编译中，system()没有 execve()安全。