环境变量与 Set-UID 程序实验报告

报告人：57119132 汪奥杰　　日期：2021 年 7 月 6 日

1. 实验目标

通过一系列的实验来了解环境变量如何影响系统或程序的行为，弄清楚环境变量是如何运作的，例如如何从父进程传递到子进程，以及如何影响系统或程序的行为等等。

2. 实验任务

2.1 操作环境变量

1. 打印环境变量 PWD

```
[07/06/21]seed@VM:~$ printenv PWD
/home/seed
```

2. 设置环境变量 ANY_NAME 的值为 0 并打印验证

```
[07/06/21]seed@VM:~$ export ANY_NAME=0
[07/06/21]seed@VM:~$ printenv ANY_NAME
0
```

3. 取消环境变量 ANY_NAME

```
[07/06/21]seed@VM:~$ unset ANY_NAME
```

2.2 将环境变量从父进程传递给子进程

1. 编译并运行如下程序 a.c，其输出为 a.out

```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);
    }
}
```

2. 将子进程的 printenv()注释，父进程的 printenv()取消注释，编译并运行程序 a1.c，其输出为 a1.out

3. 将两个程序的输出结果分别放入另外两个文件 a.txt 和 a1.txt，比较其区别

```
[07/06/21]seed@VM:~$ diff a.txt a1.txt
48c48
< _=./a.out
---
> _=./a1.out
```

4. 结论

两次输出的环境变量完全相同，使用 fork()函数会使生成的子进程继承父进程的全部环境变量。

2.3 环境变量和 execve()

1. 编译并运行以下程序 b1.c，其输出为 b1.out

```c
#include <stdio.h>
#include <stdlib.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 ;
}
```

2. 将程序中第 9 行的 NULL 改为 environ，编译并运行程序 b2.c，其输出为 b2.out

3. 结论

第一次输出为空，第二次输出为当前进程的环境变量，这说明父进程通过 environ 传递了参数。

综上可知，进程在被初始化时获取环境变量有两种方式：fork()，execve()。

2.4 环境变量和 system()

1. 编译并运行以下程序 c.c，其输出为 c.out

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

2. 结论

编译运行，程序输出为当前进程的环境变量。

2.5 环境变量和 Set-UID 程序

1. 编译并运行以下程序 d.c，其输出为 d.out，并打印当前进程的环境变量

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

2. 将拥有者改为 root，并将该程序变为 Set-UID 程序

```
[07/06/21]seed@VM:~$ sudo chown root d.out
[07/06/21]seed@VM:~$ sudo chmod 4755 d.out
```

3. 在 shell 中设置环境变量，运行原程序，发现输出的环境变量中出现了设置的环境变量

```
[07/06/21]seed@VM:~$ printenv LD_LIBRARY_NAME
[07/06/21]seed@VM:~$ export ANY_NAME=anyname
[07/06/21]seed@VM:~$ printenv ANY_NAME
anyname
[07/06/21]seed@VM:~$ ./d.out
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1950,unix/VM:/tmp/.ICE-unix/1950
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
ANY_NAME=anyname
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=1913
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed
```

4. 结论

shell 程序执行时会为每个环境变量创建名称和值都相同的 shell 变量，而 Bash 中，用户自定义的 shell 变量会被传递给子进程。因此，在 Bash 中有两种类型的 shell 变量会被传递给子进程：从环境变量复制得到的 shell 变量、用户自定义的 shell 变量。

2.6 PATH 环境变量和 Set-UID 程序

1. 编译并运行以下程序 demo.c，更改其拥有者为 root，将其变为 Set-UID 程序

```c
int main()
{
    system("ls");
    return 0;
}
```

2. 取消 Ubuntu16.04 的保护机制

```
[07/06/21]seed@VM:~$ sudo rm /bin/sh
[07/06/21]seed@VM:~$ sudo ln -s /bin/zsh /bin/sh
```

3. 编译并运行以下恶意程序 ls.c

```c
/* malicious "ls" program */
#include <stdio.h>
int main()
{
    printf("Hello World!\n");
    return 0;
}
```

4. 修改环境变量 PATH 并编译，更改拥有者为 root，并将它变为 Set-UID 程序执行，输出结果为预先定义好的“Hello world!”

```
[07/06/21]seed@VM:~$ export PATH=/home/seed:$PATH
[07/06/21]seed@VM:~$ sudo chown root ls.out
[07/06/21]seed@VM:~$ sudo chmod 4755 ls.out
[07/06/21]seed@VM:~$ ./ls.out
Hello World!
```

5. 结论

shell 程序执行命令时，如果没有提供命令的具体位置，shell 程序将使用 PATH 环境变量搜索命令。system()将环境变量传递给子进程，通过修改 PATH 环境变量，程序会先搜索当前目录，调用预先准备好的 ls 程序。

2.7 使用 system ()和 execve ()调用外部程序

1. 编译并运行以下程序 catall.c，其输出为 catall.out

```
#include <string.h>
#include <stdio.h>
#include <stdlib.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 ;
}
```

2. 更改 catall.out 程序拥有者为 root，并将它变为 Set-UID 程序

```
[07/06/21]seed@VM:~$ sudo ln -sf /bin/zsh /bin/sh
[07/06/21]seed@VM:~$ sudo chown root catall.out
[07/06/21]seed@VM:~$ sudo chmod 4755 catall.out
```

3. 设置一测试程序 test，将其权限设为 000

```
[07/06/21]seed@VM:~$ sudo chmod 000 test
```

4. 运行程序 catall.out，删除 test 文件并输出 test 程序的内容

```
[07/06/21]seed@VM:~$ ./catall.out "test;rm test -rf"
test
```

5. 将 catall.c 中的 system()注释，execve()取消注释，进行相同操作，删除 test 文件但是程序输出失败

```
[07/06/21]seed@VM:~$ ./catall.out "test;rm test -rf"
/bin/cat: 'test;rm test -rf': No such file or directory
```

6. 结论

使用 execve()调用外部程序比使用 system()更安全。