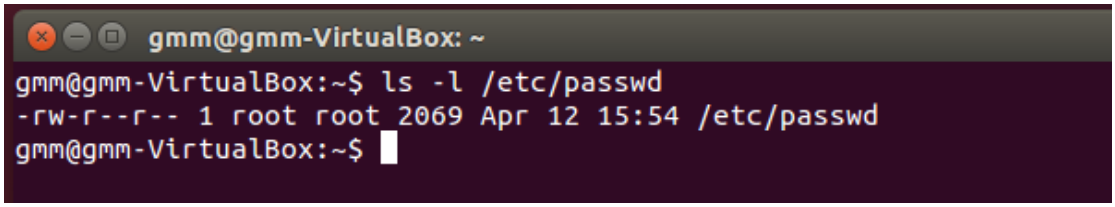
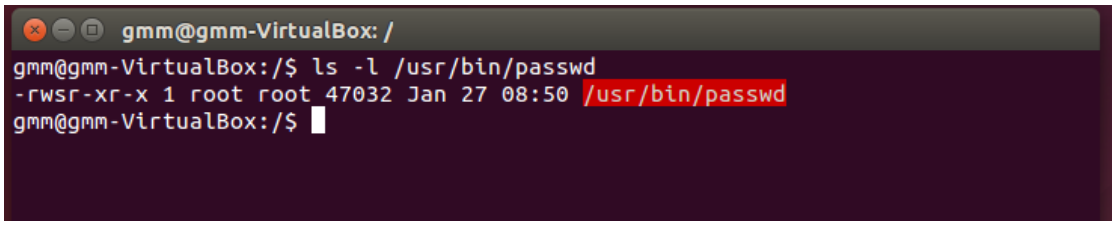
实验内容

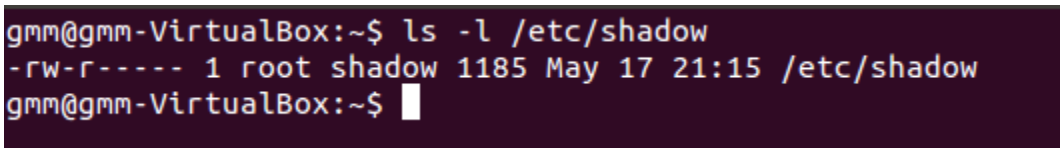
1、 查看/etc/passwd 文件权限



/etc/passwd 是一个文件，拥有者为 root,且只有 root 具有写权限。该文件主要是保存用户信息，例如:用户名、宿主目录、登陆环境、失效时间。这些信息都涉及到系统安全的敏感地带，因此设置为只有 root 才能修改。

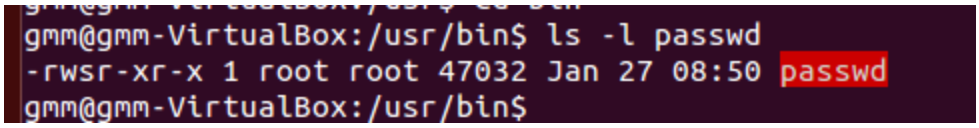
2、 查看/usr/bin/passwd 程序权限

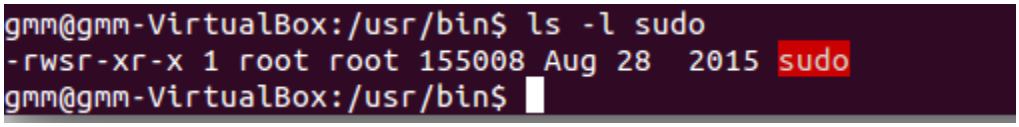




/user/bin/passwd 是一个命令，可以为用户添加、更改密码，但是，用户的密码并不保存在/etc/passwd 当中，而是保存在了/etc/shadow 当中。我们可以看到/etc/shadow 文件属于 root,且只有 root 能读写和 root 组能查看。所以普通用户是没有权限修改自己密码的。因此/usr/bin/passwd 程序需要设置 setuid 位，使得普通用户继承 root 权限，才有权限去修改自己的密码。

2.





以上为两个 setuid 位的程序。查找方法，在某一目录先输入 ls -l 命令。如在/usr/bin 下输入，找到/usr/bin/passwd,/usr/bin/sudo 两个

程序。

sudo 是为了方便用户临时获得 root 权限的一个程序，当有些程序需要用户具有 root 权限才能操作时，sudo 就显得尤为重要。

passwd 是为了方便用户修改 root 拥有的/etc/shadow 文件的程序。shadow 只有 root 有权限读写，如果不 setuid 则，用户无法修改自己的密码。

2、 1、用户拥有流星雨.txt 允许别人下载

其他人要对文件所在的每一级目录都有执行权限，对文件要有读权限。拥有目录的读权限可以找到文件，拥有文件的读权限可以复制文件内容下载。

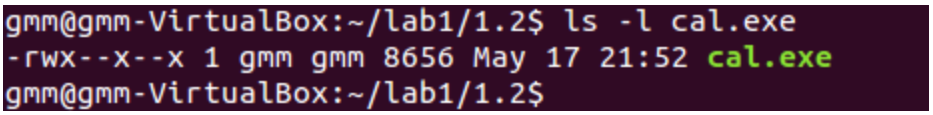
2. cal.exe 系统启动时运行

cal.c 程序

```
#include "apue.h"
#include "time.h"
void main()
{
 FILE *fp=fopen("/home/gmm/lab1/开机.txt","w");
 fprintf(fp,"我是开机启动时写入的\n");
 fclose(fp);
}
```

程序内容为在/home/gmm/lab1 下创建一个开机.txt 文件，然后里面写入一些内容。

权限设置



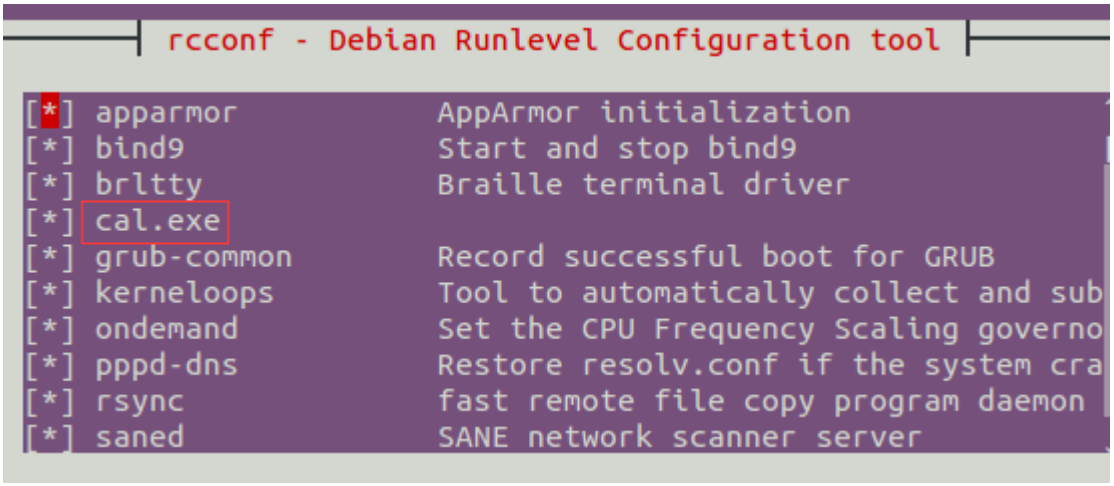
其他用户拥有执行权

添加到开机自启动

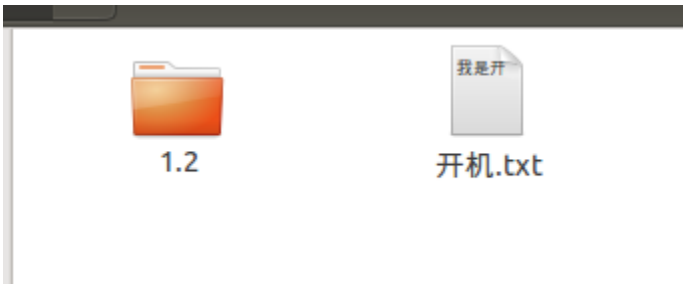
把 cal.exe 复制到/etc/init.d 目录下

执行 sudo update-rc.d cal.exe defaults

查看开机启动项



检查结果。可以看到开机.txt 文件



3. 让同组人帮忙修改 demo.txt 文件

```
gmm@gmm-VirtualBox:~/lab1/1.2$ la -l demo.txt
-rw-rw---- 1 gmm gmm 0 May 14 10:19 demo.txt
```

给同组人写权限

4. 一个 root 用户拥有的网络服务程序”netmonitor.exe”,需要设置 setuid 位才能完成其功能。

```
gmm@gmm-VirtualBox:~/lab1/1.2$ ls -l netmonitor.exe
-rws--x--x 1 root root 0 May 14 10:22 netmonitor.exe
gmm@gmm-VirtualBox:~/lab1/1.2$
```

通过 sudo chmod 4711 来 setuid

实验二、一些可执行程序运行时需要系统管理员权限，在 UNIX 中可以利用 setuid 位实现其功能，但 setuid 了的程序运行过程中拥有了 root 权限，因此在完成管理操作后需要切换到普通用户的身份执行后续操作。

(1)设想一种场景，比如提供 http 网络服务，需要设置 setuid 位，并为该场景编制相应的代码；

(2)如果用户 fork 进程后，父进程和子进程中 euid、ruid、suid 的差别；

(3)利用 execl 执行 setuid 程序后，euid、ruid、suid 是否有变化；

(4)程序何时需要临时性放弃 root 权限，何时需要永久性放弃 root 权限，并在程序中分别实现两种放弃权限方法；

(5)execl 函数族中有多个函数，比较有环境变量和无环境变量的函数使用的差异。

实验流程。

1.编写 http 服务程序，kill 服务程序，echoall 服务程序。

下面是三个服务程序的权限信息

```
---x------ 1 root root  8760 May 18 09:17 http
```

```
---x------ 1 root root  8998 May 18 09:16 kill
```

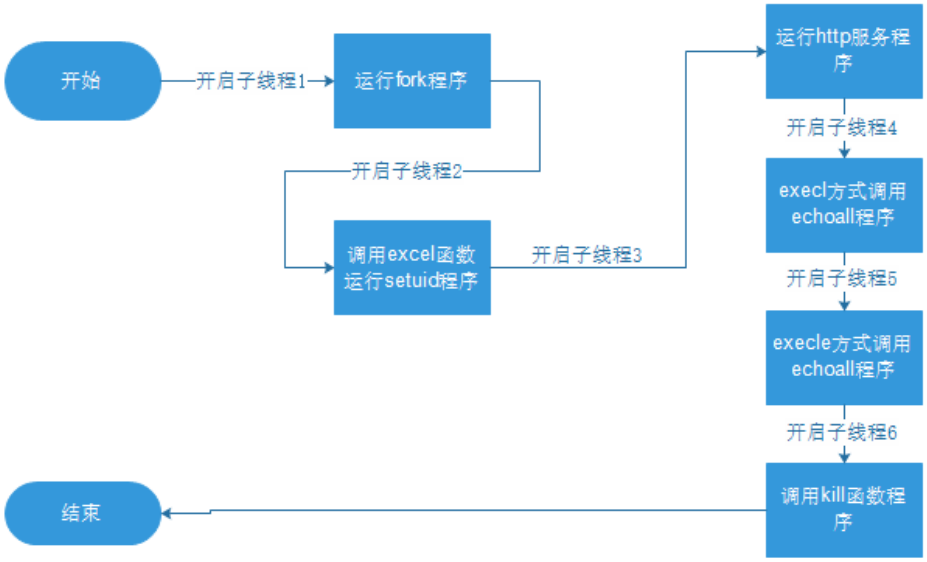
```
-rwxrwxr-x 1 gmm  gmm   8804 May 18 10:53 echoall
```

http 服务：开启每隔 1 秒打印：“您已开启 http 服务”

kill 服务：杀死指定进程

echoall 服务：打印命令参数，当前环境变量

2．流程图



1、 运行主程序./test4。此时用户的 ruid=euid=suid=1000

2、 开启 fork 线程，运行打印函数。此时子进程用户的 ruid=euid=suid=1000

3、 用户要开启 http 服务，而 http 的权限设置为只能 root 启动，所以用户通过设置了 uid 的程序来打开 http 服务。在这里主程序开启子线程通过 execl 的方式运行 setuid 程序

4、 进入 setuid 程序。进入后用户的 ruid=1000,euid=suid=0

5、 setuid 程序里面调用了四个进程函数，第一个调用的是 http 服务。第二个和第三个是调用 echoall 函数。用户在 setuid 程序里以 root 的身份去启动 http 服务，启动 http 服务后，用户要启动普通用户程序 echoall 程序。因此我们要进行权限的收回。这里我们采用临时收回权限。收回后 setuid 进程用户的 ruid=euid=1000,suid=0.

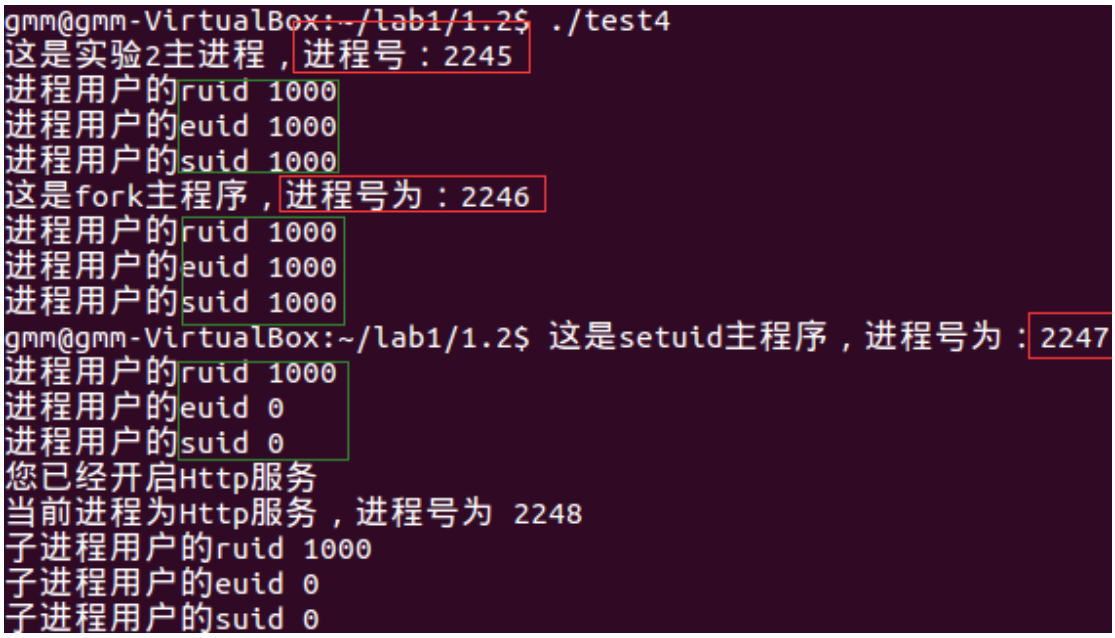
6、 用户调用 echoall 函数.一种是通过 execl 方式调用，一种是通过 execle 方式

调用。

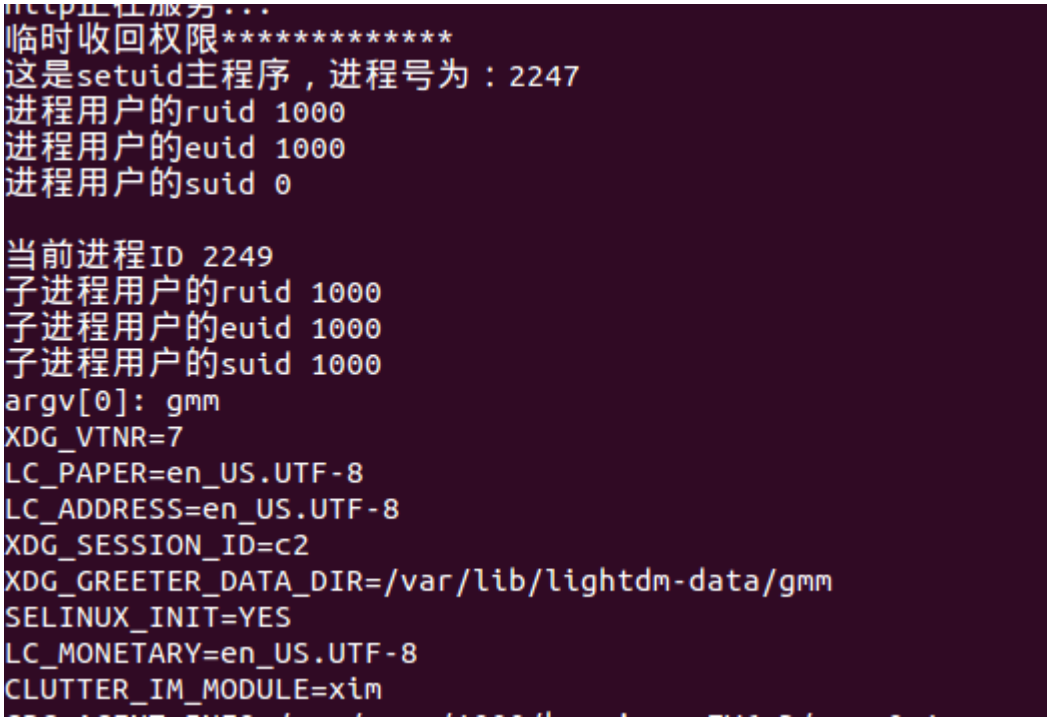
7、 用户此时想杀死 http 进程，需要用到 kill 程序，此时我们恢复用户权限。

8、 执行完 kill 程序后，我们让用户永久放弃权限

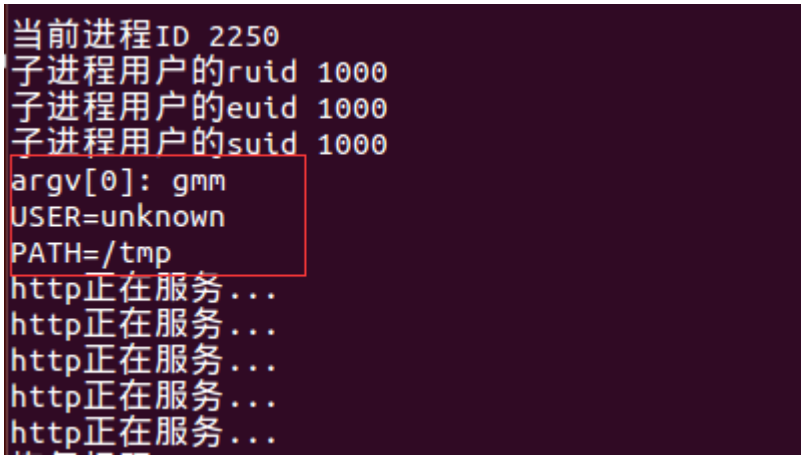
3.实验结果和关于实验内容问题回答



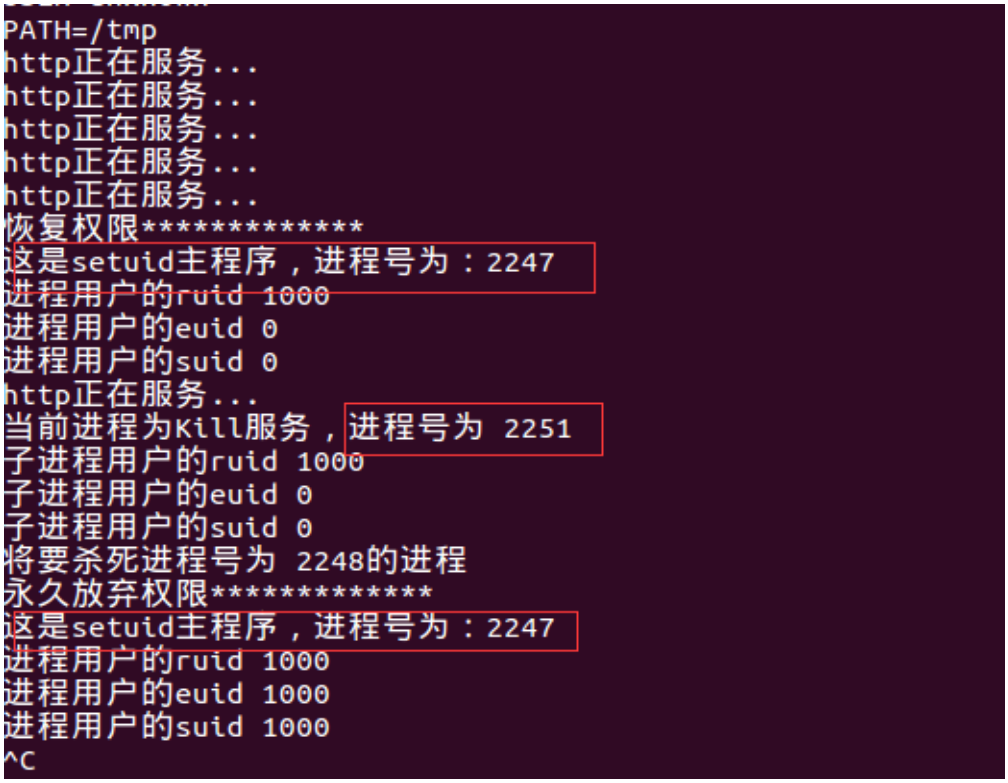
通过对比可以发现 fork 程序继承父进程的 uid。采用 exec 系列函数时，如果设置了 uid 位，则继承 root 的 euid



临时收回权限，采用 execl 的方式调用 echoall 函数，打印环境变量



采用 execle 的当时调用 echoall 函数。可以对比发现，带环境变量的函数，运行在参数指定的环境中，不带环境变量参数的函数，则继承当前环境。同时我们还发现在临时放弃权限调用 echoall 函数时，该进程的 ruid=euid=suid=1000



可以看到恢复权限后，用户权限提高。可以运行 kill 程序。从前面可以知道 http 服务程序的进程号为 2248.在这里看到 kill 要杀死进程为 2248 的进程。执行完 kill 程序后，我们发现 setuid 主进程的 ruid=euid=suid=1000.则用户永久放弃了权限

当用户执行完 root 权限后，接下来要执行普通用户权限程序时，如果后面会继续用到 root 权限，则可以使得用户暂时放弃权限，如果之后不会用到 root 权限，则让用户永久放弃权限。

5. 代码展示

代码清单

- test4.c
- http.c
- echoall.c
- kill.c
- setuid.c

test4.c

```
#include "apue.h"
int main(){
    pid_t pid;
    uid_t ruid,euid,suid;
    getresuid(&ruid,&euid,&suid);
    printf("这是实验 2 主进程，进程号：%d\n",getpid());
    printf("进程用户的 ruid %d\n",ruid);
    printf("进程用户的 euid %d\n",euid);
    printf("进程用户的 suid %d\n",suid);
    if((pid=fork())<0){printf("fork error!\n");
    }else if(pid == 0){
        if(execl("/home/gmm/lab1/1.2/fork",(char *)0) < 0){
            printf("fork 调用错误!\n");
        }
    }
    sleep(1);
    if((pid=fork())<0){printf("fork error!\n");
    }else if(pid == 0){
        if(execl("/home/gmm/lab1/1.2/setuid",(char *)0) < 0){
            printf("setuid 调用错误!\n");
        }       }    }
```

http.c

```
#include "apue.h"
int main(int argc,char *argv[]){
    pid_t pid;
    printf("您已经开启 Http 服务\n");
    printf("当前进程为 Http 服务，进程号为 %d\n",getpid());
    uid_t ruid,euid,suid;
    getresuid(&ruid,&euid,&suid);
    printf("子进程用户的 ruid %d\n",ruid);
    printf("子进程用户的 euid %d\n",euid);
    printf("子进程用户的 suid %d\n",suid);
    while(1){
        printf("http 正在服务...\n");
        sleep(2);
    }
}
```

echoall.c

```
#include "apue.h"
int main(int argc,char *argv[])
```

```
{
    int i;
    char **ptr;
    extern **environ;
    uid_t ruid,euid,suid;
    getresuid(&ruid,&euid,&suid);
    printf("\n 当前进程 ID %d \n",getpid());
    printf("子进程用户的 ruid %d\n",ruid);
    printf("子进程用户的 euid %d\n",euid);
    printf("子进程用户的 suid %d\n",suid);
    for(i = 0;i < argc; i++)
        printf("argv[%d]: %s\n",i,argv[i]);
    for(ptr=environ; *ptr != 0; ptr++)
        printf("%s\n",*ptr);
}
```

kill.c

```
#include "apue.h"
char *join(char *, char*);
int main(int argc,char *argv[]){
    printf("当前进程为 Kill 服务，进程号为 %d\n",getpid());
    uid_t ruid,euid,suid;
```

```
    getresuid(&ruid,&euid,&suid);
    printf("子进程用户的 ruid %d\n",ruid);
    printf("子进程用户的 euid %d\n",euid);
    printf("子进程用户的 suid %d\n",suid);
    printf("将要杀死进程号为 %s 的进程\n",argv[1]);
    system(join("kill ",argv[1]));
}
char* join(char *s1, char *s2)
{
      char *result = malloc(strlen(s1)+strlen(s2)+1);
      if (result == NULL) exit (1);
      strcpy(result, s1);
      strcat(result, s2);
      return result;
}
```

setuid.c

```
#include "apue.h"
char *join(char *, char*);
char *env_init[] ={"USER=unknown","PATH=/tmp",NULL};
int main(){
    pid_t pid;
```

```
char http_pid[20];
uid_t suid,euid,ruid;
getresuid(&ruid,&euid,&suid);
printf("这是 setuid 主程序，进程号为：%d\n",getpid());
printf("进程用户的 ruid %d\n",ruid);
printf("进程用户的 euid %d\n",euid);
printf("进程用户的 suid %d\n",suid);
if((pid=fork())<0){
    printf("fork error!\n");
}else if(pid == 0){
    if(execl("/home/gmm/lab1/1.2/http",(char *)0) < 0){
        printf("http 调用错误!\n");
    }
}else if(pid > 0){
    sprintf(http_pid,"%d",pid);
}
sleep(3);
setresuid(ruid,ruid,euid);//临时收回权限
printf("临时收回权限**********\n");
getresuid(&ruid,&euid,&suid);
printf("这是 setuid 主程序，进程号为：%d\n",getpid());
printf("进程用户的 ruid %d\n",ruid);
```

```
    printf("进程用户的 euid %d\n",euid);
    printf("进程用户的 suid %d\n",suid);
    if((pid=fork())<0){printf("fork error!\n");
    }else if(pid == 0){
        if(execl("/home/gmm/lab1/1.2/echoall","gmm",(char *)0) < 0){
            printf("echoall 调用错误!\n");
        }
    }
          sleep(1);
    if((pid=fork())<0){printf("fork error!\n");
    }else if(pid == 0){
        if(execle("/home/gmm/lab1/1.2/echoall","gmm",(char
*)0,env_init) < 0){
            printf("echoall 调用错误!\n");
        }
    }
    sleep(10);
    setresuid(ruid,suid,suid);//恢复权限
    printf("恢复权限**********\n");
    getresuid(&ruid,&euid,&suid);
    printf("这是 setuid 主程序，进程号为：%d\n",getpid());
    printf("进程用户的 ruid %d\n",ruid);
```

```
    printf("进程用户的 euid %d\n",euid);
    printf("进程用户的 suid %d\n",suid);
    if((pid=fork())<0){printf("fork error!\n");
    }else if(pid == 0){
        if(execl("/home/gmm/lab1/1.2/kill","kill",http_pid,(char *)0) < 0){
            printf("kill 调用错误!\n");
        }
    }
    sleep(1);
    setresuid(ruid,ruid,ruid);//永久放弃权限
    printf("永久放弃权限**********\n");
    getresuid(&ruid,&euid,&suid);
    printf("这是 setuid 主程序，进程号为：%d\n",getpid());
    printf("进程用户的 ruid %d\n",ruid);
    printf("进程用户的 euid %d\n",euid);
    printf("进程用户的 suid %d\n",suid);

}
```

实验心得

通过本次实验，自己对 linux 系统操作有了一个更进一步的认识。同时自己关于 linux 系统中文件和目录的权限管理，通过编写程序，检

验了课堂上讲解的知识。在进行实验二时自己一开始是针对单个要求编写单个小程序的方式，后来经老师指出认识到作为一个计算机学院的学生，应该能把整个知识融会贯通，应该放到实际的应用场景中去编写程序。于是自己将程序重新整合成一个场景，利用该场景完成了第二部分的实验。以后的编程中，自己要把握这种整体应用的思想