

操作系统实验二

添加Linux系统调用及熟悉常见系统调用

主要步骤几核心代码

添加Linux系统调用

源代码

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5
6
7 int main(){
8     char str[20];
9     printf("Give me a string:\n");
10    fgets(str,20,stdin);
11    int num;
12    int str_len = strlen(str);
13    int a = syscall(328, str, str_len-1,&num); //调用新增的str2num系统调用
14
15    int b = syscall(327, num); //调用新增的print_val系统调用
16    return 0;
17 }
```

str2num的实现

```
1 asmlinkage void sys_str2num(char __user *str, int str_len, int __user *ret){
2     char string[str_len];
3     //将字符串从用户空间copy到kernel空间
4     copy_from_user(string,str,str_len); //第一个参数是要拷贝到kernel空间中的地址，第二个参数是用
    户空间中字符串的首地址，第三个参数是字符串的长度（多少个byte）
5     int num=0; //num存放转换之后的数字，初始化为0
6     int i;
7     for(i=0;i<str_len;i++){ //对字符串中的每个位上的字符进行计算
8         num+=((* (str+i))-'0') * pow(10,str_len-1-i); //pow是自己实现的计算次方的函数
9     }
10    //将数字从kernel空间copy到用户空间
11    copy_to_user(ret,&num,4); //第一个参数是用户空间中要被拷贝到的地址，第二个参数是kernel空间中存
    放数字的地址，第三个是被拷贝的数据类型的大小（即多少个byte）
12    return;
13 }
```

print_val的实现

```
1 asmlinkage void sys_print_val(int a){
2     printk(KERN_EMERG "in sys_print_val:%d",a);
3     return;
4 }
```

运行截图

```
QEMU
input: AT Translated Set 2 keyboard as /class/input/input1
device-mapper: ioctl: 4.13.0-ioctl (2007-10-18) initialised: dm-devel@redhat.com
cpuidle: using governor ladder
cpuidle: using governor menu
usbcore: registered new interface driver usbhid
usbhid: v2.6:USB HID core driver
oprofile: using NMI interrupt.
TCP cubic registered
NET: Registered protocol family 10
IPv6 over IPv4 tunneling driver
NET: Registered protocol family 17
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
Using IPI No-Shortcut mode
Freeing unused kernel memory: 1596k freed
input: ImExPS/2 Generic Explorer Mouse as /class/input/input2
INIT SCRIPT

This boot took 11.29 secnods

/bin/sh: can't access tty: job control turned off
/ # ./test
Give me a string:
987654321
in sys_print_val:987654321/ # _
```

熟悉常见的系统调用

源代码

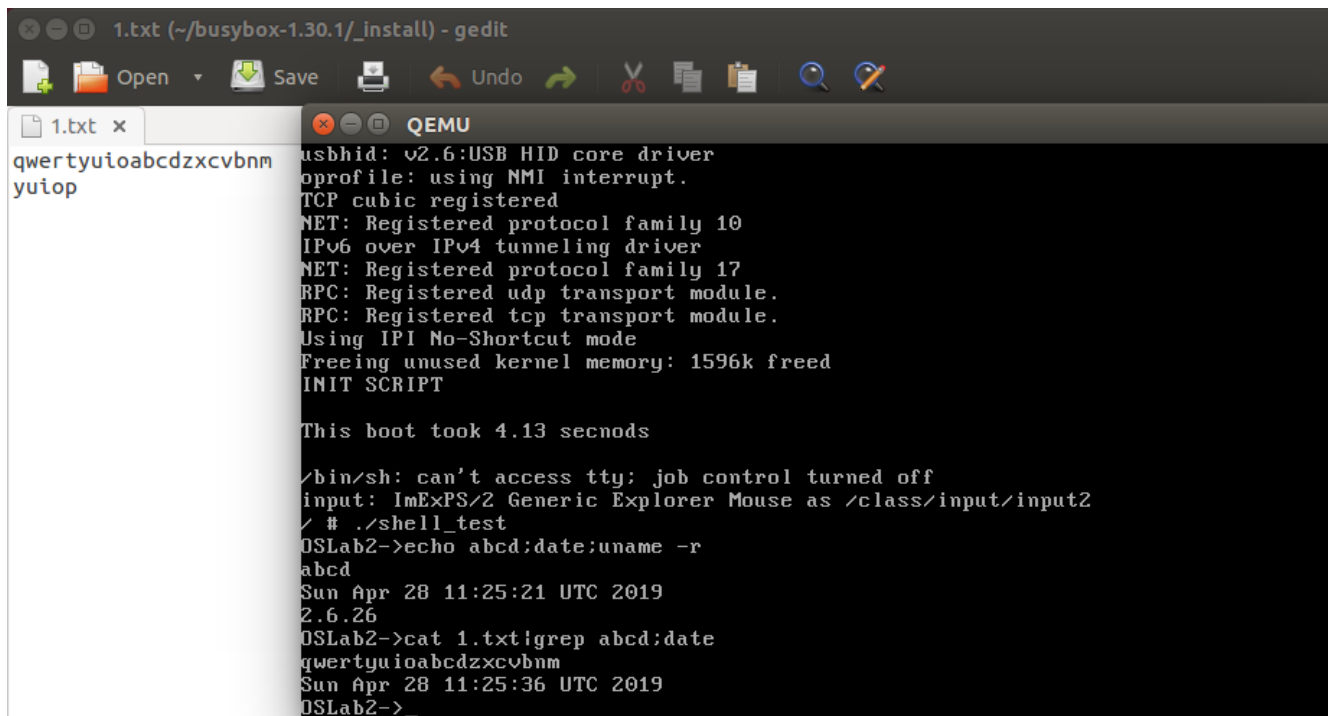
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5
6 void STRCPY(char* cmd, char cmdline[256], int a, int b){//将cmdline的第a位到第b位的指令复制到cmd中
7     int i;
8     for(i=a;i<b;i++){
9         *(cmd+i-a)=cmdline[i];
10    }
11    *(cmd+i-a]='\0';
12    return;
13 }
14
15 int main(){
16     char cmdline[256];//存放多条指令
17     int cmd_num;
18     int cmd_start,cmd_end;//cmd_start和cmd_end之间标记一条指令，cmd_start是指令的第一位，cmd_end是指令最后一位的下一位（通常是分号）
19     int pipe=0;
20     int findcmd=0;
21     int i,p;
22     FILE* fp1;
23     FILE* fp2;
24     while (1) {
25         printf("OSLab2->");
26         fgets(cmdline,256,stdin);
```

```

27     cmd_num=strlen(cmdline);
28     cmd_start=0;
29
30     for (i=0; i<cmd_num; i++) {
31         if(cmdline[i]=='|')pipe=i; //若读到了'|', 说明需要管道, 将管道标志位置为当前数组地址
32         if(cmdline[i]==';' || i==cmd_num-1){ //若读到了';' 或cmdline的末尾, 说明又找到了一
条指令, 将end置为当前地址, findcmd作为标志位置为1
33             findcmd=1;
34             cmd_end=i;
35         }
36         if(findcmd){//若findcmd为1, 说明找打了一条指令, cmd_start和cmd_end将该指令夹住
37             if (pipe) {//处理管道
38                 char *cmd1;
39                 char *cmd2;
40                 cmd1 = malloc((pipe-cmd_start)*sizeof(char));
41                 cmd2 = malloc((cmd_end-pipe-1)*sizeof(char));
42                 STRCPY(cmd1,cmdline,cmd_start,pipe);//cmd1存第一条指令
43                 STRCPY(cmd2,cmdline,pipe+1,cmd_end);//cmd2存第二条指令
44
45                 fp1 = popen(cmd1,"r");//cmd1的文件以读模式打开
46                 if(fp1==NULL){printf("file open failed!");break;}
47
48                 fp2 = popen(cmd2, "w");//cmd2的文件以写模式打开
49                 if(fp2==NULL){printf("file open failed!");break;}
50
51                 char *buffer;//缓冲区暂存cmd1打开的文件中的内容
52                 buffer=(char*)malloc(200*sizeof(char));
53                 //fgets每次读一行, 每读一行, 将buffer中的内容写到fp2中, 直到文件末尾
54                 while(fgets(buffer,200,fp1)!=NULL)fputs(buffer,fp2);
55                 //关闭管道, 释放内存
56                 pclose(fp1);
57                 pclose(fp2);
58                 free(buffer);
59                 free(cmd1);
60                 free(cmd2);
61             }
62             else {//处理非管道
63                 char *cmd;
64                 cmd = (char*)malloc((cmd_end-cmd_start)*sizeof(char));
65                 STRCPY(cmd,cmdline,cmd_start,cmd_end);//cmd存此条指令
66                 system(cmd);//调用system执行此条指令
67                 free(cmd);
68             }
69             //将各个标志位恢复/更新
70             findcmd=0;
71             cmd_start=cmd_end+1;
72             pipe=0;
73         }
74     }
75 }
76 return 0;

```

运行截图



```
1.txt x QEMU
qwertyuioabcdzxcvbnm
yuiop
usbhid: v2.6:USB HID core driver
oprofile: using NMI interrupt.
TCP cubic registered
NET: Registered protocol family 10
IPv6 over IPv4 tunneling driver
NET: Registered protocol family 17
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
Using IPI No-Shortcut mode
Freeing unused kernel memory: 1596k freed
INIT SCRIPT

This boot took 4.13 secnods

/bin/sh: can't access tty: job control turned off
input: ImExPS/2 Generic Explorer Mouse as /class/input/input2
/ # ./shell_test
OSLab2->echo abcd:date;uname -r
abcd
Sun Apr 28 11:25:21 UTC 2019
2.6.26
OSLab2->cat 1.txt|grep abcd:date
qwertyuioabcdzxcvbnm
Sun Apr 28 11:25:36 UTC 2019
OSLab2->_
```

其中1.txt的内容如左所示

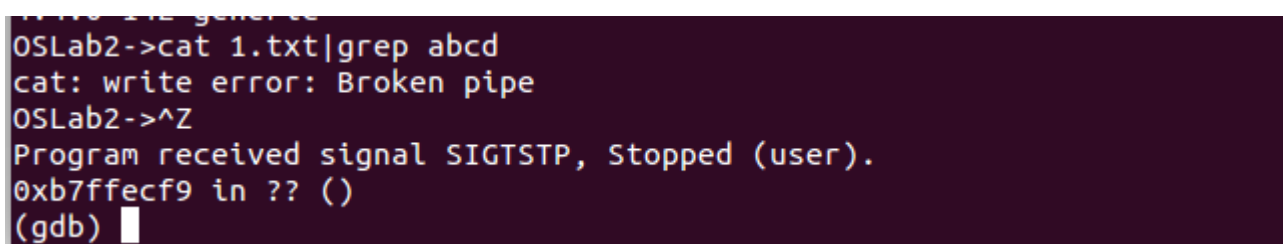
结果分析

如图所示，实验结果均正确。

其中我遇到了一个问题，就是buffer缓冲区的大小如何设置。因为fgets一次读一行，我担心如果一行内字符过多，buffer我设置的200太小会发生错误。然后我试了一下故意将buffer设置很小，设成5，再用上面的1.txt进行实验，发现实验结果还是正确的，说明fgets如果一行内的字符数把buffer要大的话会停止读取，下次从停止的位置继续读取。因此buffer的大小变成了无关紧要的问题。

技术问题

1.



```
OSLab2->cat 1.txt|grep abcd
cat: write error: Broken pipe
OSLab2->^Z
Program received signal SIGTSTP, Stopped (user).
0xb7ffecf9 in ?? ()
(gdb) _
```

调试过程中出现了上面的错误，说Broken pipe。上网搜了一下说是因为管道在还需要从cat中读内容时被grep关闭了。我检查了我的代码发现我提前关闭了cat打开的文件。

问题解答：



The reason is because the pipe is closed by `grep` when it still has some data to be read from `cat`. The signal `SIGPIPE` is caught by `cat` and it exits.

5



What usually happens in a pipeline is the shell runs `cat` in one process and `grep` in another. The stdout of `cat` is connected to the write-end of the pipe and stdin of `grep` to the read end. What happened was `grep` hit a pattern search that did not exist and exited immediately causing the read end of the pipe to be closed, which `cat` does not like since it has some more data to be write out to the pipe. Since the write actions happens to an other which has been closed other end, `SIGPIPE` is caught by the `cat` on which it immediately exits.

For such a trivial case, you could remove the pipeline usage altogether and run it as `grep "pattern" file.txt` when the file's contents are made available over the stdin of `grep` on which it could read from.

我的错误代码：

```
fp1 = popen(cmd1, "r");
if(fp1==NULL){printf("file open failed!");break;}
FILE* fp_temp=fp1;
fseek(fp_temp, 0, SEEK_END);
int len = ftell(fp_temp);
char buffer[len];
fgets(buffer, len, fp1);
fclose(fp1);

fp2 = popen(cmd2, "w");
if(fp2==NULL){printf("file open failed!");break;}
fputs(buffer, fp2);
```

(但fp1是被我关掉的而不是grep，我感觉没问题，但是改掉代码后就好了。)

2.我本来想用feek和ftell判断文件长度来规定缓冲区大小，但是始终调用不成功，于是改了代码。

实验总结

本次试验中，我学会了如何添加系统调用，需要更改四个文件

- include/asm-x86/unistd_32.h (增加系统调用号)
- arch/x86/kernel/syscall_table_32.s (增加调用号和函数的关系)
- include/linux/syscalls.h (增加函数声明)
- kernel/sys.c (函数具体实现)

注1: sys.c中若函数参数中需要用到指向用户空间内的指针，需要添加__user宏，如：

```
1 | asmlinkage void sys_str2num(char __user *str, int str_len, int __user *ret)
```

注2: 从用户空间读数据好写数据时，用copy_from_user和copy_to_user函数，它们的函数声明：

```
1 | unsigned long copy_from_user(void * to, const void __user * from, unsigned long n);
2 | unsigned long copy_to_user(void __user *to, const void *from, unsigned long n);
```

他们的第一个参数都是目标的地址destination，第二个都是原地址sourc，第三个都是要拷贝的字符长度，以八个字节为单位。