

## 2-PB17111568-郭雨轩

### 实验目的

- 向系统中添加自己定义的系统调用
- 编写代码，通过使用系统调用实现一个shell命令行程序

### 实验1

#### 实验步骤

首先按照实验指导书的步骤，先按照实验指导书中需要添加核修改的地方进行添加和修改，截图如下：

```
1. 331 #define __NR_eventfd          323
    332 #define __NR_fallocate        324
    333 #define __NR_timerfd_settime   325
    334 #define __NR_timerfd_gettime  326
    335 #define __NR_print_val        327
    336 #define __NR_str2num          328

2. 326 .long sys_fallocate
    327 .long sys_timerfd_settime /* 325 */
    328 .long sys_timerfd_gettime
    329 .long sys_print_val
    330 .long sys_str2num

3. 616 asmlinkage long sys_eventfd(unsigned int count);
    617 asmlinkage long sys_fallocate(int fd, int mode, loff_t offset, loff_t len);
    618
    619 asmlinkage void sys_print_val(int val);
    620 asmlinkage void sys_str2num(char __user *str, int str_len, int __user *ret);
    621
```

然后编写代码的逻辑部分，`print_val`比较简单，不详细说明了，只需要调用 `printk` 输出即可。

在 `str2num` 函数中，首先先将用户的输入copy到内核中，在内核中完成整个转换的过程，然后再copy会用户空间，两个函数的具体代码如下。

```

1823 asmlinkage void sys_print_val(int val)
1824 {
1825     printk("in_sys_print_val: \n");
1826     printk("%d", val);
1827 }
1828
1829 asmlinkage void sys_str2num(char __user *str, int str_len, int __user *ret)
1830 {
1831     char kernel_str[100];
1832     int i,value;
1833     copy_from_user(kernel_str, str, str_len);
1834     for (i=0,value=0;kernel_str[i]!='\0';++i)
1835     {
1836         value*=10;
1837         value+=(kernel_str[i]-'0');
1838     }
1839     copy_to_user(ret,&value,sizeof(int ));
1840 }

```

随后编写了测试的程序，使用两个系统调用实现了在控制台输出转换后的字符串。

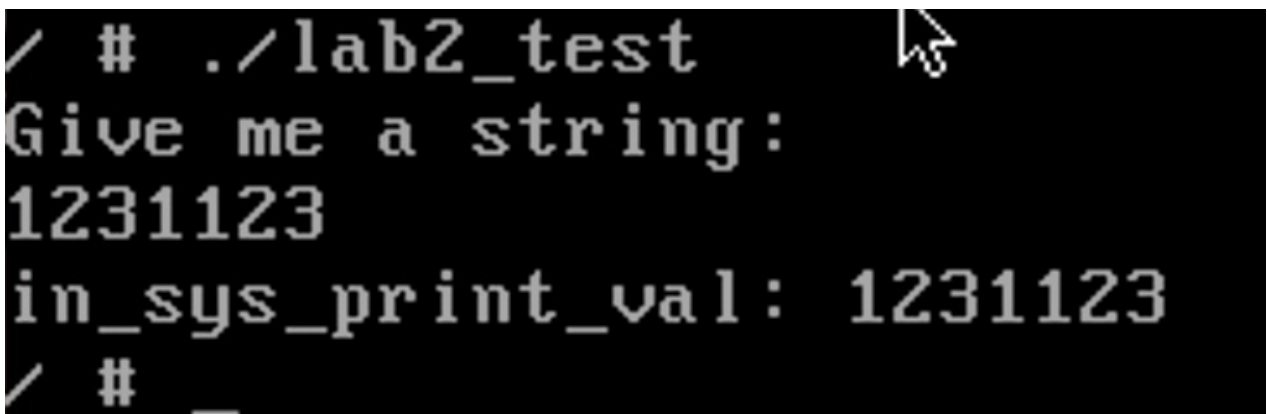
测试代码

```

1  #include <stdio.h>
2  #include <sys/syscall.h>
3  #include <unistd.h>
4  #include <string.h>
5  #include <linux/kernel.h>
6  #include <stdarg.h>
7
8  int main (void)
9  {
10     char str[100];
11     int ret;
12     printf("Give me a string:\n");
13     scanf("%s",str);
14     int len = strlen(str);
15     syscall(328,&str,len+1,&ret);
16     syscall(327,ret);
17     return 0;
18 }

```

控制台运行截图：



```

/ # ./lab2_test
Give me a string:
1231123
in_sys_print_val: 1231123
/ # _

```

## 实验2

### 实验思路

- 首先要熟悉各种系统调用的使用方法，综合助教在官网上给出的系统调用的函数和某付大佬给出的建议，我使用了 `execvp()`、`popen()`、`pclose()` 这几个系统调用。期中 `execvp()` 需要将每一个参数以字符串的形式传入，而 `popen()` 则需要将整个指令作为一个字符串传入。
- 其次就是如何处理控制台的输入，我使用两个变量 `begin` 和 `end` 来界定当前命令的范围，使用 `pipe_flag` 来标志这个命令中是否有管道符号，若有则将其置为管道符号的下标，若无则置为 0，根据有无管道符号分开处理每个指令。
- 在遍历指令的每一个字符的时候，我同时用一个数组记录了每个参数的起始地址，同时将指令中的空格和分号置为 `'\0'` 统计完当前指令中参数的个数的时候，会动态分配一个 `char *` 的数组，并按照格式将各个参数的起始地址写入。随后再 `fork()` 生成子进程并调用 `execvp()`。

### 实验代码

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <sys/syscall.h>
4  #include <unistd.h>
5  #include <stdlib.h>
6
7  int main (void)
8  {
9      char buffer[256];
10     while (1)
11     {
12         int i;
13         printf("OSLab2->");
14         gets(buffer);
15         int len=strlen(buffer);
16         int begin=0,end=0;
17         int pipe_flag=0;
18         char addr[256];
19         for (i=0;i<256;++i)
20             addr[i]=0;//all mem in argv is set to NULL
21         addr[0]=0;
22         int count=0;
23         for (end=0;end<=len;++end){
24             if (buffer[end]=='|'){
25                 pipe_flag=end;
26             }
27             else if (buffer[end]==';' || buffer[end]=='\0'){
28                 buffer[end]='\0';
29                 if (pipe_flag==0){
30                     char **argv = (char **)malloc(sizeof(char *)*(count+2));
31                     for (i=0;i<=count;++i){
```

```

32     argv[i]=&buffer[addr[i]];
33 }
34 argv[count+1]=NULL;
35 pid_t pid = fork();
36 if (pid==0){
37     execvp(argv[0],argv);
38 }
39 else {
40     waitpid(pid,NULL,0);
41 }
42 free(argv);
43 }
44 else {
45     for (i=begin;i<pipe_flag-1;++i){
46         if (buffer[i]=='\0')
47             buffer[i]=' ';
48     }
49     for (i=pipe_flag+2;i<end;++i){
50         if (buffer[i]=='\0')
51             buffer[i]=' ';
52     }
53     char *argv1=&buffer[begin];
54     char *argv2=&buffer[pipe_flag+2];
55     FILE *f1 = popen(argv1, "r");
56     FILE *f2 = popen(argv2, "w");
57     char data[1000];
58     fread(data,1000*sizeof(char),1,f1);
59     fwrite(data,1000*sizeof(char),1,f2);
60     pclose(f1);
61     pclose(f2);
62
63 }
64 begin=end+1;
65 pipe_flag=0;
66 for (i=0;i<256;++i)
67     addr[i]=0;
68 count=-1;
69
70 }
71 else if (buffer[end]==' '){
72     buffer[end]='\0';
73 }
74 else {
75     if (end>0 && buffer[end-1]=='\0')
76     {
77         count++;
78         addr[count]=end;
79     }
80 }

```

```
81     }
82 }
83 return 0;
84 }
```

## 运行截图

The screenshot shows a terminal window with the following commands and output:

```
OSLab2->ls -a;seq 1 3 10
.          bin          lab2_test      root          usr
..         dev          linuxrc       sbin
.ash_history  init          myshell       tmp
1. 1
   4
   7
  10
OSLab2->
OSLab2->seq 1 4 13 | grep ^[12]
1
2. 13
   OSLab2->_
```

## 实验总结

本次实验进行的比较顺利，一些奇奇怪怪的错误没有，只要按照指导书上面基本的流程就可以顺利的做完，难点主要是在处理字符串输入并正确的使用系统调用，在这边我没有踩坑主要感谢助教的指导书和群里面的大佬。

## 实验收获

本次实验我熟悉了linux的系统调用，同时熟悉了c语言在linux下面的编程。同时，通过向系统添加系统调用，我基本了解了这个linux系统源代码的组织以及结构，同时对宏 `__user` 有了一定的了解，总的来说收获很大。