

操作系统实验 2

题目：添加 Linux 系统调用及熟悉常见系统调用

廖洲洲，PB17081504

一、添加 Linux 系统调用

1) 分配系统调用号，修改系统调用表

- 在 Linux 源代码根目录下，找到 include/asm/unistd_32.h 文件，在文件末尾新增两个系统调用号。

```
#define __NR_print_val          327
#define __NR_str2num            328
```

- 在 arch/x86/kernel/syscall_table_32.S 文件末尾添加系统调用号和调用函数的对应关系。

```
.long sys_print_val
.long sys_str2num
```

2) 实现系统调用函数

- 在 include/linux/syscalls.h 中声明新增的两个系统调用函数。

```
asmlinkage long sys_print_val(int a);
asmlinkage long sys_str2num(char __user *str, int str_len, int
__user *ret);
```

- 在 kernel/sys.c 中实现新增的两个系统调用函数。

sys_print_val 直接调用了 printk 这个函数，直接在控制台进行打印。
sys_str2num 的功能是将一个有 str_len 个数字的字符串 str 转换成十进制数字，然后将结果写到 ret 指向的地址中。它主要是通过调用 copy_from_user 和 copy_to_user 这两个函数实现的，copy_from_user 先从用户端读取字符串到 driver_to，然后将 driver_to 转换为十进制数字保存在 value 中，再通过 copy_to_user 将 value 返回给用户端。

核心代码：

```
asmlinkage long sys_print_val(int a)
{
```

```
        printk("%d\n", a);
        return 1;
    }

    asm linkage long sys_str2num(char __user *str, int
str_len, int __user *ret)
    {
        char driver_to[100];
        int value=0;
        int i=0;
        copy_from_user(driver_to, str, (unsigned long)str_len);
        for(i=0; i<str_len; i++) {
            value *=10;
            value +=driver_to[i] - '0';
        }

        copy_to_user(ret, &value, sizeof(int));
        return 1;
    }
```

3) 编译内核

4) 编写测试文件

test.c:

```
int main() {
    printf("Give me a string:\n");
    char str[100];
    int str_len;
    int ret;
    scanf("%s", str);
    str_len = strlen(str);
    printf("GET:%s---length:%d\n", str, str_len);
}
```

```

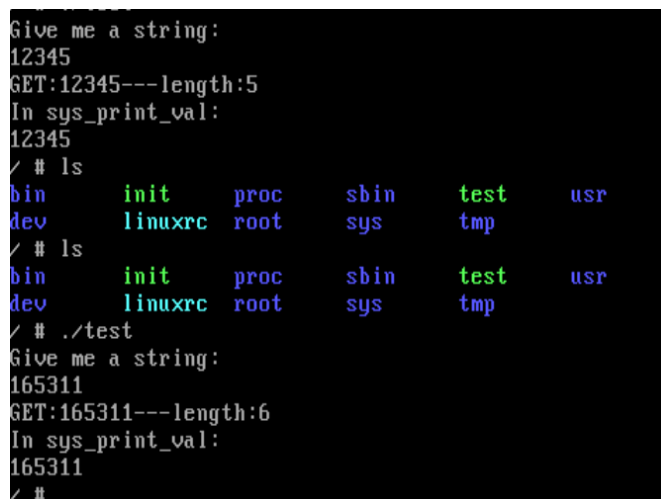
printf("In sys_print_val:\n");
syscall(328, str, str_len, &ret);

syscall(327, ret);

return 1;
}

```

5) 结果截图



```

Give me a string:
12345
GET:12345---length:5
In sys_print_val:
12345
/ # ls
bin      init     proc     sbin     test     usr
dev      linuxrc  root     sys      tmp
/ # ls
bin      init     proc     sbin     test     usr
dev      linuxrc  root     sys      tmp
/ # ./test
Give me a string:
165311
GET:165311---length:6
In sys_print_val:
165311
/ #

```

实现了将字符串转换为十进制数

6) 遇到的问题

- 在编写 `str2num.c` 函数时，错将 `str_len` 打错为 `strlen`，结果内核编译时未报错，在运行 `test.c` 时出现段错误，修改后即通过。
- 在编写 `sys_print_val` 时，同学提醒 `printk` 要添加优先级，否则无法输出到控制段，但我查资料了解到其实不添加优先级的话，`printk` 默认的优先级是可以输出到控制端的，故不添加优先级也是可以的。

二、熟悉 Linux 下常见的系统调用并编写一个 shell

1) 使用的系统调用

```

pid_t fork();//创建进程

pid_t waitpid(pid_t pid,int* status,int options);

//等待 pid 的子进程结束

int execv(const char* path,char* constargv[]);

```

//根据指定的文件名或目录名找到可执行文件，并用它来取代原调用进程的数据段、代码段和堆栈段

```
int system(const char* command);
```

//调用 fork() 产生子进程，在子进程执行参数 command 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

```
FILE* popen(const char* command, const char* mode);
```

//popen 函数先执行 fork，然后调用 exec 以执行 command，并且根据 mode 的值（"r"或"w"）返回一个指向子进程的 stdout 或指向 stdin 的文件指针

```
int pclose(FILE*stream);
```

//关闭标准 I/O 流，等待命令执行结束

2) 实现 shell 程序

核心代码：

```
//cmdline 存储输入的命令行
```

```
//cmd 存储了分割的命令
```

```
while(1){
```

```
    printf("OSLab2->");
```

```
    fgets(cmdline, 256, stdin); //输入命令
```

```
    cmd_len=strlen(cmdline); //取长度
```

```
    cmd_num=1; //每次都要将命令个数置 1
```

```
    for(i=0;i<cmd_len;i++) //命令数
```

```
        if(cmdline[i]==';')
```

```
            cmd_num++;
```

```
    j=0; //j 用于分割命令行
```

```
    for(i=0;i < cmd_num ;i++) {
```

```
        pipeflag=0; //判断是否有管道
```

```
        for(k=0;cmdline[j]!=';'&& j<cmd_len;j++,k++){
```

```
            cmd[k]=cmdline[j];
```

```
            if(cmdline[j]=='|'){
```

```
                pipeflag=1; //有管道，pipeflag 置 1
```

```

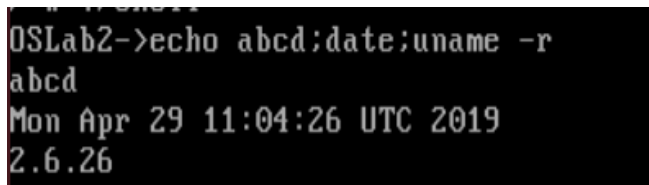
        pipeptr=k;
    }
}

cmd[k]='\0';
j++;

//      printf("%d:%s\n",i,cmd);
if(pipeflag)    {    //有管道
    fp=popen(cmd,"r");//执行命令将输出流指针赋给 fp
    while(fgets(out,256,fp)!=NULL)
        fputs(out,stdout); //将输出输出到控制台
    pclose(fp);
}
else    //无管道
    system(cmd);
}
}

```

3) 结果截图

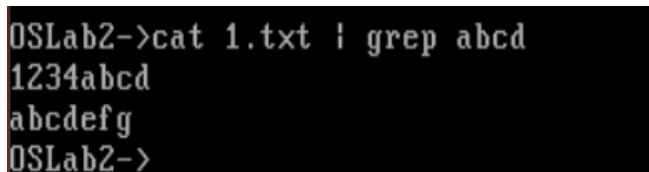


```

OSLab2->echo abcd;date;uname -r
abcd
Mon Apr 29 11:04:26 UTC 2019
2.6.26

```

echo abcd;date;uname -r
命令的执行和输出



```

OSLab2->cat 1.txt | grep abcd
1234abcd
abcdefg
OSLab2->

```

含管道命令的执行和输出

4) 遇到的问题

- 要注意将多个组合命令的分割。
- fputs 无法直接将输出流 fp 的内容输出，要将其先读出写在数组中，再将其输出到控制台。

三、 实验总结

- 通过这次实验我学习了如何添加 Linux 系统调用
- 熟悉了 Linux 下常见的系统调用，并且通过这些系统调用实现了一个 shell