

操作系统实验 2

PB 18111707 吕瑞

实验目的

- 了解系统调用的基本过程
- 学习如何添加 Linux 系统调用
- 熟悉 Linux 下常见的系统调用

实验环境

- OS: Ubuntu 18.04
- Linux 内核版本: Kernel 0.11

实验内容

第一部分 添加 Linux 系统调用

1. 分配系统调用号，修改系统调用表

kernel/system_call.s

修改调用个数：添加两个系统调用，72 改为 74；

```
1 | nr_system_calls = 74
```

include/unistd.h

增加系统调用功能号 72，73，并在下面同步增加系统调用的原型函数；

```
1 | #define __NR_print_val    72
2 | #define __NR_str2num      73
3 |
4 | /*声明供用户调用的函数*/
5 | int print_val(int a);
6 | int str2num(char *str, int str_len, long *ret);
```

include/linux/sys.h

修改函数指针表

```
1 | extern int sys_print_val();
2 | extern int sys_str2num();
3 |
4 | fn_ptr sys_call_table[]={..., sys_print_val, sys_str2num}
```

2. 实现系统调用函数

kernel/print_str.c

```

1  #define __LIBRARY__
2  #include <asm/segment.h>
3  #include <unistd.h>
4
5  int sys_print_val(int a){
6      printk("int print_val: %d \n",a);
7      return 0;
8  }
9
10 int sys_str2num(char *str, int str_len,long *ret){
11     char num[str_len];
12     for(int i = 0;i < str_len ; i++){
13         num[i] = get_fs_byte(str+i);
14     }
15
16     int sum = 0;
17     int j = 0;
18     while(j < str_len - 1){
19         sum = (sum + (num[j] - '0'))*10;
20         j++;
21     }
22     sum = sum + (num[str_len-1] - '0');
23     put_fs_long(sum,ret);
24     return 0;
25 }
26

```

修改 kernel/Makefile

```

1  OBJs = sched.o system_call.o traps.o asm.o fork.o \    panic.o printk.o
2  vsprintf.o sys.o exit.o \    signal.o mktime.o
3  /*改为*/
4  OBJs = sched.o system_call.o traps.o asm.o fork.o \    panic.o printk.o
5  vsprintf.o sys.o exit.o \    signal.o mktime.o xxx.o
6  /*文件最后需要新增*/
7  xxx.s xxx.o: xxx.c ../include/asm/segment.h

```

编译内核

```

1  make clean
2  make

```

编写测试程序

能从终端读取一串字符串，通过 str2num 系统调用转换成数字，并通过 print_val 系统调用打印该数字。

```

1  #define __LIBRARY__
2
3  #include <unistd.h>
4  #include <stdio.h>
5  #include <string.h>
6
7  _syscall1(int, print_val, int, a);
8  _syscall3(int, str2num, char *,str, int, str_len, long *, ret);

```

```

9
10 #define MAXSIZE 8
11
12 int main(int argc, char *argv[]){
13     int str_len;
14     long b;
15     char str[MAXSIZE];
16     printf("Give me a string:\n");
17     scanf("%s",str);
18     str_len = strlen(str);
19     str2num(str,str_len,&b);
20     print_val(b);
21     return 0;
22 }

```

运行测试程序

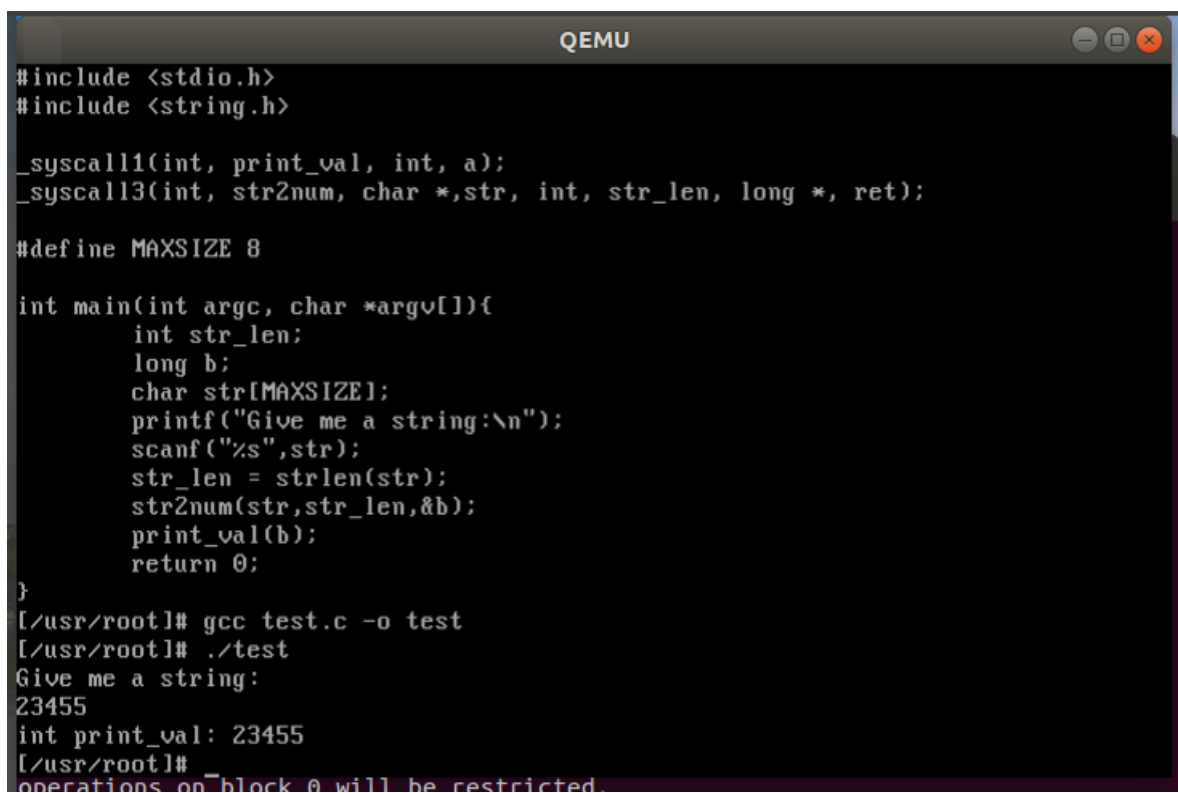
进入 Linux-0.11 系统内 (qemu 中) , gcc 编译用户程序。

```
1 | gcc test.c -o test
```

执行程序

```
1 | ./test
```

3. 结果展示



```

QEMU

#include <stdio.h>
#include <string.h>

_syscall1(int, print_val, int, a);
_syscall3(int, str2num, char *,str, int, str_len, long *, ret);

#define MAXSIZE 8

int main(int argc, char *argv[]){
    int str_len;
    long b;
    char str[MAXSIZE];
    printf("Give me a string:\n");
    scanf("%s",str);
    str_len = strlen(str);
    str2num(str,str_len,&b);
    print_val(b);
    return 0;
}

[/usr/root]# gcc test.c -o test
[/usr/root]# ./test
Give me a string:
23455
int print_val: 23455
[/usr/root]#
operations on block 0 will be restricted

```

回答问题

1. 简要描述如何在 Linux-0.11 添加一个系统调用

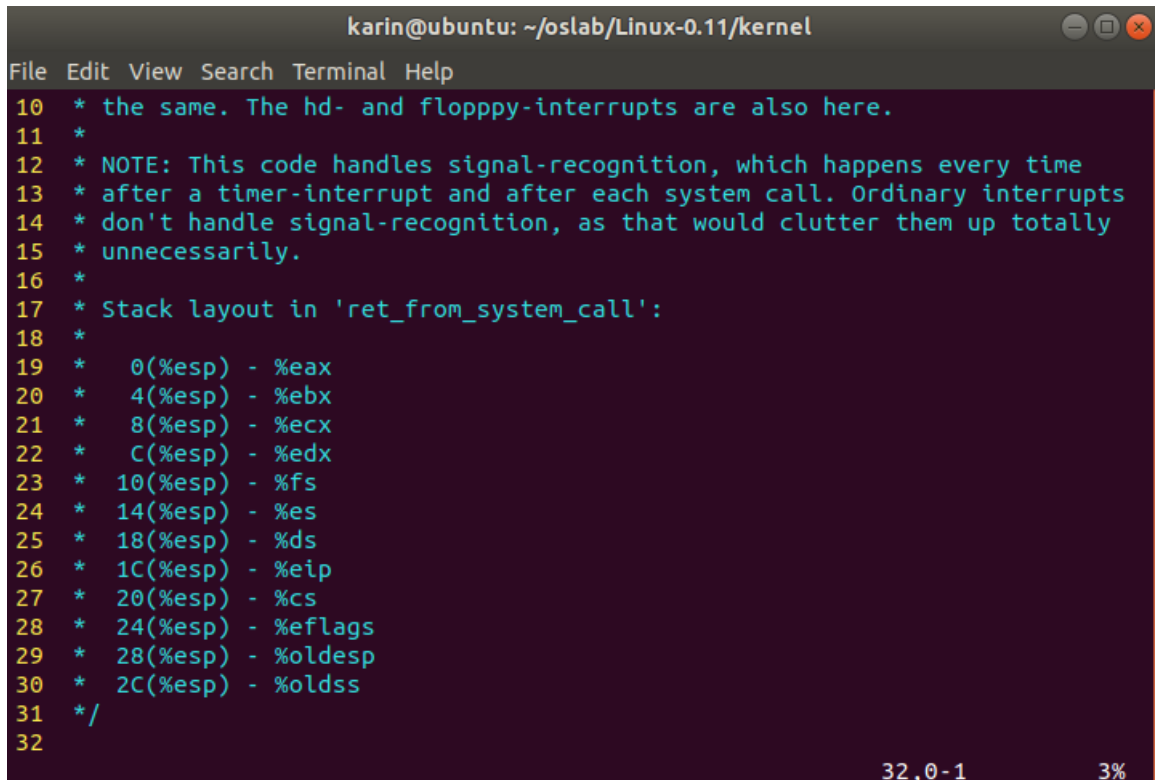
答：修改系统调用号的总数；增加系统调用功能号，并进行函数声明；修改函数指针表；在 kernel 中实现新增的系统调用函数；修改 Makefile 文件；重新编译内核代码。

2. 系统是如何通过系统调用号索引到具体的调用函数的？

答：系统调用功能号实际上对应于 include/linux/sys.h 中定义的系统调用处理程序指针数组表 sys_call_table[] 中项的索引值。sys_call_table[] 中的项都是 sys_xxx，在内核源码中实现的函数就是 sys_xxx()。

3. 在 Linux 0.11 中，系统调用最多支持几个参数？有什么方法可以超过这个限制吗？

答：最多支持三个参数，这是由参数寄存器的个数限制的。



```
karin@ubuntu: ~/oslab/Linux-0.11/kernel
File Edit View Search Terminal Help
10 * the same. The hd- and floppy-interrupts are also here.
11 *
12 * NOTE: This code handles signal-recognition, which happens every time
13 * after a timer-interrupt and after each system call. Ordinary interrupts
14 * don't handle signal-recognition, as that would clutter them up totally
15 * unnecessarily.
16 *
17 * Stack layout in 'ret_from_system_call':
18 *
19 * 0(%esp) - %eax
20 * 4(%esp) - %ebx
21 * 8(%esp) - %ecx
22 * C(%esp) - %edx
23 * 10(%esp) - %fs
24 * 14(%esp) - %es
25 * 18(%esp) - %ds
26 * 1C(%esp) - %eip
27 * 20(%esp) - %cs
28 * 24(%esp) - %eflags
29 * 28(%esp) - %oldesp
30 * 2C(%esp) - %oldss
31 */
32
32,0-1 3%
```

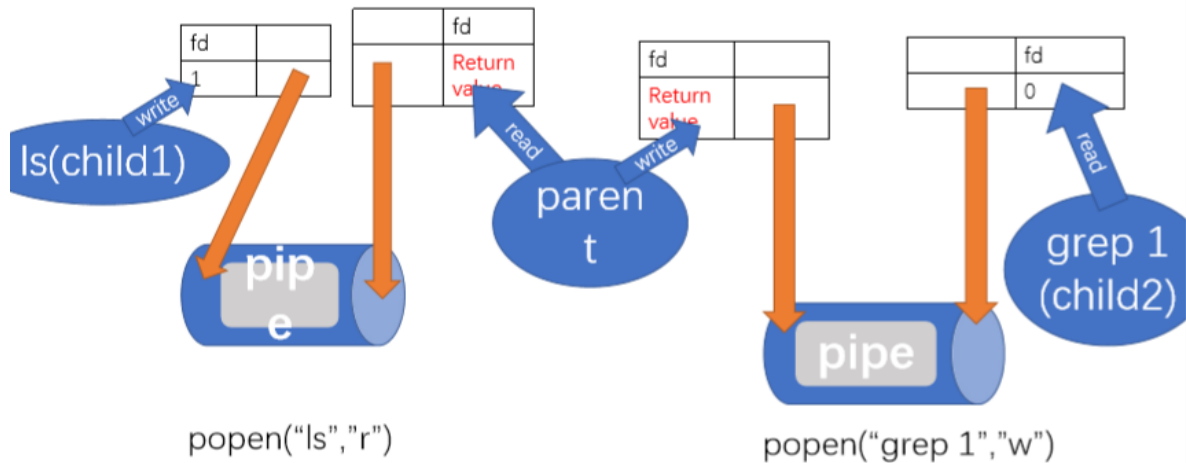
想要超过限制，就修改 kernel 中的 system_call.s 中的参数寄存器个数。

第二部分 熟悉 Linux 下常见的系统调用

1. 利用系统调用创建相关函数

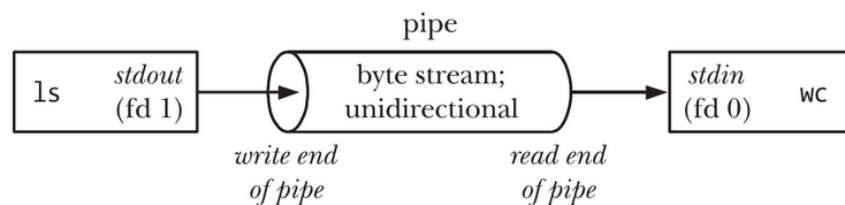
popen 管道读写实现

ls | grep 1



type == 'r': 子进程写入，关闭读端，打开写端，将程序的标准输出重定向到写端文件。

type == 'w': 父进程写入，子进程 read，子进程关闭写端，打开读端文件，并把程序的标准输入重定向到读端文件。



```

1  /* popen, 输入为命令和类型("r""w"), 输出执行命令进程的I/O文件描述符 */
2  int os_popen(const char* cmd, const char type){
3      int i, pipe_fd[2], proc_fd;
4      pid_t pid;
5
6      ...
7
8      /* 2. 子进程部分 */
9      else if (pid == 0){
10         if (type == 'r'){
11             /* 2.1 关闭pipe无用的一端，将I/O输出发送到父进程 子进程写入*/
12             close(pipe_fd[READ_END]);
13             if (pipe_fd[WRITE_END] != STDOUT_FILENO) {
14                 dup2(pipe_fd[WRITE_END], STDOUT_FILENO);
15                 close(pipe_fd[WRITE_END]);
16             }
17         } else {
18             /* 2.2 关闭pipe无用的一端，接收父进程提供的I/O输入 子进程 read*/
19             close(pipe_fd[WRITE_END]);
20             if (pipe_fd[READ_END] != STDIN_FILENO) {
21                 dup2(pipe_fd[READ_END], STDIN_FILENO);
22                 close(pipe_fd[READ_END]);
23             }
24         }
25         /* 关闭所有未关闭的子进程文件描述符（无需修改） */
26         for (i=0; i<NR_TASKS; i++){
27             if(child_pid[i]>0)

```

```

28         close(i);
29         /* 2.3 通过exec系统调用运行命令 */
30         exec1(SHELL, "sh", "-c", cmd, (char*)NULL);
31         /* 也可使用execlp execvp等 */
32         _exit(127);
33     }
34
35     /* 3. 父进程部分 */
36     else {
37         if (type == 'r') {
38             close(pipe_fd[WRITE_END]);
39             proc_fd = pipe_fd[READ_END];
40         }
41         else {
42             close(pipe_fd[READ_END]);
43             proc_fd = pipe_fd[WRITE_END];
44         }
45         child_pid[proc_fd] = pid;
46         return proc_fd; /*返回一个指向子进程的 stdout 或 stdin 的文件指针*/
47     }
48 }

```

os_system

实现较为简单，注意父进程要等待子进程运行结束。

```

1  int os_system(const char* cmdstring) {
2      pid_t pid;
3      int stat;
4
5      if(cmdstring == NULL) {
6          printf("nothing to do\n");
7          return 1;
8      }
9
10     /* 4.1 创建一个新进程 */
11     pid = fork();
12
13     if (pid < 0){
14         printf("FORK FAILED!");
15         return NULL;
16     }
17     /* 4.2 子进程部分 */
18     else if (pid == 0){
19         exec1(SHELL, "sh", "-c", cmdstring, (char*)NULL);
20     }
21
22     /* 4.3 父进程部分：等待子进程运行结束 */
23     else {
24         while (waitpid(pid, &stat, 0)<0)
25             if(errno != EINTR)
26                 return -1;
27     }
28
29     return stat;
30 }

```

2. main() 函数中的调用

使用管道：

cmd1 标准输出的内容在保存在 fd1 中；取出来，放到 buffer 里；

把 buffer 中的内容取出来放到 fd2 中，通过管道传递为 cmd2 的标准输入。

```
1  for(i=0;i<cmd_num;i++){
2      char *div = strchr(cmds[i], '|'); /*一条命令里面包含管道*/
3      if (div) {
4          /* 如果需要用到管道功能 */
5          ...
6          /* 5.1 运行cmd1, 并将cmd1标准输出存入buf中 */
7          count = 4096;
8          zeroBuff(buf,count);
9          fd1 = os_popen(cmd1,'r');
10         read(fd1,buf,count);
11         status = os_pclose(fd1);
12
13         /* 5.2 运行cmd2, 并将buf内容写入到cmd2输入中 */
14         fd2 = os_popen(cmd2,'w');
15         write(fd2,buf,strlen(buf)+1);
16         status = os_pclose(fd2);
17
18     }
19     else {
20         /* 6 一般命令的运行 */
21         status = os_system(cmds[i]);
22     }
23 }
```

3. 结果展示

```
karin@ubuntu:~/oslab/Linux-0.11/OS-LAB/Lab2_New/EXP2.2$ ./lab2_shell
os shell ->echo abcd;date;uname -r
abcd
Sat May  2 22:39:21 PDT 2020
5.3.0-46-generic
os shell ->ls | grep -a l
cmd1: ls
cmd2:  grep -a l
lab2_shell
lab2_shell.c
os shell ->
```

```
lab2_shell.c: In function os_popen:
lab2_shell.c:28: warning: return of integer from pointer lacks a cast
lab2_shell.c:32: warning: assignment of pointer from integer lacks a cast
lab2_shell.c:33: warning: return of integer from pointer lacks a cast
lab2_shell.c:38: warning: return of integer from pointer lacks a cast
lab2_shell.c:45: warning: return of integer from pointer lacks a cast
lab2_shell.c: In function os_system:
lab2_shell.c:120: warning: return of integer from pointer lacks a cast
lab2_shell.c: In function parseCmd:
lab2_shell.c:143: warning: initialization of integer from pointer lacks a cast
[/usr/root/lab2]# ./lab2_shell
os shell ->echo 1234;date;uname -r
1234
Sun May  3 05:45:23 2020
uname: command not found
os shell ->ls | grep -a l
cmd1: ls
cmd2: grep -a l
usage: grep [-CUbchilnsuwx] [-<num>] [-AB <num>] [-f file] [-e] expr [files]
os shell ->ls | grep l
cmd1: ls
cmd2: grep l
lab2_shell
lab2_shell.c
os shell ->
```