操作系统课程实验报告（三）

原创

Learner_ZWJ 于 2017-07-29 10:05:14 发布 4312 收藏 3
分类专栏： 操作系统学习 文章标签： 操作系统 linux

这是实验楼操作系统课程（李治军老师教学）的第三个实验——系统调用——的实验报告。

报告正文

我认为做这个实验，先在脑海中把系统调用的过程理一次，再动手会好一些，以下实验报告也会根据这个思路进行。

一个应用程序，在编译的过程中，将它使用的 API展开成包含一系列（也有可能没有）系统调用的语句。根据我现在的理解，也就是展开成调用了 int 0x80中断的内嵌汇编的形式。

首先从 int 0x80入手修改。int 0x80会在 IDT 中找到相对应的中断描述符，从而找到中断例程的入口，然后进行执行这个例程，到这一步为止我们都还并不需要修改文件。

接下来，int 0x80 会去执行system_call 函数了，kernel/system_call.s是我们需要修改的，仅需要改nr_system_calls = 74，这个数字说明int 0x80中断可以提供多少种功能（调用），由于我们需要添加两个调用，所以将72改为74。

```
nr_system_calls = 74

/*
 * Ok, I get parallel printer interrupts while using the floppy for some
 * strange reason. Urgel. Now I just ignore them.
 */
.globl system_call,sys_fork,timer_interrupt,sys_execve
.globl hd_interrupt,floppy_interrupt,parallel_interrupt
.globl device_not_available, coprocessor_error

.align 2
bad_sys_call:
        movl $-1,%eax
        iret
.align 2
reschedule:
        pushl $ret_from_sys_call
        jmp schedule
.align 2
system_call:
        cmpl $nr_system_calls-1,%eax
        ja bad_sys_call
        push %ds
        push %es
        push %fs
        pushl %edx
        pushl %ecx              # push %ebx,%ecx,%edx as parameters
        pushl %ebx              # to the system call
        movl $0x10,%edx         # set up ds,es to kernel space
        mov %dx,%ds
        mov %dx,%es
        movl $0x17,%edx         # fs points to local data space
```


现在system_call要执行了，它根据传进来的系统调用号（放在eax中）来执行相应的系统调用，它根据include/linux/sys.h中的定义的函数指针数组来找到这个调用的函数入口，所以我们要在sys.h中加入我们需要的两个调用的函数指针 sys_iam 和 sys_whoami，然后用两条extern声明表示这两个函数的实现在别的文件里。

```
extern int sys_ssetmask();
extern int sys_setreuid();
extern int sys_setregid();
extern int sys_whoami();
extern int sys_iam();

fn_ptr sys_call_table[] = { sys_setup, sys_exit, sys_fork, sys_read,
sys_write, sys_open, sys_close, sys_waitpid, sys_creat, sys_link,
sys_unlink, sys_execve, sys_chdir, sys_time, sys_mknod, sys_chmod,
sys_chown, sys_break, sys_stat, sys_lseek, sys_getpid, sys_mount,
sys_umount, sys_setuid, sys_getuid, sys_stime, sys_ptrace, sys_alarm,
sys_fstat, sys_pause, sys_utime, sys_stty, sys_gtty, sys_access,
sys_nice, sys_ftime, sys_sync, sys_kill, sys_rename, sys_mkdir,
sys_rmdir, sys_dup, sys_pipe, sys_times, sys_prof, sys_brk, sys_setgid,
sys_getgid, sys_signal, sys_geteuid, sys_getegid, sys_acct, sys_phys,
sys_lock, sys_ioctl, sys_fcntl, sys_mpx, sys_setpgid, sys_ulimit,
sys_uname, sys_umask, sys_chroot, sys_ustat, sys_dup2, sys_getppid,
sys_getpgrp, sys_setsid, sys_sigaction, sys_sgetmask, sys_ssetmask,
sys_setreuid,sys_setregid,sys_iam,sys_whoami };
```


来到这里，我们已经指明了 int 0x80中断怎么寻找我们新增的两个调用了，接下来就是真正去实现sys_iam和sys_whoami了，将这两个函数的实现放在kernel/who.c中。这里我写了好几个printk函数用于调试，因为我不会用gdb。说实话编写who.c的时候我遇到的最大的困难是不知道要包含哪些头文件，我只包含了几个我能搞清楚的头文件（包括printk的头文件等），侥幸实验成功。

who.c (~/osHomework/linux-0.11/kernel) - gedit

打开(O) 保存(S)

```
#include <linux/kernel.h>
#include <asm/segment.h>
#include <errno.h>

char namebuf[24] = {0};
int namesize = 0;

int sys_iam(const char * name)
{
        int letterNum = 0, i;
        printk("You are in iam\n");
        while (get_fs_byte(name+letterNum) != '\0' && letterNum != 24) {
                letterNum++;
        }
        if (letterNum >= 24) {
                errno = EINVAL;
                return -1;
        } else {
                for (i = 0; i <= letterNum; i++) {
                        namebuf[i] = get_fs_byte(name+i);
                }
                printk(namebuf);
                printk("\n");
                namesize = letterNum;
                return namesize;
        }
}

int sys_whoami(char * name, unsigned int size)
{
        int i;
        printk("You are in whoami\n");
        printk(namebuf);
        if (size < namesize) {
                errno = EINVAL;
                return -1;
        } else {
                for (i = 0; i <= namesize; i++) {
                        put_fs_byte(namebuf[i], (name+i));
                }
                return namesize;
        }
}
```


最后，我们修改一下makefile，这个跟着实验提示做就好，一般没啥大问题。

make完了，就跑虚拟机吧！

在虚拟机下面，修改usr/include/unistd.h，增加两个调用号的宏和函数声明，将来编写的应用程序里的API包含这个头文件，才能顺利展开成包含int 0x80 的形式。

```
#define __NR_getpgrp    65
#define __NR_setsid     66
#define __NR_sigaction  67
#define __NR_sgetmask   68
#define __NR_ssetmask   69
#define __NR_setreuid   70
#define __NR_setregid   71
#define __NR_iam        72
#define __NR_whoami     73
```

```
int uname(struct utsname * name);
int unlink(const char * filename);
int ustat(dev_t dev, struct ustat * ubuf);
int utime(const char * filename, struct utimbuf * times);
pid_t waitpid(pid_t pid,int * wait_stat,int options);
pid_t wait(int * wait_stat);
int write(int fildes, const char * buf, off_t count);
int dup2(int oldfd, int newfd);
int getppid(void);
pid_t getpgrp(void);
pid_t setsid(void);
int iam(const char *name);
int whoami(char *name, unsigned int size);

#endif
```


然后在虚拟机下编写两个应用程序 iam.c和 whoami.c ，测试新增的两个调用，输出符合预期即可。

```
[/usr/root]# ./iam ZWJ
You are in iam
ZWJ
[/usr/root]# ./whoami
You are in whoami
ZWJ
[/usr/root]#
```