

哈尔滨工业大学计算机科学与技术学院
2016年秋季学期《操作系统》

Lab2：系统调用

姓名	学号	联系方式
匡盟盟	1143220116	kuangmeng@msn.com
樊晨霄	15S008199	18513534698

目 录

一、实验目的	1
二、实验内容	1
实验基本内容	1
iam()	1
whoami()	1
三、实验过程	1
四、回答问题	3
实验心得	4
附录	4
A. who.c	4
B. whoami.c	5
C. iam.c	5

一、实验目的

- 建立对系统调用接口的深入认识；
- 掌握系统调用的基本过程；
- 能完成系统调用的全面控制；
- 为后续实验做准备。

二、实验内容

实验基本内容

在Linux 0.11上添加两个系统调用，并编写两个简单的应用程序测试它们。

iam()

第一个系统调用是iam(), 其原型为:

```
int iam(const char * name);
```

完成的功能是将字符串参数name的内容拷贝到内核中保存下来。要求name的长度不能超过23个字符。返回值是拷贝的字符数。如果name的字符个数超过了23，则返回“-1”，并置errno为EINVAL。

在kernel/who.c中实现此系统调用。

whoami()

第二个系统调用是whoami(), 其原型为:

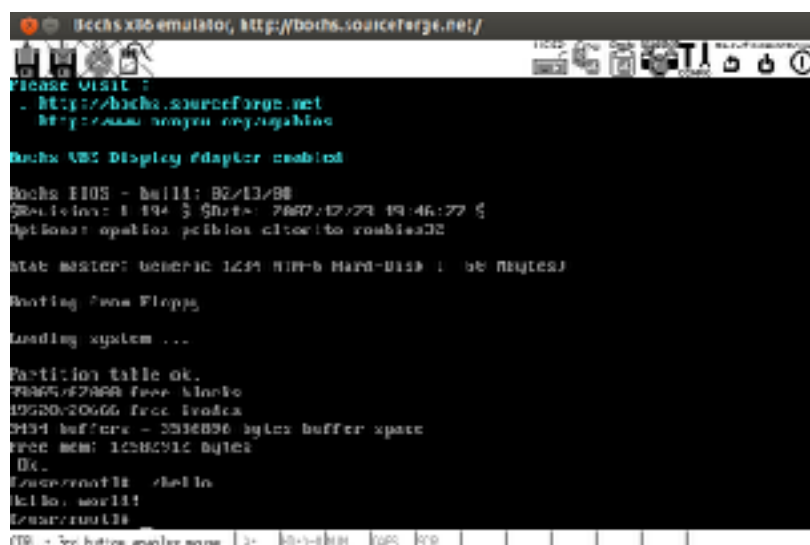
```
int whoami(char* name, unsigned int size);
```

它将内核中由iam()保存的名字拷贝到name指向的用户地址空间中，同时确保不会对name越界访存（name的大小由size说明）。返回值是拷贝的字符数。如果size小于需要的空间，则返回“-1”，并置errno为EINVAL。

也是在kernel/who.c中实现。

三、实验过程

1. 正常启动页面并调用“hello”（为了体验操作系统linux-0.11的正常系统调用过程）：



```
Bochs x86 emulator, http://bochs.sourceforge.net/
Please visit :
  - http://bochs.sourceforge.net
  - http://www.sony.com/ja/bochs

Bochs VBE Display Adapter enabled

Bochs BIOS - boot: 02/13/88
Partition: 1 194 5 50000 700000000 19 46 77 5
Options: options partition rombios32

ata0 master: generic IDE Hard-Disk 1 64 MB/128K
Booting from Floppy

Loading system ...

Partition table ok.
70000/70000 free blocks
10000/10000 free bytes
100 buffers - 100000 bytes buffer space
free mem: 1000000 bytes
Bochs root:~#
Bochs root:~# hello
hello, world!
Bochs root:~#
```

相关控制台操作如下：

```
kuangmeng@ubuntu: ~/oslab2
kuangmeng@ubuntu:~/oslab2/linux-0.11$ make all
as86 -O -a -o boot/bootsect.o boot/bootsect.s
ld86 -O -s -o boot/bootsect boot/bootsect.o
as86 -O -a -o boot/setup.o boot/setup.s
ld86 -O -s -o boot/setup boot/setup.o

kuangmeng@ubuntu:~/oslab2$ tools/build boot/bootsect boot/setup tools/kernel > image
Boot device is (1, 1)
Boot sector 512 bytes.
Setup is 312 bytes.
System is 121908 bytes.
in system.img
in tools/kernel -f
sync
kuangmeng@ubuntu:~/oslab2/linux-0.11$ cd ..
kuangmeng@ubuntu:~/oslab2$ ls
bochs  dbg-s  ydl-cmd.txt  hdc-0.11.img  mount-hdc  runydlb
dbg-aac  gdb  hdc  linux-0.11  run
kuangmeng@ubuntu:~/oslab2$ sudo ./mount-hdc
[sudo] password for kuangmeng:
kuangmeng@ubuntu:~/oslab2$ ./run
mount hdc first

=====
Bochs x86 Emulator 2.3.7
Build from CVS snapshot, on June 3, 2008
=====
000000000000[      ] reading configuration from ./bochs/bochsrc.bsrc
000000000000[      ] installing a module as the Bochs GUI
000000000000[      ] using log file ./bochsout.txt
```

- 修改 include/linux/sys.h 在 sys_call_table 数组最后加入 sys_iam 及 sys_whoami, 并仿照上面给出其他系统调用格式加上 extern int sys_iam() 及 extern int sys_whoami():

```
extern int sys_iam();
extern int sys_whoami();
```

- 修改 include/unistd.h, 加上如下截图所示的两行:

```
#define __NR_iam 72
#define __NR_whoami 73
```

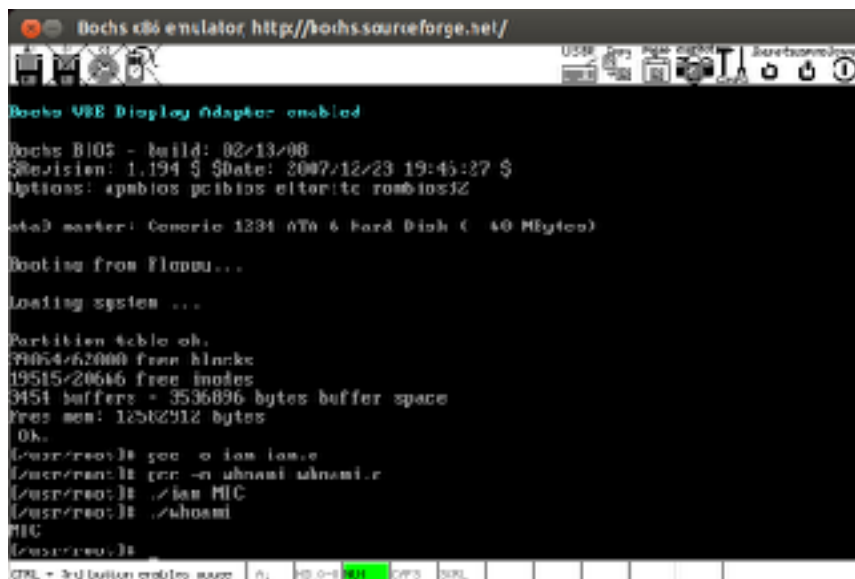
- 修改 kernel/system_call.s 中的一个变量值, 改成如下截图所示的一行:

```
nr_system_calls = 74
```

- 编写 whoami.c, iam.c 及 who.c 文件;
- 将 who.c 放入 linux-0.11/kernel/ 目录下, 修改该目录下 Makefile 文件:

```
who.s who.o: who.c ../include/linux/kernel.h ../include/unistd.h
OBJS = sched.o system_call.o traps.o asm.o fork.o \
panic.o printk.o vsprintf.o sys.o exit.o \
signal.o mktime.o who.o
```

7. 将linux-0.11挂载，并将上述whoami.c及iam.c文件放到hdc/usr/root目录下，使用“./run”命令运行bochs并编译（使用命令“gcc -o iam iam.c -Wall, gcc -o whoami whoami.c -Wall”）运行两个函数，得到如下截图所示的结果：



```
Bochs x86 emulator http://bochs.sourceforge.net/
Bochs VBE Display Adapter enabled
Bochs BIOS - build: 02/13/08
Revision: 1.194 $ $Date: 2007/12/23 19:45:27 $
Options: apmbios pcibios eltorito rombios12
ata0 master: Generic 1234 ATA 6 Hard Disk ( 40 Mbytes)
Booting from Floppy...
Loading system ...
Partition table ok.
70054/62000 free blocks
19515/20646 free inodes
9451 buffers = 3536896 bytes buffer space
Free mem: 12582912 bytes
Ok.
[usr/reco]: gcc -o iam iam.c
[usr/reco]: gcc -o whoami whoami.c
[usr/reco]: ./iam MFC
[usr/reco]: ./whoami
MFC
[usr/reco]:
```

至此本实验完毕！

四、回答问题

1. 从Linux 0.11现在的机制看，它的系统调用最多能传递几个参数？你能想出办法来扩大这个限制吗？

答：从linux-0.11/include/unistd.h中可以知道_syscall宏展开的系统调用最多3个参数，因为eax存放了系统调用功能号，只有ebx, ecx, edx三个寄存器保存用于系统调用过程中的需要传递的参数。

解决限制的方法：将需要传递的多个参数保存在有特定结构的区间中，并将该用户态地址空间的这个区间的首地址作为一个参数传递给系统调用。最后通过寄存器间接寻址方式便可以访问所有参数。当然，这么做的话，参数合法性验证尤其必要。

具体方法如下：

1. 在unistd.h中，将_syscalln对应的宏进行扩充。若n大于3，则可以采用esi,edi,ebp作为传递参数的寄存器，如此系统调用中可传递参数个数由3个扩展至7个。前提是，若n>3则要在宏展开的函数中先将esi,edi,ebp原来的值压栈，系统调用完成后再出栈恢复它们的原值。
2. 使用《Linux0.11内核完全注释》中提到的系统调用门的方法。在用户态堆栈和内核态堆栈自动复制传递参数。
3. 由于系统调用位于用户地址空间内，而系统调用处理函数位于内核地址空间内。所以若要传递3个以上的系统调用参数，可以先将所有参数连续地存放在一个预留出的用户地址内存区域内，顺序为第一个参数存在低地址处，后面的参数依次存放在越来越高的地址处。并将指向此区域的指针存入ebx.而需要的参数个数则存入ecx中。而在内核中的系统调用处理函数则通过收到的用户区域指针和参数个数，取出存放在用户地址空间中的所需参数。取出办法为：使用get_fs_long函数从用户空间中，每次取一个参数，直到取完ecx中所说明的参数个数为止，顺序为第一个参数最先取，然后取（指针+4）所指的第二个参数，依此类推。

此方法中，_syscalln真正传递给内核系统调用处理函数的参数只有系统调用功能号、指向参数存储区域的指针和参数的个数。

2. 用文字简要描述向Linux 0.11添加一个系统调用foo()的步骤。

答：首先，在内核中编写foo()对应的系统调用处理函数sys_foo()；

接着，修改 include/linux/sys.h 在sys_call_table数组最后加入sys_foo，并仿照上面给出其他系统调用格式加上extern returntype sys_foo()；

然后，添加系统调用foo()的功能号，修改include/unistd.h加上：#define __NR_foo num（num为接下来使用的系统调用号）；

再然后，修改 kernel/system_call.s 加上：nr_system_calls = num（num为在原值加1 即系统调用总数目加1），修改系统调用总数。同时相应的在include/linux/sys.h中声明新的系统调用处理函数sys_foo()以及添加系统调用处理函数指针数组表中sys_foo()的索引值，索引值的位置要和功能号对应；

接下来，在kernel中添加 foo.c，若需要支持内核态与用户态数据交互则包含include/asm/segment.h，其中有put_fs_XXX get_fs_XXX函数在foo.c实现系统调用sys_foo()

最后修改kernel的Makefile，将foo.c与内核其它代码编译链接到一起，由于系统调用用户界面需要在文件头部加上：

```
#define __LIBRARY__
#include <unistd.h>
_syscallN(.....)
```

宏展开系统调用，提供用户态的系统调用接口（参数数目确定具体宏）。

实验心得

通过这次实验我们基本了解了操作系统系统调用的基本流程，也知道了计算机操作系统是铺设在计算机硬件上的多层系统软件，不仅增强了系统的功能，而且还隐藏了对硬件操作的细节，由它实现了对计算机硬件操作的多层次的抽象。

附录

A. who.c

```
#define __LIBRARY__
#include <unistd.h>
#include <errno.h>
#include <asm/segment.h>
#include <string.h>
char username[64]={0};
int sys_iam(const char* name){
    int i=0;
    while(get_fs_byte(&name[i])!='\0')
        i++;
    if(i>23)
        return -EINVAL;
    i=0;
    while(1){
        username[i] = get_fs_byte(&name[i]);
        if(username[i] == '\0')
            break;
        i++;
    }
    return i;
}
int sys_whoami(char* name,unsigned int size){
    int i,len;
    len=strlen(username);
```

```
if (size<len)
return -1;
i=0;
while(i<len){
    put_fs_byte(username[i],&name[i]);
    i++;
}
return i;
}
```

B. whoami.c

```
#define __LIBRARY__
#include "unistd.h"
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
_syscall2(int,whoami,char*,name,unsigned int,size)
int main(int argc,char *argv[])
{
    char *username;
    username=(char *)malloc(sizeof(char)*128);
    whoami(username,128);
    printf("%s\n",username);
    free(username);
    return 0;
}
```

C. iam.c

```
#define __LIBRARY__
#include "unistd.h"
#include <errno.h>
#include <stdio.h>
_syscall1(int,iam,const char*,name)
int main(int argc,char* argv[]){
    iam(argv[1]);
    return 0;
}
```