

针对Linux x86_64内核，如何自己写系统调用

针对Linux x86_64内核，如何自己写系统调用

写一个helloworld系统调用

1. 依赖安装
2. 下载一个内核版本
3. 写helloworld系统调用
4. 写C语言程序查看成功插入helloworld系统调用模块

如何判断是否真正以模块化方式插入系统调用

参考资料

写一个helloworld系统调用

写Linux系统调用是Linux编程中经常的需求，本文将详细讲述这个过程，需要说明的是：**本文针对Linux Ubuntu 64位操作系统，32位不支持**

1. 依赖安装

注意：如果下面所示的依赖不能用apt-get的方式安装，请自行使用源码安装，当然可能你的系统中已经完成了这些依赖的安装。

```
1 sudo apt-get update #更新
2 sudo apt-get upgrade
3 sudo -s
4 apt-get install gcc
5 apt-get install python-pip python-dev libffi-dev libssl-dev libxml2-
  dev libxslt1-dev libjpeg8-dev zlib1g-dev
6 apt-get install libncursesw5-dev
```

2. 下载一个内核版本

在<https://www.kernel.org/> 下载一个需要的Linux内核版本（究竟选用那个Linux系统版本，取决于你自己的需求），并解压该文件。

3. 写helloworld系统调用

```
1 cd *.x    #进入到Linux内核解压后的根目录下
2 mkdir helloworld
3 cd helloworld
4 vim helloworld.c
```

在helloworld.c中写入下面的内容：

```
1 #include <linux/kernel.h>
2 asmlinkage long sys_helloworld(void){
3     printk("Hello World~\n");
4     return 0;
5 }
```

接着执行

```
1 vim Makefile    #和helloworld.c在同一个目录下
```

在Makefile文件中写入下面的内容：

```
1 obj-y := helloworld.o
```

接着回到Linux内核源码的根目录中，修改Linux内核自带的Makefile文件，在其中写入下面内容：

```
1 core-y += kernel/.../ helloworld/    #只有helloworld是需要我们添加的，其它的在Linux本身的Makefile文件中就全部都有
```

接着执行

```
1 cd include/linux
2 vim syscalls.h
```

在打开的文件的最后面写入下面内容

```
1  asm linkage long sys_helloworld(void);    #注意，当系统调用没有参数时，请
    写明void，否则在编译时会出错
```

接着回到Linux内核根目录，并执行下面指令

```
1  cd arch/x86/entry/syscalls
2  vim syscall_64.tbl
```

在打开的文件中写入下面内容：

```
1  333 64 helloworld sys_helloworld    #333 is index of our system call
```

回到Linux内核的根目录

```
1  make menuconfig    #保存即可
2  make oldconfig
3  make -j 4    #编译内核
4  make modules_install install #安装新的内核
5  reboot    #重启计算机
6  uname -a    #查看是否进入到新内核中
```

4. 写C语言程序查看成功插入helloworld系统调用模块

```
1  //将该文件命名为test.c，并写入下面的代码，测试helloworld系统调用是否能用
2  #include <stdio.h>
3  #include <linux/kernel.h>
4  #include <sys/syscall.h>
5  #include <unistd.h>
6
7  int main(){
8      long int s = syscall(333);    //333 is index of helloworld system
    call
```

```

9     printf("System call : sys_helloworld : return %d\n" , s);
10    return 0;
11 }

```

写完test.c之后编译并运行，查看结果，如果return后面输出的值为0，说明上述系统调用完全正确。

```

1 gcc test.c
2 ./a.out // 输出return 0，值为0说明所有的系统调用都是成功的
3 dmesg #查看kernel日志，最后一行看到存在helloworld，说明成功写出
helloworld系统调用

```

如何判断是否真正以模块化方式插入系统调用

用于测试，我们写出下面的测试函数，

```

1 //文件命名为test.c
2 #include <stdio.h>
3 #include <linux/kernel.h>
4 #include <sys/syscall.h>
5 #include <unistd.h>
6
7 long helloworld(){ //在上文已经插入的helloworld系统调用模块，该系统调
用的作用是申请一块连续的地址空间，并返回其首地址，malloc本身并不是系统调用
8     int ret;
9     __asm__("movl $333,%eax"); //将helloworld系统调用对应的系统调用号推到
寄存器中
10    //__asm__("movl $参数值1,%ebx");
11    //__asm__("movl $参数值2,%ecx");
12    //__asm__("movl $参数值,%edx"); //,%ebx, %ecx, %edx, %esi, %edi
和%ebp
13    __asm__("int $0x80"); //陷入到系统调用中，完成用户态到内核态的切换，
Linux选择128号（0x80）作为陷入指令入口
14    __asm__("movl %eax, -4(%ebp)");
15    return ret;
16 }
17
18 int main (){

```

```

19 //long int s = syscall(333); //如果采用这条命令，则不需要使用上面的
    helloworld封装例程，直接使用syscall(系统调用号，参数值1，参数值2...)
20 long int s = helloworld(); //调用上述封装例程，在这个封装例程中，能
    够写入指定个数的参数，一般是6个参数
21 printf("return %1d\n",s);
22 return 0;
23 }

```

上述程序的大致流程为：应用程序调用helloworld，接着调用封装例程（也就是写着int \$0x80的函数），通过陷入指令进入到内核中，或者直接使用syscall()函数，直接进入系统调用。

接着对上述代码执行下面的命令：

```

1 sudo -s #进入到root权限下执行下面的命令
2 dmesg #查看kernel日志，能够看到执行的系统调用
3 gcc test.c -o test
4 ./test
5 dmesg #在kernel日志中应该能够看到有helloworld字样输出，但是没有
    helloworld1字样输出，system(333)的返回值为0，由这三个输出能够得出：按照上
    述过程，我们成功在内核中插入了helloworld系统调用模块

```

参考资料

1. 相对下面几个参考资料，该资料实现系统调用的方式比较正确
<http://www.franksthinktank.com/howto/addsyscall/>
2. <https://linux.cn/article-9628-1.html>
3. http://www.cnblogs.com/hazir/p/three_methods_of_syscall.html
4. <http://mooc.study.163.com/course/1000072000#/info>
5. <https://www.linux.it/~rubini/docs/ksys/>
6. <https://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html>
7. <https://www.slideshare.net/mehershree/how-to-add-system-call-in-ubuntu-os>
8. <http://gityuan.com/2016/05/21/syscall/>
9. <https://blog.csdn.net/sdulibh/article/details/51889279/>
10. 正确的资料 https://blog.csdn.net/sinat_28750977/article/details/50837996

11. 较为正确的资料 <https://blog.csdn.net/JackLiu16/article/details/79477967> 这个资料告诉我 想要写更加复杂的系统调用逻辑，必须要学会Linux系统编程
12. malloc系统调用实现
<https://blog.csdn.net/zhangyang249/article/details/78582809>
13. 不错的中文资料 https://blog.csdn.net/sinat_28750977/article/details/50837996
- 14.