灵跃桌面云

为 Linux-3.10.1 内核添加系统调用

成/都/虫/洞/奇/迹/科/技/有/限/公/司

版权声明

商标声明

 和其他成都虫洞奇迹科技有限公司商标均为成都虫洞奇迹科技有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受成都虫洞奇迹科技有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，成都虫洞奇迹科技有限公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

成都虫洞奇迹科技有限公司

电话：400-090-2980

邮箱：contact@lingyuecloud.com

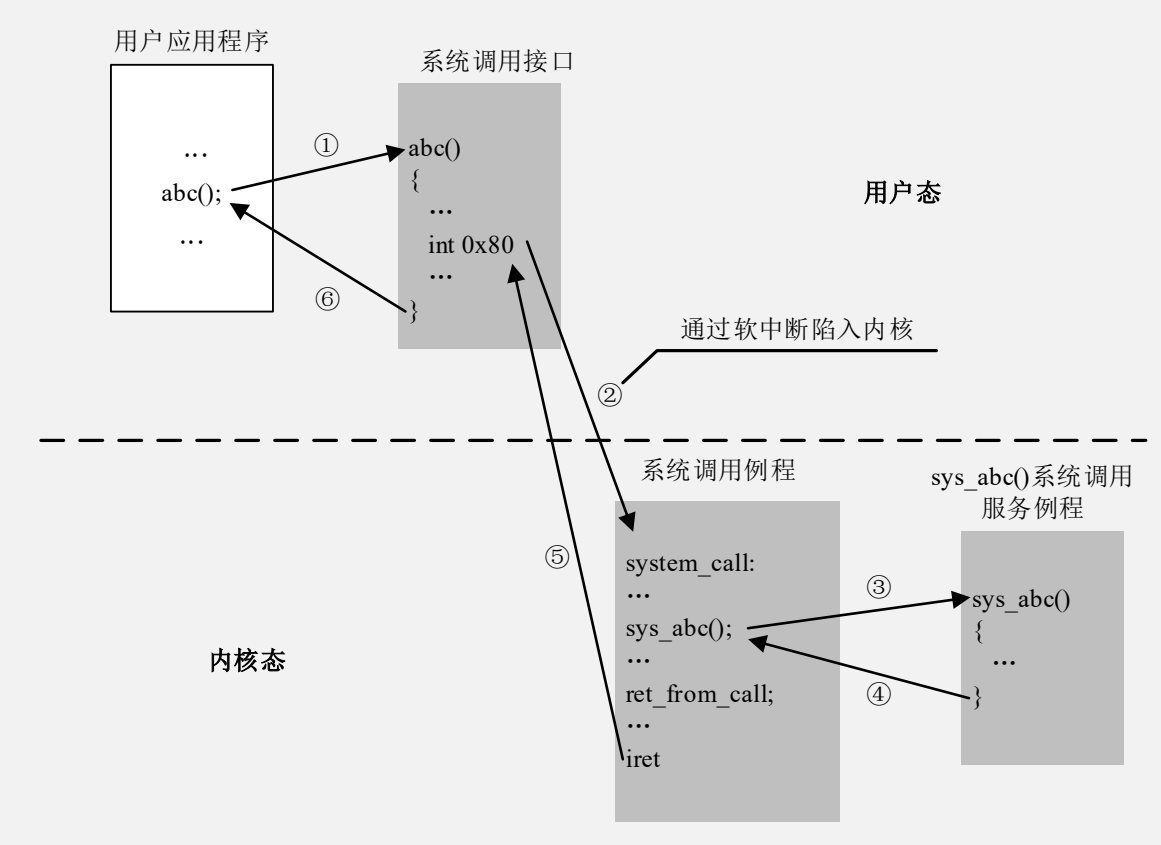
网址：www.lingyuecloud.com

1. 写在前面

系统调用是操作系统提供给用户程序调用的一组“特殊”接口。通过这组“特殊”接口，用户程序可以获得操作系统内核提供的服务，如文件系统相关系统调用提供的打开文件、关闭文件或读写文件服务，时钟相关的系统调用提供的获取系统时间、设置系统时间服务等。

从逻辑上来说，系统调用可被看成是一个内核与用户空间程序交互的接口——好比一个中间人，把用户进程的请求传达给内核，待内核把请求处理完毕后再将处理结果送回给用户进程。

一个系统调用的典型调用过程如下图所示。



系统调用的作用：

系统服务之所以需要通过系统调用提供给用户空间，其根本原因是为了对系统进行“保护”。我们知道操作系统的运行空间分为内核空间与用户空间，它们各自运行在不同的级别中，逻辑上相互隔离，故用户进程在通常情况下不允许访问内核数据、不允许使用内核函数，它们只能在用户空间操作用户数据，调用用户空间的函数。但是很多情况下，用户进程需要

获得系统服务（调用系统程序），这时就必须利用系统提供给用户的“特殊”接口——系统调用，其特殊性在于规定了用户进程进入内核的具体位置，即用户访问内核的路径是事先规定好的，只能从规定位置进入内核，而不准许肆意跳入内核。有了上述陷入内核的统一访问路径限制，才能有效保证系统内核的安全。我们可以形象地描述上述机制：作为一个游客，你可以买票要求进入野生动物园，但你必须老老实实的坐在观光车上，按照规定的路线观光游览。当然，不准下车，因为那样太危险，不是让你丢掉小命，就是让你吓坏了野生动物。

在前述文章“Linux-3.10.1内核的编译和安装”（详见http://www.lingyuecloud.com/Index/details/id/58.html）中，我们知道利用Linux内核的开源特性，可以对Linux系统进行定制化开发、编译和安装。例如，通过修改Linux内核源码，我们定制化开发了一个新功能，那么如何在应用程序中使用这个新功能呢？为此，我们还需要为“应用程序”与“内核中新增的新功能”建立桥梁——即一个能够连接两者的系统调用。

假设我们已经获得了Linux-3.10.1的源码包linux-3.10.1.tar.bz2，解压该源码包后得到linux-3.10.1文件夹，并切换到该文件夹下（如何获取包linux-3.10.1.tar.bz2？如何解压？请详见http://www.lingyuecloud.com/Index/details/id/58.html——“Linux-3.10.1内核的编译和安装”）。接下来，灵跃桌面云将以Linux-3.10.1内核为例，详细描述为内核添加一个新的系统调用的过程，下述所有操作都在linux-3.10.1目录下进行。

2. 环境说明

操作系统	Ubuntu 12.04　64 位
内核版本	linux-3.10.1

注：在下述描述中，涉及在服务器 Ubuntu 12.04 操作系统环境上的所有操作，均以 root 身份登录并执行。

3. 为 Linux-3.10.1 内核添加一个系统调用

3.1 系统调用号

Linux系统调用号的作用是在系统调用过程中，将其数值作为下标的在系统调用表中进

行索引，从而得到处理该系统调用的函数的地址。

每个系统调用都有一个唯一的系统调用号，应用程序可以通过系统调用号调用指定的系统调用。

3.2 系统调用表

与Windows系统中的SSDT(System Services Descriptor Table)的作用一样，Linux系统调用表保留着处理各个系统调用的函数的入口地址；其实际上是一个二维的指针数组[X][Y]，X代表系统调用号，Y代表系统调用函数的入口地址。

3.3 添加系统调用 lingyuecloudsyscall

1) 添加自定义的系统调用源代码

在“linux-3.10.1/kernel”目录下的sys.c文件中添加自定义的系统调用lingyuecloudsyscall的实现函数。

打开sys.c文件，添加以下示例代码：

#vim kernel/sys.c

```
asmlinkage long sys_lingyuecloudsyscall (int number){
        printk("hello lingyuecloud!Call number is %d\n",number);
        return number;
}
```

2) 修改系统调用表

系统调用表文件在“linux-3.10.1/arch/x86/syscalls”目录下的syscall_64.tbl文件中。

打开syscall_64.tbl，添加新的系统调用指针，如下所示：

#vim arch/x86/syscalls/syscall_64.tbl

```
314        64        lingyuecloudsyscall        sys_
lingyuecloudsyscall;
```

如下图所示：

```
310     64      process_vm_readv        sys_process_vm_readv
311     64      process_vm_writev       sys_process_vm_writev
312     common  kcmp                    sys_kcmp
313     common  finit_module            sys_finit_module
314     64      lingyuecloudsyscall     sys_lingyuecloudsyscall
```

其中，314为lingyuecloudsyscall的系统调用号，应用程序可通过此调用号调用lingyuecloudsyscall系统调用，也可以使用其它的系统调用号，但注意不能与已有的系统调用号重复。64表示适配于64位系统内核环境，相关描述详情可查阅https://en.wikipedia.org/wiki/X32_ABI。

3) 添加系统调用lingyuecloudsyscall的函数声明

在“linux-3.10.1/include/linux/”目录下的syscalls.h文件中添加函数声明。打开syscalls.h，在倒数第二行添加下面内容：

#vim include/linux/syscalls.h

```
asmlinkage long sys_lingyuecloudsyscall(int num);
```

如下图所示：

```
asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
                         unsigned long idx1, unsigned long idx2);
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_lingyuecloudsyscall(int num);
#endif
```

4) 重新编译

5) 内核

编译过程详见http://www.lingyuecloud.com/Index/details/id/58.html——“Linux-3.10.1内核的编译和安装”。

6) 测试系统调用

编写测试程序lingyuecloud_test.c：

```
#include<stdio.h>
#include<linux/unistd.h>
#include<sys/syscall.h>
int main()
{
        long a;
        a=syscall(314,100);     //调用第314号系统调用，即sys_lingyuecloudcall();
        printf("The number is%d\n",a);
        return 0;
}
```

编译测试程序：

#gcc -o lingyuecloud lingyuecloud_test.c

测试结果如下图所示：

```
[root@controller ~]# ./lingyuecloud
The number is 100
```

如果能够看到上述结果，表示你的第一个自定义系统调用已经添加成功。