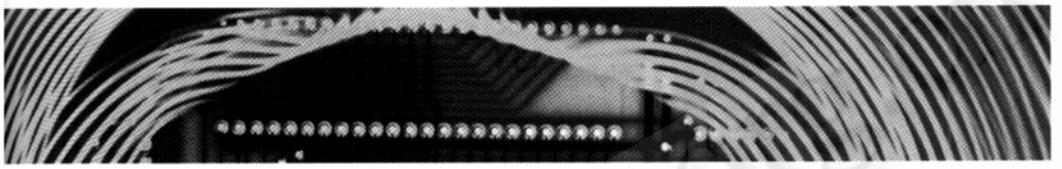
Linux 系统增加系统调用的方法

厉海燕 李新明 (北京装备指挥技术学院网络技术室 101416)



1 Linux 简介

Linux是用于多种计算机平台的操作系统，它是由全世界各地数以百计的程序员设计和实现的，其目的是建立不受任何商品化软件版权制约的、全世界都能自由使用的UNIX兼容产品。

Linux是目前唯一可免费获得的、为IBM PC及其兼容机硬件平台上的多个用户提供多任务、多进程功能的操作系统，这是人们要使用它的原因。Linux源代码开放，许多应用程序也可以从网上免费获得，但到底谁拥有Linux的所有权呢？首先Linux不是公共领域中的软件；Linux各组成部分的版权由多人拥有。Linus TorvaLds拥有Linux的基本内核的版权。Red Hat公司拥有Red Hat发行版的版权。PauL VoLkerding拥有SLackware96发行版的版权。Linux的许多实用程序受GNU通用公用许可证（GPL）的保护。事实上，Linux和大多数对Linux作出了贡献的人们把他们的工作成果置于GNU GPL的保护之下。这份证书使用于大多数由GNU项目和Free Software Foundation（自由软件基金会）生产的软件，它允许程序员创建为每个人使用的软件。GNU的基本前提是：软件应当可供每个人使用，如果某人想按他自己的需要修改软件的话，这也应当是允许的。唯一的告诫是：修改的代码不能被限制——其他人必须有权得到新代码。

2 增加系统调用

刚刚参加了一个课题，我的任务是在内核实现Linux的异步I/O功能，也就是为系统增加几个提供异步I/O功能的系统调用。

2.1 相关源代码分析

系统初始化后运行的第一个内核程序通过调用函数void __init trap_init(void)把各自陷和中断服务程序的入口地址设置到idt表中，其中系统调用总控程序system_call就是中断服务程序之一。trap_init函数通过调用一个宏set_system_gate(SYSCALL_VECTOR,&system_call)把系统调用总控程序的入口挂在中断0x80上，其中SYSCALL_VECTOR是一个常量0x80，system_call即为中断总控程序的入口地址，中断总控程序用汇编语言定义在arch/i386/kernel/entry.S中。中断总控程序主要负责保存处理机执行系统调用前的状态，检查当前调用是否合法，并根据系统调用向量，使处理机跳转到保存在sys_call_table中的相应系统服务例程的入口；从系统服务例程返回后恢复处理机状态退回用户程序。系统调用向量定义在/usr/src/linux/include/asm-i386/kernel/unistd.h中，sys_call_table定义在/usr/src/linux/arch/i386/kernel/entry.S中，同时在unistd.h中也定义了系统调用的用户编程接口。

由此可见，Linux的系统调用也像DOS系统的int 21h中断一样，它把0x80中断作为总的入口，然后转到保存在sys_call_table中的各种中断服务例程的入口地址，形成各种不同的中断服务。由以上源代码分析可知，要增加一个系统调用除了系统服务例程外还必须在sys_call_table中增加一项，并在其中保存好自己的系统服务例程的入口地址，然后重新编译内核。

2.2 对内核源码的修改

为 Linux 内核增加系统调用要修改的源代码文件有 incLude/asm-i386/unistd.h和arch/i386/kerneL/entry.S两个，其步骤如下：

(1) 增加系统服务例程。由于异步 I/O 和文件系统相关，因此我在 fs/read_write.c 中有八个服务例程。因本文主要介绍增加系统调用的方法，故省略服务例程的代码，并以增加其中两个系统调用为例。

```
asmlinkage int sys_aio_read(struct aiocb *uaiocb)
{
        ......
 }
asmlinkage int sys_aio_write(struct aiocb *uaiocb)
{
        ......
}
```

(2) 把这两个系统调用的入口地址加到 sys_caLL_tabLe 表中。

arch/i386/kerneL/entry.S 中的最后几行源代码修改前为：

```
......
.Long SYMBOL_NAME(sys_sendfiLe)
.Long SYMBOL_NAME(sys_ni_syscaLL) /* streams1 */
.Long SYMBOL_NAME(sys_ni_syscaLL) /* streams2 */
.Long SYMBOL_NAME(sys_vfork) /* 190 */
rept NR_syscaLLs-190
.Long SYMBOL_NAME(sys_ni_syscaLL)
.endr
```

修改后为：

```
......
.Long SYMBOL_NAME(sys_sendfiLe)
.Long SYMBOL_NAME(sys_ni_syscaLL) /* streams1 */
.Long SYMBOL_NAME(sys_ni_syscaLL) /* streams2 */
.Long SYMBOL_NAME(sys_vfork) /* 190 */
/* add by AIO */
.Long SYMBOL_NAME(sys_aio_read)
.Long SYMBOL_NAME(sys_aio_write)
.rept NR_syscaLLs-192
.endr
```

(3) 把增加的 sys_caLL_tabLe 表项所对应的向量，在 incLude/asm-386/unistd.h 中进行必要申明，以供用户进程和其他系统进程查询或调用。

增加后的部分 include/asm-i386/unistd.h 文件如下：

```
......
#define __NR_sendfiLe 187
#define __NR_getpmsg 188
#define __NR_putpmsg 189
#define __NR_vfork 190
/* add by AIO */
#define __NR_ aio_read 191
#define __NR_ aio_write 192
......
```

(4) 把新增加的系统调用生成一个用户库，以供用户使用。把新增加的所有系统调用函数编辑成一个*.c文件，再编译成*.o 文件，然后用 ar 命令生成一个用户库中。如将上述新增的两个系统调用编辑成 aio_syscall.c 文件：

```
#incLude</usr/src/Linux/incLude/asm-i386/unistd.h>
extern int errno;
_syscall1(int, aio_read, struct aiocb *,uaiocb)
_syscall1(int, aio_write, struct aiocb *,uaiocb)
```

然后编译成 aio_syscall.o 文件:

```
#cc -c aio_syscall.c
```

最后将 aio_syscall.o 生成一个用户库 libaio.a，并将该库放在 /usr/lib 目录下：

```
#ar -cr libaio.a aio_syscall.o
#cp libaio.a /usr/lib
```

这样，在重新编译内核后，用户便可使用系统调用了。例如，用户在程序 test.c 中使用了新增加的系统调用，那么在编译时必须指定要链接的用户库 libaio.a:

```
#gcc -o test test.c -laio
```

3 结束语

Linux 内核是庞大复杂的，与系统调用相关的代码只是内核中极其微小的一部分。为Linux 系统增加功能不一定要增加系统调用，还可以模块形式加载，本文只是给大家提供了一种增加系统功能的方法。■

参考文献

1 怀石工作室。LINUX 上的 C 编程。北京中国电力出版社，2000。

2 陈莉君。LINUX 操作系统分析。北京人民邮电出版社，2000。

3 http://www.gnu.org.。