操作系统课程设计 - 指导手册

提示：此文档只包含一些基本的实验提示，很多地方仍然需要自己去思考去实现。有能力的同学可以忽略此文档。

Fedora 7虚拟机镜像的初始登录用户名：seu，密码：seu，root用户密码：seu。

**实验0：Linux内核代码分析**

1. 解压内核

桌面上的linux-2.6.21.tar.gz是linux-2.6.21的内核代码压缩包，解压：

$ cd Desktop

$ tar zxvf linux-2.6.21.tar.gz

$ cd linux-2.6.21

1. 生成内核配置文件

将当前正在运行的内核对应的配置文件作为模板来生成.config文件，即将/boot目录下的已有的config文件复制到linux-2.6.21目录下

$ make mrproper

$ cp /boot/config-`uname –r` ./config

第一个命令make mrproper用来保证内核树是干净的，如果内核第一次编译则可以省略。其中的uname –r命令可查看当前环境下的内核版本号。

更新config文件：

$ make oldconfig

部分新配置项会提示用户选择，都选N或者缺省即可，完成后即可生成.config文件。

1. 编译安装内核

在编译内核前，可以定义自己的内核版本号，在内核代码的根目录下有Makefile文件，例如将第4行改为：

EXTRAVERSION = -seu

这样新内核版本号就是2.6.21-seu

$ make all

$ make modules\_install

$ make install

Make all的执行过程可能比较长。

如果三个命令均成功执行，可以观察引导程序grub的配置文件/boot/grub/menu.lst的内容，在hiddenmenu之后可以看到刚刚编译安装的内核版本，将hiddenmenu那一行注释或删除，方便直接操作菜单：

#hiddenmenu或者~~hiddenmenu~~

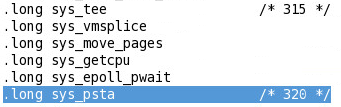
然后重启系统：

$ reboot

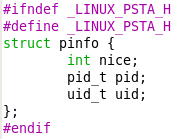
重启后可以看到grub菜单已经包含了新编译的内核。如果新内核启动失败，一般是由于配置或者内核代码修改的有问题，选择原先的内核启动，再进行修改、编译。

**实验0’：新增系统调用**

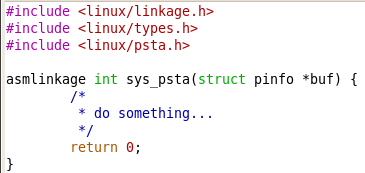
1. 在文件arch/i386/kernel/syscall\_table.S的尾部加上要新增的系统调用函数的名称，如下图中添加了psta系统调用，注释中的320表示它的系统调用号



1. 在include/linux目录下添加头文件psta.h，如：

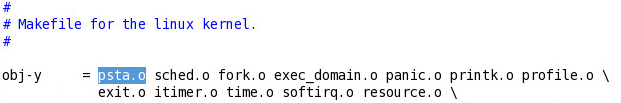


在kernel目录下新建文件psta.c，在该文件中实现sys\_psta函数：



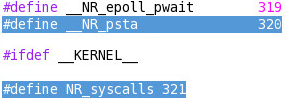
宏asmlinkage定义在linux/linkage.h中，表示函数的参数通过栈传递，而不是寄存器，所有的系统调用都遵循这种参数传递方式。

1. 修改文件kernel/Makefile，使得psta.c在编译时可见：



另外，第2步中psta的实现不一定要在一个新文件中，例如文件kernel/sys.c也许是添加这个系统调用的合适位置，这样的话第3步就不需要了。

1. 在include/asm-i386/unistd.h里加上系统调用号的宏定义：



其中NR\_syscalls表示的值应该是最大的系统调用号加一。

1. 修改include/linux/syscalls.h，加上函数sys\_psta的声明。在该文件开始添加：



在最后添加：



1. 重新编译内核

在清理上次编译生成的中间文件之前，最好将配置好的.config文件备份至别的目录下以防删除，否则必须再执行实验1中的第2步配置.config文件。

$ make mrproper

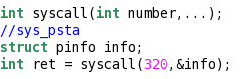
在实验1中就已提到该命令，用于清理编译内核代码的中间文件，它也会删除.config文件。在清理执行完成后，将备份的.config文件复制回来，然后执行实验1中的第3步就可以编译安装新内核了。安装好的内核必须reboot后才能生效。

以上6步成功执行后系统调用psta就已被添加到系统中，下面来进行测试。

使用glic库提供的函数syscall来间接地使用新系统调用，调用syscall必须添加以下两个头文件：



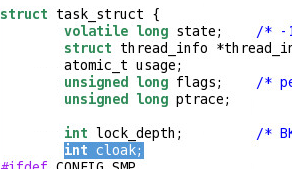
函数的原型如下图，syscall的第一个参数是系统调用号，后面的参数是该系统调用的各个参数，返回值就是系统调用的返回值。例如函数psta的系统调用号是320，且接受一个类型为struct pinfo \*的参数。



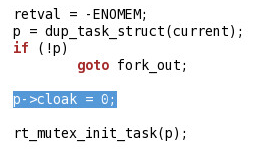
具体的测试代码请大家根据自己实现的psta来编写，只要体现出调用了psta函数即可。

**实验1：Linux进程管理及其扩展**

1. 实现系统调用hide
   1. 在include/linux/sched.h中修改task\_struct，添加一个成员cloak，用来记录进程隐藏与否。



* 1. 在进程创建时，将task\_struct的成员cloak初始化为未隐藏。fork系统调用的实现代码在kernel/fork.c中，具体实现的主要函数为do\_fork，do\_fork中调用copy\_process函数创建子进程，建议将初始化cloak的代码添加在copy\_process函数中：



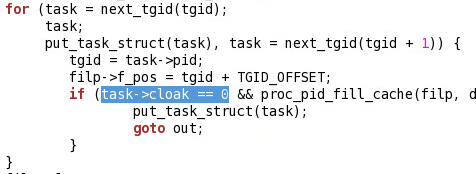
* 1. 添加hide系统调用

具体的做法参见实验2，sys\_hide的实现可以和psta一样放在新的文件中，不过也许放在文件fs/proc/base.c中更加适合。

提示：通过pid获取进程task\_struct的内核函数为find\_task\_by\_pid。在隐藏后最好调用函数proc\_flush\_task来清空VFS层的缓冲，解除已有的dentry项。

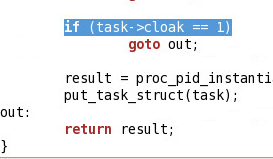
* 1. 修改proc\_pid\_readdir函数（在fs/proc/base.c文件中）

其中使用for循环遍历进程，在遍历过程中添加判断，过滤掉被隐藏的进程。以下仅作参考：



* 1. 修改proc\_pid\_lookup函数

在进程查找完成前过滤掉被隐藏的进程。仅做参考：



1. 考虑权限问题，只有根用户才能隐藏进程。

提示：在hide的实现中添加用户的判断即可。current->uid是当前用户的uid。那么根用户的uid是多少？

1. 实现系统调用hide\_user\_processes

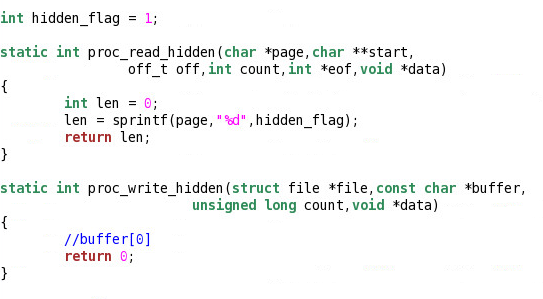
提示：可遍历系统中所有的进程，隐藏满足要求的进程。内核中有宏定义for\_each\_process可用于遍历所有进程，其实就是一个for循环。每个进程的task\_struct中有成员变量uid和comm，uid为该进程的用户id，comm为进程名。此外，在这个系统调用中也要考虑前面的权限问题，即只有根用户才能隐藏进程。

1. hidden文件

提示：proc文件系统在初始化函数proc\_root\_init中会调用proc\_misc\_init函数，此函数用于创建/proc根目录下的文件，那么将创建hidden文件的代码插入到此函数中就可以在proc初始化时得到执行。

（另外也可以参考《linux 操作系统实验教程》课本9.2.2节，使用模块的方式来实现hidden文件。）

* 1. 在/fs/proc/proc\_misc.c中添加回调函数



首先在全局作用域定义变量hidden\_flag。

然后定义hidden文件的读写回调函数，以上给出了读回调函数的参考代码，写回调函数中请大家自行编写。

提示：写回调函数中，形参buffer中存有用户输入的值，例如输入的是0，那么buffer[0] == ‘0’。

* 1. 在/fs/proc/proc\_misc.c中proc\_misc\_init函数的最后添加创建hidden文件的代码，并指定其回调函数。以下代码仅供参考，建议添加代码以判断创建文件是否成功。



代码的后两行指定了hidden文件的读写回调函数。

注意在修改后，再次编译安装内核并reboot后才能在proc文件系统中创建hidden文件。

* 1. hidden文件创建成功后，需要实现通过全局变量hidden\_flag来约束隐藏进程的函数。

根据题目：当hidden\_flag为1时，此前通过hide调用要求隐藏的进程才被隐藏。

这就意味着修改hidden\_flag不可以修改cloak，即之前使用hide和hide\_user\_process隐藏的进程需要在hidden\_flag置为1之后保持隐藏。

所以，在proc\_pid\_readdir函数和proc\_pid\_lookup函数中判断cloak值之前添加对hidden\_flag的判断就可以实现约束。

提示：使用其他文件中的全局变量需要添加extern声明。

1. hidden\_process文件

方法和创建hidden文件一样，hidden\_process文件只需要设置读的回调函数即可。

输出所有被隐藏进程的pid只要输出所有cloak为1的进程pid即可。用遍历所有进程的方法，判断cloak的值来决定是否在回调函数中输出。

提示：当向hidden文件写0、即hidden\_flag==0时，hidden\_process文件应为空，具体的设计实现请大家认真考虑。

提示1：并不是每一次修改内核都要使用make mrproper 清理所有的中间文件。

提示2：Fedora 7中自带的gcc/g++编译器版本较低，不支持一些较新的C/C++标准。比如，变量的声明必须在作用域的开头，也就是必须先声明所有需要声明的变量，再使用这些变量。

参考文献：

《Linux 操作系统实验教程》电子工业出版社，罗宇等编著