## 1. 实验目的：

掌握从中获取linux系统信息的方法，了解可以利用proc文件系统进行用户态与内核态的交互。尝试修改内核，添加一项功能，并编译、使用自己修改的内核，验证添加的功能。

## 2. 实验目标：

通过proc文件系统获取linux系统信息，编写程序获取cpu使用率，内存使用情况，网络负载情况，进程信息等并显示；修改sys内文件，调整内核参数；修改内核中的proc源码，通过系统调用实现进程的隐藏。

## 3. 实验原理：

### 3.1 /proc 信息解读

proc文件系统主要包含三大类内容，进程相关部分，系统信息部分，以及系统子系统部分。

进程文件都是以数字为名的子目录，这个数字就是相关进程的进程ID，主要包含cmdline（执行进程的命令行参数），stat（进程状态）等

内核系统信息处于proc根目录，大部分是常规只读文件可以直接查看，主要包含cpuinfo，meminfo等。

各子系统也是以非数字命名的目录下，甚至部分是可写进行调整系统内核参数，用户态和内核态进行交互，主要包含net(只读)，sys(可写)等。

### 3.2 /proc 修改与应用

显示PID的proc\_root\_readdir函数中先生成非进程目录，再通过proc\_pid\_readdir函数生成进程pid目录，这也是本次实验中需要主要关注的。对进程隐藏，实际上就是该进程不在proc文件系统体现，直接查看proc文件目录无法查找到该进程，这样ps命令就无法显示出该进程。在proc\_pid\_readdir函数中的for循环体就是往/proc添加<pid>目录，我们在填充目录之前判断该task\_struct中的hide变量，只有hide变量为0时才把该进程填充到/proc中，这样就可以达到隐藏进程的目的。其中hide变量可以通过系统调用进行修改。

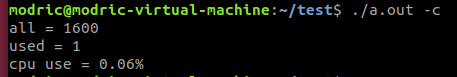
## 4. 实验方案：

实验安装的Linux发行版为Ubuntu 17.10，实验前手动升级内核到目前stable分支的最新版4.14.6，实验时使用的内核源代码也是这个版本。

### 4.1 /proc

#### 步骤1：编写程序获取cpu使用率

查看/proc/stat文件，只用使用第一行数据即可。其中第一行七个数据分别为：Use：从系统启动开始累计到当前时刻，处于用户态的运行时间，不包含 nice值为负进程。Nice：从系统启动开始累计到当前时刻，nice值为负的进程所占用的CPU时间。System：从系统启动开始累计到当前时刻，处于核心态的运行时间。Idle：从系统启动开始累计到当前时刻，除IO等待时间以外的其它等待时间。Iowait：从系统启动开始累计到当前时刻，IO等待时间(since 2.5.41)。Irq：从系统启动开始累计到当前时刻，硬中断时间(since 2.6.0-test4)。Softirq：从系统启动开始累计到当前时刻，软中断时间(since 2.6.0-test4)。

七个值的合为cpu总时间，idle为空闲时间。注意这七个值都为时刻，要想获得时间段，需要进行两次采样，做时间差，就可以获得当前时间的cpu时间。计算cpu使用率的公式为：cpu使用率=(cpu总时间-空闲时间)/cpu总时间。

#### 步骤2：编写程序获取内存使用率

查看/proc/meminfo，使用前两行的数据，前两行分别为总内存量和剩余内存量。计算内存使用率的公式为：内存使用率=（总内存量-剩余内存量）/总内存量。

#### 步骤3：编写程序获取网络负载

查看/proc/net/dev文件，其中前两行为表头，从第三行开始为相关的网络流量数据。第一列接口信息，接口一般有这几种：eth0: ethernet的简写，一般用于以太网接口；wifi0:wifi是无线局域网，因此wifi0一般指无线网络接口；ath0: Atheros的简写，一般指Atheros芯片所包含的无线网络接口；lo: local的简写，一般指本地环回接口。注意我们要忽略lo接口的网络数据。

根据表头我们可以知道二到第七列为接受数据，之后为发送数据。一般我们只需要前四个数据：bytes收发的字节数；packets收发的正确的包量；errs收发错误的包量；drop收发丢弃的包量。计算网络负载的公式：网络负载=（接收的包数+发送的包数）/2。

#### 步骤4：编写程序获取当前进程的信息

查看/proc/self/status，前两行的数据为进程名称和进程状态，第五、六行为该进程pid及父进程pid。

### 4.2修改内核信息

修改proc/sys中的文件，达到调整内核参数的效果。本次实验通过一个简单的例子：修改本机是否可被ping，来展示利用proc文件系统进行用户态与内核态的交互。

步骤1：ping locahost 成功ping

步骤2：su 以管理员权限写入

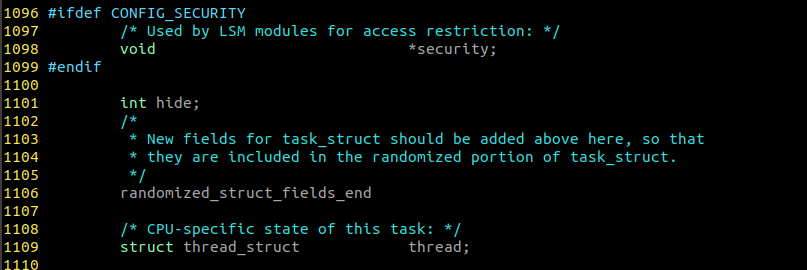
步骤3：echo 1 >/proc/sys/net/ipv4/icmp\_echo\_ignore\_all 修改参数为1

步骤4：ping localhost 无法ping

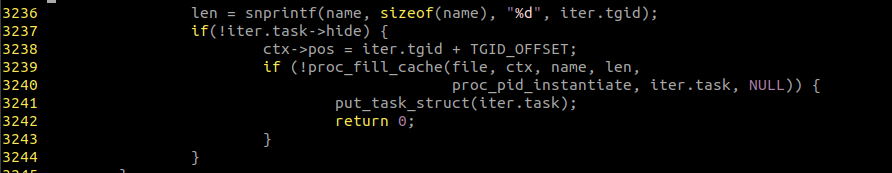


### 4.3隐藏进程

步骤1：修改内核，为task\_struct结构体（include/linux/sched.h，根目录为内核根目录，下同）添加int变量hide，要注意添加在结构体有效代码的倒数第三行（参见源码中的注释），不要影响前面变量的偏移量。



步骤2：修改proc\_pid\_readdir函数（fs/proc/base.c），在填充proc进程pid目录之前先判断该进程是否隐藏，其实就是添加一个if语句。

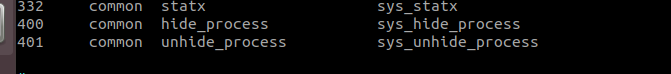


修改task\_struct->hide变量需要通过系统调用来实现。添加系统调用有三个步骤：

在arch/x86/entry/syscalls/syscall\_64.tbl里加入:

400 common hide\_process sys\_hide\_process

401 common unhide\_process sys\_unhide\_process



400/401为调用号，后面的两列为kernel/sys.c中的系统调用名和include/linux/syscalls.h中的函数定义，需要对应上才能使用。

在include/linux/syscalls.h中加入函数定义：

asmlinkage long sys\_hide\_process(pid\_t pid);

asmlinkage long sys\_unhide\_process(pid\_t pid);



在kernel/sys.c中加入系统调用函数体：

SYSCALL\_DEFINE1(hide\_process, pid\_t, pid)

{

struct task\_struct \*task;

if (pid == 0) {

task = current;

} else {

task = find\_task\_by\_vpid(pid);

if (task == NULL)

return -1;

}

task->hide = 1;

return 1;

}

SYSCALL\_DEFINE1(unhide\_process, pid\_t, pid)

{

struct task\_struct \*task;

if (pid == 0) {

task = current;

} else {

task = find\_task\_by\_vpid(pid);

if (task == NULL)

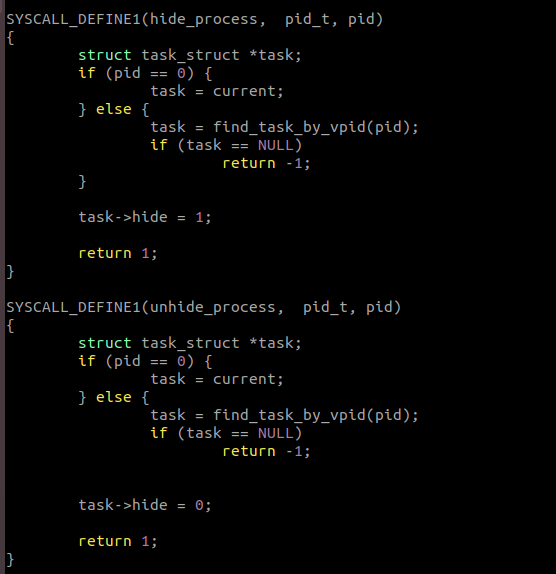
return -1;

}

task->hide = 0;

return 1;

}



SYSCALL\_DEFINE1为系统调用的实现函数，1表示函数实际上有一个参数，SYSCALL\_DEFINE1后面的三个参数为：函数名，参数类型，参数变量名。这里的操作其实就是修改task\_struct中的hide参数。

完成修改后进行内核编译。

cp /boot/config-`uname -r` .config

make mrproper

make menuconfig

sudo make install -j15

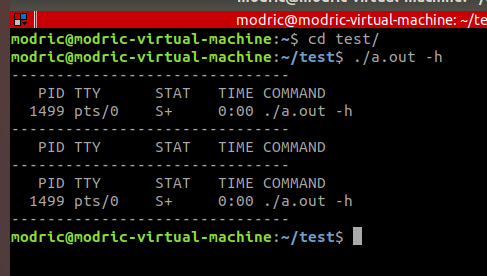
sudo make modules\_install

sudo make install

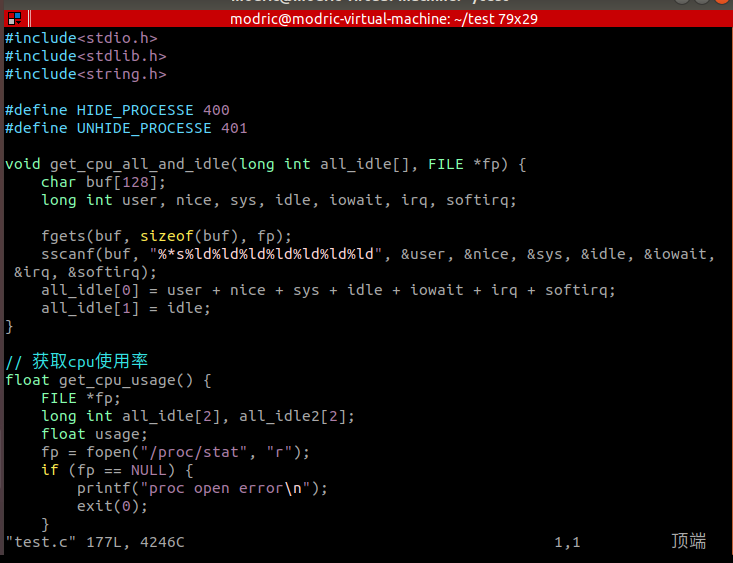
编译完成后通过修改grub进入新kernel



实验发现隐藏进程功能已经实现



部分代码如下，注意define的HIDE\_PROCESSE和UNHIDE\_PROCESSE为之前添加的调用号，需要做到对应。



## 5. 实验结论：

本次实验主要分为四个步骤:总体浏览proc目录，了解proc目录结构，通过cat命令直接查看部分文件内容，了解信息的存储方式；实际编程通过读取proc目录中相关文件的信息获取cpu使用率，内存使用率，以及网络负载和进程信息等；通过修改sys目录中相关文件调整内核参数；通过修改proc源码以及系统调用，对进程进行隐藏。通过学生逐步的实际操作，使学生认识并理解proc文件系统，掌握proc文件的特点，掌握从中获取linux系统信息的方法，了解可以利用proc文件系统进行用户态与内核态的交互。