### 实验9 定制生成proc文件

## 1实验目的

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。

## 2实验内容

查看proc目录，了解目录下各个子目录以及文件的含义；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proc源码，通过系统调用实现进程的隐藏。

## 3实验原理

3.1 proc文件系统梗概

在类Unix系统中体现了一种良好的抽象哲学，就是几乎所有的数据实体都被抽象成一个统一的接口--文件来看待，这样我们就可以用一些简单的基本工具完成大量复杂的操作。在Linux中存在着一类特殊的伪文件系统，用于使用与文件接口统一的操作来完成各种功能，例如ptyfs、devfs、sysfs和procfs。而procfs就是其中应用最广泛的一种伪文件系统。

Proc文件系统是Linux内核信息的抽象文件接口，大量内核中的信息以及可调参数都被作为常规文件映射到一个目录树中，这样我们就可以简单直接的通过echo或cat这样的文件操作命令对系统信息进行查取和调整了。同时proc文件系统也提供了一个接口，使得我们自己的内核模块或用户态程序可以通过proc文件系统进行参数的传递。在当今的Linux系统中，大量的系统工具也通过procfs获取内核参数，例如ps、lspci等等，没有proc文件系统它们将可能不能正常工作。

3.2 proc文件系统主要分类

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。

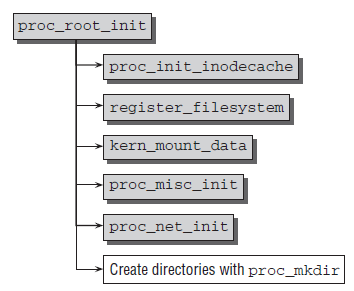
进程文件都是以数字为名的子目录，这个数字就是相关进程的进程ID，主要包含cmdline（执行进程的命令行参数），stat（进程状态）等

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。

3.3 proc文件系统初始化

在使用proc之前，我们必须首先初始化并挂载proc，并在内核内存中创建数据结构来描述文件系统。但是，不同的体系结构拥有不同的proc内容，所以，在初始化阶段并不完全创建子目录的内容，有些文件要等到系统运行时动态创建。Proc文件系统初始化的流程图如下：



**图 1 proc文件系统初始化流程图**

在第二阶段中，proc\_get\_inode(s, PROC\_ROOT\_INO, &proc\_root))传入的参数为proc\_root，这是一个全局变量，是proc的根目录的dentry，其定义为：

static const struct file\_operations proc\_root\_operations = {

.read = generic\_read\_dir,

.readdir = proc\_root\_readdir,

};

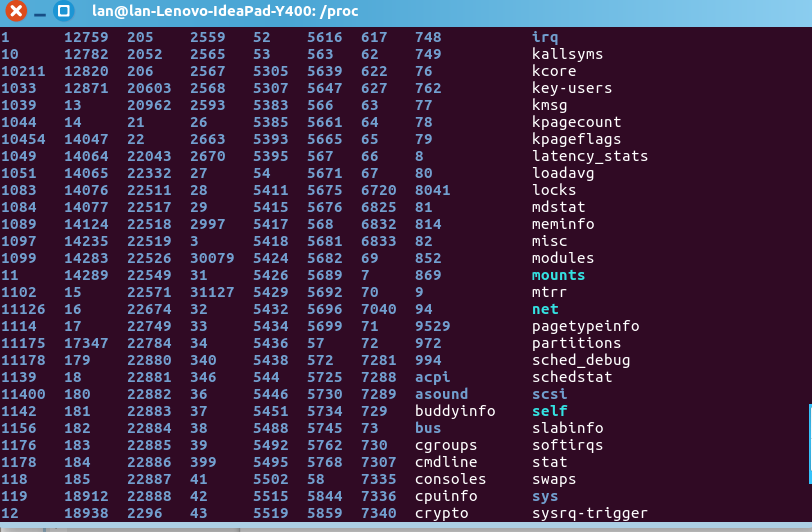
proc\_root\_readdir函数中先生成非进程目录，再通过proc\_pid\_readdir函数生成进程pid目录，这也是本次实验中需要主要关注的。对进程隐藏，实际上就是该进程不在proc文件系统体现，直接查看proc文件目录无法查找到该进程，这样ps命令就无法显示出该进程。在proc\_pid\_readdir函数中的for循环体就是往/proc添加<pid>目录，我们在填充目录之前判断该task\_struct中的hide变量，只有hide变量为0时才把该进程填充到/proc中，这样就可以达到隐藏进程的目的。其中hide变量可以通过系统调用进行修改。

## 4实验步骤

4.1查看proc目录，了解proc目录结构，并查看部分系统信息

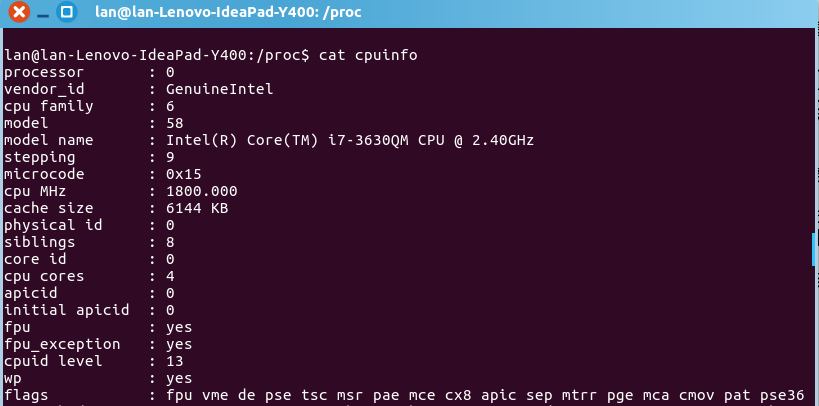
步骤1：cd /proc进入proc目录

步骤2：ls 整体查看proc目录结构



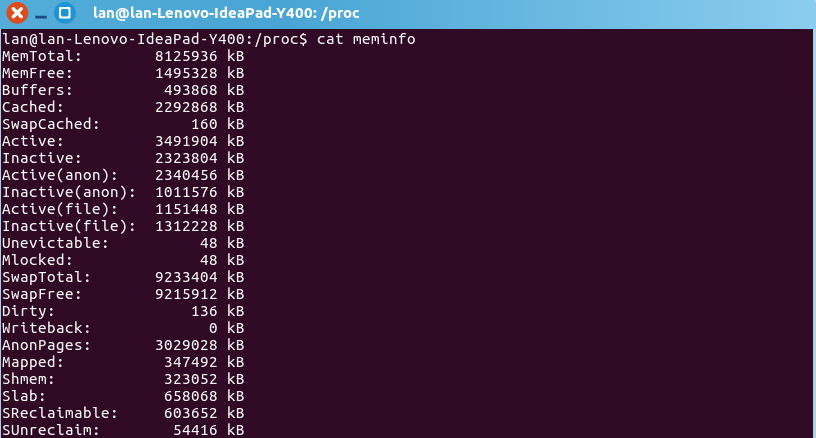
**图 2 proc目录整体结构**

步骤3：通过cat命令查看系统信息 cat cpuinfo



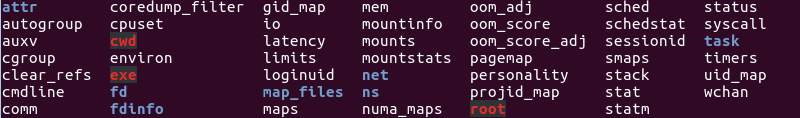
**图 3 cpuinfo内容**

步骤4：cat meminfo



**图 4 meminfo内容**

步骤5：cd PID号 进入某进程目录

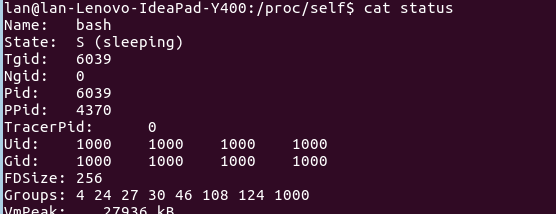


**图 5 进程目录结构**

步骤6：ls 查看进程目录结构

步骤7：cat cmdline 查看执行该进程的命令行参数

步骤8：cat status 查看该进程的信息



**图 6 进程信息**

步骤8：cd .. 返回proc根目录

步骤9：cd net 进入net目录，查看网络信息

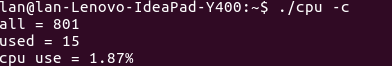
步骤10：cat dev 查看网络设备

4.2编写程序获取系统信息

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)。

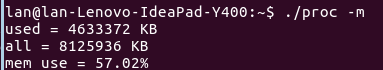
七个值的合为cpu总时间，idle为空闲时间。注意这七个值都为时刻，要想获得时间段，需要进行两次采样，做时间差，就可以获得当前时间的cpu时间。计算cpu使用率的公式为：cpu使用率=(cpu总时间-空闲时间)/cpu总时间。



**图 7 获取cpu使用率**

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。

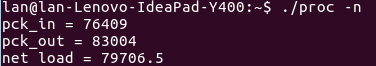


**图 8 获取内存使用率**

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。

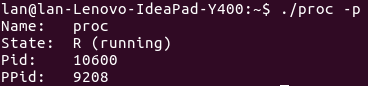
根据表头我们可以知道二到第七列为接受数据，之后为发送数据。一般我们只需要前四个数据：bytes收发的字节数；packets收发的正确的包量；errs收发错误的包量；drop收发丢弃的包量。计算网络负载的公式：网络负载=（接收的包数+发送的包数）/2。



**图 9 获取网络负载**

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。

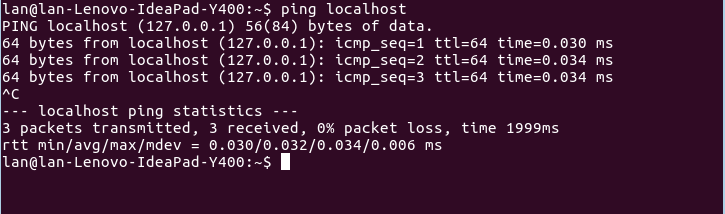


**图 10 获取进程信息**

4.3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

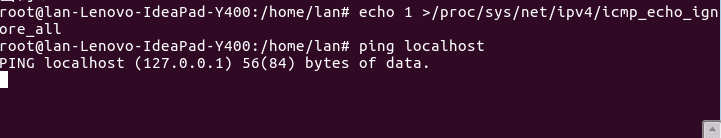


**图 11 ping本机**

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



**图 12 修改参数并再次ping主机**

4.4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最后，不要影响其他变量的偏移量

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

if(!iter.task->hide) {

filp->f\_pos = iter.tgid + TGID\_OFFSET;

if (proc\_pid\_fill\_cache(filp, dirent, \_\_filldir, iter) < 0) {

put\_task\_struct(iter.task);

goto out;

}

}

步骤3：添加两个系统调用，分别是隐藏和显示某进程。使用宏定义系统调用，要注意宏定义与直接定义的不同（参数定义方式）

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;

}

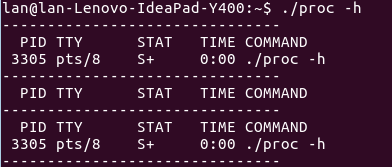
task->hide = 0;

return 1;

}

步骤4：编译新内核，选择新内核启动

步骤5：在用户态对进程的隐藏进行测试



**图 13 隐藏进程**

4.5参考代码

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define HIDE\_PROCESSE 322

#define UNHIDE\_PROCESSE 323

void get\_cpu\_all\_and\_idle(long int all\_idle[], FILE \*fp)

{

char buf[128];

long int user,nice,sys,idle,iowait,irq,softirq;

fgets(buf,sizeof(buf),fp);

sscanf(buf,"%\*s%ld%ld%ld%ld%ld%ld%ld",&user,&nice,&sys,&idle,&iowait,&irq,&softirq);

all\_idle[0] = user+nice+sys+idle+iowait+irq+softirq;

all\_idle[1] = idle;

}

// 获取cpu使用率

float get\_cpu\_usage()

{

FILE \*fp;

long int all\_idle[2],all\_idle2[2];

float usage;

fp = fopen("/proc/stat","r");

if(fp == NULL)

{

printf("proc open error\n");

exit (0);

}

get\_cpu\_all\_and\_idle(all\_idle, fp);

rewind(fp);

sleep(1);

get\_cpu\_all\_and\_idle(all\_idle2, fp);

// 采样两次 做差获得cpu时间 cpu使用率= (cpu总时间-空闲时间)/cpu总时间 \* 100

usage = (float)(all\_idle2[0]-all\_idle[0]-(all\_idle2[1]-all\_idle[1])) / (all\_idle2[0]-all\_idle[0])\*100 ;

printf("all = %ld\n",all\_idle2[0]-all\_idle[0]);

printf("used = %ld\n",all\_idle2[0]-all\_idle[0]-(all\_idle2[1]-all\_idle[1]));

printf("cpu use = %.2f%%\n",usage);

fclose(fp);

return usage;

}

// 获取内存使用率

float get\_mem\_usage()

{

FILE \*fp;

char buf[128];

long int mem\_total,mem\_free;

float usage;

fp = fopen("/proc/meminfo","r");

if(fp == NULL)

{

printf("proc open error\n");

exit (0);

}

fgets(buf,sizeof(buf),fp);

sscanf(buf,"%\*s%ld",&mem\_total);

fgets(buf,sizeof(buf),fp);

sscanf(buf,"%\*s%ld",&mem\_free);

usage = (float)(mem\_total-mem\_free)/mem\_total \* 100;

printf("used = %ld KB\n",mem\_total-mem\_free);

printf("all = %ld KB\n",mem\_total);

printf("mem use = %.2f%%\n",usage);

fclose(fp);

return usage;

}

void skipline(FILE \*f)

{

int ch;

do

{

ch = getc(f);

} while ( ch != '\n' && ch != EOF );

}

// 获取网络平均负载

float get\_net\_load()

{

FILE \*fp;

char buf[128];

char name[10];

long int pck\_in,pck\_out;

float load;

fp = fopen("/proc/net/dev","r");

if(fp == NULL)

{

printf("proc open error\n");

exit (0);

}

// 忽略头

skipline(fp);

skipline(fp);

// 忽略"lo"接口和无流量的接口

while (fgets(buf,sizeof(buf),fp))

{

sscanf(buf,"%s%\*ld%ld", name,&pck\_in);

if (strcmp(name,"lo:") != 0 && pck\_in != 0)

{

sscanf(buf,"%\*s%\*ld%\*ld%\*ld%\*ld%\*ld%\*ld%\*ld%\*ld%\*ld%ld",&pck\_out);

break;

}

}

// (接收数据包+发送数据包)/2

load = (float)(pck\_in+pck\_out)/2;

printf("pck\_in = %ld\n",pck\_in);

printf("pck\_out = %ld\n",pck\_out);

printf("net load = %.1f\n",load);

fclose(fp);

return load;

}

void output\_process()

{

FILE \*fp;

char buf[128];

fp = fopen("/proc/self/status","r");

if(fp == NULL)

{

printf("proc open error\n");

exit (0);

}

fgets(buf,sizeof(buf),fp);

printf("%s",buf);

fgets(buf,sizeof(buf),fp);

printf("%s",buf);

skipline(fp);

skipline(fp);

fgets(buf,sizeof(buf),fp);

printf("%s",buf);

fgets(buf,sizeof(buf),fp);

printf("%s",buf);

fclose(fp);

}

void hide\_process()

{

int pid = getpid();

char command[80];

sprintf(command,"ps %d\n",pid);

printf("-------------------------------\n");

system(command);

printf("-------------------------------\n");

syscall(HIDE\_PROCESSE,pid);

system(command);

syscall(UNHIDE\_PROCESSE,pid);

printf("-------------------------------\n");

system(command);

printf("-------------------------------\n");

}

int main(int argc ,char \*argv[])

{

char c1,c2;

if(argc > 1)

{

sscanf(argv[1], "%c%c", &c1, &c2);

if(c1 != '-')

exit(1);

if(c2 == 'c')

get\_cpu\_usage();

else if(c2 == 'm')

get\_mem\_usage();

else if(c2 == 'n')

get\_net\_load();

else if(c2 == 'h')

hide\_process();

else if(c2 == 'p')

output\_process();

}

return 1;

}

## 5实验总结

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。

## 6参考文献

[Linux中/proc目录下文件详解](http://bbs.chinaunix.net/thread-2175986-1-1.html)：详细介绍proc目录中各个文件的意义

## 7实验拓展

1. 根据参考文献中proc目录更详细的介绍，再通过编程获取几个自己想要的系统信息
2. 通过修改/proc/sys中的文件修改内核参数，重启后会重置，请想办法让这些参数永久保存（修改/etc/sysctl.conf）
3. 请通过proc实现内核态与用户态交换数据（linux/proc\_fs.h）