<http://www.cnblogs.com/hoys/archive/2011/04/10/2011675.html>

sysctl是一种用户应用来设置和获得运行时内核的配置参数的一种有效方式，通过这种方式，用户应用可以在内核运行的任何时刻来改变内核的配置参数，也可以在任何时候获得内核的配置参数，通常，内核的这些配置参数也出现在proc文件系统的/proc/sys目录下，用户应用可以直接通过这个目录下的文件来实现内核配置的读写操作，例如，用户可以通过

cat /proc/sys/net/ipv4/ip\_forward

来得知内核IP层是否允许转发IP包，用户可以通过

echo 1 > /proc/sys/net/ipv4/ip\_forward

把内核 IP 层设置为允许转发 IP 包，即把该机器配置成一个路由器或网关。 一般地，所有的 Linux 发布也提供了一个系统工具 sysctl，它可以设置和读取内核的配置参数，但是该工具依赖于 proc 文件系统，为了使用该工具，内核必须支持 proc 文件系统。下面是使用 sysctl 工具来获取和设置内核配置参数的例子：

# sysctl net.ipv4.ip\_forward

net.ipv4.ip\_forward = 0

# sysctl -w net.ipv4.ip\_forward=1

net.ipv4.ip\_forward = 1

# sysctl net.ipv4.ip\_forward

net.ipv4.ip\_forward = 1

注意，参数 net.ipv4.ip\_forward 实际被转换到对应的 proc 文件/proc/sys/net/ipv4/ip\_forward，选项 -w 表示设置该内核配置参数，没有选项表示读内核配置参数，用户可以使用 sysctl -a 来读取所有的内核配置参数，对应更多的 sysctl 工具的信息，请参考手册页 sysctl(8)。

但是 proc 文件系统对 sysctl 不是必须的，在没有 proc 文件系统的情况下，仍然可以，这时需要使用内核提供的系统调用 sysctl 来实现对内核配置参数的设置和读取。

在源代码中给出了一个实际例子程序，它说明了如何在内核和用户态使用sysctl。头文件 sysctl-exam.h 定义了 sysctl 条目 ID，用户态应用和内核模块需要这些 ID 来操作和注册 sysctl 条目。内核模块在文件 sysctl-exam-kern.c 中实现，在该内核模块中，每一个 sysctl 条目对应一个 struct ctl\_table 结构，该结构定义了要注册的 sysctl 条目的 ID(字段 ctl\_name)，在 proc 下的名称（字段procname），对应的内核变量（字段data，注意该该字段的赋值必须是指针），条目允许的最大长度（字段maxlen，它主要用于字符串内核变量，以便在对该条目设置时，对超过该最大长度的字符串截掉后面超长的部分），条目在proc文件系统下的访问权限（字段mode），在通过 proc设置时的处理函数（字段proc\_handler，对于整型内核变量，应当设置为&proc\_dointvec，而对于字符串内核变量，则设置为 &proc\_dostring），字符串处理策略（字段strategy，一般这是为&sysctl\_string）。

sysctl 条目可以是目录，此时 mode 字段应当设置为 0555，否则通过 sysctl 系统调用将无法访问它下面的 sysctl 条目，child 则指向该目录条目下面的所有条目，对于在同一目录下的多个条目，不必一一注册，用户可以把它们组织成一个 struct ctl\_table 类型的数组，然后一次注册就可以，但此时必须把数组的最后一个结构设置为NULL，即

{

.ctl\_name = 0

}

注册sysctl条目使用函数register\_sysctl\_table(struct ctl\_table \*, int)，第一个参数为定义的struct ctl\_table结构的sysctl条目或条目数组指针，第二个参数为插入到sysctl条目表中的位置，如果插入到末尾，应当为0，如果插入到开头，则为非0。内核把所有的sysctl条目都组织成sysctl表。

当模块卸载时，需要使用函数unregister\_sysctl\_table(struct ctl\_table\_header \*)解注册通过函数register\_sysctl\_table注册的sysctl条目，函数register\_sysctl\_table在调用成功时返 回结构struct ctl\_table\_header，它就是sysctl表的表头，解注册函数使用它来卸载相应的sysctl条目。 用户态应用sysctl-exam-user.c通过sysctl系统调用来查看和设置前面内核模块注册的sysctl条目（当然如果用户的系统内核已经支持proc文件系统，可以直接使用文件操作应用如cat, echo等直接查看和设置这些sysctl条目）。

下面是作者运行该模块与应用的输出结果示例：

# insmod ./sysctl-exam-kern.ko

# cat /proc/sys/mysysctl/myint

# cat /proc/sys/mysysctl/mystring

# ./sysctl-exam-user

mysysctl.myint = 0

mysysctl.mystring = ""

# ./sysctl-exam-user 100 "Hello, World"

old value: mysysctl.myint = 0

new value: mysysctl.myint = 100

old vale: mysysctl.mystring = ""

new value: mysysctl.mystring = "Hello, World"

# cat /proc/sys/mysysctl/myint

# cat /proc/sys/mysysctl/mystring

Hello, World

#

头文件：sysctl-exam.h：

//header: sysctl-exam.h

#ifndef \_SYSCTL\_EXAM\_H

#define \_SYSCTL\_EXAM\_H

#include <linux/sysctl.h>

#define MY\_ROOT (CTL\_CPU + 10)

#define MY\_MAX\_SIZE 256

enum {

MY\_INT\_EXAM = 1,

MY\_STRING\_EXAM = 2,

};

#endif

内核模块代码  sysctl-exam-kern.c：

//kernel module: sysctl-exam-kern.c

#include <linux/kernel.h>

#include <linux/module.h>

#include <linux/sysctl.h>

#include "sysctl-exam.h"

static char mystring[256];

static int myint;

static struct ctl\_table my\_sysctl\_exam[] = {

{

.ctl\_name = MY\_INT\_EXAM,

.procname = "myint",

.data = &myint,

.maxlen = sizeof(int),

.mode = 0666,

.proc\_handler = &proc\_dointvec,

},

{

.ctl\_name = MY\_STRING\_EXAM,

.procname = "mystring",

.data = mystring,

.maxlen = MY\_MAX\_SIZE,

.mode = 0666,

.proc\_handler = &proc\_dostring,

.strategy = &sysctl\_string,

}，

{

.ctl\_name = 0

}

};

static struct ctl\_table my\_root = {

.ctl\_name = MY\_ROOT,

.procname = "mysysctl",

.mode = 0555,

.child = my\_sysctl\_exam,

};

static struct ctl\_table\_header \* my\_ctl\_header;

static int \_\_init sysctl\_exam\_init(void)

{

my\_ctl\_header = register\_sysctl\_table(&my\_root, 0);

return 0;

}

static void \_\_exit sysctl\_exam\_exit(void)

{

unregister\_sysctl\_table(my\_ctl\_header);

}

module\_init(sysctl\_exam\_init);

module\_exit(sysctl\_exam\_exit);

MODULE\_LICENSE("GPL");