<http://www.cnblogs.com/hoys/archive/2011/04/10/2011141.html>

procfs是比较老的一种用户态与内核态的数据交换方式，内核的很多数据都是通过这种方式出口给用户的，内核的很多参数也是通过这种方式来让用户方便设置的。除了sysctl出口到/proc下的参数，procfs提供的大部分内核参数是只读的。实际上，很多应用严重地依赖于procfs，因此它几乎是必不可少的组件。本节将讲解如何使用procfs。

Procfs提供了如下API：

struct proc\_dir\_entry \*create\_proc\_entry(const char \*name, mode\_t mode, struct proc\_dir\_entry \*parent)

该函数用于创建一个正常的proc条目，参数name给出要建立的proc条目的名称，参数mode给出了建立的该proc条目的访问权限，参数 parent指定建立的proc条目所在的目录。如果要在/proc下建立proc条目，parent应当为NULL。否则它应当为proc\_mkdir 返回的struct proc\_dir\_entry结构的指针。

extern void remove\_proc\_entry(const char \*name, struct proc\_dir\_entry \*parent)

该函数用于删除上面函数创建的proc条目，参数name给出要删除的proc条目的名称，参数parent指定建立的proc条目所在的目录。

struct proc\_dir\_entry \*proc\_mkdir(const char \* name, struct proc\_dir\_entry \*parent)

该函数用于创建一个proc目录，参数name指定要创建的proc目录的名称，参数parent为该proc目录所在的目录。

extern struct proc\_dir\_entry \*proc\_mkdir\_mode(const char \*name, mode\_t mode, struct proc\_dir\_entry \*parent)

struct proc\_dir\_entry \*proc\_symlink(const char \* name, struct proc\_dir\_entry\* parent, const char \*dest)

该函数用于建立一个proc条目的符号链接，参数name给出要建立的符号链接proc条目的名称，参数parent指定符号连接所在的目录，参数dest指定链接到的proc条目名称。

struct proc\_dir\_entry \*create\_proc\_read\_entry(const char \*name, mode\_t mode, struct proc\_dir\_entry \*base, read\_proc\_t \*read\_proc, void \* data);

该函数用于建立一个规则的只读proc条目，参数name给出要建立的proc条目的名称，参数mode给出了建立的该proc条目的访问权限，参 数base指定建立的proc条目所在的目录，参数read\_proc给出读去该proc条目的操作函数，参数data为该proc条目的专用数据，它将 保存在该proc条目对应的struct file结构的private\_data字段中。

struct proc\_dir\_entry \*create\_proc\_info\_entry(const char \*name, mode\_t mode, struct proc\_dir\_entry \*base, get\_info\_t \*get\_info);

该函数用于创建一个info型的proc条目，参数name给出要建立的proc条目的名称，参数mode给出了建立的该proc条目的访问权限， 参数base指定建立的proc条目所在的目录，参数get\_info指定该proc条目的get\_info操作函数。实际上get\_info等同于 read\_proc，如果proc条目没有定义个read\_proc，对该proc条目的read操作将使用get\_info取代，因此它在功能上非常类似于函数create\_proc\_read\_entry。

struct proc\_dir\_entry \*proc\_net\_create(const char \*name, mode\_t mode, get\_info\_t \*get\_info)

该函数用于在/proc/net目录下创建一个proc条目，参数name给出要建立的proc条目的名称，参数mode给出了建立的该proc条目的访问权限，参数get\_info指定该proc条目的get\_info操作函数。

struct proc\_dir\_entry \*proc\_net\_fops\_create(const char \*name, mode\_t mode, struct file\_operations \*fops)

该函数也用于在/proc/net下创建proc条目，但是它也同时指定了对该proc条目的文件操作函数。

void proc\_net\_remove(const char \*name)

该函数用于删除前面两个函数在/proc/net目录下创建的proc条目。参数name指定要删除的proc名称。

除了这些函数，值得一提的是结构struct proc\_dir\_entry，为了创建一了可写的proc条目并指定该proc条目的写操作函数，必须设置上面的这些创建proc条目的函数返回的指针 指向的struct proc\_dir\_entry结构的write\_proc字段，并指定该proc条目的访问权限有写权限。

为了使用这些接口函数以及结构struct proc\_dir\_entry，用户必须在模块中包含头文件linux/proc\_fs.h。

在源代码包中给出了procfs示例程序procfs\_exam.c，它定义了三个proc文件条目和一个proc目录条目，读者在插入该模块后应当看到如下结构：

//kernel module: procfs\_exam.c

#include <linux/config.h>

#include <linux/kernel.h>

#include <linux/module.h>

#include <linux/proc\_fs.h>

#include <linux/sched.h>

#include <linux/types.h>

#include <asm/uaccess.h>

#define STR\_MAX\_SIZE 255

static int int\_var;

static char string\_var[256];

static char big\_buffer[65536];

static int big\_buffer\_len = 0;

static struct proc\_dir\_entry \* myprocroot;

static int first\_write\_flag = 1;

int int\_read\_proc(char \*page, char \*\*start, off\_t off, int count, int \*eof, void \*data)

{

count = sprintf(page, "%d", \*(int \*)data);

return count;

}

int int\_write\_proc(struct file \*file, const char \_\_user \*buffer,unsigned long count, void \*data)

{

unsigned int c = 0, len = 0, val, sum = 0;

int \* temp = (int \*)data;

while (count) {

if (get\_user(c, buffer)) //从用户空间中得到数据

return -EFAULT;

len++;

buffer++;

count--;

if (c == 10 || c == 0)

break;

val = c - '0';

if (val > 9)

return -EINVAL;

sum \*= 10;

sum += val;

}

\* temp = sum;

return len;

}

int string\_read\_proc(char \*page, char \*\*start, off\_t off,int count, int \*eof, void \*data)

{

count = sprintf(page, "%s", (char \*)data);

return count;

}

int string\_write\_proc(struct file \*file, const char \_\_user \*buffer, unsigned long count, void \*data)

{

if (count > STR\_MAX\_SIZE) {

count = 255;

}

copy\_from\_user(data, buffer, count);

return count;

}

int bigfile\_read\_proc(char \*page, char \*\*start, off\_t off, int count, int \*eof, void \*data)

{

if (off > big\_buffer\_len) {

\* eof = 1;

return 0;

}

if (count > PAGE\_SIZE) {

count = PAGE\_SIZE;

}

if (big\_buffer\_len - off < count) {

count = big\_buffer\_len - off;

}

memcpy(page, data, count);

\*start = page;

return count;

}

int bigfile\_write\_proc(struct file \*file, const char \_\_user \*buffer, unsigned long count, void \*data)

{

char \* p = (char \*)data;

if (first\_write\_flag) {

big\_buffer\_len = 0;

first\_write\_flag = 0;

}

if (65536 - big\_buffer\_len < count) {

count = 65536 - big\_buffer\_len;

first\_write\_flag = 1;

}

copy\_from\_user(p + big\_buffer\_len, buffer, count);

big\_buffer\_len += count;

return count;

}

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

{

#ifdef CONFIG\_PROC\_FS

struct proc\_dir\_entry \* entry;

myprocroot = proc\_mkdir("myproctest", NULL);

entry = create\_proc\_entry("aint", 0644, myprocroot);

if (entry) {

entry->data = &int\_var;

entry->read\_proc = &int\_read\_proc;

entry->write\_proc = &int\_write\_proc;

}

entry = create\_proc\_entry("astring", 0644, myprocroot);

if (entry) {

entry->data = &string\_var;

entry->read\_proc = &string\_read\_proc;

entry->write\_proc = &string\_write\_proc;

}

entry = create\_proc\_entry("bigprocfile", 0644, myprocroot);

if (entry) {

entry->data = &big\_buffer;

entry->read\_proc = &bigfile\_read\_proc;

entry->write\_proc = &bigfile\_write\_proc;

}

#else

printk("This module requires the kernel to support procfs,\n");

#endif

return 0;

}

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

{

#ifdef CONFIG\_PROC\_FS

remove\_proc\_entry("aint", myprocroot);

remove\_proc\_entry("astring", myprocroot);

remove\_proc\_entry("bigprocfile", myprocroot);

remove\_proc\_entry("myproctest", NULL);

#endif

}

module\_init(procfs\_exam\_init);

module\_exit(procfs\_exam\_exit);

MODULE\_LICENSE("GPL");