<http://www.cnblogs.com/hoys/archive/2011/04/10/2011124.html>

内核开发者经常需要向用户空间应用输出一些调试信息，在稳定的系统中可能根本不需要这些调试信息，但是在开发过程中，为了搞清楚内核的行为，调试信息非常必要，printk可能是用的最多的，但它并不是最好的，调试信息只是在开发中用于调试，而printk将一直输出，因此开发完毕后需要清除不必要 的printk语句，另外如果开发者希望用户空间应用能够改变内核行为时，printk就无法实现。因此，需要一种新的机制，那只有在需要的时候使用，它在需要时通过在一个虚拟文件系统中创建一个或多个文件来向用户空间应用提供调试信息。

有几种方式可以实现上述要求：

(1)使用procfs，在/proc创建文件输出调试信息，但是procfs对于大于一个内存页（对于x86是4K）的输出比较麻烦，而且速度慢，有时回出现一些意想不到的问题。

(2)使用sysfs（2.6内核引入的新的虚拟文件系统），在很多情况下，调试信息可以存放在那里，但是sysfs主要用于系统管理，它希望每一个文件对应内核的一个变量，如果使用它输出复杂的数据结构或调试信息是非常困难的。

(3)使用libfs创建一个新的文件系统，该方法极其灵活，开发者可以为新文件系统设置一些规则，使用libfs使得创建新文件系统更加简单，但是仍然超出了一个开发者的想象。

(4)为了使得开发者更加容易使用这样的机制，Greg Kroah-Hartman开发了debugfs（在2.6.11中第一次引入），它是一个虚拟文件系统，专门用于输出调试信息，该文件系统非常小，很容易使用，可以在配置内核时选择是否构件到内核中，在不选择它的情况下，使用它提供的API的内核部分不需要做任何改动。

使用debugfs的开发者首先需要在文件系统中创建一个目录，下面函数用于在debugfs文件系统下创建一个目录：

struct dentry \*debugfs\_create\_dir(const char \*name, struct dentry \*parent);

参数name是要创建的目录名，参数parent指定创建目录的父目录的dentry，如果为NULL，目录将创建在debugfs文件系统的根目录下。如果返回为-ENODEV，表示内核没有把debugfs编译到其中，如果返回为NULL，表示其他类型的创建失败，如果创建目录成功，返回指向该 目录对应的dentry条目的指针。

下面函数用于在debugfs文件系统中创建一个文件：

struct dentry \*debugfs\_create\_file(const char \*name, mode\_t mode, struct dentry \*parent,

void \*data, struct file\_operations \*fops);

参数name指定要创建的文件名，参数mode指定该文件的访问许可，参数parent指向该文件所在目录，参数data为该文件特定的一些数据， 参数fops为实现在该文件上进行文件操作的fiel\_operations结构指针，在很多情况下，由seq\_file提供的文件操作实现就足够了，因此使用debugfs很容易，当然，在一些情况下，开发者可能仅需要使用用户应用可以控制的变量来调试，debugfs也提供了4个这样的API方便开发者使用：

struct dentry \*debugfs\_create\_u8(const char \*name, mode\_t mode, struct dentry \*parent, u8 \*value);

struct dentry \*debugfs\_create\_u16(const char \*name, mode\_t mode, struct dentry \*parent, u16 \*value);

struct dentry \*debugfs\_create\_u32(const char \*name, mode\_t mode, struct dentry \*parent, u32 \*value);

struct dentry \*debugfs\_create\_bool(const char \*name, mode\_t mode, struct dentry \*parent, u32 \*value);

参数name和mode指定文件名和访问许可，参数value为需要让用户应用控制的内核变量指针。

当内核模块卸载时，Debugfs并不会自动清除该模块创建的目录或文件，因此对于创建的每一个文件或目录，开发者必须调用下面函数清除：

void debugfs\_remove(struct dentry \*dentry);

参数dentry为上面创建文件和目录的函数返回的dentry指针。

在下面给出了一个使用debufs的示例模块debugfs\_exam.c，为了保证该模块正确运行，必须让内核支持debugfs， debugfs是一个调试功能，因此它位于主菜单Kernel hacking，并且必须选择Kernel debugging选项才能选择，它的选项名称为Debug Filesystem。为了在用户态使用debugfs，用户必须mount它，下面是在作者系统上的使用输出：

$ mkdir -p /debugfs

$ mount -t debugfs debugfs /debugfs

$ insmod ./debugfs\_exam.ko

$ ls /debugfs

debugfs-exam

$ ls /debugfs/debugfs-exam

u8\_var u16\_var u32\_var bool\_var

$ cd /debugfs/debugfs-exam

$ cat u8\_var

$ echo 200 > u8\_var

$ cat u8\_var

$ cat bool\_var

N

$ echo 1 > bool\_var

$ cat bool\_var

Y

//kernel module: debugfs\_exam.c

#include <linux/config.h>

#include <linux/module.h>

#include <linux/debugfs.h>

#include <linux/types.h>

/\*dentry:目录项，是Linux文件系统中某个索引节点(inode)的链接。这个索引节点可以是文件，也可以是目录。

Linux用数据结构dentry来描述fs中和某个文件索引节点相链接的一个目录项(能是文件,也能是目录)。

　　（1）未使用（unused）状态：该dentry对象的引用计数d\_count的值为0，但其d\_inode指针仍然指向相关的的索引节点。该目录项仍然包含有效的信息，只是当前没有人引用他。这种dentry对象在回收内存时可能会被释放。

　　（2）正在使用（inuse）状态：处于该状态下的dentry对象的引用计数d\_count大于0，且其d\_inode指向相关的inode对象。这种dentry对象不能被释放。

　　（3）负（negative）状态：和目录项相关的inode对象不复存在（相应的磁盘索引节点可能已被删除），dentry对象的d\_inode指针为NULL。但这种dentry对象仍然保存在dcache中，以便后续对同一文件名的查找能够快速完成。这种dentry对象在回收内存时将首先被释放。

\*/

static struct dentry \*root\_entry, \*u8\_entry, \*u16\_entry, \*u32\_entry, \*bool\_entry;

static u8 var8;

static u16 var16;

static u32 var32;

static u32 varbool;

static int \_\_init exam\_debugfs\_init(void)

{

root\_entry = debugfs\_create\_dir("debugfs-exam", NULL);

if (!root\_entry) {

printk("Fail to create proc dir: debugfs-exam\n");

return 1;

}

u8\_entry = debugfs\_create\_u8("u8-var", 0644, root\_entry, &var8);

u16\_entry = debugfs\_create\_u16("u16-var", 0644, root\_entry, &var16);

u32\_entry = debugfs\_create\_u32("u32-var", 0644, root\_entry, &var32);

bool\_entry = debugfs\_create\_bool("bool-var", 0644, root\_entry, &varbool);

return 0;

}

static void \_\_exit exam\_debugfs\_exit(void)

{

debugfs\_remove(u8\_entry);

debugfs\_remove(u16\_entry);

debugfs\_remove(u32\_entry);

debugfs\_remove(bool\_entry);

debugfs\_remove(root\_entry);

}

module\_init(exam\_debugfs\_init);

module\_exit(exam\_debugfs\_exit);

MODULE\_LICENSE("GPL");