# Linux驱动\_debugfs

## 1. debugfs介绍

DebugFS，是一种用于内核调试的虚拟文件系统，内核通过debugfs和用户空间交换数据。类似procfs和sysfs等，这些文件系统都在内存里。

最常用的内核调试手段是printk。但在调试时可能需要修改某些内核变量，这时printk就无能为力，procfs和sysfs虽然也可以实现这个功能，但是偏离了本意，故debugfs从天而降。默认情况下，debugfs会被挂载在目录/sys/kernel/debug之下，如果没有自动挂载，可以用如下命令手动完成：

# mount -t debugfs none /sys/kernel/debug

我们分三种情况说明debugfs如何使用

比如生成树形目录结构如下的调试开关或变量:

mydebug 目录

subdir 目录

c 文件

a 变量

b blob变量

其中，a对应模块中的一个u8类型的变量，b和subdir下面的c都是对应模块里的一个字符数组，只是它们的实现方式不同。

在module\_init里，我们首先要建立根目录mydebug：

my\_debugfs\_root **=** debugfs\_create\_dir**(**"mydebug"**,** **NULL);**

第一个参数是目录的名称，第二个参数用来指定这个目录的上级目录，如果是NULL，则表示是放在debugfs的根目录里

子目录也是用debugfs\_create\_dir来实现

sub\_dir **=** debugfs\_create\_dir**(**"subdir"**,** my\_debugfs\_root**)**；

建立文件a的代码非常简单：

debugfs\_create\_u8**(**"a"**,** 0644**,** my\_debugfs\_root**,** **&**a**);**

这表示文件名为“a”，文件属性是0644，父目录是上面建立的“mydebug”，对应的变量是模块中的a。

b是一个32-bytes的字符数组，在debugfs里，数组可以用blob wrapper来实现。

char hello**[**32**]** **=** "hello"**;**

struct debugfs\_blob\_wrapper b**;**

b**.**data **=** **(**void**\*)**hello**;**

b**.**size **=** strlen**(**hello**)** **+** 1**;**

debugfs\_create\_blob**(**"b"**,** 0444**,** my\_debugfs\_root**,** **&**b**);**

blob wapper定义的数据只能是只读的。

c是一个文件，可以读写。读写函数在f\_ops中定义。

s\_c **=** debugfs\_create\_file**(**"c"**,** 0644**,** sub\_dir**,** **NULL,** **&**c\_fops**);**

## 2. debugfs示例代码

#include <linux/kernel.h>

#include <linux/module.h>

#include <linux/slab.h>

#include <linux/debugfs.h>

#include <asm/uaccess.h>

struct dentry **\***my\_debugfs\_root**;**

u8 a **=** 0**;**

char hello**[**32**]** **=** "Hello world!\n"**;**

struct debugfs\_blob\_wrapper b**;**

static int c\_open**(**struct inode **\***inode**,** struct file **\***filp**)**

**{**

filp**->**private\_data **=** inode**->**i\_private**;**

**return** 0**;**

**}**

static ssize\_t c\_read**(**struct file **\***filp**,** char \_\_user **\***buffer**,**size\_t count**,** loff\_t **\***ppos**)**

**{**

**if** **(\***ppos **>=** 32**)**

**return** 0**;**

**if** **(\***ppos **+** count **>** 32**)**

count **=** 32 **-** **\***ppos**;**

**if** **(**copy\_to\_user**(**buffer**,** hello **+** **\***ppos**,** count**))**

**return** **-**EFAULT**;**

**\***ppos **+=** count**;**

**return** count**;**

**}**

static ssize\_t c\_write**(**struct file **\***filp**,** const char \_\_user **\***buffer**,**size\_t count**,** loff\_t **\***ppos**)**

**{**

**if** **(\***ppos **>=** 32**)**

**return** 0**;**

**if** **(\***ppos **+** count **>** 32**)**

count **=** 32 **-** **\***ppos**;**

**if** **(**copy\_from\_user**(**hello **+** **\***ppos**,** buffer**,** count**))**

**return** **-**EFAULT**;**

**\***ppos **+=** count**;**

**return** count**;**

**}**

struct file\_operations c\_fops **=** **{**

**.**owner **=** THIS\_MODULE**,**

**.**open **=** c\_open**,**

**.**read **=** c\_read**,**

**.**write **=** c\_write**,**

**};**

static int \_\_init mydebugfs\_init**(**void**)**

**{**

struct dentry **\***sub\_dir**,** **\***r\_a**,** **\***r\_b**,** **\***s\_c**;**

printk**(**KERN\_INFO "mydebugfs\_init\n"**);**

my\_debugfs\_root **=** debugfs\_create\_dir**(**"mydebug"**,** **NULL);**

**if** **(!**my\_debugfs\_root**)**

**return** **-**ENOENT**;**

r\_a**=**debugfs\_create\_u8**(**"a"**,** 0644**,** my\_debugfs\_root**,** **&**a**);**

**if** **(!**r\_a**)**

**goto** Fail**;**

//debugfs\_blob\_wrapper is read only

b**.**data **=** **(**void **\*)**hello**;**

b**.**size **=** strlen**(**hello**)** **+** 1**;**

r\_b **=** debugfs\_create\_blob**(**"b"**,** 0644**,** my\_debugfs\_root**,** **&**b**);**

**if** **(!**r\_b**)**

**goto** Fail**;**

sub\_dir **=** debugfs\_create\_dir**(**"subdir"**,** my\_debugfs\_root**);**

**if** **(!**sub\_dir**)**

**goto** Fail**;**

//debufs\_create\_file could be readwrite

s\_c **=** debugfs\_create\_file**(**"c"**,** 0644**,** sub\_dir**,** **NULL,** **&**c\_fops**);**

**if** **(!**s\_c**)**

**goto** Fail**;**

**return** 0**;**

Fail**:**

debugfs\_remove**(**my\_debugfs\_root**);**

my\_debugfs\_root **=** **NULL;**

**return** **-**ENOENT**;**

**}**

static void \_\_exit mydebugfs\_exit**(**void**)**

**{**

printk**(**KERN\_INFO "mydebugfs\_exit\n"**);**

debugfs\_remove**(**my\_debugfs\_root**);**

**return;**

**}**

module\_init**(**mydebugfs\_init**);**

module\_exit**(**mydebugfs\_exit**);**

MODULE\_LICENSE**(**"GPL"**);**

## 3.debugfs测试

# cat /sys/kernel/debug/mydebug/a

0

# echo 1 > /sys/kernel/debug/mydebug/a

# cat /sys/kernel/debug/mydebug/a

1

# cat /sys/kernel/debug/mydebug/b

Hello world**!**

# cat /sys/kernel/debug/mydebug/subdir/c

Hello world**!**

# echo test > /sys/kernel/debug/mydebug/subdir/c

# cat /sys/kernel/debug/mydebug/subdir/c

test

world**!**

## 4.记得打开内核DEBUG\_FS宏

