苏州大学实验报告

院、系	计算机学院	年级专业	20 计科	姓名	柯骅	学号	2027405033
课程名称	操作系统课程实践					成绩	
指导教师	李培峰	同组实验者	无		实验日期	2023.06.08	

实 验 名 称 实验 11 /proc 文件系统

一、 实验目的

1. 学习使用 /proc 文件系统。
2. 使用 /proc 文件系统显示缺页状态。
3. 使用 /proc 文件系统输出超过一个页面的信息。

二、 实验内容

1. **（分析/proc 文件系统初始化）**
2. **（/proc 系统的一个简单应用）**

使用/proc 文件系统的一个简单例子。首先在/proc 目录下创建我们自己的目录 proc_example。然后在这个目录下创建三个普通文件(foo 、bar 、jiffies)和一个文件链接(jiffies_too)。

三、 实验步骤

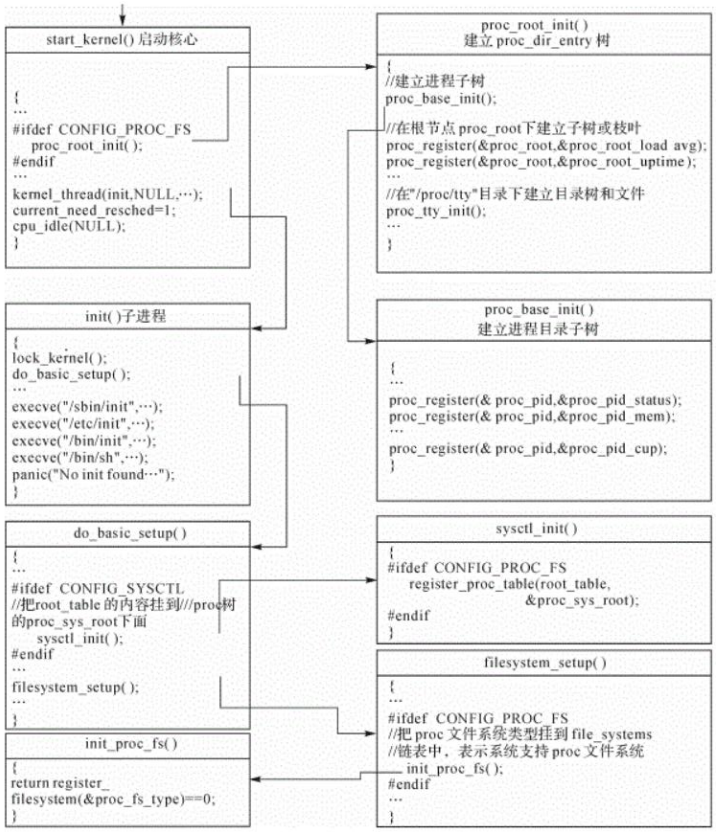
/proc 文件简介：

/proc 文件系统是一个伪文件系统，它只存在内存当中，而不占用外存空间。它以文件系统的方式为访问系统内核数据的操作提供接口。用户和应用程序可以通过/proc 得到系统的信息，并可以改变内核的某些参数。由于系统的信息，如进程，是动态改变的，所以用户或应用程序读取/proc 文件时，/proc 文件系统是动态从系统内核读出所需信息并提交的。

1. **（分析/proc 文件系统初始化）**

/proc 文件系统的初始化是从 init/main.c 中调用 proc_root_init()函数开始的。该函数在 fs/proc/root.c 文件中。

当系统启动时，内核初始化阶段会调用 init/main.c 中的 start_kernel()函数，它是 Linux 内核的入口点。在 start_kernel()函数中，会执行一系列的初始化操作，其中之一就是初始化/proc 文件系统。



教务处制

/proc 文件系统的初始化是从 init/main.c 中调用 proc_root_init()函数开始的。该函数在 fs/proc/root.c 文件中。在调用 proc_root_init()函数之前，start_kernel()函数会完成一系列的初始化操作，包括硬件初始化、内存管理的设置以及启动各个子系统等。

在 proc_root_init()函数中，首先会创建/proc 目录的根节点，并将其与 proc_root 指针关联起来。这个根节点是一个 struct proc_dir_entry 类型的数据结构，表示/proc 目录在内核中的表示。接着，函数会为一些特定的/proc 文件（例如/proc/cpuinfo 和/proc/meminfo）创建对应的 proc_dir_entry 节点，并将它们添加到/proc 目录下。

这些 proc_dir_entry 节点包含了文件的相关信息，如文件名、访问权限和操作函数等。通过这些节点，用户可以访问和操作/proc 文件系统中的数据。

/proc 文件系统的初始化过程还涉及其他一些步骤，例如初始化与进程相关的/proc 文件（如/proc/self 和/proc/<pid>）以及设备驱动程序相关的/proc 文件（如/proc/devices 和/proc/modules）等。

总而言之。/proc 文件系统的初始化是通过调用 proc_root_init()函数开始的，函数在 fs/proc/root.c 文件中实现。它负责创建/proc 目录的根节点以及其他与/proc 文件相关的节点，为用户提供访问和操作/proc 文件系统的接口。

2. （/proc 系统的一个简单应用）

Linux 内核模块介绍：

Linux 操作系统的内核是单体系结构(monolithic kernel)的，即整个内核是一个单独的非常大的程序。这样的操作系统内核把所有的模块都集成在了一起，系统的速度和性能都很好，但是可扩展性和可维护性就相对比较差。

为了改善单一体系结构内核的可扩展性和可维护性，Linux 操作系统使用了一种全新的内核模块机制——动态可加载内核模块(loadable kernel module,LKM)。用户可以根据需求，在不需要对内核重新编译的情况下，让模块能动态地装入内核或从内核移出。模块扩展了内核的功能，而无须重启系统。模块不是作为进程执行的，而是像其他静态连接的内核函数一样，在内核态代表当前进程执行。

模块与内核是在同样的地址空间中运行的，因此模块编程在一定意义上也就是内核编程。但并不是内核中所有的功能都可以使用模块来实现。Linux 内核中极为重要的一些功能，如进程管理、内存管理等，仍难以通过模块来实现，而必须直接对内核进行修改才能实现。在 Linux 系统中，经常利用内核模块实现的有文件系统、SCSI 高级驱动程序、大部分的 SCSI 普通驱动程序、多数 CD-ROM 驱动程序、以太网驱动程序等。

(1) 编写 procfs_example.c 文件

在初始化函数 static int __init init_procfs_example(void)中

通过 proc_mkdir 创建目录/procfs_example

通过 proc_create 创建 jiffies，内容为系统启动后经过的时间戳

通过 proc_create 创建 foo，并写入 name 和 value 都为“foo”的内容

通过 proc_create 创建 bar，并写入 name 和 value 都为“bar”的内容

通过 proc_symlink 创建 jiffies_too，该文件是 jiffies 的符号链接

在清理函数 static void __exit cleanup_procfs_example(void) 中

通过 remove_proc_entry 删除目录和文件

代码详见附件/test11/example1.c

(2) 编写 Makefile 文件

代码详见附件/test11/Makefile

(3) 编译代码，装入模块，卸载模块

在编写好 example1.c 文件与 Makefile 文件后，将 example.c 文件编译成模块，然后装载进我们的系统，测试能否完成相关文件的创建，并查看文件的属性，完成实验后，卸载掉实验的模块。

命令如下：

```
# sudo make //编译 example.c 文件
# sudo insmod example1.ko //安装模块
# cd /proc/procfs_example //进入 proc 目录
# cat bar foo jiffies jiffies_too //指定的四个文件
# ls -l //查看他们的属性
# vim jiffies //编辑 jiffies
```

四、 实验结果

（/proc 系统的一个简单应用）

(1) 编译 example1.c

使用附件中的代码进行编译，发现如下报错：

```
/home/kh/test/test11/example1.c: In function ‘init_procfs_example’:
/home/kh/test/test11/example1.c:64:16: error: dereferencing pointer to incomple
te type ‘struct proc_dir_entry’
     example_dir->owner = THIS_MODULE;
                ^~
./include/linux/proc_fs.h:14:6: error: field ‘read_proc’ declared as a function
  int read_proc(char *page, char **start, off_t off, int count, int *eof, void *data);
      ^~~~~~~~~
./include/linux/proc_fs.h:15:6: error: field ‘write_proc’ declared as a function
  int write_proc(struct file *file, const char *buffer, unsigned long count, void *data);
      ^~~~~~~~~~
/home/kh/test/test11/example1.c: In function ‘init_procfs_example’:
/home/kh/test/test11/example1.c:81:13: error: ‘struct proc_dir_entry’ has no member named ‘data’
     foo_file->data = &foo_data;
              ^~
/home/kh/test/test11/example1.c:93:13: error: ‘struct proc_dir_entry’ has no member named ‘data’
     bar_file->data = &bar_data;
              ^~
```

表示在内核代码中，并没有在结构体 proc_dir_entry 中定义名为 owner、read_proc、write_proc、data 的成员变量。

于是，搜索发现 proc_dir_entry 变量定义在”/usr/src/linux-4.16.10/include/proc_fs.h”使用 gedit 命令修改文件，命令如下：

```
# gedit /usr/src/linux-4.16.10/include/proc_fs.h
# cd /home/test/test11
# sudo make
```

添加如下代码将相应的成员变量添加到 proc_dir_entry 结构体中，再次使用 make 命令编译：

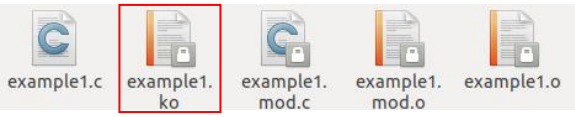
```
1. struct proc_dir_entry
2. {
3.     struct module *owner;
4.     int read_proc;
5.     int write_proc;
6.     struct fb_data_t* data
7. };
```

结果如下：

```
struct proc_dir_entry
{
        struct module *owner;
        int read_proc;
        int write_proc;
        struct fb_data_t* data
};
```

```
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/kh/test/test11/example1.o
see include/linux/module.h for more information
  LD [M]  /home/kh/test/test11/example1.ko
make[1]: 离开目录“/usr/src/linux-4.16.10”
```

查看 test/test11 目录，发现成功生成 example1.ko 模块文件：



(2) 装载模块

执行如下命令装载模块并查看是否成功：

```
# sudo insmod example1 //装载模块
# lsmod | grep example1 //查找模块
```

运行结果如下：

```
kh@ubuntu:~/test/test11$ sudo insmod example1.ko
kh@ubuntu:~/test/test11$ lsmod | grep example1
example1               16384  1
```

可以发现成功查找到了大小为 16 384 字节的 example1 模块,表明装载成功。

(3) 查看/proc/porcfs_example 目录中个文件及属性

命令如下：

```
# cd /proc/procfs_example
# cat bar foo jiffies jiffies_too
# ls -l
```

运行结果如下：

```
kh@ubuntu:~/test/test8/procfs_example$ cd /proc/procfs_example
kh@ubuntu:/proc/procfs_example$ cat bar foo jiffies jiffies_too
bar='bar'
foo='foo'
jiffies=4294950294
jiffies=4294950294
kh@ubuntu:/proc/procfs_example$ ls -l
总用量 0
-r--r--r-- 1 root root 0 6月  14 09:22 bar
-r--r--r-- 1 root root 0 6月  14 09:22 foo
-r--r--r-- 1 root root 0 6月  14 09:22 jiffies
lrwxrwxrwx 1 root root 7 6月  14 09:22 jiffies_too -> jiffies
```

可以发现成功查看了指定的四个文件的属性：

bar 和 foo 为可读文件，jiffies 为只读文件，jiffies_too 为 jiffies 的链接文件（可读可写）。

(4) 尝试修改 jiffies 文件

使用如下命令尝试修改只读文件 jiffies：

```
# vim jiffies
```

发现文件为只读文件：

```
"jiffies" [只读] 1L, 19C
```

尝试修改结果如下：

表明 jiffies 为只读文件，无法写入。

五、 实验思考与总结

1. （分析/proc 文件系统初始化）

总的来说，/proc文件系统的初始化过程通过proc_root_init()函数完成，它创建了/proc目录的根节点以及其他文件节点，为用户提供了访问和操作/proc文件系统的接口。在内核启动过程中的start_kernel()函数中，会进行一系列的初始化操作，其中之一就是初始化/proc文件系统。

2. （/proc 系统的一个简单应用）

(1) 说明为什么在编写内核模块时的输出函数常常选用了printk()而不是的printf()。

printf()在终端显示,printk()函数为内核空间里边的信息打印函数，就像c编程时用的printf()函数一样，专供内核中的信息展示用。在内核源代码中没有使用printf()的原因是在编译内核时还没有C的库函数可以供调用。

(2) 思考bar、foo、jiffies 和jiffies_too文件分别是什么类型，它们是否可以进行读写。

bar和foo为可读文件，jiffies为只读文件，jiffies_too为jiffies的链接文件（可读可写）。

3. 实验总结

本次是关于/proc 文件系统的实验,在第一个分析/proc 系统初始化的实验中,我们结合/proc 文件系统初始化函数调用示意图，理解了/proc 文件系统在初始化时是如何为用户创建可以访问的接口以及进程目录子树是在何时创建的等内容；

在第二个实验中，我们利用内核模块在/proc 目录下创建了 proc_example 目录，并在该目录下创建了三个普通文件（foo,bar,jiffies）和一个文件链接（jiffies_too），在此过程中，掌握了利用内核模块（机制）实现较复杂功能（例如创建目录与文件）的方法，同时学习并掌握了/proc 文件系统的相关知识。