## proc文件系统

proc文件系统是一个无存储的文件系统，当读其中的文件时，其内容动态生成。proc文件系统提供与内核通信的机制：内核部件可以通过该文件系统向用户空间提供接口来查询信息、修改软件行为，因而它是一种比较重要的文件系统。

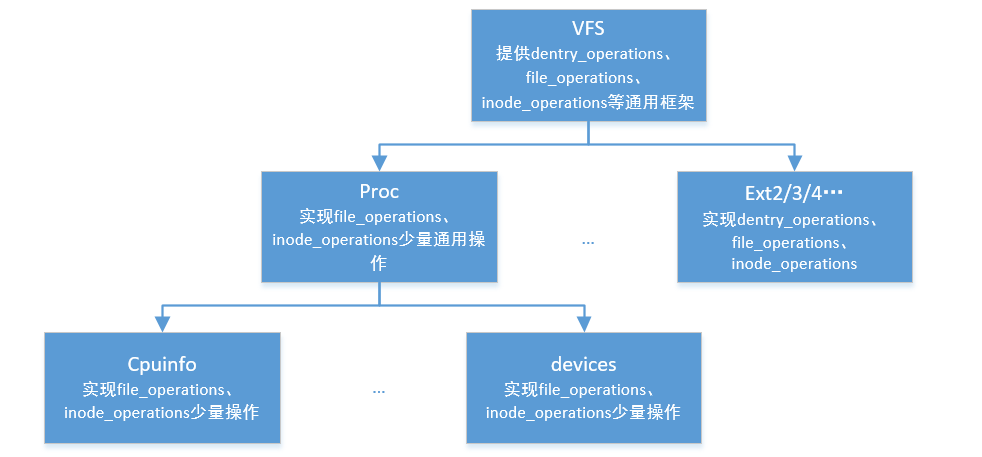
由于proc文件系统以文件的形式向用户空间提供了访问接口，这些接口可以在运行时获取相关部件的信息或者修改部件的行为，因而它是非常方便的一个接口。内核中大量使用了该文件系统。proc文件系统可以挂载在目录树的任意位置，不过通常挂载在/proc下，它大致包含了如下信息：

* 内存管理
* 每个进程的相关信息
* 文件系统
* 设备驱动程序
* 系统总线
* 电源管理
* 终端
* 系统控制参数
* 网络

这些信息几乎覆盖了内核的所有部分，因而proc文件系统对了解内核信息非常的重要。由于proc文件系统的内容取决于实现该文件的内核部件，因而各个文件的实现可能存在很大的不同。

## proc文件系统与其它磁盘文件系统的不同点

1. 存放位置不同：proc文件系统的内容均存放在RAM中，且只在读取操作时才从内核获取信息并存放在RAM中；而磁盘文件系统的内容均存放在磁盘上
2. 数据对象不同：proc文件系统最重要的数据结构是proc\_dir\_entry结构，代表了目录项；而磁盘文件系统有dentry、inode、file等等
3. 操作不同：proc文件系统仅实现了open、read等很少的文件操作；磁盘文件系统实现的操作非常多
4. 分层不同：因为内核信息众多且获取方式不一，所以proc文件系统仅提供了目录树的建立、inode\_operations、file\_operations等接口，具体的获取内核信息每个文件实现方式不同，例如cpuinfo获取CPU信息通过cpuinfo\_open，devices获取设备信息通过devinfo\_open。而磁盘文件系统的读写等操作均是统一的



## 常用命令

* 挂载proc文件系统

mount –t proc proc /proc

* 列出proc文件

# ls –l /proc/cpuinfo

-r—r—r--. 1 root root 0 Jan 21 21 : 34 /proc/cpuinfo

分别列出文件属性、硬链接数、文件拥有者、文件属组、文件大小、时间、文件名。这里的文件大小为0是因为proc文件系统只是把自己注册进了VFS(虚拟文件系统)，只有当VFS调用proc文件系统特定的读写函数时，proc文件系统才会根据内核的信息建立相应的文件。

* 查看proc文件内容

/proc/cpuinfo: cpu的信息（型号、家族、缓存大小等）

/proc/meminfo: 物理内存、交换空间等的信息

/proc/mounts: 已加载的文件系统的列表

/proc/devices: 可用设备的列表

/proc/filesystems: 被支持的文件系统

/proc/modules: 已加载的模块

/proc/version: 内核版本

/proc/cmdline: 系统启动时输入的内核命令行参数

## 内核结构

proc文件及目录在内核中用proc\_dir\_entry来表示，它在proc文件系统内部包含了proc文件系统的所有信息，其数据结构如下：

1. **struct** proc\_dir\_entry {
2. unsigned **int** low\_ino;
3. umode\_t mode;
4. nlink\_t nlink;
5. kuid\_t uid;
6. kgid\_t gid;
7. loff\_t size;
8. **const** **struct** inode\_operations \*proc\_iops;//inode操作
9. **const** **struct** file\_operations \*proc\_fops;//文件操作
10. **struct** proc\_dir\_entry \*next, \*parent, \*subdir;
11. **void** \*data;
12. read\_proc\_t \*read\_proc;
13. write\_proc\_t \*write\_proc;
14. atomic\_t count;
15. **int** pde\_users;
16. **struct** completion \*pde\_unload\_completion;
17. **struct** list\_head pde\_openers;
18. spinlock\_t pde\_unload\_lock;
19. u8 namelen;
20. **char** name[];
21. };

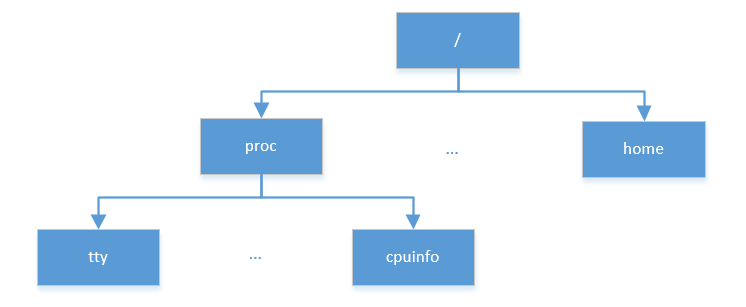
内核还提供了一个数据结构proc\_inode用于将特定于proc的数据与文件所对应的inode关联起来，其定义如下：

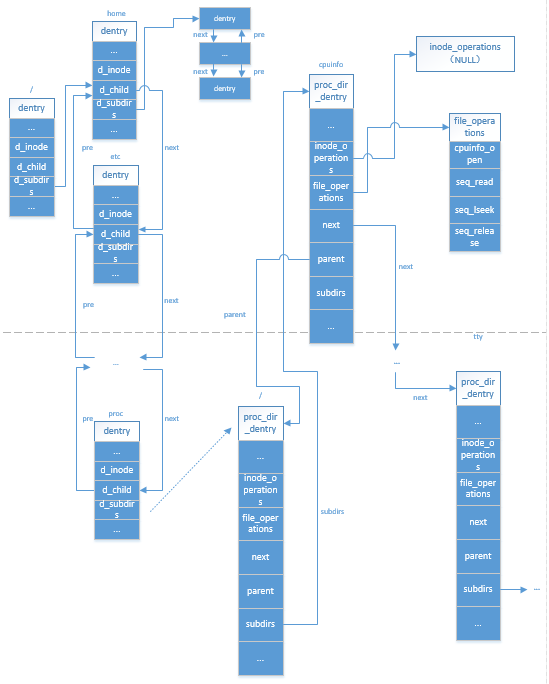
1. **struct** proc\_inode {
2. **struct** pid \*pid;
3. **int** fd;
4. **union** proc\_op op;
5. **struct** proc\_dir\_entry \*pde;
6. **struct** ctl\_table\_header \*sysctl;
7. **struct** ctl\_table \*sysctl\_entry;
8. **void** \*ns;
9. **struct** inode vfs\_inode;
10. };

内核定义proc文件系统的根节点如下：

1. **struct** proc\_dir\_entry proc\_root = {
2. .low\_ino = PROC\_ROOT\_INO,
3. .namelen = 5,
4. .mode = S\_IFDIR | S\_IRUGO | S\_IXUGO,
5. .nlink = 2,
6. .count = ATOMIC\_INIT(1),
7. .proc\_iops = &proc\_root\_inode\_operations,
8. .proc\_fops = &proc\_root\_operations,
9. .parent = &proc\_root,
10. .name = "/proc",
11. };

对于目录结构示意图与它所对应的内核结构图分别为：





## 相关操作

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

创建一个目录项，其中name为要创建的目录项的名字，parent为目录项的父目录。

* void proc\_remove(struct proc\_dir\_entry \*de)

删除一个目录

* struct proc\_dir\_entry \*proc\_create\_data(const char \*name,umode\_t mode,struct proc\_dir\_entry \*parent,const struct file\_operations \*proc\_fops,void \*data)

在proc文件系统中创建一个proc文件。其中的参数name为proc文件的名字，mode为proc文件的访问模式，parent为该proc文件所在的目录，proc\_fops为指向操作该文件的文件操作的指针，data为附加信息(通常为NULL)。

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

删除一个文件。参数name为要删除的文件名字，parent为要删除的文件所在的目录。

## 创建目录（/proc/sys举例）

对于目录/proc/sys，该目录在proc\_sys\_init（定义在/linux-3.10.61/fs/proc/proc\_sysctl.c）中被创建。函数如下：

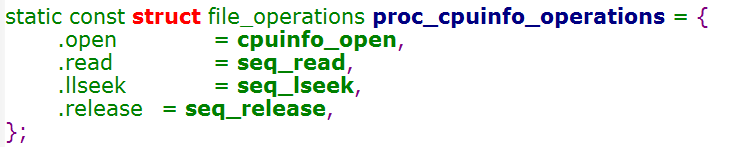
1. **int** \_\_init proc\_sys\_init(**void**)
2. {
3. **struct** proc\_dir\_entry \*proc\_sys\_root;
4. proc\_sys\_root = proc\_mkdir("sys", NULL);
5. proc\_sys\_root->proc\_iops = &proc\_sys\_dir\_operations;
6. proc\_sys\_root->proc\_fops = &proc\_sys\_dir\_file\_operations;
7. proc\_sys\_root->nlink = 0;
8. **return** 0;
9. }

创建了sys的目录结构proc\_sys\_root之后，调用proc\_register加入到目录树中，并且nlink增1：

1. **static** **int** proc\_register(struct proc\_dir\_entry \* dir, struct proc\_dir\_entry \* dp){
2. **struct** proc\_dir\_entry \*tmp;
3. int ret;
4. ret = proc\_alloc\_inum(&dp->low\_ino);
5. if (ret)
6. return ret;
7. if (S\_ISDIR(dp->mode)) {
8. dp->proc\_fops = &proc\_dir\_operations;
9. dp->proc\_iops = &proc\_dir\_inode\_operations;
10. dir->nlink++;
11. } else if (S\_ISLNK(dp->mode)) {
12. dp->proc\_iops = &proc\_link\_inode\_operations;
13. } else if (S\_ISREG(dp->mode)) {
14. BUG\_ON(dp->proc\_fops == NULL);
15. dp->proc\_iops = &proc\_file\_inode\_operations;
16. } else {
17. WARN\_ON(1);
18. return -EINVAL;
19. }
20. spin\_lock(&proc\_subdir\_lock);
21. for (tmp = dir->subdir; tmp; tmp = tmp->next)
22. if (strcmp(tmp->name, dp->name) == 0) {
23. WARN(1, "proc\_dir\_entry '%s/%s' already registered\n",
24. dir->name, dp->name);
25. break;
26. }
27. dp->next = dir->subdir;
28. dp->parent = dir;
29. dir->subdir = dp;
30. spin\_unlock(&proc\_subdir\_lock);
31. return 0;
32. }

## 读文件（/proc/cpuinfo举例）

cpuinfo的操作函数如下：



首先，读cpuinfo必须先打开它，调用cpuinfo\_open，该函数的主要作用是将文件对象file的private\_data字段初始化为seq\_file结构体，用来保存文件内容。

然后，调用seq\_read来从第一步的seq\_file结构体获取文件内容。