



作者 ice\_camel (/users/6a5b0c1d9347) 2014.02.17 14:24\*

写了61241字，被16人关注，获得了44个喜欢

(/users/6a5b0c1d9347)

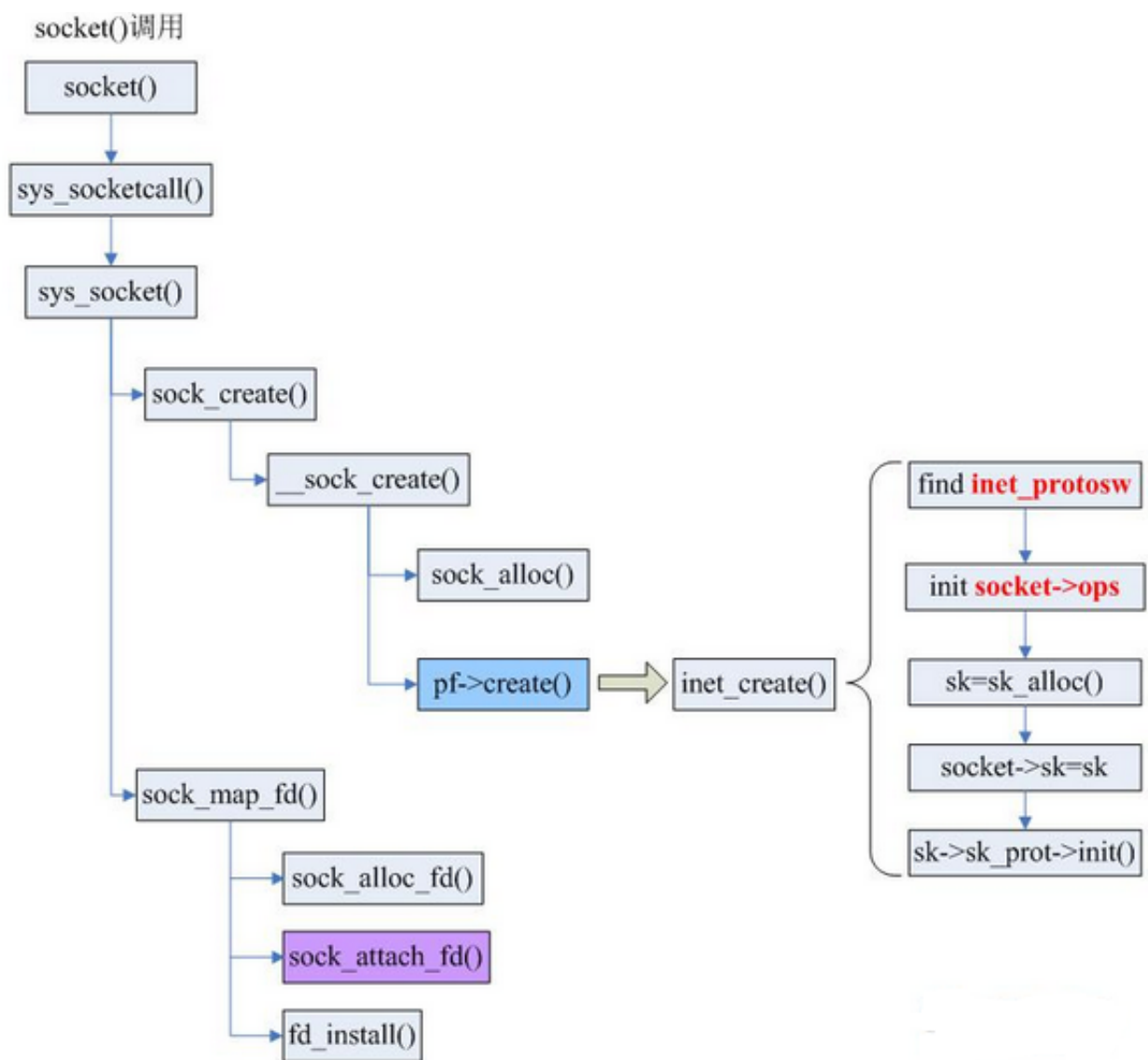
+ 添加关注 (/sign\_in)

# linux内核中socket的创建过程源码分析（总结性质）

字数2574 阅读1864 评论0 喜欢3

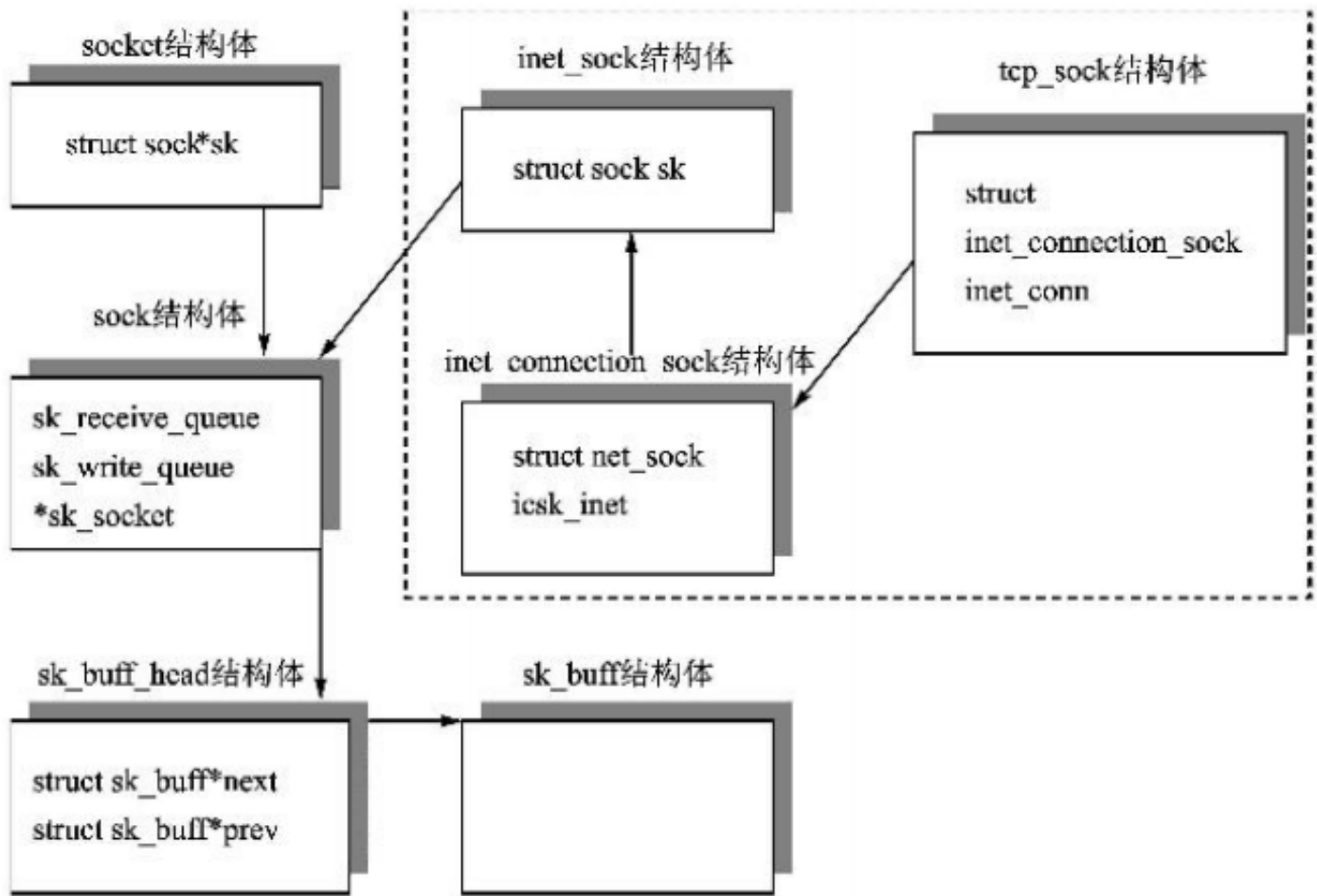
在漫长地分析完socket的创建源码后，发现一片浆糊，所以特此总结，我的博客中同时有另外一篇详细的源码分析，内核版本为3.9，建议在阅读本文后若还有兴趣再去看另外一篇博文。绝对不要单独看另外一篇。





一：调用链：

二：数据结构



——看一下每个数据结构的意义：

#### 1) socket, sock, inet\_sock, tcp\_sock的关系

创建完sk变量后，回到inet\_create函数中：

这里是根据sk变量得到inet\_sock变量的地址；这里注意区分各个不同结构体。

a. struct socket：这个是基本的BSD socket，面向用户空间，应用程序通过系统调用开始创建的socket都是该结构体，它是基于虚拟文件系统创建出来的；

类型主要有三种，即流式、数据报、原始套接字协议；

b. struct sock：它是网络层的socket；对应有TCP、UDP、RAW三种，面向内核驱动；

其状态相比socket结构更精细：

c. struct inet\_sock：它是INET域的socket表示，是对struct sock的一个扩展，提供INET域的一些属性，如TTL，组播列表，IP地址，端口等；

d. struct raw\_socket：它是RAW协议的一个socket表示，是对struct inet\_sock的扩展，它要处理

与ICMP相关的内容；

e. struct udp\_sock：它是UDP协议的socket表示，是对struct inet\_sock的扩展；

f. struct inet\_connection\_sock：它是所有面向连接的socket表示，是对struct inet\_sock的扩展；

g. struct tcp\_sock：它是TCP协议的socket表示，是对struct inet\_connection\_sock的扩展，主要增加滑动窗口，拥塞控制一些TCP专用属性；

h. struct inet\_timewait\_sock：它是网络层用于超时控制的socket表示；

i. struct tcp\_timewait\_sock：它是TCP协议用于超时控制的socket表示；

### 三：具体过程

#### 1、函数入口：

1) 示例代码如下：

```
int server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

#### 2) 入口：

net/Socket.c:sys\_socketcall()，根据子系统调用号，创建socket会执行sys\_socket()函数；

#### 2、分配socket结构：

1) 调用链：

```
net/Socket.c:sys_socket()->sock_create()->__sock_create()->sock_alloc()；
```

2) 在socket文件系统中创建i节点：

```
inode = new_inode(sock_mnt->mnt_sb);
```

这里，new\_inode函数是文件系统的通用函数，其作用是在相应的文件系统中创建一个inode；其主要代码如下(fs/Inode.c)：

上面有个条件判断：if (sb->s\_op->alloc\_inode)，意思是说如果当前文件系统的超级块有自己分配inode的操作函数，则调用它自己的函数分配inode，否则从公用的高速缓存区中分配一块inode；

3) 创建socket专用inode：

在“socket文件系统注册”一文中后面提到，在安装socket文件系统时，会初始化该文件系统的超级块，此时会初始化超级块的操作指针s\_op为sockfs\_ops结构；因此此时分配inode会调用sock\_alloc\_inode函数来完成：实际上分配了一个socket\_alloc结构体，该结构体包含socket和inode，但最终返回的是该结构体中的inode成员；至此，socket结构和inode结构均分配完毕；分配inode后，应用程序便可以通过文件描述符对socket进行read()/write()之类的操作，这个是由虚拟文件系统(VFS)来完成的。

### 3、根据inode取得socket对象：

由于创建inode是文件系统的通用逻辑，因此其返回值是inode对象的指针；但这里在创建socket的inode后，需要根据inode得到socket对象；内联函数SOCKET\_I由此而来，这里使用两个重要宏containerof和offsetof

### 4、使用协议族来初始化socket：

#### 1) 注册AF\_INET协议域：

在“socket文件系统注册”中提到系统初始化的工作，AF\_INET的注册也正是通过这个来完成的；

初始化入口net/ipv4/Af\_inet.c：这里调用sock\_register函数来完成注册：

根据family将AF\_INET协议域inet\_family\_ops注册到内核中的net\_families数组中；下面是其定义：

```
static struct net_proto_family inet_family_ops = { .family = PF_INET, .create = inet_create,
.owner = THIS_MODULE, };
```

其中，family指定协议域的类型，create指向相应协议域的socket的创建函数；

#### 2) 套接字类型

在相同的协议域下，可能会存在多个套接字类型；如AF\_INET域下存在流套接字(SOCK\_STREAM)，数据报套接字(SOCK\_DGRAM)，原始套接字(SOCK\_RAW)，在这三种类型的套接字上建立的协议分别是TCP, UDP, ICMP/IGMP等。

在Linux内核中，结构体struct proto表示域中的一个套接字类型，它提供该类型套接字上的所有操作及相关数据(在内核初始化时会分配相应的高速缓冲区，见上面提到的inet\_init函数)。

AF\_INET域的这三种套接字类型定义用结构体inet\_protosw(net/ipv4/Af\_inet.c)来表示，如下：其中，tcp\_prot(net/ipv4/Tcp\_ipv4.c)、udp\_prot(net/ipv4/Udp.c)、raw\_prot(net/ipv4/Raw.c)分别表示三种类型的套接字，分别表示相应套接字的 操作和相关数据；ops成员提供该协议域的全部操作集合，针对三种不同的套接字类型，有三种不同的域操作inet\_stream\_ops、inet\_dgram\_ops、inet\_sockraw\_ops，其定义均位于net/ipv4/Af\_inet.c下；

内核初始化时，在inet\_init中，会将不同的套接字存放到全局变量inetsw中统一管理；inetsw是一个链表数组，每一项都是一个struct inet\_protosw结构体的链表，总共有SOCK\_MAX项，在inet\_init函数对AF\_INET域进行初始化的时候，调用函数 inet\_register\_protosw把数组inetsw\_array中定义的套接字类型全部注册到inetsw数组中；其中相同套接字类型，不同协议类型的套接字通过链表存放在到inetsw数组中，以套接字类型为索引，在系统实际使用的时候，只使用inetsw，而不使用 inetsw\_array；

### 3) 使用协议域来初始化socket

了解了上面的知识后，我们再回到net/Socket.c:sys\_socket()->sock\_create()->\_\_sock\_create()中：

```
pf = rcu_dereference(net_families[family]); err = pf->create(net, sock, protocol);
```

上面的代码中，找到内核初始化时注册的协议域，然后调用其create方法；

### 5、分配sock结构：

sk是网络层对于socket的表示，结构体struct sock比较庞大，这里不详细列出，只介绍一些重要的成员，sk\_prot和sk\_prot\_creator，这两个成员指向特定的协议处理函数集，其类型是结构体struct proto，struct proto类型的变量在协议栈中总共也有三个.其调用链如下：

```
net/Socket.c:sys_socket()->sock_create()->__sock_create()-  
>net/ipv4/Af_inet.c:inet_create()；
```

inet\_create()主要完成以下几个工作：

1) 设置socket的状态为SS\_UNCONNECTED；

```
sock->state = SS_UNCONNECTED;
```

2) 根据socket的type找到对应的套接字类型：

由于同一type不同protocol的套接字保存在inetsw中的同一链表中，因此需要遍历链表来查找；在上面的例子中，会将protocol重新赋值为answer->protocol，即IPPROTO\_TCP，其值为6；

3) 使用匹配的协议族操作集初始化sk；

结合源码，sock变量的ops指向inet\_stream\_ops结构体变量；

4) 分配sock结构体变量 net/Socket.c:sys\_socket()->sock\_create()->\_\_sock\_create()->net/ipv4/Af\_inet.c:inet\_create()->net/core/Sock.c:sk\_alloc()；

其中，answer\_prot指向tcp\_prot结构体变量；

其中，sk\_prot\_alloc分配sock结构体变量；由于在inet\_init中为不同的套接字分配了高速缓冲区，因此该sock结构体变量会在该缓冲区中分配空间；分配完成后，对其做一些初始化工作：

i) 初始化sk变量的sk\_prot和sk\_prot\_creator；

ii) 初始化sk变量的等待队列；

iii) 设置net空间结构，并增加引用计数；

6、建立socket结构与sock结构的关系：

```
inet = inet_sk(sk);
```

这里为什么能直接将sock结构体变量强制转化为inet\_sock结构体变量呢？只有一种可能，那就是在分配sock结构体变量时，真正分配的是inet\_sock或是其他结构体；

我们回到分配sock结构体的那块代码(参考前面的5.4小节：net/core/Sock.c)：

```
static struct sock *sk_prot_alloc(struct proto *prot, gfp_t priority, int family) { struct sock
*sk; struct kmem_cache *slab; slab = prot->slab; if (slab != NULL) sk =
kmem_cache_alloc(slab, priority); else sk = kmalloc(prot->obj_size, priority); return sk; }
```

上面的代码在分配sock结构体时，有两种途径，一是从tcp专用高速缓存中分配；二是从内存直接分配；前者在初始化高速缓存时，指定了结构体大小为prot->obj\_size；后者也有指定大小为prot->obj\_size，

根据这点，我们看下tcp\_prot变量中的obj\_size(net/ipv4/Tcp\_ipv4.c)：

```
.obj_size = sizeof(struct tcp_sock),
```

也就是说，分配的真实结构体是tcp\_sock；由于tcp\_sock、inet\_connection\_sock、inet\_sock、sock之间均为0处偏移量，因此可以直接将tcp\_sock直接强制转化为inet\_sock。

## 2) 建立socket, sock的关系

创建完sock变量之后，便是初始化sock结构体，并建立sock与socket之间的引用关系；调用链如下：

```
net/Socket.c:sys_socket()->sock_create()->__sock_create()->net
/ipv4/Af_inet.c:inet_create()->net/core/Sock.c:sock_init_data()：
```

该函数主要工作是：

- a. 初始化sock结构的缓冲区、队列等；
- b. 初始化sock结构的状态为TCP\_CLOSE；
- c. 建立socket与sock结构的相互引用关系；

## 7、使用tcp协议初始化sock：

inet\_create()函数最后，通过相应的协议来初始化sock结构：这里调用的是tcp\_prot的init钩子函数net/ipv4/Tcp\_ipv4.c:tcp\_v4\_init\_sock()，它主要是对tcp\_sock和inet\_connection\_sock进行一些初始化；

## 8、socket与文件系统关联：

创建好与socket相关的结构后，需要与文件系统关联，详见sock\_map\_fd()函数：




- 1) 申请文件描述符，并分配file结构和目录项结构；
- 2) 关联socket相关的文件操作函数表和目录项操作函数表；
- 3) 将file->private\_data指向socket；


socket与文件系统关联后，以后便可以通过文件系统read/write对socket进行操作了；

如果觉得我的文章对您有用，请随意打赏。您的支持将鼓励我继续创作！

¥ 打赏支持

♡ 喜欢 | 3

 分享到微博

 分享到微信

更多分享 ▼

登录后发表评论 (/sign\_in)