Linux网络协议栈相关视频解析：

Tcp/ip协议栈技术专题训练营(1)

Tcp/ip协议栈技术专题训练营(2)

Linux网络协议栈是内核中最大的组件之一，由于网络部分应用的范围很广，也相对较热，该部分现有的资料很多，学起来也比较容易。首先，我们看看贯穿网络协议栈各层的一个最关键数据结构——套接字缓冲区（sk_buff结构）。

一个封包就存储在这个数据结构中。所有网络分层都会使用这个结构来存储其报头、有关数据的信息，以及用来协调工作的其他内部信息。在内核的进化历程中，这个结构经历多次变动，本文及后面的文章都是基于2.6.20版本，在2.6.32中该结构又变化了很多。该结构字段可粗略划分为集中类型：布局、通用、专用、可选（可用宏开关）。

SKB在不同网络层之间传递，可用于不同的网络协议。协议栈中的每一层往下一层传递SKB之前，首先就是调用skb_reserve函数在数据缓存区头部预留出的一定空间以保证每一层都能把本层的协议首部添加到数据缓冲区中。如果是向上层协议传递skb，则下层协议层的首部信息就没有用了，内核实现上用指针改变指向来实现。

下面看看该结构体中的字段，大部分都给了注释，后面的方法与实现中我们将看到他的各个字段的应用与实际意义。

```
struct sk_buff {         /* These two members must be first.
 */         /*??????????????????,????skb_buff_head*/
struct sk_buff            *next;         struct sk_buff
       *prev;                  /*?????????sock???????????
               ??????????????????????                    ?????
???????????L4                    ????????????????????
          ????????NULL*/         struct sock
  *sk;                 /*?????????????????         ?????????
?????????         ??????????????????         */         str
uct skb_timeval     tstamp;                  /*?????????????
??????SKB??????          ?????*/         struct net_device
   *dev;          /*?????????????????????*/         struct ne
t_device      *input_dev;          union {
 struct tcphdr     *th;                  struct udphdr    *u
h;                  struct icmphdr   *icmph;
```

```
   struct igmphdr  *igmph;                  struct iphdr
    *ipiph;                   struct ipv6hdr   *ipv6h;
             unsigned char   *raw;         } h;/*??????*/
      union {                   struct iphdr       *iph;
               struct ipv6hdr   *ipv6h;                   s
truct arphdr     *arph;                  unsigned char   *r
aw;         } nh;/*??????*/          union {
unsigned char *raw;         } mac;/*??????*/          /*????
???*/         struct  dst_entry      *dst;          /*IPSec?
??????????*/         struct        sec_path  *sp;          /
*          * This is the control buffer. It is free to use f
or every          * layer. Please put your private variables
 there. If you          * want to keep them across layers yo
u have to do a skb_clone()          * first. This is owned b
y whoever has the skb queued ATM.          */          /*SKB
??????????????????????????????????????*/         char
                cb[48];                /*??????????????????
(data??),SG???????I/O???                  ??FRAGLIST???????
I/O???????????????*/         unsigned int              len
,                /*SG???FRAGLIST??????I/O?????????*/
                           data_len,
                      /*????????????????????????????????????
*/                                  mac_len;         unio
n {                  __wsum             csum;
     __u32                  csum_offset;         };
       /*?????QOS???*/         __u32
  priority;                 /*???skb???????*/         __u8
                          local_df:1,          /*????SKB????
??*/                                  cloned:1,
/*???????????*/                                  ip_summe
d:2,                                   /*??payload
??????????????*/
nohdr:1,          /*?????*/
    nfctinfo:3;          /*????????????????????????
  ???????eth_type_trans()???*/         __u8
           pkt_type:3,          /*??????*/
                  fclone:2,
      ipvs_property:1;                /*????????????????????
?????????*/         __be16                        protocol;
```

```
                /*skb?????????skb?????????????*/         v
oid                     (*destructor)(struct sk_buff *skb); /
*??????????????????????????????????????????????????????????
????????????????????????????#ifdef????????????????????????
?????????Kconfig???*/#ifdef CONFIG_NETFILTER          /*????
?*/          struct nf_conntrack  *nfct;#if defined(CONFIG_NF
_CONNTRACK) || defined(CONFIG_NF_CONNTRACK_MODULE)          s
truct sk_buff             *nfct_reasm;#endif#ifdef CONFIG_BRI
DGE_NETFILTER          /*?????*/          struct nf_bridge_inf
o          *nf_bridge;#endif#endif /* CONFIG_NETFILTER */#ifd
ef CONFIG_NET_SCHED/*??????*/          __u16
          tc_index;  /* traffic control index */#ifdef CONFIG
_NET_CLS_ACT          /*??????*/          __u16
            tc_verd;    /* traffic control verdict */#endif#e
ndif#ifdef CONFIG_NET_DMA          dma_cookie_t             dm
a_cookie;#endif#ifdef CONFIG_NETWORK_SECMARK          __u32
                      secmark;#endif          __u32
                    mark;          /* These elements must be
at the end, see alloc_skb() for details.  */          /*?????
??+??????,???????len??????          ????len+sizeof(sk_buff)*/
          unsigned int               truesize;
  /*?????????????????*/          atomic_t              users;
                 /*head?end?????????????????????????????????
?          data?tail?????????????????????????*/          unsi
gned char             *head,
    *data,                                        *tail,
                              *end;};
```

可选功能字段

在网络模块中同时也提供了很多有用的功能，虽然这些功能都不是必须的，但对现在的应用来讲是不可缺少的一部分，例如，防火墙、组播等。为了支持这些功能，一般都需要在内核数据结构sk_buff中添加相应的成员变量。因此，sk_buff结构中包含很多想#ifdef这样的预编译指令。如下面的两个宏定义。

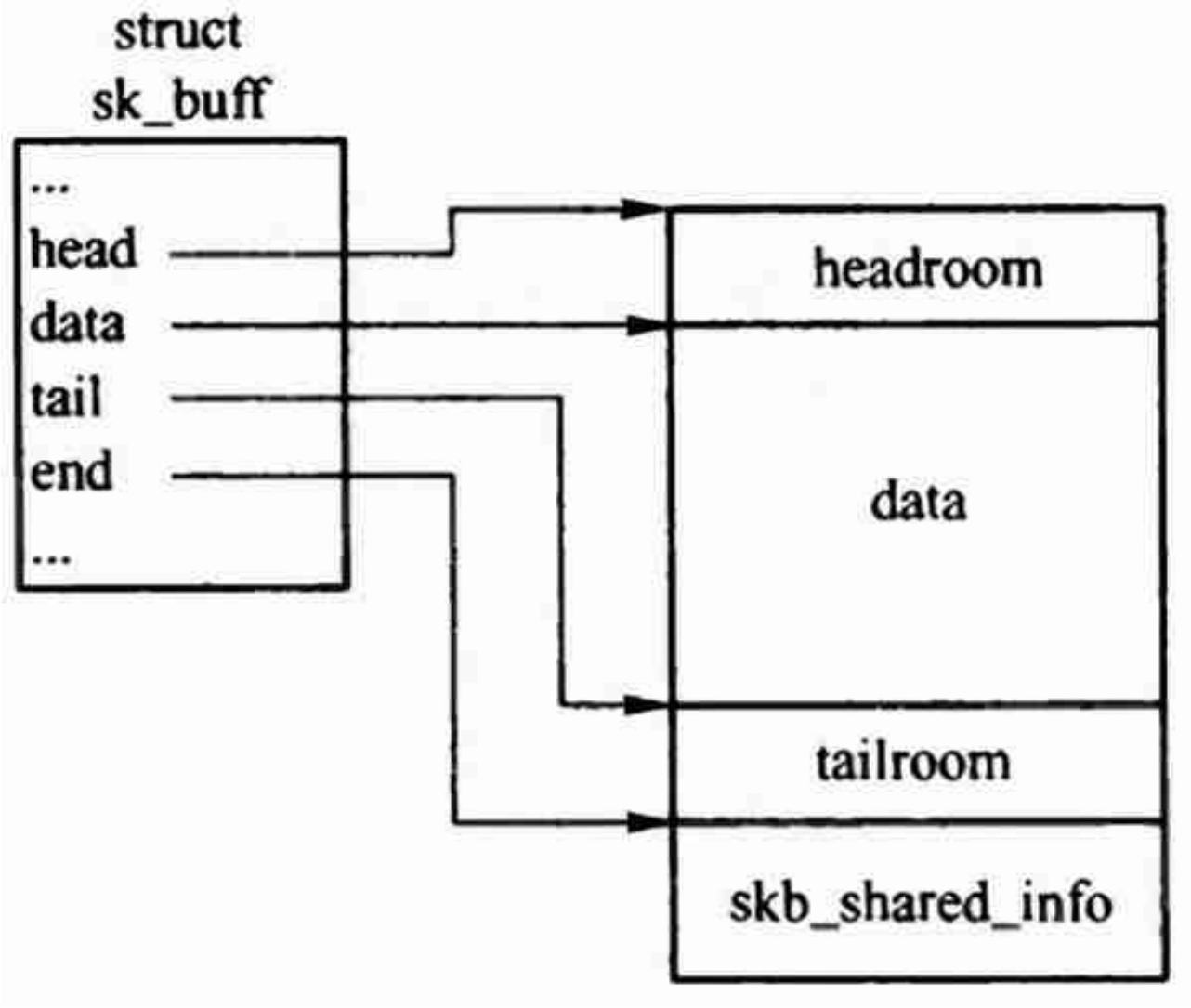
```
struct sk_buff {……/*?????????????????????????????????????
????????????????????????????????????????????#ifdef??????
????????????????????????????Kconfig???*/#ifdef CONFIG_NET_S
CHED          __u16                                tc_index;  /* tr
```

```
affic control index */#ifdef CONFIG_NET_CLS_ACT          __u1
6                           tc_verd;    /* traffic control ve
rdict */#endif          ……}
```

我们打开内核文件夹net->sched下面的Kconfig文件，发现有下面文字：

```
menu "QoS and/or fair queueing" config NET_SCHED          boo
l "QoS and/or fair queueing"……config NET_CLS_ACT          boo
l "Actions"          select NET_ESTIMATOR          ---help---…
…endif # NET_SCHED endmenu
```

与上面数据结构中的宏对应就显然了，如果需要了解内核配置选项与对应的宏，查看对应的Kconfig文件就可以了。需要指出的是，内核编译之后，由某些选项所控制的数据结构是固定的而不是动态变化的。一般来说，如果某些选项修改了内核数据结构，则包含该选项的组件就不能被编译成内核模块。



Skb初始化

网络模块中，有两个用来分配SKB描述符的高速缓存，在SKB模块初始化函数skb_init中被创建

```
void __init skb_init(void){          /*??????SKB???????????*
/          skbuff_head_cache = kmem_cache_create("skbuff_head
_cache",
 sizeof(struct sk_buff),
                  0,
            SLAB_HWCACHE_ALIGN|SLAB_PANIC,
                                    NULL, NULL);          /*??
???SKB?????????????????????          ?????*/          skbuff_fc
lone_cache = kmem_cache_create("skbuff_fclone_cache",
                                                  (2*sizeof(s
```

```
truct sk_buff)) +
            sizeof(atomic_t),
                        0,
                    SLAB_HWCACHE_ALIGN|SLAB_PANIC,
                                                NULL, NULL
);}
```

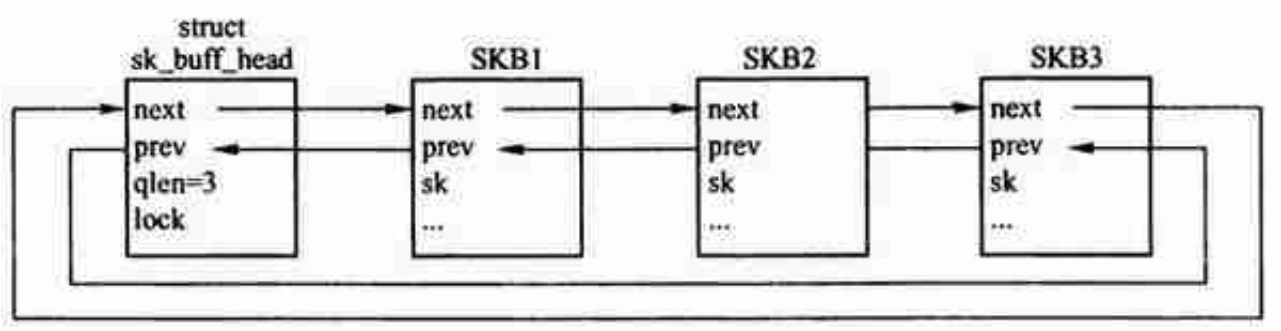
分配skb

Alloc_skb()用来分配SKB，数据缓存区描述符是两个不同的实体，这就意味着，在分配一个SKB时，需要分配两块内存，一块是数据缓存区，一块是SKB描述符。

```
struct sk_buff *__alloc_skb(unsigned int size, gfp_t gfp_mas
k,                                  int fclone, int node){
     struct kmem_cache *cache;          struct skb_shared_inf
o *shinfo;          struct sk_buff *skb;          u8 *data;
      /*??????????????????*/          cache = fclone ? skbuff
_fclone_cache : skbuff_head_cache;           /* Get the HEAD
*/          /*??????cache?*/          skb = kmem_cache_alloc_n
ode(cache, gfp_mask & ~__GFP_DMA, node);          if (!skb)
                  goto out;           /* Get the DATA. Size mu
st match skb_add_mtu(). */          /*???size*/          size
= SKB_DATA_ALIGN(size);          /*???????????size???skb_sh
ared_info????*/          data = kmalloc_node_track_caller(siz
e + sizeof(struct skb_shared_info),
   gfp_mask, node);          if (!data)                    got
o nodata;          memset(skb, 0, offsetof(struct sk_buff, t
ruesize));          /*??truesize???size+sizeof(struct sk_buff
)*/          skb->truesize = size + sizeof(struct sk_buff);
       atomic_set(&skb->users, 1);/*????????1*/          skb->
head = data;          skb->data = data;          skb->tail = d
ata;          /*??end?data+size??*/          skb->end  = data
+ size;          /* make sure we initialize shinfo sequential
ly */          /*skb_shared_info??skb->end???*/          shinf
o = skb_shinfo(skb);          /*??????*/          atomic_set(&
shinfo->dataref, 1);          shinfo->nr_frags  = 0;
shinfo->gso_size = 0;          shinfo->gso_segs = 0;
shinfo->gso_type = 0;          shinfo->ip6_frag_id = 0;
   shinfo->frag_list = NULL;          if (fclone) {/*???????
```

```
??*/                    struct sk_buff *child = skb + 1;/*???
??skb??*/                    /*??????*/                    ato
mic_t *fclone_ref = (atomic_t *) (child + 1);
    /*??????*/                    skb->fclone = SKB_FCLONE_OR
IG;                   atomic_set(fclone_ref, 1);/*????????1*/
                    child->fclone = SKB_FCLONE_UNAVAILABLE;
        }out:         return skb;nodata:         kmem_cache_
free(cache, skb);         skb = NULL;         goto out;}
```

调用该函数后生成的图如下所示：



对链表操作也增加了很多函数，包括初始化、入队列、出队列等等，也在skbuff.h中。

Skb_shared_info结构

在alloc_skb()看到，其中中分配数据部分分配了一个该结构，在数据缓存区的末尾，保存了数据块的附加信息。如下：

```
#define skb_shinfo(SKB)          ((struct skb_shared_info *
)((SKB)->end))
```

该结构定义如下：

```
struct skb_shared_info { /*??????????????SKB?????????????
??*/         atomic_t   dataref; /*ip???????????*/         u
nsigned short nr_frags; /*??GSO???MSS???GSO?????????????????
MSS????*/         unsigned short gso_size;          /* Warni
ng: this field is not always filled in (UFO)! *//*GSO?????gs
o_size???????gso_size???????????*/         unsigned short gs
o_segs; /*?SKB???????GSO??*/         unsigned short  gso_typ
```

```
e;          __be32          ip6_frag_id; /*ip?????????????:1
??????????????????????????IP???;2??UDP????????????SKB??????SK
B???????????????;3?????FRAGLIST???????I/O??????????????FR
AGLIST???????I/O,???????*/          struct sk_buff   *frag_li
st;         /*ip?????????????????????*/          skb_frag_t
       frags[MAX_SKB_FRAGS];};??skb_frag_t?????struct skb_fr
ag_struct {          /*???????????*/          struct page *pa
ge;          /*???????????????*/          __u16 page_offset
;          /*???????????????*/          __u16 size;};
```