SKB 概述

Sk_buff 描述符，用来描述将要发送或者接收到的报文。所以这个数据结构要在不同的协议层之间传递，要支持方便地添加和移除协议头部。比如当 sk_buff 从传输层传递到网络层时，要添加上传输层头部，所以在创建 SKB 时要预留出头部的空间，每一层负责填充该层的头部，然后移动指针。在 SKB 从链路层传给网络层时，需要移除链路层头部，实际上也只是移动 SKB 指针有效负载的指针。

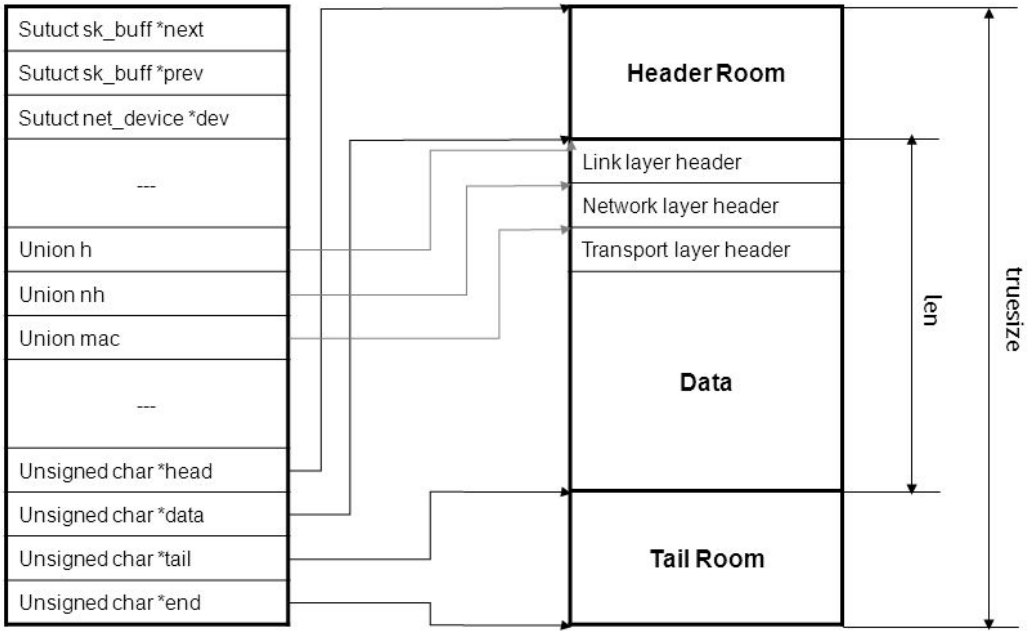
SKB 描述符的结构

```
struct sk_buff {
        /* These two members must be first. */
        struct sk_buff          *next;
        struct sk_buff          *prev;
/*  所有的 SKB 构成一个环形链表, sk_buff_header 指向这个链表的头结点  */
        struct sock             *sk;    /*    与这个 SKB 关联的 sock,只有当这个 SKB 是由本机发出或者接收才有效，如果 SKB 的源 IP 和目的 IP 都不是本机，那么 sk = NULL    */
        struct skb_timeval tstamp;    /*    这个数据包到达的时间，由接收这个数据包的网络设备来设置  */
        struct net_device   *dev;     /*  接收或发送这个 SKB 的网络设备  */
        struct net_device   *input_dev;    /*  接收这个数据包的网络设备  */

/*  传输层头部，支持各种协议，包括 tcp, udp, icmp, igmp,  所以用 union */
        union {
                struct tcphdr  *th;
                struct udphdr *uh;
                struct icmphdr      *icmph;
                struct igmphdr      *igmph;
                struct iphdr    *ipiph;    //  为啥需要这些网络层头部
                struct ipv6hdr *ipv6h;
                unsigned char*raw;
        } h;

/* network header,  网络层头部，支持 Ipv4, Ipv6, arp, raw  协议*/
        union {
                struct iphdr    *iph;
                struct ipv6hdr *ipv6h;
                struct arphdr  *arph;
                unsigned char*raw;
        } nh;
```

```
/* mac 层头部 */
    union {
        unsigned char       *raw;
    } mac;

    struct    dst_entry  *dst; /*  目的路由的缓存项*/
    struct     sec_path *sp;
    /*
     * This is the control buffer. It is free to use for every
     * layer. Please put your private variables there. If you
     * want to keep them across layers you have to do a skb_clone()
     * first. This is owned by whoever has the skb queued ATM.
     */
/*每一层都可以用这个 control buffer 来作为自己的控制块，可以每层的 private data，由每层协议自己来维护，只在本层有效。比如 tcp 层对应的结构是 tcp_skb_cb, 在访问时就会用这个宏来转换：#define TCP_SKB_CB(__skb)   ((struct tcp_skb_cb *) &(__skb->cb[0])) 来访问 skb 的 cb */
    char            cb[48];

    unsigned int        len, /* SKB 数据的长度，包括线性数据区和 SG/FRGALIST 类型的聚合分散 IO 的数据，由于对于下层来说，上层的首部也是有效负载，所以 len 包括了首部长度 */
                    data_len, //SG/FRGALIST 类型的聚合分散 IO 的数据长度
                    mac_len; //mac 层首部长度
    union {
        __wsum          csum;
        __u32           csum_offset;
    };
    __u32               priority; /* Qos 级别*/
    __u8                local_df:1,
                    cloned:1,   // 这个 SKB 是否已被克隆
                    ip_summed:2,
                    nohdr:1,
                    nfctinfo:3;

    __u8                pkt_type:3, /*  包类型，由二层的目的地址决定，比如 PACKET_HOST 是 MAC 地址就是本机的 MAC，这是发给本机的包，PACKET_BROADCAST 是 MAC 地址是个广播地址*/
                    fclone:2,
                    ipvs_property:1;

    __be16              protocol; /*从二层看到的上层协议地址，比如 IP, Ipv6, ARP 等*/
/*SKB 的析构指针函数*/
    void            (*destructor)(struct sk_buff *skb);
```

```
/* 为了支持防火墙等网络模块的功能， 包含的各种#ifdef 预编译指令 */
#ifdef CONFIG_NETFILTER
        struct nf_conntrack      *nfct;
#if defined(CONFIG_NF_CONNTRACK) || defined(CONFIG_NF_CONNTRACK_MODULE)
        struct sk_buff        *nfct_reasm;
#endif
#ifdef CONFIG_BRIDGE_NETFILTER
        struct nf_bridge_info    *nf_bridge;
#endif
#endif /* CONFIG_NETFILTER */

/*tc 流量控制，只有在编译内核时选择了 Networking->Networking Options->Qos and/or
fair queueing 才有效，通过选择这个选项在.config 文件中就定义了 CONFIG_NET_SCHED 这
个宏*/
#ifdef CONFIG_NET_SCHED
        __u16                tc_index; /* traffic control index */

/*包分类器， 只有编译内核时选择了 Networking->Networking Options->Qos and/or fair
queueing 才有效*/
#ifdef CONFIG_NET_CLS_ACT
        __u16                tc_verd;  /* traffic control verdict */
#endif
#endif

#ifdef CONFIG_NET_DMA
        dma_cookie_t        dma_cookie;
#endif

#ifdef CONFIG_NETWORK_SECMARK
        __u32                secmark;
#endif

        __u32                mark;

        /* These elements must be at the end, see alloc_skb() for details.  */
        unsigned int        truesize; /* 整个包所占的线性缓存区的大小 */
        atomic_t        users; /* 引用计数，只有 users = 0 时才会释放该 SKB */

        unsigned char        *head,  /* 整个包的线性缓存区的头*/
                        *data, /* 线性数据区的头*/
                        *tail,  /* 线性数据区的尾*/
                        *end; /* 整个包的线性缓存区的尾*/
};
```

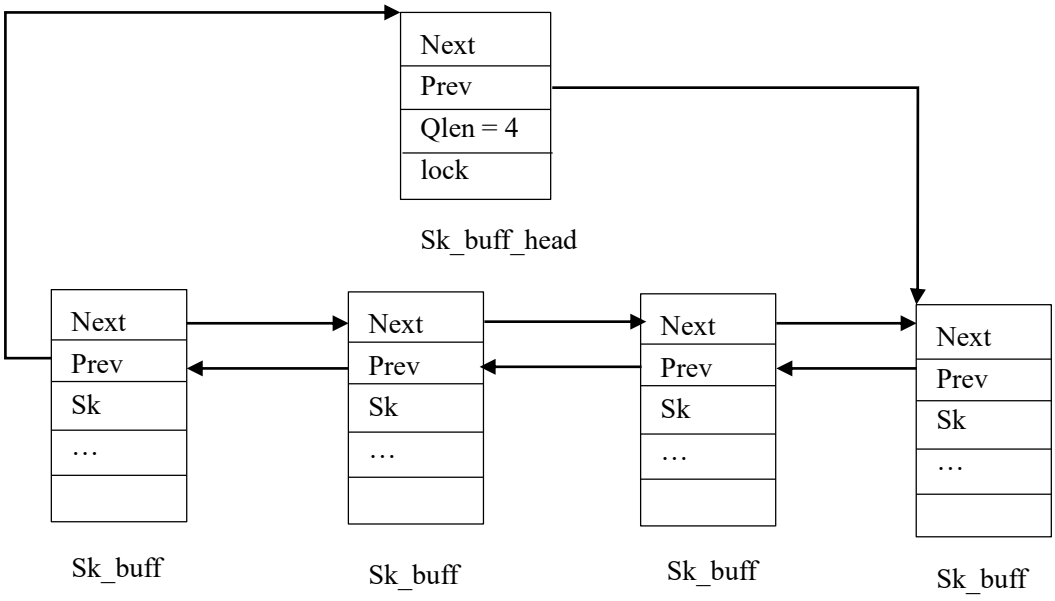
Sk_buff 中的几个指针的意义：



SKB 链表结构示意图：

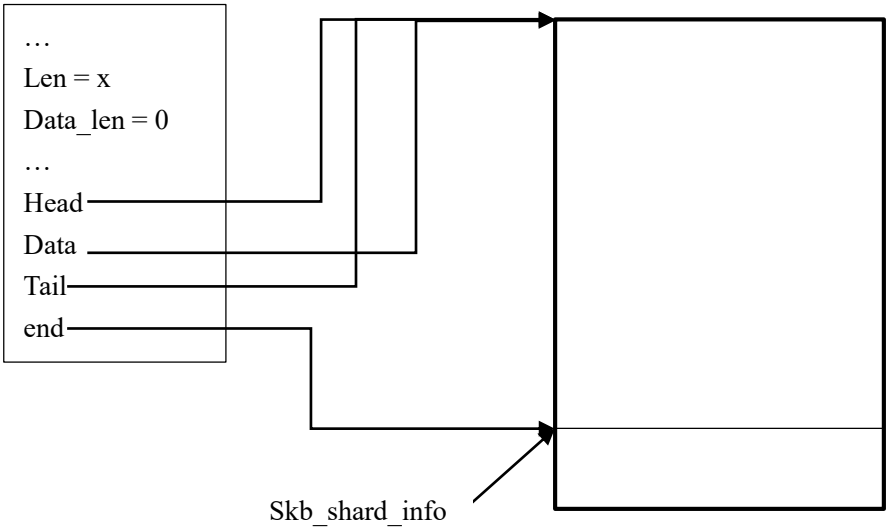
这是 skb 链表头的结构

```
struct sk_buff_head {
        /* These two members must be first. */
        struct sk_buff *next;
        struct sk_buff *prev;

        __u32           qlen;   //这个链表上有多少个节点
        spinlock_t      lock;
};
```

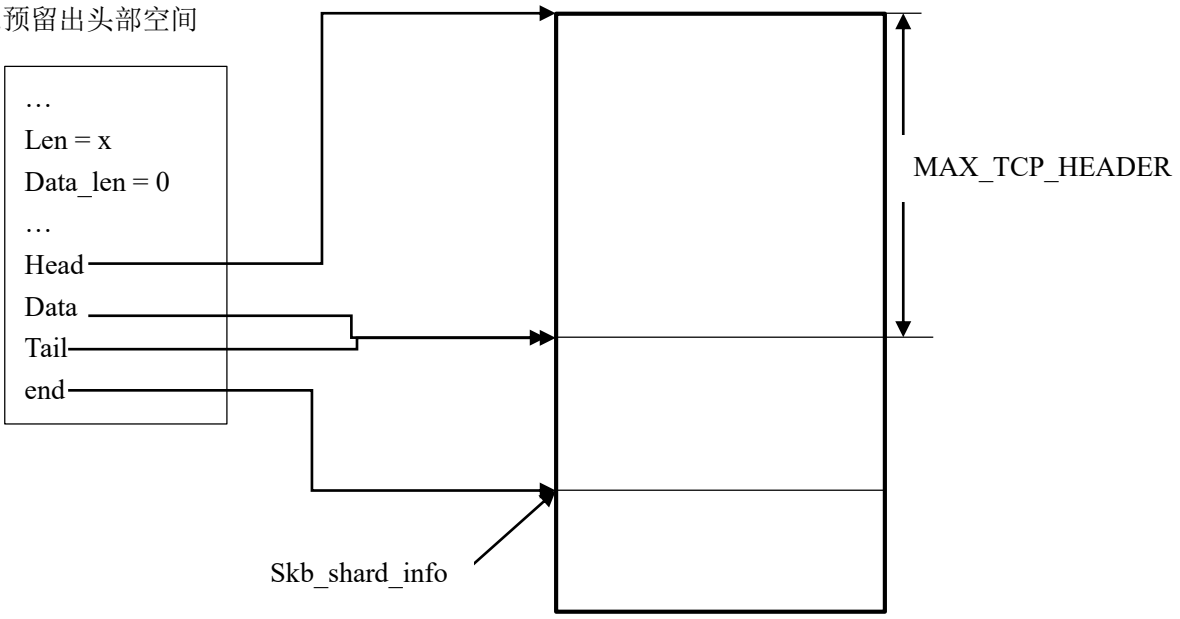


报文从 TCP 层传递到链路层 head, data, tail, end 四个指针的移动过程：

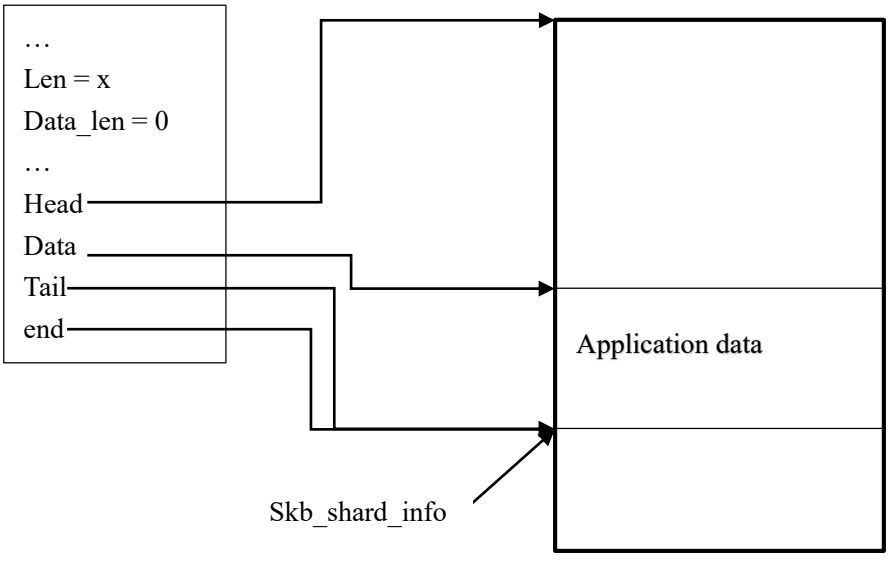
A.



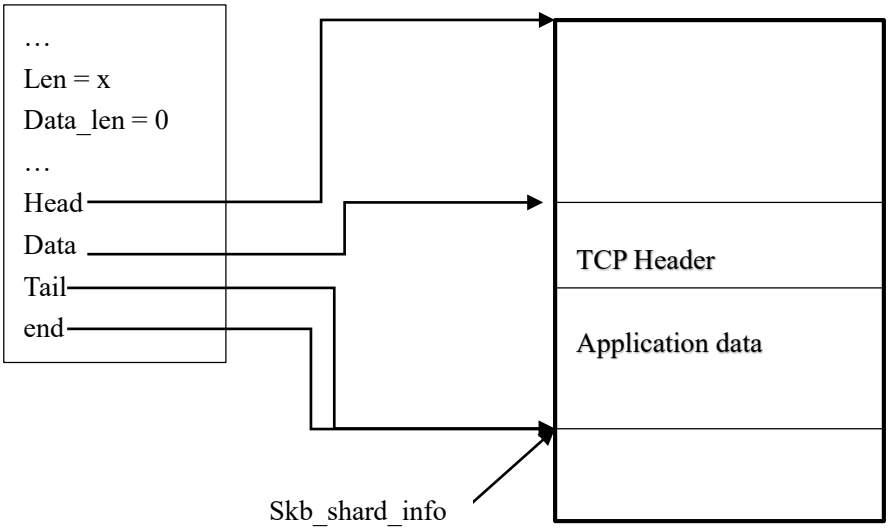
b.预留出头部空间



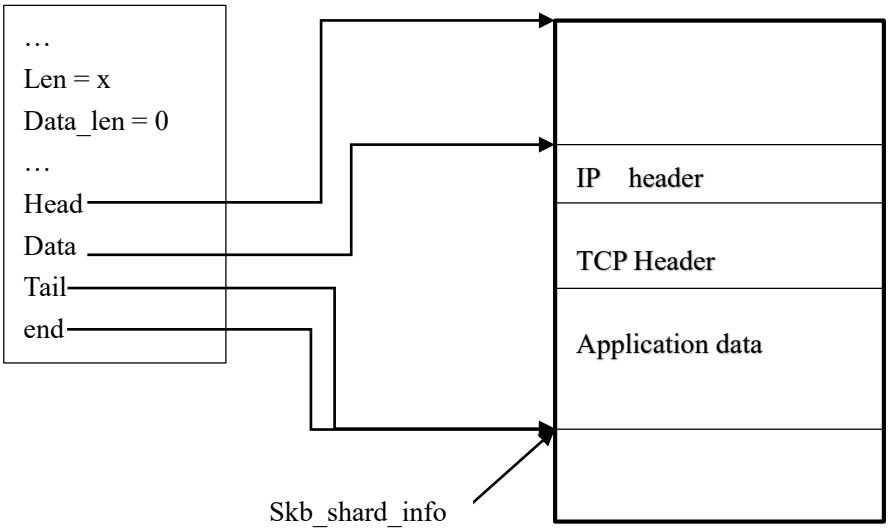
c. 填入应用层数据



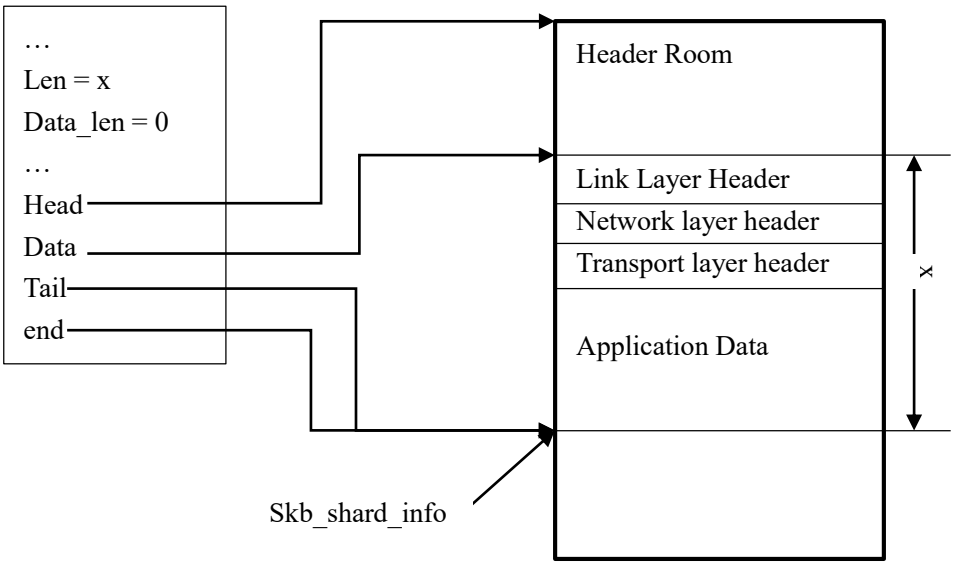
d. 填充 TCP 头部



e.填充 IP 头部



f. 填充链路层头部



SKB 克隆和 SKB 拷贝

1. 当只需要修改 SKB 描述符中的某些字段比如 h,nh 等时，就值需要复制 SKB 描述符，不需要复制数据缓存区，这就用到 skb 克隆操作。Skb_clone 只会复制 SKB 描述符，然后增加数据缓存区的引用计数。一个使用场景就是：一个接受到的包要传给多个接收者，这样原来的 sk_buff 和克隆的 sk_buff 描述符的 cloned 都置为 1，同时将数据缓存区引用计数 dataref 增加 1，因为又增加了一个 sk_buff 描述符指向它。

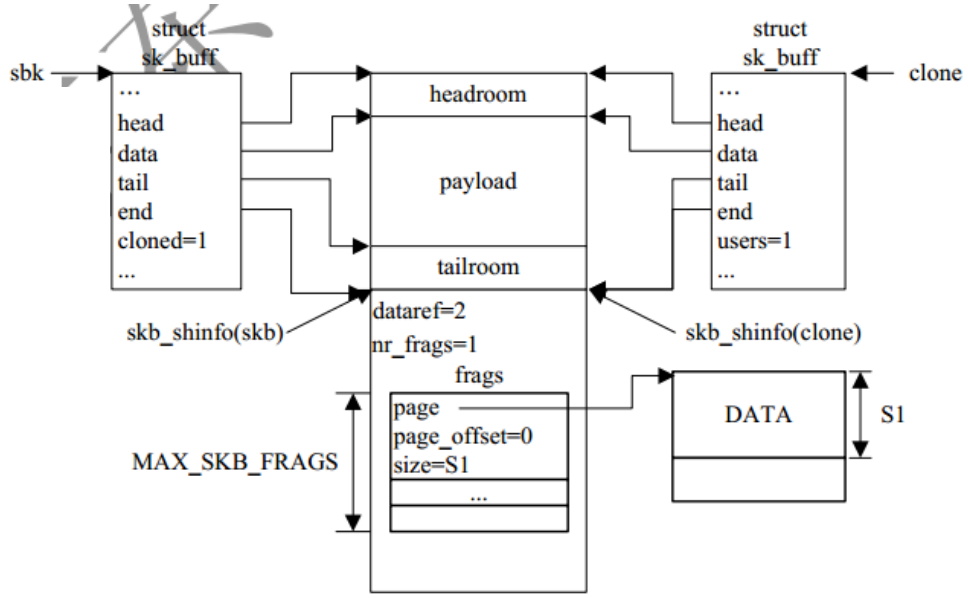


图3-20　克隆后的SKB

2. 如果不仅需要修改 SKB 描述符，还需要修改数据缓存区的数据，那么需要用到 skb 拷贝。根据数据所处的位置不同，又分为 pskb_copy()和 skb_copy()。如果要修改的数据在 skb->head 和 skb->end 之间，那么可以用 pskb_copy();如果要修改的数据在聚合分散 I/O 存储区，那么要用 skb_copy()。

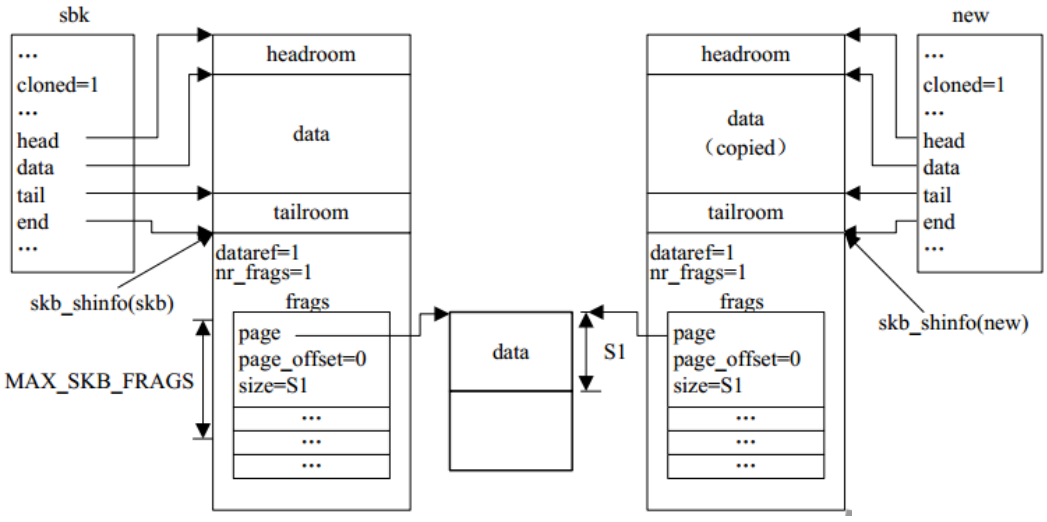


图3-21　pskb_copy()示意图

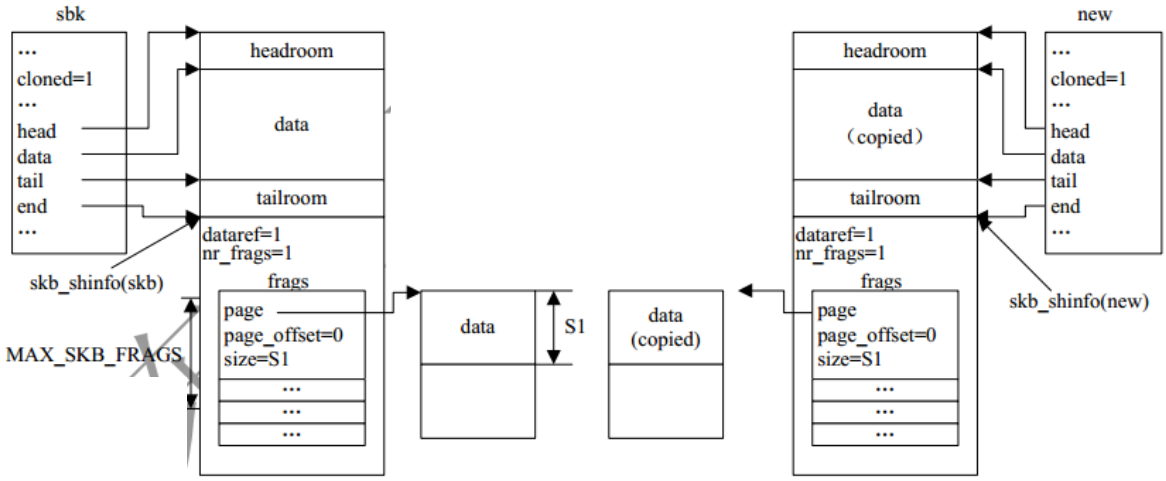


图3-22　　skb_copy()示意图