Tcp_sendmsg 主要负责把数据从用户空间拷贝到内核空间，然后根据 MSS 分段，放入到 SKB 中，调用 tcp_push 发送出去。大部分代码都是在找应该把数据复制到哪里去。

```
int tcp_sendmsg(struct kiocb *iocb, struct sock *sk, struct msghdr *msg,
            size_t size)
{
        struct iovec *iov;
        struct tcp_sock *tp = tcp_sk(sk);
        struct sk_buff *skb;
        int iovlen, flags;
        int mss_now, size_goal;
        int err, copied;
        long timeo;

        lock_sock(sk);
        TCP_CHECK_TIMER(sk);

        flags = msg->msg_flags;
        timeo = sock_sndtimeo(sk, flags & MSG_DONTWAIT); /*  如果 send_msg 是阻塞操作的
话，获取阻塞的时间  */

        /* Wait for a connection to finish. */
        if ((1 << sk->sk_state) & ~(TCPF_ESTABLISHED | TCPF_CLOSE_WAIT))/*  发送用户数据应该
处于 ESTABLISHED 状态或者是 CLOSE_WAIT 状态，  如果不在这两种状态则调用
sk_stream_wait_connnect  等连接建立完成,如果超时的话就跳转到 out_err*/
                if ((err = sk_stream_wait_connect(sk, &timeo)) != 0)
                        goto out_err;

        /* This should be in poll */
        clear_bit(SOCK_ASYNC_NOSPACE, &sk->sk_socket->flags);
/*  获取当前的 MSS，并将 MSG_OOB 清零，因为 OOB 带外数据不支持 GSO*/
        mss_now = tcp_current_mss(sk, !(flags&MSG_OOB));

/*  获取 SKB 的最大长度，这个表示 SKB 到达网络设备上时的最大长度*/
        size_goal = tp->xmit_size_goal;

        /* Ok commence sending. */
        iovlen = msg->msg_iovlen; /*  待发数据块的块数*/
        iov = msg->msg_iov;   /*待发数据指针，起始地址*/
        copied = 0; /* copied 表示有多少个数据块已经从用户空间复制到内核空间，先清零*/

        err = -EPIPE;/*  先把错误谁-EPIPE，EPIPE 表示本地已经关闭 socket 连接了*/
        if (sk->sk_err || (sk->sk_shutdown & SEND_SHUTDOWN)) /*  如果本地 socket 有错误或者
不允许发送数据，那么跳到 do_error 去处理*/
                goto do_error;
```

```
    while (--iovlen >= 0) {/*  如果还有待拷贝的数据块，这个循环用于控制拷贝所有的用户数据块到内核空间*/

        int seglen = iov->iov_len;   /*当前要复制的这个数据块的长度*/
        unsigned char __user *from = iov->iov_base; /*当前要复制的这个数据块的起始地址*/

        iov++; /*指向下一个数据块*/

        while (seglen > 0) {/*这个数据块是不是全部都拷贝完了，用于控制每一个数据块的拷贝*/
            int copy;

            skb = sk->sk_write_queue.prev;/*发送队列的最末尾的一个 skb, sk_write_queue 指向发送队列的头结点，发送队列是一个双向环链表，所以这里是链表的尾节点  */

            if (!sk->sk_send_head ||
                (copy = size_goal - skb->len) <= 0) { /*  如果 sk_send_head == NULL  表示所有发送队列上的 SKB 都已经发送过了，
                /*  或者最后一个 SKB 的长度已经到达 SKB 的最大长度了，
                /*  说明不能再往这个 SKB 上添加数据了，需要分配一个新的 SKB */
new_segment:
                /* Allocate new segment. If the interface is SG,
                 * allocate skb fitting to single page.
                 */
                if (!sk_stream_memory_free(sk)) /*  判断 sk->sk_wmem_queued  是否小于 sk->sk_sndbuf,  即发送队列中段数据的总长度是否小于发送缓冲区的大小  */
                    goto wait_for_sndbuf; /*  如果已经超过了，说明发送缓冲不够用了，  那么跳转到 wait_for_sndbuf 处理  */

                /*  分配 SKB */
                skb = sk_stream_alloc_pskb(sk, select_size(sk, tp),
                               0, sk->sk_allocation);

                /*  如果分配 SKB  失败，说明整个系统的内存不够用了，跳转到 wait_for_memory 处理  */
                if (!skb)
                    goto wait_for_memory;

                /*
                 * Check whether we can use HW checksum.
                 */
```

```
                    /*  看看网络设备能不能计算校验和  */
                    if (sk->sk_route_caps & NETIF_F_ALL_CSUM)
                        skb->ip_summed = CHECKSUM_PARTIAL;

                    /*  把这个 SKB 放到发送队列的尾部  */
                    skb_entail(sk, tp, skb);
                    copy = size_goal; /*对于新的 SKB,  可以拷贝的数据长度就等于 size_goal
*/
                }

                /* sk_send_head != NULL && (copy = size_goal - skb->len > 0),  表示这个 SKB 没
有发送过，并且还没到 size_goal 那么大，所以可以往最后一个 SKB 上添加数据  */
                /* Try to append data to the end of skb. */
                if (copy > seglen) /*  如果这个 SKB 剩余的空间大于这个数据块的大小，那么
把要拷贝的长度置为要拷贝的大小，copy = min(copy, seglen)*/
                    copy = seglen;

                /* Where to copy to? */
                /*  接下来确定拷贝到哪里去，看看是这个 SKB 的线性存储区还是聚合分散 IO
分段  */
                if (skb_tailroom(skb) > 0) {/*  看看这个 SKB 的线性存储区还有没有空间，如果
有的话，再看看剩余空间和要拷贝的数据大小的关系  */
                    /* We have some space in skb head. Superb! */
                    if (copy > skb_tailroom(skb))
                        copy = skb_tailroom(skb); /*  这就是最终这次要拷贝的数据长度了  */
                    if ((err = skb_add_data(skb, from, copy)) != 0) /*  把用户数据从 from 处拷贝
长度为 copy 的数据到这个 skb*/
                        goto do_fault; /*  如果拷贝过程出错，那跳到 do_fault 去处理*/
                } else    {
                /*  这个 SKB 的线性存储区已经没有空间了，那就要把数据复制到支持分散聚
合 I/O 的页中  */
                    int merge = 0;/*  用于标识是否在最后一个页中添加数据，先初始化为 0
*/
                    int i = skb_shinfo(skb)->nr_frags; /*获得这个 SKB 用了多少个分散的片段*/
                    struct page *page = TCP_PAGE(sk);    /*  获得上次用于拷贝的页面地
址,sk_sndmsg_page*/
                    int off = TCP_OFF(sk); /*已有数据在上一次用的页中的偏移*/

                    if (skb_can_coalesce(skb, i, page, off) &&
                        off != PAGE_SIZE) { /*看看能不能往最后一个页中追加数据，如果可
以的话 merge 赋值为 1*/
                        /* We can extend the last page
                         * fragment. */
                        merge = 1;
```

```
                } else if (i == MAX_SKB_FRAGS ||
                        (!i &&
                        !(sk->sk_route_caps & NETIF_F_SG))) { /*如果网络设备是不只是 SG 的或者分页片段已经达到上限了，那就不能再往这个 SKB 中添加数据了，而要分配新的 SKB*/
                        /* Need to add new fragment and cannot
                         * do this because interface is non-SG,
                         * or because all the page slots are
                         * busy. */
                        tcp_mark_push(tp, skb); /*设置 PUSH 标志，跟新 push_seq = write_seq，表示希望到 push_seq 的数据能尽快发送出去。*/
                        goto new_segment; /*所以跳回到 new_segment 处分配新的 SKB*/

                } else if (page) {
                        if (off == PAGE_SIZE) { /*  最后一个页的数据已经满了  */
                                put_page(page);
                                TCP_PAGE(sk) = page = NULL;
                                off = 0;
                        }
                } else /*  最后一种情况，不用分配新的 SKB,但是最后一个页也不能添加数据，所以要新开一个页，从这个页的起始处开始写数据，所以 off 要设为 0 */
                        off = 0;

                if (copy > PAGE_SIZE - off) //看看这个页还有多少剩余空间
                        copy = PAGE_SIZE - off;

                if (!sk_stream_wmem_schedule(sk, copy)) /*看看用于输出的缓存是否已经达到上限，如果已经达到的话，要等到有可用输出缓存或者超时  */
                        goto wait_for_memory;

                if (!page) {/*如果 page = NULL，一般是新开了一个 SKB 或者聚合分散 IO 的最后一个页已经用完了，那么要开辟一个新的页  */
                        /* Allocate new cache page. */
                        if (!(page = sk_stream_alloc_page(sk)))
                                goto wait_for_memory;
                }


                /*  终于分配好了内存，可以开始往页上复制数据了  */
                /* Time to copy data. We are close to
                 * the end! */
                err = skb_copy_to_page(sk, from, skb, page,
                                       off, copy);
                if (err) {
```

```
                        /* If this page was new, give it to the
                         * socket so it does not get leaked.
                         */
                        if (!TCP_PAGE(sk)) { /*  如果拷贝失败了，要记录下 sk_sndmsg_page = page, sk_sndmsg_off = 0,用以记录下来以备释放或者下一次拷贝时使用  */
                                TCP_PAGE(sk) = page;
                                TCP_OFF(sk) = 0;
                        }
                        goto do_error;
                }

                /* Update the skb. */
                if (merge) { /*  如果是在原来 SKB 的最后一个页中添加数据的话，需要更新这个页面的实际使用长度  */
                        skb_shinfo(skb)->frags[i - 1].size +=
                                        copy;
                } else { /*如果是将数据拷贝到一个新的页中*/
                        skb_fill_page_desc(skb, i, page, off, copy); /*  那么就要更新这个页的信息*/

                        if (TCP_PAGE(sk)) { /*  如果 sk_sndmsg_page != NULL, 表示用的是上次分配的页面，需要增加这个页的引用计数*/
                                get_page(page);
                        } else if (off + copy < PAGE_SIZE) { /*  否则 sk_sndmsg_page == NULL, 说明用的是最近新分配的页，并且这个页还没有用完*/
                                get_page(page);/*  那么不仅需要增加新的页的引用计数*/
                                TCP_PAGE(sk) = page; /*还需要修改 sk_sndmsg_page 为这个页，表示下次还可以接着用这个页*/
                        }
                }

                TCP_OFF(sk) = off + copy;
        } /*  完成了一次数据拷贝  */

        if (!copied) /*  如果没有拷贝数据，那么清空 PSH 标志  */
                TCP_SKB_CB(skb)->flags &= ~TCPCB_FLAG_PSH;

        tp->write_seq += copy; /*  更新发送队列中的最后一个序列号 write_seq */
        TCP_SKB_CB(skb)->end_seq += copy; /*  更新这个 SKB 的最后序列号，因为我们把往这个 SKB 中添加了新的数据  */
        skb_shinfo(skb)->gso_segs = 0;

        from += copy; /*  更新要复制的数据起始地址*/
        copied += copy; /*  更新已复制字节数的统计  */
```

```
                if ((seglen -= copy) == 0 && iovlen == 0) /*如果用户复制全部完了，那就跳到
out，跳出两层 while 循环*/
                        goto out;

                if (skb->len < mss_now || (flags & MSG_OOB)) /*如果这个 SKB 的数据长度小于
MSS，说明还可以往这个 SKB 中添加数据，那么就继续复制；如果是带外数据，也继续复
制数据*/
                        continue;

                if (forced_push(tp)) {/* 检查是否要马上发送数据，如果从上次 push 之后新增
加的数据已经超过了接收方窗口的一半，那就要马上发送数据*/
                        tcp_mark_push(tp, skb); /*给这个 SKB 打上 PUSH 标记 */
                        __tcp_push_pending_frames(sk, tp, mss_now, TCP_NAGLE_PUSH); /*调用
发包函数发包*/
                } else if (skb == sk->sk_send_head) /*如果数据没有那么多，但是以前
write_queue 上的数据都发送完了，那么也把这个 SKB 发送出去*/
                        tcp_push_one(sk, mss_now);
                continue;/* 否则就继续复制数据*/

wait_for_sndbuf: /* 发送队列中 SKB 的数据总长度达到了发送缓冲区的上限*/
                set_bit(SOCK_NOSPACE, &sk->sk_socket->flags);
wait_for_memory: /*整个系统的内存不够用了*/
                if (copied) /* 虽然分配 SKB 失败了，但是之前有复制一些用户数据，那么就
先把这些发送出去，并且去掉 MSG_MORE 标志，表示此次发送没有后续的数据了*/
                        tcp_push(sk, tp, flags & ~MSG_MORE, mss_now, TCP_NAGLE_PUSH);

                if ((err = sk_stream_wait_memory(sk, &timeo)) != 0) /*等待内存超时了，跳到
do_error 处理*/
                        goto do_error;

                /*还未超时就等到了可用内存空间，有可能 MSS 发生了变化，所以重新获取
MSS 和 size_goal,继续复制数据*/
                mss_now = tcp_current_mss(sk, !(flags&MSG_OOB));
                size_goal = tp->xmit_size_goal;
            }
        }

/*正常情况下，数据都复制完了，如果有复制数据，那就把这些数据都发送出去*/
out:
        if (copied)
            tcp_push(sk, tp, flags, mss_now, tp->nonagle);
        TCP_CHECK_TIMER(sk);
        release_sock(sk);
```

```
        return copied; /*返回从用户空间拷贝了多少数据到内核空间*/

do_fault:
        if (!skb->len) { /*如果 SKB 的长度为 0，说明这个 SKB 是新分配的*/
                if (sk->sk_send_head == skb)
                        sk->sk_send_head = NULL;
                __skb_unlink(skb, &sk->sk_write_queue); /*把这个 SKB 从发送队列中删除*/
                sk_stream_free_skb(sk, skb); /*  释放这个 SKB */
        }

do_error:
        if (copied) /*  如果已经复制了部分数据，还是要把这部分数据发送出去*/
                goto out;

out_err: /*  完全没有复制任何数据，那只能返回错误码给用户了  */
        err = sk_stream_error(sk, flags, err);
        TCP_CHECK_TIMER(sk);
        release_sock(sk);
        return err;
}
```