网络传输机制实验三

孙佳钰 2015K8009929051

2018 年 12 月 30 日

1 实验内容

TCP 协议是传输层中面向连接的通用协议。本次实验需要实现部分 TCP 协议，内容有：

- 实现网络有丢包下的超时重传机制。

2 实验流程

由于代码太多，故报告中没有加入完整代码，完整代码可见附件。

2.1 增加数据结构

需要在 *tcp_sock* 数据结构中增加发送队列的链表头、接收队列的链表头和计时器等部分。

```
struct tcp_sock {
  ...
  struct tcp_timer retrans_timer;
  int retrans_num;
  struct list_head snd_buffer;
  struct list_head rcv_ofo_buffer;
  ...
};
```

其中发送队列和接收队列定义在 *ring_buffer.h* 中。

```
struct rcv_win {
  struct list_head list;
  char *packet;
  int len;
  int seq_num;
};
```

```
7
8  struct snd_buf {
9    struct list_head list;
10   char *packet;
11   int seq_end;
12 };
```

2.2 发送队列与接收队列

1. 发送队列：发送一个带有数据或是带有 SYN|FIN 标志的数据包，都要暂存入发送队列中，待收到对应的 ACK 数据包时从发送队列中清除。

```
1  void tcp_send_packet(struct tcp_sock *tsk, char *packet, int len)
2  {
3    ...
4    struct snd_buf sndb;
5    sndb.seq_end = tsk->snd_nxt - 1;
6    sndb.packet = (char*)malloc(tcp_data_len * sizeof(char));
7    memcpy(sndb.packet, packet + len - tcp_data_len, tcp_data_len);
8    list_add_tail(&sndb.list, &tsk->snd_buffer);
9    ...
10 }
11
12 void tcp_send_control_packet(struct tcp_sock *tsk, u8 flags)
13 {
14   ...
15   if (flags & (TCP_SYN|TCP_FIN)) {
16     struct snd_buf sndb;
17     sndb.seq_end = tsk->snd_nxt - 1;
18     sndb.packet = NULL;
19     list_add_tail(&sndb.list, &tsk->snd_buffer);
20   }
21   ...
22 }
23
24 void tcp_process(struct tcp_sock *tsk, struct tcp_cb *cb, char *packet)
25 {
26   ...
27   case TCP_ACK:
```

```
    list_for_each_entry_safe(sndb, q, &tsk->snd_buffer, list) {
      if (sndb->seq_end <= cb->ack) {
        if (sndb->packet)
          free(sndb->packet);
        list_delete_entry(&sndb->list);
      }
    }
    update_timer(tsk);
    ...
    break;
  ...
}
```

2. 接收队列：在收到不连续的数据包时，暂存至接收队列中。每收到一个连续的数据包，都遍历一次接收队列，如果是下一个连续的数据包，则从接收队列中复制到环形缓冲区中，然后从接受队列中删掉。这种写法并不会完全利用接收队列的接收能力，是一种偷懒的写法。

```
void tcp_process(struct tcp_sock *tsk, struct tcp_cb *cb, char *packet)
{
  ...
  case TCP_PSH | TCP_ACK:
    if (cb->seq != tsk->rcv_nxt) {
      struct rcv_win rcvw;
      rcvw.packet = (char*)malloc(cb->pl_len * sizeof(char));
      memcpy(rcvw.packet, cb->payload, cb->pl_len);
      rcvw.len = cb->pl_len;
      rcvw.seq_num = cb->seq;
      list_add_tail(&rcvw.list, &tsk->rcv_ofo_buffer);
    } else if (TCP_ESTABLISHED == tsk->state) {
      pthread_mutex_lock(&tsk->rcv_buf->lock);
      write_ring_buffer(tsk->rcv_buf, cb->payload, cb->pl_len);
      pthread_mutex_unlock(&tsk->rcv_buf->lock);
      tsk->rcv_wnd = ring_buffer_free(tsk->rcv_buf);
      tcp_send_control_packet(tsk, TCP_ACK);
      struct rcv_win *rcvw, *q;
      list_for_each_entry_safe(rcvw, q, &tsk->rcv_ofo_buffer, list) {
        if (rcvw->seq_num == tsk->rcv_nxt) {
          pthread_mutex_lock(&tsk->rcv_buf->lock);
          write_ring_buffer(tsk->rcv_buf, rcvw->packet, rcvw->len);
```

```
            pthread_mutex_unlock(&tsk->rcv_buf->lock);
            tsk->rcv_wnd = ring_buffer_free(tsk->rcv_buf);
          }
          free(rcvw->packet);
          list_delete_entry(&rcvw->list);
        }
      }
      break;
    ...
}
```

2.3 超时重传线程

利用已有的 *tcp_scan_timer_list* 函数，在其中加入判断定时器类型与对应超时重传类型的处理流程。

```
void tcp_scan_timer_list()
{
  ...
  if (t->timeout <= 0) {
    if (TIME_WAIT == t->type) {
      ...
    } else {
      tsk = retrans_to_tcp_sock(t);
      if (!list_empty(&tsk->snd_buffer)) {
        if (3 == tsk->retrans_num) {
          tcp_sock_close(tsk);
          break;
        } else {
          struct snd_buf *sndb = list_entry( \
              tsk->snd_buffer.next, struct snd_buf, list);
          int len = sizeof(sndb->packet);
          tcp_sock_write(tsk, sndb->packet, len);
          update_timer(tsk);
          tsk->retrans_num++;
        }
      }
    }
  }
```

```
24     …
25 }
```

3 实验结果及分析

这次实验我并没有完全做完，只是根据我理解的部分大体实现了一下。比上次实验所加的代码基本都在报告中了，提交的代码在无丢包网络下可输出正常结果，在有丢包网络下无法正常运行。由于考研后紧接着就是毕设的事情，下次实验甚至可能没时间做，还望老师原谅…