网络传输机制实验一

孙佳钰 2015K8009929051

2018 年 12 月 13 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：

- 根据收到的 TCP 数据包处理相应端口的状态转换。
- 在服务器端实现端口的监听和收到同步请求数据包的端口的处理，在客户端实现与服务器端的连接。
- 实现服务器端与客户端共用的连接关闭函数。

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 状态转换处理

对照课件中的状态转换图，对于收到的每个 TCP 包，根据其各功能位将对应端口数据结构的状态进行转换。要注意对于处于监听状态的端口，应该 fork 出一个子端口来进行应答。

数据包到达该函数之前还调用了 *tcp_sock_lookup*，其两个子函数实现过程中要注意查询标准不同。

```
1 tsk->snd_una = cb->ack;
2 tsk->rcv_nxt = cb->seq_end;
3 switch (cb->flags) {
4   case TCP_RST:
5     tcp_set_state(tsk, TCP_CLOSED);
6     tcp_unhash(tsk);
7   case TCP_SYN:
8     if (TCP_LISTEN == tsk->state) {
9       struct tcp_sock *child_sock = alloc_tcp_sock();
```

```
10        ……
11        tcp_sock_bind(child_sock, &skaddr);
12        tcp_set_state(child_sock, TCP_SYN_RECV);
13        list_add_tail(&child_sock->list, &tsk->listen_queue);
14
15        tcp_send_control_packet(child_sock, TCP_SYN|TCP_ACK);
16        tcp_hash(child_sock);
17      }
18      break;
19    case TCP_SYN | TCP_ACK:
20      if (TCP_SYN_SENT == tsk->state) {
21        wake_up(tsk->wait_connect);
22      }
23      break;
24    case TCP_ACK:
25      if (TCP_SYN_RECV == tsk->state) {
26        list_delete_entry(&tsk->list);
27        tcp_sock_accept_enqueue(tsk);
28        tcp_set_state(tsk, TCP_ESTABLISHED);
29        wake_up(tsk->parent->wait_accept);
30      } else if (TCP_FIN_WAIT_1 == tsk->state) {
31        tcp_set_state(tsk, TCP_FIN_WAIT_2);
32      } else if (TCP_LAST_ACK == tsk->state) {
33        tcp_set_state(tsk, TCP_CLOSED);
34        if (!tsk->parent)
35          tcp_bind_unhash(tsk);
36        tcp_unhash(tsk);
37      }
38      break;
39    case TCP_FIN | TCP_ACK:
40      if (TCP_ESTABLISHED == tsk->state) {
41        tcp_send_control_packet(tsk, TCP_ACK);
42        tcp_set_state(tsk, TCP_CLOSE_WAIT);
43      } else if (TCP_FIN_WAIT_2 == tsk->state || TCP_FIN_WAIT_1 == tsk->state)
44        tcp_send_control_packet(tsk, TCP_ACK);
45        tcp_set_state(tsk, TCP_TIME_WAIT);
46      }
47  }
```

2.2 服务器端与客户端的连接处理

1. 在服务器端，对于监听函数，只需设置端口的最大等待应答数和状态，并将端口哈希到监听表中即可。对于接收函数，若该端口等待队列为空则阻塞，等收到该端口的同步请求数据包时将其唤醒并接受。

```
int tcp_sock_listen(struct tcp_sock *tsk, int backlog)
{
  // fprintf(stdout, "TODO: implement %s please.\n", __FUNCTION__);
  tsk->backlog = backlog;
  tcp_set_state(tsk, TCP_LISTEN);
  if (tcp_hash(tsk) < 0)
    return -1;

  return 0;
}

struct tcp_sock *tcp_sock_accept(struct tcp_sock *tsk)
{
  // fprintf(stdout, "TODO: implement %s please.\n", __FUNCTION__);
  if (list_empty(&tsk->accept_queue)) {
    sleep_on(tsk->wait_accept);
  }

  struct tcp_sock *csk = tcp_sock_accept_dequeue(tsk);
  return csk;
}
```

2. 在客户端，将端口结构设置好后发送连接请求数据包，然后阻塞；等到收到确认数据包后被唤醒，即成功连接。

```
int tcp_sock_connect(struct tcp_sock *tsk, struct sock_addr *skaddr)
{
  // fprintf(stdout, "TODO: implement %s please.\n", __FUNCTION__);
  ……
  if (tcp_sock_set_sport(tsk, 0) < 0)
  return -1;

  tcp_send_control_packet(tsk, TCP_SYN);
  tcp_set_state(tsk, TCP_SYN_SENT);
  tcp_hash(tsk);
```

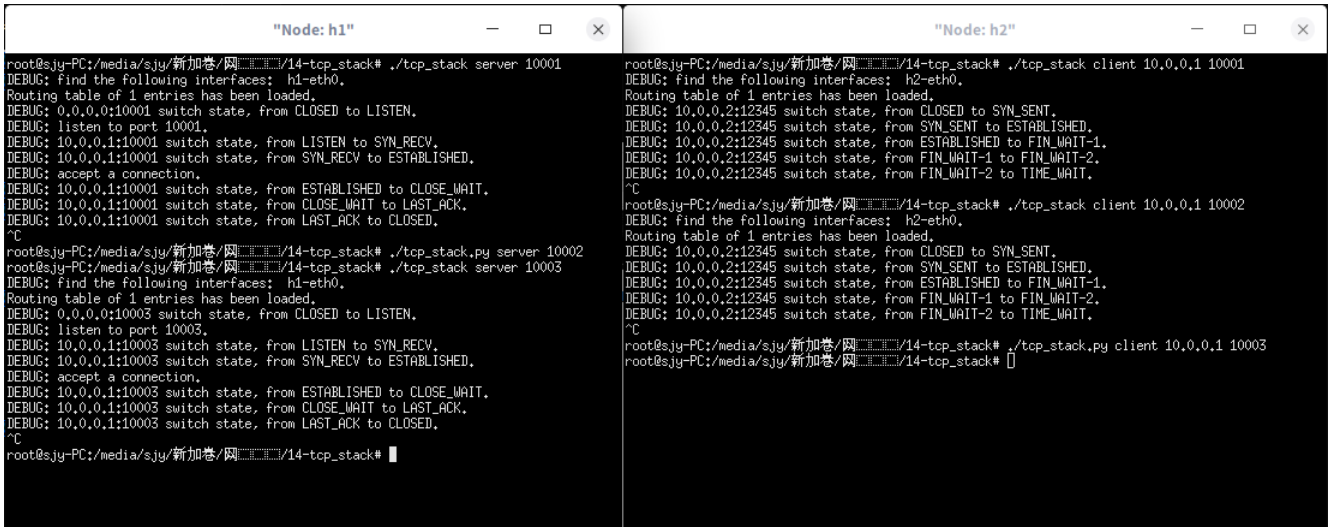
```
11    sleep_on(tsk->wait_connect);
12
13    tcp_send_control_packet(tsk, TCP_ACK);
14    tcp_set_state(tsk, TCP_ESTABLISHED);
15    tcp_hash(tsk);
16
17    return 0;
18  }
```

2.3 服务器端与客户端的关闭处理

由于该函数为客户端和服务器的共用函数，所以要根据端口是否有父端口来判断其是服务器还是客户端，如果是服务器则需先等待服务器发送结束请求。

```
1  void tcp_sock_close(struct tcp_sock *tsk)
2  {
3    // fprintf(stdout, "TODO: implement %s please.\n", __FUNCTION__);
4    if (tsk->parent)
5      while (TCP_CLOSE_WAIT != tsk->state);
6    if (TCP_ESTABLISHED == tsk->state) {
7      tcp_send_control_packet(tsk, TCP_FIN|TCP_ACK);
8      tcp_set_state(tsk, TCP_FIN_WAIT_1);
9    } else if (TCP_CLOSE_WAIT == tsk->state) {
10     tcp_send_control_packet(tsk, TCP_FIN|TCP_ACK);
11     tcp_set_state(tsk, TCP_LAST_ACK);
12   }
13 }
```

3 实验结果及分析



可以看出结果是正确的。