

14-TCP_stack 实验报告

作者：苗屹松

时间：2018年1月14日

1. Overview

1月11日调通了bug，但是忙于做实验二，今天才写实验报告

git的commit历史在这里：https://gitlab.com/miaoyisong/network_lab/tree/master/14-tcp_stack

本次实验我完成了TCP的连接协议，主要分为两部分：TCP连接的创建 与 TCP连接的释放

2. 实验结果

我圆满完成了实验~和老师的reference结果一致，结果如图：

The screenshot displays two windows. The top window, titled 'XTerm', shows the execution of a TCP stack server on 'Node: h1'. The output includes debug messages for listening on port 10001, accepting connections, and handling SYN, ACK, and FIN packets. The bottom window, titled 'Wireshark', shows a packet capture from 'h1-eth0'. The capture includes ARP requests and responses, and a series of TCP packets (SYN, ACK, FIN) between 10.0.0.1 and 10.0.0.2, demonstrating the connection establishment and termination process.

```
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
----Under tcp sock accept
----Under tcp sock lookup_established
----Under tcp sock lookup_listen
lookup listen success
----Under tcp state listen
DEBUG: 0.0.0.0:10001 switch state, from LISTEN to SYN_RECV.
Hashed child tsk!
*****Sent a TCP packet: TCP_SYN|TCP_ACK
----Under tcp sock lookup_established
lookup establish success
TODO: implement this function please syn_rcv
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
parent exist
accept Force Awaken
DEBUG: accept a connection.
----Under tcp sock lookup_established
lookup establish success
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
----Under tcp sock lookup_established
lookup establish success
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
^C
root@ubuntu:/mnt/hgfs/networking/14-tcp_stack# ./tcp_stack server 10001
```

Routing table of 1 entries has been loaded.
----Under tcp sock_connect
Doing tcp sock connect
Trying to send
DEBUG: 10.0.0.2:12346 switch state, from CLOSED to SYN_SENT.
control packet sent
State set
Hashed!!!
----Under tcp sock lookup_established
lookup establish success
----Under tcp state syn_sent
Receive TCP_SYN|TCP_ACK
*****Sent a TCP packet: TCP_ACK
DEBUG: 10.0.0.2:12346 switch state, from SYN_SENT to ESTABLISHED.
The Force Awaken
Trying to close, current stateESTABLISHED
DEBUG: 10.0.0.2:12346 switch state, from ESTABLISHED to FIN_WAIT-1.
Switched state already
----Under tcp sock lookup_established
lookup establish success
DEBUG: 10.0.0.2:12346 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
----Under tcp sock lookup_established
lookup establish success
DEBUG: 10.0.0.2:12346 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12346 switch state, from TIME_WAIT to CLOSED.
^C
root@ubuntu:/mnt/hgfs/networking/14-tcp_stack#

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	66:d5:cc:51:34:3e	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
2	0.000079000	5a:1e:03:06:f1:89	66:d5:cc:51:34:3e	ARP	42	10.0.0.1 is at 5a:1e:03:06:f1:89
3	0.002204000	10.0.0.2	10.0.0.1	TCP	54	12346-10001 [SYN] Seq=0 Win=65535 Len=0
4	0.002598000	10.0.0.1	10.0.0.2	TCP	54	10001-12346 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
5	0.006112000	10.0.0.2	10.0.0.1	TCP	54	12346-10001 [ACK] Seq=1 Ack=1 Win=65535 Len=0
6	1.007133000	10.0.0.2	10.0.0.1	TCP	54	12346-10001 [FIN, ACK] Seq=1 Ack=1 Win=65535 Len=0
7	1.007360000	10.0.0.1	10.0.0.2	TCP	54	10001-12346 [ACK] Seq=1 Ack=2 Win=65535 Len=0
8	2.006707000	10.0.0.1	10.0.0.2	TCP	54	10001-12346 [FIN, ACK] Seq=1 Ack=2 Win=65535 Len=0
9	2.006857000	10.0.0.2	10.0.0.1	TCP	54	12346-10001 [ACK] Seq=2 Ack=2 Win=65535 Len=0

3. 实验原理

本次实验的核心就是socket状态的转化，在转化的过程中要收发各种数据包，要管理hash_table, 还要对一些线程sleep_on和wake_up.

具体的转化过程比较复杂，用图文比较难说明，我在如下视频中有较为清晰的说明：<https://youtu.be/4Bn-ZWnPxCY>

4.关键函数与思想

4.1 Hash的思想

实验里最重要的数据结构就是 tcp_sock, 里面包含了这个socket所有的控制信息。

过程中，要收发各种的包，那么是谁来收发这些包呢？对，就是tcp_sock

那么我们如何找到我们需要的tcp_sock呢？就需要用hash

```
struct tcp_sock *tsk = tcp_sock_lookup(&cb);
```

老师的这段源码就告诉我：通过收到的包里的关键信息，就可以找到对应的socket。

我们分别在 tcp_sock_connect 和 tcp_sock_listen 把我们socket放入 tcp_established_sock_table 和 tcp_listen_sock_table 中。

然后每次在tcp_process时，通过lookup函数找到需要的socket:

```
struct tcp_sock *tcp_sock_lookup_established(u32 saddr, u32 daddr, u16 sport,
u16 dport)
{
    //fprintf(stdout, "TODO: implement this function please.lookup
establish\n");
    printf("-----Under tcp_sock_lookup_established\n");
    struct list_head *list;
    int hash = tcp_hash_function(saddr, daddr, sport, dport);
    list = &tcp_established_sock_table[hash];
    struct tcp_sock *tmp;
    //printf("-----Desired hash entry is %d-----\n", hash);
    list_for_each_entry(tmp, list, hash_list) {
        if(tmp->sk_sip == saddr && tmp->sk_dip == daddr && tmp->sk_sport ==
sport && tmp->sk_dport){
            return tmp;
        }
    }
    return NULL;
}
```

4.2 Sleep_on & Wake_up

这是这次实验里我觉得最有意思的地方了～

当一个资源(比如socket)还没到位时，我们先让这个函数 `暂停` (sleep_on)在此处，等到资源到位时，再继续这个函数(wake_up)

比如此处：

```
struct tcp_sock *tcp_sock_accept(struct tcp_sock *tsk)
{
    if(list_empty(&(tsk->accept_queue))){
        sleep_on(tsk->wait_accept);
        struct tcp_sock * the_tcp_socket = tcp_sock_accept_dequeue(tsk);
        return the_tcp_socket;
    }
    ...
}
```

我们要accept一个tsk，但是此时资源还没到位，就先sleep_on

```
void tcp_state_syn_recv(struct tcp_sock *tsk, struct tcp_cb *cb, char *packet)
{
    ...
    tcp_sock_accept_enqueue(tsk);
    wake_up(tsk->parent->wait_accept);
}
```

在tcp_state_syn_recv函数中，当资源到位了，就wake_up，等待资源的函数就可以dequeue，得到需要的资源。

在这里，老师已经写好了非常好用的数据结构和接口，我们只需要直接拿来用即可～

5. 实验总结

这次实验算是比较复杂的一次，原因是 `支线` 比较多，要管理hash_table, tsk的队列，还要管理状态的转换、包的收发，一开始很没有头绪，多亏了老师的注释，我才慢慢理清了思路。

本次实验我在一个地方卡了很久：客户端有时在某处报错，而有时又可以正常执行。我感到非常奇怪，甚至怀疑是不是编译器出了问题。

后来经过老师的提醒，我在客户端的一个函数的 `状态转换` 放在了 `发送相应数据包` 之后，这就导致了：当对端(服务器)返回了数据包时， `状态转换` 还不一定已完成，这就导致了报错。

另外一个收获是，当有bug思路不清时，不妨拿笔在纸上画一遍状态转换，很快就理清思路了！