14-TCP_stack 实验报告

作者：苗屹松

时间：2018年1月14日

1. Overview

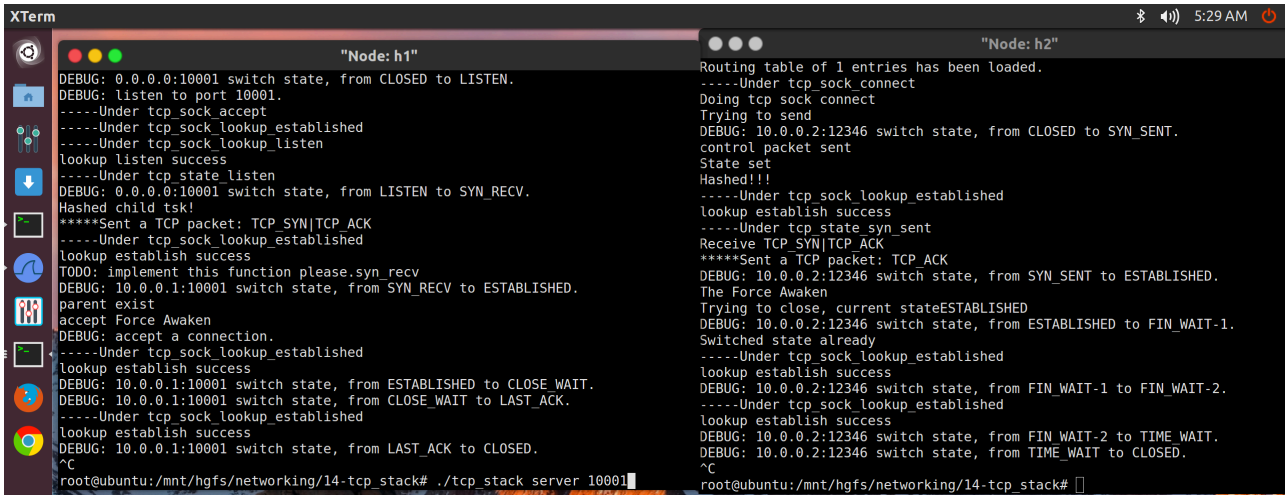
1月11日调通了bug，但是忙于做实验二，今天才写实验报告

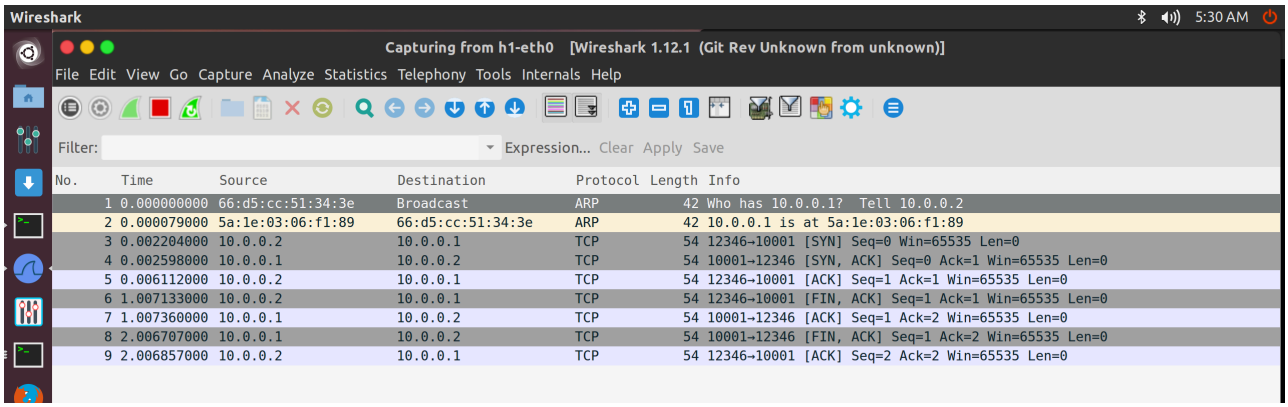
git的commit历史在这里：https://gitlab.com/miaoyisong/network_lab/tree/master/14-tcp_stack

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：





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！