

# 网络传输机制实验一

范子墨

2021K8009929006

## 一、 实验内容

### 1、 实验内容一：连接管理

- a) 运行给定网络拓扑(tcp\_topo.py)
- b) 在节点 h1 上执行 TCP 程序
  - i. 执行脚本(disable\_tcp\_rst.sh, disable\_offloading.sh), 禁止协议栈的相应功能
  - ii. 在 h1 上运行 TCP 协议栈的服务器模式 (./tcp\_stack server 10001)
- c) 在节点 h2 上执行 TCP 程序
  - i. 执行脚本(disable\_tcp\_rst.sh, disable\_offloading.sh), 禁止协议栈的相应功能
  - ii. 在 h2 上运行 TCP 协议栈的客户端模式, 连接至 h1, 显示建立连接成功后自动断开连接 (./tcp\_stack client 10.0.0.1 10001)
- d) 可以在一端用 tcp\_stack\_conn.py 替换 tcp\_stack 执行, 测试另一端
- e) 通过 wireshark 抓包来验证建立和断开连接的正确性

### 2、 实验内容二：短消息收发

- a) 参照 tcp\_stack\_trans.py, 修改 tcp\_apps.c, 使之能够收发短消息
- b) 运行给定网络拓扑(tcp\_topo.py)
- c) 在节点 h1 上执行 TCP 程序
  - i. 执行脚本(disable\_offloading.sh, disable\_tcp\_rst.sh)
  - ii. 在 h1 上运行 TCP 协议栈的服务器模式 (./tcp\_stack server 10001)
- d) 在节点 h2 上执行 TCP 程序
  - i. 执行脚本(disable\_offloading.sh, disable\_tcp\_rst.sh)
  - ii. 在 h2 上运行 TCP 协议栈的客户端模式, 连接 h1 并正确收发数据 (./tcp\_stack client 10.0.0.1 10001)
    1. client 向 server 发送数据, server 将数据 echo 给 client
- e) 使用 tcp\_stack\_trans.py 替换其中任意一端, 对端都能正确收发数据

### 3、 实验内容三：大文件传送

- a) 修改 tcp\_apps.c(以及 tcp\_stack\_trans.py), 使之能够收发文件
- b) 执行 create\_randfile.sh, 生成待传输数据文件 client-input.dat
- c) 运行给定网络拓扑(tcp\_topo.py)
- d) 在节点 h1 上执行 TCP 程序
  - i. 执行脚本(disable\_offloading.sh, disable\_tcp\_rst.sh)
  - ii. 在 h1 上运行 TCP 协议栈的服务器模式 (./tcp\_stack server 10001)
- e) 在节点 h2 上执行 TCP 程序
  - i. 执行脚本(disable\_offloading.sh, disable\_tcp\_rst.sh)
  - ii. 在 h2 上运行 TCP 协议栈的客户端模式 (./tcp\_stack client 10.0.0.1 10001)
    1. Client 发送文件 client-input.dat 给 server, server 将收到的数据存储到文件 server-output.dat
- f) 使用 md5sum 比较两个文件是否完全相同
- g) 使用 tcp\_stack\_trans.py 替换其中任意一端, 对端都能正确收发数据

## 二、 实验内容一：连接管理

### 1、 总体代码思路

首先依据原有代码，在 main 函数中首先调用函数初始化用户态协议栈、路由表和 TCP 协议栈，在初始化用户态协议栈 init\_tcp\_stack 函数中，它创建了一个新的线程用于扫描并处理 TCP 定时器列表 (tcp\_scan\_timer\_list 函数)，这个函数实现了从 TIME\_WAIT 到 CLOSED 状态转换。而后调用 run\_application 函数，传递输入参数后根据创造参数创造新的线程，新线程调用 tcp\_server 或者 tcp\_client 函数。main 函数中最后调用 ustack\_run 函数监听网络接口上的数据包并最后调用 handle\_packet 函数处理，handle\_packet 可以跳到 handle\_ip\_packet，判断上层协议为 TCP 后，调用 handle\_tcp\_packet 进行处理。

在 tcp\_server 函数中，初始化 TCP 套接字，调用 tcp\_sock\_bind 函数将套接字的源端口设置为参数传入的端口号，调用 tcp\_sock\_listen 函数监听，使用 tcp\_sock\_accept 函数更改 tsk 状态并赋给 csk，接受连接，最后调用 tcp\_sock\_close 关闭连接。关闭连接。在 TCP\_client 函数中，分配 TCP 套接字，连接服务器，进行数据传输最后关闭连接。

在 tcp\_client 函数中，初始化套接字，而后调用 tcp\_sock\_connect 函数，建立连接请求，最后调用 tcp\_sock\_close 关闭连接。

对于 TCP 数据包的处理，在 handle\_tcp\_packet 中首先会初始化一个 tcp\_cb 的结构体，该结构体中包含了 TCP 报文的相关信息（如源地址源端口号、目的地址目的端口号等），而后使用 tcp\_sock\_lookup 函数查找与 tcp 报文相关联的套接字。最终调用 tcp\_process 函数处理 tcp 报文，tcp\_process 会根据报文内容和套接字当前状态进行回复，并进行状态转换。

## 2、具体实现

### (1) 信号量的睡眠和唤醒

这是实现线程同步最重要的一步，在套接字结构体定义中含有下列四个信号量。

```
struct synch_wait *wait_connect;
struct synch_wait *wait_accept;
struct synch_wait *wait_recv;
struct synch_wait *wait_send;
```

wait\_connect 用于在套接字建立连接之前进行同步，因此在 client 端发送连接请求后挂起 (CLOSED 状态变为 SYN\_SENT 状态)，在收到 ACK|SYN 包之后说明连接 client 端连接已成功建立 (SYN\_SENT 状态变为 ESTABLISHED 状态)，因此唤醒该线程。

wait\_accept 相对于 wait\_connect 来说用在 server 端，当无连接请求时被挂起（处在 SYN\_RECV 状态），当收到 client 端发送的 ACK 包的时候被唤醒（有 SYN\_RECV 转换到 ESTABLISHED 状态），此时建立连接的三次招手已经完成。

wait\_recv 在接收到的缓冲区为空时被挂起，当收到数据时唤醒。（在实验一中未涉及到）

wait\_send 在 server 端接收窗口大小不足时被挂起，收到 PSH|ACK 包时进行确认，判断窗口大小是否足够，若足够则唤醒。（在实验一中未涉及到）

### (2) tcp\_scan\_timer\_list:

该函数对每个定时器的时间进行递减，而后判断是否定时器时间已经归零，若归零，则说明已经超时，则将该定时器从定时器列表中删除，并将对应的套接字状态设置为关闭。

```
if (timer->timeout <= 0)
{
    list_delete_entry(&timer->list);
    tsk = timewait_to_tcp_sock(timer);
    if (!tsk->parent)
        tcp_bind_unhash(tsk);
    tcp_set_state(tsk, TCP_CLOSED);
}
```

```

        free_tcp_sock(tsk);
    }

```

(3) **tcp\_sock\_listen**: 设置该套接字能够排队等待连接的最大数量，并且将状态从 CLOSED 转换为 LISTEN。

(4) **tcp\_sock\_accept**: 如果等待连接的队列不空，则弹出第一个等待连接的请求并接受。若为空，则挂起该线程。

(5) **tcp\_sock\_lookup**

该函数中首先将提取 cb 结构体地址中的源端口源地址和目的端口目的地址，而后调用 **tcp\_sock\_lookup\_established** 找到与该结构体对应的套接字并返回。具体来说，首先计算哈希值，而后获取对应的哈希表，遍历链表寻找匹配的套接字。

```

int hash = tcp_hash_function(saddr, daddr, sport, dport);
struct list_head * list = &tcp_established_sock_table[hash];

```

如果寻找到匹配的套接字，则调用 **tcp\_lookup\_listen** 函数处理一个新的连接请求，只关注源端口号，在监听套接字哈希表中寻找。

(6) **tcp\_sock\_connect**

这是由 client 端主动发起的连接请求，向 server 端发送 SYN，而后进行等待。

```

tcp_send_control_packet(tsk, TCP_SYN);
tcp_set_state(tsk, TCP_SYN_SENT);
tcp_hash(tsk);
sleep_on(tsk->wait_connect);

```

(7) **tcp\_process**

这个函数中是一个状态机的书写。整体状态转换如下图所示，准确来说可以简略成下图。

在这其中，需要注意 server 端在 LISTEN 状态收到 AYN 时，需要建立一个新的子套接字，并将子套接字添加到父套接字的监听队列中，同时将子套接字添加到相应的哈希表中。

```

struct tcp_sock *child_sk = alloc_tcp_sock();
child_sk->parent = tsk;
child_sk->sk_sip = cb->daddr;
child_sk->sk_sport = cb->dport;
child_sk->sk_dip = cb->saddr;
child_sk->sk_dport = cb->sport;
child_sk->iss = tcp_new_iss();
child_sk->rcv_nxt = cb->seq + 1;
child_sk->snd_nxt = child_sk->iss;

list_add_tail(&child_sk->list, &tsk->listen_queue);
tcp_send_control_packet(child_sk, TCP_ACK | TCP_SYN);
tcp_set_state(child_sk, TCP_SYN_RECV);
tcp_hash(child_sk);

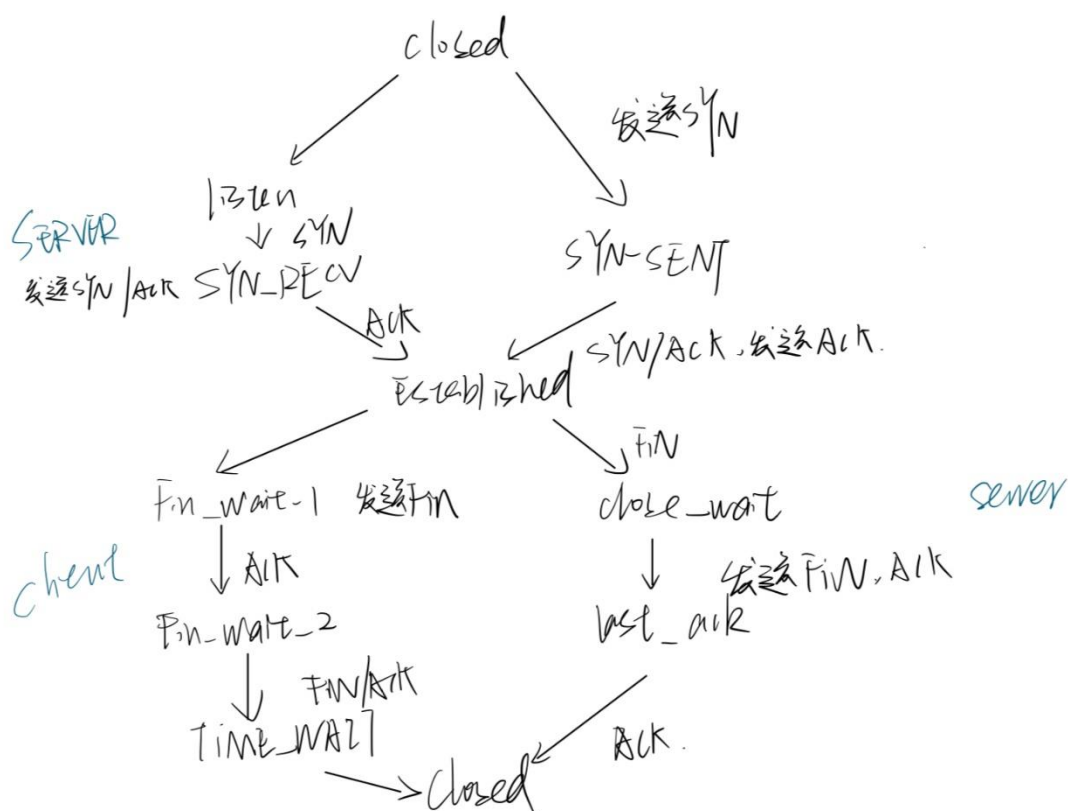
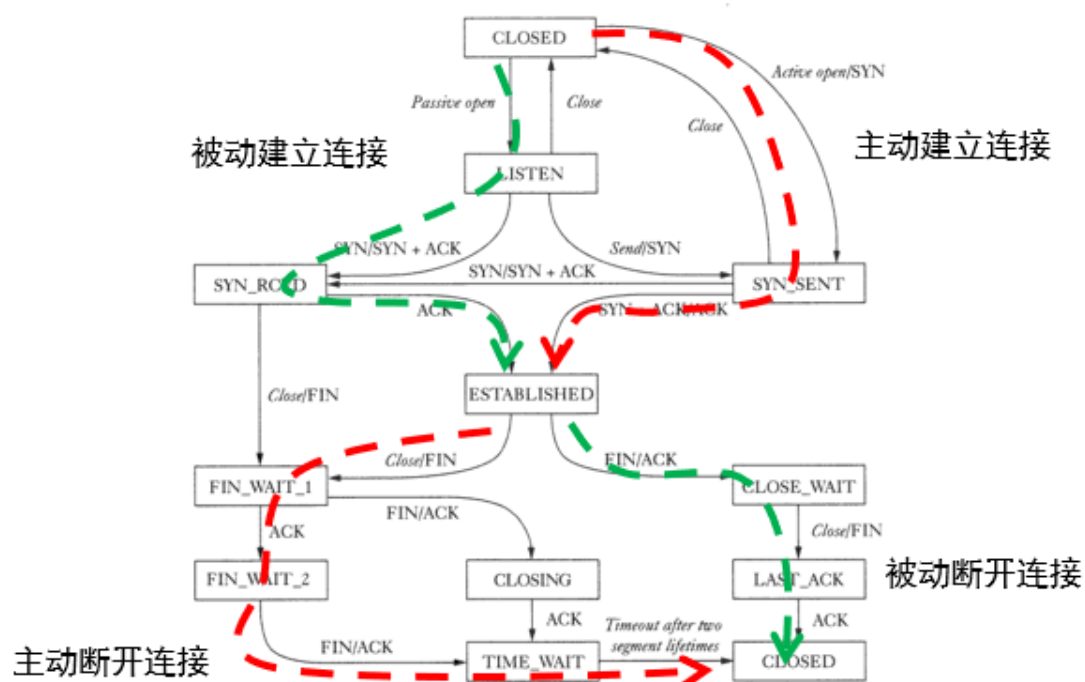
```

同时，注意每次更新下一个期望接收到的字节的序列号。信号量的唤醒按照前述的进行添加即可。此外，在 TCP\_FIN\_WAIT\_2 状态收到 FIN 包后，会进入 TCP\_TIME\_WAIT 函数，此时会设置时钟，在超时后自动转换为 CLOSED 状态。即在 **tcp\_set\_timewait\_timer** 函数中将该线程加入定时器列表。

```

tsk->timewait.timeout = time(NULL);
list_add_tail(&tsk->timewait.list, &timer_list);

```



### 3、实验结果

(1) 可以成功建立连接，并自动断开连接。

```
"Node: h1"
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# ./tcp_stack server 1000
1
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
send ack|syn
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
TCP_ESTABLISHED flags 1
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
server in established send ackDEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
[]

"Node: h2"
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# ./tcp_stack client 10.0
.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
client in established send fin
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
TCP_FIN_WAIT_1 flags 16
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
```

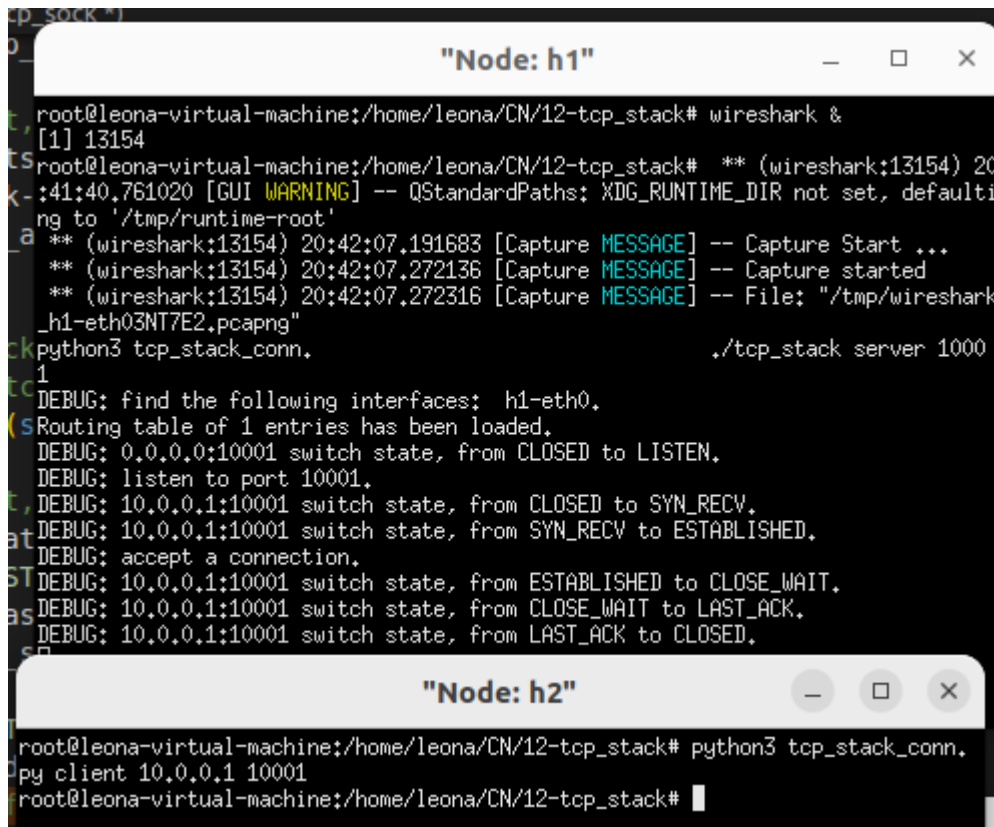
(2) 使用 wireshark 抓包结果如下

Time	Source	Destination	Protocol	Length	Info
1 0.000000000	f6:de:e8:00:72:cc	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
2 0.010392281	26:f9:35:18:5d:06	f6:de:e8:00:72:cc	ARP	42	10.0.0.1 is at 26:f9:35:18:5d:06
3 0.010434770	26:f9:35:18:5d:06	f6:de:e8:00:72:cc	ARP	42	10.0.0.1 is at 26:f9:35:18:5d:06
4 0.021600897	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [SYN] Seq=0 Win=65535 Len=0
5 0.032452710	26:f9:35:18:5d:06	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
6 0.043440269	f6:de:e8:00:72:cc	26:f9:35:18:5d:06	ARP	42	10.0.0.2 is at f6:de:e8:00:72:cc
7 0.043442188	f6:de:e8:00:72:cc	26:f9:35:18:5d:06	ARP	42	10.0.0.2 is at f6:de:e8:00:72:cc
8 0.053892286	10.0.0.1	10.0.0.2	TCP	54	10001 → 12345 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
9 0.075090158	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [ACK] Seq=1 Ack=1 Win=65535 Len=0
10 1.064686472	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [FIN] Seq=1 Win=65535 Len=0
11 1.076323787	10.0.0.1	10.0.0.2	TCP	54	10001 → 12345 [ACK] Seq=1 Ack=2 Win=65535 Len=0
12 5.099826427	10.0.0.1	10.0.0.2	TCP	54	10001 → 12345 [FIN] Seq=1 Win=65535 Len=0
13 5.110873465	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [ACK] Seq=2 Ack=2 Win=65535 Len=0

(3) 在一端用 tcp\_stack\_conn.py 替换 tcp\_stack 执行，测试另一端

```
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# python3 tcp_stack_conn.
py server 10001
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# a[]

"Node: h2"
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# ./tcp_stack client 10.0
.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
```



```
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# wireshark &
[1] 13154
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# ** (wireshark:13154) 20:41:40.761020 [GUI WARNING] -- QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
** (wireshark:13154) 20:42:07.191683 [Capture MESSAGE] -- Capture Start ...
** (wireshark:13154) 20:42:07.272136 [Capture MESSAGE] -- Capture started
** (wireshark:13154) 20:42:07.272316 [Capture MESSAGE] -- File: "/tmp/wireshark_h1-eth03NT7E2.pcapng"
python3 tcp_stack_conn. ./tcp_stack server 1000
1
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
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.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.

root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# python3 tcp_stack_conn.
py client 10.0.0.1 10001
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack#
```

### 三、 实验内容二：短消息收发

#### 1、 具体实现

相比于上个实验，这个实验中要可以收发短消息。总体来说，需要在上个实验的基础上添加状态处理逻辑和读写函数设计。

由老师给出的 tcp\_apps.c 文件中，在 server 端，使用 tcp\_sock\_read 从 client 端接收数据，通过 tcp\_sock\_write 向客户端发送数据。在 client 端，通过 tcp\_sock\_write 向 server 端发送数据，使用 tcp\_sock\_read 从服务器端接收响应数据。与此同时，在 tcp\_process 函数中添加处理 PSH|ACK 包的逻辑。

在 tcp\_sock\_read 函数中，检查缓冲区是否为空，若为空则挂起当前线程。如果收到 PSH|FIN 包，则说明有数据可读，会被唤醒，则调用 read\_ring\_buffer 函数读取数据，并将读取长度返回。

在 tcp\_sock\_write 函数中，将数据分割成适当大小的 TCP 进行发送。即每次对发送长度进行判断，尽量以 TCP 包最大容量发送，如果窗口大小不够，则当前进程挂起，在收到 ACK 包时进行判断后唤醒。

在 tcp\_process 函数中添加收到 PSH|ACK 数据包的处理逻辑，将收到的内容写入缓冲区，并恢复 ACK 包，同时由于出现缓冲区有数据可读，唤醒 tcp\_sock\_read 所在线程进行读取。

#### 2、 实验结果

(1) 节点上运行 tcp 程序并抓包



```
"Node: h2"                                     "Node: h1"
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack -2# ./tcp_stack client 1
0.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 23456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 3456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 56789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 6789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 89abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 9abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 0abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
[]

root@leona-virtual-machine:/home/leona/CN/12-tcp_stack -2# wireshark &
[1] 7328
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack -2# ** (wireshark:7328)
19:15:48.874908 [GUI WARNING] -- QStandardPaths: XDG_RUNTIME_DIR not set, default
ting to '/tmp/runtime-root'
** (wireshark:7328) 19:16:19.454368 [Capture MESSAGE] -- Capture Start ...
** (wireshark:7328) 19:16:19.957561 [Capture MESSAGE] -- Capture started
** (wireshark:7328) 19:16:19.957767 [Capture MESSAGE] -- File: "/tmp/wireshark_
h1-eth04RG4E2.pcapng"
./tcp_stack server 1
0001
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: tcp_sock_read return 0, finish transmission.
DEBUG: close this connection.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
```

(2) 使用 tcp\_stack\_trans.py 替换其中任意一端

```
"Node: h2"                                     "Node: h1"
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack -2# python3 tcp_stack_tr
ans.py client 10.0.0.1 10001
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 23456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 3456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 56789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 6789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 89abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 9abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 0abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack -2# []

root@leona-virtual-machine:/home/leona/CN/12-tcp_stack -2# wireshark &
[1] 7677
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack -2# ** (wireshark:7677)
19:24:43.423236 [GUI WARNING] -- QStandardPaths: XDG_RUNTIME_DIR not set, default
ting to '/tmp/runtime-root'
** (wireshark:7677) 19:25:09.743317 [Capture MESSAGE] -- Capture Start ...
** (wireshark:7677) 19:25:09.817166 [Capture MESSAGE] -- Capture started
** (wireshark:7677) 19:25:09.817346 [Capture MESSAGE] -- File: "/tmp/wireshark_
h1-eth0T9AE2.pcapng"
./tcp_stack client 1
server
0001
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: tcp_sock_read return 0, finish transmission.
DEBUG: close this connection.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
```

```
"Node: h2"                                     "Node: h1"
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack -2# ./tcp_stack client 1
0.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 23456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 3456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 56789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 6789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 89abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 9abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
server echoes: 0abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
[]

root@leona-virtual-machine:/home/leona/CN/12-tcp_stack -2# python3 tcp_stack_tr
ans.py server 10001
('10.0.0.2', 12345)
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack -2# []
```

## 四、 实验内容三：大文件传输

### 1、 实验结果

(1) 两端均使用自己的代码，可以看出成功传输且传输内容正确。

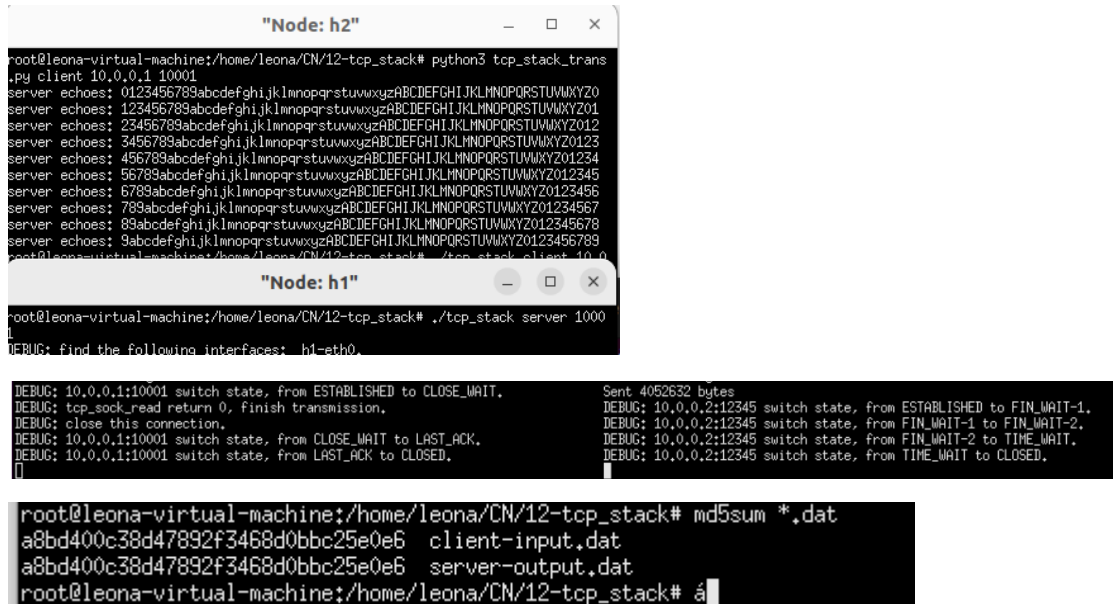
```
"Node: h2"                                     "Node: h1"
[6]+ Stopped ./tcp_stack client 10.0.0.1 10001
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack -3# ./tcp_stack client 1
0.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
INFO: all data has been sent to server, total = 4052632.
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
[]

≡ arp.o 8 static_struct_list
≡ C arpache.c 9 pthread_mutex_t ti
≡ arpache.o 10
≡ client-input.dat 11 // scan the timer
≡ $ create randfile.sh 12 void tcp_scan_time

[5]+ Stopped ./tcp_stack server 10001
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack -3# ./tcp_stack server 1
0001
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: in tcp_sock_accept
DEBUG: 10.0.0.1:10001 switch state, from LISTEN to SYN_RECV.
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept tcp_sock_accept
DEBUG: openfile success
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: received 4052632 bytes of data.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
```

```
leona@leona-virtual-machine:~/CN/12-tcp_stack -3$ md5sum *.dat
3dd790290b7945e7d98560ac9546a7c5  client-input.dat
3dd790290b7945e7d98560ac9546a7c5  server-output.dat
leona@leona-virtual-machine:~/CN/12-tcp_stack -3$ diff *.dat
leona@leona-virtual-machine:~/CN/12-tcp_stack -3$
```

(2)一端使用自己的代码，另一端使用内核版本 py，均可以正常运行



```
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# python3 tcp_stack_trans.py client 10.0.0.1 10001
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01
server echoes: 23456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012
server echoes: 3456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123
server echoes: 456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234
server echoes: 56789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345
server echoes: 6789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456
server echoes: 789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567
server echoes: 89abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345678
server echoes: 9abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack -3$ ./tcp_stack client 10.0.0.1 10001

root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# ./tcp_stack server 1000
DEBUG: find the following interfaces: h1-eth0.

DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: tcp_sock_read return 0, finish transmission.
DEBUG: close this connection.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.

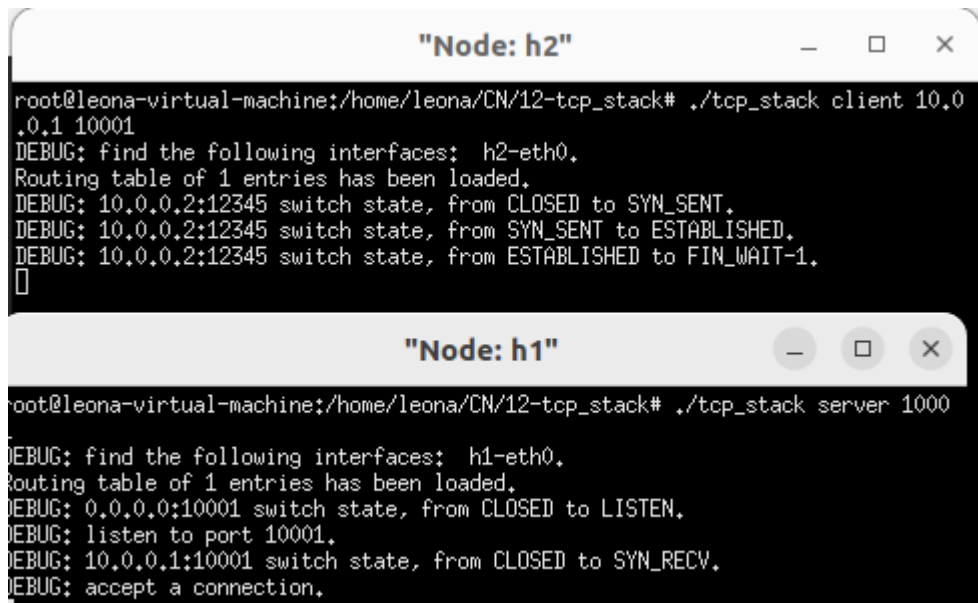
Sent 4052632 bytes
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.

root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# md5sum *.dat
a8bd400c38d47892f3468d0bbc25e0e6  client-input.dat
a8bd400c38d47892f3468d0bbc25e0e6  server-output.dat
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack#
```

## 五、 遇到的问题

### 1、 状态转换错误

在实验一中发现无法自动断开连接，第一次 h2 在 ESTABLISHED 状态停止，返回查看 tcp\_sock\_close，发现发送的数据包错误，原来为 TCP\_FIN|TCP\_ACK，但实际上应为只有 TCP\_FIN，更改后如下图可以进入 FIN\_WAIT-1 阶段，但依旧卡住。

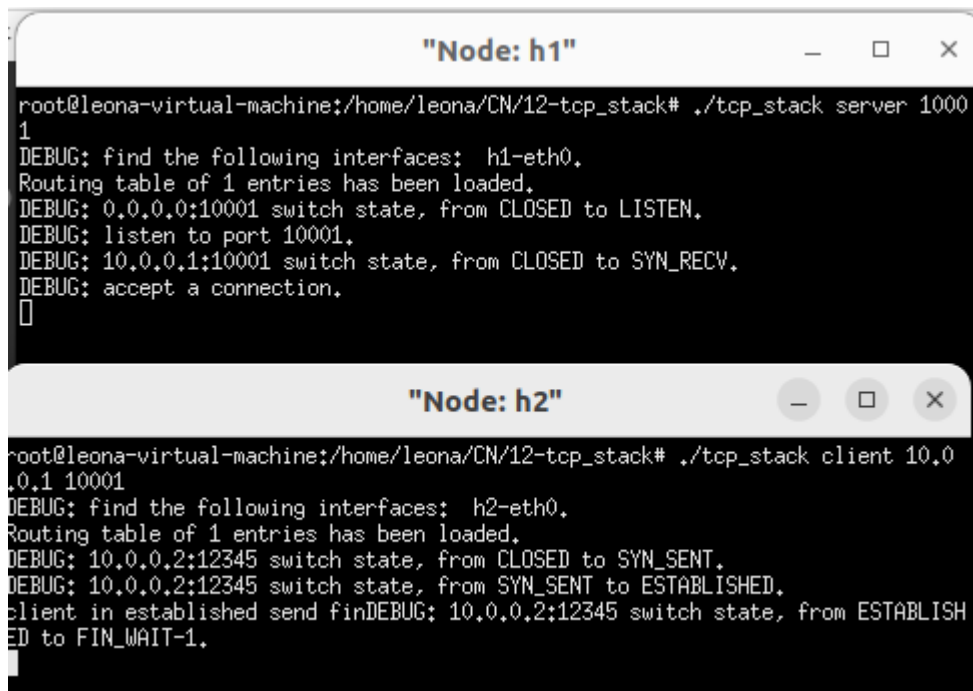


```
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# ./tcp_stack client 10.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.

root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# ./tcp_stack server 1000
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: accept a connection.
```

发现主动连接端（client）成功发送了 fin 包，但是被动连接端没有收到，因此无法断开连接。





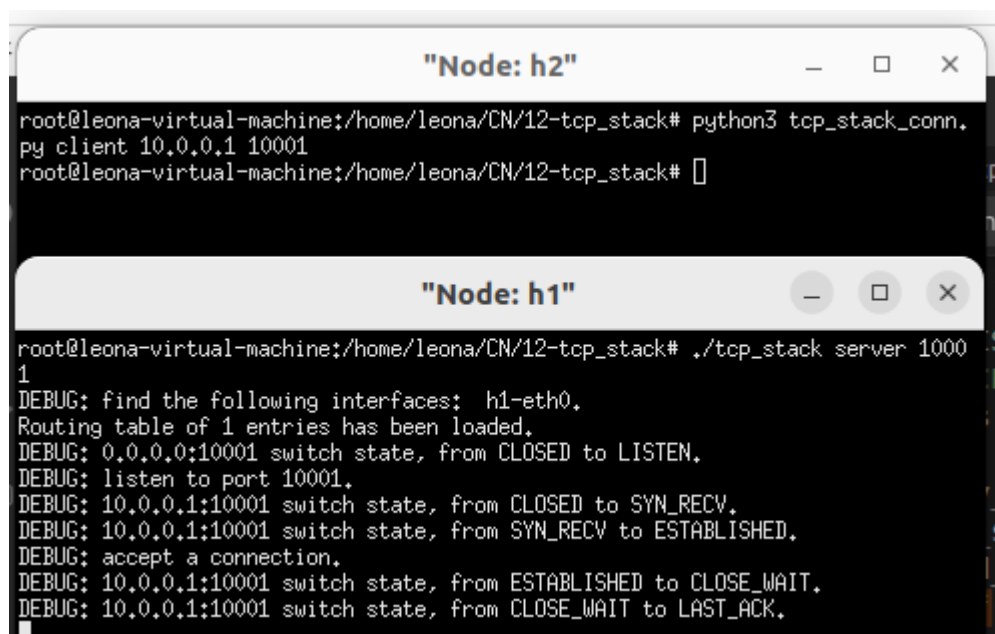
The image shows two terminal windows. The top window, titled "Node: h1", shows the server side of a TCP stack. It has executed the command `./tcp_stack server 1000 1`. The debug output shows it finding the `h1-eth0` interface, loading the routing table, and transitioning the switch state from `CLOSED` to `LISTEN`. It then listens on port 10001 and transitions the state from `CLOSED` to `SYN_RECV` upon receiving a connection. The bottom window, titled "Node: h2", shows the client side. It has executed the command `./tcp_stack client 10.0.0.1 10001`. The debug output shows it finding the `h2-eth0` interface, loading the routing table, and transitioning the switch state from `CLOSED` to `SYN_SENT` and then to `ESTABLISHED`. It then sends a `fin` and transitions the state from `ESTABLISHED` to `FIN_WAIT-1`.

```
"Node: h1"
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# ./tcp_stack server 1000 1
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: accept a connection.
[]

"Node: h2"
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# ./tcp_stack client 10.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
client in established send finDEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
```

再次进行 debug，发现其实 server 端停在 `SYN_RECV` 后不变化，但是 client 是成功变化的，因此查看状态机部分，发现在 server 端收到 `ACK` 后，虽然成功进行连接，但是没有进行状态转换，因此要添加状态转换语句。

```
case TCP_SYN_RECV:
    if(cb->flags & TCP_ACK)
    {
        tcp_sock_accept_enqueue(tsk);
        tsk->rcv_nxt = cb->seq;
        tsk->snd_una = cb->ack;
        wake_up(tsk->parent->wait_accept);
    }
```



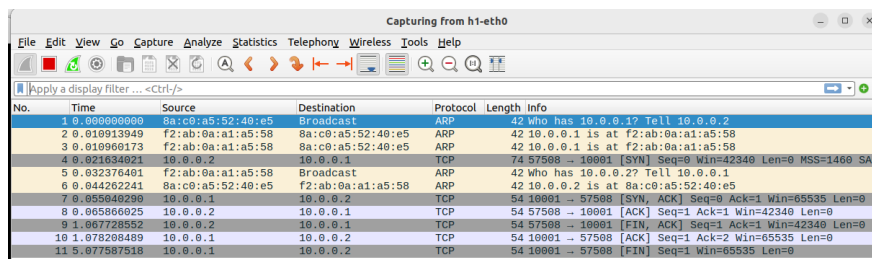
The image shows two terminal windows. The top window, titled "Node: h2", shows the client side. It has executed the command `python3 tcp_stack_conn.py client 10.0.0.1 10001`. The bottom window, titled "Node: h1", shows the server side. It has executed the command `./tcp_stack server 1000 1`. The debug output shows it finding the `h1-eth0` interface, loading the routing table, and transitioning the switch state from `CLOSED` to `LISTEN`. It then listens on port 10001 and transitions the state from `CLOSED` to `SYN_RECV`. It then accepts a connection and transitions the state from `SYN_RECV` to `ESTABLISHED`. It then sends a `fin` and transitions the state from `ESTABLISHED` to `CLOSE_WAIT` and then to `LAST_ACK`.

```
"Node: h2"
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# python3 tcp_stack_conn.py client 10.0.0.1 10001
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# []

"Node: h1"
root@leona-virtual-machine:/home/leona/CN/12-tcp_stack# ./tcp_stack server 1000 1
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
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.
```

## 2、发包错误

在 serve 端使用自己的代码，在 client 端使用内核版本时发现 serve 端会在状态到达 last\_ack 的时候停住，无法自己结束。询问助教后，使用 wireshark 抓包查看，发现最终应由 client 端向 server 端发送 ack，但是抓包中没有显示。



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	8a:c9:a5:52:40:e5	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
2	0.010913949	f2:ab:0a:a1:a5:58	8a:c9:a5:52:40:e5	ARP	42	10.0.0.1 is at f2:ab:0a:a1:a5:58
3	0.010960173	f2:ab:0a:a1:a5:58	8a:c9:a5:52:40:e5	ARP	42	10.0.0.1 is at f2:ab:0a:a1:a5:58
4	0.021634021	10.0.0.2	10.0.0.1	TCP	74	57508 → 10001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=3567738120 TSecr=3554557093
5	0.032376401	f2:ab:0a:a1:a5:58	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
6	0.044262241	8a:c9:a5:52:40:e5	f2:ab:0a:a1:a5:58	ARP	42	10.0.0.2 is at 8a:c9:a5:52:40:e5
7	0.055040290	10.0.0.1	10.0.0.2	TCP	54	10001 → 57508 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
8	0.065366025	10.0.0.2	10.0.0.1	TCP	54	57508 → 10001 [ACK] Seq=1 Ack=1 Win=42340 Len=0
9	1.007228552	10.0.0.2	10.0.0.1	TCP	54	57508 → 10001 [FIN, ACK] Seq=1 Ack=1 Win=42340 Len=0
10	1.078208489	10.0.0.1	10.0.0.2	TCP	54	10001 → 57508 [ACK] Seq=1 Ack=2 Win=65535 Len=0
11	5.077587518	10.0.0.1	10.0.0.2	TCP	54	10001 → 57508 [FIN] Seq=1 Win=65535 Len=0

由助教建议，查看都是内核版本的 wireshark 抓包，发现 server 端由 CLOSE\_WAIT 状态转换为 LAST\_ACK 时发送的是 fin|ack 包，但我只发送了 fin 包，因此进行修改。

Capturing from h1-eth0

editViewGoCaptureAnalyzeStatisticsTelephonyWirelessToolsHelp

</

修改后成功自动关闭连接，实现四次挥手。

3	0.010952178	9e:48:34:82:f9:1b	8a:3a:99:85:11:b0	ARP	42	10.0.0.1 is at 9e:48:34:82:f9:1b
4	0.021888229	10.0.0.2	10.0.0.1	TCP	74	58916 → 10001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=3567738120 TSecr=3554557093
5	0.032566129	9e:48:34:82:f9:1b	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
6	0.043339733	8a:3a:99:85:11:b0	9e:48:34:82:f9:1b	ARP	42	10.0.0.2 is at 8a:3a:99:85:11:b0
7	0.054062141	10.0.0.1	10.0.0.2	TCP	54	10001 → 58916 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
8	0.064397725	10.0.0.2	10.0.0.1	TCP	54	58916 → 10001 [ACK] Seq=1 Ack=1 Win=42340 Len=0
9	1.066873328	10.0.0.2	10.0.0.1	TCP	54	58916 → 10001 [FIN, ACK] Seq=1 Ack=1 Win=42340 Len=0
10	1.078070540	10.0.0.1	10.0.0.2	TCP	54	10001 → 58916 [ACK] Seq=1 Ack=2 Win=65535 Len=0
11	5.075691036	10.0.0.1	10.0.0.2	TCP	54	10001 → 58916 [FIN, ACK] Seq=1 Ack=2 Win=65535 Len=0
12	5.086762301	10.0.0.2	10.0.0.1	TCP	54	58916 → 10001 [ACK] Seq=2 Ack=2 Win=42340 Len=0

3、在实验二中，发现 server 端会停在 close\_wait，client 端会停在 fin\_wait\_2，根据抓包来看，发现最后 client 端已经发出了 fin 包，close\_wait 也已经对应发出了 ack 包，但是最终停止不动。最后发现是在 server 端收到 fin 包时，由于此次传输已经结束，因此需要退出所有等待的进程，即调用 wait\_exit，将 dead 位设置为 1，并向所有等待的进程广播。

```
"Node: h2"
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
plen 67
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNQP
plen 67
server echoes: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOQ
plen 67
server echoes: 23456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNQRS
plen 67
server echoes: 3456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNQRST
plen 67
server echoes: 456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNQRSTU
plen 67
server echoes: 56789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNQRSTU
plen 67
server echoes: 6789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNQRSTU
plen 67
server echoes: 789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNQRSTU
plen 67
server echoes: 89abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNQRSTU
plen 67
server echoes: 9abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNQRSTU
plen 67
server echoes: 0abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNQRSTU
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.

"Node: h1"
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
rlen 52
rlen 52
plen 52
rlen 52
plen 52
rlen 52
plen 52
rlen 52
plen 52
rlen 52
plen 52
rlen 52
plen 52
rlen 52
plen 52
rlen 52
plen 52
rlen 52
plen 52
rlen 52
plen 52
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
```

```
wait_exit(tsk->wait_recv);
```

#### 4、再次无法结束

发现是 apps 文件中 server 端无法跳出 while 循环，仔细查看代码，是由于 break 的条件是 rlen<0，但是当 read 函数读完之后返回值是 0，因此不会跳出循环，换成 rlen<=0 后即可。

```
rlen = tcp_sock_read(csk, buffer, sizeof(buffer));
if (rlen <= 0)
    break;
```

## 六、感想

由于在上个实验中使用到了哈希表映射的方式，因此这个实验中哈希表的使用比较得心应手，不会感觉到陌生。在实验中，对于线程同步的理解加深了许多。以及本次实验中由于三次实验的内容不同，因此对于自己要写什么有些许迷茫，老师给出的 apps.c 文件中也有小小的问题。另一个难点在于老师其实给出了很多已经实现的功能封装在函数中，但是就像散落的拼图碎片，需要自己理解后并在实验中拼起来，有点困难。整个实验算得上是到现在最为艰难的实验，耗费了很长的时间。在本地运行的时候，可能由于线程之间的同步完成的不是很好，导致代码有时能跑出结果有时候不能，让本就艰难的实验雪上加霜。