实验报告

网络传输机制实验二

一、实验内容

了解 TCP 数据缓存与接收、数据发送的处理流程，实现 TCP 协议栈的相应服务函数，实现 Socket 函数接口，验证功能正确性。

二、实验流程

1. 实现 TCP 协议栈数据传输功能。
2. 实现 Socket 相关函数。
3. 在给定拓扑下验证功能的正确性。
4. 通过传输文件验证功能的正确性。

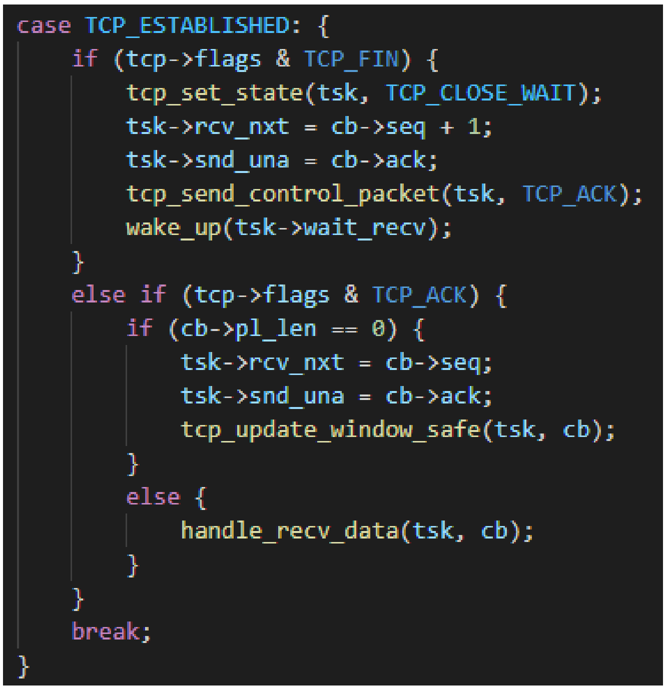
三、实验结果及分析

（一）TCP 功能实现思路

1、TCP 协议栈功能

协议栈需要实现收到数据传输相关数据包的处理，在 tcp_process 函数中增加 ESTABLISHED 状态下的数据包处理。

如下图所示，TCP_ACK 部分为增加内容，对于收到有效数据（pl_len > 0）和只收到 ACK 分别处理。对于 ACK，seq 和 ack 并不增加，不过要根据 ACK 数据包更新自己的发送窗口。对于有数据的数据包，则交由 handle_recv_data 函数进一步处理。

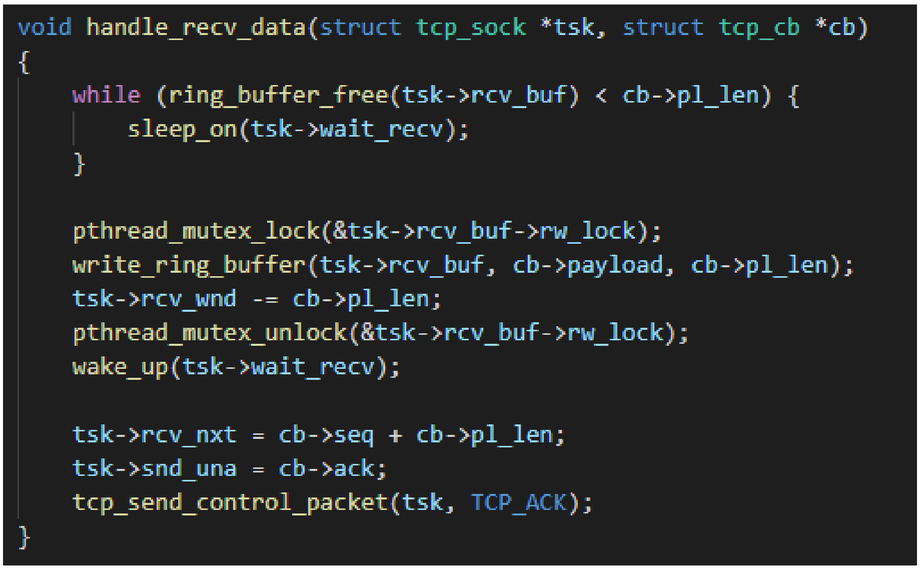


```c
case TCP_ESTABLISHED: {
    if (tcp->flags & TCP_FIN) {
        tcp_set_state(tsk, TCP_CLOSE_WAIT);
        tsk->rcv_nxt = cb->seq + 1;
        tsk->snd_una = cb->ack;
        tcp_send_control_packet(tsk, TCP_ACK);
        wake_up(tsk->wait_recv);
    }
    else if (tcp->flags & TCP_ACK) {
        if (cb->pl_len == 0) {
            tsk->rcv_nxt = cb->seq;
            tsk->snd_una = cb->ack;
            tcp_update_window_safe(tsk, cb);
        }
        else {
            handle_recv_data(tsk, cb);
        }
    }
    break;
}
```

```c
void handle_recv_data(struct tcp_sock *tsk, struct tcp_cb *cb)
```

函数，处理接收到的数据包，将数据写入用户缓存。首先判断缓存区是否足够存放这些数据，如果不足，将进程挂起，直到有足够的缓存区域。在实际写缓存前，还需要获取读写锁。仅凭 wait_recv 是不足以完成同步和互斥的，比如第二次收到数据包时，可能缓存区既有足够区域写，又是非空的。这样协议栈写和用户进程读会同时进行。因此这里需要用读写锁进一步互斥。

除了读写互斥，还需要实现流量控制。接收窗口等同于缓存区的剩余字节数，当协议栈写入缓存，接收窗口就要相应减少。写完后释放读写锁并唤醒 wait_recv。最后更新 seq 和 ack，并返回 ACK 数据包，表示已接收到数据。



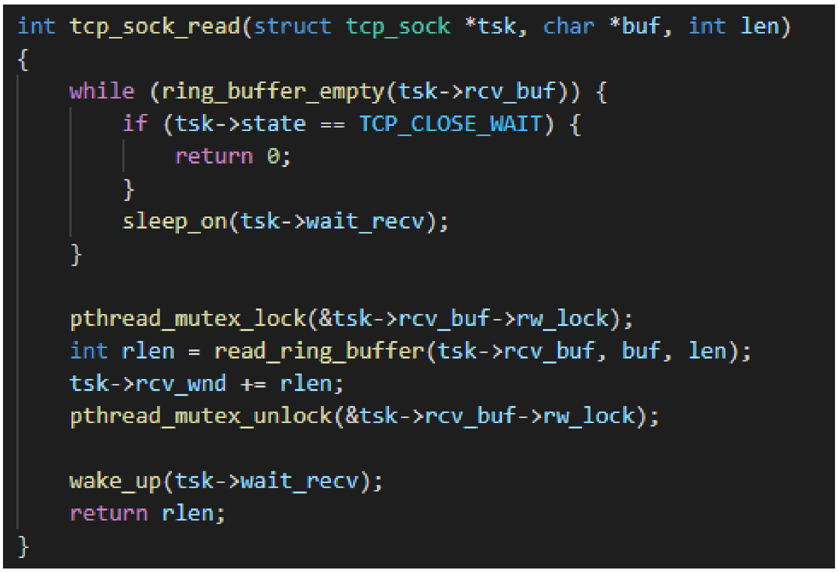
```c
void handle_recv_data(struct tcp_sock *tsk, struct tcp_cb *cb)
{
    while (ring_buffer_free(tsk->rcv_buf) < cb->pl_len) {
        sleep_on(tsk->wait_recv);
    }

    pthread_mutex_lock(&tsk->rcv_buf->rw_lock);
    write_ring_buffer(tsk->rcv_buf, cb->payload, cb->pl_len);
    tsk->rcv_wnd -= cb->pl_len;
    pthread_mutex_unlock(&tsk->rcv_buf->rw_lock);
    wake_up(tsk->wait_recv);

    tsk->rcv_nxt = cb->seq + cb->pl_len;
    tsk->snd_una = cb->ack;
    tcp_send_control_packet(tsk, TCP_ACK);
}
```

2、Socket 函数

```
int tcp_sock_read(struct tcp_sock *tsk, char *buf, int len)
```

函数，用户进程读取缓存区数据。首先判断缓存区是否有数据可读，如果没有则挂起。这里相比 handle_recv_data 函数多了一个 if 判断，这是用于断开连接后让 read 函数返回 0。未断开连接则不能在缓存区空时返回 0，因为此时不清楚之后还会不会有数据到达，返回 0 意味着不再会有数据达到。

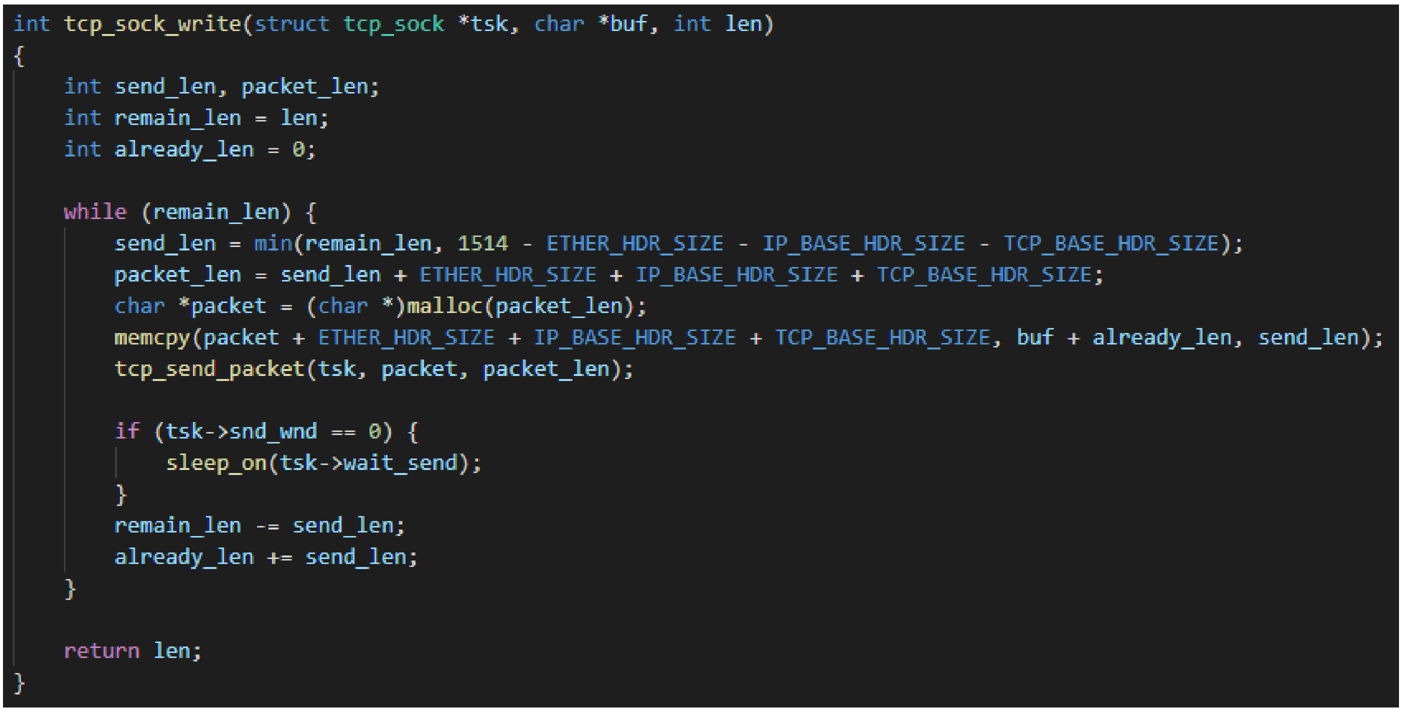
在读缓存之前先申请读写锁，原因前面已经讲过。注意读缓存读取得字节数不一定是请求的 len，如果缓存数据超过 len，则读取 len 个字节；否则读取当前所有字节。因此需要返回实际读取字节数 rlen。



```
int tcp_sock_read(struct tcp_sock *tsk, char *buf, int len)
{
    while (ring_buffer_empty(tsk->rcv_buf)) {
        if (tsk->state == TCP_CLOSE_WAIT) {
            return 0;
        }
        sleep_on(tsk->wait_recv);
    }

    pthread_mutex_lock(&tsk->rcv_buf->rw_lock);
    int rlen = read_ring_buffer(tsk->rcv_buf, buf, len);
    tsk->rcv_wnd += rlen;
    pthread_mutex_unlock(&tsk->rcv_buf->rw_lock);

    wake_up(tsk->wait_recv);
    return rlen;
}
```

```
int tcp_sock_write(struct tcp_sock *tsk, char *buf, int len)
```

函数，用户进程发送数据。发送时每个数据包最多只能携带给定数量的字节数，因此需要用多个数据包发送，当发送窗口为 0 时需要停止发送，等待对面更新接收窗口时再次唤起。



```
int tcp_sock_write(struct tcp_sock *tsk, char *buf, int len)
{
    int send_len, packet_len;
    int remain_len = len;
    int already_len = 0;

    while (remain_len) {
        send_len = min(remain_len, 1514 - ETHER_HDR_SIZE - IP_BASE_HDR_SIZE - TCP_BASE_HDR_SIZE);
        packet_len = send_len + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE;
        char *packet = (char *)malloc(packet_len);
        memcpy(packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE, buf + already_len, send_len);
        tcp_send_packet(tsk, packet, packet_len);

        if (tsk->snd_wnd == 0) {
            sleep_on(tsk->wait_send);
        }
        remain_len -= send_len;
        already_len += send_len;
    }

    return len;
}
```

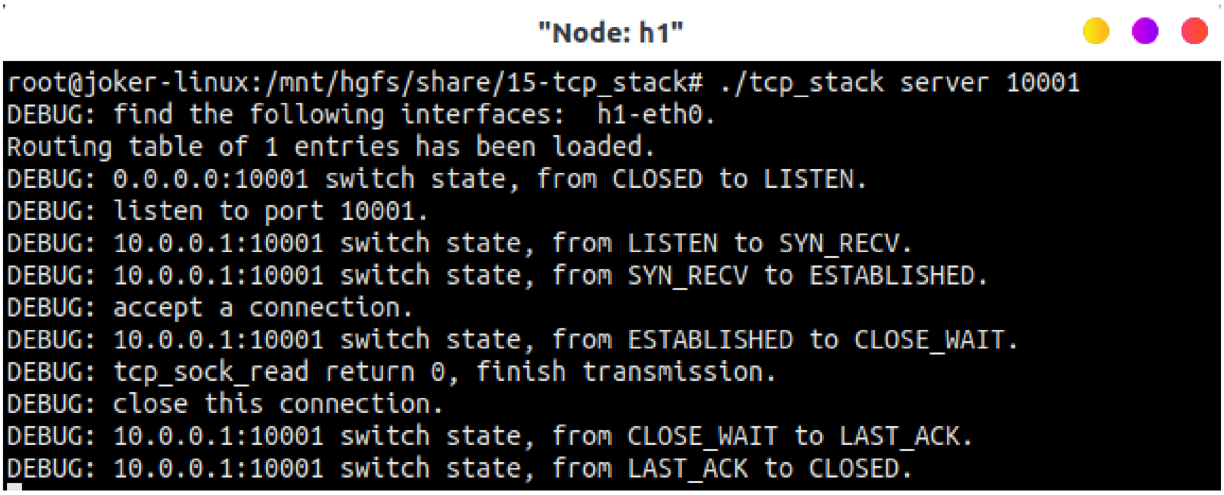
（二）实验功能验证

1、给定拓扑下简单数据传输

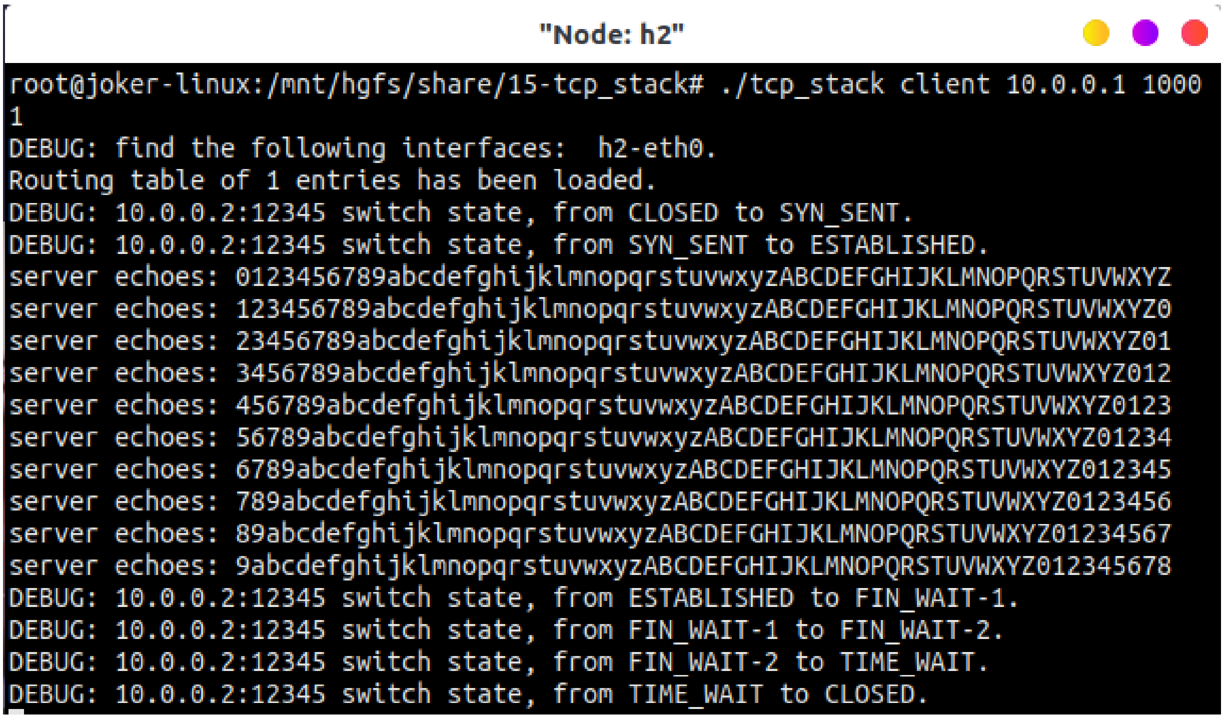
（1）本实验 server 和 client

H1：本实验 server

H2：本实验 client



```
root@joker-linux:/mnt/hgfs/share/15-tcp_stack# ./tcp_stack server 10001
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 LISTEN 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.
```



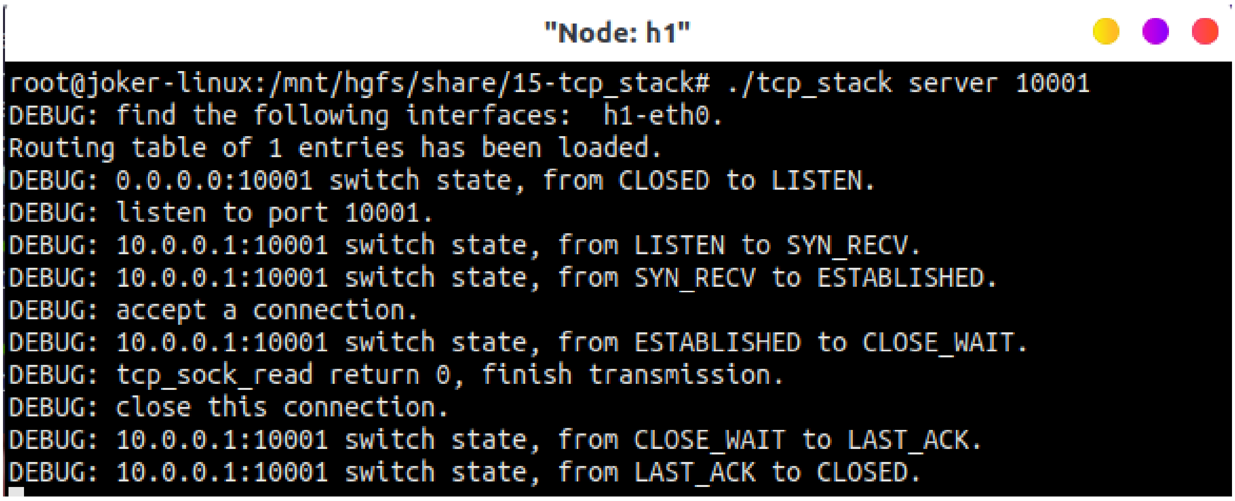
```
root@joker-linux:/mnt/hgfs/share/15-tcp_stack# ./tcp_stack client 10.0.0.1 1000
1
DEBUG: find the following interfaces:  h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0
server echoes: 23456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01
server echoes: 3456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012
server echoes: 456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123
server echoes: 56789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234
server echoes: 6789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345
server echoes: 789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456
server echoes: 89abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567
server echoes: 9abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345678
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.
```

H1、H2 能正确收发数据。

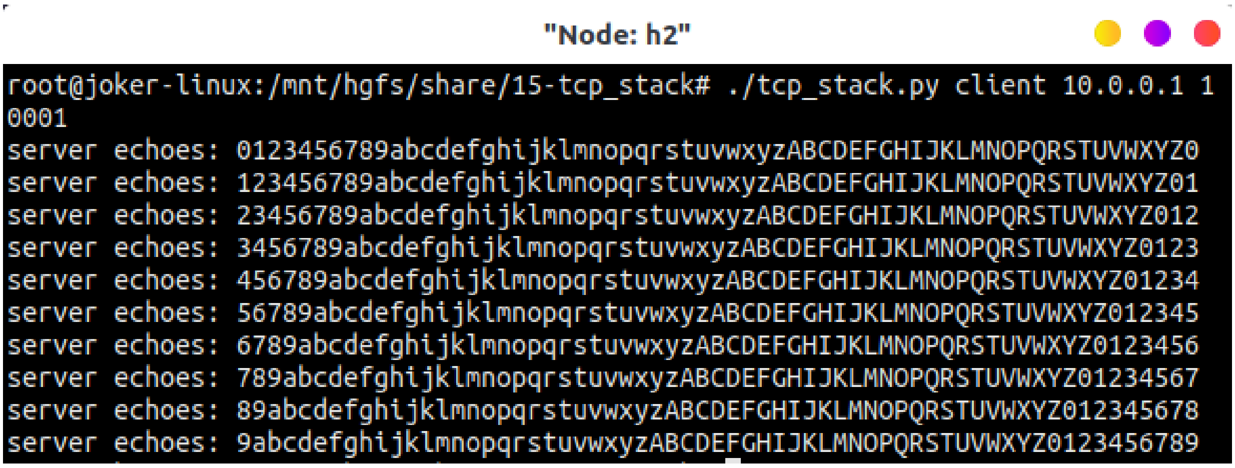
（2）本实验 server 和标准 client

H1：本实验 server

H2：标准 client



```
root@joker-linux:/mnt/hgfs/share/15-tcp_stack# ./tcp_stack server 10001
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 LISTEN 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.
```



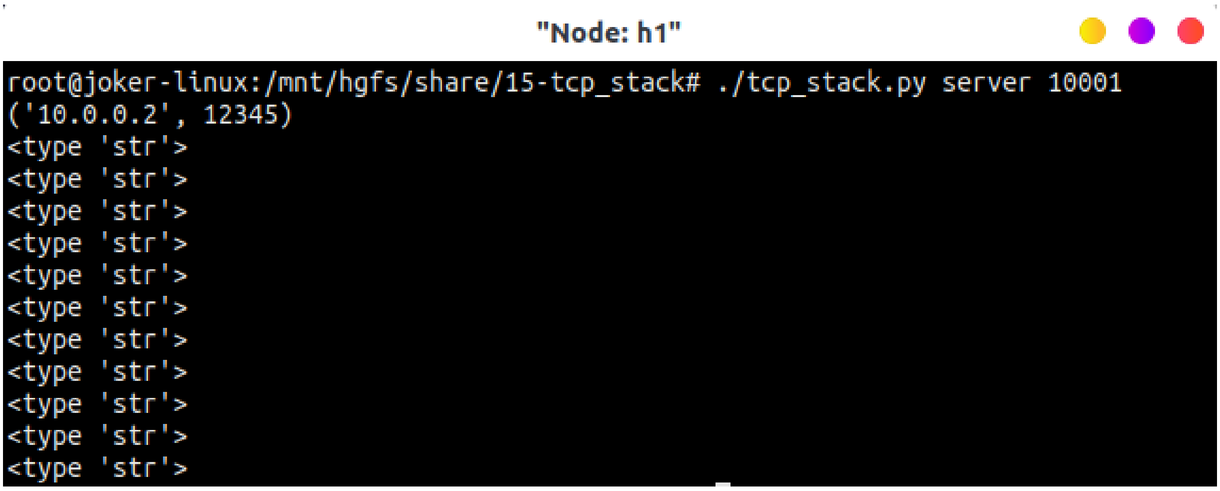
```
root@joker-linux:/mnt/hgfs/share/15-tcp_stack# ./tcp_stack.py client 10.0.0.1 1
0001
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0
server echoes: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01
server echoes: 23456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012
server echoes: 3456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123
server echoes: 456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234
server echoes: 56789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345
server echoes: 6789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456
server echoes: 789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567
server echoes: 89abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345678
server echoes: 9abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
```

H1、H2 能正确收发数据。

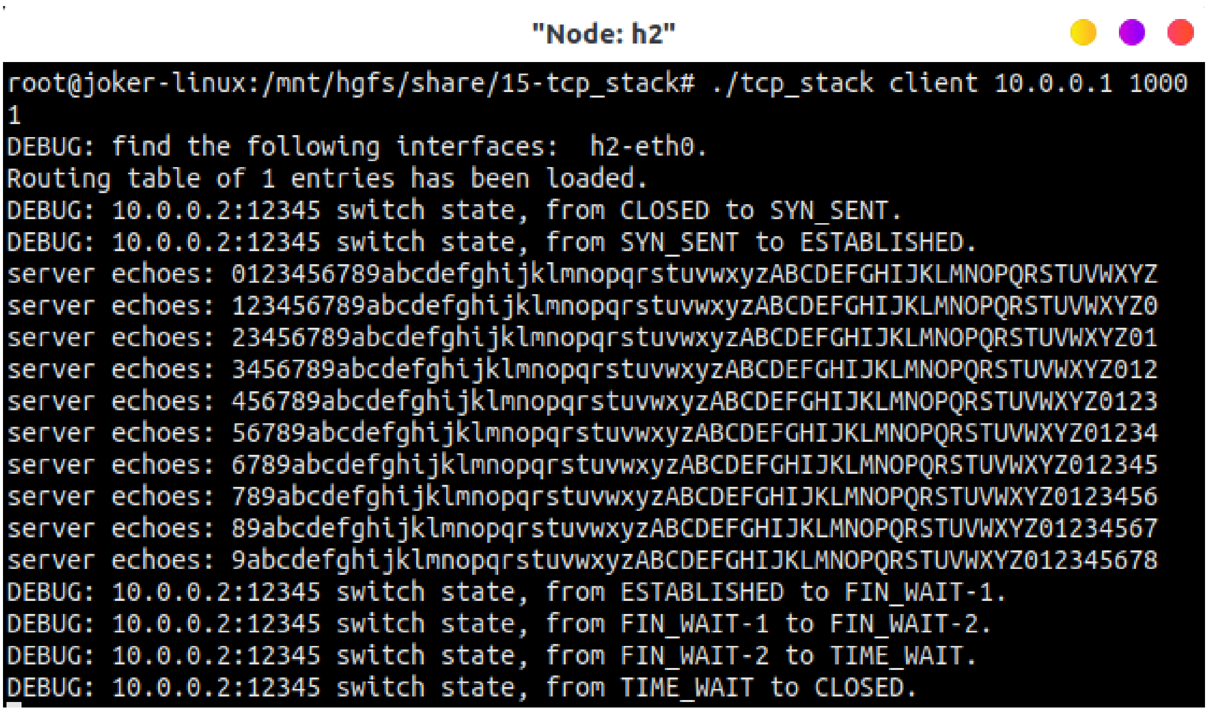
（3）标准 server 和本实验 client

H1：标准 server

H2：本实验 client



```
root@joker-linux:/mnt/hgfs/share/15-tcp_stack# ./tcp_stack.py server 10001
('10.0.0.2', 12345)
<type 'str'>
<type 'str'>
<type 'str'>
<type 'str'>
<type 'str'>
<type 'str'>
<type 'str'>
<type 'str'>
<type 'str'>
<type 'str'>
<type 'str'>
```



```
root@joker-linux:/mnt/hgfs/share/15-tcp_stack# ./tcp_stack client 10.0.0.1 1000
1
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: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0
server echoes: 23456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01
server echoes: 3456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012
server echoes: 456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123
server echoes: 56789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234
server echoes: 6789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345
server echoes: 789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456
server echoes: 89abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567
server echoes: 9abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345678
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.
```

H1、H2 能正确收发数据。

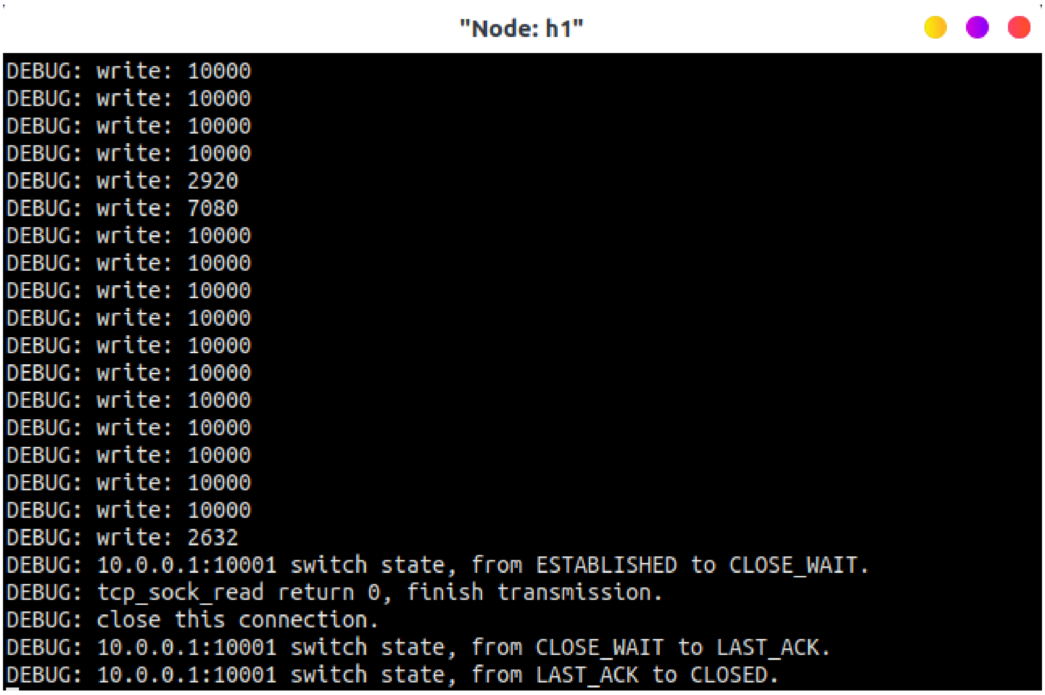
综上，本实验 server 和 client 能正确收发数据，功能正确。

2、文件传输

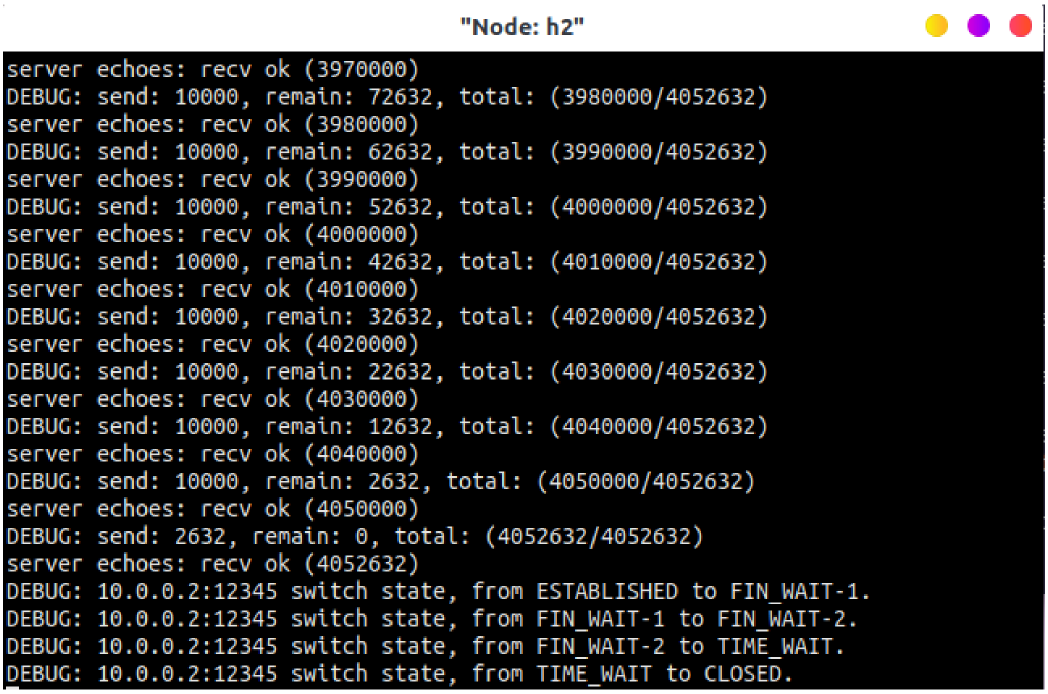
（1）本实验 server 和 client

H1：本实验 server

H2：本实验 client

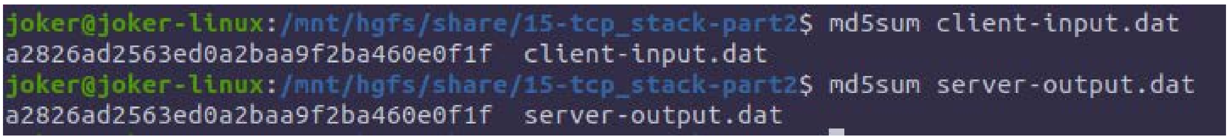


```
DEBUG: write: 10000
DEBUG: write: 10000
DEBUG: write: 10000
DEBUG: write: 10000
DEBUG: write: 2920
DEBUG: write: 7080
DEBUG: write: 10000
DEBUG: write: 10000
DEBUG: write: 10000
DEBUG: write: 10000
DEBUG: write: 10000
DEBUG: write: 10000
DEBUG: write: 10000
DEBUG: write: 10000
DEBUG: write: 10000
DEBUG: write: 10000
DEBUG: write: 10000
DEBUG: write: 2632
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.
```



```
server echoes: recv ok (3970000)
DEBUG: send: 10000, remain: 72632, total: (3980000/4052632)
server echoes: recv ok (3980000)
DEBUG: send: 10000, remain: 62632, total: (3990000/4052632)
server echoes: recv ok (3990000)
DEBUG: send: 10000, remain: 52632, total: (4000000/4052632)
server echoes: recv ok (4000000)
DEBUG: send: 10000, remain: 42632, total: (4010000/4052632)
server echoes: recv ok (4010000)
DEBUG: send: 10000, remain: 32632, total: (4020000/4052632)
server echoes: recv ok (4020000)
DEBUG: send: 10000, remain: 22632, total: (4030000/4052632)
server echoes: recv ok (4030000)
DEBUG: send: 10000, remain: 12632, total: (4040000/4052632)
server echoes: recv ok (4040000)
DEBUG: send: 10000, remain: 2632, total: (4050000/4052632)
server echoes: recv ok (4050000)
DEBUG: send: 2632, remain: 0, total: (4052632/4052632)
server echoes: recv ok (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.
```

MD5 验证



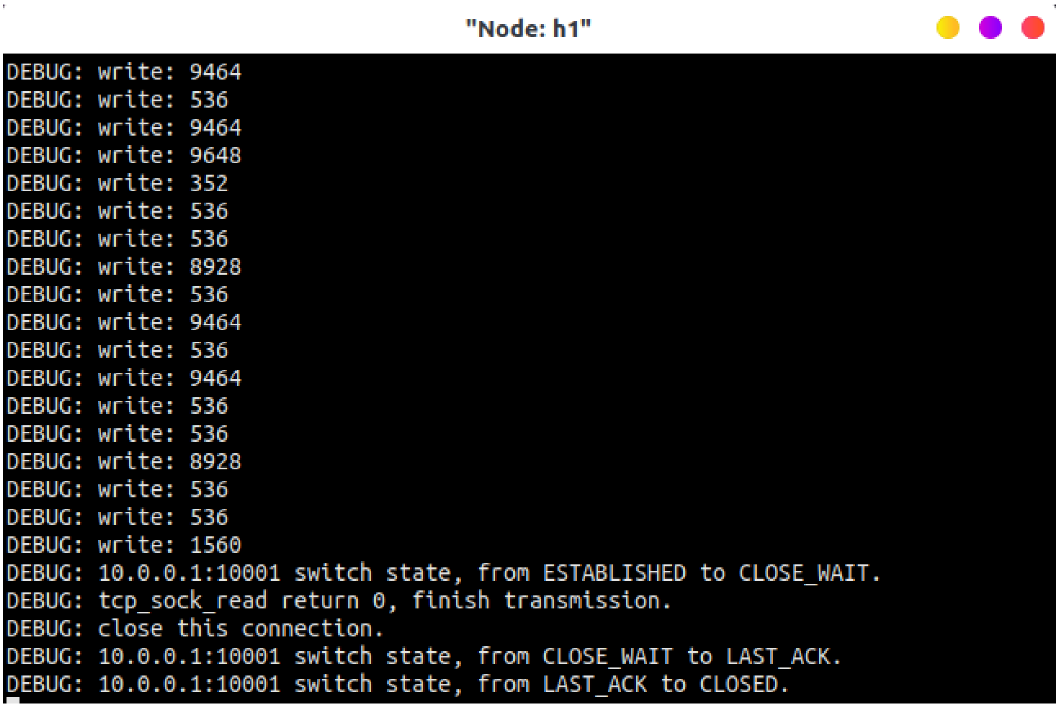
```
joker@joker-linux:/mnt/hgfs/share/15-tcp_stack-part2$ md5sum client-input.dat
a2826ad2563ed0a2baa9f2ba460e0f1f  client-input.dat
joker@joker-linux:/mnt/hgfs/share/15-tcp_stack-part2$ md5sum server-output.dat
a2826ad2563ed0a2baa9f2ba460e0f1f  server-output.dat
```

H1、H2 能正确传输文件。

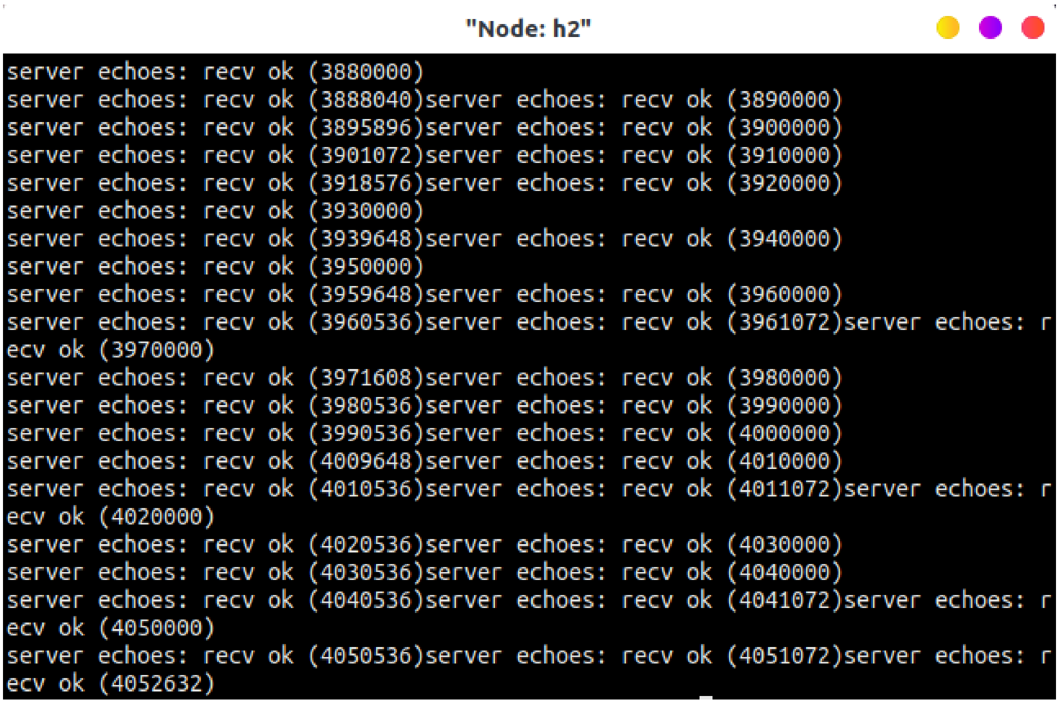
（2）本实验 server 和标准 client

H1：本实验 server

H2：标准 client

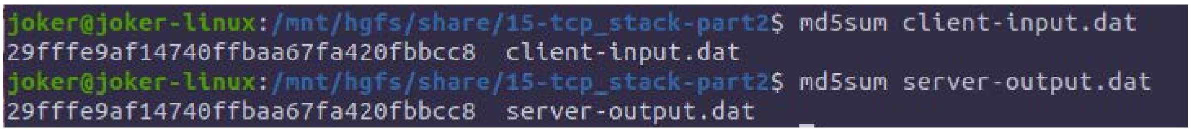


```
DEBUG: write: 9464
DEBUG: write: 536
DEBUG: write: 9464
DEBUG: write: 9648
DEBUG: write: 352
DEBUG: write: 536
DEBUG: write: 536
DEBUG: write: 8928
DEBUG: write: 536
DEBUG: write: 9464
DEBUG: write: 536
DEBUG: write: 9464
DEBUG: write: 536
DEBUG: write: 536
DEBUG: write: 8928
DEBUG: write: 536
DEBUG: write: 536
DEBUG: write: 1560
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.
```



```
server echoes: recv ok (3880000)
server echoes: recv ok (3888040)server echoes: recv ok (3890000)
server echoes: recv ok (3895896)server echoes: recv ok (3900000)
server echoes: recv ok (3901072)server echoes: recv ok (3910000)
server echoes: recv ok (3918576)server echoes: recv ok (3920000)
server echoes: recv ok (3930000)
server echoes: recv ok (3939648)server echoes: recv ok (3940000)
server echoes: recv ok (3950000)
server echoes: recv ok (3959648)server echoes: recv ok (3960000)
server echoes: recv ok (3960536)server echoes: recv ok (3961072)server echoes: r
ecv ok (3970000)
server echoes: recv ok (3971608)server echoes: recv ok (3980000)
server echoes: recv ok (3980536)server echoes: recv ok (3990000)
server echoes: recv ok (3990536)server echoes: recv ok (4000000)
server echoes: recv ok (4009648)server echoes: recv ok (4010000)
server echoes: recv ok (4010536)server echoes: recv ok (4011072)server echoes: r
ecv ok (4020000)
server echoes: recv ok (4020536)server echoes: recv ok (4030000)
server echoes: recv ok (4030536)server echoes: recv ok (4040000)
server echoes: recv ok (4040536)server echoes: recv ok (4041072)server echoes: r
ecv ok (4050000)
server echoes: recv ok (4050536)server echoes: recv ok (4051072)server echoes: r
ecv ok (4052632)
```

MD5 验证



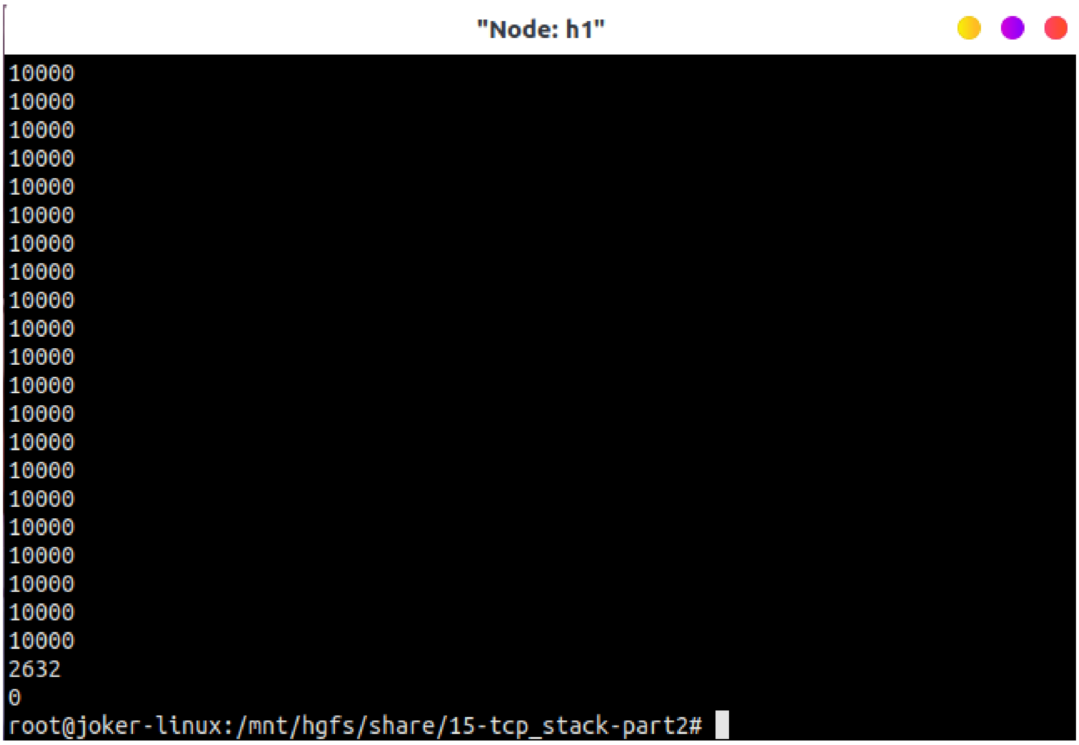
```
joker@joker-linux:/mnt/hgfs/share/15-tcp_stack-part2$ md5sum client-input.dat
29fffe9af14740ffbaa67fa420fbbcc8  client-input.dat
joker@joker-linux:/mnt/hgfs/share/15-tcp_stack-part2$ md5sum server-output.dat
29fffe9af14740ffbaa67fa420fbbcc8  server-output.dat
```

H1、H2 能正确传输文件。

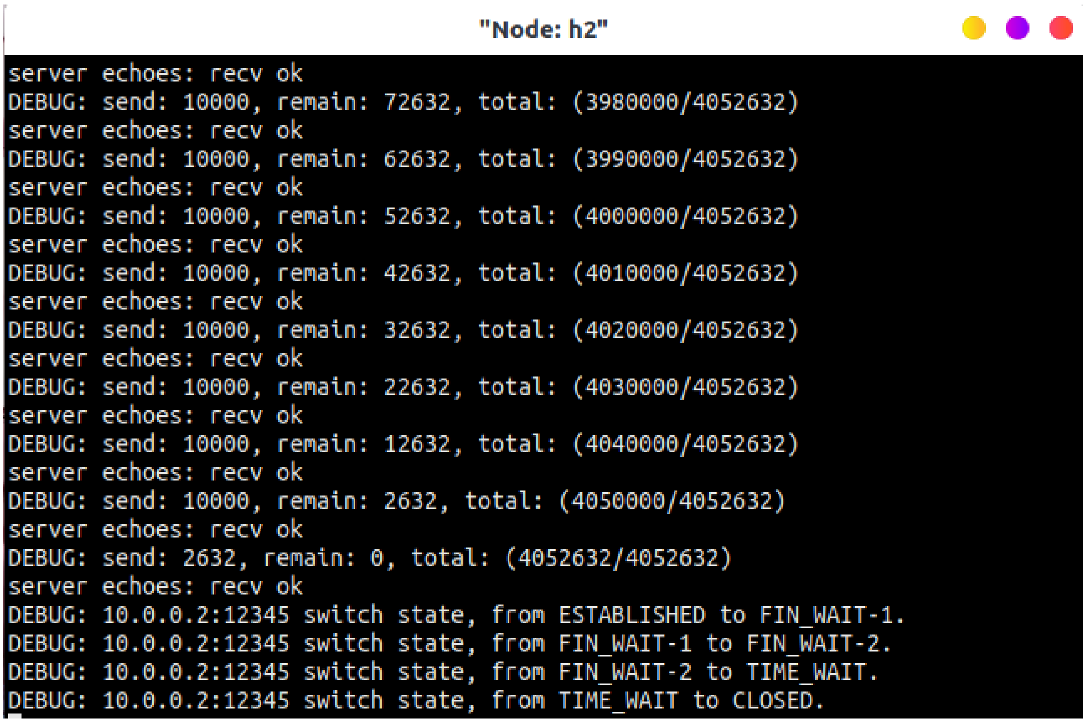
（3）标准 server 和本实验 client

H1：标准 server

H2：本实验 client

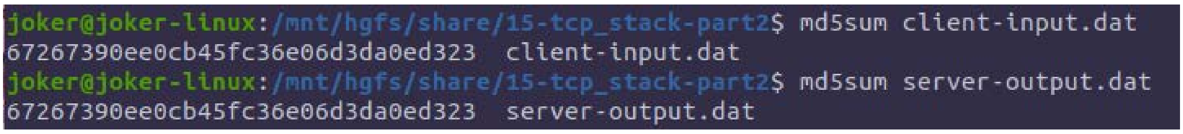


```
10000
10000
10000
10000
10000
10000
10000
10000
10000
10000
10000
10000
10000
10000
10000
10000
10000
10000
10000
10000
10000
2632
0
root@joker-linux:/mnt/hgfs/share/15-tcp_stack-part2# 
```



```
server echoes: recv ok
DEBUG: send: 10000, remain: 72632, total: (3980000/4052632)
server echoes: recv ok
DEBUG: send: 10000, remain: 62632, total: (3990000/4052632)
server echoes: recv ok
DEBUG: send: 10000, remain: 52632, total: (4000000/4052632)
server echoes: recv ok
DEBUG: send: 10000, remain: 42632, total: (4010000/4052632)
server echoes: recv ok
DEBUG: send: 10000, remain: 32632, total: (4020000/4052632)
server echoes: recv ok
DEBUG: send: 10000, remain: 22632, total: (4030000/4052632)
server echoes: recv ok
DEBUG: send: 10000, remain: 12632, total: (4040000/4052632)
server echoes: recv ok
DEBUG: send: 10000, remain: 2632, total: (4050000/4052632)
server echoes: recv ok
DEBUG: send: 2632, remain: 0, total: (4052632/4052632)
server echoes: recv ok
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.
```

MD5 验证



```
joker@joker-linux:/mnt/hgfs/share/15-tcp_stack-part2$ md5sum client-input.dat
67267390ee0cb45fc36e06d3da0ed323  client-input.dat
joker@joker-linux:/mnt/hgfs/share/15-tcp_stack-part2$ md5sum server-output.dat
67267390ee0cb45fc36e06d3da0ed323  server-output.dat
```

H1、H2 能正确传输文件。

综上，本实验 server 和 client 能正确传输文件，功能正确。

四、实验总结

通过本次实验，我了解了 TCP 协议栈的传输数据功能、socket 发送与读取数据的处理流程，同时也了解到简单的流量控制方式，这让我对 TCP 协议有了更多的认识。