

# 第十五周实验报告

## 网络传输机制实验（第二部分）

2015K8009922021

李一苇

### 一、实验内容

- 实现数据传输
  - 如何将数据封装到数据包并发送
  - 收到数据和ACK时的相应处理
- 实现流量控制
  - 通过调整rcv\_window来表达自己的接收能力
- 实现tcp\_sock相关函数
  - 类似socket函数，能够收发数据

### 二、实验流程

#### 1. 实现 tcp\_sock\_read 函数

通过 read\_ring\_buffer 函数读取缓存区收到的数据包给上层应用。同时更新 rcv\_wnd 为缓冲区的剩余值。

注意：

- 读取过程需自己添加锁 rcv\_buf\_lock
- 必须在ESTABLISHED及以前的状态才能读数据，如果在TCP\_CLOSE\_WAIT状态，说明服务器已经收到客户端的FIN包，此时执行被唤醒的 read 直接返回-1，让上层app直接跳出循环

具体代码如下：

```
int tcp_sock_read(struct tcp_sock *tsk, char *buf, int len) {
    if (tsk->state == TCP_CLOSE_WAIT)
        return -1;
    if (ring_buffer_empty(tsk->rcv_buf))
        sleep_on(tsk->wait_rcv);
    pthread_mutex_lock(&tsk->rcv_buf_lock);
    int read_len = read_ring_buffer(tsk->rcv_buf, buf, len);
    pthread_mutex_unlock(&tsk->rcv_buf_lock);
    tsk->rcv_wnd = ring_buffer_free(tsk->rcv_buf);
    return read_len;
}
```

#### 2. 实现 tcp\_sock\_write 函数

封装好一个tcp数据包后通过 tcp\_send\_packet 发送出去。

需要注意：

- 数据包的大小是 `ETH_FRAME_LEN` 和 `len` 加一系列包头二值的较小值
- 如果当前发送窗口 `tsk->snd_wnd` 足够，则发送；若不足则返回错误；否则阻塞上层应用的 `write` 函数，等待收到下一个tcp包后更新发送窗口，实现端到端的流量控制

具体代码如下：

```
int tcp_sock_write(struct tcp_sock *tsk, char *buf, int len) {
    if (tsk->snd_wnd == 0) sleep_on(tsk->wait_send);
    int packet_len =
        min(len + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE,
            ETH_FRAME_LEN);
    char *packet = malloc(packet_len);
    char *data = packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE;
    memcpy(data, buf,
        packet_len - ETHER_HDR_SIZE - IP_BASE_HDR_SIZE - TCP_BASE_HDR_SIZE);
    if (tsk->snd_wnd >= packet_len) {
        tcp_send_packet(tsk, packet, packet_len);
        return packet_len;
    }
    return 0;
}
```

### 3. 对 `tcp_process` 函数的修改

- 需要在TCP\_SYN\_RECV及以后的状态进行发送窗口更新，使用 `tcp_update_window_safe` 函数
- 在ESTABLISHED阶段，收到普通的ACK，且`pl_len>0`时，代表该包带有数据，需要用 `write_ring_buffer` 写入接收的缓冲区，同时唤醒信号量 `tsk->wait_recv` 让上层应用读取
- 在ESTABLISHED阶段，收到FIN包，代表对端要求结束连接，则需要额外唤醒信号量 `tsk->wait_recv` 通知上层应用退出阻塞阶段，继续进一步的操作

## 三、查错心得

### 1. 状况：发现两端进入ESTABLISHED状态之后死锁

原因：发送窗口的未更新。发送窗口的更新应该从收到有seq和ack域的tcp包开始，即TCP\_SYN\_RECV阶段及以后

### 2. 状况：客户端出现大量CLOSED-CLOSED状态转换

原因：服务器在ESTABLISHED阶段每收到一个包就发一个ACK请求，导致客户端在结束状态收到不应收到的包进而重复进入RST态而出现大量CLOSED-CLOSED转换记录

### 3. 状况：服务端在CLOSE\_WAIT和客户端在FIN\_WAIT\_2死锁

原因：服务器应用在收到客户端的tcp包后，仍然阻塞在 `read` 接口处。解决办法是在收到FIN包后唤醒 `wait_recv`，让其返回-1值。

## 四、实验结果和分析

测试方法：运行网络拓扑tcp\_topo.py

- 在h1上运行TCP协议栈的服务器模式 `./tcp_stack server 10001`
- 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关闭连接 `./tcp_stack client 10.0.0.1 10001`

截图如下：（为了截图方便，已经调整tcp\_app.s里的发送次数n=3）

```
"Node: h2"
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
Usage:
    tcp_stack server local_port
    tcp_stack client remote_ip remote_port
root@lywPC:/media/sf_E_DRIVE/UCAS/wlsy网/15-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 tcp_bind_hash: 0
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.
tcp_sock_read: sleep because empty rcv_buf
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
tcp_sock_read: sleep because empty rcv_buf
server echoes: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
tcp_sock_read: sleep because empty rcv_buf
server echoes: 23456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
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.
DEBUG: free tcp sock: [10.0.0.2:12345->10.0.0.1:10001].

"Node: h1"
root@lywPC:/media/sf_E_DRIVE/UCAS/wlsy网/15-tcp-stack# ./tcp_stack server
10001
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG tcp_bind_hash: 0
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG tcp_bind_hash: 0
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: 10.0.0.1:10001 switch state, from ESTABLISHED to ESTABLISHED.
DEBUG: accept a connection.
tcp_sock_read: sleep because empty rcv_buf
tcp_sock_read: sleep because empty rcv_buf
tcp_sock_read: sleep because empty rcv_buf
tcp_sock_read: sleep because empty rcv_buf
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: tcp_sock_read return negative value, finish transmission.
DEBUG: close this connection.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: free tcp sock: [10.0.0.1:10001->10.0.0.2:12345].
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
[]
```

可以看到客户端和服务端进行了正确的信息交互，且连接的维护操作都正常。

服务器端换成脚本文件后：

```
"Node: h2"
root@lywPC:/media/sf_E_DRIVE/UCAS/wlsy网/15-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 tcp_bind_hash: 0
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.
tcp_sock_read: sleep because empty rcv_buf
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZserver
echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZserver
echoes: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZserver
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.
DEBUG: free tcp sock: [10.0.0.2:12345->10.0.0.1:10001].

"Node: h1"
root@lywPC:/media/sf_E_DRIVE/UCAS/wlsy网/15-tcp-stack# python tcp_stack.p
y server 10001
( '10.0.0.2', 12345)
root@lywPC:/media/sf_E_DRIVE/UCAS/wlsy网/15-tcp-stack# []
```

客户端换成脚本文件后：

```
"Node: h2"
root@lywPC:/media/sf_E_DRIVE/UCAS/wlsy网/15-tcp-stack# python ./tcp_stack
.py client 10.0.0.1 10001
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
root@lywPC:/media/sf_E_DRIVE/UCAS/wlsy网/15-tcp-stack#

"Node: h1"
DEBUG: listen to port 10001.
DEBUG tcp_bind_hash: 0
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: 10.0.0.1:10001 switch state, from ESTABLISHED to ESTABLISHED.
DEBUG: accept a connection.
tcp_sock_read: sleep because empty rcv_buf
tcp_sock_read: sleep because empty rcv_buf
tcp_sock_read: sleep because empty rcv_buf
tcp_sock_read: sleep because empty rcv_buf
tcp_sock_read: sleep because empty rcv_buf
tcp_sock_read: sleep because empty rcv_buf
tcp_sock_read: sleep because empty rcv_buf
tcp_sock_read: sleep because empty rcv_buf
tcp_sock_read: sleep because empty rcv_buf
tcp_sock_read: sleep because empty rcv_buf
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: tcp_sock_read return negative value, finish transmission.
DEBUG: close this connection.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: free tcp sock: [10.0.0.1:10001->10.0.0.2:47540].
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
[]
```

由于脚本文件里客户端发送的文本是常数所以服务器的echo有所不同，但仍然是正确的echo。