计算机网络实验报告-16

阮星程
2015K8009929047

一、 实验题目

网络传输机制实验二。

二、 实验内容

基于前面搭建的 TCP 基本连接框架，实现在无丢包环境下的数据的收发。

三、 实验过程

本次实验代码主要在两个文件中，tcp_in.c 和 tcp_sock.c，不同于之前的基本框架构建，这次实验只需要在实验一的基础上添加收到数据的处理，以及实现两个基于框架的 write 和 read 函数，代码量很小，主要难度在于与之前框架的协调。

首先在 tcp_in 中添加收包时的数据存储

```
//9
if(tcp_data_len > 0)
{
    write_ring_buffer(tsk->rcv_buf, ((char *) tcp) + TCP_HDR_SIZE(tcp), tcp_data_len);
    log(DEBUG,"buf writen len %d", tcp_data_len);
    tsk->rcv_wnd -= tcp_data_len;
}
```

在更新过 rcv_nxt 和 snd_una 后回复 ack

```
//11 for normal data
tsk->rcv_nxt = ntohl(tcp->seq) + (tcp_data_len ? tcp_data_len : (TCP_SYN | TCP_FIN) & (tcp->flags)) ;
tsk->snd_una = ntohl(tcp->ack) - 1;

if(tcp_data_len > 0)
    tcp_send_control_packet(tsk, TCP_ACK);
return;
```

收包的部分便处理完毕了。

接下来实现 read。当 buffer 为空时，等待，当 buffer 不空时读取对应长度的数据返回

```
int tcp_sock_read(struct tcp_sock *tsk, char *buf, int len)
{
    while(!ring_buffer_used(tsk->rcv_buf))
    {
        sleep(1);
    }
    len = read_ring_buffer(tsk->rcv_buf, buf, len);
    //log(DEBUG,"receive message %s", buf);
    tsk->rcv_wnd += len;
    return len;
}
```

构建 write，对于不能一次发送或超过对方接收窗口的部分，先留下之后再发送。待全部发送完成后，返回即可。

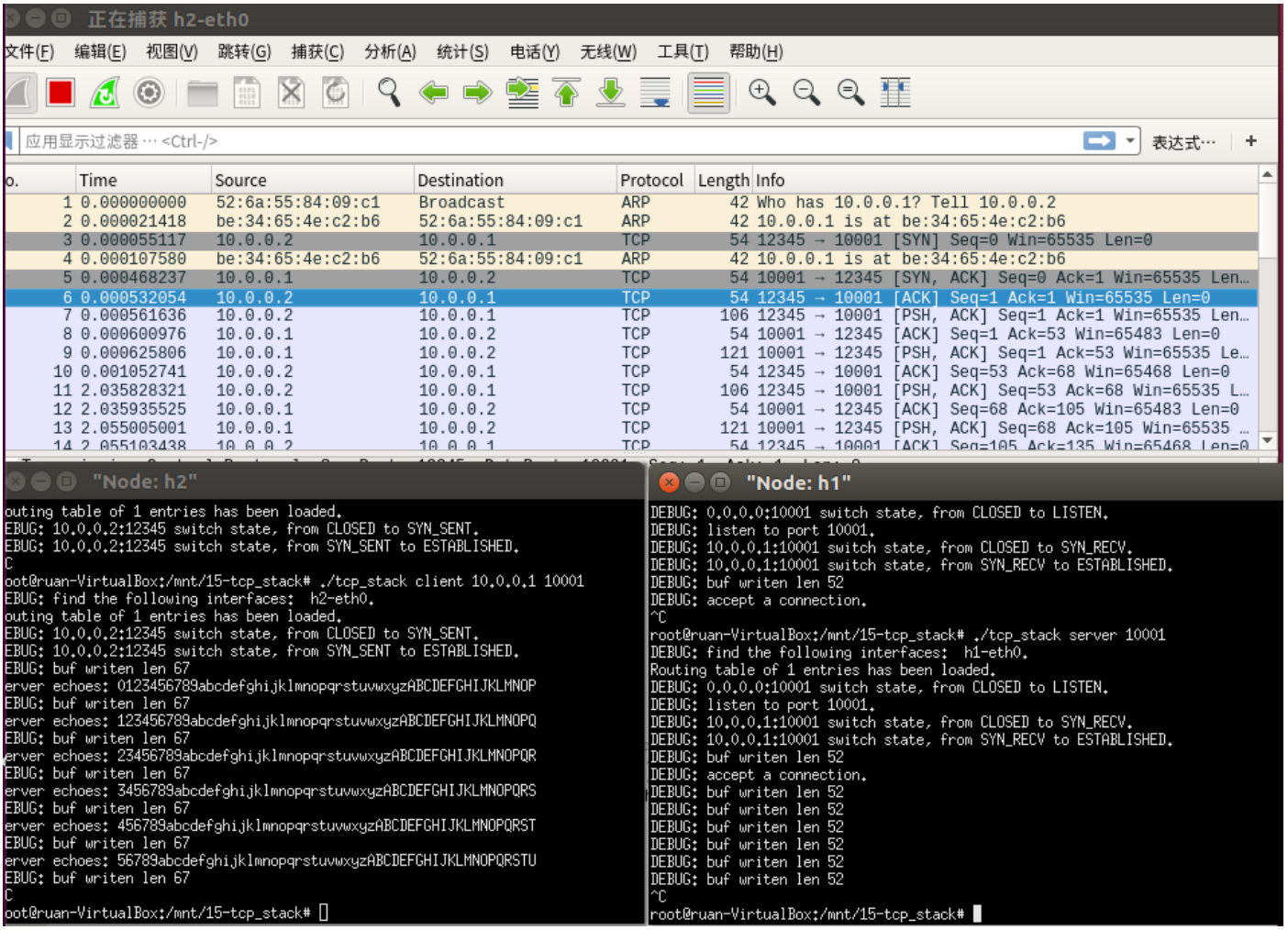
```
int tcp_sock_write(struct tcp_sock *tsk, char *buf, int len)
{
    int len_left = len;
    //log(DEBUG,"try write");
    while(len_left > 0)
    {
        int data_len = min(len_left, min(1500, tsk->snd_wnd) - ETHER_HDR_SIZE - IP_BASE_HDR_SIZE - TCP_BASE_HDR_SIZE);
        while(data_len <= 0)
        {
            log(DEBUG,"send window full, sleep 1s\n");
            sleep(1);
            data_len = min(len_left, min(1500, tsk->snd_wnd) - ETHER_HDR_SIZE - IP_BASE_HDR_SIZE - TCP_BASE_HDR_SIZE);
        }
        char * packet = (char *)malloc(ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE + data_len);
        memcpy(packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE, buf + len - len_left, data_len);
        tcp_send_packet(tsk, packet, ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE + data_len);
        len_left -= data_len;
    }
    //log(DEBUG,"writen");
    return 0;

} « end tcp_sock_write »
```

对于 ring buffer 的锁，我选择了直接构建在 ring buffer 的数据结构中

```
struct ring_buffer {
    int size;
    int head;       // read from head
    int tail;       // write from tail
    pthread_mutex_t lock;
    char buf[0];
};
```

这样能够方便并行处理，持有和释放锁都放在了 read_ring_buffer 和 write_ring_buffer 中，这样在外面调用时便不必考虑锁的问题，更加便捷。

四、 实验结果



可见上图，tcp 连接正确地建立以及传输数据，结果符合预期。

五、 实验总结

本次实验代码量很少，实现简单，理解容易，只是由于之前没有注意 read_ring_buffer 的代码，没有注意到 read 是允许后面的数字超过已收到的数据长度的，导致了一些小 bug。另外在之前实现的时候 wake_up(wait_accept)的调用出现了一点点错误，应该调 parent 的 wait 而不是自己的，这在实验一中没有展现出来。在解决了这些问题后，便得到了预期的结果。