哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2023 春季

课程名称：计算机网络

实验名称：附加题：简易 TCP 协议

学生班级：5 班

学生学号：200110513

学生姓名：宗晴

评阅教师：

报告成绩：

实验与创新实践教育中心制

2023 年 5 月

一、 实验详细设计

注意不要完全照搬实验指导书上的内容，请根据你自己的设计方案来填写

1. TCP 协议详细设计

在本实验中，我们需要完成 TCP 协议的设计与实现，具体而言，即实现 src/tcp.c 文件中的 tcp_in()函数，该函数的功能是实现服务器端的 TCP 收包。

算法实现如下所示，按照注释进行代码的编写：

在实现具体算法前，首先需要在函数的最后定义 reset_tcp 和 close_tcp 这两个标志对应的操作：

```
reset_tcp:
    printf("!!! reset tcp !!!\n");
    connect->next_seq = 0;
    connect->ack = get_seq + 1;
    buf_init(&txbuf, 0);
    tcp_send(&txbuf, connect, tcp_flags_ack_rst);

close_tcp:
    release_tcp_connect(connect);
    map_delete(&connect_table, &key);
    return;
```

然后实现具体算法的主体部分。

首先，需要进行大小检查，即检查 buf 长度是否小于 tcp 头部，如果是，则丢弃：

```
if(buf->len < sizeof(tcp_hdr_t)) return;
```

然后，需要检查 checksum 字段，如果 checksum 出错，则丢弃。注意此处在检查之前需要首先取出原 checksum 值，然后将该字段置零，再计算新的 checksum 值与原值进行比较。

```
tcp_hdr_t* tcp_hdr = (tcp_hdr_t*)buf->data;
uint16_t checksum = tcp_hdr->chunksum16;
tcp_hdr->chunksum16 = 0;

if(checksum != tcp_checksum(buf, src_ip, net_if_ip)) return;
tcp_hdr->chunksum16 = checksum;
```

接着，需要从 tcp 头部字段中获取 source port、destination port、sequence number、acknowledge number、flags，此处需要注意大小端转换：

```
uint16_t src_port = swap16(tcp_hdr->src_port16);
uint16_t dst_port = swap16(tcp_hdr->dst_port16);
uint32_t get_seq  = swap32(tcp_hdr->seq_number32);
uint32_t ack_num  = swap32(tcp_hdr->ack_number32);
tcp_flags_t flags = tcp_hdr->flags;
```

调用 map_get 函数，根据 destination port 查找对应的 handler 函数：

```
tcp_handler_t* handler = map_get(&tcp_table, &dst_port);
if(handler == NULL) return;
```

调用 new_tcp_key 函数，根据通信五元组中的源 IP 地址、目标 IP 地址、目标端口号确定一个 tcp 链接 key：

```
tcp_key_t key = new_tcp_key(src_ip, src_port, dst_port);
```

调用 map_get 函数，根据 key 查找一个 tcp_connect_t* connect，如果没有找到，则调用 map_set 建立新的链接，并设置为 CONNECT_LISTEN 状态，然后调用 mag_get 获取到该链接：

```
tcp_connect_t* connect = map_get(&connect_table, &key);
if(connect == NULL) {
    tcp_connect_t* new_connect = (tcp_connect_t* )malloc(sizeof(tcp_connect_t));
    *new_connect = CONNECT_LISTEN;
    map_set(&connect_table, &key, new_connect);
    connect = map_get(&connect_table, &key);
}
```

从 TCP 头部字段中获取对方的窗口大小，此处需要注意大小端转换：

```
uint16_t window_size = swap16(tcp_hdr->window_size16);
```

接下来，判断是否是 TCP_LISTEN 状态，若是，则需要完成如下功能：（1）如果收到的 flag 带有 rst，则 close_tcp 关闭 tcp 链接；（2）如果收到的 flag 不是 syn，则 reset_tcp 复位通知。因为收到的第一个包必须是 syn；（3）调用 init_tcp_connect_rcvd 函数，初始化 connect，将状态设为 TCP_SYN_RCVD；（4）填充 connect 字段，包括 local_port、remote_port、ip、unack_seq（设为随机值）、由于是对 syn 的 ack 应答包，next_seq 与 unack_seq 一致 ack 设为对方的 sequence number+1，设置 remote_win 为对方的窗口大小，此处需要注意大小端转换；（5）调用 buf_init 初始化 txbuf；（6）调用 tcp_send 将 txbuf 发送出去，也就是回复一个 tcp_flags_ack_syn（SYN+ACK）报文；（7）处理结束，返回。

```
if(connect->state == TCP_LISTEN) {
    if(flags.rst) goto close_tcp;

    if(!flags.syn) goto reset_tcp;

    init_tcp_connect_rcvd(connect);
    connect->local_port = dst_port;
    connect->remote_port = src_port;
    memcpy(connect->ip, src_ip, NET_IP_LEN);
    srand(time(NULL) + dst_port);
    connect->unack_seq = rand() % UINT16_MAX;
    connect->next_seq = connect->unack_seq;
    connect->ack = get_seq + 1;
    connect->remote_win = window_size;
    buf_init(&txbuf, 0);
    tcp_send(&txbuf, connect, tcp_flags_ack_syn);

    return;
}
```

然后，检查接收到的 sequence number，如果与 ack 序号不一致,则 reset_tcp 复位通知：

```
if(get_seq != connect->ack) goto reset_tcp;
```

检查 flags 是否有 rst 标志，如果有，则 close_tcp 连接重置

```
if(flags.rst) goto close_tcp;
```

序号相同时的处理，调用 buf_remove_header 去除头部后剩下的都是数据

```
buf_remove_header(buf, sizeof(tcp_hdr_t));
```

最后进行状态的转换，利用 switch 实现：

```
switch (connect->state) {
```

若状态为 TCP_LISTEN，则报错：

```
case TCP_LISTEN:
    panic("switch TCP_LISTEN", __LINE__);
    break;
```

若状态为 TCP_SYN_RCVD：如果收到的包没有 ack flag，则不做任何处理；如果是 ack 包，需要完成（1）将 unack_seq +1；（2）将状态转成 ESTABLISHED；（3）调用回调函数，完成三次握手，进入连接状态

TCP_CONN_CONNECTED。实现如下图所示：

```
case TCP_SYN_RCVD:

    /*
    12、在RCVD状态，如果收到的包没有ack flag，则不做任何处理
    */
    if(!flags.ack) break;

    /*
    13、如果是ack包，需要完成如下功能:
        (1) 将unack_seq +1
        (2) 将状态转成ESTABLISHED
        (3) 调用回调函数，完成三次握手，进入连接状态TCP_CONN_CONNECTED。
    */
    connect->unack_seq++;
    connect->state = TCP_ESTABLISHED;
    (*handler)(connect, TCP_CONN_CONNECTED);
    break;
```

若状态为 TCP_ESTABLISHED：如果收到的包没有 ack 且没有 fin 这两个标志，则不做任何处理；如果是 ack 包，且 unack_seq 小于 ack number（说明有部分数据被对端接收确认了，否则可能是之前重发的 ack，可以不处理），且 next_seq 大于 ack number，则调用 buf_remove_header 函数，去掉被对端接收确认的部分数据，并更新 unack_seq 值：

```
case TCP_ESTABLISHED:

    /*
    14、如果收到的包没有ack且没有fin这两个标志，则不做任何处理
    */
    if(!flags.ack && !flags.fin) break;

    /*
    15、这里先处理ACK的值，
        如果是ack包，
        且unack_seq小于ack number（说明有部分数据被对端接收确认了，否则可能是之前重发的ack，可以不处理），
        且next_seq大于ack number
        则调用buf_remove_header函数，去掉被对端接收确认的部分数据，并更新unack_seq值

    */
    if(flags.ack && connect->unack_seq < ack_num && connect->next_seq > ack_num) {
            buf_remove_header(connect->tx_buf, ack_num - connect->unack_seq);
            connect->unack_seq = ack_num;
    }
```

然后接收数据调用 tcp_read_from_buf 函数，把 buf 放入 rx_buf 中：

```
tcp_read_from_buf(connect, buf);
```

然后，根据当前的标志位进一步处理：（1）首先调用 buf_init 初始化 txbuf；（2）判断是否收到关闭请求（FIN），如果是，将状态改为 TCP_LAST_ACK，ack +1，再发送一个 ACK + FIN 包，并退出，这样就无需进入 CLOSE_WAIT，直接等待对方的 ACK；（3）如果不是 FIN，则看看是否有数据，如果有，

则发 ACK 相应，并调用 handler 回调函数进行处理；（4）调用 tcp_write_to_buf 函数，看看是否有数据需要发送，如果有，同时发数据和 ACK；（5）没有收到数据，可能对方只发一个 ACK，可以不响应

```
    buf_init(&txbuf, 0);
    if(flags.fin) {
        connect->state = TCP_LAST_ACK;
        connect->ack++;
        tcp_send(&txbuf, connect, tcp_flags_ack_fin);
    }
    else if(buf->len > 0){
        (*handler)(connect, TCP_CONN_DATA_RECV);
        tcp_write_to_buf(connect, &txbuf);
        tcp_send(&txbuf, connect, tcp_flags_ack);
    }

    break;
```

若状态为 TCP_CLOSE_WAIT，则报错：

```
case TCP_CLOSE_WAIT:
    panic("switch TCP_CLOSE_WAIT", __LINE__);
    break;
```

若状态为 TCP_FIN_WAIT_1：如果收到 FIN && ACK，则 close_tcp 直接关闭 TCP；如果只收到 ACK，则将状态转为 TCP_FIN_WAIT_2

```
case TCP_FIN_WAIT_1:

    /*
    18、如果收到FIN && ACK, 则close_tcp直接关闭TCP
        如果只收到ACK, 则将状态转为TCP_FIN_WAIT_2
    */
    if(flags.fin && flags.ack) goto close_tcp;
    if(flags.ack) connect->state = TCP_FIN_WAIT_2;
    break;
```

若状态为 TCP_FIN_WAIT_2：如果不是 FIN，则不做处理；如果是，则将 ACK +1，调用 buf_init 初始化 txbuf，调用 tcp_send 发送一个 ACK 数据包，再 close_tcp 关闭 TCP

```
case TCP_FIN_WAIT_2:
    /*
    19、如果不是FIN，则不做处理
        如果是，则将ACK +1，调用buf_init初始化txbuf，调用tcp_send发送一个ACK数据包，再close_tcp关闭TCP
    */
    if(flags.fin) {
        connect->ack++;
        buf_init(&txbuf, 0);
        tcp_send(&txbuf, connect, tcp_flags_ack);
        goto close_tcp;
    }
    break;
```

若状态为 TCP_LAST_ACK：如果不是 ACK，则不做处理；如果是，则调用 handler 函数，进入 TCP_CONN_CLOSED 状态，再 close_tcp 关闭 TCP

```
case TCP_LAST_ACK:
    /*
    20、如果不是ACK，则不做处理
        如果是，则调用handler函数，进入TCP_CONN_CLOSED状态，再close_tcp关闭TCP
    */

    if(flags.ack) {
        (*handler)(connect, TCP_CONN_CLOSED);
        goto close_tcp;
    }
    break;
```

其它状态则报错：

```
default:
    panic("connect->state", __LINE__);
    break;
```

最后 return，至此完成了 tcp_in()函数的全部内容。

二、实验结果截图及分析

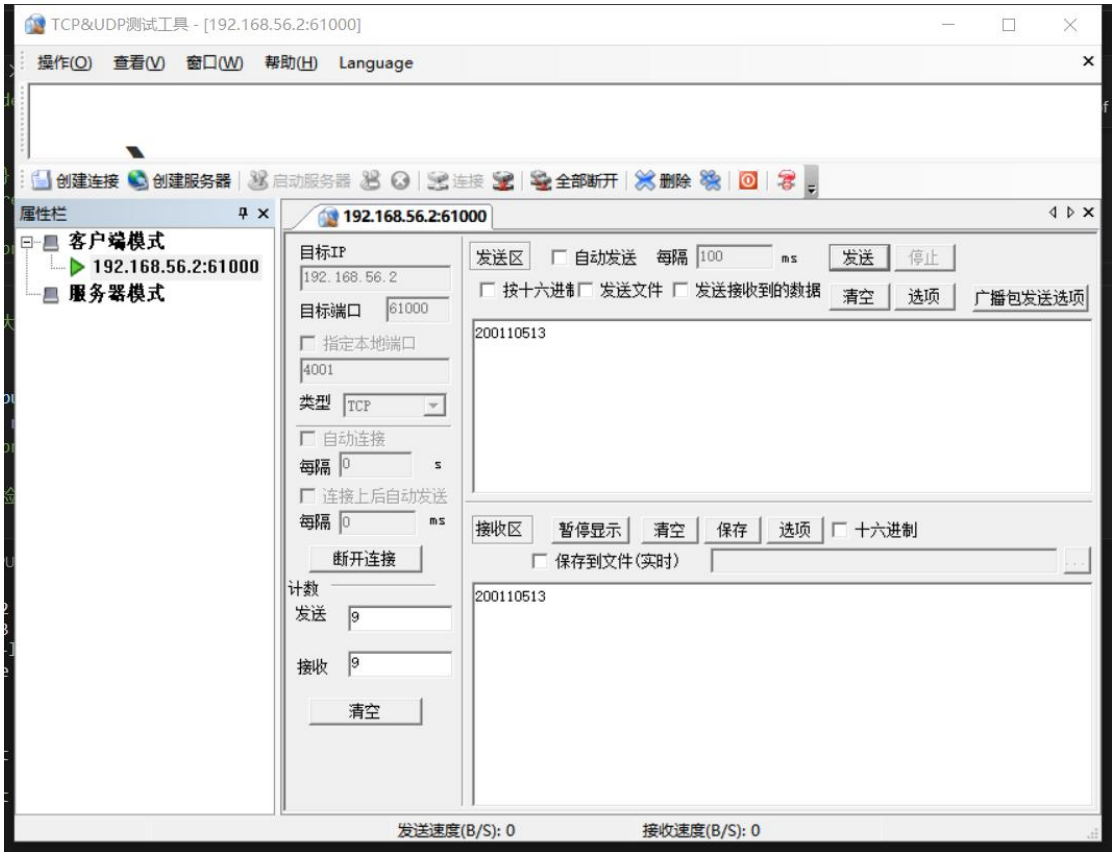
1. TCP 协议实验结果及分析

终端界面如下（展示了建立连接、发送消息、断开连接的过程）：

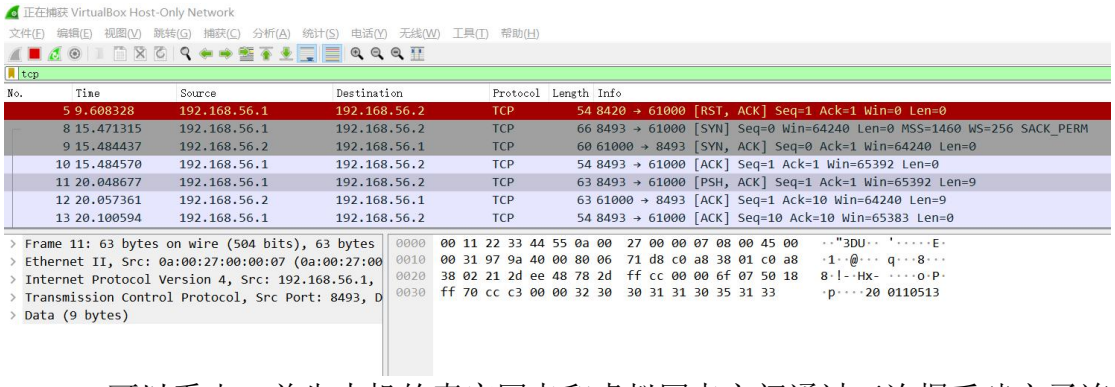
```
PS E:\lab2\net-lab-master\build> ."E:/lab2/net-lab-master/build/main.exe"
Using interface \Device\NPF_{313335BF-2B53-4379-BD77-A2DE650819CD}, my ip is 192.168.56.2.
tcp open
tcp open
flags: ack syn
recv tcp packet from 192.168.56.1:4358 len=0

recv tcp packet from 192.168.56.1:4358 len=9
200110513
flags: ack
flags: ack fin
recv tcp packet from 192.168.56.1:4358 len=0
```

及 TCP&UDP 调试工具界面如下：



抓包结果如下：



可以看出，首先本机的真实网卡和虚拟网卡之间通过三次握手建立了连

接，然后本机的真实网卡向虚拟网卡发送了数据报，数据长度为 9，内容为 200110513，然后虚拟网卡向本机的真实网卡回复了 ACK 信息，符合预期。