《计算机网络协议开发》实验报告

第二次实验： 网络协议源代码阅读实验

姓名： 吕玉龙

学号： 191220076

2019 级计算机科学与技术院/系

邮箱： 1931015836@qq.com

时间：2022.3.12

一、实验目的

通过阅读实现网络协议的源代码，了解网络协议开发中常用的协议结构定义方法、有限状态机实现方法、网络协议实现的一般流程和常用技巧。

二、实验内容

请参照 2.2 节的实验说明，仔细阅读本实验所提供的源代码，为源代码添加详细注释（所
有注释语句均以//开头），注释内容包括：
(1) 为源代码中定义的常量、全局变量、全局数据结构的成员变量添加注释，说明这些变量
的含义。
(2) 为源代码中每个实现的函数添加注释，说明这些函数的功能、其参数的含义、返回值的
含义，并请在注释中简要说明每个函数的大致处理步骤。
(3) 为源代码中的重要语句添加注释。

三、思考问题

1.采用的是停等协议，如下图

```
1
2 int tcp_established(struct MyTCB* pTcb, struct MyTcpSeg* pTcpSeg)//参数为tcp报文段（接受到的tcp报文）以及tcb控制块，由established状态变为FIN_WAIT1状态，或者仍在和客户端进行数据交互
3 {
4       struct MyTcpSeg my_seg;
5
6       if( pTcb == NULL || pTcpSeg == NULL )
7       {
8               return -1;
9       }
0
1       if( pTcb->iotype == INPUT )//此时为接受报文的状态
2       {
3               if( pTcpSeg->seq_num != pTcb->ack )//如果收到的包不是期望接收到的，则丢弃
4               {
5                       tcp_DiscardPkt((char*)pTcpSeg, TCP_TEST_SEQNO_ERROR);
6                       //to do: discard packet
7
8                       return -1;
9               }
0
1               if( (pTcpSeg->flags & 0x3f) == 0x10 )//ACK = 1，复制包中的data内容
2               {
3                       memcpy(pTcb->data, pTcpSeg->data, pTcpSeg->len - 20);
4                       pTcb->data_len = pTcpSeg->len - 20;
5
6                       if( pTcb->data_len == 0 )
7                       {
8                       }
9                       else//复制的内容不为空
0                       {
1                               pTcb->ack += pTcb->data_len;//下次期望收到的ack序列号
2                               pTcb->flags = 0x10;//设置flag
3                               tcp_construct_segment(&my_seg, pTcb, 0, NULL);
4                               tcp_kick(pTcb, &my_seg);//构造tcp报文段并发送
5                       }
6               }
7       }
8       else//为发送报文的状态
9       {
```

看注释，它每次对于收到的包的 seq 会和 tcb 控制块中记录的期望的 acknum 进行比较，如果不对它就会丢弃。这里就和停等协议一样，一定要等到它希望收到的包到来，才会发下一个包。

2.检验和显然提供了。如下图，截图中的函数就是 tcp 校验和函数

```
unsigned short tcp_calc_checksum(struct MyTCB* pTcb, struct MyTcpSeg* pTcpSeg)//计算要发送的包的检验和，第一个参数为了得到源和目的的ip地址，第二个用来计算tcp检验和
{
	int i = 0;
	int len = 0;
	unsigned int sum = 0;
	unsigned short* p = (unsigned short*)pTcpSeg;

	if( pTcb == NULL || pTcpSeg == NULL )
	{
		return 0;
	}

	for( i=0; i<10; i++)
	{
		sum += p[i];
	}//8位一截，累加到urg_ptr，计算tcp首部

	sum = sum - p[8] - p[6] + ntohs(p[6]);//由于结构体中有两个char类型，故这里转换字节序，有网络转主机

	if( (len = pTcpSeg->len) > 20 )//判断这个tcp包中是否有数据
	{
		if( len % 2 == 1 )
		{
			pTcpSeg->data[len - 20] = 0;
			len++;
		}//总长度奇数则末尾补零
```

也为每个字节提供了序列号，如下图：

```
int tcp_kick(struct MyTCB* pTcb, struct MyTcpSeg* pTcpSeg)//参数为tcp报文段以及tcb控制块，ip分组发送，以及更新tcb控制块中的序列号
{
	pTcpSeg->checksum = tcp_calc_checksum(pTcb, pTcpSeg);//检验和赋值

	convert_tcp_hdr_hton(pTcpSeg);//字节序转换

	tcp_sendIpPkt((unsigned char*)pTcpSeg, pTcpSeg->len, pTcb->local_ip, pTcb->remote_ip, 255);//ip分组发送函数
	//以下都是对tcb控制块做得操作，因为tcp包已经发送
	if( (pTcb->flags & 0x0f) == 0x00 )//flag后四位都为0，更改该tcb块的seq（加上数据字段的长度）
	{
		pTcb->seq += pTcpSeg->len - 20;
	}
	else if( (pTcb->flags & 0x0f) == 0x02 )//SYN = 1，发送连接请求（三次握手的第一次）
	{
		pTcb->seq++;//发送的序列号
	}
	else if( (pTcb->flags & 0x0f) == 0x01 )//FIN = 1，请求释放连接
	{
		pTcb->seq++;//发送的序列号
	}
	else if( (pTcb->flags & 0x3f) == 0x10 )//ACK = 1，接受，发送ack包或者连接建立之后发送报文，当然我觉得这里逻辑有问题，这个分
	{
	}

	return 0;
```

这个 tcp_kick 函数中 if((pTcb->flags & 0x0f) == 0x00)下一行，实际上对每一个要发出的包的序列号都是依赖上一个包的序列号以及它的长度而构造的，等同于我们对每个字节都分配了序列号。

这个实现有确认机制，但是没有重传机制，因为它采用的是停等协议，然而如果客户端发送的包丢失了，这样服务端就一直不会 ack，故客户端就会卡在这里（因为没有超时重传机制）

3.代码中的错误。
我个人觉得有两个地方有问题，截图在下：
第一个在 tcp_kick 函数中，最后一个分支永远进不去，原因写在注释中

```
int tcp_kick(struct MyTCB* pTcb, struct MyTcpSeg* pTcpSeg)//参数为tcp报文段以及tcb控制块，ip分组发送，以及更新tcb控制块中的序列号
{
        pTcpSeg->checksum = tcp_calc_checksum(pTcb, pTcpSeg);//检验和赋值

        convert_tcp_hdr_hton(pTcpSeg);//字节序转换

        tcp_sendIpPkt((unsigned char*)pTcpSeg, pTcpSeg->len, pTcb->local_ip, pTcb->remote_ip, 255);//ip分组发送函数
        //以下都是对tcb控制块做得操作，因为tcp包已经发送
        if( (pTcb->flags & 0x0f) == 0x00 )//flag后四位都为0，更改该tcb块的seq（加上数据字段的长度）
        {
                pTcb->seq += pTcpSeg->len - 20;
        }
        else if( (pTcb->flags & 0x0f) == 0x02 )//SYN = 1，发送连接请求（三次握手的第一次）
        {
                pTcb->seq++;//发送的序列号
        }
        else if( (pTcb->flags & 0x0f) == 0x01 )//FIN = 1，请求释放连接
        {
                pTcb->seq++;//发送的序列号
        }
        else if( (pTcb->flags & 0x3f) == 0x10 )//ACK = 1，接受，发送ack包或者连接建立之后发送报文，当然我觉得这里逻辑有问题，这个分支应该是永远不会被覆盖的，因为走到这里flag要为010000，但它会进入第一个分支
        {
        }

        return 0;
}
```

第二个问题：

```
int tcp_finwait_2(struct MyTCB* pTcb, struct MyTcpSeg* pTcpSeg)//参数为tcp报文段（接受到的tcp报文）以及tcb控制块，由FIN_WAIT2状态变为CLOSE状态
{
        struct MyTcpSeg my_seg;

        if( pTcb == NULL || pTcpSeg == NULL )
        {
                return -1;
        }

        if( pTcb->iotype != INPUT )//此时应该为接收状态
        {
                return -1;
        }

        if( pTcpSeg->seq_num != pTcb->ack )//不是期望应该接收的包
        {
                tcp_DiscardPkt((char*)pTcpSeg, TCP_TEST_SEQNO_ERROR);

                return -1;
        }

        if( (pTcpSeg->flags & 0x0f) == 0x01 )//检测收到的包FIN是否为1
        {
                pTcb->ack++;
                pTcb->flags = 0x10;

                tcp_construct_segment( &my_seg, pTcb, 0, NULL );//构造ack包（四次挥手中最后一次）
                tcp_kick( pTcb, &my_seg );//发送
                pTcb->current_state = CLOSED;//tcb状态进入close，不知道为啥不进入time_wait状态
        }
        else
        {
                //to do
        }

        return 0;
}
```

这里 tcp_finwait_2 函数，最后直接进入了 close 状态，但是根据手册，它需要先进入 time_wait 状态，等待服务端发送 FIN 包。

附：总用时 5.5h（报告 1h，其它 4.5h）