《计算机网络协议开发》实验报告

第 13 次实验

STCP 协议:GBN 和校验和实现

姓名：元玉慧

学号：101220151

10 级 计算机 系 4 班

邮箱：njucsyyh@gmail.com

时间：2013/05/31

一、 实验目的

通过本实验熟悉控制层数据传输协议的设计，实现。熟悉滑动窗口协议和校验和计算。通过完成本实验可以掌握传输控制层协议的核心组件的设计和实现，包括支持并发连接，支持按序可靠的字节流，能够处理数据包的丢失和损坏。

二、实验设计背景

从 STCB 的数据传输的设计和 GBN 设计：

-1-数据传输阶段

STCP 建立连接后，客户端可以使用 STCP 的 API 函数 stcp_client_send() 发送数据给服务器，以及服务器用 stcp_client_recv()函数接收数据，这些 API 只有在客户端和服务器处于 CONNECTED 状态下才可以有效

stcp_client_send()：

是一个非阻塞函数调用，它在数据被传递到发送缓冲区后就返回， 每一个客户端 TCB 维护一个发送缓冲区来存储待发送的数据，STCP 使用 GBN 数据传输机制。

stcp_client_recv()：

发送缓冲区是一个包含所有未确认数据段的链表，包含在段中数据来自 stcp_client_send()函数中调用。发送缓冲区链表中的段按顺序逐个发送到网络中。

STCP 的 GBN 机制要求任一时刻在发送缓冲区中，最多只能由 GBN_WINDOW 数量的发送但未被确认的 segment 存在，客户端发送数据报文，服务器响应 DATAACK 控制报文，当客户端接收到该报文后，被确认的段就从发送缓冲区中删除掉。

每一个客户端的 TCB 都维护一个自己唯一的发送缓冲区。

客户端 TCB 结构中：

Next_seqNum 包含要与新的数据段相关联的下一个序号，当一个新的数据段被添加到缓冲区中是，它使用 next_seqNum 作为他的序号，然后 next_seqNum 将增加数据段的数据长度。

BufMutex 是一个指向互斥结构的指针，它用于控制对于发送缓冲区的访问，因为可能会有多个线程同时访问发送缓冲区，所以我们需要使用互斥来解决同步问题，我们可以在 stcp_client_sock()中初始化 TCB，动态分配这个互斥结构。

UnAck_segNum 是已经发送但还没有被服务器确认的段的数量。

发送缓冲区的生存期：

每一个客户端 TCB 维护一个唯一的发送缓冲区，当 stcp_client_sock()初始化一个客户端 TCB 时，发送缓冲区设置为空，同时动态创建一个互斥结构。

当 stcp_client_send()被调用的时候，传递给该函数的数据用于创建新的段，这些新段的类型为 DATA, 并被封装进 segBuf 结构，这些 segBuf 然后被附加到发送缓冲区中，此时我们应从第一个未被发送的段开始发送，知道已经发送但是还没有确认的段的数量达到 GBN_WINDOW 为止，当发送缓冲区中的一个段被发送出去时，它的发送时间就记录在它的 seg_Buf 的 sentTime 字段。

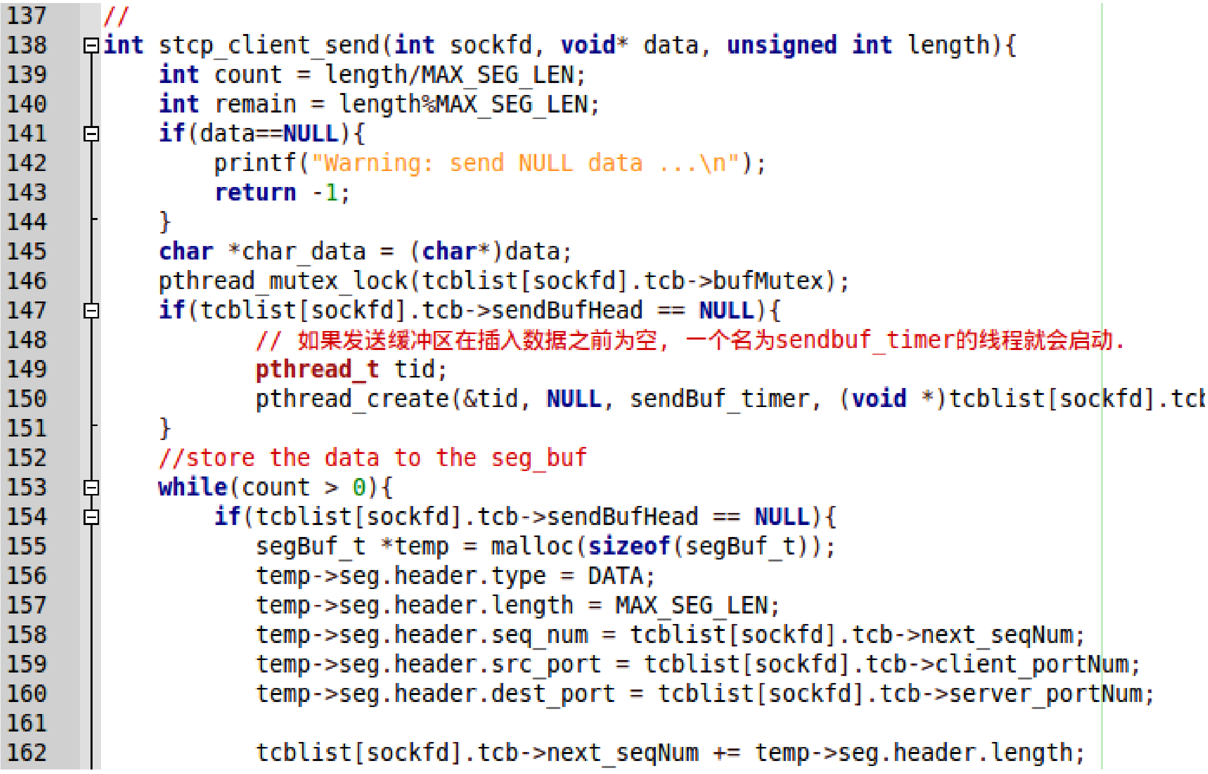
（seghandler 中实习）
当发送缓冲区所有的数据段都被确认后，sendBuf_timer 线程将会终止自己。
当一个 DATAACK 段被客户端的 seghandler 接收到以后，被确认的段，就从发送缓冲区中删除并释放，注意，当 DATAACK 段中的序号，是服务器期望接收的下一个数据段的序号。

接收缓冲区的数据结构
接收缓冲区是一个有固定长度的字节数组，它用于保存提取自书 udande 接收数据。
每个服务器 TCB 维护一个唯一的接收缓冲区。
Expect_seqNum 包含服务器期望从客户端接收到的下一个数据段的序号，当服务器的 seghandler 接收到来自客户端的一个新的数据段，并其序号与 expect_seqNum.

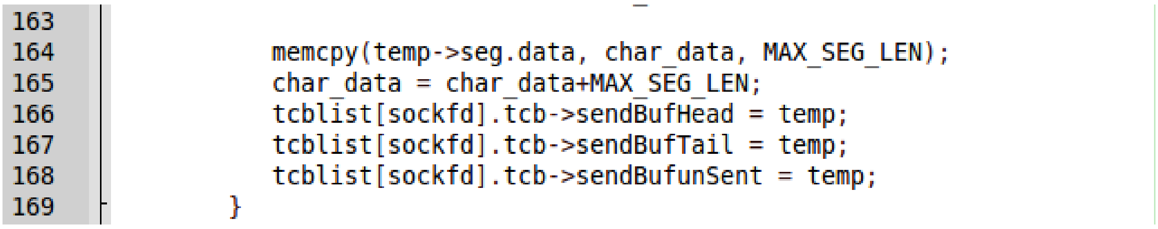
三、实验内容

简单 STCP 的发送数据的函数：

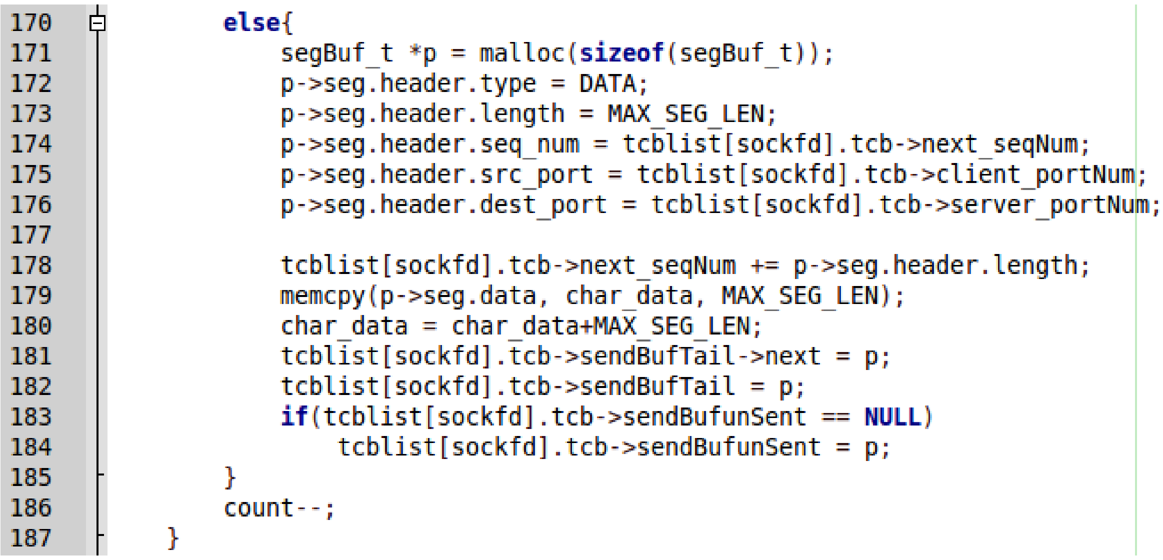
int stcp_client_send() 的实现：



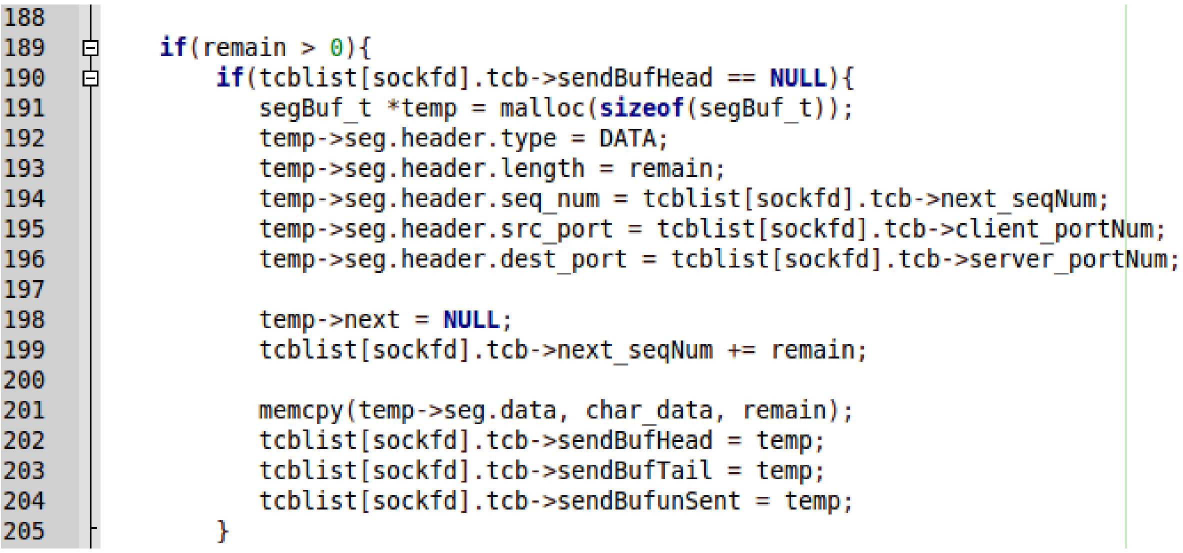
```
//
int stcp_client_send(int sockfd, void* data, unsigned int length){
    int count = length/MAX_SEG_LEN;
    int remain = length%MAX_SEG_LEN;
    if(data==NULL){
        printf("Warning: send NULL data ...\n");
        return -1;
    }
    char *char_data = (char*)data;
    pthread_mutex_lock(tcblist[sockfd].tcb->bufMutex);
    if(tcblist[sockfd].tcb->sendBufHead == NULL){
           // 如果发送缓冲区在插入数据之前为空，一个名为sendbuf_timer的线程就会启动.
           pthread_t tid;
           pthread_create(&tid, NULL, sendBuf_timer, (void *)tcblist[sockfd].tcb
    }
    //store the data to the seg_buf
    while(count > 0){
        if(tcblist[sockfd].tcb->sendBufHead == NULL){
           segBuf_t *temp = malloc(sizeof(segBuf_t));
           temp->seg.header.type = DATA;
           temp->seg.header.length = MAX_SEG_LEN;
           temp->seg.header.seq_num = tcblist[sockfd].tcb->next_seqNum;
           temp->seg.header.src_port = tcblist[sockfd].tcb->client_portNum;
           temp->seg.header.dest_port = tcblist[sockfd].tcb->server_portNum;

           tcblist[sockfd].tcb->next_seqNum += temp->seg.header.length;
```



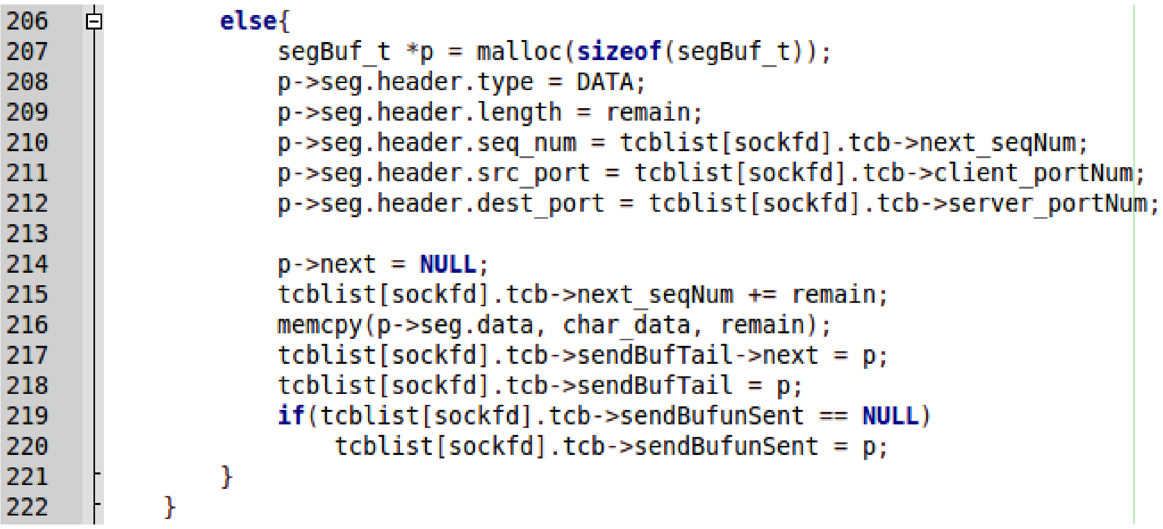
```

           memcpy(temp->seg.data, char_data, MAX_SEG_LEN);
           char_data = char_data+MAX_SEG_LEN;
           tcblist[sockfd].tcb->sendBufHead = temp;
           tcblist[sockfd].tcb->sendBufTail = temp;
           tcblist[sockfd].tcb->sendBufunSent = temp;
        }
```



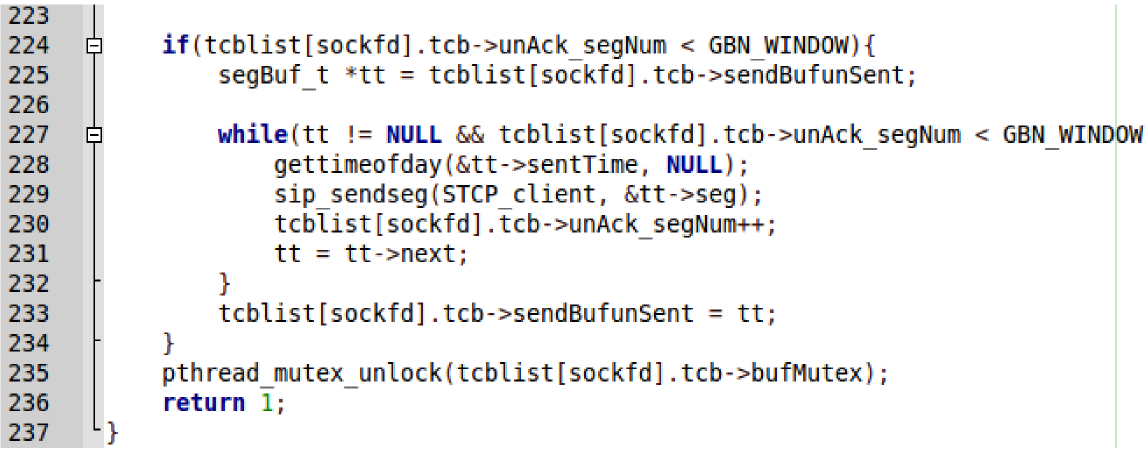
```
        else{
            segBuf_t *p = malloc(sizeof(segBuf_t));
            p->seg.header.type = DATA;
            p->seg.header.length = MAX_SEG_LEN;
            p->seg.header.seq_num = tcblist[sockfd].tcb->next_seqNum;
            p->seg.header.src_port = tcblist[sockfd].tcb->client_portNum;
            p->seg.header.dest_port = tcblist[sockfd].tcb->server_portNum;

            tcblist[sockfd].tcb->next_seqNum += p->seg.header.length;
            memcpy(p->seg.data, char_data, MAX_SEG_LEN);
            char_data = char_data+MAX_SEG_LEN;
            tcblist[sockfd].tcb->sendBufTail->next = p;
            tcblist[sockfd].tcb->sendBufTail = p;
            if(tcblist[sockfd].tcb->sendBufunSent == NULL)
                tcblist[sockfd].tcb->sendBufunSent = p;
        }
        count--;
    }
```



```

    if(remain > 0){
        if(tcblist[sockfd].tcb->sendBufHead == NULL){
           segBuf_t *temp = malloc(sizeof(segBuf_t));
           temp->seg.header.type = DATA;
           temp->seg.header.length = remain;
           temp->seg.header.seq_num = tcblist[sockfd].tcb->next_seqNum;
           temp->seg.header.src_port = tcblist[sockfd].tcb->client_portNum;
           temp->seg.header.dest_port = tcblist[sockfd].tcb->server_portNum;

           temp->next = NULL;
           tcblist[sockfd].tcb->next_seqNum += remain;

           memcpy(temp->seg.data, char_data, remain);
           tcblist[sockfd].tcb->sendBufHead = temp;
           tcblist[sockfd].tcb->sendBufTail = temp;
           tcblist[sockfd].tcb->sendBufunSent = temp;
        }
```

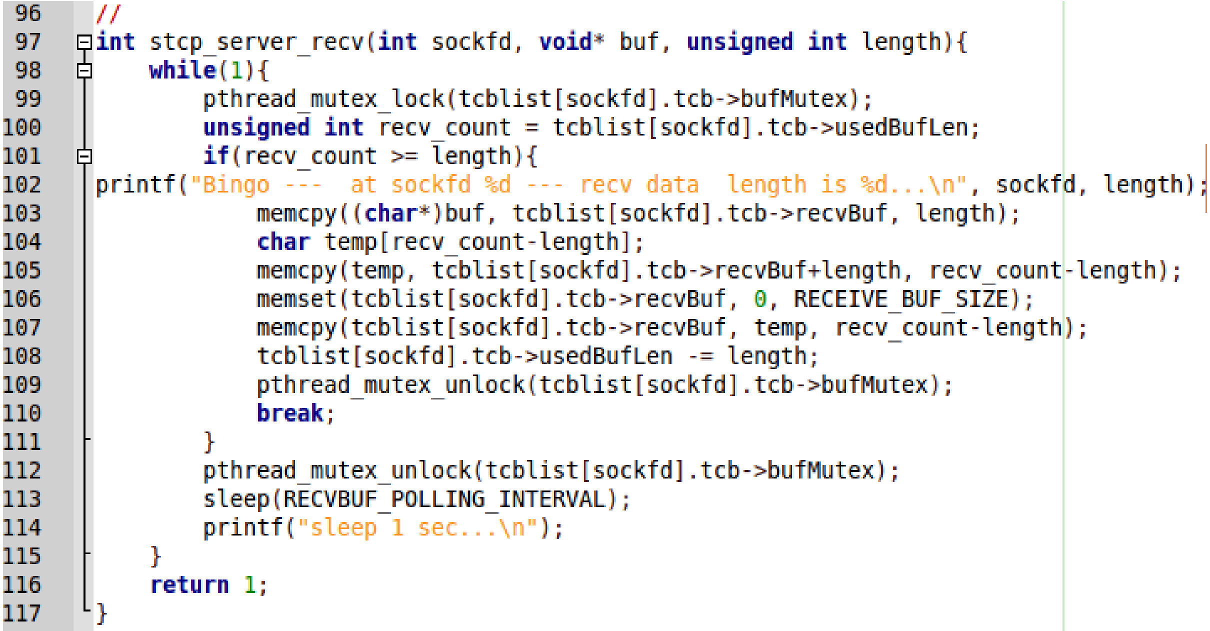


```
        else{
            segBuf_t *p = malloc(sizeof(segBuf_t));
            p->seg.header.type = DATA;
            p->seg.header.length = remain;
            p->seg.header.seq_num = tcblist[sockfd].tcb->next_seqNum;
            p->seg.header.src_port = tcblist[sockfd].tcb->client_portNum;
            p->seg.header.dest_port = tcblist[sockfd].tcb->server_portNum;

            p->next = NULL;
            tcblist[sockfd].tcb->next_seqNum += remain;
            memcpy(p->seg.data, char_data, remain);
            tcblist[sockfd].tcb->sendBufTail->next = p;
            tcblist[sockfd].tcb->sendBufTail = p;
            if(tcblist[sockfd].tcb->sendBufunSent == NULL)
                tcblist[sockfd].tcb->sendBufunSent = p;
        }
    }
```



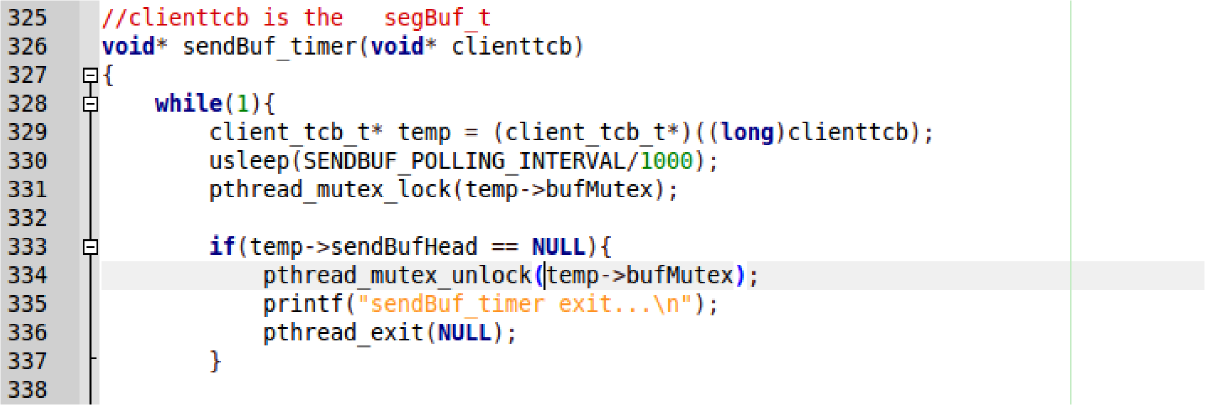
```c

    if(tcblist[sockfd].tcb->unAck_segNum < GBN_WINDOW){
        segBuf_t *tt = tcblist[sockfd].tcb->sendBufunSent;

        while(tt != NULL && tcblist[sockfd].tcb->unAck_segNum < GBN_WINDOW){
            gettimeofday(&tt->sentTime, NULL);
            sip_sendseg(STCP_client, &tt->seg);
            tcblist[sockfd].tcb->unAck_segNum++;
            tt = tt->next;
        }
        tcblist[sockfd].tcb->sendBufunSent = tt;
    }
    pthread_mutex_unlock(tcblist[sockfd].tcb->bufMutex);
    return 1;
}
```

int stcp_server_send() 的实现

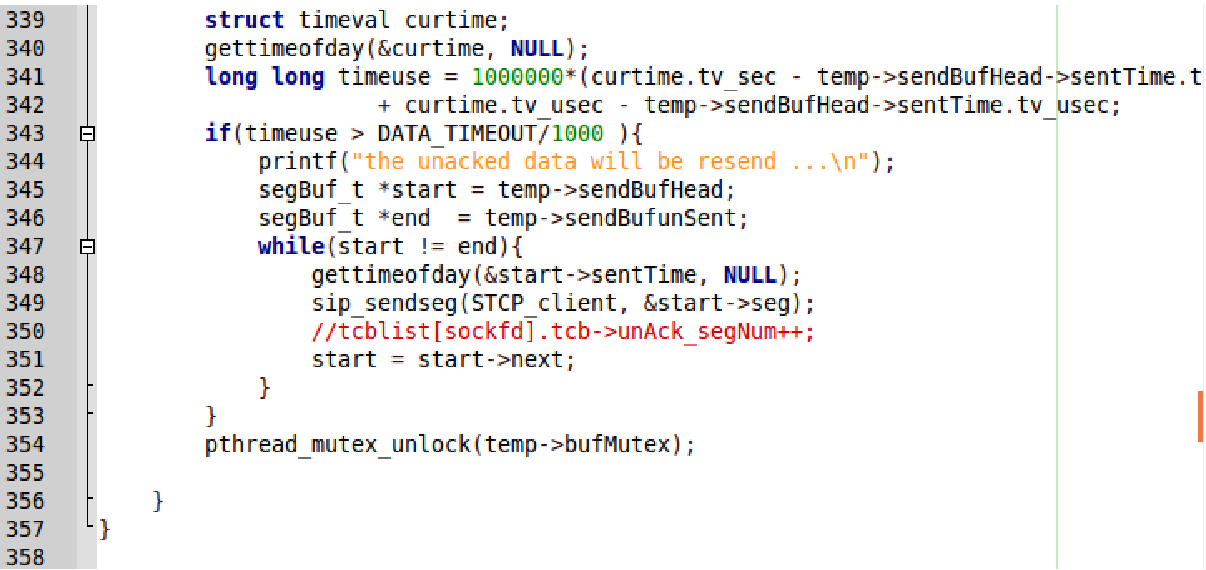


```c
//
int stcp_server_recv(int sockfd, void* buf, unsigned int length){
    while(1){
        pthread_mutex_lock(tcblist[sockfd].tcb->bufMutex);
        unsigned int recv_count = tcblist[sockfd].tcb->usedBufLen;
        if(recv_count >= length){
printf("Bingo ---  at sockfd %d --- recv data  length is %d...\n", sockfd, length);
            memcpy((char*)buf, tcblist[sockfd].tcb->recvBuf, length);
            char temp[recv_count-length];
            memcpy(temp, tcblist[sockfd].tcb->recvBuf+length, recv_count-length);
            memset(tcblist[sockfd].tcb->recvBuf, 0, RECEIVE_BUF_SIZE);
            memcpy(tcblist[sockfd].tcb->recvBuf, temp, recv_count-length);
            tcblist[sockfd].tcb->usedBufLen -= length;
            pthread_mutex_unlock(tcblist[sockfd].tcb->bufMutex);
            break;
        }
        pthread_mutex_unlock(tcblist[sockfd].tcb->bufMutex);
        sleep(RECVBUF_POLLING_INTERVAL);
        printf("sleep 1 sec...\n");
    }
    return 1;
}
```

Void* sendBuf_timer (void* clienttcb) 的实现

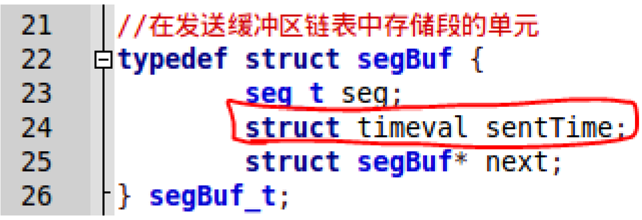


```c
//clienttcb is the   segBuf_t
void* sendBuf_timer(void* clienttcb)
{
    while(1){
        client_tcb_t* temp = (client_tcb_t*)((long)clienttcb);
        usleep(SENDBUF_POLLING_INTERVAL/1000);
        pthread_mutex_lock(temp->bufMutex);

        if(temp->sendBufHead == NULL){
            pthread_mutex_unlock(temp->bufMutex);
            printf("sendBuf_timer exit...\n");
            pthread_exit(NULL);
        }

```



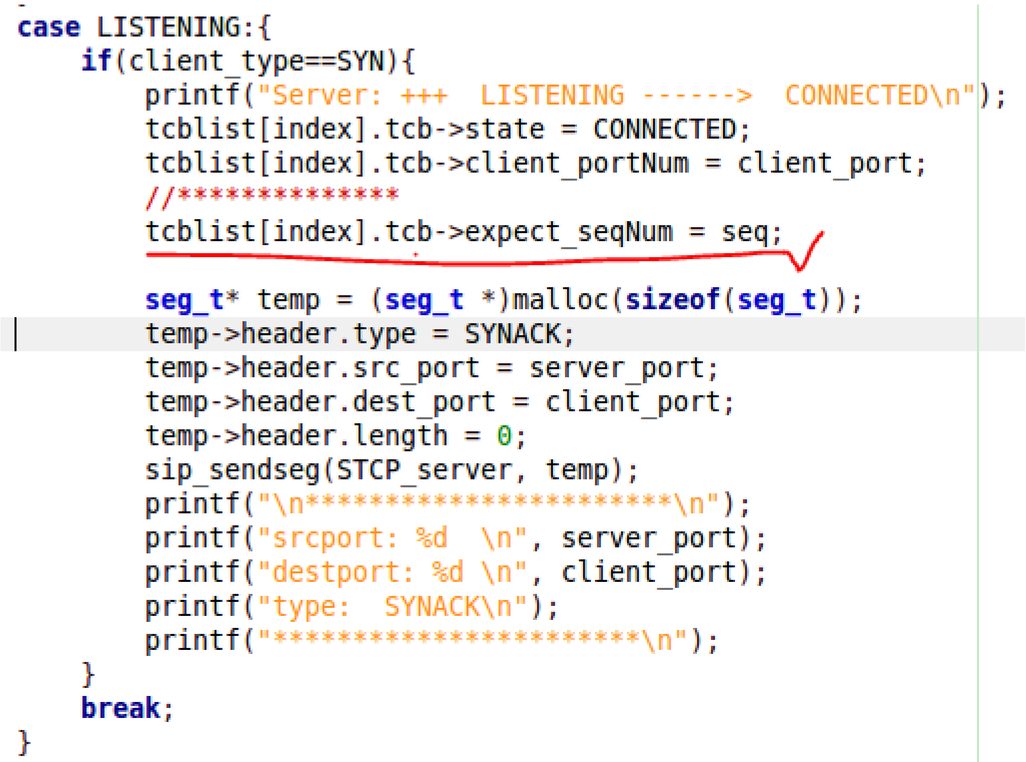
```c
        struct timeval curtime;
        gettimeofday(&curtime, NULL);
        long long timeuse = 1000000*(curtime.tv_sec - temp->sendBufHead->sentTime.t
                     + curtime.tv_usec - temp->sendBufHead->sentTime.tv_usec;
        if(timeuse > DATA_TIMEOUT/1000 ){
            printf("the unacked data will be resend ...\n");
            segBuf_t *start = temp->sendBufHead;
            segBuf_t *end  = temp->sendBufunSent;
            while(start != end){
                gettimeofday(&start->sentTime, NULL);
                sip_sendseg(STCP_client, &start->seg);
                //tcblist[sockfd].tcb->unAck_segNum++;
                start = start->next;
            }
        }
        pthread_mutex_unlock(temp->bufMutex);

    }
}
```

为了计算 sendBuf_timer 线程中的时间间隔，修改过的数据结构：



```c
//在发送缓冲区链表中存储段的单元
typedef struct segBuf {
        seg_t seg;
        struct timeval sentTime;
        struct segBuf* next;
} segBuf_t;
```

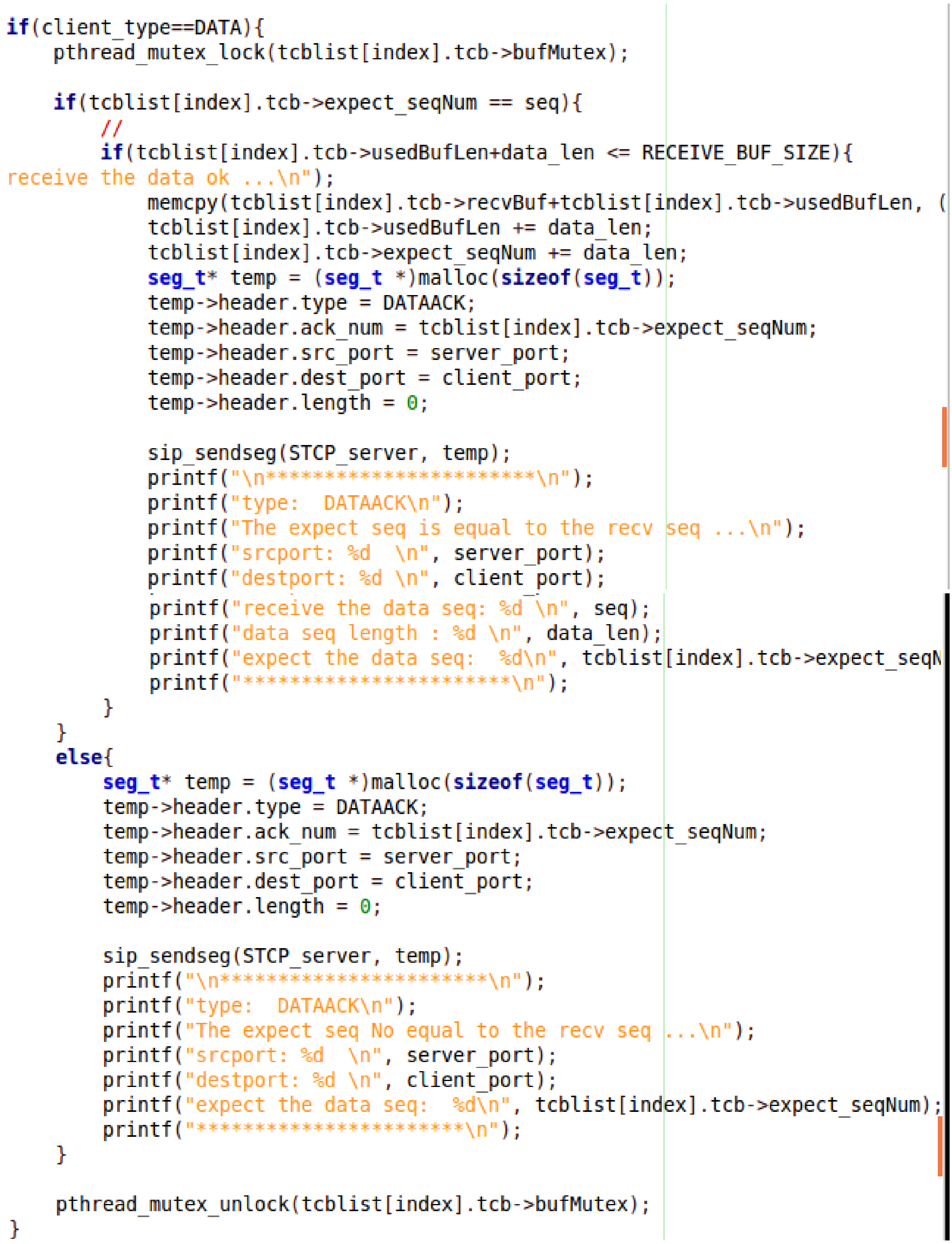
服务器端 seghandler 部分修改代码：



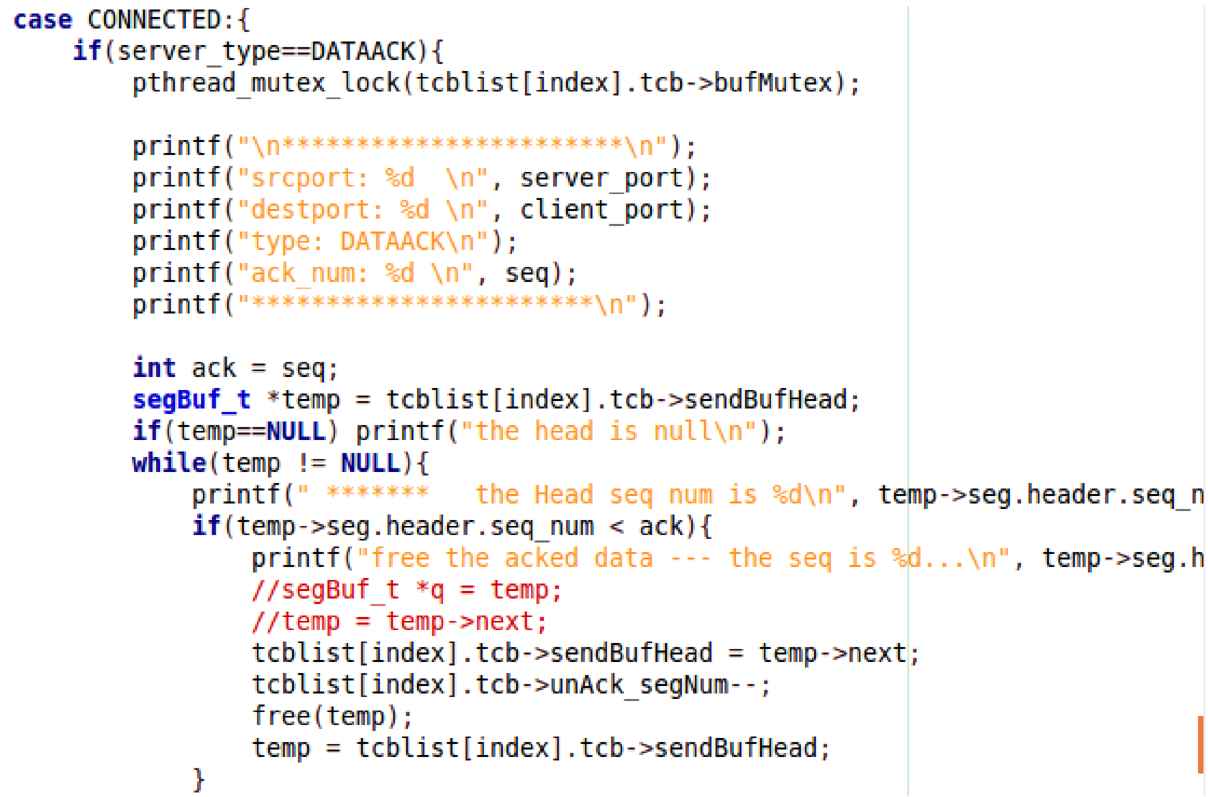
```c
case LISTENING:{
    if(client_type==SYN){
        printf("Server: +++  LISTENING ------>  CONNECTED\n");
        tcblist[index].tcb->state = CONNECTED;
        tcblist[index].tcb->client_portNum = client_port;
        //**************
        tcblist[index].tcb->expect_seqNum = seq;

        seg_t* temp = (seg_t *)malloc(sizeof(seg_t));
        temp->header.type = SYNACK;
        temp->header.src_port = server_port;
        temp->header.dest_port = client_port;
        temp->header.length = 0;
        sip_sendseg(STCP_server, temp);
        printf("\n**********************\n");
        printf("srcport: %d  \n", server_port);
        printf("destport: %d \n", client_port);
        printf("type:  SYNACK\n");
        printf("**********************\n");
    }
    break;
}
```

服务器第一次收到报文后，同步设置报文的序号。

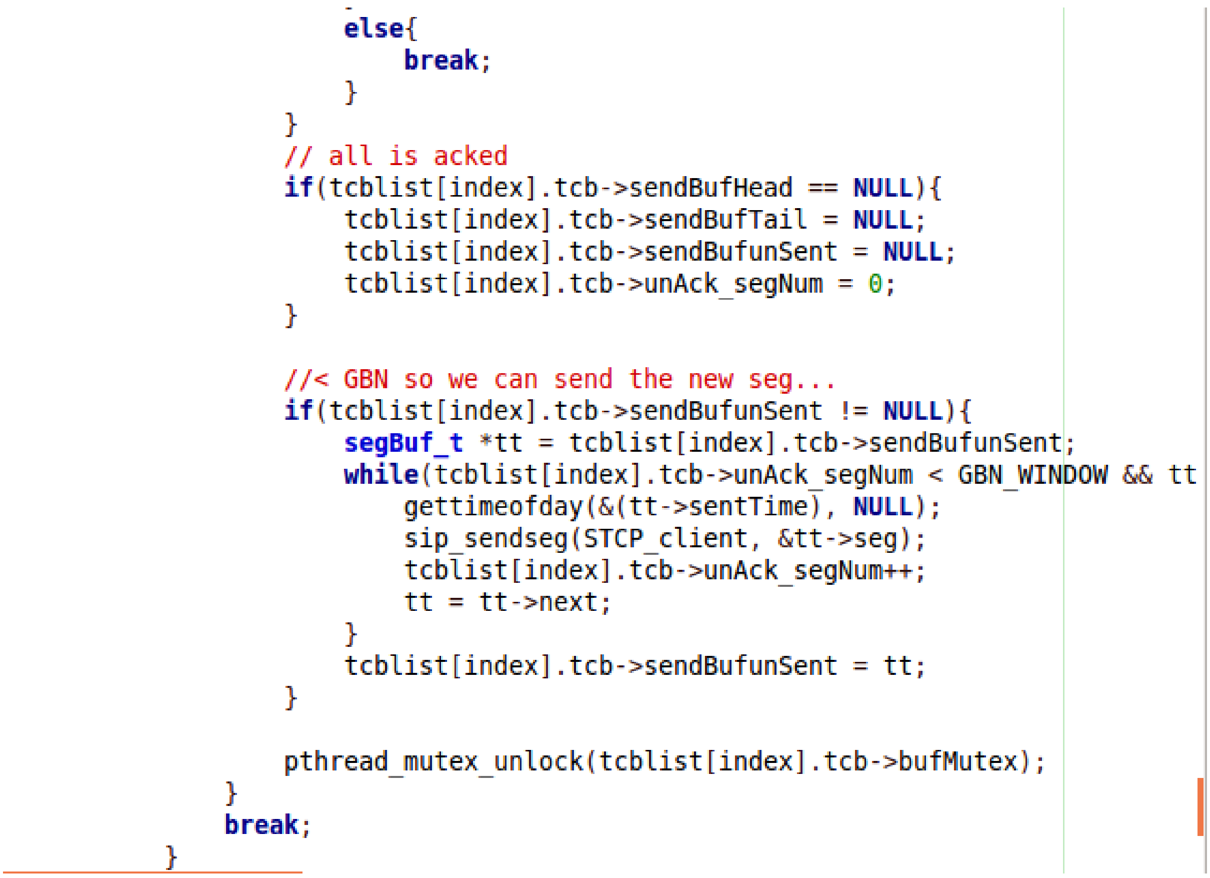
Connected 状态下接收到 DATA 报文的时候了：



```
if(client_type==DATA){
    pthread_mutex_lock(tcblist[index].tcb->bufMutex);

    if(tcblist[index].tcb->expect_seqNum == seq){
        //
        if(tcblist[index].tcb->usedBufLen+data_len <= RECEIVE_BUF_SIZE){
receive the data ok ...\n");
            memcpy(tcblist[index].tcb->recvBuf+tcblist[index].tcb->usedBufLen, (
            tcblist[index].tcb->usedBufLen += data_len;
            tcblist[index].tcb->expect_seqNum += data_len;
            seg_t* temp = (seg_t *)malloc(sizeof(seg_t));
            temp->header.type = DATAACK;
            temp->header.ack_num = tcblist[index].tcb->expect_seqNum;
            temp->header.src_port = server_port;
            temp->header.dest_port = client_port;
            temp->header.length = 0;

            sip_sendseg(STCP_server, temp);
            printf("\n************************\n");
            printf("type:  DATAACK\n");
            printf("The expect seq is equal to the recv seq ...\n");
            printf("srcport: %d  \n", server_port);
            printf("destport: %d \n", client_port);
            printf("receive the data seq: %d \n", seq);
            printf("data seq length : %d \n", data_len);
            printf("expect the data seq:  %d\n", tcblist[index].tcb->expect_seqN
            printf("************************\n");
        }
    }
    else{
        seg_t* temp = (seg_t *)malloc(sizeof(seg_t));
        temp->header.type = DATAACK;
        temp->header.ack_num = tcblist[index].tcb->expect_seqNum;
        temp->header.src_port = server_port;
        temp->header.dest_port = client_port;
        temp->header.length = 0;

        sip_sendseg(STCP_server, temp);
        printf("\n************************\n");
        printf("type:  DATAACK\n");
        printf("The expect seq No equal to the recv seq ...\n");
        printf("srcport: %d  \n", server_port);
        printf("destport: %d \n", client_port);
        printf("expect the data seq:  %d\n", tcblist[index].tcb->expect_seqNum);
        printf("************************\n");
    }

    pthread_mutex_unlock(tcblist[index].tcb->bufMutex);
}
```

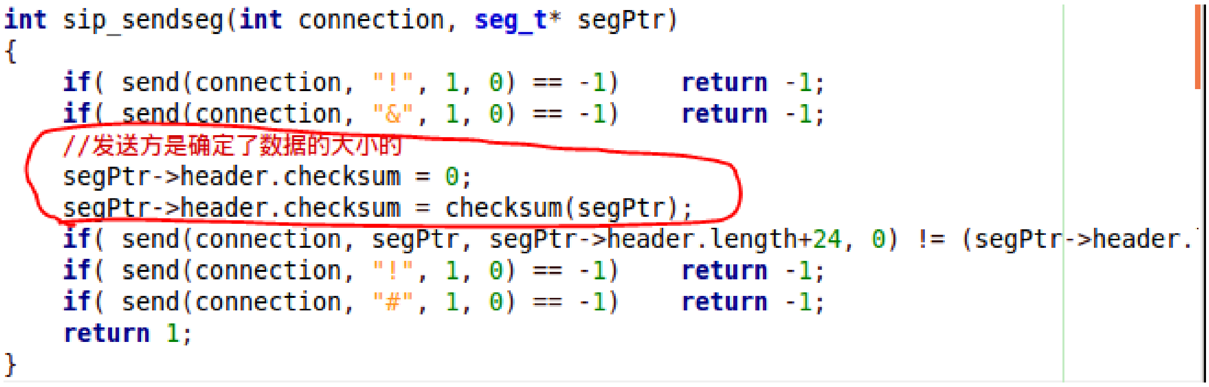


```
case CONNECTED:{
    if(server_type==DATAACK){
        pthread_mutex_lock(tcblist[index].tcb->bufMutex);

        printf("\n**********************\n");
        printf("srcport: %d  \n", server_port);
        printf("destport: %d \n", client_port);
        printf("type: DATAACK\n");
        printf("ack_num: %d \n", seq);
        printf("***********************\n");

        int ack = seq;
        segBuf_t *temp = tcblist[index].tcb->sendBufHead;
        if(temp==NULL) printf("the head is null\n");
        while(temp != NULL){
            printf(" *******   the Head seq num is %d\n", temp->seg.header.seq_n
            if(temp->seg.header.seq_num < ack){
                printf("free the acked data --- the seq is %d...\n", temp->seg.h
                //segBuf_t *q = temp;
                //temp = temp->next;
                tcblist[index].tcb->sendBufHead = temp->next;
                tcblist[index].tcb->unAck_segNum--;
                free(temp);
                temp = tcblist[index].tcb->sendBufHead;
            }
```



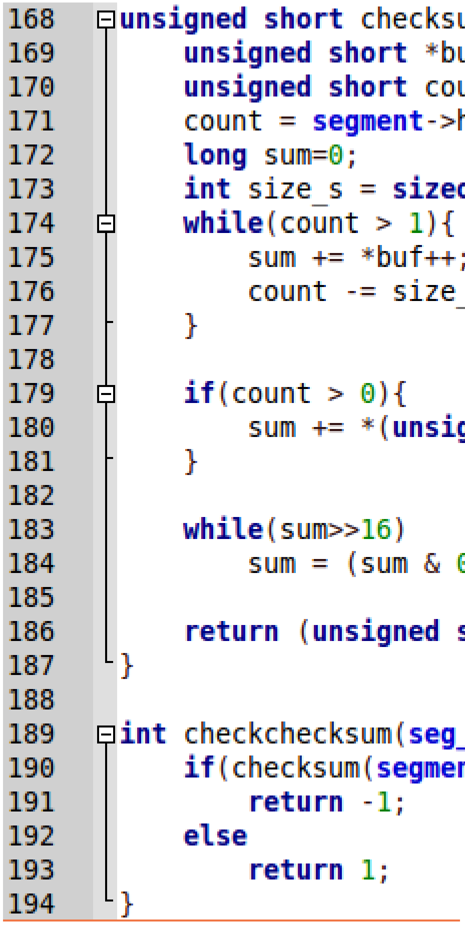
```
            else{
                break;
            }
        }
        // all is acked
        if(tcblist[index].tcb->sendBufHead == NULL){
            tcblist[index].tcb->sendBufTail = NULL;
            tcblist[index].tcb->sendBufunSent = NULL;
            tcblist[index].tcb->unAck_segNum = 0;
        }

        //< GBN so we can send the new seg...
        if(tcblist[index].tcb->sendBufunSent != NULL){
            segBuf_t *tt = tcblist[index].tcb->sendBufunSent;
            while(tcblist[index].tcb->unAck_segNum < GBN_WINDOW && tt
                gettimeofday(&(tt->sentTime), NULL);
                sip_sendseg(STCP_client, &tt->seg);
                tcblist[index].tcb->unAck_segNum++;
                tt = tt->next;
            }
            tcblist[index].tcb->sendBufunSent = tt;
        }

        pthread_mutex_unlock(tcblist[index].tcb->bufMutex);
    }
    break;
}
```

Sip_sendseg()函数添加校验和的计算：



```
int sip_sendseg(int connection, seg_t* segPtr)
{
    if( send(connection, "!", 1, 0) == -1)    return -1;
    if( send(connection, "&", 1, 0) == -1)    return -1;
    //发送方是确定了数据的大小的
    segPtr->header.checksum = 0;
    segPtr->header.checksum = checksum(segPtr);
    if( send(connection, segPtr, segPtr->header.length+24, 0) != (segPtr->header.
    if( send(connection, "!", 1, 0) == -1)    return -1;
    if( send(connection, "#", 1, 0) == -1)    return -1;
    return 1;
}
```

校验和计算函数:

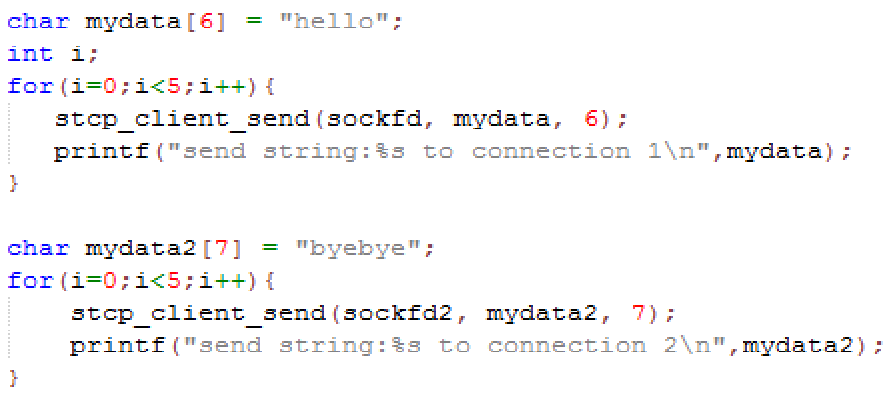


```
unsigned short checksum(seg_t* segment){
    unsigned short *buf = (unsigned short *)segment;
    unsigned short count;
    count = segment->header.length + 24;
    long sum=0;
    int size_s = sizeof(unsigned short int);
    while(count > 1){
        sum += *buf++;
        count -= size_s;
    }

    if(count > 0){
        sum += *(unsigned char*)buf;
    }

    while(sum>>16)
        sum = (sum & 0xFFFF) + (sum >> 16);

    return (unsigned short) (~sum);
}

int checkchecksum(seg_t* segment){
    if(checksum(segment))
        return -1;
    else
        return 1;
}
```

四、实验结果及报文分析

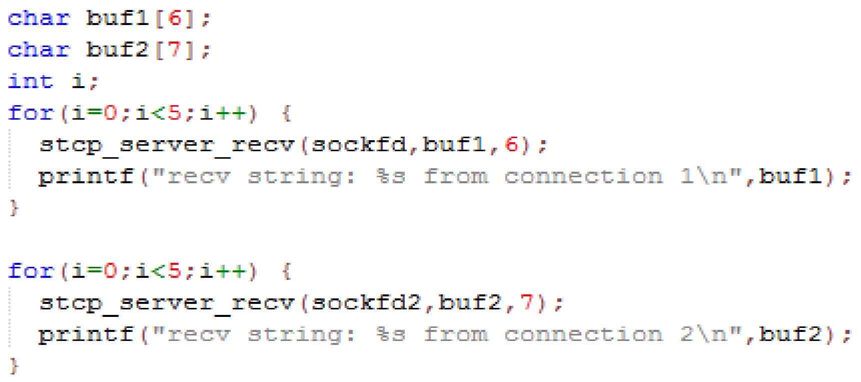
简单测试结果分析:

客户端发送数据部分的程序，



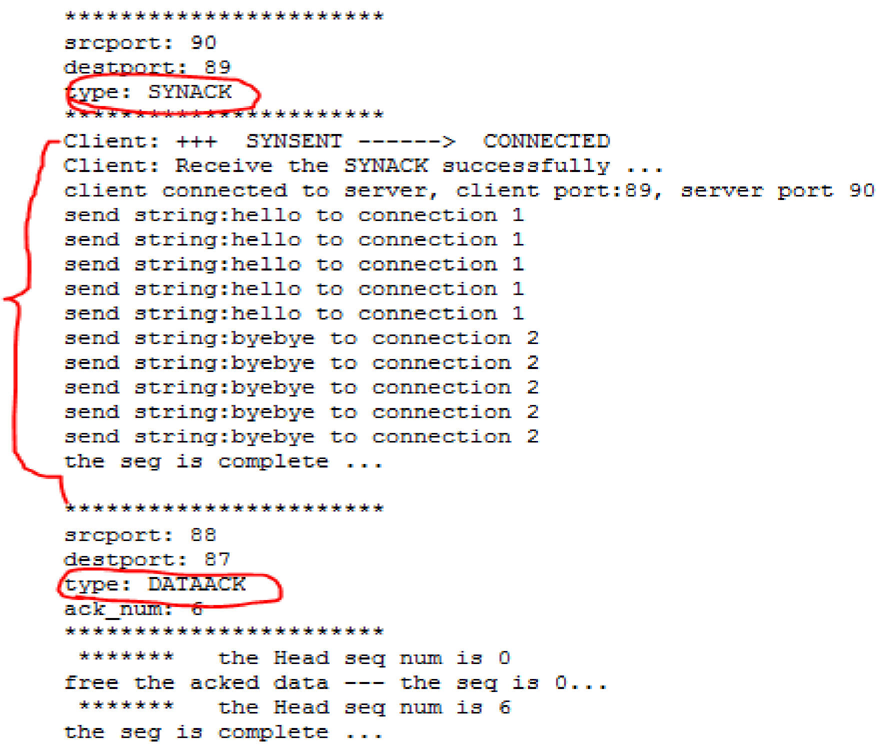
```
char mydata[6] = "hello";
int i;
for(i=0;i<5;i++){
   stcp_client_send(sockfd, mydata, 6);
   printf("send string:%s to connection 1\n",mydata);
}

char mydata2[7] = "byebye";
for(i=0;i<5;i++){
    stcp_client_send(sockfd2, mydata2, 7);
    printf("send string:%s to connection 2\n",mydata2);
}
```

服务器部分接收数据的部分:



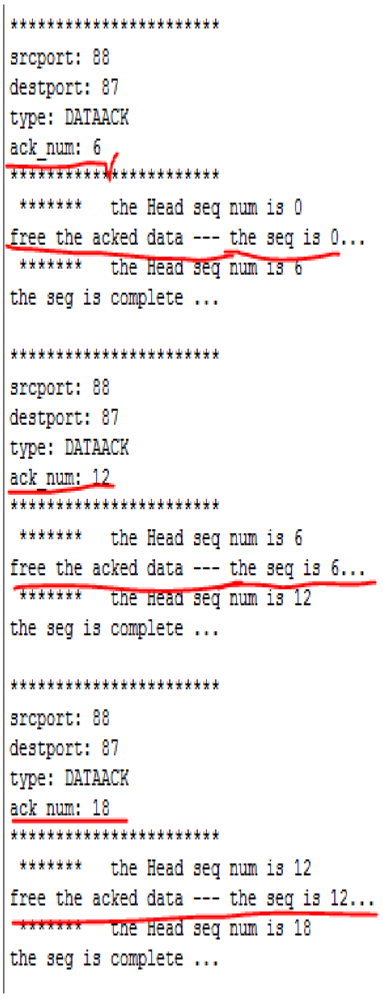
```
char buf1[6];
char buf2[7];
int i;
for(i=0;i<5;i++) {
  stcp_server_recv(sockfd,buf1,6);
  printf("recv string: %s from connection 1\n",buf1);
}

for(i=0;i<5;i++) {
  stcp_server_recv(sockfd2,buf2,7);
  printf("recv string: %s from connection 2\n",buf2);
}
```

测试的建立连接，以及发送报文的测试结果截图:

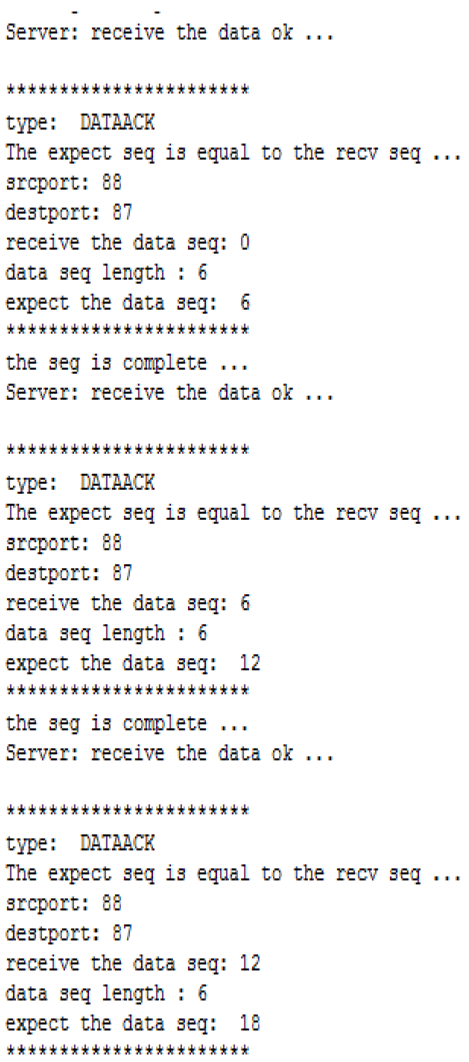


```
***********************
srcport: 90
destport: 89
type: SYNACK
***********************
Client: +++  SYNSENT ------>  CONNECTED
Client: Receive the SYNACK successfully ...
client connected to server, client port:89, server port 90
send string:hello to connection 1
send string:hello to connection 1
send string:hello to connection 1
send string:hello to connection 1
send string:hello to connection 1
send string:byebye to connection 2
send string:byebye to connection 2
send string:byebye to connection 2
send string:byebye to connection 2
send string:byebye to connection 2
the seg is complete ...

***********************
srcport: 88
destport: 87
type: DATAACK
ack_num: 6
***********************
 *******   the Head seq num is 0
free the acked data --- the seq is 0...
 *******   the Head seq num is 6
the seg is complete ...
```

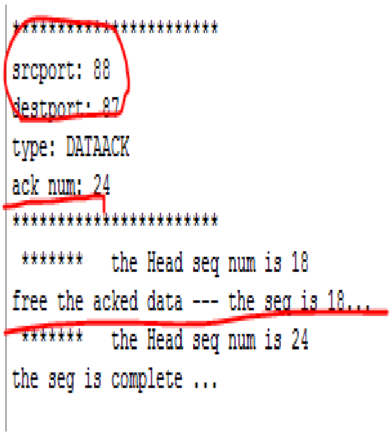
客户端在收到 SYNACK 后，确认连接建立成功，进入了 CONNECTED 状态，然后发送数据且发送数据部分的时候，没有发生数据丢失，之后受到了服务器发送的 DATAACK 确认帧。



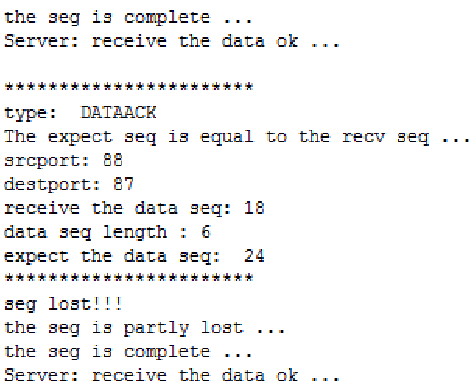
```
***********************
srcport: 88
destport: 87
type: DATAACK
ack_num: 6
***********************
 *******   the Head seq num is 0
free the acked data --- the seq is 0...
 *******   the Head seq num is 6
the seg is complete ...

***********************
srcport: 88
destport: 87
type: DATAACK
ack_num: 12
***********************
 *******   the Head seq num is 6
free the acked data --- the seq is 6...
 *******   the Head seq num is 12
the seg is complete ...

***********************
srcport: 88
destport: 87
type: DATAACK
ack_num: 18
***********************
 *******   the Head seq num is 12
free the acked data --- the seq is 12...
 *******   the Head seq num is 18
the seg is complete ...
```



```
Server: receive the data ok ...

***********************
type:  DATAACK
The expect seq is equal to the recv seq ...
srcport: 88
destport: 87
receive the data seq: 0
data seq length : 6
expect the data seq:  6
***********************
the seg is complete ...
Server: receive the data ok ...

***********************
type:  DATAACK
The expect seq is equal to the recv seq ...
srcport: 88
destport: 87
receive the data seq: 6
data seq length : 6
expect the data seq:  12
***********************
the seg is complete ...
Server: receive the data ok ...

***********************
type:  DATAACK
The expect seq is equal to the recv seq ...
srcport: 88
destport: 87
receive the data seq: 12
data seq length : 6
expect the data seq:  18
***********************
```



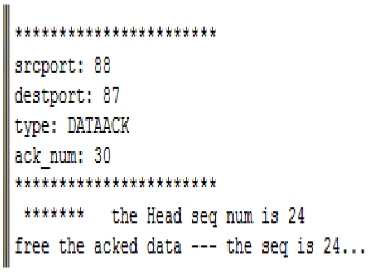
```
***********************
srcport: 88
destport: 87
type: DATAACK
ack_num: 24
***********************
 *******   the Head seq num is 18
free the acked data --- the seq is 18...
 *******   the Head seq num is 24
the seg is complete ...
```



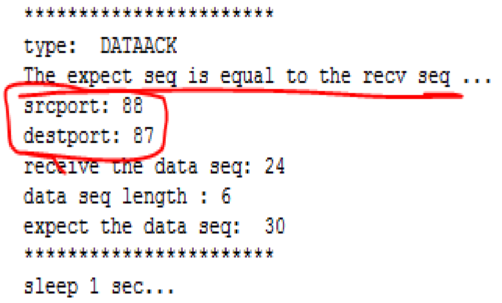
```
the seg is complete ...
Server: receive the data ok ...

***********************
type:  DATAACK
The expect seq is equal to the recv seq ...
srcport: 88
destport: 87
receive the data seq: 18
data seq length : 6
expect the data seq:  24
***********************
seg lost!!!
the seg is partly lost ...
the seg is complete ...
Server: receive the data ok ...
```

上图服务器端现实最后一个确认帧丢失了！！ 需要重新发送，seglost!!!

这是在中间端口和 90 和 89 之间的通信结束的时候，才重发的确认帧



```
**********************
srcport: 88
destport: 87
type: DATAACK
ack_num: 30
**********************
 *******   the Head seq num is 24
free the acked data --- the seq is 24...
```

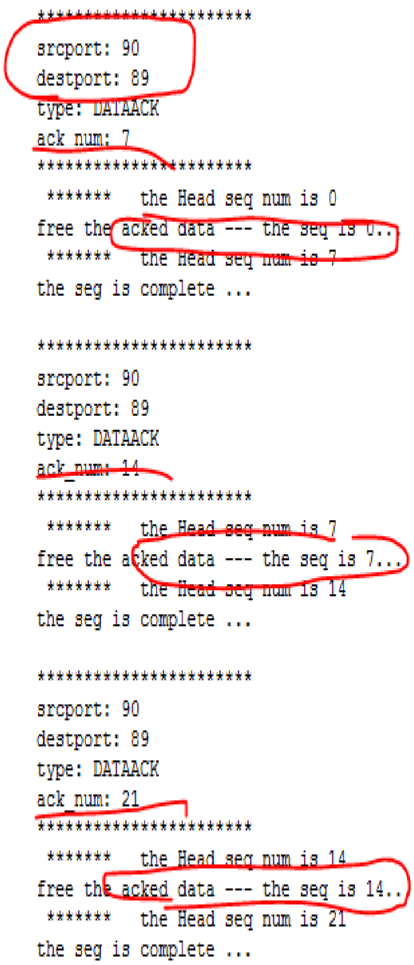


```
***********************
type:  DATAACK
The expect seq is equal to the recv seq ...
srcport: 88
destport: 87
receive the data seq: 24
data seq length : 6
expect the data seq:  30
***********************
sleep 1 sec...
```

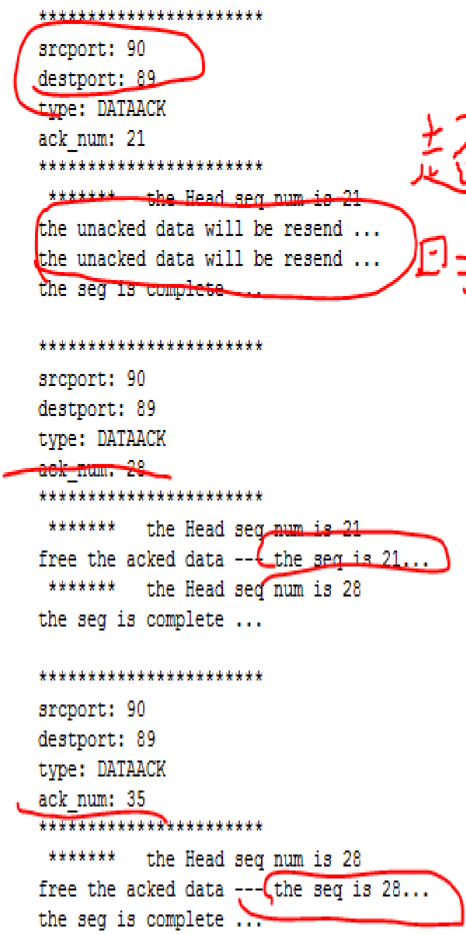
上图显示的是，端口 87 和 88 之间的数据通信

左侧是客户端，右侧是服务器端

客户端收到了 DATAACK 后，根据 acknum 需要释放已经发送但是未被确认的窗口，故维护的链表表头的数据序列号增大。

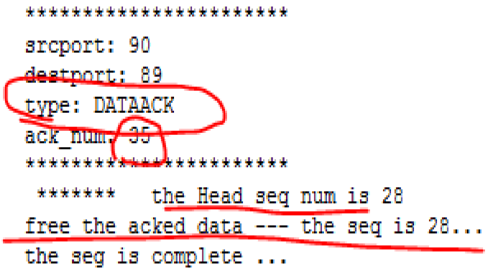


```
**********************
srcport: 90
destport: 89
type: DATAACK
ack_num: 7
**********************
 *******   the Head seq num is 0
free the acked data --- the seq is 0...
 *******   the Head seq num is 7
the seq is complete ...

**********************
srcport: 90
destport: 89
type: DATAACK
ack_num: 14
**********************
 *******   the Head seq num is 7
free the acked data --- the seq is 7...
 *******   the Head seq num is 14
the seq is complete ...

**********************
srcport: 90
destport: 89
type: DATAACK
ack_num: 21
**********************
 *******   the Head seq num is 14
free the acked data --- the seq is 14...
 *******   the Head seq num is 21
the seq is complete ...
```



```
**********************
srcport: 90
destport: 89
type: DATAACK
ack_num: 21
**********************
 *******   the Head seq num is 21
the unacked data will be resend ...
the unacked data will be resend ...
the seq is complete ...

**********************
srcport: 90
destport: 89
type: DATAACK
ack_num: 28
**********************
 *******   the Head seq num is 21
free the acked data --- the seq is 21...
 *******   the Head seq num is 28
the seq is complete ...

**********************
srcport: 90
destport: 89
type: DATAACK
ack_num: 35
**********************
 *******   the Head seq num is 28
free the acked data --- the seq is 28...
the seq is complete ...
```

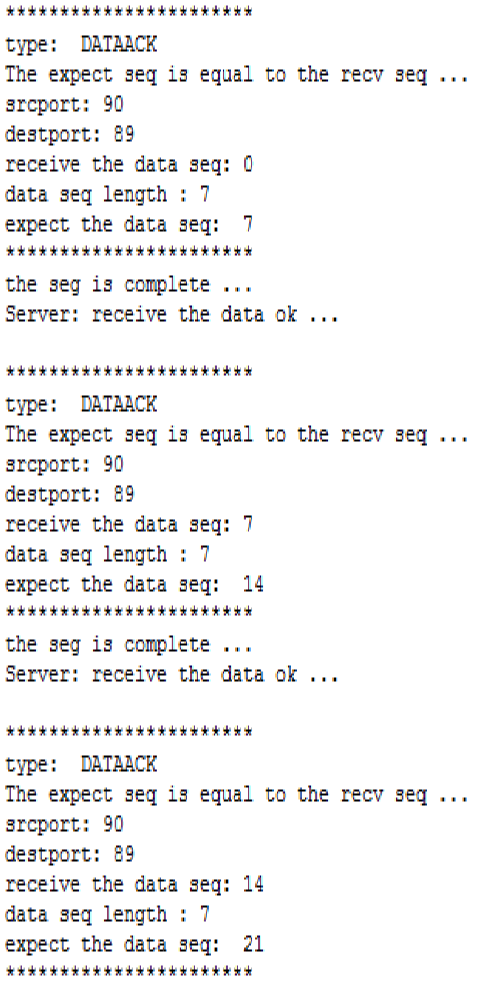
超时



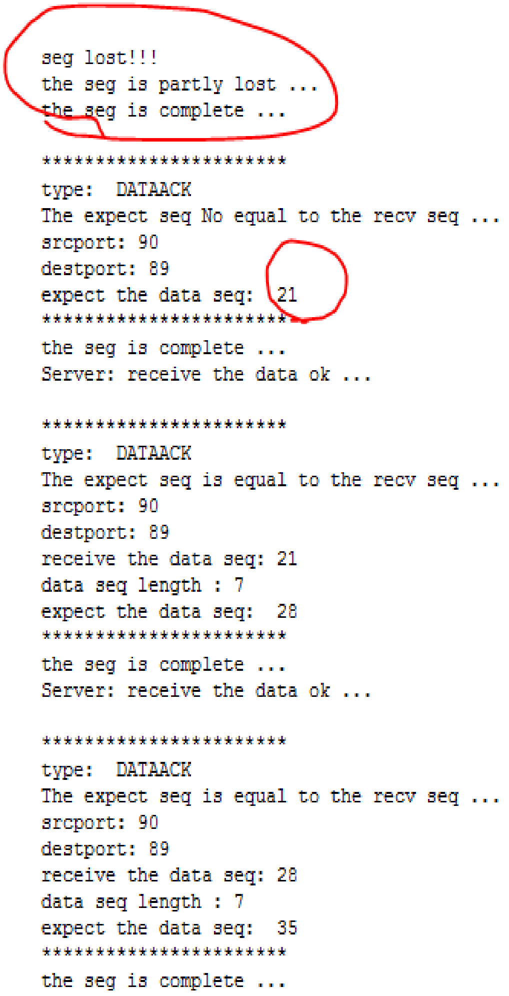
```
**********************
srcport: 90
destport: 89
type: DATAACK
ack_num: 35
**********************
 *******   the Head seq num is 28
free the acked data --- the seq is 28...
the seq is complete ...
```

上图显示的客户端是端口 89 和 90 之间的数据传输，且传输过程中发生了超时，未被确认的帧，全部都被重发。

下图是服务器端的信息：

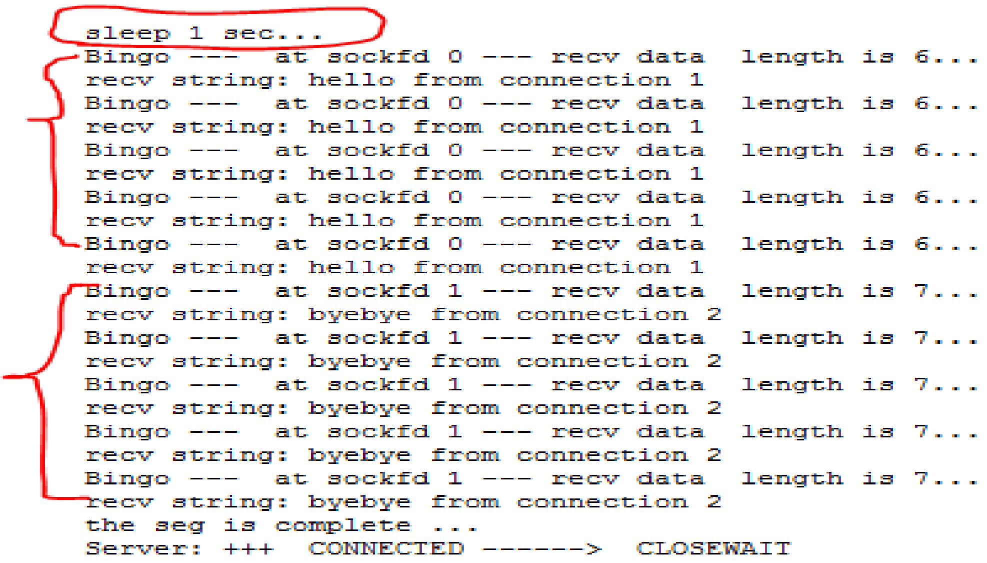


```
**********************
type:  DATAACK
The expect seq is equal to the recv seq ...
srcport: 90
destport: 89
receive the data seq: 0
data seq length : 7
expect the data seq:  7
**********************
the seq is complete ...
Server: receive the data ok ...

**********************
type:  DATAACK
The expect seq is equal to the recv seq ...
srcport: 90
destport: 89
receive the data seq: 7
data seq length : 7
expect the data seq:  14
**********************
the seq is complete ...
Server: receive the data ok ...

**********************
type:  DATAACK
The expect seq is equal to the recv seq ...
srcport: 90
destport: 89
receive the data seq: 14
data seq length : 7
expect the data seq:  21
**********************
```



```
seq lost!!!
the seq is partly lost ...
the seq is complete ...

**********************
type:  DATAACK
The expect seq No equal to the recv seq ...
srcport: 90
destport: 89
expect the data seq:  21
**********************
the seq is complete ...
Server: receive the data ok ...

**********************
type:  DATAACK
The expect seq is equal to the recv seq ...
srcport: 90
destport: 89
receive the data seq: 21
data seq length : 7
expect the data seq:  28
**********************
the seq is complete ...
Server: receive the data ok ...

**********************
type:  DATAACK
The expect seq is equal to the recv seq ...
srcport: 90
destport: 89
receive the data seq: 28
data seq length : 7
expect the data seq:  35
**********************
the seq is complete ...
```

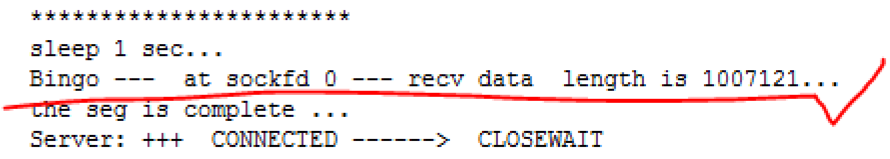
最后服务器端显示接收信息 OK!!!



```
sleep 1 sec...
Bingo ---  at sockfd 0 --- recv data  length is 6...
recv string: hello from connection 1
Bingo ---  at sockfd 0 --- recv data  length is 6...
recv string: hello from connection 1
Bingo ---  at sockfd 0 --- recv data  length is 6...
recv string: hello from connection 1
Bingo ---  at sockfd 0 --- recv data  length is 6...
recv string: hello from connection 1
Bingo ---  at sockfd 0 --- recv data  length is 6...
recv string: hello from connection 1
Bingo ---  at sockfd 1 --- recv data  length is 7...
recv string: byebye from connection 2
Bingo ---  at sockfd 1 --- recv data  length is 7...
recv string: byebye from connection 2
Bingo ---  at sockfd 1 --- recv data  length is 7...
recv string: byebye from connection 2
Bingo ---  at sockfd 1 --- recv data  length is 7...
recv string: byebye from connection 2
Bingo ---  at sockfd 1 --- recv data  length is 7...
recv string: byebye from connection 2
the seg is complete ...
Server: +++  CONNECTED ------>  CLOSEWAIT
```

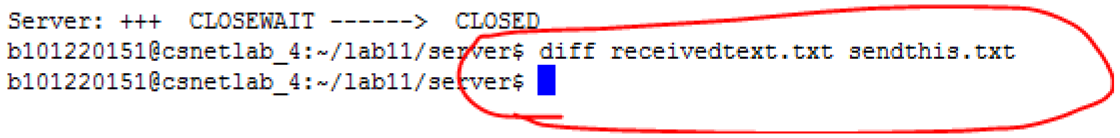
压力测试结果分析：

接收 1M 多的文件成功：



```
*************************
sleep 1 sec...
Bingo ---  at sockfd 0 --- recv data  length is 1007121...
the seg is complete ...
Server: +++  CONNECTED ------>  CLOSEWAIT
```

做 diff 判断是可以的。



```
Server: +++  CLOSEWAIT ------>  CLOSED
b101220151@csnetlab_4:~/lab11/server$ diff receivedtext.txt sendthis.txt
b101220151@csnetlab_4:~/lab11/server$ 
```

六、 实验的启示/意见和建议

这次实验估计花了 25 个小时以上吧，实验中遇到的 BUG 主要是跟一些发送段信息时，初始化不正确。

在进行压力测试的时候主要遇到的问题是接收数据的缓冲区的大小不够，可能是因为编码的原因，我的 1M 的文件需要开设 1007121 个 char 大小，因此，后来调整了接收缓冲区后，接收数据没有

问题，还有就是由于 receivedtext.txt 中重复测试，存在大量重复的数据，所以在执行 diff 操作的时候会出错，后来对其清空后再进行测试即可。