《计算机网络协议开发》实验报告

第五次实验：重叠网络协议栈开发

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：2022.5.16

一、实验目的

熟悉传输控制层信号协议的设计、实现。熟悉传输控制层协议是如何建立连接，以及如何解决信号数据包的丢失和损坏。

二、成品功能

完成了 pdf 中 5.2 所要求的所有基本功能，即 GBN 和检验和实现，可以完成验收展示。

三、实验内容

1)STCP CLIENT

stcp_client_send 实现，构造包的方式和实验 5.1 中构造 SYN 和 FIN 的方式一样，只不过这里要先把包加入发送缓冲区，以实现 GBN 功能。加入缓冲区之后，调用 seg_send 函数，如果未响应的包的个数为 0，那么就启动 timer 线程，判断期待收到的包是否超时发送。

```
void sendBuf_send(client_tcb_t* clienttcb) {
  pthread_mutex_lock(clienttcb->bufMutex);

  while(clienttcb->unAck_segNum<GBN_WINDOW && clienttcb->sendBufunSent!=0) {
    sip_sendseg(Sockfd, (seg_t*)clienttcb->sendBufunSent);
    struct timeval currentTime;
    gettimeofday(&currentTime,NULL);
    clienttcb->sendBufunSent->sentTime = currentTime.tv_sec*1000+ currentTime.tv_usec;

    if(clienttcb->unAck_segNum ==0) {
      pthread_t timer;
      pthread_create(&timer,NULL,sendBuf_timer, (void*)clienttcb);
    }
    clienttcb->unAck_segNum++;
    if(clienttcb->sendBufunSent != clienttcb->sendBufTail)
      clienttcb->sendBufunSent= clienttcb->sendBufunSent->next;
    else
      clienttcb->sendBufunSent = 0;
  }
  pthread_mutex_unlock(clienttcb->bufMutex);
}
```

如果 timer 线程判断超时，那么就调用 sendBuf_timeout 函数，进行对超时的包重传。当然重传的包的要求是只重传从 sendBufHead 开始数目为 unAck_segNum 的包。

2)STCP SERVER

接收包的时候，对于 5.1 的接收 FSM，需要判断是否收到 DATA，如果收到，则需要判断接收缓冲区是否还能接收以及收到的序列号是否是 expect_num，如果不是的话就丢弃。之后构造 DATAACK 包发回，包中一定要包含 expect_num，这样才能实现 GBN 机制。

```
else if(packet.header.type == DATA){
    printf("DATA received\n");
    if(packet.header.seq_num == ServerTCB[tcbnum].expect_seqNum){
        if(ServerTCB[tcbnum].usedBufLen + packet.header.length < RECEIVE_BUF_SIZE){
            printf("able to recv in recvbuf\n");
            //ServerTCB[tcbnum].usedBufLen += packet.header.length;
            ServerTCB[tcbnum].expect_seqNum += packet.header.length;
            recvBuf_addbuf(&ServerTCB[tcbnum], &packet);
        }
        else printf("drop data\n");
    }
```

对于收到的 DATA 包，接收进缓冲区之后，由 stcp_server_recv 从缓冲区中读取。这里读取时候一定要注意 tcb 中 bufMutex 的使用，防止读写冲突。

```
pthread_mutex_lock(&mutex_read);
if(length <= ServerTCB[sockfd].usedBufLen){
    //printf("%d\n",ServerTCB[sockfd].usedBufLen);
    pthread_mutex_lock(ServerTCB[sockfd].bufMutex);
    memcpy(buf,ServerTCB[sockfd].recvBuf,length);
    memcpy(ServerTCB[sockfd].recvBuf,ServerTCB[sockfd].recvBuf + length,ServerTCB[sockfd].usedBufLen - length);
    ServerTCB[sockfd].usedBufLen = ServerTCB[sockfd].usedBufLen - length;
    pthread_mutex_unlock(ServerTCB[sockfd].bufMutex);
    pthread_mutex_unlock(&mutex_read);
    return 1;
}
```

Mutex_read 是用来判断是否读取完的互斥锁，这样可以防止没有读取完就关闭连接的情况发生。

3) 校验和

对于校验和的实现，我上网查找了校验和的计算方式之后，决定对整个包的指针强行转换成 unsigned short*类型，然后对于整个包进行每 16 位进行校验和的计算。最后取反即可。

```
unsigned short checksum(seg_t* segment)
{
    unsigned int sum = 0;
    unsigned short* p = (unsigned short*)segment;
    int length = 12 + segment->header.length / 2;
    if(segment->header.length%2 != 0)
        length = length + 1;
    for(int i=0;i<length;i++)
        sum = sum + p[i];
    while(sum >> 16)
        sum = (unsigned short)(sum >> 16) + (sum & 0xffff);
    //printf("checksum is %x\n",(unsigned short)~sum);
    return (unsigned short)~sum;
}
```

这里一定要注意，计算的时候一定要对高于 16 位的数位清零，否则算出来的校验和有问题。

四、问题及解决方法

5.2 的问题主要出现在压力测试。

1. 校验和的计算一开始没有对高位清零，所以计算的校验和以及检查时得到的结果不匹配。

2. 对于 seglost 的调用，seglost 要传入指针，但是传进去时，校验和函数对这个指针指向的内容有修改，所以返回也会有问题。