《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 5.1 所要求的所有基本功能，即 STCP 协议信号的实现，可以完成验收展示。

三、实验内容

1)STCP CLIENT

对于所有的 tcb 块，都给 struct 添加了一个成员 is_used，来标志是否被使用过。下面创建的 tcb 池直接使用了结构体数组的形式，每次初始化就将 is_used 置为 used，不使用就置为 unused。同时这里采用的收发方式跟实验四类似，接收线程和发送线程分开，故这里有互斥锁处理对于是否收到 SYNACK 以及 FINACK 的访问。

```
//客户端传输控制块. 一个STCP连接的客户端使用这个数据结构记录连接信息.
typedef struct client_tcb {
    unsigned int server_nodeID;         //服务器节点ID, 类似IP地址, 本实验未使用
    unsigned int server_portNum;        //服务器端口号
    unsigned int client_nodeID;     //客户端节点ID, 类似IP地址, 本实验未使用
    unsigned int client_portNum;    //客户端端口号
    unsigned int state;         //客户端状态
    unsigned int is_used;
    unsigned int next_seqNum;       //新段准备使用的下一个序号
    pthread_mutex_t* bufMutex;      //发送缓冲区互斥量
    segBuf_t* sendBufHead;          //发送缓冲区头
    segBuf_t* sendBufunSent;        //发送缓冲区中的第一个未发送段
    segBuf_t* sendBufTail;          //发送缓冲区尾
    unsigned int unAck_segNum;      //已发送但未收到确认段的数量
} client_tcb_t;

client_tcb_t ClientTCB[MAX_TRANSPORT_CONNECTIONS];
int Sockfd;
pthread_mutex_t mutex_syn;
extern bool isSYNACK;

pthread_mutex_t mutex_fin;
extern bool isFINACK;
//
```

对于与服务器的连接与之前类似，构造发送的包的代码也和之前类似，这里不加以赘述。

接收的线程处理，使用了手册中推荐的 FSM 风格代码，用一个大循环一直接收发送过来的包，按照现在 client 所在的不同的状态进行处理即可。

```
while(ret = sip_recvseg(Sockfd,&packet) > 0){
  if(packet.header.dest_port == 0){
    printf("actually lose\n");
  }
  else{
    int tcbnum = gettcb(packet.header.dest_port);
    switch (ClientTCB[tcbnum].state)
    {
    case CLOSED:
      break;
    case SYNSENT:
      if(packet.header.type == SYNACK && packet.header.src_port == ClientTCB[tcbnum].server_portNum){
        printf("SYNACK received\n");
        pthread_mutex_lock(&mutex_syn);
        isSYNACK = true;
        pthread_mutex_unlock(&mutex_syn);
      }
      else{
        printf("client in SYNSENT, no SYNACK receive\n");
      }
      break;
    case CONNECTED:
      break;
    case FINWAIT:
      if(packet.header.type == FINACK && packet.header.src_port == ClientTCB[tcbnum].server_portNum){
        printf("FINACK received\n");
        pthread_mutex_lock(&mutex_fin);
        isFINACK = true;
        pthread_mutex_unlock(&mutex_fin);
      }
      else{
```

2) STCP SERVER

数据结构和 CLIENT 类似，tcb 块的处理和 CLIENT 几乎相同。当然和 CLIENT 不同的地方在于，并不需要将接收线程和发送线程分开。在接收到对应的包之后，直接立即发送对应的 ack 即可。

```
switch (ServerTCB[tcbnum].state)
{
    case CLOSED:
        break;
    case LISTENING:
        if(packet.header.type == SYN){
            printf("SYN received\n");
            ServerTCB[tcbnum].state = CONNECTED;
            ServerTCB[tcbnum].client_portNum = packet.header.src_port;
            seg_t SYNACKpacket;
            memset(&SYNACKpacket, 0 , sizeof(seg_t));
            SYNACKpacket.header.src_port = ServerTCB[tcbnum].server_portNum;
            SYNACKpacket.header.dest_port = ServerTCB[tcbnum].client_portNum;
            SYNACKpacket.header.type = SYNACK;
            SYNACKpacket.header.length = 24;
            printf("sending SYNACK packet\n");
            sip_sendseg(Sockfd,&SYNACKpacket);
        }
        else{
            printf("server in LISTENING, no SYN receive\n");
        }
        break;
```

上图就是对 SYNACK 包的构造以及发送。

3) SEG

发送函数所需要的就是在开始和末尾加上!&以及!#，所以这里发送的时候，我采用的方式是在发送缓冲区中加上开头的!&，然后写上要发送的内容，最后再写上!#。

```
seg_t temp = *segPtr;
//printf("get %d\n",temp.header.dest_port);
memcpy(tempbuf,&temp,sizeof(seg_t));
//printf("%d\n",tempbuf[4]);
char head[3] = "!&";
char tail[3] = "!#";
memcpy(sendline,head,sizeof(head));
memcpy(sendline+2,tempbuf,sizeof(seg_t));
memcpy(sendline+2+sizeof(seg_t),tail,sizeof(tail));
//printf("%s\n",sendline);
```