**TCP实验上机报告**

1. **实验目的**

本实验的主要目的是学习和了解 TCP 协议的原理和设计实现的机制;实现客户端角色的、 “停－等”模式的 TCP 协议，能够正确的建立和拆除连接，接收和发送 TCP 报文，并向应用层提供客户端需要的 Socket 函数为应用层提供网络编程接口，即socket接口。

1. **实验要求**
2. 理解TCP的主要原理，针对客户端角色的、“停-等”模式的TCP，设计接收和发送流程。
3. 编程实现TCP段的接收流程，重点是段接收的有限状态机。
4. 编程实现TCP段的发送流程，完成TCP段的封装处理。
5. 编程实现客户端Socket接口函数。
6. **编程实现**
7. 停-等协议的实现

停-等协议主要体现在stud\_tcp\_send()函数和stud\_tcp\_recv()和函数中。主要思路是：

1. 调用stud\_tcp\_send()发送一个数据包时，直到收到ack包并把ack交给stud\_tcp\_input()处理才退出
2. 调用stud\_tcp\_recv()接收一个数据包成功后，紧接着发送一个相应的ack包才退出

上述两个设计保证了发送和接收窗口大小为1。

1. 接收流程的有限状态机

首先检查校验和与序号是否正确，只有当这两者都无误之后才进入状态机部分。

状态机基于switch结构实现，根据当前TCB的状态分情形讨论状态的变化。

1. 发送时的封装和有限状态机
2. 查看当前的TCB是否为空，为空则创建当前的TCB
3. 封装数据
4. 改变状态
5. 变换字节序（将字节序从本地序变成网络序）
6. 各接口函数的实现
7. stud\_tcp\_socket()
   * 1. 检查domain、type、protocol是否合法；
     2. 新建TCB，初始化；
     3. 更改全局变量sockfd、srcPort、seq；
8. stud\_tcp\_connect()
   * 1. 修改TCB相关变量，将状态改为SYN\_SENT;
     2. 发送SYN，等待应答；
     3. 若收到SYN+ACK,回复ACK；
     4. 状态变为ESTABLISHED;
9. stud\_tcp\_send()
   * 1. 检查sockfd，不合法返回；
     2. 检查状态是否为ESTABLISHED,不是返回；
     3. 调用tcp\_output();
     4. 等待ACK, 调用tcp\_input()处理ACK；
10. stud\_tcp\_recv()
    * 1. 检查sockfd，不合法返回；
      2. 检查状态是否为ESTABLISHED,不是返回；
      3. 等待数据包, 调用tcp\_output()发送ACK；
11. stud\_tcp\_close()
    * 1. 删除当前TCB；
      2. 检查状态是否为ESTABLISHED;
      3. 发送FIN\_ACK
      4. 状态转移到FIN\_WAIT1;
      5. 等待ACK；
      6. 状态转移到FIN\_ACK2；
      7. 等待FIN\_ACK包；
      8. 状态转移到TIME\_WAIT;
12. 校验和的计算

要对伪首部进行校验。伪首部由以下部分组成：

1. 32位IP源地址
2. 32位IP目的地址
3. 8位填充位（全0）
4. 8位协议号
5. 16位TCP头部长度
6. 源代码

#include "sysInclude.h"

extern void tcp\_DiscardPkt(char \*pBuffer, int type);

extern void tcp\_sendIpPkt(unsigned char \*pData, UINT16 len, unsigned int srcAddr, unsigned int dstAddr, UINT8 ttl);

extern int waitIpPacket(char \*pBuffer, int timeout);

extern unsigned int getIpv4Address();

extern unsigned int getServerIpv4Address();

typedef struct tcphead{

UINT16 srcPort;

UINT16 destPort;

UINT32 seqNo;

UINT32 ackNo;

UINT8 headLen;

UINT8 flag;

UINT16 windowsize;

UINT16 checksum;

UINT16 urgentPointer;

char data[100];

};

typedef struct TCB

{

unsigned int srcAddr;

unsigned int dstAddr;

unsigned short srcPort;

unsigned short dstPort;

unsigned int seq;

unsigned int ack;

int sockfd;

BYTE state;

unsigned char\* data;

};

typedef struct tcb\_node{

TCB \*current;

struct tcb\_node \*next;

};

struct tcb\_node \*tcb\_table;

struct TCB \*current\_tcb;

enum status{CLOSED,SYN\_SENT,ESTABLISHED,FIN\_WAIT1,FIN\_WAIT2,TIME\_WAIT};

int gSrcPort = 2005;

int gDstPort = 2006;

int gSeqNum = 1;

int gAckNum = 0;

int socknum = 5;

unsigned int getchecksum(bool fromLocal, tcphead\* thead,unsigned int srcAddr, unsigned int dstAddr,unsigned short len,char\* data)

{

unsigned int checksum = 0;

checksum += (srcAddr>>16)+srcAddr&0xffff;

checksum += (dstAddr>>16)+dstAddr&0xffff;

checksum += IPPROTO\_TCP;

checksum += 0x14;

if(fromLocal)

{

checksum += thead->srcPort+thead->destPort;

checksum += thead->windowsize;

checksum += thead->urgentPointer;

}else{

checksum += ntohs(thead->srcPort)+ntohs(thead->destPort);

checksum += ntohs(thead->windowsize);

checksum += ntohs(thead->urgentPointer);

}

checksum += ((thead->seqNo)>>16)+(thead->seqNo)&0xffff;

checksum += ((thead->ackNo)>>16)+(thead->ackNo)&0xffff;

checksum += ((thead->headLen)<<8)+thead->flag;

if(thead->flag == PACKET\_TYPE\_DATA){

checksum += len;

int length=len;

char\* p=data;

while(length>0){

checksum += (\*p)<<8;

p++;

checksum += (\*p);

p++;

length=length-2;

}

}

checksum = (checksum>>16)+checksum&0xffff;

checksum = (checksum>>16)+checksum&0xffff;

checksum = (~checksum)&0xffff;

return checksum;

}

int stud\_tcp\_input(char \*pBuffer, unsigned short len, unsigned int srcAddr, unsigned int dstAddr)

{

tcphead\* header = (tcphead\*)pBuffer;

header->seqNo = ntohl(header->seqNo);

header->ackNo = ntohl(header->ackNo);

if(getchecksum(0,header,ntohl(srcAddr),ntohl(dstAddr),len,NULL) != ntohs(header->checksum)){

return -1;

}

int seqAdd=1; //SYN-SENT FINWAIT1

if(current\_tcb->state == FIN\_WAIT2){

seqAdd=0;

}

else if(len > 20)

{

seqAdd=len-20;

}

if(header->ackNo != (current\_tcb->seq+seqAdd)){

tcp\_DiscardPkt(pBuffer, STUD\_TCP\_TEST\_SEQNO\_ERROR);

return -1;

}

//状态转移

switch(current\_tcb->state)

{

case SYN\_SENT:

if(header->flag == PACKET\_TYPE\_SYN\_ACK )

{

current\_tcb->state = ESTABLISHED;

current\_tcb->ack= header->seqNo+1;

current\_tcb->seq = header->ackNo;

stud\_tcp\_output(NULL, 0, PACKET\_TYPE\_ACK, current\_tcb->srcPort, current\_tcb->dstPort, ntohl(srcAddr), ntohl(dstAddr));

break;

}

else

return -1;

case ESTABLISHED:

if(header->flag == PACKET\_TYPE\_ACK)

{

if(len > 20)

{

current\_tcb->ack = header->seqNo + len-20;

current\_tcb->seq = header->ackNo;

break;

}

else if(len==20)

{

current\_tcb->ack= header->seqNo+1;

current\_tcb->seq = header->ackNo;

break;

}

else return -1;

}

else

return -1;

case FIN\_WAIT1:

if(header->flag == PACKET\_TYPE\_ACK)

{

current\_tcb->ack= header->seqNo+1;

current\_tcb->seq = header->ackNo;

current\_tcb->state = FIN\_WAIT2;

break;

}

else

return -1;

case FIN\_WAIT2:

if(header->flag == PACKET\_TYPE\_FIN\_ACK)

{

current\_tcb->state = TIME\_WAIT;

stud\_tcp\_output(NULL, 0, PACKET\_TYPE\_ACK, current\_tcb->srcPort, current\_tcb->dstPort, ntohl(srcAddr), ntohl(dstAddr));

break;

}

else

return -1;

default:

return -1;

}

return 0;

}

void stud\_tcp\_output(char \*pData, unsigned short len, unsigned char flag, unsigned short srcPort, unsigned short dstPort, unsigned int srcAddr, unsigned int dstAddr)

{

if(current\_tcb==NULL){

current\_tcb = new TCB;

current\_tcb->seq = gSeqNum;

current\_tcb->ack = gAckNum;

current\_tcb->srcPort = srcPort;

current\_tcb->dstPort = dstPort;

current\_tcb->srcAddr = srcAddr;

current\_tcb->dstAddr = dstAddr;

current\_tcb->state = CLOSED;

}

tcphead\* thead = new tcphead;

for(int i=0; i<len; i++){

thead->data[i] = pData[i];

}

thead->srcPort = srcPort;

thead->destPort = dstPort;

thead->seqNo = current\_tcb->seq;

thead->ackNo = current\_tcb->ack;

thead->headLen = 0x50;

thead->flag = flag;

thead->windowsize = 1;

thead->urgentPointer = 0;

switch(current\_tcb->state)

{

case CLOSED:

if(flag == PACKET\_TYPE\_SYN)

{

current\_tcb->state = SYN\_SENT;

}

else

return;

break;

case ESTABLISHED:

if(flag == PACKET\_TYPE\_FIN\_ACK)

{

current\_tcb->state = FIN\_WAIT1;

break;

}

else if(flag == PACKET\_TYPE\_DATA || flag == PACKET\_TYPE\_ACK)

{

break;

}

else

return;

break;

defalut:

return;

}

thead->checksum = ntohs(getchecksum(1,thead,srcAddr,dstAddr,len,pData));

thead->srcPort = ntohs(thead->srcPort);

thead->destPort = ntohs(thead->destPort);

thead->seqNo = ntohl(thead->seqNo);

thead->ackNo = ntohl(thead->ackNo);

thead->windowsize = ntohs(thead->windowsize);

thead->urgentPointer = ntohs(thead->urgentPointer);

tcp\_sendIpPkt((unsigned char\*)thead, 20+len, current\_tcb->srcAddr, current\_tcb->dstAddr, 60);

}

int stud\_tcp\_socket(int domain, int type, int protocol)

{

if(domain!=AF\_INET || type!= SOCK\_STREAM || protocol!=IPPROTO\_TCP)

return -1;

current\_tcb = new TCB;

if(tcb\_table==NULL){

tcb\_table = new tcb\_node;

tcb\_table->current = current\_tcb;

tcb\_table->next = NULL;

}

else{

tcb\_node \*head = tcb\_table;

while(head->next != NULL){

head = head->next;

}

head->next = new tcb\_node;

head->next->current = current\_tcb;

head->next->next = NULL;

}

current\_tcb->sockfd = socknum++;

current\_tcb->srcPort = gSrcPort++;

current\_tcb->seq = gSeqNum++;

current\_tcb->ack = gAckNum;

current\_tcb->state = CLOSED;

return current\_tcb->sockfd;

}

int getSockfd(int sockfd)

{

tcb\_node \*current\_p = tcb\_table;

while(current\_tcb != NULL && current\_p->current!=NULL)

{

if(current\_p->current->sockfd == sockfd)

{

current\_tcb = current\_p->current;

return 0;

}

current\_p = current\_p->next;

}

if(current\_p==NULL)

return -1;

}

int stud\_tcp\_connect(int sockfd, struct sockaddr\_in \*addr, int addrlen)

{

if(getSockfd(sockfd)==-1)

return -1;

UINT32 srcAddr = getIpv4Address();

UINT32 dstAddr = htonl(addr->sin\_addr.s\_addr);

current\_tcb->srcAddr = srcAddr;

current\_tcb->dstAddr = dstAddr;

current\_tcb->dstPort = ntohs(addr->sin\_port);

current\_tcb->state = SYN\_SENT;

stud\_tcp\_output(NULL, 0, PACKET\_TYPE\_SYN, current\_tcb->srcPort, current\_tcb->dstPort, srcAddr, dstAddr);

tcphead\* receive = new tcphead;

int res = -1;

while(res == -1)

res = waitIpPacket((char\*)receive, 5000);

stud\_tcp\_input((char \*)receive, 20, ntohl(current\_tcb->srcAddr), ntohl(current\_tcb->dstAddr));

return 0;

}

int stud\_tcp\_send(int sockfd, const unsigned char \*pData, unsigned short datalen, int flags)

{

if(getSockfd(sockfd)==-1)

return -1;

if(current\_tcb->state != ESTABLISHED)

return -1;

UINT32 srcAddr = getIpv4Address();

UINT32 dstAddr = current\_tcb->dstAddr;

current\_tcb->data = new unsigned char(datalen);

strcpy((char\*)current\_tcb->data,(char\*)pData);

stud\_tcp\_output((char \*)current\_tcb->data, datalen, PACKET\_TYPE\_DATA, current\_tcb->srcPort, current\_tcb->dstPort, srcAddr, dstAddr);

tcphead\* receive = new tcphead;

int res = -1;

while(res == -1)

res = waitIpPacket((char\*)receive, 5000);

stud\_tcp\_input((char \*)receive, datalen+20, ntohl(current\_tcb->srcAddr), ntohl(current\_tcb->dstAddr));

return 0;

}

int stud\_tcp\_recv(int sockfd, unsigned char \*pData, unsigned short datalen, int flags)

{

if(getSockfd(sockfd)==-1)

return -1;

if(current\_tcb->state != ESTABLISHED)

return -1;

UINT32 srcAddr = getIpv4Address();

UINT32 dstAddr = current\_tcb->dstAddr;

tcphead\* receive = new tcphead;

int res = -1;

while(res == -1)

res = waitIpPacket((char\*)receive, 5000);

strcpy((char\*)pData,(char\*)receive->data);

datalen=sizeof(pData);

stud\_tcp\_output(NULL, 0, PACKET\_TYPE\_ACK, current\_tcb->srcPort, current\_tcb->dstPort, srcAddr, dstAddr);

return 0;

}

int stud\_tcp\_close(int sockfd)

{

tcb\_node \*current\_p = tcb\_table;

tcb\_node \*preCurrent=current\_p;

while(current\_p != NULL && current\_p->current!=NULL)

{

if(current\_p->current->sockfd == sockfd){

current\_tcb = current\_p->current;

break;

}

preCurrent=current\_p;

current\_p = current\_p->next;

}

if(current\_p==NULL)

return -1;

UINT32 srcAddr = getIpv4Address();

UINT32 dstAddr = current\_tcb->dstAddr;

if(current\_tcb->state != ESTABLISHED)

{

if(current\_p!=preCurrent)

{

preCurrent->next=current\_p->next;

delete current\_p;

}

else

delete current\_tcb;

current\_tcb=NULL;

return -1;

}

stud\_tcp\_output(NULL, 0, PACKET\_TYPE\_FIN\_ACK, current\_tcb->srcPort, current\_tcb->dstPort, srcAddr, dstAddr);

current\_tcb->state = FIN\_WAIT1;

tcphead\* receive = new tcphead;

int res = -1;

while(res == -1)

res = waitIpPacket((char\*)receive, 5000);

stud\_tcp\_input((char \*)receive, 20,ntohl(current\_tcb->srcAddr), ntohl(current\_tcb->dstAddr)); //ack

res = -1;

while(res == -1)

res = waitIpPacket((char\*)receive, 5000);

stud\_tcp\_input((char \*)receive, 20, ntohl(current\_tcb->srcAddr), ntohl(current\_tcb->dstAddr));//fin/ack

return 0;

}