1. 本实验所提供的源代码实现的 TCP 协议是采用停等协议，在 TCP 函数中等待 IP 返回帧来确认该帧是否已经被接受：

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

tcp_output((char *)pData, datalen, flags, gTCB[sockfd].local_port, gTCB[sockfd].remote_port, gTCB[sockfd].local_ip, gTCB[sockfd].remote_ip);

len = waitIpPacket(buffer, 10);

if( len < 20 )
{
    return -1;
}

tcp_input(buffer, len, htonl(gTCB[sockfd].remote_ip), htonl(gTCB[sockfd].local_ip));
```

上述 waitIpPacket()函数应该是在一定时间内接受 IP 返回的报文来确认传输成功

2.

a 提供校验和已经实现：

实现代码如下：

```
unsigned short tcp_calc_checksum(struct MyTCB* pTcb, struct MyTcpSeg* pTcpSeg)
//这个函数用来计算MyTCP头部的checksum校验和（pTcp是TCP的报文，pTcpSeg是TCP的头部字段，返回值成功为校验和，失败为-1）
{
    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];
    }

    sum = sum - p[8] - p[6] + ntohs(p[6]);

    if( (len = pTcpSeg->len) > 20 )
    {
        if( len % 2 == 1 )
        {
            pTcpSeg->data[len - 20] = 0;
            len++;
        }

        for( i=10; i<len/2; i++ )
        {
            sum += ntohs(p[i]);
        }
    }

    sum = sum + (unsigned short)(pTcb->local_ip>>16)
        + (unsigned short)(pTcb->local_ip&0xffff)
        + (unsigned short)(pTcb->remote_ip>>16)
        + (unsigned short)(pTcb->remote_ip&0xffff);
    sum = sum + 6 + pTcpSeg->len;
    sum = ( sum & 0xFFFF ) + ( sum >> 16 );
    sum = ( sum & 0xFFFF ) + ( sum >> 16 );

    return (unsigned short)(~sum);
}
```

这段代码中的 tcp_calc_checksum()为校验和计算函数；

```
int tcp_check(struct MyTCB* pTcb, struct MyTcpSeg* pTcpSeg)
//检查接受到的TCP的checksun校验和是否正确（参数同上，如果正确返回0，错误返回-1）
{
    int i = 0;
    int len = 0;
    unsigned int sum = 0;
    unsigned short* p = (unsigned short*)pTcpSeg;
    unsigned short *pIp;
    unsigned int myip1 = pTcb->local_ip;
    unsigned int myip2 = pTcb->remote_ip;

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

    for( i=0; i<10; i++)
    {
        sum = sum + p[i];
    }
    sum = sum - p[6] + ntohs(p[6]);

    if( (len = pTcpSeg->len) > 20 )
    {
        if( len % 2 == 1 )
        {
            pTcpSeg->data[len - 20] = 0;
            len++;
        }

        for( i=10; i<len/2; i++ )
        {
            sum += ntohs(p[i]);
        }
    }

    sum = sum + (unsigned short)(myip1>>16)
        + (unsigned short)(myip1&0xffff)
        + (unsigned short)(myip2>>16)
        + (unsigned short)(myip2&0xffff);
    sum = sum + 6 + pTcpSeg->len;

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

    if( (unsigned short)(~sum) != 0 )
    {
        // TODO:
        printf("check sum error!\n");

        return -1;
        //return 0;
    }
    else
    {
        return 0;
    }
```

这段代码中的 tcp_check()函数实现了 TCP 的校验；

b 为每个字节分配序列号的功能也已经实现：

```
int tcp_kick(struct MyTCB* pTcb, struct MyTcpSeg* pTcpSeg)
//这个函数用来讲TCP封装好的报文通过IP报文的形式发送出去（两个参数分别是要发送的TCP报文头和数据，成功返回0）
{
    pTcpSeg->checksum = tcp_calc_checksum(pTcb, pTcpSeg);

    convert_tcp_hdr_hton(pTcpSeg);

    tcp_sendIpPkt((unsigned char*)pTcpSeg, pTcpSeg->len, pTcb->local_ip, pTcb->remote_ip, 255);

    if( (pTcb->flags & 0x0f) == 0x00 )
    {
        pTcb->seq += pTcpSeg->len - 20;
    }
    else if( (pTcb->flags & 0x0f) == 0x02 )
    {
        pTcb->seq++;
    }
    else if( (pTcb->flags & 0x0f) == 0x01 )
    {
        pTcb->seq++;
    }
    else if( (pTcb->flags & 0x3f) == 0x10 )
    {
    }

    return 0;
}
```

上述代码中实现了序列号的循环累加，使得每个 TCP 报文都有其独立的序列号；

C 提供确认功能也已经实现，具体函数如问题 1 中一致，每一次发送后都有一定时间用来接收返回的确认帧；

D 该代码没有实现重传机制；