编程模拟实现 rdt2.0 的数据发送和接收

Programming to Conduct Data Sending and Receiving under rdt 2.0 protocol

Rdt (reliable data transfer) is a reliable data transfer protocol conducted on unreliable channel. This project simulate the process with socket programming.

英才学院 2011 级 1 班

刘北北

学号：2011019060027

【基本原理】： 不可靠信道上进行可靠的数据传输

rdt 是在不可靠信道上进行可靠的数据传输协议。它是一系列协议，从 rdt1.0，rdt2.0，rdt2.1，rdt2.2，rdt3.0 逐步完善机制，增加不同的手段，保障数据的可靠传输。

①rdt1.0 是完全没有比特错误，没有分组丢失的情况，即底层信道是完全可靠的；

②rdt2.0 是具有比特错误，没有分组丢失的情况，即网络的物理部件可能出错，这时增加一位接收方的反馈信息 0 或 1（即 ACK 和 NAK），并增加了差错检测；

③rdt2.1 进而考虑了 ACK 和 NAK 分组受损的可能性，数据发送方对数据分组进行编号，如果接收方反馈的 ACK 或 NAK 丢失或损坏，则发送方连着接收两个 ACK_1 或两个 ACK_0，就能知道有 ACK 或 NAK 丢失或损坏；

④rdt2.2 是在具有比特差错的信道上实现的一个无 NAK 的数据传输，发送方通过带序号的 ACK 分组来确认之前的报文是否正确抵达，收到相同编号的两个 ACK 说明发生了错误。

⑤rdt3.0 是既具有比特错误，也有分组丢失的情况，这时发送方增加了定时，若经历了一个特别大的延时，即使数据分组和 ACK 没有丢失，也重传。从 2.2 开始就可以用序号区别重传和一个新的分组了。

本实验用编程实现 rdt2.0 的传输。

【实验思路】：

如图是 rdt2.0 协议的有限状态机示意图。发送方有两个状态：

状态 1：等待上层的调用：此时，应用层把 data 从传给 rdt_send 函数，然后发送方把数据计算校验和 checksum，并用函数 make_pkt 进行打包，形成运输层报文段 sndpkt，

再由不可靠的传输函数 udt_send 发送出去。并把状态转换到状态 2

状态 2：等待 ACK 或 NAK：此时，发送方不能接受任何来自上层的 data，只能等待接收方反馈的运输层报文段 rcvpkt，检查是 ACK 还是 NAK，若是 ACK，则把状态转换到状态 1，若是 NAK，则把状态转换到状态 2。

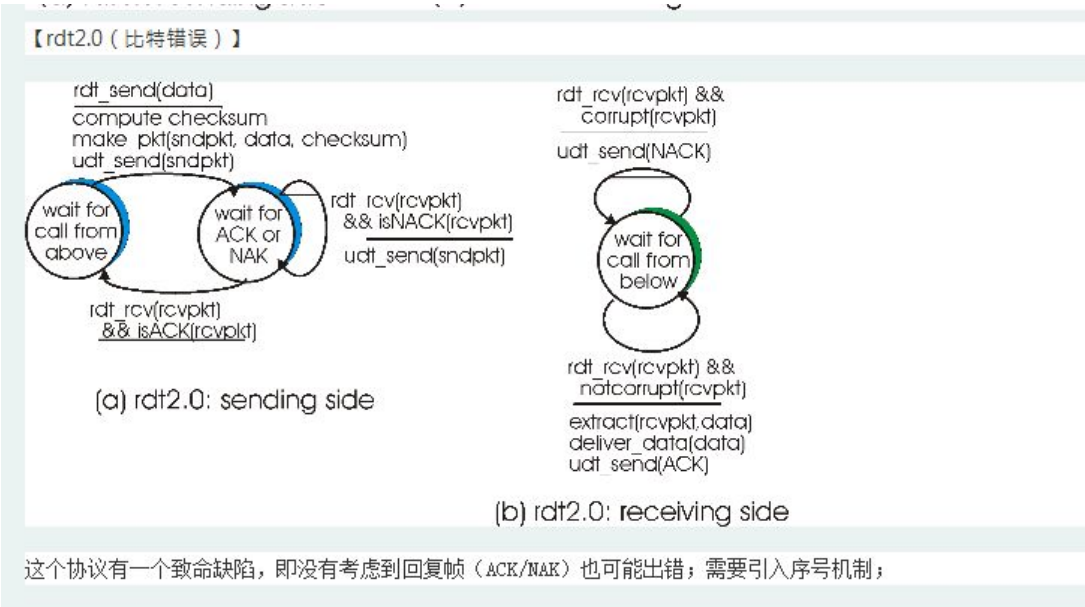
接收方只有一个状态： 等待来自下层的调用：

一旦有链路层报文段 rcvpkt 传给函数 rdt_rcv，即收到报文段，就用函数 corrupt 检测是否被损坏，

⑴若被损坏了，则反馈 NAK 给发送端(这个反馈过程和发送方发送报文段的过程类似，只是不用加校验和 checksum 了)

⑵若没有损坏，则反馈 ACK 给发送端 (过程同发送报文段) , 然后用 extract 函数将包拆开，把其中的 data 提取出来，用 deliever 函数把 data 送给上层（应用层）

示意图如下：



【实验过程】：

1. 先写实现各个小功能的函数：

（I）发送端：

① 发送端的初始化，建立套接字 sockfd_send，使用 udp 协议，并配置本地地址：

```
//sender初始化
void udt_init_send()
{
    sockfd_send = socket(AF_INET, SOCK_DGRAM, 0);
                        //建立套接字sockfd_recv,SOCK_DGRAM表明使用udp协议

    send_addr.sin_family = AF_INET; //协议族使用tcp/ip协议
    send_addr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
                        //操作系统可能随时增减IP地址,所以用INADDR_ANY表示任意地址
    send_addr.sin_port = htons(SERV_PORT);

}
```

②计算校验和：

```
//计算校验和checksum:
int checksum(unsigned long *buffer, int size)
{
    unsigned long cksum=0;

    while (size > 1)
    {
        cksum += *buffer++;  //将buffer中的各个位求和
        size -= sizeof(unsigned long);  //修改buffer长度
    }
    if (size)
    {
        cksum += *(unsigned long *)buffer;
    }
    //待校验的数据按16位位一个单位相加，采用端循环进位，最后对所得16位的数据取反码。
    cksum = (cksum >> 16) + (cksum & 0xffff);   //cksum & 0xffff是保留低16位（即清除进位），
                                                //整个式子是将高16位挪到低16位，与低16位相加。即循环相加。
    cksum += (cksum >>16);  //将超过16位的进位加到最低位
    printf("checksum:%hu\n",cksum);
    return (unsigned long)(cksum);
}
```

③将数据打包，把校验和装到包内：

```
//将数据打包成pkt,打包好的包用参数char * pkt返回
void make_pkt(char * data, char * pkt )
{
    char sum[3];
    char len[3];
    len[2] = '\0';
    sum[2] = '\0';
    len[0] = ( strlen(data) >> 8 ) + 1;
    len[1] = strlen(data) + 1;

    strcpy(pkt, len); //这两行是把数据长度值加在包的开头
    strcat(pkt, data);

    sum[0] = ( checksum( (unsigned long *) pkt, sizeof(pkt)) >> 8 ) + 2;
    sum[1] = checksum( (unsigned long *) pkt, sizeof(pkt)) + 2;

    strcat(pkt, sum); //把校验和加在数据最后
}
```

④ 数据的 udt 不可靠发送和反馈的 udt 不可靠接收(即使用 udp 协议的通信)：

发送数据：使用套接字 sockfd_send，将数据 data 通过接口送入运输层；

接受反馈：使用套接字 sockfd_send，把接收方反馈的 ACK 或 NAK 存入缓存 buf 中。

（注：两个过程均设置错误提示“error”）

```
//sender发送数据给receiver
void udt_send(char * data)
{
    int n;
    n = sendto(sockfd_send, data, strlen(data), 0, (struct sockaddr *)&recv_addr, sizeof(recv_addr) );
    if (n==-1)
        printf("udt:sendto error!");
}

//sender从receiver接收反馈的ACK或NAK
void udt_recv_respond(char * buf)
{
    int n;
    n = recvfrom(sockfd_send, buf, MAXLINE, 0,NULL,0 );
    if (n==-1)
        printf("udt:recvfrom error");
}
```

（II）接收端：

① 发送端的初始化，建立套接字 sockfd_recv，使用 udp 协议，并配置本地地址：

```
//接收端初始化
void udt_init_recv()
{
    sockfd_recv = socket(AF_INET, SOCK_DGRAM, 0); //建立套接字sockfd_recv,SOCK_DGRAM表明使用udp协议

    recv_addr.sin_family = AF_INET; //协议族使用tcp/ip协议
    recv_addr.sin_addr.s_addr = htonl(INADDR_ANY); //操作系统可能随时增减IP地址,所以用INADDR_ANY表示任意地址
    recv_addr.sin_port = htons(SERV_PORT);

    bind(sockfd_recv, (struct sockaddr *)&recv_addr, sizeof(recv_addr) );
    //应用程序（用套接字sockfd_recv标明）要把服务器的某地址（recv_addr）上的某端口（recv_addr.sin_port）占为己用。
    //调用bind()的时候，相当于告诉操作系统：我需要在SERV_PORT端口上侦听，
    //所以发送到服务器的这个端口的不管是哪个网卡/哪个IP地址接收到的数据，都是由我处理的。
    //这时候，服务器程序则在0.0.0.0这个地址（即INADDR_ANY）上进行侦听。

    send_addr_len = sizeof(send_addr); //对方地址长度
}
```

②计算校验和的函数和发送方一样；

③从收到的包中提取数据、长度和校验和：

```
//从收到的包中提取数据：
void extract(char* pkt,int* len, char* data,int* sum)
{
    //把pkt拆成三部分后分别由参数*len、*data、*sum返回
    unsigned long int l;
    l=strlen(pkt);

    *len = (*pkt - 1) * 256 + ( *(pkt+1) - 1 );
    *sum = (*(pkt+l-2) - 2) * 256 + ( *(pkt+l-1) - 2 );

    *(pkt+l-2) = '\0';
    *(pkt+l-1) = '\0';
    strcpy(data, pkt+2 );
}
```

④ 数据的 udt 不可靠接收和反馈的 udt 不可靠发送(即使用 udp 协议的通信)：

数据接收：使用套接字 sockfd_recv ,将运输层的报文段通过端口传给接收缓存 rcvpkt (由指针 data 传参)；

反馈发送：使用套接字 sockfd_recv，将产生的反馈信息（ ACK 或 NAK)(它们存在发送缓存 sndpkt 中) 通过端口送给运输层。

（注：两个过程均设置错误提示“error”)

```
//receiver从sender接收数据
void udt_recv(char * data)
{
    int n;
    n = recvfrom(sockfd_recv, data, MAXLINE, 0,(struct sockaddr *)&send_addr, &send_addr_len );
    if (n==-1)
        printf("udt:recvfrom error");
}

//receiver反馈ACK或NAK给sender
void feedback(char* buf)
{
    int n;
    n = sendto(sockfd_recv, buf, strlen(data), 0, (struct sockaddr *)&send_addr, sizeof(send_addr) );
    if (n==-1)
        printf("udt:sendto error!");
}
```

2. 发送端和接收端交换开始通信，实现有限状态机的几个状态之间的转换：

此时的发送和接收函数与之前的模块不同。之前的模块实现的是不可靠的传输（ udt),现在的总的发送函数实现的是可靠的传输，因为加入了检错和反馈机制。

发送端：

```
//发送端发送数据：（不用传参数flag，因为此函数执行完只有一种新状态，即状态2）
void rdt_send(char * data,unsigned int time)
{
    char sndpkt[PKTLEN];
    char rcvpkt[PKTLEN];
    unsigned int  len,sum,tmp;
    char pkt[80];
    tmp = time; //tmp作为定时器,记要发送的内容的时长

    memset(rcvpkt, 0, sizeof(rcvpkt) );//将接收和发送的缓存数组清零
    memset(sndpkt, 0, sizeof(sndpkt) );
    udt_init_send(); //发送端（sender)初始化

    while(1){
        make_pkt(data, sndpkt); //将数据打包
        if(tmp--!= 0){ //tmp倒计时，若没得0，说明还没把内容向缓存中输送完
            *sndpkt += 1;
        }
        udt_send(sndpkt); //要发送的内容已经向缓存中输送完了。sender向发receiver送数据

        printf("Waiting ACK or NAK..\n");
        return;
    }
}
```

```
//发送端接收确认：

int rdt_rcv(char * pkt,int flag)
{
    char sndpkt[PKTLEN]; //先定义发送和接收的缓存数组
    char rcvpkt[PKTLEN];
    unsigned int sum,len,tmp;
    tmp = time; //tmp作为计时器，若很长时间没收到反馈，则认为未顺利到达。

    udt_init_send(); //发送端初始化

    while(1){
        memset(rcvpkt, 0, sizeof(rcvpkt) ); //将接收的缓存数组清零

        if(tmp--!= 0){ //在其限定的时间内
            udt_recv_respond(rcvpkt); //从接收方接收回应（ACK或NAK)
            if ( strcmp(rcvpkt, "ACK") == 0 ){  //收到ACK，显示接收时间
                printf("ACK Received\n\n");
                printf("Receive time : %hu\n", time ); //%hu是短整型
                flag=1; //状态变为1，即进入等待上层调用的状态
                return;
            }
            else{
                printf("NAK Received\n"); //收到NAK
                printf("Wrong response\n \n");
                flag=2; //状态变为2，即仍在等待接收方反馈的状态，不能接收上层的数据

            }
        }
    return flag;
}
```

发送端主函数：

```c
//发送端主函数
int main()
{
    WORD wVersionRequested;
    WSADATA wsaData;
    int err;
    int recv_addr_len,send_addr_len;
    char buf[MAXLINE];
    char data[MAXLINE];
    int flag=1; //flag=1 表示在状态1：等待上层调用；
                //flag=2 表示在状态2：等待ACK或NAK

    //WSA初始化，验证版本号
    err = WSAStartup( MAKEWORD( 1, 1 ), &wsaData );
    if ( err != 0 ) {
        return -1;
    }
    if ( LOBYTE( wsaData.wVersion ) != 1 ||HIBYTE( wsaData.wVersion ) != 1 ) {
        WSACleanup( );
        return -1;
    }

    udt_init_send(); //客户机端初始化
    udt_send(data); //sender发送数据给receiver
    flag=2;
    rdt_rcv(buf,flag); //sender从receiver接收数据
    WSACleanup();
    return 0;
}
```

接收端：

```c
//接收端接收数据并发送反馈：
void rdt_rcv_snd(char * pkt)
{
    char sndpkt[PKTLEN];
    char rcvpkt[PKTLEN];
    unsigned int sum,len;
    udt_init_recv();

    while(1){
        memset(rcvpkt, 0, sizeof(rcvpkt) );
        udt_recv(rcvpkt);
        extract(rcvpkt, &len, pkt, &sum );
        printf("extract sum :%hu\n",sum);
        if (corrupt(rcvpkt, sum)){  //通过检验校验和，如果发现包没有被破坏
            printf("Right packet Received!\n");
            printf("ACK sended!\n\n");
            printf("Data Received: %s \n\n",pkt);
            feedback("ACK"); //用udp协议发送ACK
            return;
        }
        else { //包被破坏了
            printf("Wrong packet Received!\n");
            printf("NAK sended!\n");
            printf("Waiting for resend...\n\n");
            feedback("NAK"); //用udp协议发送NAK
        }
    }
}
```

接收端主函数：

```
//接收端主函数：（接收端没有flag，因为它只有一种状态）
int main()
{
    WORD wVersionRequested;
    WSADATA wsaData;
    int err;
    int recv_addr_len,send_addr_len;
    char buf[MAXLINE];
    char data[MAXLINE];

    //WSA初始化，验证版本号
    err = WSAStartup( MAKEWORD( 1, 1 ), &wsaData );
    if ( err != 0 ) {
        return -1;
    }
    if ( LOBYTE( wsaData.wVersion ) != 1 ||HIBYTE( wsaData.wVersion ) != 1 ) {
        WSACleanup( );
        return -1;
    }
    udt_init_recv(); //接收端初始化
    rdt_rcv_snd(buf); //receiver从sender接收数据

    WSACleanup();
    return 0;
}
```

---

3. 学习了一种新的编程方法——写.h 文件

.h 文件和.c 文件配合使用，在.c 文件头部#include .h 文件，可以使软件结构清晰，实现软件的模块化。函数定义要放在.c 文件中，在.h 文件中写一些宏定义和变量、函数声明，编译后不产生代码。

【实验结果与收获】：

1. 通过本次实验，我对 windows 操作系统的 API 有了进一步了解。并且通过具体写代码，深入理解了 rdt2.0 的通信过程，包括一些反馈和检错的机制，实现不可靠信道上的可靠传输。

2. 我的程序经编译后还有一定的问题，出现死循环，有待进一步改进（这次时间关系还没调出来）。但整个框架、算法的设计还是使我的思路更加清晰。

【附：代码】

```
//////////////////////发送端////////////////////////////////

#include <stdio.h>
#include <WINSOCK2.h>
#pragma  comment(lib,"ws2_32.lib")

#define MAXLINE 80
#define SERV_PORT 8000
#define PKTLEN 80


SOCKADDR_IN recv_addr,send_addr; //地址

SOCKET sockfd_recv,sockfd_send; //套接字

int send_addr_len;


//sender 初始化

void udt_init_send()
{
    sockfd_send = socket(AF_INET, SOCK_DGRAM, 0);

                        //建立套接字 sockfd_recv,SOCK_DGRAM 表明使用 udp 协议

    send_addr.sin_family = AF_INET; //协议族使用 tcp/ip 协议

    send_addr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);

                        //操作系统可能随时增减 IP 地址,所以用 INADDR_ANY 表示

任意地址

    send_addr.sin_port = htons(SERV_PORT);
}

//sender 发送数据给 receiver

void udt_send(char * data)
{
    int n;
    n = sendto(sockfd_send, data, strlen(data), 0, (struct sockaddr *)&recv_addr,
sizeof(recv_addr) );
    if (n==-1)
        printf("udt:sendto error!");
}

//sender 从 receiver 接收反馈的 ACK 或 NAK
```

```
void udt_recv_respond(char * buf)
{
    int n;
    n = recvfrom(sockfd_send, buf, MAXLINE, 0,NULL,0 );
    if (n==-1)
        printf("udt:recvfrom error");
}
```

//计算校验和 checksum：

```
int checksum(unsigned long *buffer, int size)
{
    unsigned long cksum=0;

    while (size > 1)
    {
        cksum += *buffer++;   //将 buffer 中的各个位求和

        size -= sizeof(unsigned long);   //修改 buffer 长度
    }
    if (size)
    {
        cksum += *(unsigned long *)buffer;
    }
```

//待校验的数据按 16 位位一个单位相加，采用端循环进位，最后对所得 16 位的数据取反码。

```
    cksum = (cksum >> 16) + (cksum & 0xffff);  //cksum & 0xffff 是保留低 16 位（即清除进位），
                                                //整个式子是将高 16 位挪到低 16 位，与低 16 位相加。即循环相加。

    cksum += (cksum >>16);   //将超过 16 位的进位加到最低位
    printf("checksum:%hu\n",cksum);
    return (unsigned long)(cksum);
}
```

//将数据打包成 pkt,打包好的包用参数 char * pkt 返回

```
void make_pkt(char * data, char * pkt )
```

```
{
    char sum[3];
    char len[3];
    len[2] = '\0';
    sum[2] = '\0';
    len[0] = ( strlen(data) >> 8 ) + 1;
    len[1] = strlen(data) + 1;

    strcpy(pkt, len); //这两行是把数据长度值加在包的开头

    strcat(pkt, data);
    sum[0] = ( checksum( (unsigned long *) pkt, sizeof(pkt)) >> 8 ) + 2;
    sum[1] = checksum( (unsigned long *) pkt, sizeof(pkt)) + 2;

    strcat(pkt, sum); //把校验和加在数据最后

}

//发送端发送数据：( 不用传参数 flag，因为此函数执行完只有一种新状态，即状态 2 )

void rdt_send(char * data,unsigned int time)
{
    char sndpkt[PKTLEN];
    char rcvpkt[PKTLEN];
    unsigned int   tmp;
//  char pkt[80];

    tmp = time; //tmp 作为定时器,记要发送的内容的时长


    memset(rcvpkt, 0, sizeof(rcvpkt) );//将接收和发送的缓存数组清零

    memset(sndpkt, 0, sizeof(sndpkt) );

    udt_init_send(); //发送端 ( sender)初始化

    while(1){

        make_pkt(data, sndpkt); //将数据打包


        if(tmp--!= 0){ //tmp 倒计时，若没得 0，说明还没把内容向缓存中输送完

            *sndpkt += 1;
        }

        udt_send(sndpkt); //要发送的内容已经向缓存中输送完了。sender 向发

receiver 送数据

        printf("Waiting ACK or NAK..\n");
        return;
    }
```

```
}
//发送端接收确认:
int rdt_rcv(char * pkt,int flag)
{
    char sndpkt[PKTLEN]; //先定义发送和接收的缓存数组
    char rcvpkt[PKTLEN];
    unsigned int sum,len,tmp;
    tmp = time; //tmp 作为计时器，若很长时间没收到反馈，则认为未顺利到达。
    udt_init_send(); //发送端初始化
    while(1){
        memset(rcvpkt, 0, sizeof(rcvpkt) ); //将接收的缓存数组清零
        if(tmp--!= 0){ //在其限定的时间内
            udt_recv_respond(rcvpkt); //从接收方接收回应（ACK 或 NAK)
            if ( strcmp(rcvpkt, "ACK") == 0 ){   //收到 ACK，显示接收时间
                printf("ACK Received\n\n");
                printf("Receive time : %hu\n", time ); //%hu 是短整型
                flag=1; //状态变为 1，即进入等待上层调用的状态
                return;
            }
            else{
                printf("NAK Received\n"); //收到 NAK
                printf("Wrong response\n \n");
                flag=2; //状态变为 2，即仍在等待接收方反馈的状态，不能接收上层的
数据
            }
    }
    return flag;
}
//发送端主函数
int main()
```

```
{
    WORD wVersionRequested;
    WSADATA wsaData;
    int err;
    int recv_addr_len,send_addr_len;
    char buf[MAXLINE];
    char data[MAXLINE];

    int flag=1; //flag=1 表示在状态 1：等待上层调用；

                //flag=2 表示在状态 2：等待 ACK 或 NAK

     //WSA 初始化，验证版本号

    err = WSAStartup( MAKEWORD( 1, 1 ), &wsaData );
    if ( err != 0 ) {
        return -1;
    }
    if ( LOBYTE( wsaData.wVersion ) != 1 ||HIBYTE( wsaData.wVersion ) != 1 ) {
        WSACleanup( );
        return -1;
    }

    udt_init_send(); //客户机端初始化

    udt_send(data); //sender 发送数据给 receiver

    flag=2;

    rdt_rcv(buf,flag); //sender 从 receiver 接收数据

    WSACleanup();
    return 0;
}

//////////////////////////接收端/////////////////////////

#include <stdio.h>
#include <WINSOCK2.h>
#pragma  comment(lib,"ws2_32.lib")

#define MAXLINE 80
#define SERV_PORT 8000
#define PKTLEN 80

SOCKADDR_IN recv_addr,send_addr; //地址
```

```
SOCKET sockfd_recv,sockfd_send; //套接字
int send_addr_len;
//接收端初始化
void udt_init_recv()
{
    sockfd_recv = socket(AF_INET, SOCK_DGRAM, 0); //建立套接字
sockfd_recv,SOCK_DGRAM 表明使用 udp 协议

    recv_addr.sin_family = AF_INET; //协议族使用 tcp/ip 协议
    recv_addr.sin_addr.s_addr = htonl(INADDR_ANY); //操作系统可能随时增减 IP
地址,所以用 INADDR_ANY 表示任意地址
    recv_addr.sin_port = htons(SERV_PORT);
    bind(sockfd_recv, (struct sockaddr *)&recv_addr, sizeof(recv_addr) );
    //应用程序（用套接字 sockfd_recv 标明）要把服务器的某地址（recv_addr）上的
某端口（recv_addr.sin_port）占为已用。
    //调用 bind()的时候，相当于告诉操作系统：我需要在 SERV_PORT 端口上侦听，
    //所以发送到服务器的这个端口的不管是哪个网卡/哪个 IP 地址接收到的数据，都是
由我处理的。
    //这时候，服务器程序则在 0.0.0.0 这个地址（即 INADDR_ANY）上进行侦听。

    send_addr_len = sizeof(send_addr); //对方地址长度
}
//receiver 从 sender 接收数据
void udt_recv(char * data)
{
    int n;
    n = recvfrom(sockfd_recv, data, MAXLINE, 0,(struct sockaddr *)&send_addr,
&send_addr_len );
```

```
    if (n==-1)
        printf("udt:recvfrom error");
}

//receiver 反馈 ACK 或 NAK 给 sender

void feedback(char* buf)
{
    int n;
    n = sendto(sockfd_recv, buf, strlen(buf), 0, (struct sockaddr *)&send_addr,
sizeof(send_addr) );
    if (n==-1)
        printf("udt:sendto error!");
}

//从收到的包中提取数据：

void extract(char* pkt,int* len, char* data,int* sum)
{

    //把 pkt 拆成三部分后分别由参数*len、*data、*sum 返回

    unsigned long int l;
    l=strlen(pkt);
    *len = (*pkt - 1) * 256 + ( *(pkt+1) - 1 );
    *sum = (*(pkt+l-2) - 2) * 256 + ( *(pkt+l-1) - 2 );
    *(pkt+l-2) = '\0';
    *(pkt+l-1) = '\0';
    strcpy(data, pkt+2 );
}

//计算校验和 checksum：

int checksum(unsigned long *buffer, int size)
{
    unsigned long cksum=0;
    while (size > 1)
    {

        cksum += *buffer++;   //将 buffer 中的各个位求和

        size -= sizeof(unsigned long);   //修改 buffer 长度

    }
    if (size)
    {
        cksum += *(unsigned long *)buffer;
    }

    //待校验的数据按 16 位位一个单位相加，采用端循环进位，最后对所得 16 位的数据
```

取反码。

```
    cksum = (cksum >> 16) + (cksum & 0xffff);  //cksum & 0xffff 是保留低 16 位（即
```

清除进位），

```
    //整个式子是将高 16 位挪到低 16 位，与低 16 位相加。即循环相加。

    cksum += (cksum >>16);   //将超过 16 位的进位加到最低位

    printf("checksum:%hu\n",cksum);
    return (unsigned long)(cksum);
}
//判断包是否被破坏的函数。本程序只判断校验和，若还有其他检验，一并加进来。
int corrupt(char * pkt,int sum )
{   //若完好，返回 1，若被破坏，返回 0。
    return ( checksum((int * )pkt, sizeof(pkt)) ) == (sum) ? 1 : 0 ;
}
//将数据打包成 pkt
void make_pkt(char * data, char * pkt )
{
    char sum[3];
    char len[3];
    len[2] = '\0';
    sum[2] = '\0';
    len[0] = ( strlen(data) >> 8 ) + 1;
    len[1] = strlen(data) + 1;

    strcpy(pkt, len); //这两行是把数据长度值加在包的开头

    strcat(pkt, data);
    sum[0] = ( checksum( (unsigned long *) pkt, sizeof(pkt)) >> 8 ) + 2;
    sum[1] = checksum( (unsigned long *) pkt, sizeof(pkt)) + 2;

    strcat(pkt, sum); //把校验和加在数据最后

}
//接收端接收数据并发送反馈：
void rdt_rcv_snd(char * pkt)
{
    int i=0;
    //char sndpkt[PKTLEN];
```

```
    char rcvpkt[PKTLEN];
    unsigned int sum,len;
    udt_init_recv();
    while(1){
        memset(rcvpkt, 0, sizeof(rcvpkt) );
        udt_recv(rcvpkt);
        extract(rcvpkt, &len, pkt, &sum );
        printf("extract sum :%hu\n",sum);

        if (corrupt(rcvpkt, sum)){   //通过检验校验和，如果发现包没有被破坏

            printf("Right packet Received!\n");
            printf("ACK sended!\n\n");
            printf("Data Received: %s \n\n",pkt);

            feedback("ACK"); //用 udp 协议发送 ACK

            return;
        }

        else { //包被破坏了

            printf("Wrong packet Received!\n");
            printf("NAK sended!\n");
            printf("Waiting for resend...\n\n");

            feedback("NAK"); //用 udp 协议发送 NAK

        }
    }
}

//接收端主函数：( 接收端没有 flag，因为它只有一种状态 )

int main()
{
//   WORD wVersionRequested;
    WSADATA wsaData;
    int err;
    //int recv_addr_len,send_addr_len;
    char buf[MAXLINE];
    //char data[MAXLINE];


    //WSA 初始化，验证版本号

    err = WSAStartup( MAKEWORD( 1, 1 ), &wsaData );
    if ( err != 0 ) {
        return -1;
    }
```

```
    if ( LOBYTE( wsaData.wVersion ) != 1 ||HIBYTE( wsaData.wVersion ) != 1 ) {
        WSACleanup( );
        return -1;
    }

    udt_init_recv(); //接收端初始化

    rdt_rcv_snd(buf); //receiver 从 sender 接收数据


    WSACleanup();
    return 0;
}
```