

**计算机网络实验报告**

**实验3-1**

****

专 业 信息安全

学 号 2113662

姓 名 张丛

班 级 信安一班

1. **实验目的**

利用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：建立连接、差错检测、接收确认、超时重传等。流量控制采用停等机制，完成给定测试文件的传输。

1. **实验要求**

（1）实现单向数据传输（一端发数据，一端返回确认）。

（2）对于每个任务要求给出详细的协议设计。

（3）完成给定测试文件的传输，显示传输时间和平均吞吐率。

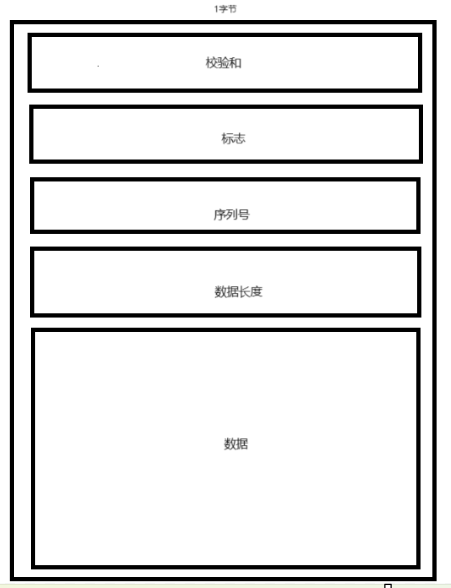
（4）性能测试指标：吞吐率、延时，给出图形结果并进行分析。

1. **实验内容**

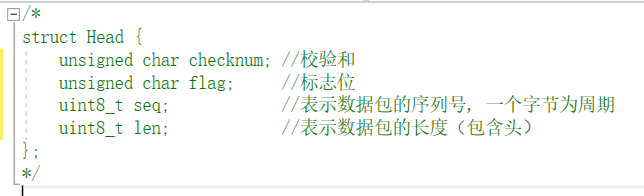
## 协议设计：

* 数据报文划分为头部和数据两部分
* 头部包含：校验和、标志位、序列号、数据长度

简易表示如下：



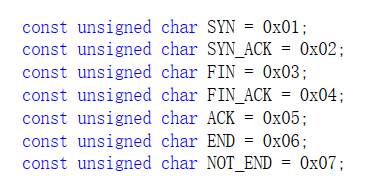
定义报首如下：



其中：

**校验和**用于差错检测；

**标志位**的内容在实验3-1中有如下定义：



前四个标志用于握手和挥手阶段，后三者主要用于数据传输阶段；

**序列号**所在的序列空间为0~99；

**数据包长度**为 MAX\_UDP\_LEN+4 = 10004 字节。（除了最后一个可能没有达到最大长度的数据包。）

## 差错检测：

发送方发送数据包时会计算报首和数据的校验和，将结果放到报首的校验和字段。

接收方在接收数据后，会对整个数据包计算校验和，此时校验和为0则证明传输过程中没有发生数据损坏，完成了差错检测。

计算检验和代码如下：

1. unsigned char check\_sum(char\* package, int len)
2. {
3. if (len == 0) {
4. return ~(0);
5. }
6. unsigned long sum = 0;
7. int i = 0;
8. while (len--) {
9. sum += (unsigned char)package[i++];
10. if (sum & 0xFF00) {
11. sum &= 0x00FF;
12. sum++;
13. }
14. }
15. return ~(sum & 0x00FF);
16. }

实现思路是：遍历数据包中的每个字节，将其加到 sum 中；如果 sum 的高位不为 0，将高位清零，然后将 sum 加 1；最后，返回 sum 的补码取反。

## 三次握手与两次挥手：

通过三次握手与两次挥手实现了建立连接和断开连接的过程。

三次握手与四次挥手在上一次实验中已经进行验证，对原理和过程也有较为详细的说明，这里不再赘述。

在3-1中采用两次挥手主要是因为此次实验中不需要担心有未完成的任务，两次挥手即可进行断开。

发送端（Client）的三次握手代码如下：

1. void three\_way\_handshake()
2. {
3. *//三次握手建立连接*
4. while (true)
5. {
6. char sendBuf[2];
7. sendBuf[1] = SYN;
8. sendBuf[0] = check\_sum(sendBuf + 1, 1);
9. int ret = sendto(m\_ClientSocket, sendBuf, 2, 0, (sockaddr\*)&m\_ServerAddress, sizeof(m\_ServerAddress));
10. if (ret != SOCKET\_ERROR)
11. cout << "已发送第一次握手请求SYN" << endl;
12. else {
13. cout << "发送握手请求失败……" << endl;
14. return;
15. }
16. *//开始计时*
17. int begin = clock();
18. char recv[2];  *//接收缓存区*
19. int is\_timeout = 0;
20. *//cout << "测试点2" << endl;*
21. u\_long mode = 1;  *// 1 表示非阻塞，0 表示阻塞*
22. ioctlsocket(m\_ClientSocket, FIONBIO, &mode);
23. while (recvfrom(m\_ClientSocket, recv, 2, 0, (sockaddr\*)&m\_ServerAddress, &len\_addr) == SOCKET\_ERROR)
24. {
25. if (clock() - begin > TIMEOUT)
26. {
27. is\_timeout = 1;
28. break;
29. }
30. }
31. *//没有超时，接收到的数据包检验正确，是二次握手*
32. if (is\_timeout == 0 && check\_sum(recv, 2) == 0 && recv[1] == SYN\_ACK)
33. {
34. cout << "接收到服务端二次握手SYN\_ACK" << endl;
35. sendBuf[1] = ACK;
36. sendBuf[0] = check\_sum(sendBuf + 1, 1);
37. sendto(m\_ClientSocket, sendBuf, 2, 0, (sockaddr\*)&m\_ServerAddress, sizeof(m\_ServerAddress));
38. cout << "已发送第三次握手ACK" << endl;
39. break;
40. }
41. }
42. cout << "连接到服务端" << endl;
43. return;
44. }

其余代码拥有相似性，不再赘述。

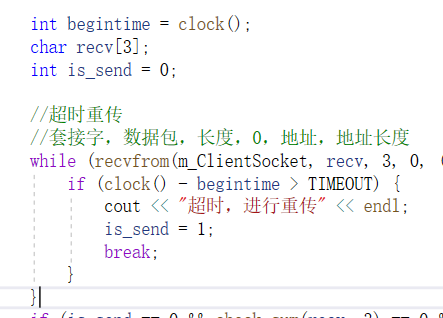
## 等停机制和确认重传：

**确认重传：**

维护了一个计时器，自发送方发送数据包并开始尝试捕获ACK时开启计时，此步骤需要接受有效ACK，每超时一次重发一个包并重置计时器；

还可维护一个重发计数器，若超过最大重发次数则发送失败。

如下：



**停等机制：**

未能通过差错检测的包将不予回复ACK；

必须等对方发送一条消息我方才能发一条消息；

## 依据主函数的脉络来讲解剩余的内容：

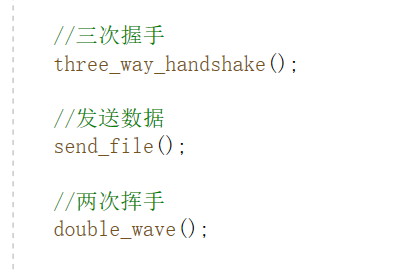
在了解了上述的机制后，为了不显得杂乱，我们沿着主函数来看。（以发送方为例）

首先是进行环境初始化、套接字的创建、套接字的绑定，这些步骤与实验一的TCP聊天程序是类似的，但此次实验使用UDP。

如下：



然后进行此次实验的三大块：先进行三次握手，建立连接后发送数据，在发送数据后，进行两次挥手。



发送数据（发送文件）的代码如下：

1. void send\_file()
2. {
3. *// 发送文件*
4. int len = 0;
5. char\* buffer = new char[100000000];
6. string filename;
7. cout << "请输入发送的文件名" << endl;
8. while (true)
9. {
10. cin >> filename;
11. ifstream fin(filename.c\_str(), ifstream::binary);*//打开输入的文件，以二进制模式读取*
12. if (!fin) {
13. cout << "文件名有误,请重新输入" << endl;
14. continue;
15. }
16. unsigned char t = fin.get();*//按字节读取数据*
17. while (fin) {
18. buffer[len++] = t;
19. t = fin.get();
20. }
21. fin.close();
22. break;
23. }
24. int time\_begin = clock();
25. send\_buffer((char\*)(filename.c\_str()), filename.length()); *//文件名*
26. send\_buffer(buffer, len);         *//文件内容*
27. int time\_end = clock();
28. cout << filename << "文件传输完毕" << endl;
29. int time = (time\_end - time\_begin) / CLOCKS\_PER\_SEC;
30. cout << "传输时间：" << time << endl;
31. if (time != 0)
32. {
33. double kbps = ((len \* 8.0) / 1000) / time;
34. cout << "吞吐量：" << kbps << " kbps" << endl;
35. }
36. cout << "丢包率：" << (SUCCESS\_SEND - SUCCESS\_RECV) \* 100 / SUCCESS\_SEND << "%" << endl;
37. cout << endl << endl;
38. }

首先是根据输入的文件名打开文件，按字节读取，即可以一次性读取所有字节避免反复打开文件（但文件过大时可能缓存区溢出），也可以按数据包的字节大小多次读取。我采用的是一次性读取。

通过**send\_buff()**发送完数据后，会输出传输时间、吞吐量、丢包率这些信息。

send\_buff()主要实现：将文件数据分为一个一个数据包并赋予序列号然后发通过send\_package()发送，且对于最后一个数据包有赋予END标志，否则NOT\_END。

而关键的**send\_package()**代码如下：

1. bool send\_package(char\* start, int length, uint8\_t req, int is\_END = 0) {
2. char\* package;
3. int Len = length + 4;     *//数据包长度（含头4字节）*
4. package = new char[length + 4];   *//数据包*
5. *//组装数据包*
6. if (is\_END) {
7. package[1] = END;        *//标志位*
8. }
9. else
10. {
11. package[1] = NOT\_END;
12. }
13. package[2] = req;      *//序号*
14. package[3] = length;     *//数据长度（不含头）*
15. for (int i = 0; i < length; i++) {*// 数据*
16. package[i + 4] = start[i];
17. }
18. package[0] = check\_sum(package + 1, Len-1);*//校验和*
19. while (1) {
20. *//套接字，数据包，长度，0，目的地址，地址大小*
21. sendto(m\_ClientSocket, package, Len, 0, (sockaddr\*)&m\_ServerAddress, sizeof(m\_ServerAddress));
22. SUCCESS\_SEND++;
23. int begintime = clock();
24. char recv[3];
25. int is\_send = 0;
26. *//超时重传*
27. *//套接字，数据包，长度，0，地址，地址长度*
28. while (recvfrom(m\_ClientSocket, recv, 3, 0, (sockaddr\*)&m\_ServerAddress, &len\_addr) == SOCKET\_ERROR) {
29. if (clock() - begintime > TIMEOUT) {
30. cout << "超时，进行重传" << endl;
31. is\_send = 1;
32. break;
33. }
34. }
35. if (is\_send == 0 && check\_sum(recv, 3) == 0 && recv[1] == ACK && recv[2] == (uint8\_t)req)
36. {
37. cout << "接收到ACK： " << int(recv[2])<< endl;
38. SUCCESS\_RECV++;
39. return true;
40. }
41. }
42. }

可见，send\_package()依次实现了：组装数据包并计算校验和，发送数据包，等待接收正确的ACK；

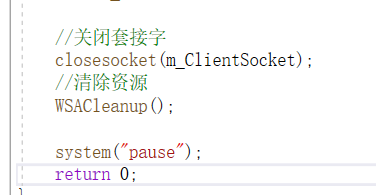
只有在接收到正确的ACK后，才会发送下一个数据包；

如果计时器超时则重新发送数据包。

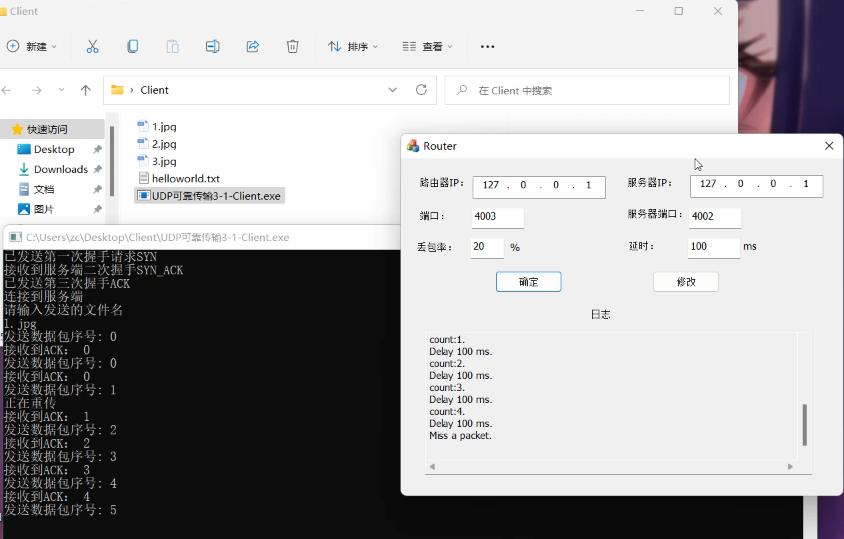
这些步骤保证了UDP的可靠传输。

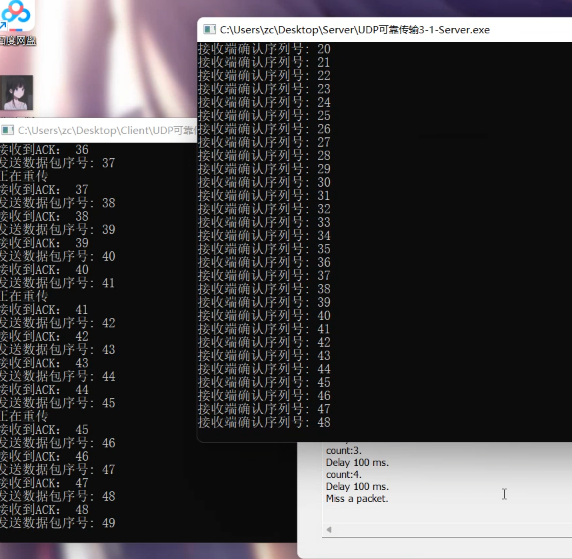
主函数在进行两次挥手后，则进行套接字关闭、清理资源。

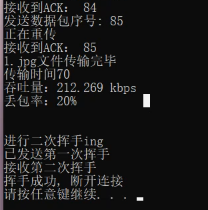
如下：



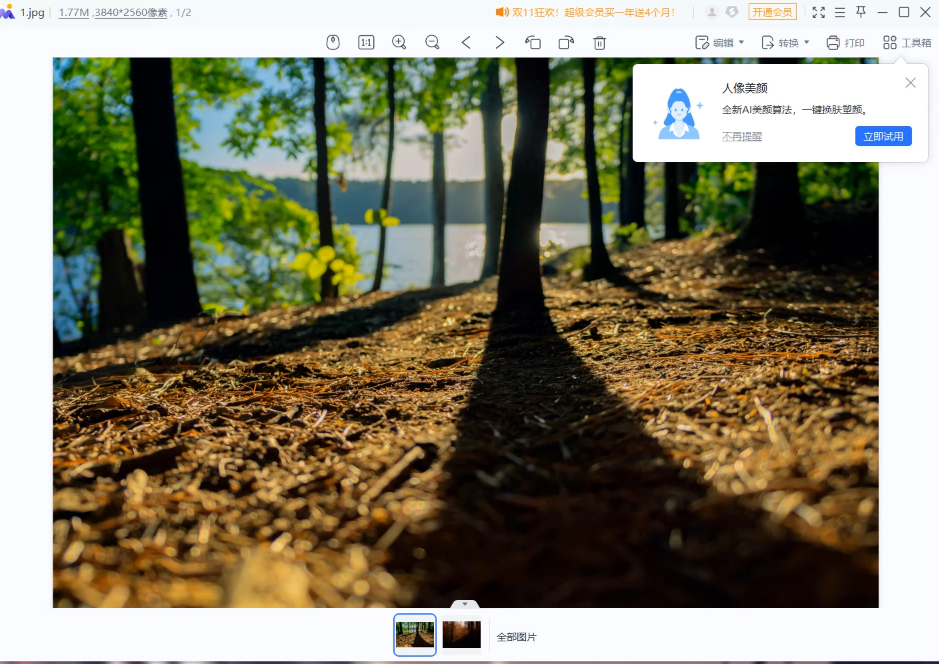
实验效果如下：







文件也确实成功发送了：



1. **思考与总结**

在检查时，经过学长指点，发现代码中确实有一个bug：

1. 发送方序列号为X的发送数据包；
2. 接收方接收，发送ACK\_X，下一个有效的序列号为X+1；
3. 接收方的ACK\_X丢失；
4. 发送方等待超时，重新发送序列号为X的数据包；
5. 接收方收到X,丢弃;
6. 从此陷入超时-发送-丢弃的循环。

解决的办法也很简单，接收方接收到非有效的数据包时，再次发送上一个有效数据包的ACK即可，避免ACK丢失出现问题。

这次实验让我更深刻体会到了可靠传输的建立，亲手编程验证了书上的理论。