【数据报格式】

数据报由两部分组成，第一部分是head，即数据报头，第二部分是所携带的数据部分，长度上限为2048字节。在数据报头UDPpackethead中包含以下内容：

|  |
| --- |
| struct UDPpackethead  {  unsigned int seq;//序列号 16位  u\_short Check;//校验 16位  u\_short len;//数据部分总长度  unsigned char flags;//标志位  UDPpackethead()  {  len = 0;  seq = 0;  Check = 0;  flags = 0;  }  }; |
| struct UDPpacket  {  UDPpackethead head;  char data[2048];//每段data长度上限  }; |

【协议设计】

本次实验基于rdt3.0以及停等机制设计了协议，在正式传输数据时大致流程如下：

客户端向服务器发送seq=0的数据包：此时Cseq=0,SSeq=0

1. Seq=0数据包在传输过程中丢失，服务器一直收不到包，仍然处于等待状态（不发送ACK包），一直到客户端发现已经超时未收到ACK包，客户端重传seq=0的数据包。
2. 服务器收到数据包，假如校验和错误或者收到的包不是当前等待的包（SSeq！=seq），则应该返回Seq=SSeq（0）的ACK包。收到的数据包均正确，应该将SSeq+=1，传回Seq=SSeq（1）的ACK包。
3. 返回的Seq=0或者Seq=1的ACK包丢失，则客户端会发现超时未收到ACK包，因此重传Seq=0的数据包。
4. 客户端收到返回的Seq=0的ACK包，如果校验和正确，此时Seq==Cseq，因此重新传送Seq=0的数据包。如果校验和错误，仍然应该重新发送Seq=0的数据包。
5. 客户端收到返回的Seq=1的ACK包，如果校验和正确，此时Seq==Cseq+1，因此将Cseq+1，接下来传送Seq=Cseq=1的数据包。如果校验和错误，则重发Seq=0的数据包。
6. 重复以上循环即可
7. 当客户端发送到最后一个包时，将最后一个数据包的标志位设置为FINAL，服务器由头部标志位判断出其为最后一个包，因此将对此数据包的返回确认包头部设置为FINAL\_ACK
8. 客户端若收到确认包头部为ACK或者超时未收到返回确认包或者收到的返回确认包校验和错误则重发最后一个数据包，若收到完全正确的FINAL\_ACK确认包则代表文件已经全部正确传输，因此跳出循环。
9. 服务器发出最后一个确认包后等待2倍的CLOCKS\_PER\_SEC，倘若此时间内没有收到任何包，则说明客户端已经完成数据传输并且全部确认，服务器结束接收消息；如果收到了数据包，说明之前发出的确认包可能出了问题，重发确认包。

客户端发送文件信息函数：

|  |
| --- |
| int sendmessage(int messagelength, char\* message, int addrlength, SOCKET & client, SOCKADDR\_IN & ServerAddr)  {  //设置为非阻塞模式，避免卡在recvfrom  int iMode = 1; //0：阻塞  ioctlsocket(client, FIONBIO, (u\_long FAR\*) & iMode);//非阻塞设置  //计算要传多少个包  int packetnum = messagelength / 2048;  if (messagelength % 2048)//向上取整  {  packetnum += 1;  }  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "共有" << messagelength << "字节，因此共";  cout << "要传送" << packetnum << "个包" << endl;  int sequence = 0;//记录要发送的数据包的序列号  clock\_t starttime;  UDPpacket recvpacket;//来自服务器发回的数据包  UDPpacket sendpack;//发送包  char\* sendbuffer = new char[sizeof(sendpack)];  //接收缓冲区  char\* recvbuffer = new char[sizeof(recvpacket)];  while (1)  {  //当前要发送的是最后一个包了  if (sequence == packetnum - 1)  {  //cout << "wrong here maybe" << endl;  //1代表是最后一个包，从message + 1024 \* sequence开始传送  //要传送的长度为messagelength - 1024 \* sequence  sendpack.head.flags = FINAL;  sendpack.head.len = messagelength - 2048 \* sequence;//数据部分长度  sendpack.head.seq = sequence;//序列号  sendpack.head.Check = 0;  //消息复制 报错——内存太小  memcpy(&(sendpack.data), message + 2048 \* sequence, sendpack.head.len);  //设置校验和  sendpack.head.Check = packetcheck((u\_short\*)&sendpack, sizeof(sendpack));  //进行传输  memcpy(sendbuffer, &(sendpack), sizeof(sendpack));  int flag = sendto(client, sendbuffer, sizeof(sendpack), 0, (sockaddr\*)&ServerAddr, addrlength);  if (flag == -1)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "包seq" << sendpack.head.seq << "发送失败" << endl;  }  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  //cout << "客户端发送head为：check" << sendpack.head.Check << "len:" << sendpack.head.len << "flags:" << (u\_short)sendpack.head.flags << "seq:" << sendpack.head.seq << endl;  //cout << "包seq" << sendpack.head.seq << "发送成功" << endl;  cout << "成功发送包seq：" << sendpack.head.seq << " datalength：" << sendpack.head.len << " Check：" << sendpack.head.Check << " flags：" << (u\_short)sendpack.head.flags << endl;  }  //cout << "客户端传送最后一个包seq：" << sequence << endl;  //此时我需要收到来自服务器的FINAL\_ACK包表示结束传送  starttime = clock();//发送时间  UDPpacket recvpacket;//来自服务器发回的数据包  //接收缓冲区  char\* recvbuffer = new char[sizeof(recvpacket)];  while (recvfrom(client, recvbuffer, sizeof(recvpacket), 0, (sockaddr\*)&ServerAddr, &addrlength) <= 0)  {  if (clock() - starttime >= 2 \* CLOCKS\_PER\_SEC)  {  //超时重传  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————超时重传包——————" << sequence << endl;  //sendpacket(1, message + 2048 \* sequence, messagelength - 2048 \* sequence, sequence, addrlength, client, ServerAddr);  sendpack.head.flags = FINAL;  sendpack.head.len = messagelength - 2048 \* sequence;//数据部分长度  sendpack.head.seq = sequence;//序列号  sendpack.head.Check = 0;  //消息复制 报错——内存太小  memcpy(&(sendpack.data), message + 2048 \* sequence, sendpack.head.len);  //设置校验和  sendpack.head.Check = packetcheck((u\_short\*)&sendpack, sizeof(sendpack));  //进行传输  memcpy(sendbuffer, &(sendpack), sizeof(sendpack));  int flag = sendto(client, sendbuffer, sizeof(sendpack), 0, (sockaddr\*)&ServerAddr, addrlength);  if (flag == -1)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "包seq" << sendpack.head.seq << "发送失败" << endl;  }  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  //cout << "客户端发送head为：check" << sendpack.head.Check << "len:" << sendpack.head.len << "flags:" << (u\_short)sendpack.head.flags << "seq:" << sendpack.head.seq << endl;  //cout << "包seq" << sendpack.head.seq << "发送成功" << endl;  cout << "成功发送包seq：" << sendpack.head.seq << " datalength：" << sendpack.head.len << " Check：" << sendpack.head.Check << " flags：" << (u\_short)sendpack.head.flags << endl;  }    starttime = clock();  }  }  //收到了ACK包  //cout << "recvlength=" << recvlength << endl;  memcpy(&(recvpacket), recvbuffer, sizeof(recvpacket));  u\_short res = packetcheck((u\_short\*) &recvpacket, sizeof(recvpacket));  //cout << "客户端收到的包head为：check" << recvpacket.head.Check << "len:" << recvpacket.head.len << "flags:" << (u\_short)recvpacket.head.flags << "seq:" << recvpacket.head.seq << endl;  //cout << "checkres=" << res << endl;  if (recvpacket.head.flags == FINAL\_ACK && res == 0)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  //文件全部正确传送，结束  cout << "文件已经全部正确传送完毕" << endl;  //等待两个时钟周期以跟server同步  starttime = clock();  while (clock() - starttime <= 2 \* CLOCKS\_PER\_SEC)  {  }  return 1;  }  else  {  //说明最后一个包传送不成功  //那就需要重传最后一个包  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————差错重传包——————" << sequence << endl;  continue;  }  }  else//代表传的不是最后一个  {  //cout << "发的不是最后一个包" << endl;  //步骤：发送，获得ack，seq++  //0代表不是最后一个包，从message + 2048 \* sequence开始传送  //要传送的长度为2048  sendpack.head.flags = 0;  sendpack.head.len = 2048;//数据部分长度  sendpack.head.seq = sequence;//序列号  sendpack.head.Check = 0;  //消息复制 报错——内存太小  memcpy(&(sendpack.data), message + 2048 \* sequence, sendpack.head.len);  //设置校验和  sendpack.head.Check = packetcheck((u\_short\*)&sendpack, sizeof(sendpack));  //进行传输  memcpy(sendbuffer, &(sendpack), sizeof(sendpack));  int flag = sendto(client, sendbuffer, sizeof(sendpack), 0, (sockaddr\*)&ServerAddr, addrlength);  if (flag == -1)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "包seq" << sendpack.head.seq << "发送失败" << endl;  }  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  //cout << "客户端发送head为：check" << sendpack.head.Check << "len:" << sendpack.head.len << "flags:" << (u\_short)sendpack.head.flags << "seq:" << sendpack.head.seq << endl;  //cout << "包seq" << sendpack.head.seq << "发送成功" << endl;  cout << "成功发送包seq：" << sendpack.head.seq << " datalength：" << sendpack.head.len << " Check：" << sendpack.head.Check << " flags：" << (u\_short)sendpack.head.flags << endl;  }  starttime = clock();  //此时我需要收到来自服务器的ACK包表示上一个包传送成功  while (recvfrom(client, recvbuffer, sizeof(recvpacket), 0, (sockaddr\*)&ServerAddr, &addrlength) <= 0)  {  if (clock() - starttime > 2 \* CLOCKS\_PER\_SEC)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "————超时重传————" << endl;  //走到这里说明需要重发上一个包  sendpack.head.flags = 0;//不设置标志位  sendpack.head.len = 2048;//数据部分长度  sendpack.head.seq = sequence;//序列号  sendpack.head.Check = 0;  //消息复制 报错——内存太小  memcpy(&(sendpack.data), message + 2048 \* sequence, sendpack.head.len);  //设置校验和  sendpack.head.Check = packetcheck((u\_short\*)&sendpack, sizeof(sendpack));  //进行传输  memcpy(sendbuffer, &(sendpack), sizeof(sendpack));  int flag = sendto(client, sendbuffer, sizeof(sendpack), 0, (sockaddr\*)&ServerAddr, addrlength);  if (flag == -1)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "包seq" << sendpack.head.seq << "发送失败" << endl;  }  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  //cout << "客户端发送head为：check" << sendpack.head.Check << "len:" << sendpack.head.len << "flags:" << (u\_short)sendpack.head.flags << "seq:" << sendpack.head.seq << endl;  //cout << "包seq" << sendpack.head.seq << "发送成功" << endl;  cout << "成功发送包seq：" << sendpack.head.seq << " datalength：" << sendpack.head.len << " Check：" << sendpack.head.Check << " flags：" << (u\_short)sendpack.head.flags << endl;  }  starttime = clock();  }  }  //走到这里说明收到包  memcpy(&recvpacket, recvbuffer, sizeof(recvpacket));  //上一个包传送成功  u\_short res = packetcheck((u\_short\*)&recvpacket, sizeof(recvpacket));  if (recvpacket.head.flags == ACK && recvpacket.head.seq == sequence + 1 && res == 0)  {  /\*  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "包seq" << sequence << "正确传递接收" << endl;  \*/  sequence += 1;  continue;//继续这个循环，传下一个  }  else if (recvpacket.head.flags == ACK && recvpacket.head.seq == sequence && res == 0)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "客户端重传包seq" << sequence << endl;  //这种情况说明上一个包出问题了，那就重新再传这个。  continue;  }    }  }  } |

服务器接收消息函数：

|  |
| --- |
| //接收消息  u\_long ReceiveMessage(char\* message, int& addrlength, SOCKET& server, SOCKADDR\_IN& ClientAddr)  {  long int filelength = 0;//记录文件长度  UDPpacket recvpacket, sendpacket;  //接收缓冲区  char \*recvbuffer = new char[sizeof(recvpacket)];  //发送缓冲区  char\* sendbuffer = new char[sizeof(sendpacket)];  unsigned int Ssequence = 0;//所期望得到的下一个数据包的编号  while (1)  {  //接收报文长度  int recvlength = recvfrom(server, recvbuffer, sizeof(recvpacket), 0, (sockaddr\*)&ClientAddr, &addrlength);  //cout << "recvlength=" << recvlength << endl;  memcpy(&recvpacket, recvbuffer, sizeof(recvpacket));  u\_short res = packetcheck((u\_short\*)&recvpacket, sizeof(recvpacket));  //u\_short res = packetcheck((u\_short\*)&recvpacket.head, sizeof(recvpacket.head));  //cout << "服务器收到head为：check" << recvpacket.head.Check << "len:" << recvpacket.head.len << "flags:" << (u\_short)recvpacket.head.flags << "seq:" << recvpacket.head.seq << endl;  //cout << "校验结果为：" << res << endl;  if (res != 0)//出错了  {    cout << "信息传输校验码错误！" << endl;  sendpacket.head.flags = ACK;  //此时传回的sequence仍然为上一次传送的值  //表示客户端之前传来的包错误  sendpacket.head.seq = Ssequence;  //计算校验和  sendpacket.head.Check = packetcheck((u\_short\*)&sendpacket, sizeof(sendpacket));  memcpy(sendbuffer, &sendpacket, sizeof(sendpacket));  //期待重传  int flag = sendto(server, sendbuffer, sizeof(sendpacket), 0, (sockaddr\*)&ClientAddr, addrlength);  if (flag == -1)  {  cout << "发送失败" << endl;  }  continue;  }  else//至少校验码是对的  {  //判断一下这是不是最后一个包  if (recvpacket.head.flags == FINAL)  {  //放进消息里面  memcpy(message + filelength, recvpacket.data, recvpacket.head.len);  filelength += recvpacket.head.len;  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "————最后一个包seq:"<<recvpacket.head.seq<<"已经接收完毕————" << endl;  sendpacket.head.flags = FINAL\_ACK;  sendpacket.head.Check = 0;  sendpacket.head.Check = packetcheck((u\_short\*)&sendpacket, sizeof(sendpacket));  //cout << "服务器发送的确认包head为：check" << sendpacket.head.Check << "len:" << sendpacket.head.len << "flags:" << (u\_short)sendpacket.head.flags << "seq:" << sendpacket.head.seq << endl;  memcpy(sendbuffer, &sendpacket, sizeof(sendpacket));  int flag = sendto(server, sendbuffer, sizeof(sendpacket), 0, (sockaddr\*)&ClientAddr, addrlength);  clock\_t starttime = clock();//最后一个ACK的发送时间  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "发送最后一个确认包seq:" << sendpacket.head.seq << " datalength:" << sendpacket.head.len << " Check:" << sendpacket.head.Check << " flags:" << (u\_short)sendpacket.head.flags << endl;  //为了防止最后一个包的ACK发送丢失，需要等2MSL  //设置为非阻塞模式，避免卡在recvfrom  int iMode = 1; //0：阻塞  ioctlsocket(server, FIONBIO, (u\_long FAR\*) & iMode);//非阻塞设置  while (clock() - starttime <= 2 \* CLOCKS\_PER\_SEC)  {  if (recvfrom(server, recvbuffer, sizeof(recvpacket), 0, (sockaddr\*)&ClientAddr, &addrlength) <= 0)  {  //一直没有收到包，说明对面正确接收了ACK  continue;  }  //否则需要重传最后一个包的ACK  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "重传最后一个确认包seq:" << sendpacket.head.seq << " datalength:" << sendpacket.head.len << " Check:" << sendpacket.head.Check << " flags:" << (u\_short)sendpacket.head.flags << endl;  sendto(server, sendbuffer, sizeof(sendpacket), 0, (sockaddr\*)&ClientAddr, addrlength);  starttime = clock();  }  }  //恢复阻塞模式  iMode = 0; //0：阻塞  ioctlsocket(server, FIONBIO, (u\_long FAR\*) & iMode);//非阻塞设置  return filelength;  }  else//后面还有别的包  {  //如果收到的包不是想要的包的话  if (Ssequence != recvpacket.head.seq)  {  //说明出了问题，返回ACK  sendpacket.head.flags = ACK;  sendpacket.head.len = 0;  //此时仍然应该返回之前的seq  sendpacket.head.seq =Ssequence;  //计算校验和  sendpacket.head.Check = packetcheck((u\_short\*)&sendpacket, sizeof(sendpacket));  memcpy(sendbuffer, &sendpacket, sizeof(sendpacket));  //重发该包的ACK  int flag = sendto(server, sendbuffer, sizeof(sendpacket), 0, (sockaddr\*)&ClientAddr, addrlength);  if (flag == -1)  {  //int i=WSAGetLastError();  //cout << "wrong code:" << i << endl;  //错误代码10038  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout <<"——————server发送序列号为：" << (int)sendpacket.head.seq<< "的ACK失败————" << endl;  continue;//丢弃该数据包  }  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————server发送序列号为：" << (int)sendpacket.head.seq << "的ACK成功————" << endl;  continue;  }    }  else//收到的数据包是此时服务器想要的  {  //放进消息里面  int cur\_length = recvpacket.head.len;  memcpy(message+filelength, recvpacket.data, cur\_length);  filelength += int(recvpacket.head.len);  //接下来发送ACK  Ssequence += 1;  sendpacket.head.flags = ACK;  sendpacket.head.seq = Ssequence;  sendpacket.head.Check = packetcheck((u\_short\*)&sendpacket, sizeof(sendpacket));  memcpy(sendbuffer, &sendpacket, sizeof(sendpacket));  int flag = sendto(server, sendbuffer, sizeof(sendpacket), 0, (SOCKADDR \*) &ClientAddr, addrlength);  //为什么第三个发送失败？——buffer太小了……  if (flag == -1)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "正确接收包：" << recvpacket.head.seq << endl;  cout << "——————ACK发送失败————" << endl;  }  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "正确接收包：" << recvpacket.head.seq << endl;  cout << "发送确认包seq:" << sendpacket.head.seq << " datalength:" << sendpacket.head.len << " Check:" << sendpacket.head.Check << " flags:" << (u\_short)sendpacket.head.flags << endl;  }    }  }  }  }  //  } |

【建立连接】

此处协议设计模拟了TCP的三次握手协议。

第一次握手：客户端向服务器发送头部标志位为SYN的数据包，表示想与服务器建立连接：

1. 此数据包校验和错误/中途丢失，服务器接收后直接丢弃，继续处于等待接收状态，不发送数据包——客户端在经过一段时间后收不到返回数据包，因此重发标志位为SYN的数据包。
2. 此数据包被服务器正确接收，进入第二次握手。

第二次握手：服务器向客户端发送头部标志位为SYN\_ACK的数据包

1. 此数据包校验和错误/中途丢失，客户端接收后直接丢弃，继续处于等待接收状态，不发送数据包——服务器在经过一段时间后收不到返回数据包，因此重发标志位为SYN\_ACK的数据包。
2. 此数据包被客户端正确接收，进入第三次握手。

第三次握手：客户端向服务器发送头部标志位为ACK的数据包

1. 此数据包校验和错误/中途丢失，服务器接收后直接丢弃，继续处于等待接收状态，不发送数据包——客户端在经过一段时间后收不到返回数据包，因此重发标志位为ACK的数据包。
2. 此数据包被服务器正确接收，握手完成。

客户端建立连接函数：

|  |
| --- |
| int connect(SOCKET& client, SOCKADDR\_IN& ClientAddr, int &len)  {  //设置为非阻塞模式，避免卡在recvfrom  int iMode = 1; //0：阻塞  ioctlsocket(client, FIONBIO, (u\_long FAR\*) & iMode);//非阻塞设置  //第一次握手，客户端发送SYN  UDPpacket pack1;  pack1.head.flags = SYN;  //pack1.head.Check = packetcheck((u\_short\*)&(pack1.head), sizeof(pack1.head));  pack1.head.Check = packetcheck((u\_short\*)&(pack1), sizeof(pack1));  //cout << "客户端head为：check" << pack1.head.Check << "len:" << pack1.head.len << "flags:" << (u\_short)pack1.head.flags << "seq:" << pack1.head.seq << endl;  //发送缓冲区  char\* buffer1 = new char[sizeof(pack1)];  memcpy(buffer1, &pack1, sizeof(pack1));  //s：已建好连接的socket,如果利用UDP协议则不需经过connect操作。  //msg：待发送数据的缓冲区  //len：缓冲区长度  //flags：调用方式标志位, 一般为0, 改变flags，将会改变sendto发送的形式。  //addr：（可选）指针，指向目的套接字的地址  //tolen：addr所指地址的长度  int flag = sendto(client, buffer1, sizeof(pack1), 0, (sockaddr\*)&ClientAddr, len);  //第一次握手发送时间  clock\_t pack1starttime = clock();  if (flag == -1)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第一次握手发送失败————" << endl;  }  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第一次握手发送成功————" << endl;  }  //第二次握手，等待服务器发来的SYN\_ACK  UDPpacket pack2;  //缓冲区  char\* buffer2 = new char[sizeof(pack2)];  //一直没有收到包，说明之前的包出问题了，要重新发送第一次握手  while (1)  {  //如果超时则重传  if (clock()-pack1starttime>= 2 \* CLOCKS\_PER\_SEC)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "————第一次握手包超时重传————" << endl;  sendto(client, buffer1, sizeof(pack1), 0, (sockaddr\*)&ClientAddr, len);  pack1starttime = clock();  }  flag = recvfrom(client, buffer2, sizeof(pack2), 0, (sockaddr\*)&ClientAddr, &len);  if (flag <= 0)//没收到包的话就继续recvfrom  {  continue;  }  //收到了包  memcpy(&pack2, buffer2, sizeof(pack2));  int res = packetcheck((u\_short\*)&pack2, sizeof(pack2));  //是第二次握手  if (pack2.head.flags == SYN\_ACK && res == 0)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第二次握手接收成功————" << endl;  break;  }  //否则一直处于等待状态  }  //第三次握手 客户端发送ACK  UDPpacket pack3;  pack3.head.flags = ACK;  pack3.head.Check = packetcheck((u\_short\*)&pack3, sizeof(pack3));  //发送缓冲区  char\* buffer3 = new char[sizeof(pack3)];  memcpy(buffer3, &pack3, sizeof(pack3));  //s：已建好连接的socket,如果利用UDP协议则不需经过connect操作。  //msg：待发送数据的缓冲区  //len：缓冲区长度  //flags：调用方式标志位, 一般为0, 改变flags，将会改变sendto发送的形式。  //addr：（可选）指针，指向目的套接字的地址  //tolen：addr所指地址的长度  flag = sendto(client, buffer3, sizeof(pack3), 0, (sockaddr\*)&ClientAddr, len);  if (flag == -1)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第三次握手发送失败————" << endl;  return 0;  }  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第三次握手发送成功————" << endl;  }  clock\_t checktime = clock();//记录发送第三个包的时间  while ((clock() - checktime) <= 2\* CLOCKS\_PER\_SEC)  {  //又把第二个包传过来了  //否则对面应该就结束传包  if (recvfrom(client, buffer2, sizeof(pack2), 0, (sockaddr\*)&ClientAddr, &len) <= 0)  {  continue;  }  //如果收到了包，那么就说明第三个包对面没收到，重传  sendto(client, buffer3, sizeof(pack3), 0, (sockaddr\*)&ClientAddr, len);  checktime = clock();  }  iMode = 0; //0：阻塞  ioctlsocket(client, FIONBIO, (u\_long FAR\*) & iMode);//恢复成阻塞模式  //cout << "客户端head为：check" << pack3.head.Check << "len:" << pack3.head.len << "flags:" << (u\_short)pack3.head.flags << "seq:" << pack3.head.seq << endl;  return 1;  } |

服务器端建立连接函数：

|  |
| --- |
| int connect(SOCKET& server, SOCKADDR\_IN& ServerAddr, int& len)  {  //设置为非阻塞模式，避免卡在recvfrom  int iMode = 1; //0：阻塞  ioctlsocket(server, FIONBIO, (u\_long FAR\*) & iMode);//非阻塞设置  UDPpacket pack1;  //缓冲区  char\* buffer1 = new char[sizeof(pack1)];  //接收第一次握手请求信息  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "进入连接阶段" << endl;  int flag;  while (1)  {  //没收到包就一直循环  flag = recvfrom(server, buffer1, sizeof(pack1), 0, (sockaddr\*)&ServerAddr, &len);  if (flag <= 0)  {  continue;  }  //接收成功  memcpy(&(pack1), buffer1, sizeof(pack1.head));  //cout << "服务器收到head为：check" << pack1.head.Check << "len:" << pack1.head.len << "flags:" << (u\_short)pack1.head.flags << "seq:" << pack1.head.seq << endl;  u\_short res = packetcheck((u\_short\*)&pack1,sizeof(pack1));  //只有当收到的数据包正确时才算是成功  if (pack1.head.flags == SYN && res == 0)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第一次握手接收成功————" << endl;  break;  }  //如果发来的包不正确的话，继续这个循环，等待客户端重新发送  }  //第二次握手，服务器发送SYN\_ACK  UDPpacket pack2;  pack2.head.flags = SYN\_ACK;  pack2.head.Check = packetcheck((u\_short\*)&pack2, sizeof(pack2));  //发送缓冲区  char\* buffer2 = new char[sizeof(pack2)];  memcpy(buffer2, &pack2, sizeof(pack2));  //s：已建好连接的socket,如果利用UDP协议则不需经过connect操作。  //msg：待发送数据的缓冲区  //len：缓冲区长度  //flags：调用方式标志位, 一般为0, 改变flags，将会改变sendto发送的形式。  //addr：（可选）指针，指向目的套接字的地址  //tolen：addr所指地址的长度  flag = sendto(server, buffer2, sizeof(pack2), 0, (sockaddr\*)&ServerAddr,len);  //超时重传机制  clock\_t starttime = clock();//第二次握手发送时间  if (flag == -1)//第二次握手发送失败，重传  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第二次握手发送失败————" << endl;  }  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第二次握手发送成功————" << endl;  }  //第三次握手 服务器接收  UDPpacket pack3;  //缓冲区  char\* buffer3 = new char[sizeof(pack3)];  //如果一直没有收到包的话，说明上一个包需要重传  while (1)  {  //超时重传  if (clock\_t() - starttime >= CLOCKS\_PER\_SEC)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第二次握手包正在重传————" << endl;  sendto(server, buffer2, sizeof(pack2), 0, (sockaddr\*)&ServerAddr, len);  starttime = clock();//更新发送时间  }  //没收到包就一直循环  flag = recvfrom(server, buffer3, sizeof(pack3), 0, (sockaddr\*)&ServerAddr, &len);  if (flag <= 0)  {  continue;  }  //第三次包接收成功  memcpy(&(pack3), buffer3, sizeof(pack3));  //cout << "服务器收到head为：check" << pack3.head.Check << "len:" << pack3.head.len << "flags:" << (u\_short)pack3.head.flags << "seq:" << pack3.head.seq << endl;  u\_short res = packetcheck((u\_short\*)&pack3, sizeof(pack3));  if (pack3.head.flags == ACK && res == 0)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第三次握手接收成功————" << endl;  break;//跳出循环  }  //否则的话就一直等到能接收正确的包  }  //等两个时钟周期  starttime = clock();  while ((clock() - starttime) <= 2 \* CLOCKS\_PER\_SEC)  {  }  iMode = 0; //0：阻塞  ioctlsocket(server, FIONBIO, (u\_long FAR\*) & iMode);//恢复成阻塞模式  return 1;  } |

【断开连接】

此处协议设计模拟了TCP的四次挥手协议。

第一次挥手：客户端向服务器发送头部标志位为FIN的数据包，如果超时未收到来自服务器发送的FIN\_ACK包，则重发第一次挥手包FIN。如果收到了来自服务器发送的正确的FIN\_ACK包，则进入第三次握手。

第二次挥手：服务器收到客户端发来的第一次挥手包后，确认无误后发送第二次挥手包，其标志位为FIN\_ACK。等待两个CLOCKS\_PER\_SEC，如果在此期间没有收到客户端发来的包的话则说明客户端正确接收，第二次挥手结束。如果收到了来自客户端的包则需要重传第二次挥手。

第三、四次挥手机制与第一、二次相同，只是FIN发送端更改为服务器，FIN\_ACK为客户端。自此正式断开连接。

客户端断开连接函数：

|  |
| --- |
| int disconnect(SOCKET& client, SOCKADDR\_IN& ClientAddr, int len)  {  //设置为非阻塞模式，避免卡在recvfrom  int iMode = 1; //0：阻塞  ioctlsocket(client, FIONBIO, (u\_long FAR\*) & iMode);//非阻塞设置  //第一次挥手，客户端发送FIN  UDPpacket pack1;  pack1.head.flags = FIN;  pack1.head.Check = packetcheck((u\_short\*)&pack1, sizeof(pack1));  //发送缓冲区  char\* buffer1 = new char[sizeof(pack1)];  memcpy(buffer1, &pack1, sizeof(pack1));  int flag = sendto(client, buffer1, sizeof(pack1), 0, (sockaddr\*)&ClientAddr, len);  if (flag == -1)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第一次挥手发送失败————" << endl;  }  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第一次挥手发送成功————" << endl;  }  //记录第一次挥手发送时间  clock\_t starttime = clock();  //第二次挥手，接收服务器发来的FIN\_ACK  UDPpacket pack2;  //缓冲区  char\* buffer2 = new char[sizeof(pack2)];  //接收第二次挥手请求信息  while (1)  {  if (clock() - starttime >= 2 \* CLOCKS\_PER\_SEC)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  //超时了，重传  cout << "超时重发第一次挥手" << endl;  sendto(client, buffer1, sizeof(pack1), 0, (sockaddr\*)&ClientAddr, len);  starttime = clock();  }  flag = recvfrom(client, buffer2, sizeof(pack2), 0, (sockaddr\*)&ClientAddr, &len);  if (flag <= 0 )  {  continue;  }  //接收成功  memcpy(&(pack2), buffer2, sizeof(pack2));  u\_short res = packetcheck((u\_short\*)&pack2, sizeof(pack2));  //是第二次挥手  if (pack2.head.flags == FIN\_ACK && res == 0)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第二次挥手接收成功————" << endl;  break;  }  //传来的包不正确则一直等待  }  starttime = clock();  //等待两个时钟周期，跟server同步  while (clock() - starttime <= CLOCKS\_PER\_SEC)  {  }  //接收第三次挥手信息，头部为ACK  UDPpacket pack3;  //缓冲区  char\* buffer3 = new char[sizeof(pack3)];  //接收第三次挥手请求信息  while (1)  {  flag = recvfrom(client, buffer3, sizeof(pack3), 0, (sockaddr\*)&ClientAddr, &len);  if (flag <= 0)  {  continue;  }  //接收成功  memcpy(&(pack3), buffer3, sizeof(pack3));  u\_short res = packetcheck((u\_short\*)&pack3, sizeof(pack3));  //是第三次挥手  if (pack3.head.flags == FIN && res == 0)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第三次挥手接收成功————" << endl;  break;  }  //否则一直处于等待状态  }  //第四次挥手，客户端发送FIN\_ACK  UDPpacket pack4;  pack4.head.flags = FIN\_ACK;  pack4.head.Check = packetcheck((u\_short\*)&pack4, sizeof(pack4));  //发送缓冲区  char\* buffer4 = new char[sizeof(pack4)];  memcpy(buffer4, &pack4, sizeof(pack4));  flag = sendto(client, buffer4, sizeof(pack4), 0, (sockaddr\*)&ClientAddr, len);  if (flag == -1)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第四次挥手发送失败————" << endl;  }  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第四次挥手发送成功————" << endl;  }  starttime = clock();//第四次挥手发送时间  while (clock() - starttime <= 2 \* CLOCKS\_PER\_SEC)  {  //没收到  if (recvfrom(client, buffer3, sizeof(pack1), 0, (sockaddr\*)&ClientAddr, &len) <= 0)  {  continue;  }  else//收到了，说明第四次挥手没发出去，需要重新发送  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "超时重发第四次挥手" << endl;  sendto(client, buffer4, sizeof(pack4), 0, (sockaddr\*)&ClientAddr, len);  //更新发送时间  starttime = clock();  }  }  return 1;  } |

服务器端断开连接函数：

|  |
| --- |
| int disconnect(SOCKET& server, SOCKADDR\_IN& ServerAddr, int& len)  {  //设置为非阻塞模式，避免卡在recvfrom  int iMode = 1; //0：阻塞  ioctlsocket(server, FIONBIO, (u\_long FAR\*) & iMode);//非阻塞设置  UDPpacket pack1;  //缓冲区  char\* buffer1 = new char[sizeof(pack1)];  int flag;  //接收第一次挥手请求信息  while (1)  {  flag = recvfrom(server, buffer1, sizeof(pack1), 0, (sockaddr\*)&ServerAddr, &len);  //没有收到消息则一直循环  if ( flag <= 0)  {  continue;  }  //接收成功  memcpy(&(pack1), buffer1, sizeof(pack1));  u\_short res = packetcheck((u\_short\*)&pack1, sizeof(pack1));  //是第一次握手  if (pack1.head.flags == FIN && res == 0)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第一次挥手接收成功————" << endl;  break;  }  //否则就一直等传送来正确的包  }  //第二次挥手，服务器发送FIN\_ACK  UDPpacket pack2;  pack2.head.flags = FIN\_ACK;  pack2.head.Check = packetcheck((u\_short\*)&pack2, sizeof(pack2));  //发送缓冲区  char\* buffer2 = new char[sizeof(pack2)];  memcpy(buffer2, &pack2, sizeof(pack2));  flag = sendto(server, buffer2, sizeof(pack2), 0, (sockaddr\*)&ServerAddr, len);  if (flag == -1)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第二次挥手发送失败————" << endl;  }  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第二次挥手发送成功————" << endl;  }  clock\_t starttime = clock();//记录第二次挥手发送时间  while (clock() - starttime <= 2 \* CLOCKS\_PER\_SEC)  {  //没收到  flag = recvfrom(server, buffer1, sizeof(pack1), 0, (sockaddr\*)&ServerAddr, &len);  if (flag <= 0)  {  continue;  }  else//收到了，说明第二次挥手没发出去，需要重新发送  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "超时重发第二次挥手" << endl;  sendto(server, buffer2, sizeof(pack2), 0, (sockaddr\*)&ServerAddr, len);  //更新发送时间  starttime = clock();  }  }  //第三次挥手，服务器要发送FIN  UDPpacket pack3;  pack3.head.flags = FIN;  pack3.head.Check = packetcheck((u\_short\*)&pack3, sizeof(pack3));  //发送缓冲区  char\* buffer3 = new char[sizeof(pack3)];  memcpy(buffer3, &pack3, sizeof(pack3));  flag = sendto(server, buffer3, sizeof(pack3), 0, (sockaddr\*)&ServerAddr, len);  if (flag == -1)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第三次挥手发送失败————" << endl;  }  else  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第三次挥手发送成功————" << endl;  }  starttime = clock();//第三次挥手发送时间  //接收第四次挥手请求信息  UDPpacket pack4;  //缓冲区  char\* buffer4 = new char[sizeof(pack4)];  while (1)  {  if (clock() - starttime >= 2 \* CLOCKS\_PER\_SEC)  {  starttime = clock();  //重新发送第三次挥手  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "超时重发第三次挥手" << endl;  sendto(server, buffer3, sizeof(pack3), 0, (sockaddr\*)&ServerAddr, len);  }  flag = recvfrom(server, buffer4, sizeof(pack4), 0, (sockaddr\*)&ServerAddr, &len);  if (flag <= 0)  {  continue;  }  //接收成功  memcpy(&(pack4), buffer4, sizeof(pack4));  u\_short res = packetcheck((u\_short\*)&pack4, sizeof(pack4));  //是第四次挥手  if (pack4.head.flags == FIN\_ACK && res == 0)  {  now = time(nullptr);  curr\_time = ctime(&now);  cout << curr\_time << "[System]:";  cout << "——————第四次挥手接收成功————" << endl;  break;  }  //否则的话一直循环等待  }  return 1;  } |

【差错检测】

客户端在发送数据包时，先设置该数据包头部中的序列号seq、数据长度len、标志位flags，将校验和check置0，之后将数据部分用文件信息填充。之后调用packetcheck函数，其作用为将整个数据包按16位为一组进行划分，不足16位的添加0补齐，之后进行直接相加的运算（UDP发送端真正校验和计算中是按照反码计算，但是由于数据本身就以反码形式计算，因此代码中直接相加），如果相加出现进位，则舍弃进位，所得结果+1，最终返回16位计算结果的反码。

服务器在接收数据包时可以直接调用packetcheck函数计算校验结果是否为0，如果为0则说明数据包无误（在UDP接收端中是将所有的数据以及包头部按照16位反码相加，其无误的结果应该为1111 1111 1111 1111，但是由于packetcheck最终返回值取反，因此调用此函数后无误结果应为0）。具体packetcheck函数实现如下：

|  |
| --- |
| u\_short packetcheck(u\_short\* packet, int packelength)  {  //UDP检验和的计算方法是：  //按每16位求和得出一个32位的数；  //如果这个32位的数，高16位不为0，则高16位加低16位再得到一个32位的数；  //重复第2步直到高16位为0，将低16位取反，得到校验和。  //register关键字命令编译器尽可能的将变量存在CPU内部寄存器中  //而不是通过内存寻址访问以提高效率。  //u\_long32位，不能使用unsigned int 只有16位  register u\_long sum = 0;  int count = (packelength + 1) / 2;//两个字节的计算  u\_short\* buf = (u\_short\*)malloc(packelength + 1);  memset(buf, 0, packelength + 1);  memcpy(buf, packet, packelength);  //cout << "count=" << packelength << endl;  while (count--)  {  sum += \*buf++;  if (sum & 0xFFFF0000)  {  sum &= 0xFFFF;  sum++;  }  }  return ~(sum & 0xFFFF);  } |

【确认重传】

此处重传分为两种情况：超时重传和差错重传。

对于超时重传，设定时间上限为2CLOCKS\_PER\_SEC，如果超过这段时间时客户端仍然未收到服务器发来的ACK包，说明之前发送的数据包或者服务器传来的确认包可能丢失了，因此处理办法均为客户端重发上一个数据包。

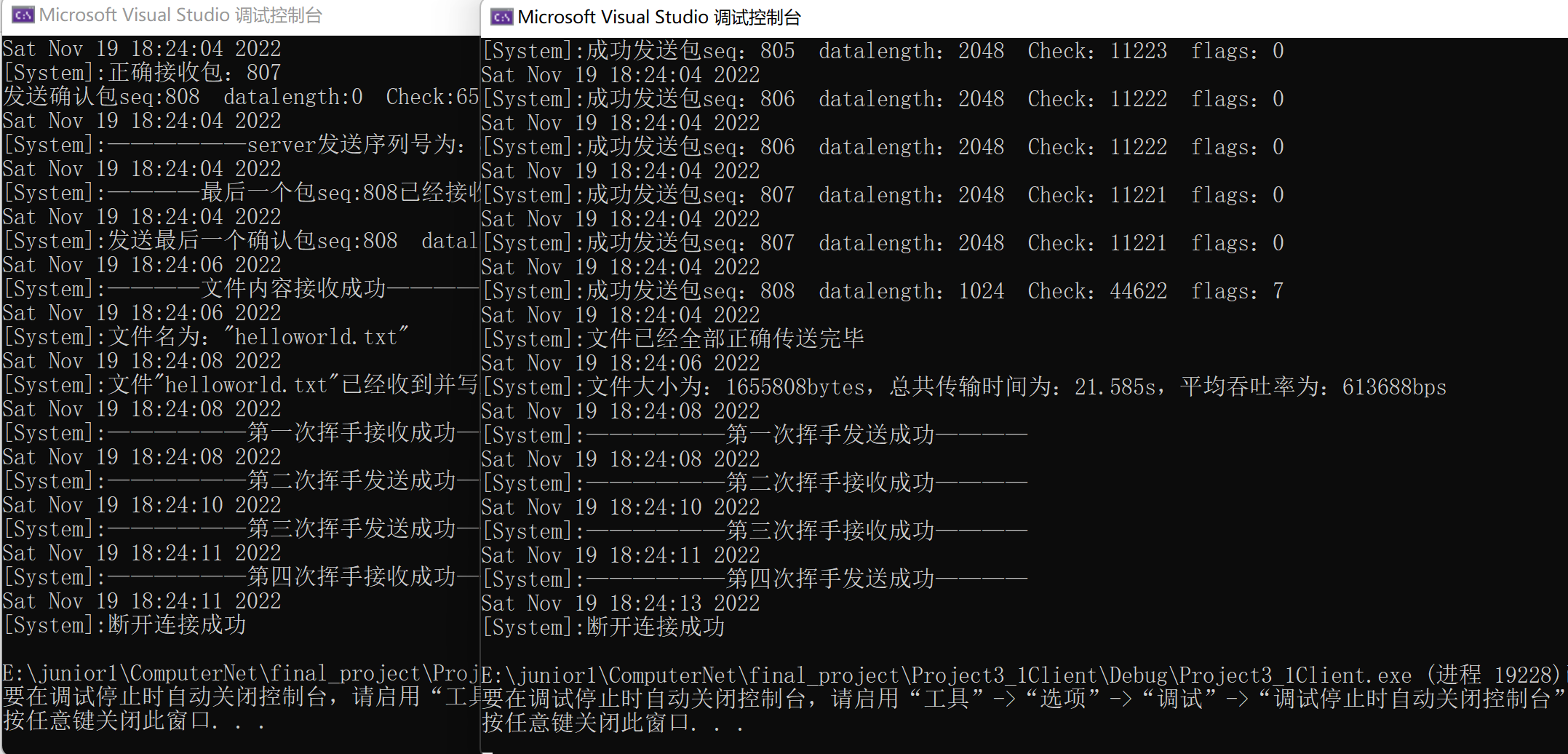
而对于差错重传，服务器接收到的数据包若不正确（check！=0）则返回上一个数据包的确认包，客户端接收到后重发上一个数据包；客户端接收到的确认包若不正确（check！=0）则什么也不做，直到超时重传上一个数据包。

【流量控制】

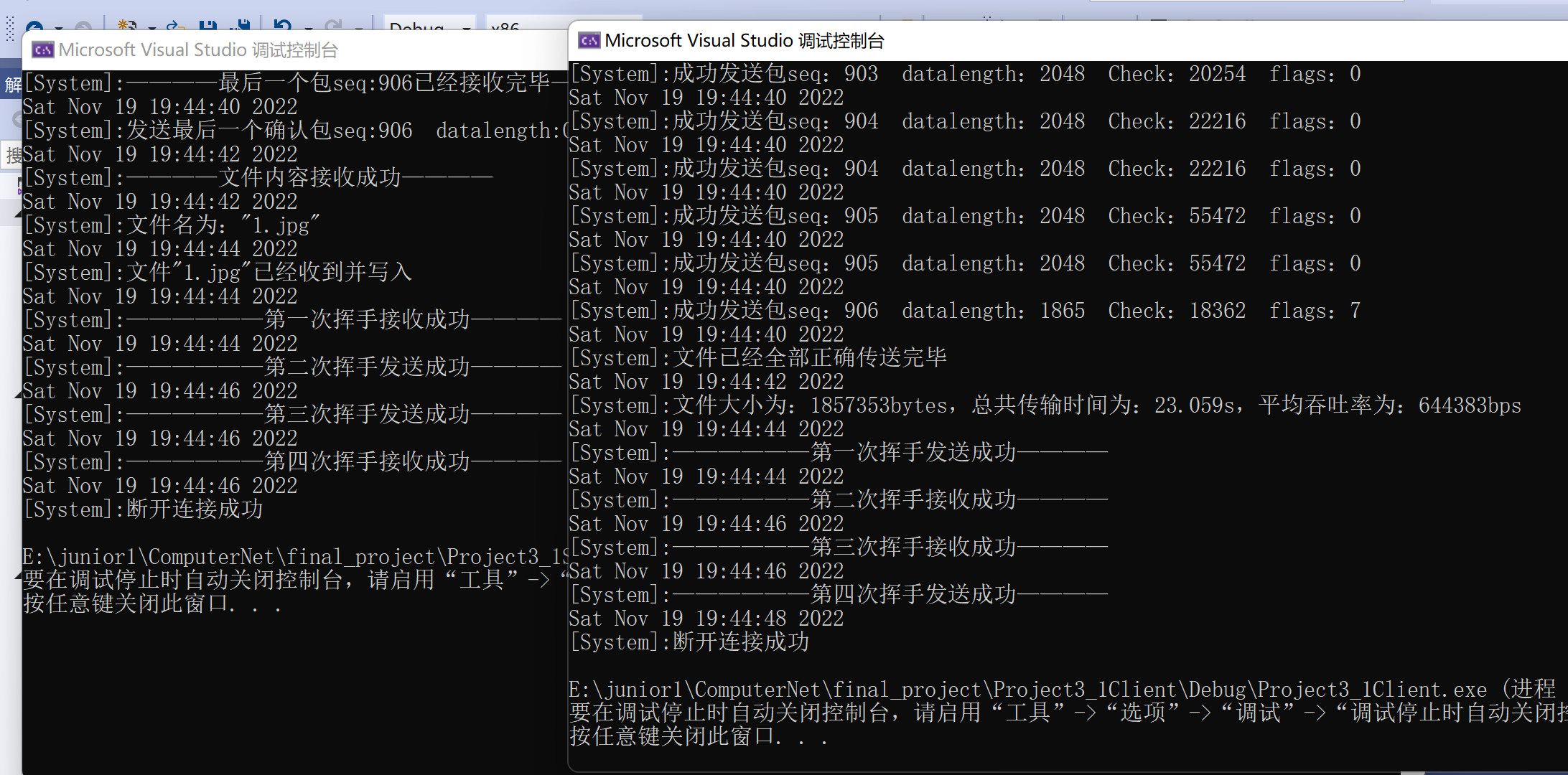
使用停等机制，具体实现方式见【协议设计】

【性能测试】

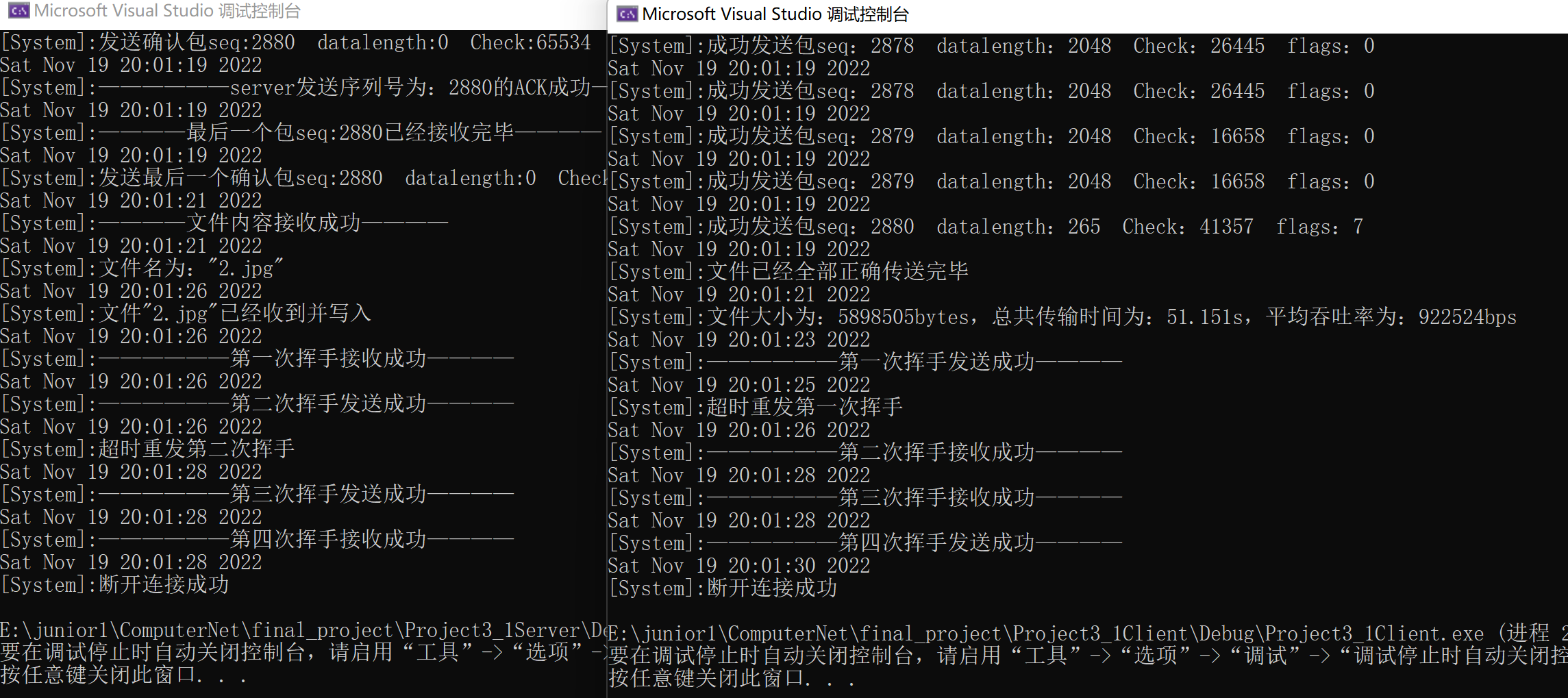
Helloworld.txt文件传输情况为：



1.jpg：



2.jpg:



3.jpg

