**--**

**计算机网络**

**课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 可靠数据传输协议-GBN 协议的设计与实现 | | | | | |
| 姓名 | 袁野 | | 院系 | 计算学部计算机科学与技术专业 | | |
| 班级 | 1903102 | | 学号 | 1190200122 | | |
| 任课教师 | 刘亚维 | | 指导教师 | 刘亚维 | | |
| 实验地点 | 格物207 | | 实验时间 | 2021.11.7 | | |
| 实验课表现 | 出勤、表现得分(10) |  | 实验报告  得分(40) |  | 实验总分 |  |
| 操作结果得分(50) |  |
| 教师评语 | | | | | | |
|  | | | | | | |

****

|  |
| --- |
| 实验目的： |
| 理解可靠数据传输的基本原理；掌握停等协议的工作原理；掌握基于 UDP 设计并实现一个停等协议的过程与技术。  理解滑动窗口协议的基本原理；掌握 GBN 的工作原理；掌握基于 UDP 设计并实现一个 GBN 协议的过程与技术。 |
| 实验内容： |
| 1) 基于 UDP 设计一个简单的停等协议，实现单向可靠数据传输（服务器到客户的数据传输）。  2) 模拟引入数据包的丢失，验证所设计协议的有效性。  3) 改进所设计的停等协议，支持双向数据传输；（选作内容，加分项目，可以当堂完成或课下完成）  4）基于所设计的停等协议，实现一个 C/S 结构的文件传输应用。（选作内容，加分项目，可以当堂完成或课下完成）  1) 基于 UDP 设计一个简单的 GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）。  2) 模拟引入数据包的丢失，验证所设计协议的有效性。  3) 改进所设计的 GBN 协议，支持双向数据传输；（选作内容，加分项目，可以当堂完成或课下完成）  4）将所设计的 GBN 协议改进为 SR 协议。（选作内容，加分项目，可以当堂完成或课下完成） |
| 实验过程： |
| 1. **基于UDP的GBN协议**      * 1. **GBN（SR）协议数据分组格式**     1. Seq为数据包的序号，大小为一个字节，取值为0~255。  2. Data为数据包内容，大小不超过1024字节。  3. 最后一个字符为‘\0’，表示数据包的结尾。  4. 缓冲区大小共1026字节。   * 1. **确认分组格式与各个域作用**     该分组包含两个字节，第一个字节表示序列号，第二个字节为‘\0’。   * 1. **协议两端程序流程图**   以自动机形式给出：    服务器端（发送端）    客户端（接收端）     * 1. **协议典型交互过程**      1. **GBN协议**      * + 1. **SR协议**      * 1. **数据分组丢失验证模拟方法**   丢失模拟方式在客户端实现，如果数据包丢失，则客户端认为没有接受到；如果ACK丢失，则客户端不向服务器发送ACK数据包。  在lossInLossRatio函数，将填写的X或者Y丢失率作为参数，在该函数中在0-100中随机生成一个数，如果该数小于丢失率\*100则认为丢失。  在服务器端有测试文件serverfile.txt,将其与客户端的接收文件进行比较，如果相同则认为其可靠。   1. **程序实现的主要类（或函数）及其主要作用**    1. **服务器端GBN\_Server**       1. **ackHandler(char c)**     处理收到的ACK确认信息，并修改窗口内的信息状态。   * + 1. **getCurTime(char \*ptime)**     获取当前时间并将其储存在ptime中。   * + 1. **lossInLossRatio()**     等概率随机生成一个[0,99]的数字，并判断其是否在[0,100\*LossRatio-1]以内。   * + 1. **seqIsAvailable()**     判断当前序列号是否可用。   * + 1. **timeoutHandler()**     将还未收到ACK确认信号且超时的数据包重新设置为未发送。   * 1. **客户端GBN\_Client**      1. **lossInLossRatio()**   与服务器中的同名函数同理。   1. **支持双向数据传输**   将服务器端文件内支持发动数据包的相关代码复制到客户端代码中，再将客户端代码中支持接收数据包的代码复制到服务器端即可。   1. **SR协议**    1. **客户端修改**   **增加函数seqRecvAvailable()**    判断接收的包是否在接收窗口内，如果在则接收，否则丢弃。   * 1. **服务器端修改**   改进累计确认机制，允许收到乱序的ack，超时后只重传已发送但未收到ack的分组。  如果当前接收的ack序号为窗口第一个序号，则右移窗口直到窗口第一个序号为为接收的ack序号，否则直接打上标记为已接受的ack即可。 |
| **验证过程以及实验结果**： |
| 采用演示截图、文字说明等方式，给出本次实验的实验结果。   1. GBN传输协议     客户端发出指令并进行接收    服务器端界面，其中数据包12未收到ack信息    客户端接收完毕    服务器端发送完毕    双向通信中的客户端    双向通信中的服务器端，所有信息接收完毕   1. SR传输协议     SR协议中的客户端    SR协议中的服务器端  可见其中多次发生未收到ACK的情况，分别进行了重新发送 |
| 问题讨论： |
| 1. 问题：返回数据错乱   原因：数据包序列号需要减一才能和窗口的位置相对应。   1. 问题：窗口总是被覆盖   原因：忘记保证发送端滑动窗口的序号数目 + 接受端滑动窗口的序号数目 <= 分组编号数目 |
| 心得体会： |
| 经过此次实验，熟悉了停等协议、GBN协议和SR协议的工作原理以及区别，了解了滑动窗口这一巧妙的设计思想，通过停等协议到GBN协议，再到SR协议的演变，数据传输的效率不停增加，工作原理也越来越复杂。 |

**GBN\_Server.cpp**

|  |
| --- |
| #include <stdlib.h>  #include <time.h>  #include <WinSock2.h>  #include <fstream>  #pragma comment(lib,"ws2\_32.lib")  #define SERVER\_PORT 12340 //端口号  #define SERVER\_IP "0.0.0.0" //IP 地址  const int PACKET\_NUM = 25;  const int BUFFER\_LENGTH = 1026; //缓冲区大小，（以太网中 UDP 的数据帧中包长度应小于 1480 字节）  const int SEND\_WIND\_SIZE = 10;//发送窗口大小为 10，GBN 中应满足 W + 1 <=N（W 为发送窗口大小，N 为序列号个数）  //本例取序列号 0...19 共 20 个  //如果将窗口大小设为 1，则为停-等协议  const int SEQ\_SIZE = 20; //序列号的个数，从 0~19 共计 20 个  //由于发送数据第一个字节如果值为 0，则数据会发送失败，因为0直接代表字符串的结尾了  //因此接收端序列号为 1~20，与发送端一一对应  BOOL ack[SEQ\_SIZE];//收到 ack 情况，对应 0~19 的 ack  int curSeq;//当前数据包的 seq  int curAck;//当前等待确认的 ack  int totalSeq;//收到的包的总数  int totalPacket;//需要发送的包总数  int totalAck;//已经确认的总数,用于判断是否可以停止传输  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: lossInLossRatio  // FullName: lossInLossRatio  // Access: public  // Returns: BOOL  // Qualifier: 根据丢失率随机生成一个数字，判断是否丢失,丢失则返回TRUE，否则返回 FALSE  // Parameter: float lossRatio [0,1]  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  BOOL lossInLossRatio(float lossRatio) {  int lossBound = (int)(lossRatio \* 100);  int r = rand() % 100;  if (r < lossBound) {  return TRUE;  }  return FALSE;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: getCurTime  // FullName: getCurTime  // Access: public  // Returns: void  // Qualifier: 获取当前系统时间，结果存入 ptime 中  // Parameter: char \* ptime  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void getCurTime(char\* ptime) {  char buffer[128];  memset(buffer, 0, sizeof(buffer));  time\_t c\_time;  struct tm\* p;  time(&c\_time);  p = localtime(&c\_time);  sprintf\_s(buffer, "%d/%d/%d %d:%d:%d", p->tm\_year + 1900,  p->tm\_mon, p->tm\_mday, p->tm\_hour, p->tm\_min, p->tm\_sec);  strcpy\_s(ptime, sizeof(buffer), buffer);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: seqIsAvailable  // FullName: seqIsAvailable  // Access: public  // Returns: bool  // Qualifier: 当前序列号 curSeq 是否可用  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  bool seqIsAvailable() {  int step;  step = curSeq - curAck;  step = step >= 0 ? step : step + SEQ\_SIZE;//序列号是循环使用的  //序列号是否在当前发送窗口之内  if (step >= SEND\_WIND\_SIZE) {  return false;  }  if (ack[curSeq]) {  return true;  }  return false;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: timeoutHandler  // FullName: timeoutHandler  // Access: public  // Returns: void  // Qualifier: 超时重传处理函数，滑动窗口内的数据帧都要重传  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void timeoutHandler() {  printf("Timer out error.\n");  int index;  int res = 0;  for (int i = 0; i < SEND\_WIND\_SIZE; ++i) {  //将已经发出去的，还没收到ack的，重新置为还没发  index = (i + curAck) % SEQ\_SIZE;  if (ack[index] == FALSE)res++;  ack[index] = TRUE;  }  totalSeq -= res;  curSeq = curAck;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: ackHandler  // FullName: ackHandler  // Access: public  // Returns: void  // Qualifier: 收到 ack，累积确认，取数据帧的第一个字节  //由于发送数据时，第一个字节（序列号）为 0（ASCII）时发送失败，因此加一了，此处需要减一还原  // Parameter: char c  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void ackHandler(char c) {  unsigned char index = (unsigned char)c - 1; //序列号减一  printf("Recv a ack of %d\n", index);  if (curAck <= index) {  for (int i = curAck; i <= index; ++i) {  ack[i] = TRUE;  totalAck++;  }  curAck = (index + 1) % SEQ\_SIZE;  }  else {  //这种情况可能有两种原因导致，一种是发生丢包，另一种是用于序列号循环使用  //但考虑到发生连续好几个ack都丢了的概率较低，根据两者差值判断是哪种情况  //ack 超过了最大值，回到了 curAck 的左边  if (curAck - index > SEND\_WIND\_SIZE) {  for (int i = curAck; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  totalAck++;  }  for (int i = 0; i <= index; ++i) {  ack[i] = TRUE;  totalAck++;  }  }  curAck = index + 1;  }  }  //主函数  int main(int argc, char\* argv[]) {  //加载套接字库（必须）  WORD wVersionRequested;  WSADATA wsaData;  //套接字加载时错误提示  int err;  //版本 2.2  wVersionRequested = MAKEWORD(2, 2);  //加载 dll 文件 Scoket 库  err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  //找不到 winsock.dll  printf("WSAStartup failed with error: %d\n", err);  return -1;  }  if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)  {  printf("Could not find a usable version of Winsock.dll\n");  WSACleanup();  }  else {  printf("The Winsock 2.2 dll was found okay\n");  }  //双向传输，所以服务器端也要设置丢包率  float packetLossRatio = 0;  float ackLossRatio = 0;  srand((unsigned)time(NULL));  SOCKET sockServer = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);  //设置套接字为非阻塞模式  int iMode = 1; //1：非阻塞，0：阻塞  ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);//非阻塞设置  SOCKADDR\_IN addrServer; //服务器地址  //addrServer.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER\_IP);  addrServer.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);//两者均可  addrServer.sin\_family = AF\_INET;  addrServer.sin\_port = htons(SERVER\_PORT);  err = bind(sockServer, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  if (err) {  err = GetLastError();  printf("Could not bind the port %d for socket.Error code is % d\n", SERVER\_PORT, err);  WSACleanup();  return -1;  }  SOCKADDR\_IN addrClient; //客户端地址  int length = sizeof(SOCKADDR);  char buffer[BUFFER\_LENGTH]; //数据发送接收缓冲区  ZeroMemory(buffer, sizeof(buffer));  //将测试数据读入内存  std::ifstream icin;  icin.open("../test.txt");  char data[1024 \* PACKET\_NUM];  ZeroMemory(data, sizeof(data));  icin.read(data, 1024 \* PACKET\_NUM);  icin.close();  totalPacket = sizeof(data) / 1024;  //printf("%d\n", totalPacket);  int recvSize;  for (int i = 0; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  }  while (true) {  //非阻塞接收，若没有收到数据，返回值为-1  recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);  if (recvSize < 0) {  Sleep(200);  continue;  }  printf("recv from client: %s\n", buffer);  if (strcmp(buffer, "-time") == 0) {  getCurTime(buffer);  }  else if (strcmp(buffer, "-quit") == 0) {  strcpy\_s(buffer, strlen("Good bye!") + 1, "Good bye!");  }  else if (strcmp(buffer, "-testgbn") == 0) {  //进入 gbn 测试阶段  //首先 server（server 处于 0 状态）向 client 发送 205 状态码（server进入 1 状态）  //server 等待 client 回复 200 状态码，如果收到（server 进入 2 状态），则开始传输文件，否则延时等待直至超时  //在文件传输阶段，server 发送窗口大小设为  for (int i = 0; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  }  ZeroMemory(buffer, sizeof(buffer));  int recvSize;  int waitCount = 0;  printf("Begin to test GBN protocol,please don't abort the process\n");  //加入了一个握手阶段  //首先服务器向客户端发送一个 205 的状态码（我自己定义的）表示服务器准备好了，可以发送数据  //客户端收到 205 之后回复一个 200 的状态码，表示客户端准备好了，可以接收数据了  //服务器收到 200 状态码之后，就开始使用 GBN 发送数据了  printf("Shake hands stage\n");  int stage = 0;  bool runningFlag = true;  int isFinished = 0;  while (runningFlag) {  if (isFinished) {  break;  }  switch (stage) {  case 0://发送 205 阶段  buffer[0] = 205;  sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  Sleep(100);  stage = 1;  break;  case 1://等待接收 200 阶段，没有收到则计数器+1，超时则放弃此次“连接”，等待从第一步开始  recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);  if (recvSize < 0) {  ++waitCount;  if (waitCount > 20) {  runningFlag = false;  printf("Timeout error\n");  break;  }  Sleep(500);  continue;  }  else {  if ((unsigned char)buffer[0] == 200) {  printf("Begin a file transfer\n");  printf("File size is %dB, each packet is 1024B and packet total num is % d\n", sizeof(data), totalPacket);  curSeq = 0;  curAck = 0;  totalSeq = 0;  totalAck = 0;  waitCount = 0;  stage = 2;  }  }  break;  case 2://数据传输阶段  if (seqIsAvailable() && totalSeq<totalPacket) {//第二个判断是处理已经发出但因为部分ack丢失，还未确认的情况。这时是暂时不需要传的  //发送给客户端的序列号从 1 开始  buffer[0] = curSeq + 1;  ack[curSeq] = FALSE;  //数据发送的过程中应该判断是否传输完成  //为简化过程此处并未实现  memcpy(&buffer[1], data + 1024 \* totalSeq, 1024);  printf("send a packet with a seq of %d\n", curSeq);  sendto(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  ++curSeq;  curSeq %= SEQ\_SIZE;  ++totalSeq;  Sleep(500);  }  //等待 Ack，若没有收到，则返回值为-1，计数器+1  recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);  if (recvSize < 0) {  waitCount++;  //20 次等待 ack 则超时重传  if (waitCount > 20)  {  timeoutHandler();  waitCount = 0;  }  }  else {  //收到 ack  ackHandler(buffer[0]);  waitCount = 0;  //printf("%d\n", totalAck);  if (totalAck == totalPacket ) {  isFinished = 1;  strcpy(buffer, "All Finished!\n");  }  }  Sleep(500);  break;  }  }  }  else if (strcmp(buffer, "-testgbn\_send") == 0) {//双向传输的情况，服务器传的情况已经测试过了，所以这里测试服务器收  FILE\* fser = fopen("serverfile.txt", "w+");  int iMode = 0; //1：非阻塞，0：阻塞  ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*)& iMode);//阻塞设置  printf("%s\n", "Begin to test GBN protocol, please don't abort the process");  printf("The loss ratio of packet is %.2f,the loss ratio of ack is %.2f\n", packetLossRatio, ackLossRatio);  int waitCount = 0;  int stage = 0;  BOOL b;  unsigned char u\_code;//状态码  unsigned short seq;//包的序列号  unsigned short recvSeq;//接收窗口大小为 1，已确认的序列号  unsigned short waitSeq;//等待的序列号  int len = sizeof(SOCKADDR);  while (true)  {  //等待 server 回复设置 UDP 为阻塞模式  recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, &len);  if (strcmp(buffer, "All Finished!\n") == 0) {  fclose(fser);//写完之后关闭文件  break;  }  switch (stage) {  case 0://等待握手阶段  u\_code = (unsigned char)buffer[0];  if ((unsigned char)buffer[0] == 205)  {  printf("Ready for file transmission\n");  buffer[0] = 200;  buffer[1] = '\0';  sendto(sockServer, buffer, 2, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  stage = 1;  recvSeq = 0;  waitSeq = 1;  }  break;  case 1://等待接收数据阶段  seq = (unsigned short)buffer[0];  //随机法模拟包是否丢失  b = lossInLossRatio(packetLossRatio);  if (b) {  printf("The packet with a seq of %d loss\n", seq);  continue;  }  printf("recv a packet with a seq of %d\n", seq);  //如果是期待的包，正确接收，正常确认即可  if (!(waitSeq - seq)) {  ++waitSeq;  fwrite(&buffer[1], sizeof(char), strlen(buffer) - 1, fser);  if (waitSeq == 21) {  waitSeq = 1;  }  //输出数据  printf("%s\n",&buffer[1]);  buffer[0] = seq;  recvSeq = seq;  buffer[1] = '\0';  }  else {  //如果当前一个包都没有收到，则等待 Seq 为 1 的数据包，不是则不返回 ACK（因为并没有上一个正确的 ACK）  if (!recvSeq) {  continue;  }  buffer[0] = recvSeq;  buffer[1] = '\0';  }  b = lossInLossRatio(ackLossRatio);  if (b) {  printf("The ack of %d loss\n", (unsigned char)buffer[0]);  continue;  }  sendto(sockServer, buffer, 2, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  printf("send a ack of %d\n", (unsigned char)buffer[0]);  break;  }  Sleep(500);  }  }  //printf("%s\n", buffer);  sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  Sleep(500);  }  //关闭套接字，卸载库  closesocket(sockServer);  WSACleanup();  return 0;  } |

**GBN\_Client.cpp**

|  |
| --- |
| // GBN\_client.cpp : 定义控制台应用程序的入口点。  //  #include <stdlib.h>  #include <WinSock2.h>  #include <time.h>  #include <stdio.h>  #include <fstream>  #pragma comment(lib,"ws2\_32.lib")  #define SERVER\_PORT 12340 //接收数据的端口号  #define SERVER\_IP "127.0.0.1" // 服务器的 IP 地址  const int BUFFER\_LENGTH = 1026;  const int PACKET\_NUM = 25;  const int SEQ\_SIZE = 20;//接收端序列号个数，为 1~20  const int SEND\_WIND\_SIZE = 10;//发送窗口大小为 10，GBN 中应满足 W + 1 <=N（W 为发送窗口大小，N 为序列号个数）  BOOL ack[SEQ\_SIZE]; //收到 ack 情况，对应 0~19 的 ack  int curSeq; //当前数据包的 seq  int curAck; //当前等待确认的 ack  int totalPacket; //需要发送的包总数  int totalSeq; //已发送的包的总数  int totalAck; //确认收到（ack）的包的总数  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* -time 从服务器端获取当前时间  -quit 退出客户端  -testgbn [X][Y] 测试 GBN 协议实现可靠数据传输  [X] [0,1] 模拟数据包丢失的概率  [Y] [0,1] 模拟 ACK 丢失的概率  \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void printTips() {  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf("| -time to get current time |\n");  printf("| -quit to exit client |\n");  printf("| -testgbn [X] [Y] to test the gbn |\n");  printf("| -testgbn\_send [X] [Y] to test the gbn |\n");  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: lossInLossRatio  // FullName: lossInLossRatio  // Access: public  // Returns: BOOL  // Qualifier: 根据丢失率随机生成一个数字，判断是否丢失,丢失则返回TRUE，否则返回 FALSE  // Parameter: float lossRatio [0,1]  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  BOOL lossInLossRatio(float lossRatio) {  int lossBound = (int)(lossRatio \* 100);  int r = rand() % 100;  if (r < lossBound) {  return TRUE;  }  return FALSE;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: seqIsAvailable  // FullName: seqIsAvailable  // Access: public  // Returns: bool  // Qualifier: 当前序列号 curSeq 是否可用  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  bool seqIsAvailable() {  int step;  step = curSeq - curAck;  step = step >= 0 ? step : step + SEQ\_SIZE;  //序列号是否在当前发送窗口之内  if (step >= SEND\_WIND\_SIZE) {  return false;  }  if (ack[curSeq]) {  return true;  }  return false;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: timeoutHandler  // FullName: timeoutHandler  // Access: public  // Returns: void  // Qualifier: 超时重传处理函数，滑动窗口内的数据帧都要重传  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void timeoutHandler() {  printf("Timer out error.\n");  int index;  int res = 0;  for (int i = 0; i < SEND\_WIND\_SIZE; ++i) {//将已经发出去的，还没收到ack的，重新置为还没发  index = (i + curAck) % SEQ\_SIZE;  if (ack[index] == FALSE)res++;  ack[index] = TRUE;  }  totalSeq -= res; // 重置数量  curSeq = curAck;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: ackHandler  // FullName: ackHandler  // Access: public  // Returns: void  // Qualifier: 收到 ack，累积确认，取数据帧的第一个字节  //由于发送数据时，第一个字节（序列号）为 0（ASCII）时发送失败，因此加一了，此处需要减一还原  // Parameter: char c  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void ackHandler(char c) {  unsigned char index = (unsigned char)c - 1; //序列号减一  printf("Recv a ack of %d\n", index);  if (curAck <= index) {  for (int i = curAck; i <= index; ++i) { //对当前的ack进行标记  ack[i] = TRUE;  totalAck++;  }  curAck = (index + 1) % SEQ\_SIZE;  }  else {  //这种情况可能有两种原因导致，一种是发生丢包，另一种是用于序列号循环使用  //但考虑到发生连续好几个ack都丢了的概率较低，根据两者差值判断是哪种情况  //ack 超过了最大值，回到了 curAck 的左边  if (curAck - index > SEND\_WIND\_SIZE) {  for (int i = curAck; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  totalAck++;  }  for (int i = 0; i <= index; ++i) {  ack[i] = TRUE;  totalAck++;  }  }  curAck = index + 1;  }  }  int main(int argc, char\* argv[]) {  //加载套接字库（必须）  WORD wVersionRequested;  WSADATA wsaData;  //套接字加载时错误提示  int err;  //版本 2.2  wVersionRequested = MAKEWORD(2, 2);  //加载 dll 文件 Scoket 库  err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  //找不到 winsock.dll  printf("WSAStartup failed with error: %d\n", err);  return 1;  }  if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) {  printf("Could not find a usable version of Winsock.dll\n");  WSACleanup();  }  else {  printf("The Winsock 2.2 dll was found okay\n");  }  SOCKET socketClient = socket(AF\_INET, SOCK\_DGRAM, 0);  SOCKADDR\_IN addrServer;  addrServer.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER\_IP);  addrServer.sin\_family = AF\_INET;  addrServer.sin\_port = htons(SERVER\_PORT);  //接收缓冲区  char buffer[BUFFER\_LENGTH];  ZeroMemory(buffer, sizeof(buffer));  int len = sizeof(SOCKADDR);  //为了测试与服务器的连接，可以使用 -time 命令从服务器端获得当前时间  //使用 -testgbn [X] [Y] 测试 GBN 其中[X]表示数据包丢失概率  // [Y]表示 ACK 丢包概率  printTips();  int ret;  int interval = 1;//收到数据包之后返回 ack 的间隔，默认为 1 表示每个都返回 ack，0 或者负数均表示所有的都不返回 ack  char cmd[128];  float packetLossRatio = 0; //默认包丢失率 0.2  float ackLossRatio = 0; //默认 ACK 丢失率 0.2  //用时间作为随机种子，放在循环的最外面  srand((unsigned)time(NULL));    //将测试数据读入内存  std::ifstream icin;  icin.open("../test.txt");  char data[1024 \* PACKET\_NUM];  ZeroMemory(data, sizeof(data));  icin.read(data, 1024 \* PACKET\_NUM);  icin.close();  totalPacket = sizeof(data) / 1024;  int recvSize;  for (int i = 0; i < SEQ\_SIZE; ++i) ack[i] = TRUE;  while (true) {  gets\_s(buffer);  ret = sscanf(buffer, "%s%f%f", &cmd, &packetLossRatio, &ackLossRatio);  //开始 GBN 测试，使用 GBN 协议实现 UDP 可靠文件传输  if (!strcmp(cmd, "-testgbn")) {  FILE\* fcli = fopen("clientfile.txt", "w+");  printf("%s\n", "Begin to test GBN protocol, please don't abort the process");  printf("The loss ratio of packet is %.2f,the loss ratio of ack is % .2f\n",packetLossRatio,ackLossRatio);  int waitCount = 0;  int stage = 0;  BOOL b;  unsigned char u\_code;//状态码  unsigned short seq;//包的序列号  unsigned short recvSeq;//接收窗口大小为 1，已确认的序列号  unsigned short waitSeq;//等待的序列号  sendto(socketClient, "-testgbn", strlen("-testgbn") + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  while (true) {  //等待 server 回复设置 UDP 为阻塞模式  recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, &len);  if (strcmp(buffer, "All Finished!\n") == 0) {  fclose(fcli);  break;  }  switch (stage) {  case 0://等待握手阶段  u\_code = (unsigned char)buffer[0];  if ((unsigned char)buffer[0] == 205)  {  printf("Ready for file transmission\n");  buffer[0] = 200;  buffer[1] = '\0';  sendto(socketClient, buffer, 2, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  stage = 1;  recvSeq = 0;  waitSeq = 1;  }  break;  case 1://等待接收数据阶段  seq = (unsigned short)buffer[0];  //随机法模拟包是否丢失  b = lossInLossRatio(packetLossRatio);  if (b) {  printf("The packet with a seq of %d loss\n", seq);  continue;  }  printf("recv a packet with a seq of %d\n", seq);  //如果是期待的包，正确接收，正常确认即可  if (!(waitSeq - seq)) {  ++waitSeq;  // 将接收的文件写入输出文件中  fwrite(&buffer[1], sizeof(char), strlen(buffer) - 1, fcli);  if (waitSeq == 21) {  waitSeq = 1;  }  //输出数据  printf("%s\n",&buffer[1]);  buffer[0] = seq;  recvSeq = seq;  buffer[1] = '\0';  }  else {  //如果当前一个包都没有收到，则等待 Seq 为 1 的数据包，不是则不返回 ACK（因为并没有上一个正确的 ACK）  if (!recvSeq) {  continue;  }  buffer[0] = recvSeq;  buffer[1] = '\0';  }  b = lossInLossRatio(ackLossRatio);  if (b) {  printf("The ack of %d loss\n", (unsigned char)buffer[0]);  continue;  }  sendto(socketClient, buffer, 2, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  printf("send a ack of %d\n", (unsigned char)buffer[0]);  break;  }  Sleep(500);  }  }  else if (strcmp(cmd, "-testgbn\_send") == 0) {  //进入 gbn 测试阶段  //首先 server（server 处于 0 状态）向 client 发送 205 状态码（server进入 1 状态）  //server 等待 client 回复 200 状态码，如果收到（server 进入 2 状态），则开始传输文件，否则延时等待直至超时  //在文件传输阶段，server 发送窗口大小设为 20  for (int i = 0; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  }  ZeroMemory(buffer, sizeof(buffer));  int recvSize;  int waitCount = 0;  printf("Begin to test GBN protocol,please don't abort the process\n");  //加入了一个握手阶段  //首先服务器向客户端发送一个 205 的状态码（我自己定义的）表示服务器准备好了，可以发送数据  //客户端收到 205 之后回复一个 200 的状态码，表示客户端准备好了，可以接收数据了  //服务器收到 200 状态码之后，就开始使用 GBN 发送数据了  printf("Shake hands stage\n");  int stage = 0;  bool runningFlag = true;  int flag = 0;  sendto(socketClient, "-testgbn\_send", strlen("-testgbn\_send") + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  int iMode = 1; //1：非阻塞，0：阻塞  ioctlsocket(socketClient, FIONBIO, (u\_long FAR\*)& iMode);//非阻塞设置  while (runningFlag) {  if (flag) {  break;  }  switch (stage) {  case 0://发送 205 阶段  buffer[0] = 205;  sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  Sleep(100);  stage = 1;  break;  case 1://等待接收 200 阶段，没有收到则计数器+1，超时则放弃此次“连接”，等待从第一步开始  recvSize = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrServer), &len);  if (recvSize < 0) {  ++waitCount;  if (waitCount > 20) {  runningFlag = false;  printf("Timeout error\n");  break;  }  Sleep(500);  continue;  }  else {  if ((unsigned char)buffer[0] == 200) {  printf("Begin a file transfer\n");  printf("File size is %dB, each packet is 1024B and packet total num is %d\n", sizeof(data), totalPacket);  curSeq = 0;  curAck = 0;  totalSeq = 0;  totalAck = 0;  waitCount = 0;  stage = 2;  }  }  break;  case 2://数据传输阶段  if (seqIsAvailable() && totalSeq < totalPacket) {//第二个判断是处理已经发出但因为部分ack丢失，还未确认的情况。这时是暂时不需要传的  //发送给客户端的序列号从 1 开始  buffer[0] = curSeq + 1;  ack[curSeq] = FALSE;  //数据发送的过程中应该判断是否传输完成  //为简化过程此处并未实现  memcpy(&buffer[1], data + 1024 \* totalSeq, 1024);  printf("send a packet with a seq of %d\n", curSeq);  sendto(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  ++curSeq;  curSeq %= SEQ\_SIZE;  ++totalSeq;  Sleep(500);  }  //等待 Ack，若没有收到，则返回值为-1，计数器+1  recvSize = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrServer), &len);  if (recvSize < 0) {  waitCount++;  //20 次等待 ack 则超时重传  if (waitCount > 20)  {  timeoutHandler();  waitCount = 0;  }  }  else {  //收到 ack  ackHandler(buffer[0]);  waitCount = 0;  // printf("%d\n", totalAck);  if (totalAck == totalPacket) {  flag = 1;  strcpy(buffer, "All Finished!\n");  }  }  Sleep(500);  break;  }  }  }  //printf("%s\n", buffer);  sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  ret = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, &len);  printf("%s\n", buffer);  if (!strcmp(buffer, "Good bye!")) {  break;  }  printTips();  }  //关闭套接字  closesocket(socketClient);  WSACleanup();  return 0;  } |

**SR\_Server.cpp**

|  |
| --- |
| #include <stdlib.h>  #include <time.h>  #include <WinSock2.h>  #include <fstream>  #pragma comment(lib,"ws2\_32.lib")  #define SERVER\_PORT 12340 //端口号  #define SERVER\_IP "127.0.0.1" //IP 地址  const int PACKET\_NUM = 25;  const int BUFFER\_LENGTH = 1026; //缓冲区大小，（以太网中 UDP 的数据帧中包长度应小于 1480 字节）  const int SEND\_WIND\_SIZE = 10;//发送窗口大小为 10，GBN 中应满足 W + 1 <=N（W 为发送窗口大小，N 为序列号个数）  //本例取序列号 0...19 共 20 个  //如果将窗口大小设为 1，则为停-等协议  const int SEQ\_SIZE = 20; //序列号的个数，从 0~19 共计 20 个  //由于发送数据第一个字节如果值为 0，则数据会发送失败，因为0直接代表字符串的结尾了  //因此接收端序列号为 1~20，与发送端一一对应  BOOL ack[SEQ\_SIZE];//收到 ack 情况，对应 0~19 的 ack，值为false不能用（已经被接受但还在窗口里）  int curSeq;//当前数据包的 seq  int curAck;//当前等待确认的 ack,同时也代表了当前窗口位置  int totalPacket;//需要发送的包总数  int totalSeq;//已经发的总数  int totalAck;//已经确认的总数,用于判断是否可以停止传输  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: getCurTime  // FullName: getCurTime  // Access: public  // Returns: void  // Qualifier: 获取当前系统时间，结果存入 ptime 中  // Parameter: char \* ptime  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void getCurTime(char\* ptime) {  char buffer[128];  memset(buffer, 0, sizeof(buffer));  time\_t c\_time;  struct tm\* p;  time(&c\_time);  p = localtime(&c\_time);  sprintf\_s(buffer, "%d/%d/%d %d:%d:%d", p->tm\_year + 1900, p->tm\_mon, p->tm\_mday, p->tm\_hour, p->tm\_min, p->tm\_sec);  strcpy\_s(ptime, sizeof(buffer), buffer);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: seqIsAvailable  // FullName: seqIsAvailable  // Access: public  // Returns: bool  // Qualifier: 当前序列号 curSeq 是否可用  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  bool seqIsAvailable() {  int step;  step = curSeq - curAck;  step = step >= 0 ? step : step + SEQ\_SIZE;//序列号是循环使用的  //序列号是否在当前发送窗口之内  if (step >= SEND\_WIND\_SIZE) {  return false;  }  //与gbn不同的是，只要在窗口内就发送  return true;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: timeoutHandler  // FullName: timeoutHandler  // Access: public  // Returns: void  // Qualifier: 超时重传处理函数  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void timeoutHandler() {  printf("Timer out error.\n");  totalSeq -= SEND\_WIND\_SIZE;  curSeq = curAck;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: ackHandler  // FullName: ackHandler  // Access: public  // Returns: void  // Qualifier: 收到 ack  //由于发送数据时，第一个字节（序列号）为 0（ASCII）时发送失败，因此加一了，此处需要减一还原  // Parameter: char c  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void ackHandler(char c) {  unsigned char index = (unsigned char)c - 1; //序列号减一  printf("Recv a ack of %d\n", index);  totalAck++;  //如果ack的是窗口的第一个，右移窗口  if (curAck == index) {  ack[index] = TRUE;  curAck = (index + 1) % SEQ\_SIZE;  //看可以右移几位  for (int i = 1; i < SEND\_WIND\_SIZE; ++i) {  int nxt = (i + index) % SEQ\_SIZE;  if (ack[nxt] == FALSE) {  ack[nxt] == TRUE;  curAck = (nxt + 1) % SEQ\_SIZE;  totalSeq++;  }  else break;  }  curSeq = curAck;  }  else {  ack[index] = FALSE;  }  }  //主函数  int main(int argc, char\* argv[])  {  //加载套接字库（必须）  WORD wVersionRequested;  WSADATA wsaData;  //套接字加载时错误提示  int err;  //版本 2.2  wVersionRequested = MAKEWORD(2, 2);  //加载 dll 文件 Scoket 库  err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  //找不到 winsock.dll  printf("WSAStartup failed with error: %d\n", err);  return -1;  }  if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)  {  printf("Could not find a usable version of Winsock.dll\n");  WSACleanup();  }  else {  printf("The Winsock 2.2 dll was found okay\n");  }  //双向传输，所以服务器端也要设置丢包率  float packetLossRatio = 0;  float ackLossRatio = 0;  srand((unsigned)time(NULL));  SOCKET sockServer = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);  //设置套接字为非阻塞模式  int iMode = 1; //1：非阻塞，0：阻塞  ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);//非阻塞设置  SOCKADDR\_IN addrServer; //服务器地址  //addrServer.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER\_IP);  addrServer.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);//两者均可  addrServer.sin\_family = AF\_INET;  addrServer.sin\_port = htons(SERVER\_PORT);  err = bind(sockServer, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  if (err) {  err = GetLastError();  printf("Could not bind the port %d for socket.Error code is % d\n", SERVER\_PORT, err);  WSACleanup();  return -1;  }  SOCKADDR\_IN addrClient; //客户端地址  int length = sizeof(SOCKADDR);  char buffer[BUFFER\_LENGTH]; //数据发送接收缓冲区  ZeroMemory(buffer, sizeof(buffer));  //将测试数据读入内存  std::ifstream icin;  icin.open("../test.txt");  char data[1024 \* PACKET\_NUM];  ZeroMemory(data, sizeof(data));  icin.read(data, 1024 \* PACKET\_NUM);  icin.close();  totalPacket = sizeof(data) / 1024;  //printf("%d\n", totalPacket);  int recvSize;  for (int i = 0; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  }  while (true) {  //非阻塞接收，若没有收到数据，返回值为-1  recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);  if (recvSize < 0) {  Sleep(200);  continue;  }  printf("recv from client: %s\n", buffer);  if (strcmp(buffer, "-time") == 0) {  getCurTime(buffer);  }  else if (strcmp(buffer, "-quit") == 0) {  strcpy\_s(buffer, strlen("Good bye!") + 1, "Good bye!");  }  else if (strcmp(buffer, "-testsr") == 0) {  //进入 gbn 测试阶段  //首先 server（server 处于 0 状态）向 client 发送 205 状态码（server进入 1 状态）  //server 等待 client 回复 200 状态码，如果收到（server 进入 2 状态），则开始传输文件，否则延时等待直至超时  //在文件传输阶段，server 发送窗口大小设为  for (int i = 0; i < SEQ\_SIZE; ++i) {  ack[i] = TRUE;  }  ZeroMemory(buffer, sizeof(buffer));  int recvSize;  int waitCount = 0;  printf("Begin to test GBN protocol,please don't abort the process\n");  //加入了一个握手阶段  //首先服务器向客户端发送一个 205 的状态码（我自己定义的）表示服务器准备好了，可以发送数据  //客户端收到 205 之后回复一个 200 的状态码，表示客户端准备好了，可以接收数据了  //服务器收到 200 状态码之后，就开始使用 GBN 发送数据了  printf("Shake hands stage\n");  int stage = 0;  bool runFlag = true;  int flag = 0;  while (runFlag) {  if (flag) {  break;  }  switch (stage) {  case 0://发送 205 阶段  buffer[0] = 205;  sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  Sleep(100);  stage = 1;  break;  case 1://等待接收 200 阶段，没有收到则计数器+1，超时则放弃此次“连接”，等待从第一步开始  recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);  if (recvSize < 0) {  ++waitCount;  if (waitCount > 20) {  runFlag = false;  printf("Timeout error\n");  break;  }  Sleep(500);  continue;  }  else {  if ((unsigned char)buffer[0] == 200) {  printf("Begin a file transfer\n");  printf("File size is %dB, each packet is 1024B and packet total num is % d\n", sizeof(data), totalPacket);  curSeq = 0;  curAck = 0;  waitCount = 0;  stage = 2;  }  }  break;  case 2://数据传输阶段  if (seqIsAvailable()) {  //发送给客户端的序列号从 1 开始  buffer[0] = curSeq + 1;  ack[curSeq] = TRUE;//设为等待接收  //数据发送的过程中应该判断是否传输完成  //为简化过程此处并未实现  memcpy(&buffer[1], data + 1024 \* totalSeq, 1024);  printf("send a packet with a seq of %d\n", curSeq);  sendto(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  ++curSeq;  curSeq %= SEQ\_SIZE;  ++totalSeq;  Sleep(500);  }  //等待 Ack，若没有收到，则返回值为-1，计数器+1  recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);  if (recvSize < 0) {  waitCount++;  //20 次等待 ack 则超时重传  if (waitCount > 20)  {  timeoutHandler();  waitCount = 0;  }  }  else {  //收到 ack  ackHandler(buffer[0]);  waitCount = 0;  //printf("%d\n", totalAck);  if (totalAck == totalPacket) {  flag = 1;  strcpy(buffer, "All Finished!\n");  }  }  Sleep(500);  break;  }  }  }  sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));  Sleep(500);  }  //关闭套接字，卸载库  closesocket(sockServer);  WSACleanup();  return 0;  } |

**SR\_Client.cpp**

|  |
| --- |
| // GBN\_client.cpp : 定义控制台应用程序的入口点。  //  #include <stdlib.h>  #include <WinSock2.h>  #include <time.h>  #include <stdio.h>  #include <fstream>  #pragma comment(lib,"ws2\_32.lib")  #define SERVER\_PORT 12340 //接收数据的端口号  #define SERVER\_IP "127.0.0.1" // 服务器的 IP 地址  const int BUFFER\_LENGTH = 1026;  const int PACKET\_NUM = 25;  const int SEQ\_SIZE = 20;//接收端序列号个数，为 1~20  const int RECV\_WIND\_SIZE = 10;//接收窗口大小为 10，SR中应满足接收窗口和发送窗口的大小和小于等于总序列号个数  int curSeq; //当前数据包的 seq  int curAck; //当前等待确认的 ack  int totalPacket; //需要发送的包总数  BOOL ack[SEQ\_SIZE];//接收方窗口  int totalSeq; //已发送的包的总数  int totalAck; //确认收到（ack）的包的总数  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* -time 从服务器端获取当前时间  -quit 退出客户端  -testgbn [X][Y] 测试 GBN 协议实现可靠数据传输  [X] [0,1] 模拟数据包丢失的概率  [Y] [0,1] 模拟 ACK 丢失的概率  \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void printTips() {  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf("| -time to get current time |\n");  printf("| -quit to exit client |\n");  printf("| -testsr [X] [Y] to test the gbn |\n");  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: lossInLossRatio  // FullName: lossInLossRatio  // Access: public  // Returns: BOOL  // Qualifier: 根据丢失率随机生成一个数字，判断是否丢失,丢失则返回TRUE，否则返回 FALSE  // Parameter: float lossRatio [0,1]  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  BOOL lossInLossRatio(float lossRatio) {  int lossBound = (int)(lossRatio \* 100);  int r = rand() % 100;  if (r < lossBound) {  return TRUE;  }  return FALSE;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Method: seqIsAvailable  // FullName: seqIsAvailable  // Access: public  // Returns: bool  // Qualifier: 当前序列号seq是否可接收  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  bool seqRecvAvailable(int seq) {  int step;  step = seq - curAck;  step = step >= 0 ? step : step + SEQ\_SIZE;  //序列号是否在当前接收窗口之内  if (step >= RECV\_WIND\_SIZE) {  return false;  }  return true;  }  int main(int argc, char\* argv[])  {  //加载套接字库（必须）  WORD wVersionRequested;  WSADATA wsaData;  //套接字加载时错误提示  int err;  //版本 2.2  wVersionRequested = MAKEWORD(2, 2);  //加载 dll 文件 Scoket 库  err = WSAStartup(wVersionRequested, &wsaData);  if (err != 0) {  //找不到 winsock.dll  printf("WSAStartup failed with error: %d\n", err);  return 1;  }  if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)  {  printf("Could not find a usable version of Winsock.dll\n");  WSACleanup();  }  else {  printf("The Winsock 2.2 dll was found okay\n");  }  SOCKET socketClient = socket(AF\_INET, SOCK\_DGRAM, 0);  SOCKADDR\_IN addrServer;  addrServer.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER\_IP);  addrServer.sin\_family = AF\_INET;  addrServer.sin\_port = htons(SERVER\_PORT);  //接收缓冲区  char buffer[BUFFER\_LENGTH];  ZeroMemory(buffer, sizeof(buffer));  int len = sizeof(SOCKADDR);  //为了测试与服务器的连接，可以使用 -time 命令从服务器端获得当前时间  //使用 -testgbn [X] [Y] 测试 GBN 其中[X]表示数据包丢失概率  // [Y]表示 ACK 丢包概率  printTips();  int ret;  int interval = 1;//收到数据包之后返回 ack 的间隔，默认为 1 表示每个都返回 ack，0 或者负数均表示所有的都不返回 ack  char cmd[128];  float packetLossRatio = 0; //默认包丢失率 0.2  float ackLossRatio = 0; //默认 ACK 丢失率 0.2  //用时间作为随机种子，放在循环的最外面  srand((unsigned)time(NULL));  //将测试数据读入内存  std::ifstream icin;  icin.open("../test.txt");  char data[1024 \* PACKET\_NUM];  ZeroMemory(data, sizeof(data));  icin.read(data, 1024 \* PACKET\_NUM);  icin.close();  totalPacket = sizeof(data) / 1024;  int recvSize;  while (true) {  gets\_s(buffer);  ret = sscanf(buffer, "%s%f%f", &cmd, &packetLossRatio, &ackLossRatio);  //开始 SR 测试，使用 SR 协议实现 SR 可靠文件传输  if (!strcmp(cmd, "-testsr")) {  for (int i = 0; i < SEQ\_SIZE; ++i) {  ack[i] = FALSE;  }  FILE\* fcli = fopen("clientfile.txt", "w+");  printf("%s\n", "Begin to test GBN protocol, please don't abort the process");  printf("The loss ratio of packet is %.2f,the loss ratio of ack is % .2f\n", packetLossRatio, ackLossRatio);  int waitCount = 0;  int stage = 0;  curAck = 0;  BOOL b;  unsigned char u\_code;//状态码  unsigned short seq;//包的序列号  unsigned short recvSeq;//接收窗口大小为 1，已确认的序列号  unsigned short waitSeq;//等待的序列号  sendto(socketClient, "-testsr", strlen("-testgbn") + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  while (true)  {  //等待 server 回复设置 UDP 为阻塞模式  recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, &len);  if (strcmp(buffer, "All Finished!\n") == 0) {  fclose(fcli);  break;  }  switch (stage) {  case 0://等待握手阶段  u\_code = (unsigned char)buffer[0];  if ((unsigned char)buffer[0] == 205)  {  printf("Ready for file transmission\n");  buffer[0] = 200;  buffer[1] = '\0';  sendto(socketClient, buffer, 2, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  stage = 1;  recvSeq = 0;  }  break;  case 1://等待接收数据阶段  seq = (unsigned short)buffer[0];  //随机法模拟包是否丢失  b = lossInLossRatio(packetLossRatio);  if (b) {  printf("The packet with a seq of %d loss\n", seq);  continue;  }  printf("recv a packet with a seq of %d\n", seq);  //在窗口内即接收  if (seqRecvAvailable(seq)) {  fwrite(&buffer[1], sizeof(char), strlen(buffer) - 1, fcli);  //输出数据  printf("%s\n",&buffer[1]);  recvSeq = seq;  ack[recvSeq - 1] = TRUE;  buffer[0] = seq;  buffer[1] = '\0';  if (seq - 1 == curAck) {  int nxt;  for (int i = 0; i < RECV\_WIND\_SIZE; ++i) {  nxt = (curAck + i) % SEQ\_SIZE;  if (!ack[nxt])break;  }  curAck = (nxt + 1) % SEQ\_SIZE;  }  }  else {  //如果当前一个包都没有收到，则等待 Seq 为 1 的数据包，不是则不返回 ACK（因为并没有上一个正确的 ACK）  recvSeq = seq;  buffer[0] = recvSeq;  buffer[1] = '\0';  }  b = lossInLossRatio(ackLossRatio);  if (b) {  printf("The ack of %d loss\n", (unsigned char)buffer[0]);  continue;  }  sendto(socketClient, buffer, 2, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  printf("send a ack of %d\n", (unsigned char)buffer[0]);  break;  }  Sleep(500);  }  }  //printf("%s\n", buffer);  sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));  ret = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, &len);  printf("%s\n", buffer);  if (!strcmp(buffer, "Good bye!")) {  break;  }  printTips();  }  //关闭套接字  closesocket(socketClient);  WSACleanup();  return 0;  } |