

**计算机网络**

**课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 实验2&3 可靠数据传输协议-停等& GBN协议的设计与实现 | | | | | |
| 姓名 | 何栩晟 | | 院系 | 计算机科学与技术 | | |
| 班级 | 2103101 | | 学号 | 2021113634 | | |
| 任课教师 | 刘亚维 | | 指导教师 | 刘亚维 | | |
| 实验地点 | 格物213 | | 实验时间 | 2023.10.23 | | |
| 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： |
| 本次实验的主要目的：   1. 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。 2. 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。 3. 掌握 SR 协议的工作原理，设计并实现一个 SR 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；  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；  5) 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）；  6) 模拟引入数据包的丢失，验证所设计协议的有效性；  7) 改进所设计的GBN协议，支持双向数据传输；  8) 将所设计的GBN协议改进为SR协议。 |
| 实验过程： |
| 1. 设计 GBN 协议   服务器端：在本实验中，采用UDP协议传输数据（具体实现中，数据源为预备的测试文件）。首先，服务器端等待客户端的请求。当服务器端接收到客户端的数据传输请求后，它将数据分割成数据报并将其发送出去。同时，服务器端会在命令行界面打印相关数据报的信息。在数据发送完成后，客户端启动计时器，等待来自客户端的ACK信息。一旦收到客户端回复的ACK，服务器端会确认已收到ACK以及ACK之前的所有数据报，然后继续发送下一个数据报，同时重置计时器。如果在计时器超时之前没有收到ACK，服务器端会触发重传机制，重传窗口内已发送的所有数据报，直到接收到最大ACK。  客户端：客户端使用UDP协议向服务器端请求数据，并接收服务器端发送的数据报，然后返回ACK确认信息。在实际程序中，需要模拟ACK丢失的情况，以便服务器端可以检测到超时并触发重传机制。  当执行数据传输指令（-testgbn）时，客户端首先向服务器端发送请求信息，然后服务器端会解析该请求并进行握手。服务器端向客户端发送一个状态码205（根据示例程序定义）。一旦客户端收到服务器端发来的状态码205，客户端将回复一个状态码200，表示客户端已准备好接收数据。握手完成后，服务器端可以读取本地文件，将其分割成数据报，并根据GBN协议的规则发送数据（采用累积确认机制，如果有数据报丢失，将重传之后的所有数据报等等）。服务器端将本地文件存储在缓存中，并将数据发送给客户端。  (i)命令解析  首先，服务器接收客户端发来的请求数据：  “-time”表示客户端请求获取当前时间，服务器回复当前时间；  “-quit”表示客户端退出，服务器回复“Good bye!”；  “-testgbn”表示客户端请求开始测试GBN协议，服务器开始进入GBN传输状态；  其他数据：则服务器直接回复原数据。   1. **void** printTips() 2. { 3. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); 4. printf("|     -time to get current time                |\n"); 5. printf("|     -quit to exit client                     |\n"); 6. printf("|     -testgbn [X] [Y] to test the gbn         |\n"); 7. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); 8. }   (ii)数据传输数据帧格式定义  在以太网中，数据帧的MTU为1500字节，所以UDP数据报的数据部分应小于1472字节（除去IP头部20字节与UDP头的8字节），为此，定义UDP数据报的数据部分格式为：    图1 UDP数据报的数据部分格式  Seq为1个字节，取值为0~255，（故序列号最多为256个）；  Data≤1024 个字节，为传输的数据；  最后一个字节放入EOF0，表示结尾。   1. buffer[0] = curSeq + 1; 2. ack[curSeq] = FALSE; 3. memcpy(&buffer[1], data + 1024 \* totalSeq, 1024);   (iii)传输开始与结束  传输开始时，向客户端发送序列号为200的分组，表示服务器准备好传输数据，等待客户端。  客户端回复205分组，表示客户端已经准备好传输数据，开始传输数据。  totalSeq > totalPacket时，表示已经输出分组数量大于总要传输的分组数量，但这时可能有的ACK还没有收到，我们需要检查分组ACK是否收到，若有没收到的ACK继续向客户端发送空分组，否则当客户端的ACK丢失，客户端不会向服务器端重传ACK；若都收到，则向客户端发送序列号为204的分组，表示传输完成，结束分组传输。  2.客户端(client)：  (i) ACK数据帧定义  因为是从服务器端到客户端的单向数据传输，因此ACK数据帧不包含任何数据，只需要将ACK发送给服务器端即可。  ACK字段为一个字节，表示序列号数值；末尾放入EOF0，表示数据结束。  (ii)命令设置  客户端的命令和服务器端的解析命令向对应，获取当前用户输入并发送给服务器并等待服务器返回数据，例如输入  “-time”可以得到服务器的当前时间。  “-testgbn [X] [Y]”命令，[X],[Y]均为[0,1]的小数，其中：  [X]表示客户端的数据包丢失率，模拟网络中报文丢失；  [Y]表示客户端的ACK的丢失率（使用随机函数完成）。  如果用户不输入，则默认丢失率均为0.2。  (iii)丢包  使用LossInLossRatio()函数，输入丢失率（分组或客户端），在函数内首先获得1到100的随机数，根据丢失率获得随机数限制，当随机数在限制范围内表示超时，输出true；否则表示非超时，输出false。  (iv)传输开始与结束  收到服务器端的序列号为200的分组，回复205分组，表示准备好传输，等待服务器端的数据。  收到服务器端的序列号为204的分组后，结束分组接收。   1. 设计SR协议   本次实验侧重在GBN的基础上设计SR协议，以下将详细介绍二者的不同之处。   1. 改变ack数组的含义   对于GBN协议，接收方只要维护一个当前收到的序列是否等于waitseq，接收方则只需累积确认；而在SR协议中则复杂许多，不论是接受方还是发送方都需要维护一个队列，以应对乱序到达和乱序接受的情况。  发送方：0 表示还没发，1 表示发了没收到， 2 表示收到了；一开始初始化为 0，当发送之后改为 1，若乱序到达，状态变为 2，按序到达则连同之前连续的状态 2 一起改为 0，实现重复可利用性。  接收方：0表示初始状态， 1 表示待接受状态，2 表示已确认未发送；要把发送窗口内的所有状态都更新成 1，若是按序到达，则把当前设为 0，以便之后循环利用序列号，之后滑窗后移一位（设成1）；若是乱序到达，则设为 2，等到 waitSeq 到来后，再一次性设为 0。   1. 缓存失序分组   SR 协议中，分组可以失序到达被接受，为了保证收到内容的正确性，需要对乱序分组进行缓存，直到按序分组到来后再统一发送。   1. **for** (**int** i = waitSeq; i < waitSeq + SEND\_WIND\_SIZE; i++) 2. { 3. **int** t = (i % SEQ\_SIZE == 0) ? SEQ\_SIZE : i % SEQ\_SIZE; 4. waitSeq = t; 5. // printf("i : %d, Ack[i] : %d\n", i, Ack[i]); 6. **if** (Ack[t] == 2) 7. { 8. Ack[t] = 0; 9. // printf("I'm here!!%d\n", t); 10. k = t + SEND\_WIND\_SIZE; 11. k = (k % SEQ\_SIZE == 0) ? SEQ\_SIZE : k % SEQ\_SIZE; 12. Ack[k] = 1; 13. printf("[receive:success]seq %d will be Cached out now\n", t); 14. // printf("Cache %d write in len %d: %s\n", t, cache\_lengths[t], cache[t]); 15. out.write(cache[t], cache\_lengths[t]); 16. } 17. **else** 18. **break**; 19. } 20. 为每个分组维护一个定时器   SR 协议中需要为每个分组都维护一个定时器，超时即重传。这与 GBN 协议有所差异。   1. **void** click() { 2. **for** (**int** i = 0; i < SEQ\_SIZE; i++) { 3. **if** (counter[i] >= 0) { 4. counter[i] += 1; 5. } 6. } 7. } 9. **int** checkTimeout() { 10. **for** (**int** i = 0; i < SEQ\_SIZE; i++) { 11. **if** (counter[i] >= 6) { 12. **return** i; 13. } 14. } 15. **return** -1; 16. } |
| 实验结果： |
| 本次实验采用的测试数据有两个，其中，server\_in是来自 BBC 的英语新闻，全长 41 KB，client\_in则是带有段落号标识的、内容为“This is client”重复数百次的大小为 26 KB 的数据。这两个数据的特点是，分包数量均超过序列号数量，这就意味着序列号需要循环使用。 一、GBN 协议  首先测试双向-time，可以看到双方都能正常发出指令，并接到来自对方的返回值。  再测试数据传输，这里直接同时进行双向传输。  可以看到两边在同时进行传输，数据传输完成后，两边文件夹下分别出现server\_out.txt与client\_out.txt，经检查，内容正确无误，说明GBN协议设计政权，可以实现传输文件的功能：    二、SR协议  本次实验中，SR花费了大量时间，因此也将更加详细的介绍。  在 SR 中为了调试方便，增加了一些提示信息，首先测试双方互发 -time。    继续测试双向互发文件，设置丢包率和丢ACK概率均为0.2，可以看到区分了send信息和receive信息，这样能够更好的分辨信息。    从输出的提示中可以看到发送窗口与接收窗口的移动过程，以及丢包之后超时重传的过程，结束后双方均收到对方发送的文件内容，经验证，内容正确无误。      在过程中也能看到序列号被循环使用的信息，足以证明循环的设计是正确无误的。    以下给出SR协议的完整代码：  服务端：   1. #include <stdlib.h> 2. #include <time.h> 3. #include <WinSock2.h> 4. #include <fstream> 5. #include <iostream> 6. #include <cmath> 7. #include <process.h> 8. #include <cstdio> 10. #pragma comment(lib,"ws2\_32.lib") 11. #pragma warning(disable:4996) 12. #define SERVER\_PORT 12340 //端口号 13. #define CLIENT\_PORT 12341 //端口号 14. #define SERVER\_IP "0.0.0.0" //IP 地址 15. #define CLIENT\_IP "127.0.0.1" //客户端IP 17. #define DATA\_SIZE 1024 19. **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） 20. **const** **int** SEND\_WIND\_SIZE = 1; 21. **const** **int** SEQ\_SIZE = 20; //序列号的个数，从 0~19 共计 20 个 23. **int** ack[SEQ\_SIZE + 1];//收到 ack 情况，对应 0~19 的 ack, 0 表示还没发，1 表示发了没收到， 2 表示收到了 25. **int** counter[SEQ\_SIZE + 1];//计时器，当为负数时表示未启动，为正数时表示启动；其他同GBN 27. **int** curSeq;//当前数据包的 seq 28. **int** curAck;//当前等待确认的 ack 29. **int** totalSeq;//收到的包的总数 30. **int** totalPacket;//需要发送的包总数 32. unsigned **int** \_\_stdcall ProxyThread(**LPVOID** lpParameter); 34. **void** getCurTime(**char**\* ptime) { 35. SYSTEMTIME sys; 36. GetLocalTime(&sys); 37. sprintf\_s(ptime, 20, "%d-%d-%d %d:%d:%d", sys.wYear, sys.wMonth, sys.wDay, sys.wHour, sys.wMinute, sys.wSecond); 38. } 40. **BOOL** lossInLossRatio(**float** lossRatio) { 41. **int** lossBound = (**int**)(lossRatio \* 100); 42. **int** r = rand() % 101; 43. **if** (r <= lossBound) { 44. **return** TRUE; 45. } 46. **return** FALSE; 47. } 49. **bool** seqIsAvailable() { 50. **int** step = curSeq - curAck; 51. **if** (step < 0) { 52. step += SEQ\_SIZE; 53. } 54. // printf("ack[%d] = %d\n", curSeq, ack[curSeq]); 55. **return** step < SEND\_WIND\_SIZE ? (ack[curSeq] == 0) : **false**; 56. } 58. **void** printTips() 59. { 60. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); 61. printf("|     -time to get current time                |\n"); 62. printf("|     -quit to exit client                     |\n"); 63. printf("|     -testsr [X] [Y] to test the sr           |\n"); 64. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); 65. } 67. **void** ackHandler(**char** c) { 68. unsigned **char** index = (unsigned **char**)c - 1; //序列号减一 69. printf("[send:success]Recv a ack of %d\n", index + 1); 70. **if** (curAck != index) {//分组失序，暂时缓存 71. ack[index] = 2; 72. } 73. **else** { 74. //一次分组到达，窗口向前移动（可能不止一次移动） 75. //printf("index = %d\n", index); 76. ack[index] = 0; 77. curAck = index + 1; 78. curAck %= SEQ\_SIZE; 79. **for** (**int** i = index + 1; i < index + SEND\_WIND\_SIZE; i++) { 80. i %= SEQ\_SIZE; 81. **if** (ack[i] == 2) { 82. counter[i] = -1;//计时器关闭 83. curAck = i + 1;//修改curAck 84. ack[i] = 0; 85. } 86. **else** **break**; 87. } 88. curAck %= SEQ\_SIZE; 89. printf("[send:info]send\_base move to %d\n", curAck); 90. } 91. } 93. **void** click() { 94. **for** (**int** i = 0; i < SEQ\_SIZE; i++) { 95. **if** (counter[i] >= 0) { 96. counter[i] += 1; 97. } 98. } 99. } 101. **int** checkTimeout() { 102. **for** (**int** i = 0; i < SEQ\_SIZE; i++) { 103. **if** (counter[i] >= 6) { 104. **return** i; 105. } 106. } 107. **return** -1; 108. } 110. **struct** ProxyParam { 111. }; 113. //主函数 114. **int** main(**int** argc, **char**\* argv[]) 115. { 116. //加载套接字库（必须） 117. **WORD** wVersionRequested; 118. WSADATA wsaData; 119. //套接字加载时错误提示 120. **int** err; 121. //版本 2.2 122. wVersionRequested = MAKEWORD(2, 2); 123. //加载 dll 文件 Scoket 库 124. err = WSAStartup(wVersionRequested, &wsaData); 125. **if** (err != 0) { 126. //找不到 winsock.dll 127. printf("WSAStartup failed with error: %d\n", err); 128. **return** -1; 129. } 130. **if** (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) 131. { 132. printf("Could not find a usable version of Winsock.dll\n"); 133. WSACleanup(); 134. } 135. **else** { 136. printf("The Winsock 2.2 dll was found okay\n"); 137. } 138. SOCKET sockServer = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP); 139. //设置套接字为非阻塞模式 140. **int** iMode = 1; //1：非阻塞，0：阻塞 141. ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);//非阻塞设置 142. SOCKADDR\_IN addrServer; //服务器地址 143. //addrServer.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER\_IP); 144. addrServer.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);//两者均可 145. addrServer.sin\_family = AF\_INET; 146. addrServer.sin\_port = htons(SERVER\_PORT); 147. err = bind(sockServer, (SOCKADDR\*)&addrServer, **sizeof**(SOCKADDR)); 148. **if** (err) { 149. err = GetLastError(); 150. printf("Could not bind the port %d for socket.Error code is % d\n", SERVER\_PORT, err); 151. WSACleanup(); 152. **return** -1; 153. } 154. SOCKADDR\_IN addrClient; //客户端地址 155. **int** length = **sizeof**(SOCKADDR); 156. **char** buffer[BUFFER\_LENGTH]; //数据发送接收缓冲区 157. ZeroMemory(buffer, **sizeof**(buffer)); 158. //将测试数据读入内存 159. std::ifstream icin; 160. icin.open("server\_in.txt"); 161. **char** data[DATA\_SIZE \* 113];//需要发送的数据 162. ZeroMemory(data, **sizeof**(data)); 163. icin.read(data, DATA\_SIZE \* 113); 164. icin.close(); 165. totalPacket = ceil(strlen(data) / 1024.0); 166. printf("File size is %dB, each packet is 1024B and packet total num is %d\n", strlen(data), totalPacket); 167. **int** recvSize; 168. **for** (**int** i = 0; i < SEQ\_SIZE; ++i) { 169. ack[i] = 0; 170. counter[i] = -1; 171. } 172. **char** cache[SEQ\_SIZE + 1][DATA\_SIZE + 1];//缓存，暂时保存发送但未受到ack的分组 173. **bool** sendFlag = **true**; 174. ProxyParam\* lpProxyParam = **new** ProxyParam; 175. **HANDLE** hThread = (**HANDLE**)\_beginthreadex(NULL, 0, &ProxyThread, (**LPVOID**)lpProxyParam, 0, 0); 176. **while** (**true**) { 177. sendFlag = **true**; 178. //非阻塞接收，若没有收到数据，返回值为-1 179. recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length); 180. **if** (recvSize < 0) { 181. Sleep(200); 182. **continue**; 183. } 184. printf("[send:info]recv from client: %s\n", buffer); 185. **if** (strcmp(buffer, "-time") == 0) { 186. getCurTime(buffer); 187. } 188. **else** **if** (strcmp(buffer, "-quit") == 0) { 189. strcpy\_s(buffer, strlen("Good bye!") + 1, "Good bye!"); 190. } 191. **else** **if** (strcmp(buffer, "-testsr") == 0) { 192. printf("[send:info]The send window size is %d\n", SEND\_WIND\_SIZE); 193. printf("[send:info]The available seq is 1~%d\n", SEQ\_SIZE); 194. ZeroMemory(buffer, **sizeof**(buffer)); 195. **int** recvSize; 196. **int** waitCount = 0; 197. printf("[send:info]Begin to test SR protocol,please don't abort the process\n"); 198. printf("[send:info]Shake hands stage\n"); 199. **int** stage = 0; 200. **bool** runFlag = **true**; 201. **while** (runFlag) { 202. **switch** (stage) { 203. **case** 0://发送 205 阶段 204. buffer[0] = 205; 205. buffer[1] = '\0'; 206. sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, **sizeof**(SOCKADDR)); 207. Sleep(100); 208. stage = 1; 209. **break**; 210. **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 211. recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length); 212. **if** (recvSize < 0) { 213. ++waitCount; 214. **if** (waitCount > 6) { 215. runFlag = **false**; 216. printf("[send:warning]Timeout error\n"); 217. **break**; 218. } 219. Sleep(500); 220. **continue**; 221. } 222. **else** { 223. **if** ((unsigned **char**)buffer[0] == 200) { 224. printf("[send:info]Begin a file transfer\n"); 225. printf("[send:info]File size is %dB, each packet is 1024B and packet total num is %d\n", strlen(data), totalPacket); 226. curSeq = 0; 227. curAck = 0; 228. totalSeq = 0; 229. //waitCount = 0; 230. stage = 2; 231. } 232. } 233. **break**; 234. **case** 2://数据传输阶段 235. **if** (seqIsAvailable()) { 236. **if** (totalSeq <= totalPacket - 1) { 237. //发送给客户端的序列号从 1 开始 238. //printf("curSeq = %d\n", curSeq + 1); 239. buffer[0] = curSeq + 1; 240. ack[curSeq] = 1; 241. memcpy(&buffer[1], data + DATA\_SIZE \* totalSeq, DATA\_SIZE); 242. memcpy(cache[curSeq], data + DATA\_SIZE \* totalSeq, DATA\_SIZE);//缓存分组 243. printf("[send:success]send a packet with a seq of %d\n", curSeq + 1); 244. //printf("buffer = %s\n", buffer); 245. sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, **sizeof**(SOCKADDR)); 246. counter[curSeq] = 0;//计时器开启 247. ++curSeq; 248. curSeq %= SEQ\_SIZE; 249. ++totalSeq; 250. Sleep(500); 251. } 252. } 253. //等待 Ack，若没有收到，则返回值为-1，计数器+1 254. recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length); 255. **if** (recvSize < 0) { 256. click(); 257. //20 次等待 ack 则超时重传 258. **if** (checkTimeout() != -1) { 259. **int** index = checkTimeout(); 260. printf("[send:warning]Seq %d time out.\n", index + 1); 261. buffer[0] = index + 1; 262. memcpy(&buffer[1], cache[index], DATA\_SIZE); 263. printf("[send:success]Resend a packet with a seq of %d\n", index + 1); 264. sendto(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, **sizeof**(SOCKADDR)); 265. counter[index] = 0;//重置计时器 266. } 267. } 268. **else** { 269. //收到 ack 270. ackHandler(buffer[0]); 271. counter[buffer[0] - 1] = -1;//计时器关闭 272. //数判断是否传输完成 273. **if** (totalSeq > totalPacket - 1) {//传输完成，若都收到Ack则传输结束，否则不发送数据等待超时重传 274. //puts("I'm in here！"); 275. **bool** finish = **true**; 276. **for** (**int** i = 0; i < SEQ\_SIZE; i++) { 277. **if** (ack[i] == 1) { 278. finish = **false**; 279. **break**; 280. } 281. } 282. **if** (finish) { 283. printf("\n[send:info]Server send finish!\n"); 284. buffer[0] = 204; 285. buffer[1] = '\0'; 286. sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, **sizeof**(SOCKADDR)); 287. Sleep(100); 288. runFlag = **false**; 289. sendFlag = **false**; 290. **break**; 291. } 292. } 293. } 294. Sleep(500); 295. **break**; 296. } 297. } 298. } 299. **if** (sendFlag) 300. sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, **sizeof**(SOCKADDR)); 301. printTips(); 302. Sleep(500); 303. } 304. //关闭套接字，卸载库 305. closesocket(sockServer); 306. WSACleanup(); 307. **return** 0; 308. } 310. **int** cache\_lengths[SEQ\_SIZE + 1]; 312. unsigned **int** \_\_stdcall ProxyThread(**LPVOID** lpParameter) { 313. //加载套接字库（必须） 314. //加载套接字库（必须） 315. **WORD** wVersionRequested; 316. WSADATA wsaData; 317. //套接字加载时错误提示 318. **int** err; 319. //版本 2.2 320. wVersionRequested = MAKEWORD(2, 2); 321. //加载 dll 文件 Scoket 库 322. err = WSAStartup(wVersionRequested, &wsaData); 323. **if** (err != 0) { 324. //找不到 winsock.dll 325. printf("WSAStartup failed with error: %d\n", err); 326. **return** 1; 327. } 328. **if** (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) 329. { 330. printf("Could not find a usable version of Winsock.dll\n"); 331. WSACleanup(); 332. } 333. **else** { 334. printf("The Winsock 2.2 dll was found okay\n"); 335. } 336. SOCKET socketClient = socket(AF\_INET, SOCK\_DGRAM, 0); 337. SOCKADDR\_IN addrServer; 338. addrServer.sin\_addr.S\_un.S\_addr = inet\_addr(CLIENT\_IP); 339. addrServer.sin\_family = AF\_INET; 340. addrServer.sin\_port = htons(CLIENT\_PORT); 341. //接收缓冲区 342. **char** buffer[BUFFER\_LENGTH]; 343. ZeroMemory(buffer, **sizeof**(buffer)); 344. **int** len = **sizeof**(SOCKADDR); 345. printTips(); 346. **int** ret;//受到数据大小 347. **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 348. **char** cmd[128]; 349. **float** packetLossRatio = 0.2f; //默认包丢失率 0.2 350. **float** ackLossRatio = 0.2f; //默认 ACK 丢失率 0.2 351. //用时间作为随机种子，放在循环的最外面 352. srand((unsigned)time(NULL)); 353. std::ofstream out; 354. **char** cache[SEQ\_SIZE + 1][DATA\_SIZE];//缓存，暂时保存失序但未确认的分组 355. **bool** sendFlag = **true**; 356. **int** Ack[SEQ\_SIZE + 1]; 357. **while** (**true**) { 358. sendFlag = **true**; 359. gets\_s(buffer); 360. //printf("buffer:%s\n", buffer); 361. ret = sscanf\_s(buffer, "%s%f%f", &cmd, **sizeof**(cmd), &packetLossRatio, &ackLossRatio); 362. printf("[info]buffer:%s\n", cmd); 363. **if** (!strcmp(cmd, "-testsr")) { 364. //printf("[receive:info]packet loss ratio:%f\n", packetLossRatio); 365. //printf("[receive:info]ack loss ratio:%f\n", ackLossRatio); 366. printf("[receive:info]The receive window is %d\n", SEND\_WIND\_SIZE); 367. **bool** runFlag = **true**; 368. out.open("server\_out.txt"); 369. printf("%s\n", "[receive:info]Begin to test SR protocol, please don't abort the process"); 370. printf("[receive:info]The loss ratio of packet is %.2f,the loss ratio of Ack is % .2f\n", packetLossRatio, ackLossRatio); 371. **int** waitCount = 0; 372. **int** stage = 0; 373. **BOOL** b; 374. unsigned **short** seq;//包的序列号 375. unsigned **short** recvSeq;//接收窗口大小为 1，已确认的序列号 376. unsigned **short** waitSeq;//等待的序列号 377. sendto(socketClient, "-testsr", strlen("-testsr") + 1, 0, (SOCKADDR\*)&addrServer, **sizeof**(SOCKADDR)); 378. memset(Ack, 0, **sizeof**(Ack)); 380. **while** (runFlag) 381. { 382. //等待 server 回复设置 UDP 为阻塞模式 383. recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, &len); 384. **switch** (stage) { 385. **case** 0://等待握手阶段 386. **if** ((unsigned **char**)buffer[0] == 205) 387. { 388. printf("[receive:info]Ready for file transmission\n"); 389. buffer[0] = 200; 390. buffer[1] = '\0'; 391. sendto(socketClient, buffer, 2, 0, 392. (SOCKADDR\*)&addrServer, **sizeof**(SOCKADDR)); 393. stage = 1; 394. recvSeq = 0; 395. waitSeq = 1; 396. **for** (**int** i = 1; i <= SEND\_WIND\_SIZE; i++) { 397. Ack[i] = 1; 398. } 399. } 400. **break**; 401. **case** 1://等待接收数据阶段 402. **if** ((unsigned **char**)buffer[0] == 204) { 403. printf("\n[receive:info]Receive finished\n"); 404. out.close(); 405. sendFlag = **false**; 406. runFlag = **false**; 407. **break**; 408. } 409. seq = (unsigned **short**)buffer[0]; 410. //随机法模拟包是否丢失 411. b = lossInLossRatio(packetLossRatio); 412. **if** (b) { 413. printf("[receive:warning]The packet with a seq of %d loss\n", seq); 414. **continue**; 415. } 416. printf("[receive:success]recv a packet with a seq of %d\n", seq); 417. **if** (waitSeq == seq) { 418. //当前接收分组直接写入文件 419. out.write(&buffer[1], strlen(&buffer[1])); 420. //printf("buffer %d: %s\n", seq, &buffer[1]); 421. Ack[seq] = 0; 422. waitSeq += 1; 423. waitSeq = (waitSeq % SEQ\_SIZE == 0) ? SEQ\_SIZE : waitSeq % SEQ\_SIZE; 424. **int** k = waitSeq + SEND\_WIND\_SIZE - 1; 425. k = (k % SEQ\_SIZE == 0) ? SEQ\_SIZE : k % SEQ\_SIZE; 426. Ack[k] = 1; 427. //查看是否有失序分组需要写入文件 428. **for** (**int** i = waitSeq; i < waitSeq + SEND\_WIND\_SIZE; i++) { 429. **int** t = (i % SEQ\_SIZE == 0) ? SEQ\_SIZE : i % SEQ\_SIZE; 430. waitSeq = t; 431. //printf("i : %d, Ack[i] : %d\n", i, Ack[i]); 432. **if** (Ack[t] == 2) { 433. Ack[t] = 0; 434. //printf("I'm here!!%d\n", t); 435. k = t + SEND\_WIND\_SIZE; 436. k = (k % SEQ\_SIZE == 0) ? SEQ\_SIZE : k % SEQ\_SIZE; 437. Ack[k] = 1; 438. printf("[receive:success]seq %d will be Cached out now\n", t); 439. //printf("Cache %d write in len %d: %s\n", t, cache\_lengths[t], cache[t]); 440. out.write(cache[t], cache\_lengths[t]); 441. } 442. **else** **break**; 443. } 444. printf("[receive:info]rec\_base move to %d, rightest to %d\n", waitSeq, k); 445. buffer[0] = seq; 446. recvSeq = seq; 447. buffer[1] = '\0'; 448. } 449. **else** **if** ((seq > waitSeq && seq < waitSeq + SEND\_WIND\_SIZE) || (Ack[seq] == 1)) {//分组失序到达 450. cache\_lengths[seq] = strlen(&buffer[1]) + 1; 451. memcpy(cache[seq], &buffer[1], cache\_lengths[seq]);//缓存收到的数据，不修改下一个需要的分组序列号 452. cache[seq][cache\_lengths[seq]] = '\0'; 453. printf("[receive:info]cache %d len %d buffer len %d\n", seq, strlen(cache[seq]), strlen(&buffer[1]) + 1); 454. buffer[0] = seq; 455. buffer[1] = '\0'; 456. Ack[seq] = 2; 457. printf("[receive:warning]seq %d receive, but not in order, Ack[seq] = %d\n", seq, Ack[seq]); 458. } 459. **else** { 460. buffer[0] = seq; 461. buffer[1] = '\0'; 462. } 463. b = lossInLossRatio(ackLossRatio); 464. **if** (b) { 465. printf("[receive:warning]The Ack of %d loss\n", (unsigned **char**)buffer[0]); 466. **continue**; 467. } 468. sendto(socketClient, buffer, 2, 0, (SOCKADDR\*)&addrServer, **sizeof**(SOCKADDR)); 469. printf("[receive:info]send a Ack of %d\n", (unsigned **char**)buffer[0]); 470. **break**; 471. } 472. Sleep(500); 473. } 474. } 475. **if** (sendFlag) { 476. sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrServer, **sizeof**(SOCKADDR)); 477. ret = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, &len); 478. printf("%s\n", buffer); 479. **if** (!strcmp(buffer, "Good bye!")) { 480. **break**; 481. } 482. } 483. printTips(); 484. } 485. closesocket(socketClient); 486. WSACleanup(); 487. **return** 0; 488. }   客户端：   1. #include <stdlib.h> 2. #include <WinSock2.h> 3. #include <time.h> 4. #include <stdio.h> 5. #include <fstream> 6. #include <iostream> 7. #include <process.h> 8. #include <cstdio> 10. #pragma warning(disable:4996) 11. #pragma comment(lib,"ws2\_32.lib") 12. #define SERVER\_PORT 12341 //端口号 13. #define CLIENT\_PORT 12340 //端口号 14. #define SERVER\_IP "127.0.0.1" //IP 地址 15. #define CLIENT\_IP "127.0.0.1" //客户端IP 16. **const** **int** BUFFER\_LENGTH = 1026; 17. **const** **int** SEND\_WIND\_SIZE = 1; 18. **const** **int** SEQ\_SIZE = 20;//接收端序列号个数，为 1~20 20. #define DATA\_SIZE 1024 22. **int** ack[SEQ\_SIZE + 1]; //收到 ack 情况，对应 0~19 的 ack, 0 表示还没发，1 表示发了没收到， 2 表示收到了 24. **int** cache\_lengths[SEQ\_SIZE + 1]; 25. **int** counter[SEQ\_SIZE + 1];//计时器，当为负数时表示未启动，为正数时表示启动；其他同GBN 26. **int** curSeq;//当前数据包的 seq 27. **int** curAck;//当前等待确认的 ack 28. **int** totalSeq;//收到的包的总数 29. **int** totalPacket;//需要发送的包总数 31. unsigned **int** \_\_stdcall ProxyThread(**LPVOID** lpParameter);  34. **void** getCurTime(**char**\* ptime) { 35. SYSTEMTIME sys; 36. GetLocalTime(&sys); 37. sprintf\_s(ptime, 20, "%d-%d-%d %d:%d:%d", sys.wYear, sys.wMonth, sys.wDay, sys.wHour, sys.wMinute, sys.wSecond); 38. } 40. **BOOL** lossInLossRatio(**float** lossRatio) { 41. **int** lossBound = (**int**)(lossRatio \* 100); 42. **int** r = rand() % 101; 43. **if** (r <= lossBound) { 44. **return** TRUE; 45. } 46. **return** FALSE; 47. } 49. **bool** seqIsAvailable() { 50. **int** step = curSeq - curAck; 51. **if** (step < 0) { 52. step += SEQ\_SIZE; 53. } 54. //printf("ack[%d] = %d\n", curSeq, ack[curSeq]); 55. **return** step < SEND\_WIND\_SIZE ? (ack[curSeq] == 0) : **false**; 56. } 58. **void** printTips() 59. { 60. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); 61. printf("|     -time to get current time                |\n"); 62. printf("|     -quit to exit client                     |\n"); 63. printf("|     -testsr [X] [Y] to test the sr           |\n"); 64. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); 65. } 67. **void** ackHandler(**char** c) { 68. unsigned **char** index = (unsigned **char**)c - 1; //序列号减一 69. printf("[send:success]Recv a ack of %d\n", index + 1); 70. **if** (curAck != index) {//分组失序，暂时缓存 71. ack[index] = 2; 72. } 73. **else** { 74. //一次分组到达，窗口向前移动（可能不止一次移动） 75. //printf("index = %d\n", index); 76. ack[index] = 0; 77. curAck = index + 1; 78. curAck %= SEQ\_SIZE; 79. **for** (**int** i = index + 1; i < index + SEND\_WIND\_SIZE; i++) { 80. i %= SEQ\_SIZE; 81. **if** (ack[i] == 2) { 82. counter[i] = -1;//计时器关闭 83. curAck = i + 1;//修改curAck 84. ack[i] = 0; 85. } 86. **else** **break**; 87. } 88. curAck %= SEQ\_SIZE; 89. printf("[send:info]send\_base move to %d\n", curAck); 90. } 91. } 93. **void** click() { 94. **for** (**int** i = 0; i < SEQ\_SIZE; i++) { 95. **if** (counter[i] >= 0) { 96. counter[i] += 1; 97. } 98. } 99. } 101. **int** checkTimeout() { 102. **for** (**int** i = 0; i < SEQ\_SIZE; i++) { 103. **if** (counter[i] >= 6) { 104. **return** i; 105. } 106. } 107. **return** -1; 108. } 110. **struct** ProxyParam { 111. }; 113. **int** main(**int** argc, **char**\* argv[]) 114. { 115. //加载套接字库（必须） 116. **WORD** wVersionRequested; 117. WSADATA wsaData; 118. //套接字加载时错误提示 119. **int** err; 120. //版本 2.2 121. wVersionRequested = MAKEWORD(2, 2); 122. //加载 dll 文件 Scoket 库 123. err = WSAStartup(wVersionRequested, &wsaData); 124. **if** (err != 0) { 125. //找不到 winsock.dll 126. printf("WSAStartup failed with error: %d\n", err); 127. **return** -1; 128. } 129. **if** (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) 130. { 131. printf("Could not find a usable version of Winsock.dll\n"); 132. WSACleanup(); 133. } 134. **else** { 135. printf("The Winsock 2.2 dll was found okay\n"); 136. } 137. SOCKET sockServer = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP); 138. //设置套接字为非阻塞模式 139. **int** iMode = 1; //1：非阻塞，0：阻塞 140. ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);//非阻塞设置 141. SOCKADDR\_IN addrServer; //服务器地址 142. //addrServer.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER\_IP); 143. addrServer.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);//两者均可 144. addrServer.sin\_family = AF\_INET; 145. addrServer.sin\_port = htons(SERVER\_PORT); 146. err = bind(sockServer, (SOCKADDR\*)&addrServer, **sizeof**(SOCKADDR)); 147. **if** (err) { 148. err = GetLastError(); 149. printf("Could not bind the port %d for socket.Error code is % d\n", SERVER\_PORT, err); 150. WSACleanup(); 151. **return** -1; 152. } 153. SOCKADDR\_IN addrClient; //客户端地址 154. **int** length = **sizeof**(SOCKADDR); 155. **char** buffer[BUFFER\_LENGTH]; //数据发送接收缓冲区 156. ZeroMemory(buffer, **sizeof**(buffer)); 157. //将测试数据读入内存 158. std::ifstream icin; 159. icin.open("client\_in.txt"); 160. **char** data[DATA\_SIZE \* 113];//需要发送的数据 161. ZeroMemory(data, **sizeof**(data)); 162. icin.read(data, DATA\_SIZE \* 113); 163. icin.close(); 164. totalPacket = ceil(strlen(data) / 1024.0); 165. printf("File size is %dB, each packet is 1024B and packet total num is %d\n", strlen(data), totalPacket); 166. **int** recvSize; 168. **char** cache[SEQ\_SIZE + 1][DATA\_SIZE + 1];//缓存，暂时保存发送但未受到ack的分组 169. **bool** sendFlag = **true**; 170. ProxyParam\* lpProxyParam = **new** ProxyParam; 171. **HANDLE** hThread = (**HANDLE**)\_beginthreadex(NULL, 0, &ProxyThread, (**LPVOID**)lpProxyParam, 0, 0); 172. **while** (**true**) { 173. **for** (**int** i = 0; i < SEQ\_SIZE; ++i) { 174. ack[i] = 0; 175. counter[i] = -1; 176. } 177. sendFlag = **true**; 178. //非阻塞接收，若没有收到数据，返回值为-1 179. recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length); 180. **if** (recvSize < 0) { 181. Sleep(200); 182. **continue**; 183. } 184. printf("[send:info]recv from client: %s\n", buffer); 185. **if** (strcmp(buffer, "-time") == 0) { 186. getCurTime(buffer); 187. } 188. **else** **if** (strcmp(buffer, "-quit") == 0) { 189. strcpy\_s(buffer, strlen("Good bye!") + 1, "Good bye!"); 190. } 191. **else** **if** (strcmp(buffer, "-testsr") == 0) { 192. printf("[send:info]The send window size is %d\n", SEND\_WIND\_SIZE); 193. printf("[send:info]The available seq is 1~%d\n", SEQ\_SIZE); 194. ZeroMemory(buffer, **sizeof**(buffer)); 195. **int** recvSize; 196. **int** waitCount = 0; 197. printf("[send:info]Begin to test SR protocol,please don't abort the process\n"); 198. printf("[send:info]Shake hands stage\n"); 199. **int** stage = 0; 200. **bool** runFlag = **true**; 201. **while** (runFlag) { 202. **switch** (stage) { 203. **case** 0://发送 205 阶段 204. buffer[0] = 205; 205. buffer[1] = '\0'; 206. sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, **sizeof**(SOCKADDR)); 207. Sleep(100); 208. stage = 1; 209. **break**; 210. **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 211. recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length); 212. **if** (recvSize < 0) { 213. ++waitCount; 214. **if** (waitCount > 6) { 215. runFlag = **false**; 216. printf("[warning]Timeout error\n"); 217. **break**; 218. } 219. Sleep(500); 220. **continue**; 221. } 222. **else** { 223. **if** ((unsigned **char**)buffer[0] == 200) { 224. printf("[send:info]Begin a file transfer\n"); 225. printf("[send:info]File size is %dB, each packet is 1024B and packet total num is %d\n", strlen(data), totalPacket); 226. curSeq = 0; 227. curAck = 0; 228. totalSeq = 0; 229. stage = 2; 230. } 231. } 232. **break**; 233. **case** 2://数据传输阶段 234. **if** (seqIsAvailable()) { 235. **if** (totalSeq <= totalPacket - 1) { 236. //发送给客户端的序列号从 1 开始 237. //printf("curSeq = %d\n", curSeq + 1); 238. buffer[0] = curSeq + 1; 239. ack[curSeq] = 1; 240. memcpy(&buffer[1], data + DATA\_SIZE \* totalSeq, DATA\_SIZE); 241. memcpy(cache[curSeq], data + DATA\_SIZE \* totalSeq, DATA\_SIZE);//缓存分组 242. printf("[send:success]send a packet with a seq of %d\n", curSeq + 1); 243. //printf("buffer = %s\n", buffer); 244. sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, **sizeof**(SOCKADDR)); 245. counter[curSeq] = 0;//计时器开启 246. ++curSeq; 247. curSeq %= SEQ\_SIZE; 248. ++totalSeq; 249. Sleep(500); 250. } 251. } 252. //等待 Ack，若没有收到，则返回值为-1，计数器+1 253. recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length); 254. **if** (recvSize < 0) { 255. click(); 256. //20 次等待 ack 则超时重传 257. **if** (checkTimeout() != -1) { 258. **int** index = checkTimeout(); 259. printf("[send:warning]Seq %d time out.\n", index + 1); 260. buffer[0] = index + 1; 261. memcpy(&buffer[1], cache[index], DATA\_SIZE); 262. printf("[send:success]Resend a packet with a seq of %d\n", index + 1); 263. sendto(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, **sizeof**(SOCKADDR)); 264. counter[index] = 0;//重置计时器 265. } 266. } 267. **else** { 268. //收到 ack 269. ackHandler(buffer[0]); 270. counter[buffer[0] - 1] = -1;//计时器关闭 271. //数判断是否传输完成 272. **if** (totalSeq > totalPacket - 1) {//传输完成，若都收到Ack则传输结束，否则不发送数据等待超时重传 273. //puts("I'm in here！"); 274. **bool** finish = **true**; 275. **for** (**int** i = 0; i < SEQ\_SIZE; i++) { 276. **if** (ack[i] == 1) { 277. finish = **false**; 278. **break**; 279. } 280. } 281. **if** (finish) { 282. printf("\n[send:info]Server send finish!\n"); 283. buffer[0] = 204; 284. buffer[1] = '\0'; 285. sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, **sizeof**(SOCKADDR)); 286. Sleep(100); 287. runFlag = **false**; 288. sendFlag = **false**; 289. **break**; 290. } 291. } 292. } 293. Sleep(500); 294. **break**; 295. } 296. } 297. } 298. **if** (sendFlag) 299. sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, **sizeof**(SOCKADDR)); 300. printTips(); 301. Sleep(500); 302. } 303. //关闭套接字，卸载库 304. closesocket(sockServer); 305. WSACleanup(); 306. **return** 0; 307. }  310. unsigned **int** \_\_stdcall ProxyThread(**LPVOID** lpParameter) 311. { 312. //加载套接字库（必须） 313. **WORD** wVersionRequested; 314. WSADATA wsaData; 315. //套接字加载时错误提示 316. **int** err; 317. //版本 2.2 318. wVersionRequested = MAKEWORD(2, 2); 319. //加载 dll 文件 Scoket 库 320. err = WSAStartup(wVersionRequested, &wsaData); 321. **if** (err != 0) { 322. //找不到 winsock.dll 323. printf("WSAStartup failed with error: %d\n", err); 324. **return** 1; 325. } 326. **if** (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) 327. { 328. printf("Could not find a usable version of Winsock.dll\n"); 329. WSACleanup(); 330. } 331. **else** { 332. printf("The Winsock 2.2 dll was found okay\n"); 333. } 334. SOCKET socketClient = socket(AF\_INET, SOCK\_DGRAM, 0); 335. SOCKADDR\_IN addrServer; 336. addrServer.sin\_addr.S\_un.S\_addr = inet\_addr(CLIENT\_IP); 337. addrServer.sin\_family = AF\_INET; 338. addrServer.sin\_port = htons(CLIENT\_PORT); 339. //接收缓冲区 340. **char** buffer[BUFFER\_LENGTH]; 341. ZeroMemory(buffer, **sizeof**(buffer)); 342. **int** len = **sizeof**(SOCKADDR); 343. printTips(); 344. **int** ret;//受到数据大小 345. **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 346. **char** cmd[128]; 347. **float** packetLossRatio = 0.2f; //默认包丢失率 0.2 348. **float** ackLossRatio = 0.2f; //默认 ACK 丢失率 0.2 349. //用时间作为随机种子，放在循环的最外面 350. srand((unsigned)time(NULL)); 351. std::ofstream out; 352. **char** cache[SEQ\_SIZE + 1][DATA\_SIZE];//缓存，暂时保存失序但未确认的分组 353. **bool** sendFlag = **true**; 354. **int** Ack[SEQ\_SIZE + 1]; 355. **while** (**true**) { 356. sendFlag = **true**; 357. gets\_s(buffer); 358. //printf("buffer:%s\n", buffer); 359. ret = sscanf\_s(buffer, "%s%f%f", &cmd, **sizeof**(cmd), &packetLossRatio, &ackLossRatio); 360. printf("[info]buffer:%s\n", cmd); 361. //printf("packet:%f\n", packetLossRatio); 362. //printf("Ack:%f\n", ackLossRatio); 363. **if** (!strcmp(cmd, "-testsr")) { 364. printf("[receive:info]The receive window is %d\n", SEND\_WIND\_SIZE); 365. **bool** runFlag = **true**; 366. out.open("client\_out.txt"); 367. printf("%s\n", "[receive:info]Begin to test SR protocol, please don't abort the process"); 368. printf("[receive:info]The loss ratio of packet is %.2f,the loss ratio of Ack is % .2f\n", packetLossRatio, ackLossRatio); 369. **int** waitCount = 0; 370. **int** stage = 0; 371. **BOOL** b; 372. unsigned **short** seq;//包的序列号 373. unsigned **short** recvSeq;//接收窗口大小为 1，已确认的序列号 374. unsigned **short** waitSeq;//等待的序列号 375. sendto(socketClient, "-testsr", strlen("-testsr") + 1, 0, (SOCKADDR\*)&addrServer, **sizeof**(SOCKADDR)); 376. memset(Ack, 0, **sizeof**(Ack)); 378. **while** (runFlag) 379. { 380. //等待 server 回复设置 UDP 为阻塞模式 381. recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, &len); 382. **switch** (stage) { 383. **case** 0://等待握手阶段 384. **if** ((unsigned **char**)buffer[0] == 205) 385. { 386. printf("[receive:info]Ready for file transmission\n"); 387. buffer[0] = 200; 388. buffer[1] = '\0'; 389. sendto(socketClient, buffer, 2, 0, 390. (SOCKADDR\*)&addrServer, **sizeof**(SOCKADDR)); 391. stage = 1; 392. recvSeq = 0; 393. waitSeq = 1; 394. **for** (**int** i = 1; i <= SEND\_WIND\_SIZE; i++) { 395. Ack[i] = 1; 396. } 397. } 398. **break**; 399. **case** 1://等待接收数据阶段 400. **if** ((unsigned **char**)buffer[0] == 204) { 401. printf("\n[receive:info]Receive finished\n"); 402. out.close(); 403. sendFlag = **false**; 404. runFlag = **false**; 405. **break**; 406. } 407. seq = (unsigned **short**)buffer[0]; 408. //随机法模拟包是否丢失 409. b = lossInLossRatio(packetLossRatio); 410. **if** (b) { 411. printf("[receive:warning]The packet with a seq of %d loss\n", seq); 412. **continue**; 413. } 414. printf("[receive:success]recv a packet with a seq of %d\n", seq); 415. **if** (waitSeq == seq) { 416. //当前接收分组直接写入文件 417. out.write(&buffer[1], strlen(&buffer[1])); 418. //printf("buffer %d: %s\n", seq, &buffer[1]); 419. Ack[seq] = 0; 420. waitSeq += 1; 421. waitSeq = (waitSeq % SEQ\_SIZE == 0) ? SEQ\_SIZE : waitSeq % SEQ\_SIZE; 422. **int** k = waitSeq + SEND\_WIND\_SIZE - 1; 423. k = (k % SEQ\_SIZE == 0) ? SEQ\_SIZE : k % SEQ\_SIZE; 424. Ack[k] = 1; 425. //查看是否有失序分组需要写入文件 426. **for** (**int** i = waitSeq; i < waitSeq + SEND\_WIND\_SIZE; i++) { 427. **int** t = (i % SEQ\_SIZE == 0) ? SEQ\_SIZE : i % SEQ\_SIZE; 428. waitSeq = t; 429. //printf("i : %d, Ack[i] : %d\n", i, Ack[i]); 430. **if** (Ack[t] == 2) { 431. Ack[t] = 0; 432. //printf("[receive:info]I'm here!!%d\n", t); 433. k = t + SEND\_WIND\_SIZE; 434. k = (k % SEQ\_SIZE == 0) ? SEQ\_SIZE : k % SEQ\_SIZE; 435. Ack[k] = 1; 436. printf("[receive:success]seq %d will be Cached out now\n", t); 437. //printf("Cache %d write in len %d: %s\n", t, cache\_lengths[t], cache[t]); 438. out.write(cache[t], cache\_lengths[t]); 439. } 440. **else** **break**; 441. } 442. printf("[receive:info]rec\_base move to %d, rightest to %d\n", waitSeq, k); 443. buffer[0] = seq; 444. recvSeq = seq; 445. buffer[1] = '\0'; 446. } 447. **else** **if** ((seq > waitSeq && seq < waitSeq + SEND\_WIND\_SIZE) || (Ack[seq] == 1)) {//分组失序到达 448. cache\_lengths[seq] = strlen(&buffer[1]) + 1; 449. memcpy(cache[seq], &buffer[1], cache\_lengths[seq]);//缓存收到的数据，不修改下一个需要的分组序列号 450. cache[seq][cache\_lengths[seq]] = '\0'; 451. printf("[receive:info]cache %d len %d buffer len %d\n", seq, strlen(cache[seq]), strlen(&buffer[1]) + 1); 452. buffer[0] = seq; 453. buffer[1] = '\0'; 454. Ack[seq] = 2; 455. printf("[receive:warning]seq %d receive, but not in order, Ack[seq] = %d\n", seq, Ack[seq]); 456. } 457. **else** { 458. buffer[0] = seq; 459. buffer[1] = '\0'; 460. } 461. b = lossInLossRatio(ackLossRatio); 462. **if** (b) { 463. printf("[receive:warning]The Ack of %d loss\n", (unsigned **char**)buffer[0]); 464. **continue**; 465. } 466. sendto(socketClient, buffer, 2, 0, (SOCKADDR\*)&addrServer, **sizeof**(SOCKADDR)); 467. printf("[receive:success]send a Ack of %d\n", (unsigned **char**)buffer[0]); 468. **break**; 469. } 470. Sleep(500); 471. } 472. } 473. **if** (sendFlag) { 474. sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrServer, **sizeof**(SOCKADDR)); 475. ret = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, &len); 476. printf("%s\n", buffer); 477. **if** (!strcmp(buffer, "Good bye!")) { 478. **break**; 479. } 480. } 481. printTips(); 482. } 483. closesocket(socketClient); 484. WSACleanup(); 485. **return** 0; 486. } |
| 问题讨论： |
| 以下是该实验的流程图（具体图片见文件）：  SR 发送端：    SR接收端： |
| 心得体会： |
| 经过这次实验，我对GBN协议、SR协议的原理与具体实现过程有了较深的了解，特别是修复bug的过程中，对于很多细节的理解更加到位，比如SR协议中接收窗口的移动方式、循环使用序列号的实现方式、超时重传的实现方式等等，总的来说，虽然花费了相当大量的时间，但总体还是收获颇丰的。 |