|  |
| --- |
| 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： |
| **停等协议的设计与实现：**  （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；  **GBN协议的设计与实现：**  （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；  （4）将所设计的 GBN 协议改进为 SR 协议。 |
| 实验过程： |
| 以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。  **一、停等协议的设计与实现**  **1. 数据分组格式、确认分组格式及各个域作用**  （1）数据分组格式   |  |  | | --- | --- | | Seq | Data |   Seq：发送数据的序列号  Data：传输的数据  （2）确认分组格式   |  | | --- | | ACK |   ACK：确认已经收到的序列号  **2. 协议两端程序流程图**    **3. 协议典型交互过程**  发送方封装数据分组，分配唯一序列号Seq，并启动一个计时器，用于监测接收方的确认响应，在发送完一个数据分组后会暂停发送，等待接收方的确认分组ACK。若发送方在计时器超时前收到接收方返回的与发送数据分组Seq相同的ACK确认，则认为该数据帧已成功传输，随即清零计时器并准备发送下一个数据分组；若在计时器超时后仍未收到ACK确认，则判断该数据分组传输失败，需要重新发送。  接收方收到发送方发送的数据分组后，会对其进行校验，并判断接收序列号是否为期望序列号，以判断数据分组是否完整且无误。若校验无误，则向发送方返回一个与该分组序号相同的ACK确认，表示已成功接收；若校验发现错误，则向发送方返回一个上一次确认序列号的ACK确认。  **4. 数据分组丢失验证模拟方法**  在发送/接收数据时，调用函数 lossInLossRatio 进行模拟，将丢包概率乘100转换为一个整数 lossBound（0~100），然后使用 rand() 函数生成随机数，若对100求余后小于lossBound，则执行丢包，否则正常进行。    **5. 程序实现的主要类（或函数）及其主要作用**  （1）seqIsAvailable 函数：判断当前序列号是否在发送窗口内且对应的 ACK 未收到，如果序列号在窗口内且 ACK 已收到则返回TRUE，否则返回FALSE。  （2）lossInLossRatio 函数：根据给定的丢包率 lossRatio，生成随机数字，以判断是否执行模拟丢包。  （3）timeoutHandler函数：处理超时情况，当发送方在超时后未收到 ACK，将上一次成功收到的 ACK 标记为已收到，更新相关序列号和数据分组计数变量，实现回退重传。  （4）ackHandler函数：处理接收到的 ACK，根据 ACK 中的序列号，更新已收到 ACK 的状态数组 ack，并正确移动当前等待确认的 ACK 指针 curAck。  **二、GBN协议的设计与实现**  **1. 数据分组格式、确认分组格式及各个域作用**  （1）数据分组格式   |  |  | | --- | --- | | Seq | Data |   Seq：发送数据的序列号（1-20）  Data：传输的数据  （2）确认分组格式   |  | | --- | | ACK |   ACK：确认已经收到的序列号  **2. 协议两端程序流程图**    **3. 协议典型交互过程**  发送方初始化一个长度为N的发送窗口，连续发送窗口内的数据分组，并为窗口中的第一个分组设置一个定时器，之后等待接收方的确认 ACK。若在超时前未收到接收方的 ACK，则会重传当前窗口中所有已发送但未被确认的数据分组，并重新启动定时器。  接收方收到发送方发送的数据分组后，会对其进行校验，并判断接收序列号是否为期望序列号，以判断数据分组是否完整且无误。若校验无误，则向发送方返回一个与该分组序号相同的ACK确认，表示已成功接收；若校验发现错误，则向发送方返回一个上一次确认序列号的ACK确认。  **4. 数据分组丢失验证模拟方法**  在发送/接收数据时，调用函数 lossInLossRatio 进行模拟，将丢包概率乘100转换为一个整数 lossBound（0~100），然后使用 rand() 函数生成随机数，若对100求余后小于lossBound，则执行丢包，否则正常进行。    5. 程序实现的主要类（或函数）及其主要作用  （1）seqIsAvailable 函数：判断当前序列号是否在发送窗口内且对应的 ACK 未收到，如果序列号在窗口内且 ACK 已收到则返回TRUE，否则返回FALSE。  （2）lossInLossRatio 函数：根据给定的丢包率 lossRatio，生成随机数字，以判断是否执行模拟丢包。  （3）timeoutHandler函数：处理超时情况，当发送方在超时后未收到 ACK，将上一次成功收到的 ACK 标记为已收到，更新相关序列号和数据分组计数变量，实现回退重传。  （4）ackHandler函数：处理接收到的 ACK，根据 ACK 中的序列号，更新已收到 ACK 的状态数组 ack，并正确移动当前等待确认的 ACK 指针 curAck。 |
| 实验结果： |
| ·-time指令请求当前时间，用于测试客户端与服务器的连接，运行结果：    ·-quit指令用于退出客户端，运行结果：    **一、停等协议双向数据传输**  1.client.txt 上传结果：    2.server.txt 下载结果：    **二、GBN协议双向数据传输（窗口大小为10）**  1.client.txt 上传结果：    2.server.txt 下载结果：    **三、SR协议双向数据传输（窗口大小为10）**  1.client.txt 上传结果：    2.server.txt 下载结果： |
| 问题讨论： |
| **1.实验结果分析**  完成了实验内容的全部任务，设计了基于UDP协议的停-等协议、GBN协议以及改进的SR协议，可以实现单向可靠数据传输和双向数据传输（一个 C/S 文件传输应用，支持上传和下载），并模拟引入数据包的丢失，以验证协议的有效性。  通过模拟引入数据丢包情况，发现在高丢包率环境下， GBN协议比停-等协议的效率更高，延迟更低；而SR协议避免了GBN协议在高丢包率环境下的恶性循环问题。  UDP 协议虽然实现简单，但需要设计额外的协议机制来实现可靠数据传输。  **2.UDP 编程的主要特点**  （1）不保证数据传输的可靠性，不确认数据包是否送达，也不提供丢包的重传机制；  （2）多路复用/分用，简单的错误校验，不保证延迟和带宽；  （2）是一种无连接的协议，在数据传输之前不需要先建立连接；  （3）没有拥塞控制机制，因此网络上的拥塞不会使源主机的发送速率降低；  （4）支持一对多、多对一等交互通信。  **3. 超时等待时间设置**  判定超时的时间长短应合适，过长会导致传输速度过慢，过短则会导致没来得及传输的数据包和确认被误认为丢包，重传导致浪费时间。 |
| 心得体会： |
| （1）逐步实现了停-等协议、GBN协议和SR协议，掌握了它们的工作原理，学会了C/S 文件传输应用中上传、下载等功能的实现；  （2）掌握了可靠数据传输和滑动窗口协议的基本原理，对传输层协议有了更深的理解；  （3）提高了Socket编程水平，增强了问题分析和解决能力，对相关网络协议设计有了浓厚兴趣。 |

**程序源码：**

1.客户端client：

1. #include <stdlib.h>
2. #include <WinSock2.h>
3. #include <time.h>
4. #include <stdio.h>
5. #include<fstream>
6. #include<sstream>
7. #include<cstdio>
8. #pragma comment(lib,"ws2\_32.lib")
9. #pragma warning(disable:4996)
10. #define SERVER\_PORT 12340  *// 接收数据的端口号*
11. #define SERVER\_IP "127.0.0.1" *// 服务器的 IP 地址*
12. using namespace std;
13. const int BUFFER\_LENGTH = 1027;
14. const int SEQ\_SIZE = 20; *// 接收端序列号个数，为 1~20*
15. BOOL ack[SEQ\_SIZE]; *// 收到 ack 情况，对应 0~19 的 ack*
16. int curSeq; *// 当前数据包的 seq*
17. int curAck; *// 当前等待确认的 ack*
18. int totalSeq; *// 收到的包的总数*
19. int totalPacket;*// 需要发送的包总数*
20. int waitSeq;
21. const int SEND\_WIND\_SIZE = 10;
22. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*
23. */\* -time 从服务器端获取当前时间*
24. -quit 退出客户端
25. -testgbn [X] [Y] op filename 测试 GBN 协议实现可靠数据传输
26. -testsr [X] [Y] op filename 测试 SR 协议实现可靠数据传输
27. [X] [0,1] 模拟数据包丢失的概率
28. [Y] [0,1] 模拟 ACK 丢失的概率
29. \*/
30. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*
31. void printTips() {
32. printf("| -time to get current time       |\n");
33. printf("| -quit to exit client        |\n");
34. printf("| -testgbn [X] [Y] op filename to test GBN protocol |\n");
35. printf("| -testsr [X] [Y] op filename to test SR protocol   |\n");
36. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
37. }
38. void getTime(int flg) {
39. time\_t timep;
40. struct tm\* p;
41. char cur\_time[28];
42. time(&timep);
43. p = gmtime(&timep);
44. sprintf(cur\_time, "%d.%d.%d %d:%d:%d", 1900 + p->tm\_year, 1 + p->tm\_mon, p->tm\_mday, 8 + p->tm\_hour, p->tm\_min, p->tm\_sec);
45. if (flg == 1) {
46. printf("数据包发送时间：%s", cur\_time);
47. }
48. else if (flg == 2) {
49. printf("ACK接收时间：%s", cur\_time);
50. }
51. }
52. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
53. *// Method: lossInLossRatio*
54. *// FullName: lossInLossRatio*
55. *// Access: public*
56. *// Returns: BOOL*
57. *// 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*
58. *// Parameter: float lossRatio [0,1]*
59. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
60. BOOL lossInLossRatio(float lossRatio) {
61. int lossBound = (int)(lossRatio \* 100);
62. int r = rand() % 100;
63. if (r < lossBound) {
64. return TRUE;
65. }
66. return FALSE;
67. }
68. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
69. *// Method: seqIsAvailable*
70. *// Access: public*
71. *// Returns: BOOL*
72. *// Qualifier: 判断序列号是否在窗口中，返回TRUE则说明在其中*
73. *// Parameter: nothing*
74. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
75. bool seqIsAvailable() {
76. int step;
77. step = curSeq - curAck;
78. step = step >= 0 ? step : step + SEQ\_SIZE;
79. *// 序列号是否在当前发送窗口之内*
80. if (step >= SEND\_WIND\_SIZE) {
81. return false;
82. }
83. if (ack[curSeq]) {
84. return true;
85. }
86. return false;
87. }
88. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
89. *// Method: timeoutHandler*
90. *// Access: public*
91. *// Returns: void*
92. *// Qualifier: 超时处理（seq回退重传）*
93. *// Parameter: nothing*
94. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
95. void timeoutHandler() {
96. printf("Timer out error.\n");
97. int index;
98. for (int i = 0; i < (curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE; ++i) {
99. index = (i + curAck) % SEQ\_SIZE;
100. ack[index] = TRUE;
101. }
102. totalSeq -= ((curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE);
103. curSeq = curAck;
104. }
105. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
106. *// Method: ackHandler*
107. *// Access: public*
108. *// Returns: void*
109. *// Qualifier: 接收到ack并处理*
110. *// Parameter: char c*
111. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
112. void ackHandler(char c) {
113. unsigned char index = (unsigned char)c - 1; *// 序列号减一*
114. printf("Recv a ack of %d\n", index);
115. getTime(2);
116. printf("  序列号：%d\n", index);
117. if (curAck <= index) {
118. for (int i = curAck; i <= index; ++i) {
119. ack[i] = TRUE;
120. }
121. curAck = (index + 1) % SEQ\_SIZE;
122. }
123. else {
124. *// ack 超过了最大值，回到了 curAck 的左边*
125. for (int i = curAck; i < SEQ\_SIZE; ++i) {
126. ack[i] = TRUE;
127. }
128. for (int i = 0; i <= index; ++i) {
129. ack[i] = TRUE;
130. }
131. curAck = index + 1;
132. }
133. }
134. int main()
135. {
136. *// 加载套接字库（必须）*
137. WORD wVersionRequested;
138. WSADATA wsaData;
139. *// 套接字加载时错误提示*
140. int err;
141. *// 版本 2.2*
142. wVersionRequested = MAKEWORD(2, 2);
143. *// 加载 dll 文件 Scoket 库*
144. err = WSAStartup(wVersionRequested, &wsaData);
145. if (err != 0) {
146. *// 找不到 winsock.dll*
147. printf("WSAStartup failed with error: %d\n", err);
148. return 1;
149. }
150. if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) {
151. printf("Could not find a usable version of Winsock.dll\n");
152. WSACleanup();
153. }
154. else {
155. printf("The Winsock 2.2 dll was found okay\n");
156. }
157. SOCKET socketClient = socket(AF\_INET, SOCK\_DGRAM, 0);
158. SOCKADDR\_IN addrServer;
159. addrServer.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER\_IP);
160. addrServer.sin\_family = AF\_INET;
161. addrServer.sin\_port = htons(SERVER\_PORT);
162. *// 接收缓冲区*
163. char buffer[BUFFER\_LENGTH];
164. ZeroMemory(buffer, sizeof(buffer));
165. int len = sizeof(SOCKADDR);
166. *// 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*
167. *// 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*
168. int ret;
169. 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*
170. char cmd[128];
171. float packetLossRatio = 0.2; *// 默认包丢失率 0.2*
172. float ackLossRatio = 0.2;  *// 默认 ACK 丢失率 0.2*
173. char operation[10];
174. char filename[100];
175. int sendack = 0;
176. int iMode = 0;
177. int loct = 0;
178. int waitCount = 0;
179. srand((unsigned)time(NULL));
180. while (true) {
181. printTips();
182. gets\_s(buffer);
183. ret = sscanf(buffer, "%s %f %f %s %s", &cmd, &packetLossRatio, &ackLossRatio, &operation, &filename);
184. if (!strcmp(cmd, "-testsr")) {
185. printf("%s\n", "Begin SR protocol, please don't abort the process");
186. printf("The loss ratio of packet is %.2f,the loss ratio of ack is % .2f\n", packetLossRatio, ackLossRatio);
187. int waitCount = 0;
188. int stage = 0;
189. BOOL b;
190. unsigned char u\_code; *// 状态码*
191. unsigned short seq;  *// 包的序列号*
192. unsigned short recvSeq; *// 接收窗口大小为 1，已确认的序列号*
193. unsigned short waitSeq; *// 等待的序列号*
194. sendto(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));
195. if (!strcmp(operation, "download")) {
196. *// sr 0.2 0.2 download testdownload.txt*
197. char data[1024 \* 113];
198. BOOL recvd[20] = { FALSE };
199. iMode = 0;
200. ioctlsocket(socketClient, FIONBIO, (u\_long FAR\*) & iMode);
201. sendack = 0;
202. while (true) {
203. recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, &len);
204. switch (stage) {
205. case 0: *// 等待握手阶段*
206. u\_code = (unsigned char)buffer[0];
207. if ((unsigned char)buffer[0] == 205) {
208. printf("Ready for file transmission\n");
209. buffer[0] = 200;
210. buffer[1] = '\0';
211. sendto(socketClient, buffer, 2, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));
212. stage = 1;
213. recvSeq = 0;
214. waitSeq = 0;
215. totalSeq = 0;
216. loct = -2;
217. }
218. break;
219. case 1: *// 等待接收数据阶段*
220. seq = (unsigned short)buffer[0];
221. *// 随机法模拟包是否丢失*
222. b = lossInLossRatio(packetLossRatio);
223. if (b) {
224. printf("The packet with a seq of %d loss\n", seq - 1);
225. continue;
226. }
227. printf("recv a packet with a seq of %d\n", seq - 1);
228. *// 如果是期待的包，正确接收，正常确认即可*
229. seq -= 1;
230. if (!(waitSeq - seq)) {
231. recvd[waitSeq] = TRUE;
232. memcpy(data + 1024 \* totalSeq, buffer + 2, 1024);
233. if (buffer[1] == '0') loct = totalSeq;
234. int cnt = 10;
235. while (cnt--) {
236. if (recvd[waitSeq]) {
237. recvd[waitSeq] = FALSE;
238. ++waitSeq;
239. ++totalSeq;
240. if (waitSeq == 20) waitSeq = 0;
241. }
242. else break;
243. }
244. }
245. else {
246. int index = (seq + SEQ\_SIZE - waitSeq) % SEQ\_SIZE;
247. if (index < 10 && !recvd[seq]) {
248. recvd[seq] = TRUE;
249. memcpy(data + 1024 \* (totalSeq + index), buffer + 2, 1024);
250. if (buffer[1] == '0') loct = totalSeq + index;
251. }
252. }
253. buffer[0] = (char)(seq + 1);
254. buffer[2] = '\0';
255. b = lossInLossRatio(ackLossRatio);
256. if (b) {
257. printf("The ack of %d loss\n", (unsigned char)buffer[0] - 1);
258. continue;
259. }
260. ++sendack;
261. sendto(socketClient, buffer, 3, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));
262. printf("send a ack of %d\n", (unsigned char)buffer[0] - 1);
263. break;
264. }
265. if (sendack == loct + 1) {
266. printf("接收完成\n");
267. break;
268. }
269. Sleep(20);
270. }
271. char buff[1300];
272. ofstream ofs;
273. ofs.open(filename, ios::out);
274. for (int i = 0; i <= loct; ++i) {
275. memcpy(buff, data + 1024 \* i, 1024);
276. ofs << buff << endl;
277. }
278. ofs.close();
279. if (sendack == loct + 1) {
280. ZeroMemory(buffer, sizeof(buffer));
281. continue;
282. }
283. }
284. else if (!(strcmp(operation, "upload"))) {
285. iMode = 1;
286. ioctlsocket(socketClient, FIONBIO, (u\_long FAR\*) & iMode);
287. std::ifstream fin;
288. fin.open(filename, ios\_base::in);
289. if (!fin.is\_open()) {
290. printf("无法打开文件");
291. continue;
292. }
293. char buff[1024] = { 0 };
294. char data[1024 \* 113];
295. loct = 0;
296. while (fin.getline(buff, sizeof(buff))) {
297. if (buff[0] == '0') break;
298. memcpy(data + 1024 \* loct, buff, 1024);
299. ++loct;
300. }
301. fin.close(); *// read file*
302. totalPacket = loct;
303. ZeroMemory(buffer, sizeof(buffer));
304. int recvSize;
305. int waitCounts[21] = { 0 };
306. waitCount = 0;
307. printf("Begain to test SR protocol,please don't abort the process\n");
308. printf("Shake hands stage\n");
309. int stage = 0;
310. bool runFlag = true;
311. while (runFlag) {
312. switch (stage) {
313. case 0: *// 发送 205 阶段*
314. buffer[0] = 205;
315. sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));
316. Sleep(100);
317. stage = 1;
318. break;
319. 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*
320. recvSize = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrServer), &len);
321. if (recvSize < 0) {
322. ++waitCount;
323. if (waitCount > 20) {
324. runFlag = false;
325. printf("Timeout error\n");
326. break;
327. }
328. Sleep(20);
329. continue;
330. }
331. else {
332. if ((unsigned char)buffer[0] == 200) {
333. printf("Begin a file transfer\n");
334. printf("File size is %dB, each packet is 1024B and packet total num is % d\n", totalPacket \* 1024, totalPacket);
335. curSeq = 0;
336. curAck = 0;
337. totalSeq = 0;
338. waitCount = 0;
339. waitSeq = 0;
340. stage = 2;
341. for (int i = 0; i < SEQ\_SIZE; ++i) {
342. ack[i] = TRUE;
343. }
344. }
345. }
346. break;
347. case 2:
348. if (seqIsAvailable() && totalSeq < loct) {
349. *// 发送给客户端的序列号从 1 开始*
350. buffer[0] = curSeq + 1;
351. if (totalSeq == loct - 1) buffer[1] = '0';
352. else buffer[1] = '1';
353. ack[curSeq] = FALSE;
354. memcpy(&buffer[2], data + 1024 \* totalSeq, 1024);
355. printf("send a packet with a seq of %d\n", curSeq);
356. sendto(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));
357. ++curSeq;
358. curSeq %= SEQ\_SIZE;
359. ++totalSeq;
360. Sleep(20);
361. }
362. *// 等待 Ack，若没有收到，则返回值为-1，计数器+1*
363. recvSize = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrServer), &len);
364. if (recvSize >= 0) {
365. *// 收到 ack*
366. int i = (int)(buffer[0]) - 1;
367. ack[i] = TRUE;
368. waitCounts[i] = 0;
369. printf("Recv a ack of %d\n", i);
370. if (i == curAck) {
371. if (curSeq < curAck) {
372. for (; curAck < SEQ\_SIZE;) {
373. if (ack[curAck]) ++curAck;
374. else break;
375. }
376. if (curAck == SEQ\_SIZE) {
377. for (curAck = 0; curAck < curSeq;) {
378. if (ack[curAck]) ++curAck;
379. else break;
380. }
381. }
382. }
383. else {
384. for (; curAck < curSeq;) {
385. if (ack[curAck]) ++curAck;
386. else break;
387. }
388. }
389. }
390. if (curAck == curSeq && totalSeq == loct) break;
391. }
392. int index;
393. *// time out*
394. for (int i = 0; i < (curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE; ++i) {
395. index = (i + curAck) % SEQ\_SIZE;
396. if (!ack[index]) {
397. ++waitCounts[index];
398. if (waitCounts[index] > 20) {
399. buffer[0] = index + 1;
400. if (totalSeq - ((curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE) + i == loct - 1) buffer[1] = '0';
401. else buffer[1] = '1';
402. memcpy(&buffer[2], data + 1024 \* (totalSeq - ((curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE) + i), 1024);
403. printf("send a packet with a seq of %d\n", index);
404. sendto(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));
405. waitCounts[index] = 0;
406. }
407. }
408. }
409. Sleep(20);
410. break;
411. }
412. if (curAck == curSeq && totalSeq == loct) break;
413. }
414. if (curAck == curSeq && totalSeq == loct) {
415. printf("传输完成\n");
416. ZeroMemory(buffer, sizeof(buffer));
417. continue;
418. }
419. }
420. }
421. else if (!strcmp(cmd, "-testgbn")) {
422. printf("%s\n", "Begin GBN protocol, please don't abort the process");
423. printf("The loss ratio of packet is %.2f,the loss ratio of ack is % .2f\n", packetLossRatio, ackLossRatio);
424. int waitCount = 0;
425. int stage = 0;
426. BOOL b;
427. unsigned char u\_code; *// 状态码*
428. unsigned short seq;  *// 包的序列号*
429. unsigned short recvSeq; *// 接收窗口大小为 1，已确认的序列号*
430. unsigned short waitSeq; *// 等待的序列号*
431. unsigned short recvPacket;
432. sendto(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));
433. if (!strcmp(operation, "download")) {
434. char data[1024 \* 113];
435. loct = 0;
436. iMode = 0;
437. int flg = 1;
438. ioctlsocket(socketClient, FIONBIO, (u\_long FAR\*) & iMode);
439. while (true) {
440. *// 等待 server 回复设置 UDP 为阻塞模式*
441. recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, &len);
442. switch (stage) {
443. case 0: *// 等待握手阶段*
444. u\_code = (unsigned char)buffer[0];
445. if ((unsigned char)buffer[0] == 205) {
446. printf("Ready for file transmission\n");
447. buffer[0] = 200;
448. buffer[1] = '\0';
449. sendto(socketClient, buffer, 2, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));
450. stage = 1;
451. recvSeq = 0;
452. waitSeq = 1;
453. loct = 0;
454. }
455. break;
456. case 1: *// 等待接收数据阶段*
457. seq = (unsigned short)buffer[0];
458. *// 模拟随机丢包*
459. b = lossInLossRatio(packetLossRatio);
460. if (b) {
461. printf("The packet with a seq of %d loss\n", seq - 1);
462. continue;
463. }
464. printf("recv a packet with a seq of %d\n", seq - 1);
465. *// 如果是期待的包，正确接收，正常确认即可*
466. if (!(waitSeq - seq)) {
467. memcpy(data + 1024 \* loct, buffer + 2, 1024);
468. if (buffer[1] == '0') flg = 0;
469. ++loct;
470. ++waitSeq;
471. if (waitSeq == 21) {
472. waitSeq = 1;
473. }
474. buffer[0] = seq;
475. recvSeq = seq;
476. recvPacket = (unsigned short)buffer[1];
477. buffer[2] = '\0';
478. }
479. else {
480. *// 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）*
481. if (!recvSeq) {
482. continue;
483. }
484. buffer[0] = recvSeq;
485. buffer[1] = recvPacket;
486. buffer[2] = '\0';
487. }
488. b = lossInLossRatio(ackLossRatio);
489. if (b) {
490. printf("The ack of %d loss\n", (unsigned char)buffer[0] - 1);
491. continue;
492. }
493. sendto(socketClient, buffer, 3, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));
494. printf("send a ack of %d\n", (unsigned char)buffer[0] - 1);
495. break;
496. }
497. if (flg == 0) {
498. printf("接收完成\n");
499. break;
500. }
501. Sleep(20);
502. }
503. ofstream ofs;
504. ofs.open(filename, ios::out);
505. char buff[1300];
506. *//printf("%d", loct);*
507. for (int i = 0; i < loct; ++i) {
508. memcpy(buff, data + 1024 \* i, 1024);
509. ofs << buff << endl;
510. }
511. ofs.close();
512. if (flg == 0) {
513. ZeroMemory(buffer, sizeof(buffer));
514. continue;
515. }
516. }
517. else if (!strcmp(operation, "upload")) {
518. std::ifstream fin;
519. fin.open(filename, ios\_base::in);
520. if (!fin.is\_open()) {
521. printf("无法打开文件");
522. continue;
523. }
524. iMode = 1;
525. ioctlsocket(socketClient, FIONBIO, (u\_long FAR\*) & iMode);
526. char buff[1024] = { 0 };
527. char data[1024 \* 113];
528. loct = 0;
529. int flg = 1;
530. while (fin.getline(buff, sizeof(buff))) {
531. if (buff[0] == '0') break;
532. memcpy(data + 1024 \* loct, buff, 1024);
533. ++loct;
534. }
535. fin.close();
536. totalPacket = loct;
537. ZeroMemory(buffer, sizeof(buffer));
538. int recvSize;
539. waitCount = 0;
540. printf("Begain to test GBN protocol,please don't abort the process\n");
541. *// 加入了一个握手阶段*
542. *// 首先服务器向客户端发送一个 205 大小的状态码表示服务器准备好了，可以发送数据*
543. *// 客户端收到 205 之后回复一个 200 大小的状态码，表示客户端准备好了，可以接收数据了*
544. *// 服务器收到 200 状态码之后，就开始使用 GBN 发送数据了*
545. printf("Shake hands stage\n");
546. int stage = 0;
547. bool runFlag = true;
548. while (runFlag) {
549. switch (stage) {
550. case 0: *// 发送 205 阶段*
551. buffer[0] = 205;
552. sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));
553. Sleep(100);
554. stage = 1;
555. break;
556. 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*
557. recvSize = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrServer), &len);
558. if (recvSize < 0) {
559. ++waitCount;
560. if (waitCount > 20) {
561. runFlag = false;
562. printf("Timeout error\n");
563. break;
564. }
565. Sleep(20);
566. continue;
567. }
568. else {
569. if ((unsigned char)buffer[0] == 200) {
570. printf("Begin a file transfer\n");
571. printf("File size is %dB, each packet is 1024B and packet total num is % d\n", totalPacket \* 1024, totalPacket);
572. *// 准备传输，初始化*
573. curSeq = 0;
574. curAck = 0;
575. totalSeq = 0;
576. waitCount = 0;
577. stage = 2;
578. for (int i = 0; i < SEQ\_SIZE; ++i) {
579. ack[i] = TRUE;
580. }
581. }
582. }
583. break;
584. case 2: *// 数据传输阶段*
585. if (seqIsAvailable() && totalSeq < loct) {
586. *// 发送给客户端的序列号从 1 开始*
587. buffer[0] = curSeq + 1;
588. if (totalSeq == loct - 1) buffer[1] = '0';
589. else buffer[1] = '1';
590. ack[curSeq] = FALSE;
591. memcpy(&buffer[2], data + 1024 \* totalSeq, 1024);
592. printf("send a packet with a seq of %d\n", curSeq);
593. getTime(1);
594. printf("  序列号：%d\n", curSeq);
595. sendto(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));
596. ++curSeq;
597. curSeq %= SEQ\_SIZE;
598. ++totalSeq;
599. Sleep(20);
600. }
601. *// 等待 Ack，若没有收到，则返回值为-1，计数器+1*
602. recvSize = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrServer), &len);
603. if (recvSize < 0) {
604. waitCount++;
605. *// 20 次等待 ack 则超时重传*
606. if (waitCount > 20) {
607. timeoutHandler();
608. waitCount = 0;
609. }
610. }
611. else {
612. *// 收到 ack*
613. if (buffer[1] == '0')
614. {
615. flg = 0;
616. break;
617. }
618. ackHandler(buffer[0]);
619. waitCount = 0;
620. }
621. Sleep(20);
622. break;
623. }
624. if (flg == 0) break;
625. }
626. if (flg == 0) {
627. printf("传输完成\n");
628. ZeroMemory(buffer, sizeof(buffer));
629. continue;
630. }
631. }
632. }
633. sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));
634. ret = recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrServer, &len);
635. printf("%s\n", buffer);
636. if (!strcmp(buffer, "Good bye!")) {
637. break;
638. }
639. }
640. *// 关闭套接字*
641. closesocket(socketClient);
642. WSACleanup();
643. return 0;
644. }

2.服务器端server：

1. #include <stdlib.h>
2. #include <time.h>
3. #include <WinSock2.h>
4. #include<windows.h>
5. #include <fstream>
6. #include<sstream>
7. #include<cstdio>
8. #pragma comment(lib,"ws2\_32.lib")
9. #pragma warning(disable:4996)
10. #define SERVER\_PORT 12340 *//端口号*
11. #define SERVER\_IP "0.0.0.0" *//IP 地址*
12. using namespace std;
13. const int BUFFER\_LENGTH = 1027; *// 缓冲区大小，（以太网中 UDP 的数据帧中包长度应小于 1480 字节）*
14. 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）*
15. *//本例取序列号 0...19 共 20 个*
16. *//如果将窗口大小设为 1，则为停-等协议*
17. const int SEQ\_SIZE = 20; *// 序列号的个数，从 0~19 共计 20 个*
18. *//由于发送数据第一个字节如果值为 0，则数据会发送失败*
19. *//因此接收端序列号为 1~20，与发送端一一对应*
20. BOOL ack[SEQ\_SIZE]; *// ack缓存*
21. int curSeq; *// 当前数据包的 seq*
22. int curAck; *// 当前等待确认的 ack*
23. int totalSeq; *// 收到的包的总数*
24. int totalPacket;*// 需要发送的包总数*
25. int waitSeq;
26. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
27. *// Method: getCurTime*
28. *// FullName: getCurTime*
29. *// Access: public*
30. *// Returns: void*
31. *// Qualifier: 获取当前系统时间，结果存入 ptime 中*
32. *// Parameter: char \* ptime*
33. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
34. void getCurTime(char\* ptime) {
35. char buffer[128];
36. memset(buffer, 0, sizeof(buffer));
37. SYSTEMTIME sys;
38. GetLocalTime(&sys);
39. sprintf\_s(buffer, "%4d/%02d/%02d %02d:%02d:%02d",
40. sys.wYear,
41. sys.wMonth,
42. sys.wDay,
43. sys.wHour,
44. sys.wMinute,
45. sys.wSecond);
46. strcpy\_s(ptime, sizeof(buffer), buffer);
47. }
48. void getTime(int flg) {
49. time\_t timep;
50. struct tm\* p;
51. char cur\_time[28];
52. time(&timep);
53. p = gmtime(&timep);
54. sprintf(cur\_time, "%d.%d.%d %d:%d:%d", 1900 + p->tm\_year, 1 + p->tm\_mon, p->tm\_mday, 8 + p->tm\_hour, p->tm\_min, p->tm\_sec);
55. if (flg == 1) {
56. printf("数据包发送时间：%s", cur\_time);
57. }
58. else if (flg == 2) {
59. printf("ACK接收时间：%s", cur\_time);
60. }
61. }
62. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
63. *// Method: seqIsAvailable*
64. *// FullName: seqIsAvailable*
65. *// Access: public*
66. *// Returns: bool*
67. *// Qualifier: 当前序列号 curSeq 是否可用*
68. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
69. bool seqIsAvailable() {
70. int step;
71. step = curSeq - curAck;
72. step = step >= 0 ? step : step + SEQ\_SIZE;
73. *// 序列号是否在当前发送窗口之内*
74. if (step >= SEND\_WIND\_SIZE) {
75. return false;
76. }
77. if (ack[curSeq]) {
78. return true;
79. }
80. return false;
81. }
82. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
83. *// Method: timeoutHandler*
84. *// FullName: timeoutHandler*
85. *// Access: public*
86. *// Returns: void*
87. *// Qualifier: 超时重传处理函数，滑动窗口内的数据帧都要重传*
88. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
89. void timeoutHandler() {
90. printf("Timer out error.\n");
91. int index;
92. for (int i = 0; i < (curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE; ++i) {
93. index = (i + curAck) % SEQ\_SIZE;
94. ack[index] = TRUE;
95. }
96. totalSeq = totalSeq - ((curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE);
97. curSeq = curAck;
98. }
99. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
100. *// Method: ackHandler*
101. *// FullName: ackHandler*
102. *// Access: public*
103. *// Returns: void*
104. *// Qualifier: 收到 ack，累积确认，取数据帧的第一个字节*
105. *//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*
106. *// Parameter: char c*
107. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
108. void ackHandler(char c) {
109. unsigned char index = (unsigned char)c - 1; *// 序列号-1*
110. printf("Recv a ack of %d\n", index);
111. getTime(2);
112. printf("  序列号：%d\n", index);
113. if (curAck <= index) {
114. for (int i = curAck; i <= index; ++i) {
115. ack[i] = TRUE;
116. }
117. curAck = (index + 1) % SEQ\_SIZE;
118. }
119. else {
120. *// ack 超过了最大值，回到了 curAck 的左边*
121. for (int i = curAck; i < SEQ\_SIZE; ++i) {
122. ack[i] = TRUE;
123. }
124. for (int i = 0; i <= index; ++i) {
125. ack[i] = TRUE;
126. }
127. curAck = index + 1;
128. }
129. }
130. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
131. *// Method: lossInLossRatio*
132. *// Access: public*
133. *// Returns: BOOL*
134. *// Qualifier: 模拟随机丢包，返回TRUE则执行丢包*
135. *// Parameter: float lossRatio*
136. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
137. BOOL lossInLossRatio(float lossRatio) {
138. int lossBound = (int)(lossRatio \* 100);
139. int r = rand() % 100;
140. if (r < lossBound) {
141. return TRUE;
142. }
143. return FALSE;
144. }
145. *//主函数*
146. int main()
147. {
148. *// 加载套接字库（必须）*
149. WORD wVersionRequested;
150. WSADATA wsaData;
151. *// 套接字加载时错误提示*
152. int err;
153. *// 版本 2.2*
154. wVersionRequested = MAKEWORD(2, 2);
155. *// 加载 dll 文件 Scoket 库*
156. err = WSAStartup(wVersionRequested, &wsaData);
157. if (err != 0) {
158. *// 找不到 winsock.dll*
159. printf("WSAStartup failed with error: %d\n", err);
160. return -1;
161. }
162. if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) {
163. printf("Could not find a usable version of Winsock.dll\n");
164. WSACleanup();
165. }
166. else {
167. printf("The Winsock 2.2 dll was found okay\n");
168. }
169. SOCKET sockServer = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);
170. *// 设置套接字为非阻塞模式*
171. int iMode; *//1：非阻塞，0：阻塞*
172. SOCKADDR\_IN addrServer; *// 服务器地址*
173. *// addrServer.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER\_IP);*
174. addrServer.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY); *// 两者均可*
175. addrServer.sin\_family = AF\_INET;
176. addrServer.sin\_port = htons(SERVER\_PORT);
177. err = bind(sockServer, (SOCKADDR\*)&addrServer, sizeof(SOCKADDR));
178. if (err) {
179. err = GetLastError();
180. printf("Could not bind the port %d for socket.Error code is % d\n", SERVER\_PORT, err);
181. WSACleanup();
182. return -1;
183. }
184. SOCKADDR\_IN addrClient; *// 客户端地址*
185. int length = sizeof(SOCKADDR);
186. char buffer[BUFFER\_LENGTH]; *// 数据发送接收缓冲区*
187. ZeroMemory(buffer, sizeof(buffer));
188. *// 将测试数据读入内存*
189. int recvSize;
190. int loct = 0;
191. int waitCount = 0;
192. float packetLossRatio = 0.2; *// 默认包丢失率 0.2*
193. float ackLossRatio = 0.2;  *// 默认 ACK 丢失率 0.2*
194. srand((unsigned)time(NULL));
195. while (true) {
196. *// 非阻塞接收，若没有收到数据，返回值为-1*
197. recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);
198. printf("recv from client: %s\n", buffer);
199. if (strcmp(buffer, "-time") == 0) {
200. getCurTime(buffer);
201. }
202. else if (strcmp(buffer, "-quit") == 0) {
203. strcpy\_s(buffer, strlen("Good bye!") + 1, "Good bye!");
204. }
205. else {
206. char filename[100];
207. char operation[10];
208. char cmd[10];
209. int ret;
210. unsigned char u\_code; *// 状态码*
211. unsigned short seq;  *// 包的序列号*
212. unsigned short recvSeq; *// 接收窗口大小为 1，已确认的序列号*
213. unsigned short waitSeq; *// 等待的序列号*
214. unsigned short recvPacket;
215. int sendack = 0;
216. int stage = 0;
217. ret = sscanf(buffer, "%s %f %f %s %s", &cmd, &packetLossRatio, &ackLossRatio, &operation, &filename);
218. if (!strcmp(cmd, "-testgbn")) {
219. if (!strcmp(operation, "download")) {
220. iMode = 1;
221. int flg = 1;
222. ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);
223. std::ifstream fin;
224. fin.open(filename, ios\_base::in);
225. if (!fin.is\_open()) {
226. printf("无法打开文件");
227. iMode = 0;
228. ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);
229. continue;
230. }
231. char buff[1024] = { 0 };
232. char data[1024 \* 113];
233. loct = 0;
234. while (fin.getline(buff, sizeof(buff))) {
235. if (buff[0] == '0') break;
236. memcpy(data + 1024 \* loct, buff, 1024);
237. ++loct;
238. }
239. fin.close();
240. totalPacket = loct;
241. ZeroMemory(buffer, sizeof(buffer));
242. int recvSize;
243. waitCount = 0;
244. printf("Begain to test GBN protocol,please don't abort the process\n");
245. *// 加入了一个握手阶段*
246. *// 首先服务器向客户端发送一个 205 大小的状态码（我自己定义的）表示服务器准备好了，可以发送数据*
247. *// 客户端收到 205 之后回复一个 200 大小的状态码，表示客户端准备好了，可以接收数据了*
248. *// 服务器收到 200 状态码之后，就开始使用 GBN 发送数据了*
249. printf("Shake hands stage\n");
250. int stage = 0;
251. bool runFlag = true;
252. while (runFlag) {
253. switch (stage) {
254. case 0: *// 发送 205 阶段*
255. buffer[0] = 205;
256. sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));
257. Sleep(100);
258. stage = 1;
259. break;
260. 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*
261. recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);
262. if (recvSize < 0) {
263. ++waitCount;
264. if (waitCount > 20) {
265. runFlag = false;
266. printf("Timeout error\n");
267. break;
268. }
269. Sleep(500);
270. continue;
271. }
272. else {
273. if ((unsigned char)buffer[0] == 200) {
274. printf("Begin a file transfer\n");
275. printf("File size is %dB, each packet is 1024B and packet total num is % d\n", totalPacket \* 1024, totalPacket);
276. curSeq = 0;
277. curAck = 0;
278. totalSeq = 0;
279. waitCount = 0;
280. stage = 2;
281. for (int i = 0; i < SEQ\_SIZE; ++i) {
282. ack[i] = TRUE;
283. }
284. }
285. }
286. break;
287. case 2: *// 数据传输阶段*
288. if (seqIsAvailable() && totalSeq < loct) {
289. *// 发送给客户端的序列号从 1 开始*
290. buffer[0] = curSeq + 1;
291. if (totalSeq == loct - 1) buffer[1] = '0';
292. else buffer[1] = '1';
293. ack[curSeq] = FALSE;
294. *// 数据发送的过程中应该判断是否传输完成*
295. *// 为简化过程此处并未实现*
296. memcpy(&buffer[2], data + 1024 \* totalSeq, 1024);
297. printf("send a packet with a seq of %d\n", curSeq);
298. getTime(1);
299. printf("  序列号：%d\n", curSeq);
300. sendto(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));
301. ++curSeq;
302. curSeq %= SEQ\_SIZE;
303. ++totalSeq;
304. Sleep(500);
305. }
306. *// 等待 Ack，若没有收到，则返回值为-1，计数器+1*
307. recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);
308. if (recvSize < 0) {
309. waitCount++;
310. *// 20 次等待 ack 则超时重传*
311. if (waitCount > 20) {
312. timeoutHandler();
313. waitCount = 0;
314. }
315. }
316. else {
317. *// 收到 ack*
318. if (buffer[1] == '0')
319. {
320. flg = 0;
321. break;
322. }
323. ackHandler(buffer[0]);
324. waitCount = 0;
325. }
326. Sleep(500);
327. break;
328. }
329. if (flg == 0) break;
330. }
331. if (flg == 0) {
332. printf("传输完成\n");
333. iMode = 0;
334. ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);
335. ZeroMemory(buffer, sizeof(buffer));
336. continue;
337. }
338. }
339. else if (!strcmp(operation, "upload")) {
340. char data[1024 \* 113];
341. loct = 0;
342. int flg = 1;
343. BOOL b;
344. *// gbn 0 0 download test.txt*
345. while (true) {
346. *// 等待 server 回复设置 UDP 为阻塞模式*
347. recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, &length);
348. switch (stage) {
349. case 0: *// 等待握手阶段*
350. u\_code = (unsigned char)buffer[0];
351. if ((unsigned char)buffer[0] == 205) {
352. printf("Ready for file transmission\n");
353. buffer[0] = 200;
354. buffer[1] = '\0';
355. sendto(sockServer, buffer, 2, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));
356. stage = 1;
357. recvSeq = 0;
358. waitSeq = 1;
359. loct = 0;
360. }
361. break;
362. case 1: *// 等待接收数据阶段*
363. seq = (unsigned short)buffer[0];
364. b = lossInLossRatio(packetLossRatio);
365. if (b) {
366. printf("The packet with a seq of %d loss\n", seq - 1);
367. continue;
368. }
369. printf("recv a packet with a seq of %d\n", seq - 1);
370. *// 如果是期待的包，正确接收，正常确认即可*
371. if (!(waitSeq - seq)) {
372. if (buffer[1] == '0') flg = 0;
373. memcpy(data + 1024 \* loct, buffer + 2, 1024);
374. ++loct;
375. ++waitSeq;
376. if (waitSeq == 21) {
377. waitSeq = 1;
378. }
379. *// 输出数据*
380. *// printf("%s\n",&buffer[1]);*
381. buffer[0] = seq;
382. recvSeq = seq;
383. recvPacket = (unsigned short)buffer[1];
384. buffer[2] = '\0';
385. }
386. else {
387. *// 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）*
388. if (!recvSeq) {
389. continue;
390. }
391. buffer[0] = recvSeq;
392. buffer[1] = recvPacket;
393. buffer[2] = '\0';
394. }
395. b = lossInLossRatio(ackLossRatio);
396. if (b) {
397. printf("The ack of %d loss\n", (unsigned char)buffer[0] - 1);
398. continue;
399. }
400. sendto(sockServer, buffer, 3, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));
401. printf("send a ack of %d\n", (unsigned char)buffer[0] - 1);
402. break;
403. }
404. if (flg == 0) {
405. printf("接收完成\n");
406. break;
407. }
408. Sleep(500);
409. }
410. ofstream ofs;
411. ofs.open(filename, ios::out);
412. char buff[1300];
413. *//printf("%d", loct);*
414. for (int i = 0; i < loct; ++i) {
415. memcpy(buff, data + 1024 \* i, 1024);
416. ofs << buff << endl;
417. }
418. ofs.close();
419. if (flg == 0) {
420. ZeroMemory(buffer, sizeof(buffer));
421. continue;
422. }
423. }
424. }
425. else if (!strcmp(cmd, "-testsr"))
426. {
427. if (!strcmp(operation, "download")) {
428. iMode = 1;
429. ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);
430. std::ifstream fin;
431. fin.open(filename, ios\_base::in);
432. if (!fin.is\_open()) {
433. printf("无法打开文件");
434. iMode = 0;
435. ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);
436. continue;
437. }
438. char buff[1024] = { 0 };
439. char data[1024 \* 113];
440. loct = 0;
441. while (fin.getline(buff, sizeof(buff))) {
442. if (buff[0] == '0') break;
443. memcpy(data + 1024 \* loct, buff, 1024);
444. ++loct;
445. }
446. fin.close();
447. totalPacket = loct;
448. ZeroMemory(buffer, sizeof(buffer));
449. int recvSize;
450. int waitCounts[21] = { 0 };
451. waitCount = 0;
452. printf("Begain to test SR protocol,please don't abort the process\n");
453. *// 加入了一个握手阶段*
454. *// 首先服务器向客户端发送一个 205 大小的状态码,表示服务器准备好了，可以发送数据*
455. *// 客户端收到 205 之后回复一个 200 大小的状态码，表示客户端准备好了，可以接收数据了*
456. *// 服务器收到 200 状态码之后，就开始使用 GBN 发送数据了*
457. printf("Shake hands stage\n");
458. int stage = 0;
459. bool runFlag = true;
460. while (runFlag) {
461. switch (stage) {
462. case 0: *// 发送 205 阶段*
463. buffer[0] = 205;
464. sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));
465. Sleep(100);
466. stage = 1;
467. break;
468. 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*
469. recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);
470. if (recvSize < 0) {
471. ++waitCount;
472. if (waitCount > 20) {
473. runFlag = false;
474. printf("Timeout error\n");
475. break;
476. }
477. Sleep(500);
478. continue;
479. }
480. else {
481. if ((unsigned char)buffer[0] == 200) {
482. printf("Begin a file transfer\n");
483. printf("File size is %dB, each packet is 1024B and packet total num is % d\n", totalPacket \* 1024, totalPacket);
484. curSeq = 0;
485. curAck = 0;
486. totalSeq = 0;
487. waitCount = 0;
488. waitSeq = 0;
489. stage = 2;
490. for (int i = 0; i < SEQ\_SIZE; ++i) {
491. ack[i] = TRUE;
492. }
493. }
494. }
495. break;
496. case 2:
497. if (seqIsAvailable() && totalSeq < loct) {
498. *// 发送给客户端的序列号从 1 开始*
499. buffer[0] = curSeq + 1;
500. if (totalSeq == loct - 1) buffer[1] = '0';
501. else buffer[1] = '1';
502. ack[curSeq] = FALSE;
503. *// 数据发送的过程中应该判断是否传输完成*
504. *// 为简化过程此处并未实现*
505. memcpy(&buffer[2], data + 1024 \* totalSeq, 1024);
506. printf("send a packet with a seq of %d\n", curSeq);
507. sendto(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));
508. ++curSeq;
509. curSeq %= SEQ\_SIZE;
510. ++totalSeq;
511. Sleep(500);
512. }
513. *// 等待 Ack，若没有收到，则返回值为-1，计数器+1*
514. recvSize = recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)&addrClient), &length);
515. if (recvSize >= 0) {
516. *// 收到 ack*
517. int i = (int)(buffer[0]) - 1;
518. ack[i] = TRUE;
519. waitCounts[i] = 0;
520. printf("Recv a ack of %d\n", i);
521. if (i == curAck) {
522. if (curSeq < curAck) {
523. for (; curAck < SEQ\_SIZE;) {
524. if (ack[curAck]) ++curAck;
525. else break;
526. }
527. if (curAck == SEQ\_SIZE) {
528. for (curAck = 0; curAck < curSeq;) {
529. if (ack[curAck]) ++curAck;
530. else break;
531. }
532. }
533. }
534. else {
535. for (; curAck < curSeq;) {
536. if (ack[curAck]) ++curAck;
537. else break;
538. }
539. }
540. }
541. if (curAck == curSeq && totalSeq == loct) break;
542. }
543. int index;
544. for (int i = 0; i < (curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE; ++i) {
545. index = (i + curAck) % SEQ\_SIZE;
546. if (!ack[index]) {
547. ++waitCounts[index];
548. if (waitCounts[index] > 20) {
549. buffer[0] = index + 1;
550. if (totalSeq - ((curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE) + i == loct - 1) buffer[1] = '0';
551. else buffer[1] = '1';
552. memcpy(&buffer[2], data + 1024 \* (totalSeq - ((curSeq - curAck + SEQ\_SIZE) % SEQ\_SIZE) + i), 1024);
553. printf("send a packet with a seq of %d\n", index);
554. sendto(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));
555. waitCounts[index] = 0;
556. }
557. }
558. }
559. Sleep(500);
560. break;
561. }
562. if (curAck == curSeq && totalSeq == loct) break;
563. }
564. if (curAck == curSeq && totalSeq == loct) {
565. printf("传输完成\n");
566. iMode = 0;
567. ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*) & iMode);
568. ZeroMemory(buffer, sizeof(buffer));
569. continue;
570. }
571. }
572. else if (!strcmp(operation, "upload")) {
573. *// sr 0 0 download test.txt*
574. char data[1024 \* 113];
575. BOOL recvd[20] = { FALSE };
576. sendack = 0;
577. BOOL b;
578. while (true) {
579. *// 等待 server 回复设置 UDP 为阻塞模式*
580. recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)&addrClient, &length);
581. switch (stage) {
582. case 0: *// 等待握手阶段*
583. u\_code = (unsigned char)buffer[0];
584. if ((unsigned char)buffer[0] == 205) {
585. printf("Ready for file transmission\n");
586. buffer[0] = 200;
587. buffer[1] = '\0';
588. sendto(sockServer, buffer, 2, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));
589. stage = 1;
590. recvSeq = 0;
591. waitSeq = 0;
592. totalSeq = 0;
593. loct = -2;
594. }
595. break;
596. case 1: *// 等待接收数据阶段*
597. seq = (unsigned short)buffer[0];
598. *// 随机法模拟包是否丢失*
599. b = lossInLossRatio(packetLossRatio);
600. if (b) {
601. printf("The packet with a seq of %d loss\n", seq - 1);
602. continue;
603. }
604. printf("recv a packet with a seq of %d\n", seq - 1);
605. *// 如果是期待的包，正确接收，正常确认即可*
606. seq -= 1;
607. if (!(waitSeq - seq)) {
608. recvd[waitSeq] = TRUE;
609. memcpy(data + 1024 \* totalSeq, buffer + 2, 1024);
610. if (buffer[1] == '0') loct = totalSeq;
611. int cnt = 10;
612. while (cnt--) {
613. if (recvd[waitSeq]) {
614. recvd[waitSeq] = FALSE;
615. ++waitSeq;
616. ++totalSeq;
617. if (waitSeq == 20) waitSeq = 0;
618. }
619. else break;
620. }
621. }
622. else {
623. int index = (seq + SEQ\_SIZE - waitSeq) % SEQ\_SIZE;
624. if (index < 10 && !recvd[seq]) {
625. recvd[seq] = TRUE;
626. memcpy(data + 1024 \* (totalSeq + index), buffer + 2, 1024);
627. if (buffer[1] == '0') loct = totalSeq + index;
628. }
629. }
630. buffer[0] = (char)(seq + 1);
631. buffer[2] = '\0';
632. b = lossInLossRatio(ackLossRatio);
633. if (b) {
634. printf("The ack of %d loss\n", (unsigned char)buffer[0] - 1);
635. continue;
636. }
637. ++sendack;
638. sendto(sockServer, buffer, 3, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));
639. printf("send a ack of %d\n", (unsigned char)buffer[0] - 1);
640. break;
641. }
642. if (sendack == loct + 1) {
643. printf("接收完成\n");
644. break;
645. }
646. Sleep(500);
647. }
648. char buff[1300];
649. ofstream ofs;
650. ofs.open(filename, ios::out);
651. for (int i = 0; i <= loct; ++i) {
652. memcpy(buff, data + 1024 \* i, 1024);
653. ofs << buff << endl;
654. }
655. ofs.close();
656. if (sendack == loct + 1) {
657. ZeroMemory(buffer, sizeof(buffer));
658. continue;
659. }
660. }
661. }
662. }
663. *// 进入 gbn 测试阶段*
664. *// 首先 server（server 处于 0 状态）向 client 发送 205 状态码（server进入 1 状态）*
665. *// 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\*
666. *// 在文件传输阶段，server 发送窗口大小设为*
667. sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)&addrClient, sizeof(SOCKADDR));
668. Sleep(500);
669. }
670. *// 关闭套接字，卸载库*
671. closesocket(sockServer);
672. WSACleanup();
673. return 0;
674. }