**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 | | | | | |
| 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： |
| （注：实验报告模板中的各项内容仅供参考，可依照实际实验情况进行修改。）  本次实验的主要目的。  掌握滑动窗口协议的基本原理和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，  改进所设计的 GBN 协议，支持双向数据传输，实现一个 C/S 结构的文件传输应用。  将所设计的 GBN 协议改进为 SR 协议。 |
| 实验内容： |
| 概述本次实验的主要内容，包含的实验项等。  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。  5）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。 |
| 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。   1. **基于UDP的GBN协议**      * 1. **GBN（或SR）协议数据分组格式与各个域作用**     （1）Seq 为 1 个字节，取值为 0~255，（故序列号最多为 256 个），在程序中设定Seq范围为0-20。  （2）Data≤1024 个字节，为传输的数据。  （3）最后一个字节放入 EOF0，表示结尾。  （4）因此缓冲区大小设置为1026字节。  **1.2 确认分组ACK帧格式与各个域作用**    ACK帧由两个字节组成，第一个字节表示序列号，第二个字节用”\0”填充。  **1.3 协议两端程序流程图**  服务器端（发送端）：    客户端（接收端）：    **1.4 协议典型交互过程**  **1.4.1 GBN协议**    **1.4.2 SR协议**    **1.5 数据分组丢失验证模拟方法**  （1）数据分组  服务器端对发送的数据帧进行计数，以概率1- packetLossRatio发送数据分组（即对于每个数据分组，然后对总数进行模 N 运算，若规定求模运算结果为零则发送数据分组。当 N 取值大于服务器端的超时阀值时，则会出现服务器端超时现象。  （2）ACK分组  客户端对接收的数据帧进行计数，然后对总数进行模 N 运算，若规定求模运算结果为零则返回 ACK，则每接收 N 个数据帧才返回 1 个 ACK。当 N 取值大于服务器端的超时阀值时，则会出现服务器端超时现象。  **2. 程序实现的主要类（或函数）及其主要作用**  **2.1 服务器端 GBN\_server.cpp**   |  |  | | --- | --- | | 主要函数 | 作用 | | lossInLossRatio() | 按照概率模拟产生数据丢失 | | seqIsAvailable() | 判断下一个序列号是否可以发送 | | timeoutHandler() | 超时处理函数，发生超时时所有已发送但没有收到ack的数据需要重传 | | Main函数 | 主程序 | | ackHandler() | 累计确认ack | |  |  |   **2.1.1 函数的具体实现**  （1）ackHandler()函数  流程图：    由于要发送的数据报的数量可能会超过序列号数量SEQ\_NUMBER，因此要考虑当前数据报序列号小于上一个被确认的序列号的情况。  （2）lossInLossRatio()函数  函数实现：模拟数据丢失。  假设数据丢失的概率是p，使用rand()函数随机生成一个数字r，若r%100的余数小于p\*100，则表示数据丢失。    （3）seqIsAvailable()函数  设计思想：判断当前序列号是否在发送窗口内    （4）timeoutHandler()函数  流程图：    函数代码：    **2.2 客户端 GBN\_client.cpp**   |  |  | | --- | --- | | 主要函数 | 作用 | | lossInLossRatio() | 按照概率模拟产生ack丢失 | | main函数 | 主程序 |   **2.2.1 函数的具体实现**  （1）lossInLossRatio()函数  函数实现：模拟ack丢失。  假设ack丢失的概率是p，使用rand()函数随机生成一个数字r，若r%100的余数小于p\*100，则表示数据丢失。    **3.支持双向数据传输**  单向传输是由服务器端向客户端传输数据，因此要实现双向数据传输，需要使得客户端向服务器端发送数据。  在服务器端增加与客户端相同的代码，来实现接收文件功能。  在客户端增加与服务器端相同的代码，来实现发送文件功能。  **4. SR协议**  **4.1 协议两端程序流程图**    **4.2 改进部分**  **4.2.1 客户端增加接收窗口**  **（1）seqRecvAvailable()函数**  功能：判断接收的包是否在接收窗口内，如果在则接收，否则丢弃。  实现代码：    **4.2.2 服务器端改进累计确认机制，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**  **（1）ackHandler()函数**  设计思想：在发送窗口内，对于非顺序到达的ack，对分组进行确认；对于顺序到达的ack，窗口右移至下一个未ACK的对象。  “非顺序”部分代码如下：    “顺序到达”部分代码如下： |
| 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协议的实现** 2. **窗口大小设置为1（实现停等协议）**   **服务器端（发送数据）：**    **客户端（接收数据）：**    **（2）窗口大小不为1**  **服务器端（发送数据）：**    发送的文件：    **客户端（接收数据）：**    接收的文件：     1. **双向传输的实现**   **服务器端（接收数据）：**    接收的文件：    **客户端（发送数据）：**    发送的文件：     1. **SR协议**   **服务器端（发送数据）：**    **客户端（接收数据）：** |
| 问题讨论： |
| 1. 在实现一个 C/S 结构的文件传输应用时，报错如下：debug assertion failed expression:(fh>=0 && ( unsigned) fh   原因：在网络查询报错后发现是文件读写过程资源未及时释放，虽然有fclose，但while循环不终止，因此未关闭文件  解决方案：添加判断条件if (curAck - 1 == totalPacket) break;使得循环终止并关闭文件。   1. 数据传输结束后服务器仍然发送数据，并收到201的ack   C:\Users\19470\Documents\Tencent Files\1947099691\FileRecv\MobileFile\Image\6]1UXQ%O0$D`ML]UTO{`~G9.png  原因：数据发送完成后未及时终止发送状态，再次发送收到的ack是200(需要减1)  解决方案：改正判断条件，及时退出循环   1. 在实现全双工传输时，接收数据的客户器端始终显示recvieve the seq 6 和send the ack 6，不继续发送数据报7,8   原因：阻塞状态是调用结果返回前当前线程会被挂起；非阻塞是在调用结果返回前不会阻塞当前线程，立即返回  解决方案：设置非阻塞状态 |
| 心得体会： |
| 结合实验过程和结果给出实验的体会和收获。  在本次实验，我使用C++程序设计停等协议、GBN协议和SR协议，更加深入地了解了三者之间的关系。我是先实现停等协议，在其基础上设计GBN协议，最后在GBN协议的基础上改进成了SR协议。在确保所有数据正确传输的前提下，所耗费的时间：停等协议>GBN协议>SR协议。通过本次实验，我更加深刻地感受到滑动窗口协议的优点，可以极大地提高网络数据传输速率。SR协议则是在GBN协议下的改进，针对GBN协议的回退N步机制，对于未按序到达的分组进行缓存，确保回退后只重传已发送但未收到ack的分组，避免重传已经接收ack的分组。同时我也认识到了自己的不足：实验前对GBN协议和SR协议概念不清晰，造成实验过程中出错，以后学习过程中要加深对相关概念的联系和对比。 |

GBN\_client.cpp源程序

// GBN\_client.cpp : 定义控制台应用程序的入口点。

#include <stdio.h>

#include <stdlib.h>

#include <WinSock2.h>

#include <time.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 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

const int SEQ\_NUMBER = 17; //设置seq的数量

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

int overflag;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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(buffer, "%d/%d/%d %d:%d:%d",

p->tm\_year + 1900,

p->tm\_mon + 1,

p->tm\_mday,

p->tm\_hour,

p->tm\_min,

p->tm\_sec);

strcpy(ptime, buffer);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* -time 从服务器端获取当前时间

-quit 退出客户端

-testgbn [X] 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() % 101;

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.");

int index;

for (int i = 0; i < SEND\_WIND\_SIZE; ++i) {

index = (i + curAck) % SEQ\_SIZE;

ack[index] = TRUE;

}

totalSeq -= SEND\_WIND\_SIZE;

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 an ack of seq %d \n", index);//从接收方收到的确认收到的序列号

/\*判断数据传输是否完成添加或修改的\*/

if (curAck <= index) {

for (int i = curAck; i <= index; ++i) {

ack[i] = TRUE;

}

printf("\t\tcurAck <= index , totalAck += %d\n", (index - curAck + 1));

totalAck += (index - curAck + 1);

curAck = (index + 1) % SEQ\_SIZE;

}

else if (curAck != index + 1 && curAck > index) {

//ack 超过了最大值，回到了 curAck 的左边

for (int i = curAck; i < SEQ\_SIZE; ++i) {

ack[i] = TRUE;

}

for (int i = 0; i <= index; ++i) {

ack[i] = TRUE;

}

totalAck += (SEQ\_SIZE - curAck + index + 1);

printf("\t\tcurAck > index , totalAck += %d\n", (SEQ\_SIZE - curAck + index + 1));

curAck = index + 1;

}

/\*判断数据传输是否完成添加或修改\*/

}

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];

int length = sizeof(SOCKADDR);

float packetLossRatio = 0.2; //默认包丢失率 0.2

float ackLossRatio = 0.2; //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));

ZeroMemory(buffer, sizeof(buffer));

//将测试数据读入内存

std::ifstream icin;

icin.open("test\_Client.txt");

FILE\* clientwrite = fopen("clientwrite.txt","w+");

char data[1024 \* SEQ\_NUMBER];

ZeroMemory(data, sizeof(data));

icin.read(data, 1024 \* SEQ\_NUMBER);

icin.close();

totalPacket = sizeof(data) / 1024;

printf("totalPacket is ：%d\n\n", totalPacket);

int recvSize;

for (int i = 0; i < SEQ\_SIZE; ++i) {

ack[i] = TRUE;

}

int maxseqorder = 0;//定义当前收到的数据序号，避免重复写入文件

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")) {

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) {

//为什么这里要设置为阻塞模式

//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, "数据传输全部完成！！！\n")) {

fclose(clientwrite);

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];

if (seq > maxseqorder&& seq<= SEQ\_NUMBER) {

//服务器端可能重传

fwrite(&buffer[1], sizeof(char), strlen(buffer) - 1, clientwrite);

maxseqorder = seq;

}

//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;

if (waitSeq == 21) {

waitSeq = 1;

}

//输出数据

printf("\n\n\t%s\n\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, "-time") == 0) {

getCurTime(buffer);

}

else if (!strcmp(cmd, "-testgbn\_Send")) {

overflag = 0;

for (int i = 0; i < SEQ\_SIZE; ++i) {

ack[i] = TRUE;

}

//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

ZeroMemory(buffer, sizeof(buffer));

int recvSize;

int waitCount = 0;

printf("Begain 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 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

//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

//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;

overflag = 0;

sendto(socketClient, "-testgbn\_Send", strlen("-testgbn\_Send") + 1, 0,

(SOCKADDR\*)& addrServer, sizeof(SOCKADDR));

Sleep(100);

recvfrom(socketClient, buffer, BUFFER\_LENGTH, 0, ((SOCKADDR\*)& addrServer), &length);

printf("\n%s\n\n", buffer);

ZeroMemory(buffer, sizeof(buffer));

int iMode = 1; //1：非阻塞，0：阻塞

ioctlsocket(socketClient, FIONBIO, (u\_long FAR\*)& iMode);//非阻塞设置

while (runFlag) {

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), &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;

totalSeq = 0;

waitCount = 0;

stage = 2;

}

}

break;

case 2://数据传输阶段

/\*为判断数据传输是否完成添加或修改的语句\*/

if (seqIsAvailable() && totalSeq <= (totalPacket - 1) && curSeq <= SEQ\_NUMBER) {

//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), &length);

if (recvSize < 0) {

waitCount++;

//20 次等待 ack 则超时重传

if (waitCount > 20) {

timeoutHandler();

printf("\t----totalSeq Now is : %d\n", totalSeq);

waitCount = 0;

}

}

else {

//收到 ack

ackHandler(buffer[0]);

waitCount = 0;

/\*为判断数据传输是否完成添加或修改的语句\*/

if (curAck == totalPacket) {//数据传输完成

overflag = 1;

printf("数据传输全部完成！！！\n");

strcpy(buffer, "数据传输全部完成！！！\n");

sendto(socketClient, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)& addrServer, sizeof(SOCKADDR));

break;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

}

Sleep(500);

break;

}

if (overflag == 1) {

break;

}

}

}

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;

}

GBN\_server.cpp源代码

#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 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 = 1; //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

const int SEQ\_NUMBER = 17; // seq 的数目

//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 overflag;//0表示未结束，1表示结束

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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(buffer, "%d/%d/%d %d:%d:%d",

p->tm\_year + 1900,

p->tm\_mon + 1,

p->tm\_mday,

p->tm\_hour,

p->tm\_min,

p->tm\_sec);

strcpy(ptime, 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;

//step用于判断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;

for (int i = 0; i < SEND\_WIND\_SIZE; ++i) {

index = (i + curAck) % SEQ\_SIZE;

ack[index] = TRUE;

}

totalSeq -= SEND\_WIND\_SIZE;

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);

//如果收到的序列号大于 curAck 则在这之前的报文段全部被接收

if (curAck <= index) {

for (int i = curAck; i <= index; ++i) {

ack[i] = TRUE;

}

printf("\t\tcurAck <= index , totalAck = %d\n", (index - curAck + 1));

curAck = (index + 1) % SEQ\_SIZE;

}

else {

//ack 超过了最大值，回到了 curAck 的左边

for (int i = curAck; i < SEQ\_SIZE; ++i) {

ack[i] = TRUE;

}

for (int i = 0; i <= index; ++i) {

ack[i] = TRUE;

}

printf("\t\tcurAck > index , totalAck += %d\n", (SEQ\_SIZE - curAck + index + 1));

curAck = index + 1;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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() % 101;

if (r <= lossBound) {

return TRUE;

}

return 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");

}

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;

}

/\*双向传输的丢包率\*/

float packetLossRatio = 0.2; //默认包丢失率 0.2

float ackLossRatio = 0.2; //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));

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 \* SEQ\_NUMBER];

FILE\* serverwrite = fopen("serverwrite.txt", "w+");

ZeroMemory(data, sizeof(data));

icin.read(data, 1024 \* SEQ\_NUMBER);

icin.close();

totalPacket = sizeof(data) / 1024;

printf("totalPacket is ：%d\n\n", totalPacket);

int recvSize;

//初始化ack

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(buffer, "Good bye!");

}

else if (strcmp(buffer, "-testgbn") == 0) {

for (int i = 0; i < SEQ\_SIZE; ++i) {

ack[i] = TRUE;

}

//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

ZeroMemory(buffer, sizeof(buffer));

int recvSize;

int waitCount = 0;

printf("Begain 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 大小的状态码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 大小的状态码，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;

overflag = 0;

while (runFlag) {

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;

totalSeq = 0;

waitCount = 0;

stage = 2;

}

}

break;

case 2://数据传输阶段

/\*为判断数据传输是否完成添加或修改的语句\*/

if (seqIsAvailable() && totalSeq <= (totalPacket - 1) && curSeq <= SEQ\_NUMBER) {

//发送给客户端的序列号从 1 开始

buffer[0] = curSeq + 1;

ack[curSeq] = FALSE;//将ACK变为待确认的状态

//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 == -1) {

waitCount++;

//20 次等待 ack 则超时重传

if (waitCount > 20) {

timeoutHandler();

printf("\t----totalSeq Now is : %d\n", totalSeq);

waitCount = 0;

}

}

else {

//收到 ack，定时器将被重新启动

ackHandler(buffer[0]);

waitCount = 0;

/\*为判断数据传输是否完成添加或修改的语句\*/

if (curAck - 1 == totalPacket) {//数据传输完成

overflag = 1;

printf("数据传输全部完成！！！\n");

strcpy(buffer, "数据传输全部完成！！！\n");

sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)& addrClient, sizeof(SOCKADDR));

break;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

}

Sleep(500);

break;

}

if (overflag == 1) {

break;

}

}

}

/\* 双向数据传输 \*/

else if (strcmp(buffer, "-testgbn\_Send") == 0) {

iMode = 0; //1：非阻塞，0：阻塞

ioctlsocket(sockServer, FIONBIO, (u\_long FAR\*)& iMode);//非阻塞设置

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 maxseqorder = 0;

sendto(sockServer, "-testgbn\_Send", strlen("-testgbn\_Send") + 1, 0,

(SOCKADDR\*)& addrClient, sizeof(SOCKADDR));

while (true) {

//等待 server 回复设置 UDP 为阻塞模式

int len = sizeof(SOCKADDR);

recvfrom(sockServer, buffer, BUFFER\_LENGTH, 0, (SOCKADDR\*)& addrClient , & len);

if (!strcmp(buffer, "数据传输全部完成！！！\n")) {

fclose(serverwrite);

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];

if (seq > maxseqorder && seq <= SEQ\_NUMBER) {

//服务器端可能重传

fwrite(&buffer[1], sizeof(char), strlen(buffer) - 1, serverwrite);

maxseqorder = seq;

}

//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;

if (waitSeq == 21) {

waitSeq = 1;

}

//输出数据

printf("\n\n\t%s\n\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);

}

}

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;

}

SRClient.cpp源代码

#include <stdlib.h>

#include <WinSock2.h>

#include <time.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 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

const int SEQ\_NUMBER = 17;

//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

char dataBuffer[SEQ\_SIZE][BUFFER\_LENGTH];

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

int finish;//标志位：数据传输是否完成（finish=1->数据传输已完成）

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* -time 从服务器端获取当前时间

-quit 退出客户端

-testsr [X] 测试 SR 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

-testsr\_Send 双向数据传输

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void printTips() {

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("| -time to get current time |\n");

printf("| -quit to exit client |\n");

printf("| -testsr [X] [Y] to test the sr (Receive message from the Server)|\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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(buffer, "%d/%d/%d %d:%d:%d",

p->tm\_year + 1900,

p->tm\_mon + 1,

p->tm\_mday,

p->tm\_hour,

p->tm\_min,

p->tm\_sec);

strcpy(ptime, buffer);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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() % 101;

if (r <= lossBound) {

return TRUE;

}

return FALSE;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: seqRecvAvailable

// FullName: seqRecvAvailable

// Access: public

// Returns: bool

// Qualifier: 当前收到的序列号 recvSeq 是否在可收范围内

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BOOL seqRecvAvailable(int recvSeq) {

int step;

int index;

index = recvSeq - 1;

step = index - curAck;

step = step >= 0 ? step : step + SEQ\_SIZE;

//序列号是否在当前发送窗口之内

if (step <= 0&&step >= SEND\_WIND\_SIZE) {

return FALSE;

}

return TRUE;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: timeoutHandler

// FullName: timeoutHandler

// Access: public

// Returns: void

// Qualifier: 超时重传处理函数，哪个没收到 ack ，就要重传哪个

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void timeoutHandler() {

printf("Timer out error.");

int index;

if (totalSeq == totalPacket) {//之前发送到了最后一个数据包

if (curSeq > curAck) {

totalSeq -= (curSeq - curAck);

}

else if (curSeq < curAck) {

totalSeq -= (curSeq - curAck + 20);

}

}

else {//之前没发送到最后一个数据包

totalSeq -= SEND\_WIND\_SIZE;

}

curSeq = curAck;

}

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 -testsr [X] [Y] 测试 SR 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 = 0.2; //默认包丢失率 0.2

float ackLossRatio = 0.2; //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\_Client.txt");

char data[1024 \* SEQ\_NUMBER];

ZeroMemory(data, sizeof(data));

//icin.read(data,1024 \* 113);

//icin.read(data,1024 \* 4);

icin.read(data, 1024 \* SEQ\_NUMBER);

icin.close();

totalPacket = sizeof(data) / 1024;

printf("totalPacket is ：%d\n\n", totalPacket);

int recvSize;

finish = 0;

for (int i = 0; i < SEQ\_SIZE; ++i) {

ack[i] = FALSE;

}

finish = 0;

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 UDP 可靠文件传输

if (!strcmp(cmd, "-testsr")) {

for (int i = 0; i < SEQ\_SIZE; ++i) {

ack[i] = FALSE;

}

printf("%s\n", "Begin to test SR 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;

finish = 0;

BOOL b;

curAck = 0;

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 next;

sendto(socketClient, "-testsr", strlen("-testsr") + 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);

/\*判断数据传输是否完成添加或修改\*/

if (!strcmp(buffer, "数据传输全部完成！！！\n")) {

finish = 1;

break;

}

/\*判断数据传输是否完成添加或修改\*/

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 (seqRecvAvailable(seq)) {

recvSeq = seq;

ack[seq - 1] = TRUE;//TRUE表示已经接收，FALSE表示未接收

ZeroMemory(dataBuffer[seq - 1], sizeof(dataBuffer[seq - 1]));

strcpy(dataBuffer[seq - 1], &buffer[1]);

buffer[0] = recvSeq;

buffer[1] = '\0';

int tempt = curAck;

//如果是按序，则接收窗口右移

if (seq - 1 == curAck) {

for (int i = 0; i < SEQ\_SIZE; i++) {

next = (tempt + i) % SEQ\_SIZE;

if (ack[next]) {//移动至下一个

//输出数据

printf("\n\n\t\tACK SEQ: %d\n%s\n\n", (next + 1) % SEQ\_SIZE, dataBuffer[next]);

curAck = (next + 1) % SEQ\_SIZE;

}

else {//遇到下一个未接收的窗口为止

break;

}

}

}

}

else {//如果不是期待的包，忽略

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);

}

}

else if (strcmp(cmd, "-time") == 0) {

getCurTime(buffer);

}

/\*判断数据传输是否完成添加或修改\*/

if (finish == 1) {

printf("数据传输全部完成！！！\n\n");

printTips();

continue;

}

/\*判断数据传输是否完成添加或修改\*/

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;

}

SRServer.cpp源代码

#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 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

const int SEQ\_NUMBER = 17;

//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

char dataBuffer[SEQ\_SIZE][BUFFER\_LENGTH];

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 finish; //标志位：数据传输是否完成（finish=1->数据传输已完成）

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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(buffer, "%d/%d/%d %d:%d:%d",

p->tm\_year + 1900,

p->tm\_mon + 1,

p->tm\_mday,

p->tm\_hour,

p->tm\_min,

p->tm\_sec);

strcpy(ptime, buffer);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: seqIsAvailable

// FullName: seqIsAvailable

// Access: public

// Returns: bool

// Qualifier: 当前序列号 curSeq 是否可用

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int 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 0;//不在窗口内将不再发送

}

return 1;//可以发送

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: timeoutHandler

// FullName: timeoutHandler

// Access: public

// Returns: void

// Qualifier: 超时重传处理函数

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void timeoutHandler() {

printf("Timer out error.");

int index;

/\*为判断数据传输是否完成添加或修改的语句\*/

if (totalSeq == totalPacket) {//之前发送到了最后一个数据包

if (curSeq > curAck) {

totalSeq -= (curSeq - curAck);

}

else if (curSeq < curAck) {

totalSeq -= (curSeq - curAck + SEQ\_SIZE);

}

}

else {//之前没发送到最后一个数据包

totalSeq -= SEND\_WIND\_SIZE;

}

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 seq %d \n", index + 1); //从接收方收到的确认收到的序列号

int all;

int next;

int add;

all = 1;

//如果顺序，窗口右移至下一个未ACK的对象，即seq = curack + 1

if (curAck == index) {

add = 1;

ack[index] = FALSE;

curAck = (index + 1) % SEQ\_SIZE;

for (int i = 1; i < SEQ\_SIZE; i++) {

next = (i + index) % SEQ\_SIZE;

if (ack[next] == TRUE) {

ack[next] = FALSE;

curAck = (next + 1) % SEQ\_SIZE;

totalSeq++;

++curSeq;

curSeq %= SEQ\_SIZE;

}

else {

break;

}

}

printf("\t\tcurAck == index\n");

}

//在滑动窗口内进行单个ACK

else if (curAck < index && index - curAck + 1 <= SEND\_WIND\_SIZE) {

//要保证是要接受的消息（在滑动窗口内）

printf("\t\tcurAck < index\n");

ack[index] = TRUE;//用true表示非按序单个ack已经接收的状态

}

else if (SEQ\_SIZE - curAck + index + 1 <= SEND\_WIND\_SIZE && curAck > index) {

//要保证是要接受的消息（在滑动窗口内）

printf("\t\tcurAck > index\n");

ack[index] = TRUE;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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() % 101;

if (r <= lossBound) {

return TRUE;

}

return 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");

}

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;

}

float packetLossRatio = 0.2; //默认包丢失率 0.2

float ackLossRatio = 0.2; //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));

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 \* SEQ\_NUMBER];

ZeroMemory(data, sizeof(data));

icin.read(data, 1024 \* SEQ\_NUMBER);

icin.close();

totalPacket = sizeof(data) / 1024;

printf("totalPacket is ：%d\n\n", totalPacket);

int recvSize;

for (int i = 0; i < SEQ\_SIZE; ++i) {

ack[i] = FALSE;

}

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) {

printf("收到 -time");

getCurTime(buffer);

}

else if (strcmp(buffer, "-quit") == 0) {

strcpy(buffer, "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] = FALSE;

}

ZeroMemory(buffer, sizeof(buffer));

int recvSize;

int waitCount = 0;

finish = 0;

printf("Begain to test SR 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 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

//客户端收到 205 之后回复一个 200 大小的状态码，表示客户端备好了，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;

while (runFlag) {

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;

totalSeq = 0;

waitCount = 0;

finish = 0;

stage = 2;

}

}

break;

case 2://数据传输阶段

//窗口内且可以发送的序列号

if (seqIsAvailable() == 1 && totalSeq <= (totalPacket - 1)) {

//totalSeq<=(totalPacket-1)：未传到最后一个数据包

//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->现在此代码已经实现了ok

//为简化过程此处并未实现->现在此代码已经实现了ok

memcpy(&buffer[1], data + 1024 \* totalSeq, 1024);

printf("send a packet with a seq of : %d \t totalSeq now is : %d\n", curSeq + 1, totalSeq + 1);

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();

printf("\t----totalSeq Now is : %d\n", totalSeq);

waitCount = 0;

}

}

else {

//收到 ack

ackHandler(buffer[0]);

waitCount = 0;

/\*为判断数据传输是否完成添加或修改的语句\*/

if (curAck == totalPacket) {//数据传输完成

finish = 1;

break;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

}

Sleep(500);

break;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

if (finish == 1) {

printf("数据传输全部完成！！！\n");

strcpy(buffer, "数据传输全部完成！！！\n");

sendto(sockServer, buffer, strlen(buffer) + 1, 0, (SOCKADDR\*)& addrClient, sizeof(SOCKADDR));

break;

}

/\*为判断数据传输是否完成添加或修改的语句\*/

}

}

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;

}