可靠传输协议3-1实验报告

学号：2012307 姓名：聂志强 专业：信息安全

一. 实验要求

利用数据报套接字在用户空间实现面向连接的可靠数据传输，功能包括：建立连接、差错检测、确认重传等。流量控制采用停等机制，完成给定测试文件的传输。

二. 功能实现

1. 基本功能

- **建立连接：**

  实现类似于 TCP 的三次握手、四次挥手过程

- **差错检测：**

  利用校验和进行差错检测，发送端将数据报看成 16 位整数序列，将整个数据报相加然后取反写入校验和域段，接收端将数据报用 0 补齐为 16 位整数倍，然后相加求和，如果计算结果为全 1，没有检测到错误；否则说明数据报存在差错。

- **流量控制（停等机制）：**

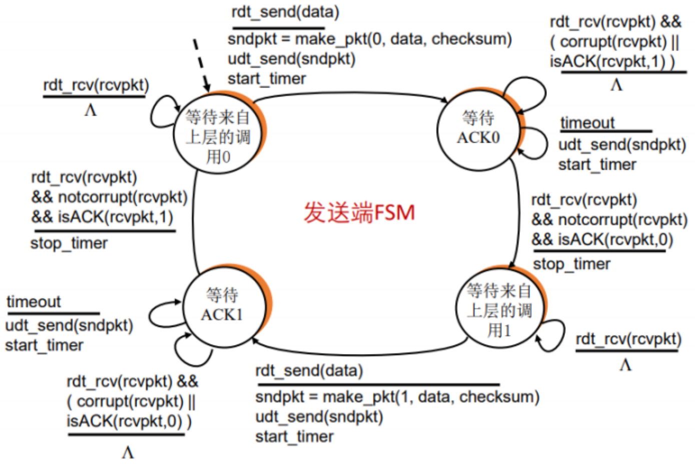
  采用停等协议，发送端发送一个分组，然后等待接收端响应

- **日志输出：**

  打印出三次握手四次挥手过程、序列号、确认序列号、数据大小、时延、吞吐率。

- **超时重传**

  采用 rdt3.0 机制，由于通道既可能有差错，又可能有丢失，所以我们考虑利用rdt3.0 机制实现可靠数据传输。



2. 附加功能

- **MSS协商**

  双方将会协商MSS，选择双方需求的最小MSS作为通信MSS。

- **多线程**

  为了兼容后期拥塞控制的实验，本次代码在设计上采用多线程控制，由发送线程和接收线程互相配合完成发送或者接收的任务。

- **异常检测**

  1. 断开方式与TCP基本相同，为了保证通信状态正常，在没有任何信息需要发送时，双方也会在固定的时间内发送一个小数据包，以检测连接状态和报告自身情况。当数据包出现10次连续丢失时，双方将认为通信异常，自动启动断开程序。

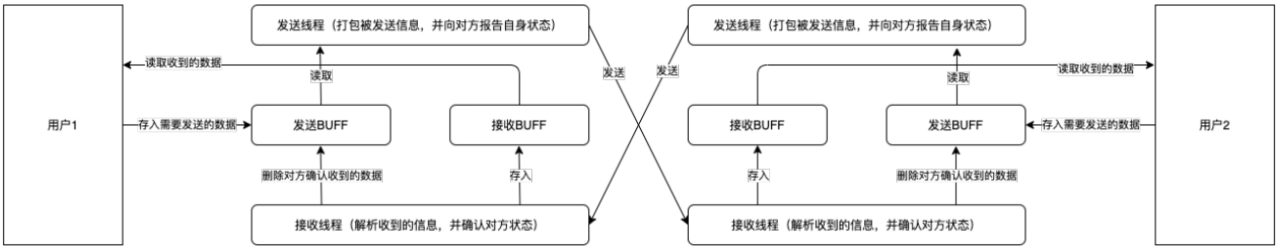
三. 程序架构设计

1. 协议头结构

```
-------------------------
|   本机端口   |   接收端口   |
-------------------------
|          序号          |
-------------------------
|          确认号         |
-------------------------
|首部长度|标志|   接收窗口    |
-------------------------
|     校验和    |紧急数据指针|
-------------------------
|      报文最大长度MSS      |
-------------------------
|----------数据---------|
```

2. 整体实现框架

当用户需要发送信息时，将会把被发送的信息放到一个发送缓存中，发送线程将逐步读取发送的信息，然后进行可靠传输。同样，对于接收线程，线程将接收到的信息放入接收缓存中，当用户需要接收信息时，直接查看接收缓存里是否有内容即可，接收缓存有固定的大小。当没有任何信息需要传输时，线程也会不断发送一个小信息包，在报告连接正常的同时，为后期检查拥塞情况和网络状态保留。



四. 核心代码

1. Socket基础设置

```cpp
UDP::UDP(const char* host, unsigned short port, unsigned short local_port, int mss, int bufsize, unsigned short window_size) :
    MSS_default(mss), local_port(local_port), host(host), port(port), isconnect(false), window_size(window_size), window_size_default(window_size),
    MSS(mss), recvbuf(deque<recv_pkg>(ceil(bufsize / (float)mss))), bufsize(bufsize) {
    // 创建Socket
    WORD wVersionRequested = MAKEWORD(2, 2);
    WSADATA wsaData;
    WSAStartup(wVersionRequested, &wsaData);
    //ipv4的地址类型；数据报的服务类型；Protocol（协议）为UDP
    this->sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    // 目的地址
    sockaddr_in* temp_addr = new sockaddr_in;
    temp_addr->sin_family = AF_INET;
    temp_addr->sin_port = htons(port);
    inet_pton(AF_INET, host, &(temp_addr->sin_addr.s_addr));
    this->addr = (sockaddr*)temp_addr;

    // 源地址
    temp_addr = new sockaddr_in;
    temp_addr->sin_family = AF_INET;
    temp_addr->sin_port = htons(local_port);
    inet_pton(AF_INET, "127.0.0.1", &(temp_addr->sin_addr.s_addr));
    this->local_addr = (sockaddr*)temp_addr;

    // 服务器端将本地地址绑定到一个Socket
    bind(this->sock, this->local_addr, sizeof(sockaddr));

    // 初始化锁
    InitializeCriticalSection(&(this->sendbuf_lock));
}
```

2. 建立连接(包含三次握手+MSS协商)

◦ 客户端

```cpp
bool UDP::connect() {
    if (this->isconnect) return true;
    if (this->tcp_runner_recv) WaitForSingleObject(this->tcp_runner_recv, INFINITE);
    if (this->tcp_runner_send) WaitForSingleObject(this->tcp_runner_send, INFINITE);

    cout << "正在连接..." << endl;
    cout << endl;
    cout << "=================================================" << endl;
    reset();
    unsigned char flag = 0;
    set_flag_syn(&flag, true); // 置位SYN握手信号
    if (sendmeg("", flag) == -1) // 发送消息失败，return false
        return false;
    cout << "[SYN]" << endl; // 握手信号提示

    unsigned char* buf = new unsigned char[this->head_length];
    int length = recvmeg(buf, this->head_length, CONNECT_RECV_TIMEOUT);

    if (length == -1 || length < this->head_length || !get_flag_syn(get_flag(buf))|| !get_flag_ack(get_flag(buf))) {
        cout << " [SYNACK] WRONG!" << endl;
        delete[] buf;
        return false;
    }
    cout << "[SYN & ACK] -> " << " [SEQ] " << get_seq(buf) << " [ACK] " << get_ack(buf) << " [SYN_FLAG] " << get_flag_syn(get_flag(buf)) << " [ACK_FLAG] " << get_flag_ack(get_flag(buf)) << endl;
    // 设置MSS和窗口大小
    this->MSS = get_MSS(buf);
    this->window_size = get_window_size(buf);
    recvbuf.resize(ceil(this->bufsize / (float)(this->MSS)));
    this->seq = get_ack(buf);
    this->ack = get_seq(buf) + 1;
    delete[] buf;

    set_flag_syn(&flag, false);
    set_flag_ack(&flag, true);

    if (sendmeg("", flag) == -1) {
        cout << "[ACK] WRONG!" << endl;
        return false;
    }
    cout << "[ACK] " << endl;
    cout<<"连接成功！ " << endl;
    cout << "=================================================" << endl;
    cout << endl;
    this->isconnect = true; // 建立连接成功，建连标志置位
    // 接收线程和发送线程
    this->tcp_runner_send = CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)Send_thread, (LPVOID)this, 0, 0);
    this->tcp_runner_recv = CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)Recv_thread, (LPVOID)this, 0, 0);
    return true;
}
```

- **服务器端**

```cpp
bool UDP::accept() {
    if (this->isconnect) return true;
    if (this->tcp_runner_send) WaitForSingleObject(this->tcp_runner_send, INFINITE);
    if (this->tcp_runner_recv) WaitForSingleObject(this->tcp_runner_recv, INFINITE);

    unsigned char* buf = new unsigned char[this->head_length]; //head_length代表协议头长度（24字节）
    int length = -1;
    unsigned char flag = 0;
    while (true) {
        reset(); //重置参数
        flag = 0;
        cout << "等待连接..." << endl;
        cout << endl;
        cout << "=================================================" << endl;

        length = recvmeg(buf, this->head_length);
        // 检测SYN
        if (length == -1 || length < this->head_length || !get_flag_syn(get_flag(buf)))
            continue;

        // 设置MSS
        if (get_MSS(buf) < this->MSS) {
            this->MSS = get_MSS(buf);
            // 当MSS改变时，同时需要resize改变deque双端队列中元素个数
            recvbuf.resize(ceil(this->bufsize / (float)(this->MSS)));
        }
        // 协商窗口大小
        if (get_window_size(buf) < this->window_size) {
            this->window_size = get_window_size(buf);
        }

        // 三次握手
        cout << "[SYN] -> " <<  " [SEQ] " << get_seq(buf) << " [ACK] " << get_ack(buf) << " [SYN_FLAG] " << get_flag_syn(get_flag(buf)) << " [ACK_FLAG] " << get_flag_ack(get_flag(buf)) << endl;

        // ack=seq+1
        this->ack = get_seq(buf) + 1;

        // 设置SYN|ACK建连标志
        set_flag_syn(&flag, true);
        set_flag_ack(&flag, true);
        if (sendmeg("", flag) == -1) {
            cout << "[SYN & ACK] WRONG!" << endl;
            continue;
        }
        cout << "[SYN & ACK] " << endl;

        length = recvmeg(buf, this->head_length, CONNECT_RECV_TIMEOUT);
        flag = get_flag(buf);


        if (length == -1 || length < this->head_length || !get_flag_ack(get_flag(buf)) || get_ack(buf) != this->seq + 1 || get_seq(buf) != this->ack)
        {
            cout << "[ACK] WRONG!" << endl;
            continue;
        }
        this->seq = get_ack(buf);
        cout << "[ACK] -> " << "[SEQ] " << get_seq(buf) << "[ACK] " << get_ack(buf) << "[SYN_FLAG] " << get_flag_syn(get_flag(buf)) << "[ACK_FLAG] " << get_flag_ack(get_flag(buf)) << endl;
        cout << "连接成功！" << endl;
        cout << "=================================================" << endl;
        cout << endl;
        break;
    }
    this->isconnect = true;
    cout.flush();
    delete[] buf;
    // 接收线程
    this->tcp_runner_send = CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)Send_thread, (LPVOID)this, 0, 0);
    // 发送线程
    this->tcp_runner_recv = CreateThread(NULL, NULL, (LPTREAD_START_ROUTINE)Recv_thread, (LPVOID)this, 0, 0);
    return true;
}
```

3. 封装协议头

```
bool UDP::generate_meg_head(unsigned char* message, int length, unsigned char flag, int* seq_spec) {
    if (length < this->head_length)
        return false;

    // 源端口
    message[0] = (unsigned char)(this->local_port >> 8);
    message[1] = (unsigned char)this->local_port;
    // 目的端口
    message[2] = (unsigned char)(this->port >> 8);
    message[3] = (unsigned char)this->port;

    // seq序列号
    if (seq_spec) {
        message[4] = (unsigned char)((*seq_spec) >> 24);
        message[5] = (unsigned char)((*seq_spec) >> 16);
        message[6] = (unsigned char)((*seq_spec) >> 8);
        message[7] = (unsigned char)(*seq_spec);
    }
    else {
        message[4] = (unsigned char)(this->seq >> 24);
        message[5] = (unsigned char)(this->seq >> 16);
        message[6] = (unsigned char)(this->seq >> 8);
        message[7] = (unsigned char)(this->seq);
    }

    // ACK确认序列号
    message[8] = (unsigned char)(ack >> 24);
    message[9] = (unsigned char)(ack >> 16);
    message[10] = (unsigned char)(ack >> 8);
    message[11] = (unsigned char)ack;

    // head_length 8 | flag 8
    // 协议头长度head_length为24
    message[12] = this->head_length << 2;
    message[13] = flag;

    // 窗口大小
    message[14] = (unsigned char)(window_size >> 8);
    message[15] = (unsigned char)window_size;

    // 校验和：初始化为0
    message[16] = 0;
    message[17] = 0;

    // MSS最大段长度
    message[20] = (unsigned char)(MSS >> 24);
    message[21] = (unsigned char)(MSS >> 16);
    message[22] = (unsigned char)(MSS >> 8);
    message[23] = (unsigned char)MSS;

    // 生成校验和：所有数据2字节求和取反，不足2字节补零
    unsigned short val = 0;
    for (int i = 0; i < length / 2; i++)
        val += (unsigned short)message[i * 2] << 8 | (unsigned short)message[i * 2 + 1];
    if (length % 2) val += (unsigned short)message[length - 1] << 8;
    val = ~val;

    // 存入校验和
    message[16] = (unsigned char)(val >> 8);
    message[17] = (unsigned char)val;
    return true;
}
```

4. 获得协议头信息

```
bool get_flag_cwr(unsigned char flag) { return flag & (unsigned char)1 << 7; }
bool get_flag_ece(unsigned char flag) { return flag & (unsigned char)1 << 6; }
bool get_flag_over(unsigned char flag) { return flag & (unsigned char)1 << 5; }
bool get_flag_ack(unsigned char flag) { return flag & (unsigned char)1 << 4; }
bool get_flag_end(unsigned char flag) { return flag & (unsigned char)1 << 3; }
bool get_flag_rst(unsigned char flag) { return flag & (unsigned char)1 << 2; }
bool get_flag_syn(unsigned char flag) { return flag & (unsigned char)1 << 1; }
bool get_flag_fin(unsigned char flag) { return flag & (unsigned char)1; }
```

5. 差错检验

在按照协议设计格式生成协议头函数 generate_meg_head 里，我们将所有数据 2 字节求和取反，不足 2 字节补零，生成校验和；在差错检测函数 check_message 里，我们对接收的数据报的 16bits 数组进行求和，如果结果全 1，则数据报正确；否则数据报存在错误

```
bool UDP::check_message(unsigned char* message, int length) {
    unsigned short val = 0;
    // 所有数据2字节求和
    for (int i = 0; i < length / 2; i++)
        val += (unsigned short)message[i * 2] << 8 | (unsigned short)message[i * 2 + 1];
    if (length % 2) val += (unsigned short)message[length - 1] << 8;
    // 对接收的数据报的 16bits 数组进行求和，如果结果全 1，则数据报正确；否则数据报存在错误。
    return !(unsigned short)(val + 1);
}
```

6. 发送线程

在发送线程中，当发送缓冲区不为空时，读取缓冲区数据打包数据报并发送，设置 END 标识来标记是否为最后一个数据报，打印相应的序列号信息。

```
// 在发送线程中，当发送缓冲区不为空时，读取缓冲区数据打包数据报并发送，设置 END 标识来标记是否为最后一个数据报，打印相应的序列号信息。
DWORD WINAPI Send_thread(LPVOID s) {
    UDP* cls = (UDP*)s;
    unsigned char flag;
    unsigned long long last_stamp = GetTickCount64(); // 计时开始
    while (cls->isconnect) {
        if (!cls->immsend && GetTickCount64() - last_stamp < CONNECT_RECV_TIMEOUT * 0.5)
        {
            Sleep(0);
            continue;
        }
        last_stamp = GetTickCount64();
        cls->immsend = false;
        flag = 0;
        cls->set_flag_end(&flag, true); // 初始化为最后一个数据报
        cls->set_flag_ack(&flag, true); // ACK有效
        string sendcontent;
        // send
        //加锁 接下来的代码处理过程中不允许其他线程进行操作，除非遇到LeaveCriticalSection
        EnterCriticalSection(&(cls->sendbuf_lock));
        int seq_temp = cls->seq; // 发送seq
        if (cls->sendbuf.size()) { // 发送缓冲区不为空时
            string& sendpkg = *(cls->sendbuf).begin();
            // 发送缓冲区 > 数据报大小（MSS * window_size），只读取数据报大小的数据并标记非最后一个数据报
            // 否则直接读取全部发送缓冲区内容并标识为最后一个数据报
            if (sendpkg.length() > cls->MSS * cls->window_size) {
                sendcontent.assign(sendpkg, 0, cls->MSS * cls->window_size);
                cls->set_flag_end(&flag, false);
            }
            else sendcontent = sendpkg;
        }
        //解锁 到EnterCriticalSection之间代码资源已经释放了，其他线程可以进行操作
        LeaveCriticalSection(&(cls->sendbuf_lock));
        if (sendcontent.length() == 0) {
            cls->sendmeg(sendcontent, flag);
        }
        else {
            unsigned char flag_copy = flag;
            for (int i = 0; i < sendcontent.length(); i += cls->MSS) {
                flag = flag_copy;
                if (i + cls->MSS < sendcontent.length())
                    cls->set_flag_end(&flag, false);
                cls->sendmeg(sendcontent.substr(i, ((i + cls->MSS) >= sendcontent.length() ? sendcontent.length() - i : cls->MSS)), flag, &seq_temp);
                // 每次seq+mss表示发送序列号
                seq_temp += cls->MSS;
                cout << "Send: " << sendcontent.length() << " [SEQ] " << seq_temp << " [ACK] " << cls->ack << endl;
            }
        }
        Sleep(0);
    }
    cls->sendbuf.clear();
    return 0;
}
```

7. 接收线程

接收线程中，当超过设定的时限 CONNECT_RECV_TIMEOUT，则设置 immsend 立即重传。当超过 10 次丢失，通信异常，自动断开连接，或者收到断开请求 FIN 标志置位也断开连接，利用四次挥手断开连接。当接收序列号与期待的序列号不相等时，则标记立即重传。在接受线程中，处理接收数据，对数据报拆包去掉数据报头，并将数据放入接收缓冲区。由于使用了共享的临时缓冲区，为保证线程顺序读取数据，防止冲突，设置锁机制进行保护。

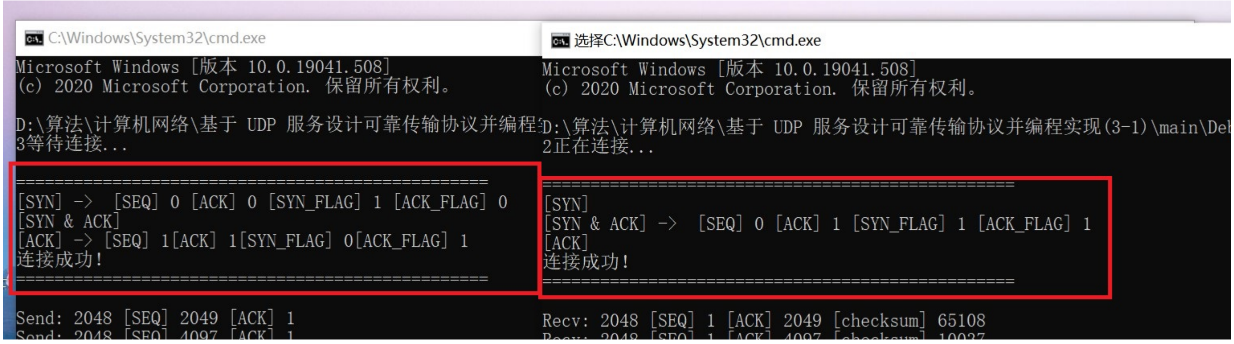
```
//接收线程
DWORD WINAPI Recv_thread(LPVOID s) {
    UDP* cls = (UDP*)s;
    unsigned char* buf = new unsigned char[cls->MSS + cls->head_length];
    int timeout_round = 0;
    while (cls->isconnect) {
        // 超时重传：当超过时限CONNECT_RECV_TIMEOUT，则设置immsend重传分组
        // lenth包括协议头长度+数据段长度
        int length = cls->recvmeg(buf, cls->MSS + cls->head_length, CONNECT_RECV_TIMEOUT);
        // 10次丢失，通信异常，自动断开
        if (length == -1) {
            timeout_round++;
            if (timeout_round >= cls->autoclose_tcp_loop) break;
            Sleep(0);
        }
        else if (length >= cls->head_length) {
            // 重新设置丢失次数
            timeout_round = 0;
            // FIN标志置位，断开连接
            if (cls->get_flag_fin(cls->get_flag(buf))) {
                cout << "Closing..." << endl;
                break;
            }
            EnterCriticalSection(&(cls->sendbuf_lock));
            // 接收的ACK与接收线程期待的序列号SEQ不相等
            // 当重复ACK时，标记立即重传
            if (cls->seq != cls->get_ack(buf)) {

                // cout << "ACK!" << cls->get_ack(buf) << " " << cls->seq << endl;
                cls->sendbuf.begin()->assign(*(cls->sendbuf.begin()), cls->get_ack(buf) - cls->seq);
                if (cls->sendbuf.begin()->length() == 0)
                    cls->sendbuf.pop_front();
                cls->seq = cls->get_ack(buf);
                cls->immsend = true;
            }
            LeaveCriticalSection(&(cls->sendbuf_lock));

            // 处理接收数据
            if (length > cls->head_length) {
                if (cls->ack == cls->get_seq(buf)) {
                    if (cls->recvbuf.max_size() > cls->recvbuf.size()) {
                        // 拆包：去掉数据报头
                        unsigned char* temp = new unsigned char[length - cls->head_length];
                        memcpy(temp, buf + cls->head_length, length - cls->head_length);
                        // 将数据放入接收缓冲区（双端队列尾部增加数据）
                        cls->recvbuf.push_back({ length - cls->head_length, cls->get_flag_end(cls->get_flag(buf)), temp });
                        cls->ack = cls->get_seq(buf) + length - cls->head_length;
                    }
                }
                cout << "Recv: " << length - cls->head_length << " [SEQ] " << cls->seq << " [ACK] " << cls->ack << " [checksum] " << cls->get_checksum(buf) << endl;
                cls->immsend = true;
            }
        }
    }

    // 四次挥手断开连接
    unsigned char flag = 0;
    if (cls->isconnect) { // 接收端主动断开连接
        cls->isconnect = false;
        cls->set_flag_end(&flag, true);
        cls->set_flag_fin(&flag, true);
        cls->set_flag_ack(&flag, true);
        cls->sendmeg("", flag);
        cout << "四次挥手: [ACK] -> [FIN]";
        cls->recvmeg(buf, cls->MSS + cls->head_length, CONNECT_RECV_TIMEOUT);
        cout << " -> [ACK]" << endl;
    }
    else { // 发送端
        cls->set_flag_end(&flag, true);
        cls->set_flag_fin(&flag, true);
        cls->set_flag_ack(&flag, false);
        cout << "Close: [FIN]";
        cls->sendmeg("", flag);

        cls->recvmeg(buf, cls->MSS + cls->head_length, CONNECT_RECV_TIMEOUT);
        cout << " -> [ACK] -> [FIN]";

        flag = 0;
        cls->set_flag_end(&flag, true);
        cls->set_flag_fin(&flag, true);
        cls->set_flag_ack(&flag, true);
        cls->sendmeg("", flag);
        cout << " -> [ACK]" << endl;
    }
    delete[] buf;
    for (auto& i : cls->recvbuf) delete[] i.buf;
    cls->recvbuf.clear();

    if (timeout_round >= cls->autoclose_tcp_loop) cout << "Time out!" << endl;
    cout << "断开连接!" << endl;
    return 0;
}
```

8. 与缓冲区交互

```
// 将数据放入发送缓冲区
bool UDP::send(string data) {
    if (!this->isconnect)
        return false;
    // sendbuf 类型为 list<string>
    this->sendbuf.push_back(data);
    return true;
}

// 从接收缓冲区读取数据
string UDP::recv() {
    string res;
    while (this->isconnect) {
        if (this->recvbuf.size() == 0) {
            Sleep(0);
            continue;
        }
        bool isend = false;
        while (!isend && this->recvbuf.size()) {
            auto buf = *(this->recvbuf.begin());
            isend = buf.isend;
            res += string((const char*)buf.buf, buf.size);
            delete[] buf.buf;
            // 删除双端队列buf中最前一个元素
            this->recvbuf.pop_front();
        }
        if (isend) break;
    }
    return res;
}
```

五. 程序演示

1. 三次握手建立连接

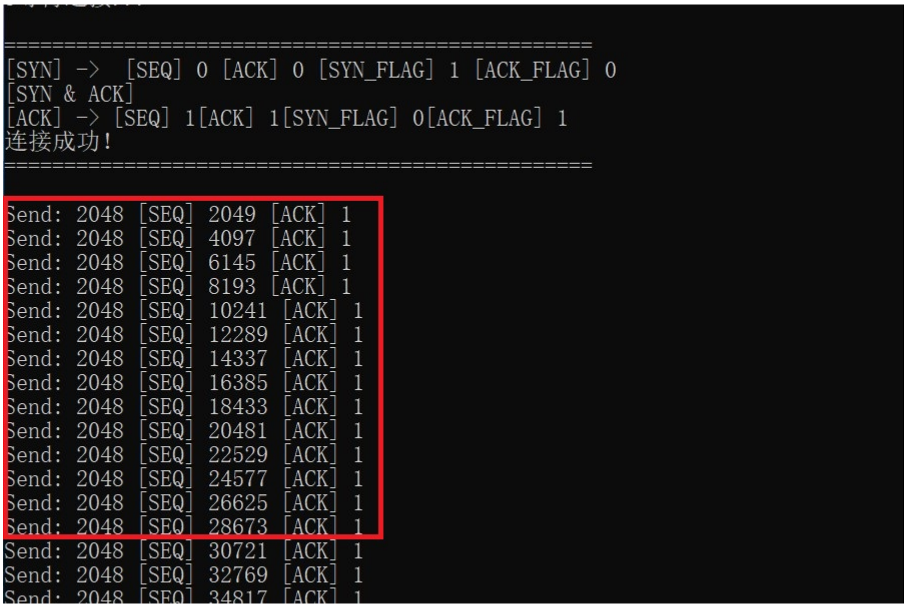


```
Microsoft Windows [版本 10.0.19041.508]
(c) 2020 Microsoft Corporation. 保留所有权利。

D:\算法\计算机网络\基于 UDP 服务设计可靠传输协议并编程
3等待连接...

=================================================
[SYN] ->  [SEQ] 0 [ACK] 0 [SYN_FLAG] 1 [ACK_FLAG] 0
[SYN & ACK]
[ACK] -> [SEQ] 1[ACK] 1[SYN_FLAG] 0[ACK_FLAG] 1
连接成功!
=================================================

Send: 2048 [SEQ] 2049 [ACK] 1
Send: 2048 [SEQ] 4097 [ACK] 1
```

```
Microsoft Windows [版本 10.0.19041.508]
(c) 2020 Microsoft Corporation. 保留所有权利。

D:\算法\计算机网络\基于 UDP 服务设计可靠传输协议并编程实现(3-1)\main\De
2正在连接...

=================================================
[SYN]
[SYN & ACK] ->  [SEQ] 0 [ACK] 1 [SYN_FLAG] 1 [ACK_FLAG] 1
[ACK]
连接成功!
=================================================

Recv: 2048 [SEQ] 1 [ACK] 2049 [checksum] 65108
Recv: 2048 [SEQ] 1 [ACK] 4097 [checksum] 10037
```

2. 发送端发送报文

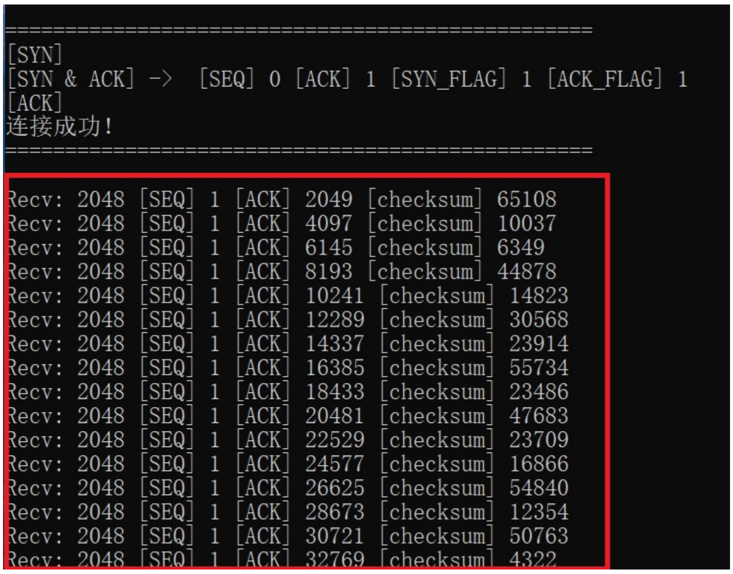
可以观察收到第一列2048代表MSS大小，SEQ代表序列号，即发送报文段的字节流编号



```
=================================================
[SYN] ->  [SEQ] 0 [ACK] 0 [SYN_FLAG] 1 [ACK_FLAG] 0
[SYN & ACK]
[ACK] -> [SEQ] 1[ACK] 1[SYN_FLAG] 0[ACK_FLAG] 1
连接成功!
=================================================

Send: 2048 [SEQ] 2049 [ACK] 1
Send: 2048 [SEQ] 4097 [ACK] 1
Send: 2048 [SEQ] 6145 [ACK] 1
Send: 2048 [SEQ] 8193 [ACK] 1
Send: 2048 [SEQ] 10241 [ACK] 1
Send: 2048 [SEQ] 12289 [ACK] 1
Send: 2048 [SEQ] 14337 [ACK] 1
Send: 2048 [SEQ] 16385 [ACK] 1
Send: 2048 [SEQ] 18433 [ACK] 1
Send: 2048 [SEQ] 20481 [ACK] 1
Send: 2048 [SEQ] 22529 [ACK] 1
Send: 2048 [SEQ] 24577 [ACK] 1
Send: 2048 [SEQ] 26625 [ACK] 1
Send: 2048 [SEQ] 28673 [ACK] 1
Send: 2048 [SEQ] 30721 [ACK] 1
Send: 2048 [SEQ] 32769 [ACK] 1
Send: 2048 [SEQ] 34817 [ACK] 1
```

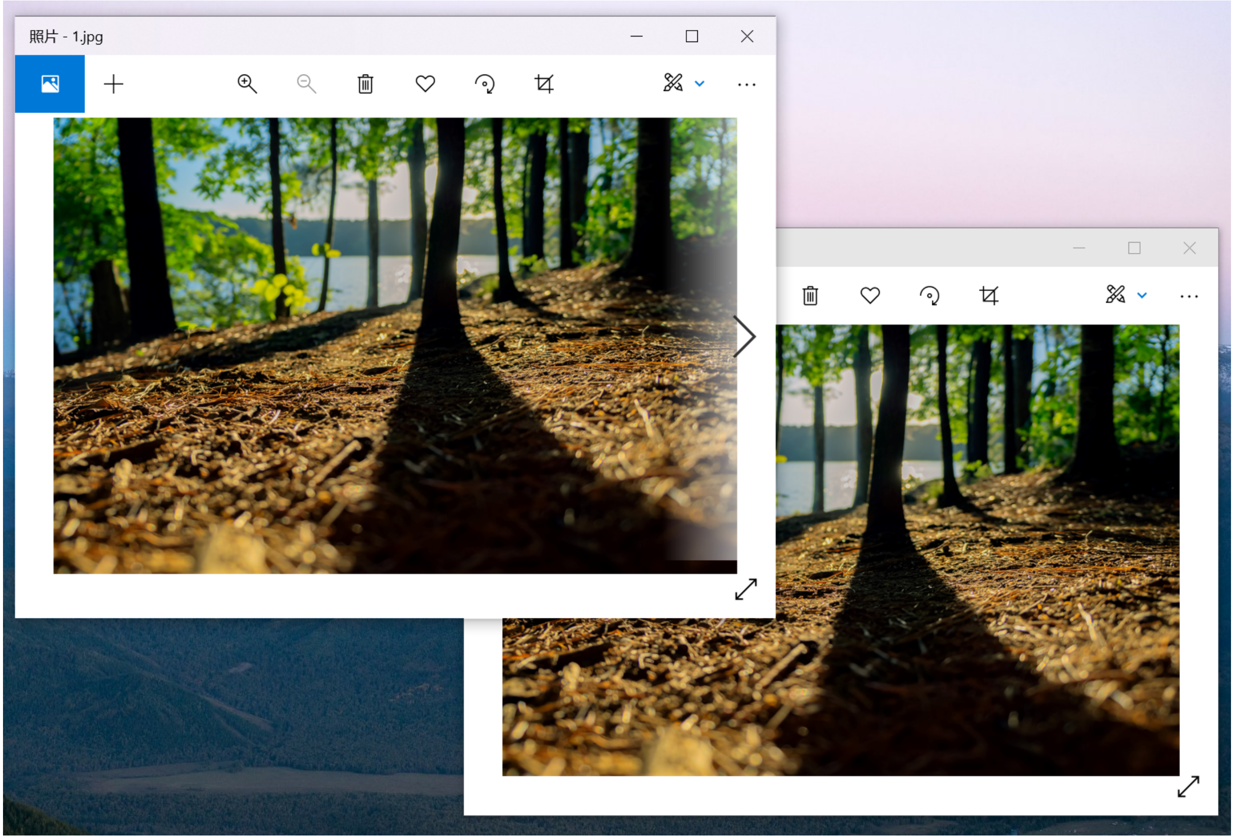
3. 接收端接收报文

可以观察收到第一列2048代表接收到数据报大小，checksum代表校验和

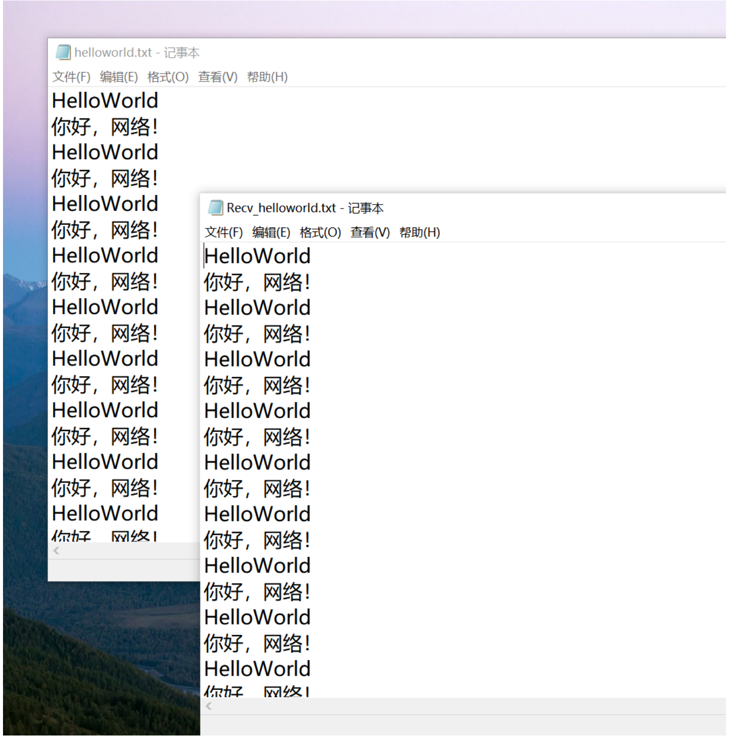


```
==================================================
[SYN]
[SYN & ACK] ->  [SEQ] 0 [ACK] 1 [SYN_FLAG] 1 [ACK_FLAG] 1
[ACK]
连接成功!
==================================================

Recv: 2048 [SEQ] 1 [ACK] 2049 [checksum] 65108
Recv: 2048 [SEQ] 1 [ACK] 4097 [checksum] 10037
Recv: 2048 [SEQ] 1 [ACK] 6145 [checksum] 6349
Recv: 2048 [SEQ] 1 [ACK] 8193 [checksum] 44878
Recv: 2048 [SEQ] 1 [ACK] 10241 [checksum] 14823
Recv: 2048 [SEQ] 1 [ACK] 12289 [checksum] 30568
Recv: 2048 [SEQ] 1 [ACK] 14337 [checksum] 23914
Recv: 2048 [SEQ] 1 [ACK] 16385 [checksum] 55734
Recv: 2048 [SEQ] 1 [ACK] 18433 [checksum] 23486
Recv: 2048 [SEQ] 1 [ACK] 20481 [checksum] 47683
Recv: 2048 [SEQ] 1 [ACK] 22529 [checksum] 23709
Recv: 2048 [SEQ] 1 [ACK] 24577 [checksum] 16866
Recv: 2048 [SEQ] 1 [ACK] 26625 [checksum] 54840
Recv: 2048 [SEQ] 1 [ACK] 28673 [checksum] 12354
Recv: 2048 [SEQ] 1 [ACK] 30721 [checksum] 50763
Recv: 2048 [SEQ] 1 [ACK] 32769 [checksum] 4322
```

4. 发送图片对比



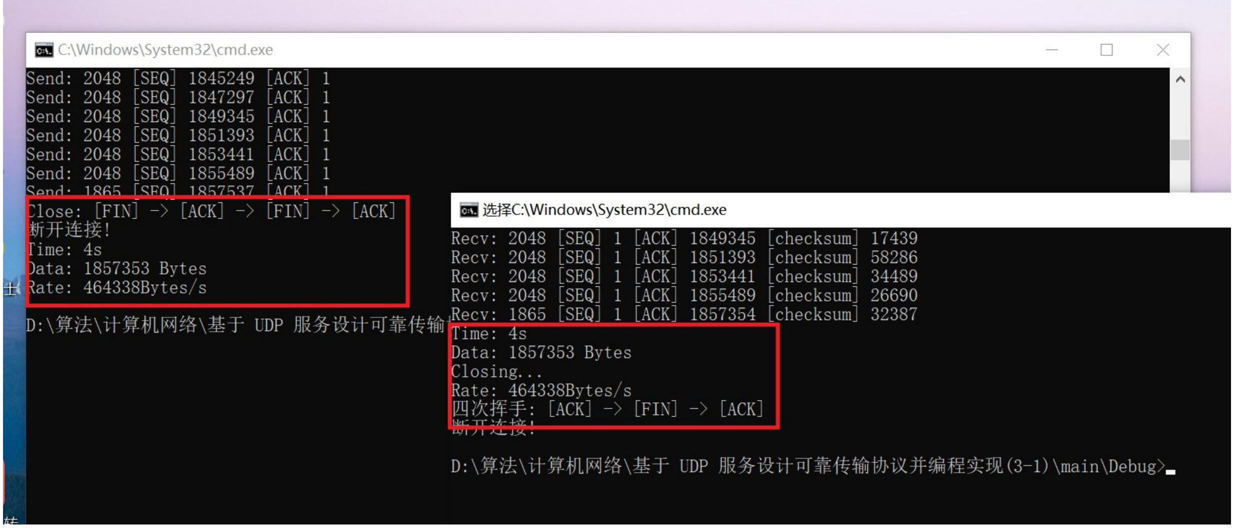
5. 发送文档对比


helloworld.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
Recv_helloworld.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
```

```
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
HelloWorld
你好，网络!
```

6. 四次挥手断开连接+传输信息



```
Send: 2048 [SEQ] 1845249 [ACK] 1
Send: 2048 [SEQ] 1847297 [ACK] 1
Send: 2048 [SEQ] 1849345 [ACK] 1
Send: 2048 [SEQ] 1851393 [ACK] 1
Send: 2048 [SEQ] 1853441 [ACK] 1
Send: 2048 [SEQ] 1855489 [ACK] 1
Send: 1865 [SEQ] 1857537 [ACK] 1
Close: [FIN] -> [ACK] -> [FIN] -> [ACK]
断开连接!
Time: 4s
Data: 1857353 Bytes
Rate: 464338Bytes/s

D:\算法\计算机网络\基于 UDP 服务设计可靠传输
```

```
Recv: 2048 [SEQ] 1 [ACK] 1849345 [checksum] 17439
Recv: 2048 [SEQ] 1 [ACK] 1851393 [checksum] 58286
Recv: 2048 [SEQ] 1 [ACK] 1853441 [checksum] 34489
Recv: 2048 [SEQ] 1 [ACK] 1855489 [checksum] 26690
Recv: 1865 [SEQ] 1 [ACK] 1857354 [checksum] 32387
Time: 4s
Data: 1857353 Bytes
Closing...
Rate: 464338Bytes/s
四次挥手: [ACK] -> [FIN] -> [ACK]
断开连接!

D:\算法\计算机网络\基于 UDP 服务设计可靠传输协议并编程实现(3-1)\main\Debug>
```