哈爾濱工業大學
Harbin Institute of Technology

计算机网络
课程实验报告

实验名称	GBN 协议的设计与实现				
姓名	方烨		院系	计算学部人工智能	
班级	1903601		学号	1190202418	
任课教师	李全龙		指导教师	李全龙	
实验地点	格物 207		实验时间	2021.11.6	
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					

计算机科学与技术学院 SINCE 1956...
School of Computer Science and Technology

实验目的：

(1) 理解可靠数据传输的基本原理；掌握停等协议的工作原理；掌握基于 UDP 设计并实现一个停等协议的过程与技术。

(2) 理解滑动窗口协议的基本原理；掌握 GBN 的工作原理；掌握基于 UDP 设计并实现一个 GBN 协议的过程与技术。

实验内容：

(1) 基于 UDP 设计一个简单的 GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）

(2) 模拟引入数据包的丢失，验证所设计协议的有效性；

(3) 改进所设计的 GBN 协议，支持双向数据传输（选作，加分项）；

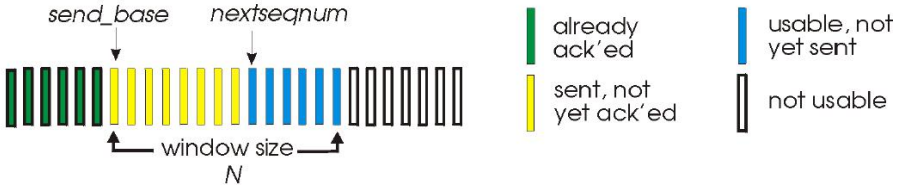
(4) 将所设计的 GBN 协议改进为 SR 协议（选作，加分项）。

实验过程：

1.GBN协议：

1.1 GBN协议概述

GBN协议数据传输采用流水线机制，GBN的发送方在收到ACK之前可以连续发送多个分组，这则使得GBN需要更大的存储空间以缓存分组。窗口尺寸为N，最多允许N个分组未确认；GBN协议采用的是累计确认机制，ACK(n)表示确认到序列号n(包含n) 的分组均已被正确接收；GBN协议为空中的分组设置计时器（1个），如果发生超时（Timeout）事件，重传序列号大于等于n，还未收到ACK的所有分组。注意：停等协议只需要将GBN协议的发送窗口大小设置为1便可实现。



1.2 单向数据传输

1.2.1 GBN发送方：

GBN发送方，也就是服务器端，需要处理三个事件：

A. 上层的调用：

上层要发送数据时，发送方先检查发送窗口是否已满，如果未满，则产生一个帧并将其发送，如果窗口已满，发送方只需要将数据返回给上层，暗示上层窗口已满，上层等一会再发送。（实际实现中，发送方可以缓存这些数据，窗口不满时再发送帧）。

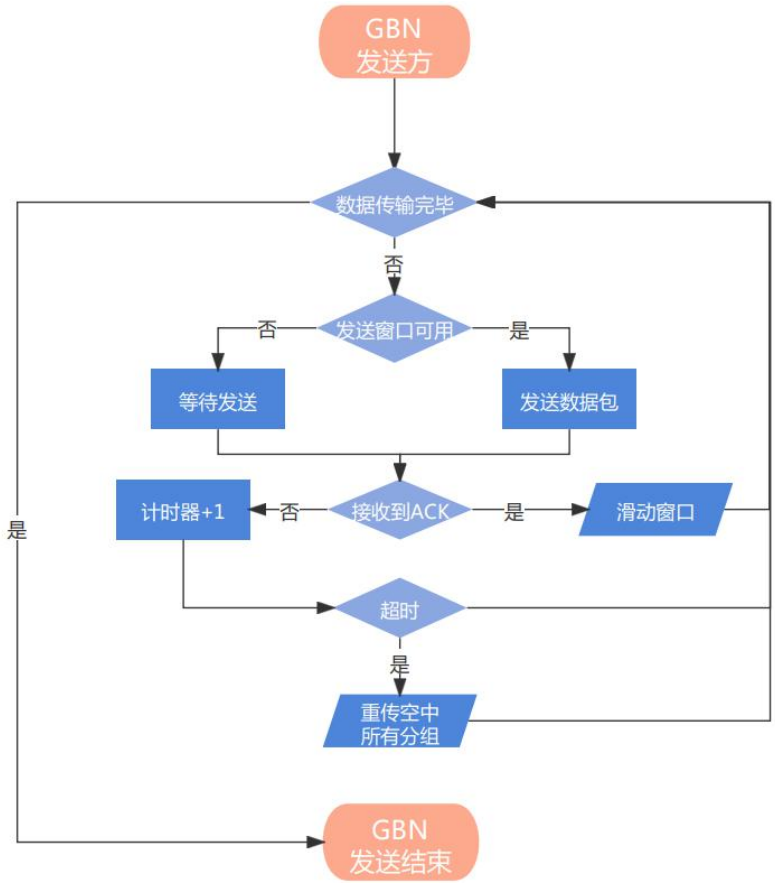
B. 收到ACK

GBN协议中，对n号帧的确认采用累计确认的方式，标明接收方已经收到n号帧和它之前的全部帧。如果收到更新的ACK，则滑动窗口向前移动。

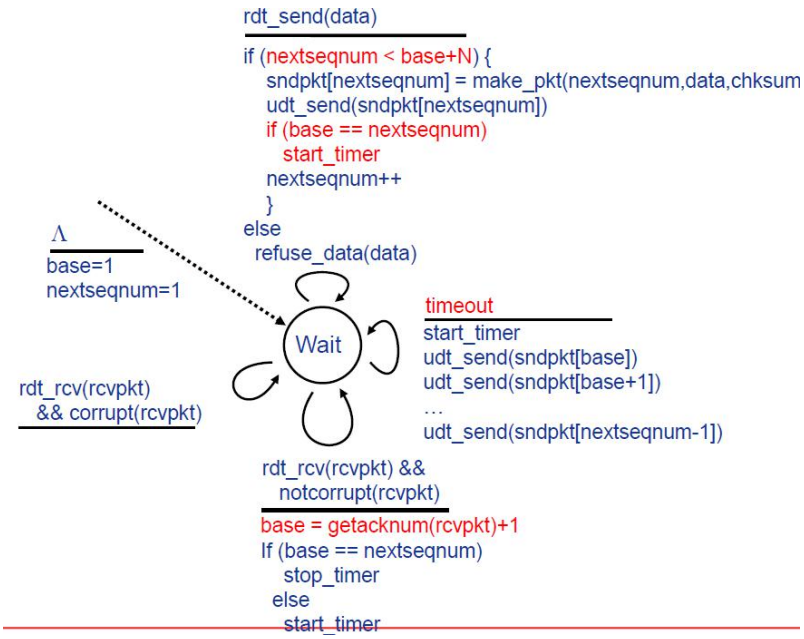
C. 超时事件

协议的名字为后退N帧/回退N帧，来源于出现丢失和时延过长帧时发送方的行为。就像在停等协议中一样，定时器将再次用于恢复数据帧或确认帧的丢失。如果出现真的超时，发送方重传所有已发送但未被确认的帧。

- 发送方流程图



➢ 发送方扩展FSM：



1.2.2 GBN接收方：

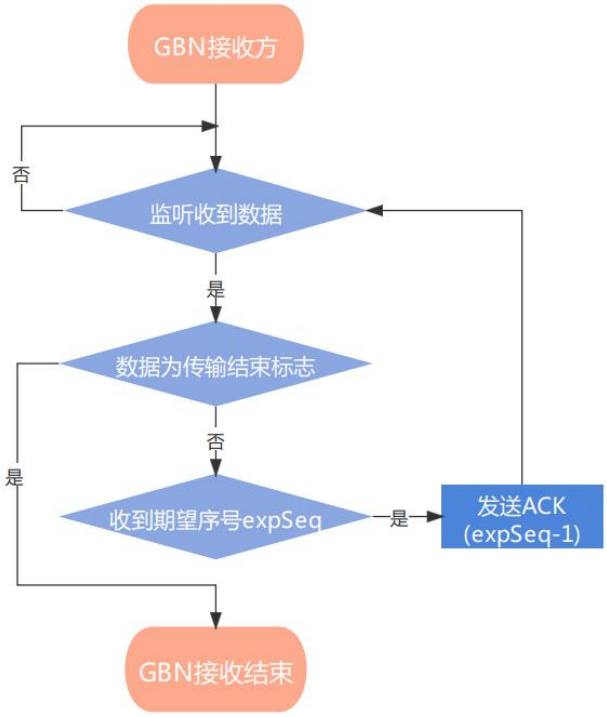
GBN接收方，也就是客户端，需要处理以下事件：

A. 如果正确收到n号帧，并且有序，那么接收方为n帧发送一个ACK，并将该帧中的数据部分交付给上层。

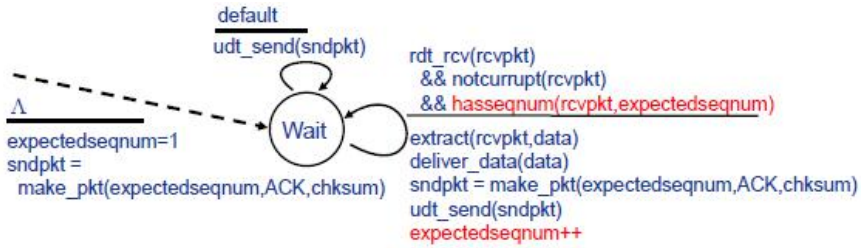
B. 其余情况都丢帧，并为最近按序接收的帧重新发送ACK。接收方无需缓存任何失序帧，

只需要维护一个信息：expSeq（下一个按序的帧序号）。

- 接收方流程图：



- 接收方扩展FSM：

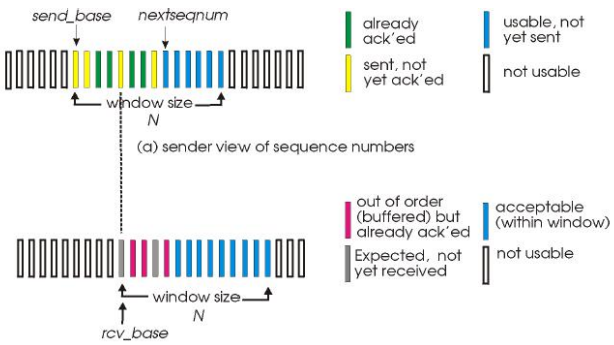


2.SR协议：

2.1 SR协议概述

首先我们通过对 GBN 协议的分析，可以知道 GBN 协议本身存在缺陷——GBN 在重传的时候回重传很多分组（比如序列号分组 n 丢失时，需要重传 n 以及 n 以后没有确认的分组）。这样就会导致网络中充斥着很多重传的分组，影响分组。

因此，一种改进的机制就是不使用累积确认，采用单个确认，同时我们也不丢弃乱序的分组，将其缓存起来。这时，发送方只需要重传那些没收到 ACK 的分组。现在对每个分组设置单独的定时器，当某个分组的定时器超时后对其进行重传。



2.2 单向数据传输

2.2.1 SR发送方（服务器端）

➢ SR发送方的事件与动作：

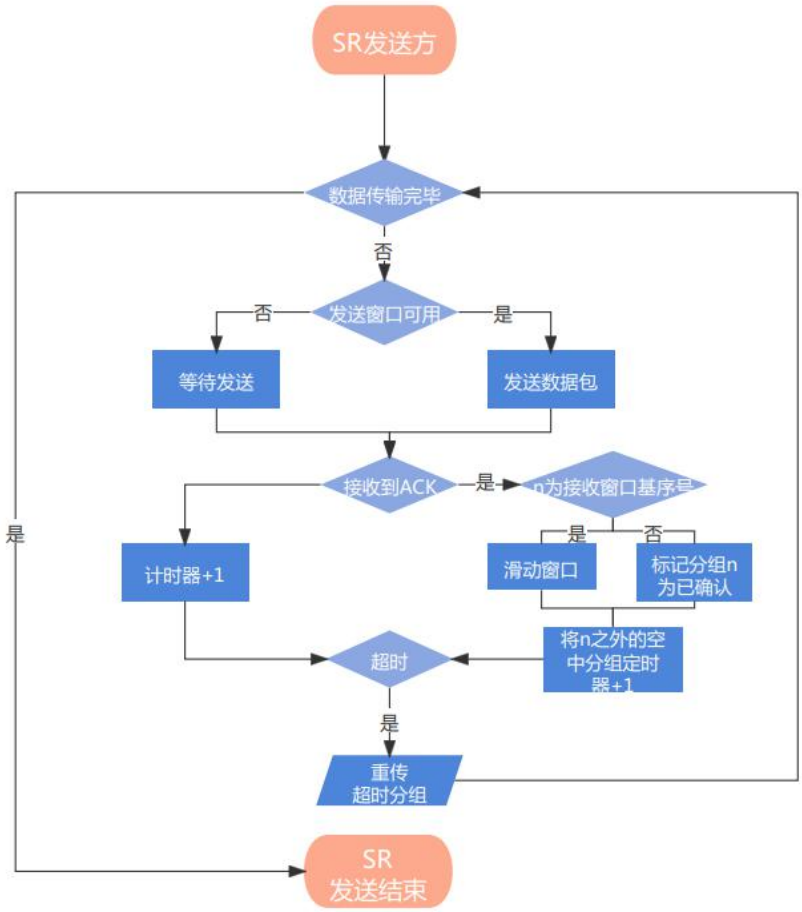
A. 从上层收到数据。当从上层收到数据后，SR 检查发送方下一个可用于该分组的序号。如果序号位于发送方的窗口内，则将数据打包并发送；否则就像 GBN 一样进行处理。

B. 超时。定时器再次被用来防止丢失分组。然而，现在每一个分组必须由自己的逻辑定时器，因为超时发送后只能发送一个分组。

C. 收到 ACK。如果收到 ACK，若该分组序号在窗口内，则 SR 发送方将那个被确认的分组标记为已接收。如果该序号范围等于 sendBase,则窗口基序号向前移动到具有最小序号的未被确认的分组处。如果窗口移动了并且有序号落在窗口内的未发送分组，则发送这些分组。

SR 的接收方将确认一个正确接收的分组而不管其是否有序。失序的分组将被缓存直到所有丢失的分组（即序号更小的分组）都被收到为止，这是才将一批分组按序交付给上层。

➢ 发送方流程图：



2.2.2 SR接收方

➢ SR接收方的事件与动作：

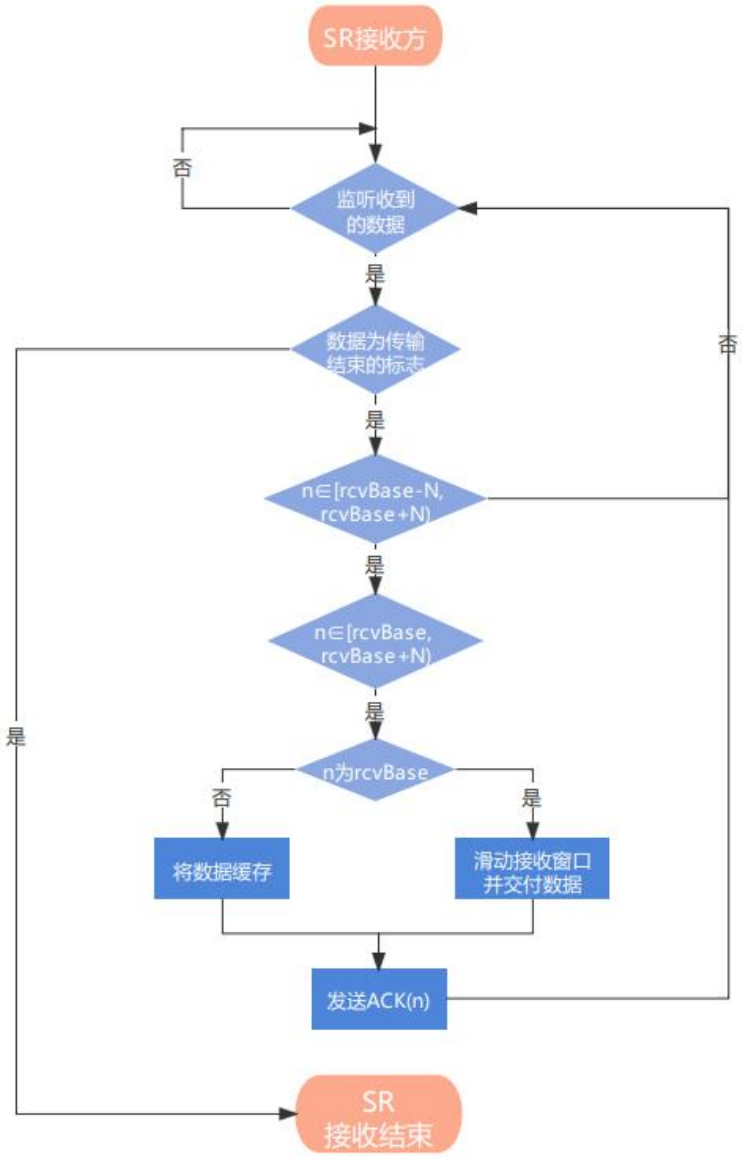
A. 序号在[rcvBase, rcvBase+N-1]内的分组被正确接收。在此情况下，收到的分组落在接收方窗口内，一个选择 ACK 被传回给发送方。如果该分组以前没有收到过，则缓存该分组。如果该分组的序号等于接收方窗口基序号，则将该分组以及以前缓存的序号连续的分组交付给上层。然后，接收窗口按向前移动分组的编号向上交付这些分组。

B. 序号在[rcvBase-N, rcvBase-1]内的分组被正确接收到。在此情况下，必须产生一个ACK,

即该分组是接收方以前确认过的分组。

C. 其他情况。忽略该分组

➢ 接收方流程图：



3.实验说明：

(1) 由于停等协议只需要将GBN协议的发送窗口大小设置为1便可实现，故本实验直接展示GBN和SR的实现过程，不再赘述停等协议。

(2) 本实验用读取文件中的数据流模拟上层协议中数据的到达，用写入数据到新的文件中模拟成功将数据交付上层协议。

4.数据传输格式：

4.1 数据格式说明

规定数据报格式为: [类型&序号&数据]；编码格式为: utf-8

发送数据格式	["data", seqNum, data]
发送ACK格式	["ack", expNum, 0]
发送结束格式	["end", 0, 0]

4.2 代码实现

```
@staticmethod
def make_pkt(label, pkt_num, data):
    """
    @params: pkt_num对应数据报序号或ACK序号
             data发送数据
             label数据类型
    @return: 规定格式的数据报

    产生一个发送数据包或者确认数据包
    规定数据报格式为: [类型&序号&数据]; 编码格式为: utf-8
    发送数据格式: ["data", seqNum, data], 发送ACK格式: ["ack", expNum, 0], 发送结束格式: ["end", 0, 0]
    """
    return (label + '&' + str(pkt_num) + '&' + str(data)).encode(encoding='utf-8')
```

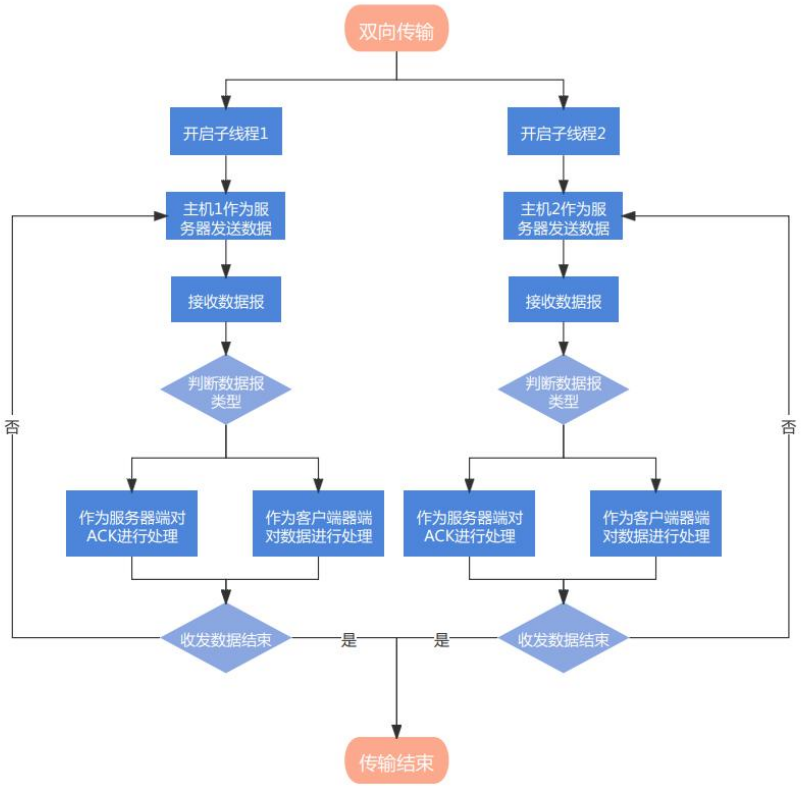
5.双向数据传输：

5.1 双向数据传输概述

- 双工：指二台通讯设备之间，允许有双向的资料传输。通常有两种双工模式。一种叫半双工，另一种叫全双工。
- 全双工：全双工（full-duplex）的系统允许二台设备间同时进行双向数据传输。一般的电话、手机就是全双工的系统，因为在讲话时同时也可以听到对方的声音。
- 半双工：半双工（half-duplex）的系统允许二台设备之间的双向数据传输，但不能同时进行。因此同一时间只允许一设备传送资料，若另一设备要传送资料，需等原来传送资料的设备传送完成后再处理。

本实验实现的gbn_both和sr_both类将服务器和客户端维护的变量和函数进行整合，同时加入了bothRun函数支持同时收发，既能支持半双工也能支持全双工，由于半双工的流程和之前所述服务器/客户端各自的流程类似，下面主要阐述全双工的实现。

5.2 双向传输示意图



6.模拟丢包方法：

6.1 模拟数据丢包

发送方（服务器端）在上层数据到达且窗口空闲、超时重传情况下需要发送数据。在发送数据包时，调用random.random()函数产生一个(0,1)区间的随机数，如果该数字大于设定的数据包丢包率，则发送数据；否则不发送数据。

6.2 模拟ACK丢包

接收方（客户端）在收到发送方的传输数据时会发送ACK，我们还是采用上述逻辑，利用产生的随机数以及设定的ACK丢包率模拟ACK丢包。

6.3 代码示例

服务器模拟数据丢包

```
if random.random() > self.pktLossRate:  # 发送数据, 并模拟丢包
    self.socket.sendto(Host.make_pkt('data', self.nextSeq, self.data[self.nextSeq]), self.remoteAddr)
info = str(self.localAddr)+'  服务器: 成功发送数据' + str(self.nextSeq)
print(info)
```

客户端模拟ACK丢包

```
if random.random() >= self.ackLossRate:  # 接收方发送ACK, 模拟ACK丢包
    self.socket.sendto(Host.make_pkt('ack', self.expSeq - 1, 0), self.remoteAddr)
```

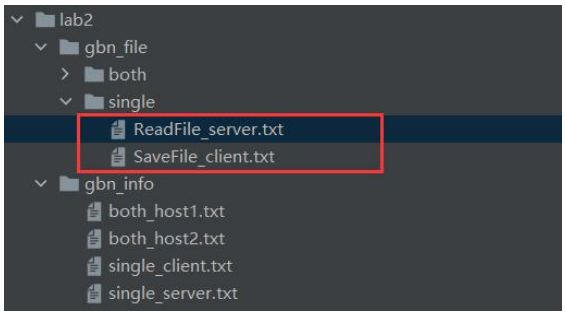
实验结果：

1.GBN协议：

1.1 单向数据传输

1.1.1 传输数据说明

如下图，服务器端读入文件为ReadFile_server.txt，客户端写入文件为SaveFile_client.txt



1.1.2 传输过程

```
C:\Users\方烨\AppData\Local\Programs\Python\Python38\python.exe E:/pythonProject/JW/lab2/main.py
GBN or SR? (input 0 for GBN, 1 for SR)
0
unidirectional or bidirectional? (input 0 for uni, 1 for bi)
0
('127.0.0.1', 12320)  服务器: 成功发送数据0
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为0
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 0
('127.0.0.1', 12320)  服务器: 成功发送数据1
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为1
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 1
('127.0.0.1', 12320)  服务器: 成功发送数据2
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为2
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 2
('127.0.0.1', 12320)  服务器: 成功发送数据3('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为3

('127.0.0.1', 12320)  服务器: 收到客户端ACK: 3
('127.0.0.1', 12320)  服务器: 成功发送数据4
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为4
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 4
('127.0.0.1', 12320)  服务器: 成功发送数据5
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为5
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 5
```

查看客户端和服务器各自的传输信息：

```
====================客户端信息====================
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为0
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为1
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为2
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为3
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为4
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为5
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为6
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为7
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为8
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为9
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为10
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为11
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为12
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为13
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为14
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为15
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为16
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为17
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为18
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为19
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为20
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为21
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为22
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为23
('127.0.0.1', 10086) ===客户端: 所有数据报都已成功接收且均发送ACK, 传输数据结束===
```

客户端按序接收数据报，直到所有数据报接收完毕。

```
====================服务器端信息====================
('127.0.0.1', 12320)  服务器: 成功发送数据0
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 0
('127.0.0.1', 12320)  服务器: 成功发送数据1
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 1
('127.0.0.1', 12320)  服务器: 成功发送数据2
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 2
('127.0.0.1', 12320)  服务器: 成功发送数据3
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 3
('127.0.0.1', 12320)  服务器: 成功发送数据4
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 4
('127.0.0.1', 12320)  服务器: 成功发送数据5
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 5
('127.0.0.1', 12320)  服务器: 成功发送数据6
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 6
('127.0.0.1', 12320)  服务器: 成功发送数据7
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 12320)  【超时】--开始重传
('127.0.0.1', 12320)  数据已重发7
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 7
('127.0.0.1', 12320)  服务器: 成功发送数据8
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 8
('127.0.0.1', 12320)  服务器: 成功发送数据9
```

服务器端当窗口已满时则暂不发送数据，数据包超时则会进行重传，当所有数据发送完毕则发送完成。

1.1.3 传输文件对比

服务器端发送文件：

```
19980101-01-005-001/m  挂/v  起/v  红灯/n  迎/v  新年/t  （/w  图片/n  ）/w
19980101-01-005-002/m  元旦/t  来临/v  ，/w  安徽省/ns  合肥市/ns  长江路/ns  悬挂/v  起/v  3300/m  盏
19980101-01-005-003/m  （/w  传真/n  照片/n  ）/w

19980101-02-001-001/m  全总/j  致/v  全国/n  各族/r  职工/n  慰问信/n
19980101-02-001-002/m  勉励/v  广大/b  职工/n  发挥/v  工人阶级/n  主力军/n  作用/n  ，/w  为/p  企业/n
19980101-02-001-003/m  本报/r  北京/ns  1月/t  1日/t  讯/Ng  [中华/nz  全国/n  总工会/n]nt  今日/t  发
19980101-02-001-004/m  慰问信/n  说/v  ，/w  实现/v  党/n  的/u  十五大/j  提出/v  的/u  宏伟/a  目标/n
19980101-02-001-005/m  慰问信/n  指出/v  ，/w  广大/b  职工/n  要/v  以/p  主人翁/n  的/u  姿态/n  ，/w
19980101-02-001-006/m  慰问信/n  最后/f  说/v  ，/w  让/v  我们/r  在/p  邓小平理论/n  和/c  党/n  的/u

60
```

客户端接收文件：

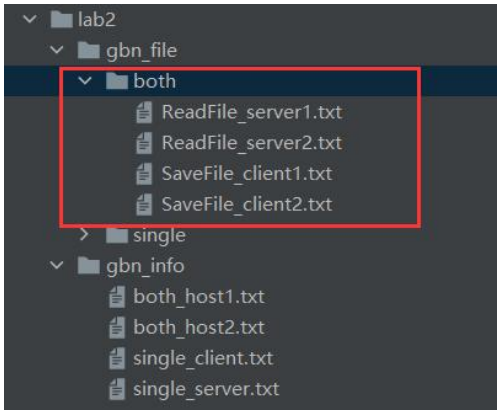
```

19980101-02-001-001/m  全总/j  致/v  全国/n  各族/r  职工/n  慰问信/n
19980101-02-001-002/m  勉励/v  广大/b  职工/n  发挥/v  工人阶级/n  主力军/n  作用/n  ，/w  为/p  企业
19980101-02-001-003/m  本报/r  北京/ns  1月/t  1日/t  讯/Ng  [中华/nz  全国/n  总工会/n]nt  今日/t
19980101-02-001-004/m  慰问信/n  说/v  ，/w  实现/v  党/n  的/u  十五大/j  提出/v  的/u  宏伟/a  目标
19980101-02-001-005/m  慰问信/n  指出/v  ，/w  广大/b  职工/n  要/v  以/p  主人翁/n  的/u  姿态/n  ，
19980101-02-001-006/m  慰问信/n  最后/f  说/v  ，/w  让/v  我们/r  在/p  邓小平理论/n  和/c  党/n  的

60
```

同样是60行，且对比后发现文件内容相同。

1.2 双向数据传输

1.2.1 传输数据说明

如下图所示，主机1从ReadFile_server1.txt中读入数据，将从主机2接收到的数据写入文件SaveFile_client1.txt中；主机2从ReadFile_server2.txt中读入数据，将从主机1接收到的数据写入文件SaveFile_client2.txt中。



1.2.2 传输过程

```
C:\Users\方烨\AppData\Local\Programs\Python\Python38\python.exe E:/pythonProject/JW/lab2/main.py
GBN or SR? (input 0 for GBN, 1 for SR)
0
unidirectional or bidirectional? (input 0 for uni, 1 for bi)
1
('127.0.0.1', 12320)  服务器: 成功发送数据0
('127.0.0.1', 10086)  服务器: 成功发送数据0
('127.0.0.1', 12320)  客户端: 收到期望数据, 序号为0
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为0
('127.0.0.1', 12320)  服务器: 成功发送数据1
('127.0.0.1', 10086)  服务器: 成功发送数据1
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 0
('127.0.0.1', 10086)  服务器: 收到客户端ACK: 0
('127.0.0.1', 10086)  服务器: 成功发送数据2
('127.0.0.1', 12320)  服务器: 成功发送数据2
('127.0.0.1', 10086)  【乱序】客户端: 收到非期望数据, 期望序号为1, 实际序号为2
('127.0.0.1', 12320)  【乱序】客户端: 收到非期望数据, 期望序号为1, 实际序号为2
('127.0.0.1', 10086)  服务器: 成功发送数据3
('127.0.0.1', 12320)  服务器: 成功发送数据3
('127.0.0.1', 10086)  服务器: 收到客户端ACK: 0
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 0
('127.0.0.1', 10086)  服务器: 成功发送数据4
('127.0.0.1', 12320)  服务器: 成功发送数据4
('127.0.0.1', 10086)  【乱序】客户端: 收到非期望数据, 期望序号为1, 实际序号为3
```

查看主机1和主机2各自的传输信息：

```
====================('127.0.0.1', 12320)====================
('127.0.0.1', 12320)  服务器: 成功发送数据0
('127.0.0.1', 12320)  客户端: 收到期望数据, 序号为0
('127.0.0.1', 12320)  服务器: 成功发送数据1
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 0
('127.0.0.1', 12320)  服务器: 成功发送数据2
('127.0.0.1', 12320)  【乱序】客户端: 收到非期望数据, 期望序号为1, 实际序号为2
('127.0.0.1', 12320)  服务器: 成功发送数据3
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 0
('127.0.0.1', 12320)  服务器: 成功发送数据4
('127.0.0.1', 12320)  【乱序】客户端: 收到非期望数据, 期望序号为1, 实际序号为3
('127.0.0.1', 12320)  服务器: 成功发送数据5
('127.0.0.1', 12320)  【乱序】客户端: 收到非期望数据, 期望序号为1, 实际序号为4
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 0
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 12320)  【乱序】客户端: 收到非期望数据, 期望序号为1, 实际序号为5
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 12320)  服务器: 收到客户端ACK: 0
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
```

```
====================('127.0.0.1', 10086)====================
('127.0.0.1', 10086)  服务器: 成功发送数据0
('127.0.0.1', 10086)  客户端: 收到期望数据, 序号为0
('127.0.0.1', 10086)  服务器: 成功发送数据1
('127.0.0.1', 10086)  服务器: 收到客户端ACK: 0
('127.0.0.1', 10086)  服务器: 成功发送数据2
('127.0.0.1', 10086)  【乱序】客户端: 收到非期望数据, 期望序号为1, 实际序号为2
('127.0.0.1', 10086)  服务器: 成功发送数据3
('127.0.0.1', 10086)  服务器: 收到客户端ACK: 0
('127.0.0.1', 10086)  服务器: 成功发送数据4
('127.0.0.1', 10086)  【乱序】客户端: 收到非期望数据, 期望序号为1, 实际序号为3
('127.0.0.1', 10086)  服务器: 成功发送数据5
('127.0.0.1', 10086)  【乱序】客户端: 收到非期望数据, 期望序号为1, 实际序号为4
('127.0.0.1', 10086)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 10086)  服务器: 收到客户端ACK: 0
('127.0.0.1', 10086)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 10086)  服务器: 收到客户端ACK: 0
('127.0.0.1', 10086)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 10086)  服务器: 收到客户端ACK: 0
('127.0.0.1', 10086)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 10086)  【窗口已满】服务器: 窗口已满，暂不发送数据。
```

可以看到，主机1和主机2分别作为服务器和客户端进行数据发送和接收。

1.2.3 传输文件对比

主机1发送文件：

```
40  19980101-01-004-001/m  李/nr  鹏/nr  在/p  北京/ns  考察/v  企业/n
41  19980101-01-004-002/m  向/p  广大/b  职工/n  祝贺/v  新年/t  ，/w  对/p  节日/n
42  19980101-01-004-003/m  新华社/nt  北京/ns  十二月/t  三十一日/t  电/n  （/w  [中
43  19980101-01-004-004/m  上午/t  九时/t  二十分/t  ，/w  李/nr  鹏/nr  总理/n  在/
44  19980101-01-004-005/m  在/p  总厂/n  所/u  属/v  的/u  [石景山/ns  热电厂/n]nt
45  19980101-01-004-006/m  （/w  A/nx  、/w  B/nx  ）/w
46  19980101-01-004-007/m  李/nr  鹏/nr  说/v  ：/w  “/w  作为/p  首都/n  的/u  电力
47  19980101-01-004-008/m  参观/v  工厂/n  结束/v  后/f  ，/w  李/nr  鹏/nr  又/d
48  19980101-01-004-009/m  陪同/v  考察/v  企业/n  并/c  看望/v  慰问/v  职工/n  的/u
49  19980101-01-005-001/m  挂/v  起/v  红灯/n  迎/v  新年/t  （/w  图片/n  ）/w
```

主机2接收文件：

```
43  19980101-01-004-004/m  上午/t  九时/t  二十分/t  ，/w  李/nr  鹏/nr  总理/n  在/p  [北京/ns
44  19980101-01-004-005/m  在/p  总厂/n  所/u  属/v  的/u  [石景山/ns  热电厂/n]nt  ，/w  李/nr
45  19980101-01-004-006/m  （/w  A/nx  、/w  B/nx  ）/w
46  19980101-01-004-007/m  李/nr  鹏/nr  说/v  ：/w  “/w  作为/p  首都/n  的/u  电力/n  工作者/
47  19980101-01-004-008/m  参观/v  工厂/n  结束/v  后/f  ，/w  李/nr  鹏/nr  又/d  来到/v  工厂
48  19980101-01-004-009/m  陪同/v  考察/v  企业/n  并/c  看望/v  慰问/v  职工/n  的/u  国务院/nt
49  19980101-01-005-001/m  挂/v  起/v  红灯/n  迎/v  新年/t  （/w  图片/n  ）/w
```

可以看到，均为49行，且文件内容相同。

主机2发送文件：

```
195  41, Federal-gov, 130760, Bachelors, 13, Married-civ-spouse, Tech-support, Husb
196  23, Private, 197387, 5th-6th, 3, Married-civ-spouse, Transport-moving, Other-r
197  36, Private, 99374, Some-college, 10, Divorced, Craft-repair, Not-in-family, W
198  40, Federal-gov, 56795, Masters, 14, Never-married, Exec-managerial, Not-in-fa
199  35, Private, 138992, Masters, 14, Married-civ-spouse, Prof-specialty, Other-re
200  24, Self-emp-not-inc, 32921, HS-grad, 9, Never-married, Sales, Not-in-family,
201
```

主机1接收文件：

```
197  36, Private, 99374, Some-college, 10, Divorced, Craft-repair, Not-in-family, Whi
198  40, Federal-gov, 56795, Masters, 14, Never-married, Exec-managerial, Not-in-fami
199  35, Private, 138992, Masters, 14, Married-civ-spouse, Prof-specialty, Other-rela
200  24, Self-emp-not-inc, 32921, HS-grad, 9, Never-married, Sales, Not-in-family, Wh
201
```

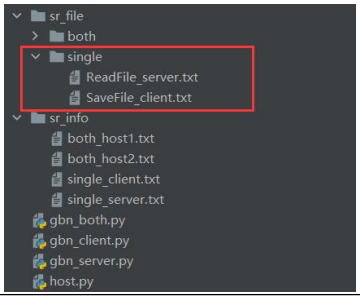
均为201行，且文件内容相同。

2.SR协议：

2.1 单向数据传输

2.1.1 传输数据说明

如下图，服务器端读入文件为ReadFile_server.txt，客户端写入文件为SaveFile_client.txt



2.1.2 传输过程

```
C:\Users\方烨\AppData\Local\Programs\Python\Python38\python.exe E:/pythonProject/J
GBN or SR? (input 0 for GBN, 1 for SR)
1
unidirectional or bidirectional? (input 0 for uni, 1 for bi)
0
('127.0.0.1', 12320)  服务器:成功发送数据0
('127.0.0.1', 10086)  客户端:收到数据0
('127.0.0.1', 10086)  客户端:窗口滑动到1
('127.0.0.1', 10086)  客户端:发送ACK0
('127.0.0.1', 12320)  服务器:收到有用ACK0
('127.0.0.1', 12320)  服务器:窗口滑动到1
('127.0.0.1', 12320)  服务器:成功发送数据1
('127.0.0.1', 10086)  客户端:收到数据2
('127.0.0.1', 12320)  服务器:成功发送数据2
('127.0.0.1', 10086)  客户端:发送ACK2
('127.0.0.1', 12320)  服务器:收到有用ACK2
('127.0.0.1', 12320)  服务器:成功发送数据3
('127.0.0.1', 10086)  客户端:收到数据3
('127.0.0.1', 10086)  客户端:发送ACK3('127.0.0.1', 12320)  服务器:收到有用ACK3

('127.0.0.1', 12320)  服务器:成功发送数据4
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 12320)  【超时】--开始重传
('127.0.0.1', 12320)  数据已重发:1
('127.0.0.1', 10086)  客户端:收到数据1
```

查看客户端和服务器端各自的传输信息：

```
====================客户端信息====================
('127.0.0.1', 10086)  客户端:收到数据0
('127.0.0.1', 10086)  客户端:窗口滑动到1
('127.0.0.1', 10086)  客户端:发送ACK0
('127.0.0.1', 10086)  客户端:收到数据2
('127.0.0.1', 10086)  客户端:发送ACK2
('127.0.0.1', 10086)  客户端:收到数据3
('127.0.0.1', 10086)  客户端:发送ACK3
('127.0.0.1', 10086)  客户端:收到数据1
('127.0.0.1', 10086)  客户端:窗口滑动到2
('127.0.0.1', 10086)  客户端:窗口滑动到3
('127.0.0.1', 10086)  客户端:窗口滑动到4
('127.0.0.1', 10086)  客户端:发送ACK1
('127.0.0.1', 10086)  客户端:收到数据5
('127.0.0.1', 10086)  客户端:发送ACK5
('127.0.0.1', 10086)  客户端:收到数据6
('127.0.0.1', 10086)  客户端:发送ACK6
('127.0.0.1', 10086)  客户端:收到数据4
('127.0.0.1', 10086)  客户端:窗口滑动到5
('127.0.0.1', 10086)  客户端:窗口滑动到6
('127.0.0.1', 10086)  客户端:窗口滑动到7
```

SR协议中客户端也需要进行窗口滑动。

```
====================服务器信息====================
('127.0.0.1', 12320)  服务器:成功发送数据0
('127.0.0.1', 12320)  服务器:收到有用ACK0
('127.0.0.1', 12320)  服务器:窗口滑动到1
('127.0.0.1', 12320)  服务器:成功发送数据1
('127.0.0.1', 12320)  服务器:成功发送数据2
('127.0.0.1', 12320)  服务器:收到有用ACK2
('127.0.0.1', 12320)  服务器:成功发送数据3
('127.0.0.1', 12320)  服务器:收到有用ACK3
('127.0.0.1', 12320)  服务器:成功发送数据4
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 12320)  【超时】--开始重传
('127.0.0.1', 12320)  数据已重发:1
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 12320)  服务器:收到有用ACK1
('127.0.0.1', 12320)  服务器:窗口滑动到2
('127.0.0.1', 12320)  服务器:窗口滑动到3
('127.0.0.1', 12320)  服务器:窗口滑动到4
('127.0.0.1', 12320)  服务器:成功发送数据5
('127.0.0.1', 12320)  服务器:收到有用ACK5
```

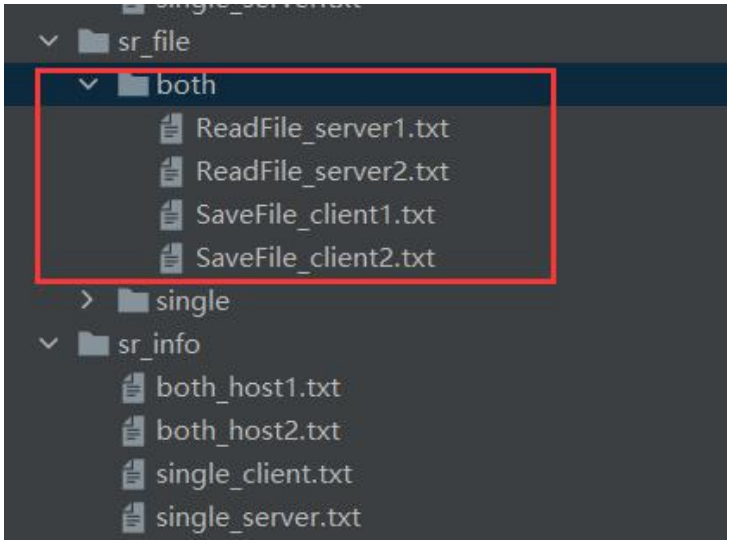
SR协议采用单个确认和重传，如上图中所示，在确认收到1之前，可以收到ACK2和ACK3，发现1超时，再对数据包1进行重传。

对比传输内容可知，读取和写入文件中内容大小一致。

2.2 双向数据传输

2.2.1 传输数据说明

如下图所示，主机1从ReadFile_server1.txt中读入数据，将从主机2接收到的数据写入文件SaveFile_client1.txt中；主机2从ReadFile_server2.txt中读入数据，将从主机1接收到的数据写入文件SaveFile_client2.txt中。



2.2.2 传输过程

```
C:\Users\方烨\AppData\Local\Programs\Python\Python38\python.exe E:/pythonProject
GBN or SR? (input 0 for GBN, 1 for SR)
1
unidirectional or bidirectional? (input 0 for uni, 1 for bi)
1
('127.0.0.1', 12320)  服务器:成功发送数据0
('127.0.0.1', 10086)  服务器:成功发送数据0
('127.0.0.1', 12320)  客户端:收到数据0
('127.0.0.1', 10086)  客户端:收到数据0
('127.0.0.1', 12320)  客户端:窗口滑动到1
('127.0.0.1', 10086)  客户端:窗口滑动到1
('127.0.0.1', 12320)  客户端:发送ACK0
('127.0.0.1', 10086)  客户端:发送ACK0
('127.0.0.1', 12320)  服务器:成功发送数据1
('127.0.0.1', 10086)  服务器:成功发送数据1
('127.0.0.1', 12320)  服务器:收到有用ACK0
('127.0.0.1', 10086)  服务器:收到有用ACK0
```

查看主机1和主机2各自的传输信息：

```
====================('127.0.0.1', 12320)====================
('127.0.0.1', 12320)  服务器:成功发送数据0
('127.0.0.1', 12320)  客户端:收到数据0
('127.0.0.1', 12320)  客户端:窗口滑动到1
('127.0.0.1', 12320)  客户端:发送ACK0
('127.0.0.1', 12320)  服务器:成功发送数据1
('127.0.0.1', 12320)  服务器:收到有用ACK0
('127.0.0.1', 12320)  服务器:窗口滑动到1
('127.0.0.1', 12320)  服务器:成功发送数据2
('127.0.0.1', 12320)  客户端:收到数据1
('127.0.0.1', 12320)  客户端:窗口滑动到2
('127.0.0.1', 12320)  客户端:发送ACK1
('127.0.0.1', 12320)  服务器:成功发送数据3
('127.0.0.1', 12320)  客户端:收到数据2
('127.0.0.1', 12320)  客户端:窗口滑动到3
('127.0.0.1', 12320)  客户端:发送ACK2
('127.0.0.1', 12320)  服务器:成功发送数据4
('127.0.0.1', 12320)  服务器:收到有用ACK2
('127.0.0.1', 12320)  【窗口已满】服务器: 窗口已满，暂不发送数据。
('127.0.0.1', 12320)  客户端:收到数据4
('127.0.0.1', 12320)  客户端:发送ACK4
```

```
====================('127.0.0.1', 10086)====================
('127.0.0.1', 10086)  服务器:成功发送数据0
('127.0.0.1', 10086)  客户端:收到数据0
('127.0.0.1', 10086)  客户端:窗口滑动到1
('127.0.0.1', 10086)  客户端:发送ACK0
('127.0.0.1', 10086)  服务器:成功发送数据1
('127.0.0.1', 10086)  服务器:收到有用ACK0
('127.0.0.1', 10086)  服务器:窗口滑动到1
('127.0.0.1', 10086)  服务器:成功发送数据2
('127.0.0.1', 10086)  客户端:收到数据2
('127.0.0.1', 10086)  客户端:发送ACK2
('127.0.0.1', 10086)  服务器:成功发送数据3
('127.0.0.1', 10086)  服务器:收到有用ACK1
('127.0.0.1', 10086)  服务器:窗口滑动到2
('127.0.0.1', 10086)  服务器:成功发送数据4
('127.0.0.1', 10086)  客户端:收到数据3
('127.0.0.1', 10086)  客户端:发送ACK3
('127.0.0.1', 10086)  服务器:成功发送数据5
('127.0.0.1', 10086)  服务器:收到有用ACK2
```

可以看到，主机1和主机2分别作为服务器和客户端进行数据发送和接收。
对比传输内容可知，读取和写入文件中内容大小一致。

问题讨论：

1. GBN协议和SR协议的对比：

- 客户端缓存设置：为了避免GBN协议发送方重传失序到达接收方的正确分组（不必要），SR协议在接收方设立接收方窗口（缓存）以存储失序到达的正确分组。
- 确认机制：GBN采用累积确认机制，SR接收方则对每个分组单独进行确认，而发送方只重传那些没有收到ACK的分组，所以需要为每个空中分组设置定时器。

所以本实验中，在实现了GBN协议后，根据两种协议的共同点和不同点对客户端和服务器端维护的变量和实现的函数进行适当改造，可以实现SR协议。

2. 双向传输的不同方式

双向传输有半双工和全双工方式，本实验两种方式均实现，重点呈现了全双工的传输方式，全双工和半双工优缺点对比如下：

- 半双工传输模式采用载波侦听多路访问/冲突检测。传统的共享型LAN以半双工模式运行，线路上容易发生传输冲突。与集线器相连的节点（即多个节点共享一条到交换机端口的连接）必须以半双工模式运行。因为这种节点必须能够冲突检测。
- 全双工传输模式可以用于点到点以太网连接和快速以太网连接，同时不会发生冲突，因为他们使用双绞线中两条不同线路。

心得体会：

本次实验让我加深了对GBN和SR协议的理解，从代码层面上去理解整个收发机制是如何进行的，明确了GBN协议和SR协议的区别，以及全双工和半双工方式通信的区别，并自己实现了单双向的数据传输，并模拟数据丢包，还原真实丢包场景。同时实现了多线程编程，掌握了基于UDP的通信过程，并有了更深入的理解。