【摘要】

在本次实验中，我在之前实验的基础上做了一些改进。具体修改如下：将3-2中的实验修改为双线程以实现能够同时收发数据包并进行相应处理的功能；此外在本次实验中我选择实现了拥塞控制算法为TCP拥塞控制的RENO算法，以实现发送窗口的动态调整。

【实验目的】

在实验3-2的基础上，实现一种拥塞控制算法，也可以是改进的算法，实现给定测试文件的传输。

——————实验知识点————————

【网络拥塞】

在某段时间内，如果对网络中某一资源的需求超出了该资源所能提供的上限，则会导致网络性能的下降，这种情况被称为网络拥塞。例如当主机发送的数据过多过快时会造成网络中的路由器无法及时处理，因而引入了排队时延。

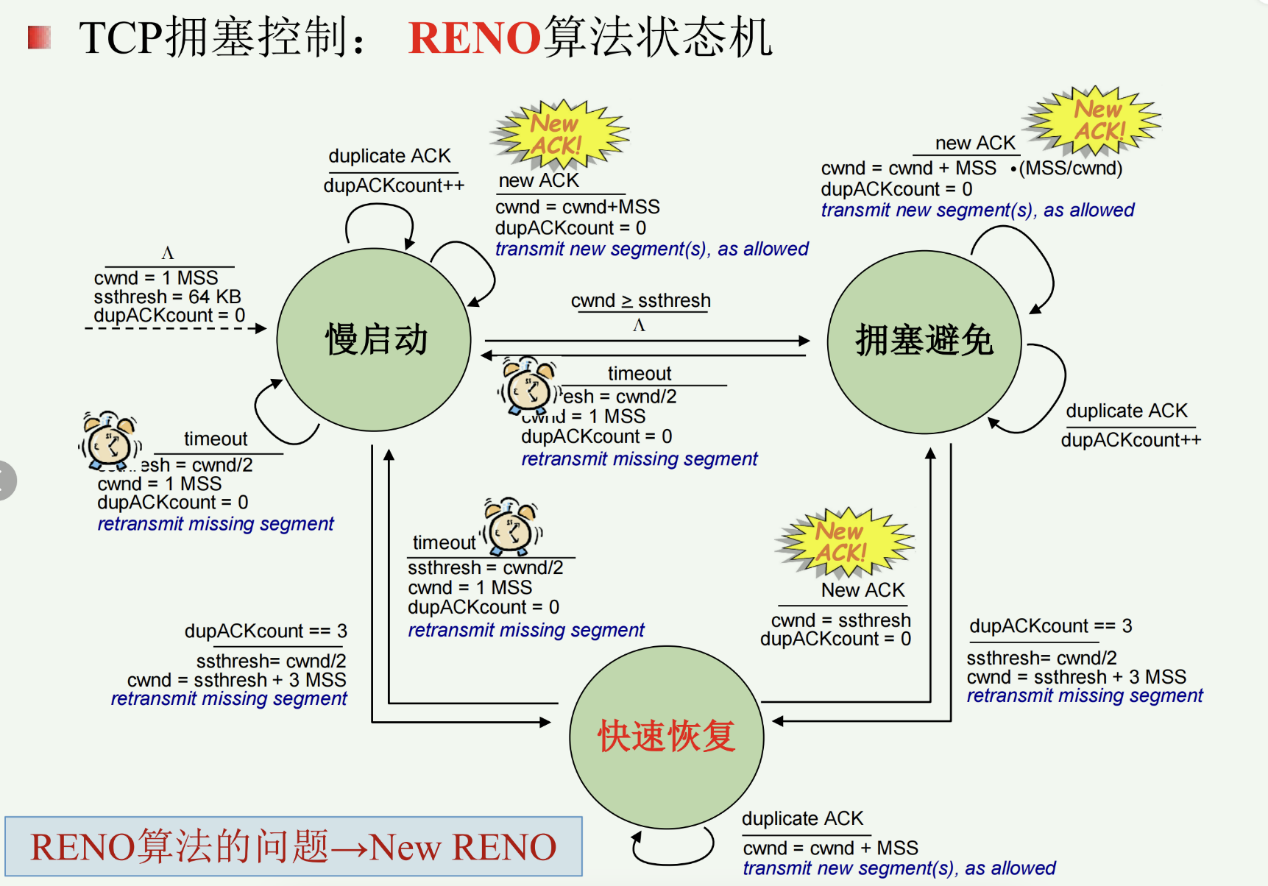
【拥塞控制】

当出现网络拥塞时，为了降低网络的阻塞同时尽快传输数据而引入了拥塞控制。当发送完数据包后依据是否接收到ACK包来进行带宽的检测。如果接收到ACK包说明网络并未拥塞，可以提高发送速率；发生丢失事件则降低发送速率。

【RENO算法】

RENO算法将发送窗口增大的过程分为两个阶段：第一个是慢启动阶段，在这一阶段中窗口大小被初始化为1，每当收到一个新的ACK时窗口大小变为原来的两倍，当窗口尺寸大于等于门限值ssthresh时则进入拥塞避免阶段。在拥塞避免阶段，每收到一个新的ACK，则将窗口大小+1，出现超时未收到ACK的情况则返回慢启动阶段，将ssthresh置为当前窗口大小的一半，窗口大小恢复为1。

而当在慢启动阶段或拥塞避免阶段连续收到三个重复ACK时则进入快速恢复阶段，ssthresh设置为当前窗口的一半，窗口大小=ssthresh+3



————————代码实现——————

【双线程代码实现】

对于客户端，将发送数据包作为主线程，而将收到ACK包并确认作为分线程。其具体流程如下：

1. 主线程调用发送文件函数，在此函数中完成对全局变量总发送数据包个数的定义后开辟新线程，用于接收服务器端返回的ACK包。
2. 主线程发送初始窗口中（不包含最后一个数据包）的所有包，在发送第一个数据包时开始计时；每发送一个数据包则更新当前窗口的已发送未确认seq号sentbutnotchecked以及下一个要发送数据包的序列号sequence，之后只要发送窗口的sentbutnotchecked != ceiling则说明收到了新的ACK包，继续发送窗口内未发送的数据包（序列号在sentbutnotchecked~ceiling之间）；一旦超时则重发整个窗口内所有的数据包。当识别到全局变量final=1时表示已经接收到最后一个数据包的ACK，结束运行。
3. 新线程负责接收确认包，对于接收到的确认包，如果其校验码正确且序列号大于等于当前窗口底部，则将发送窗口对应右移，保证当前窗口内序列号对应数据包均为未确认状态；如果其序列号为最后一个确认包（校验码正确且头部标志位为FINAL\_ACK），则代表均已正确完成，等待两个时钟周期后以return方式关闭此线程。

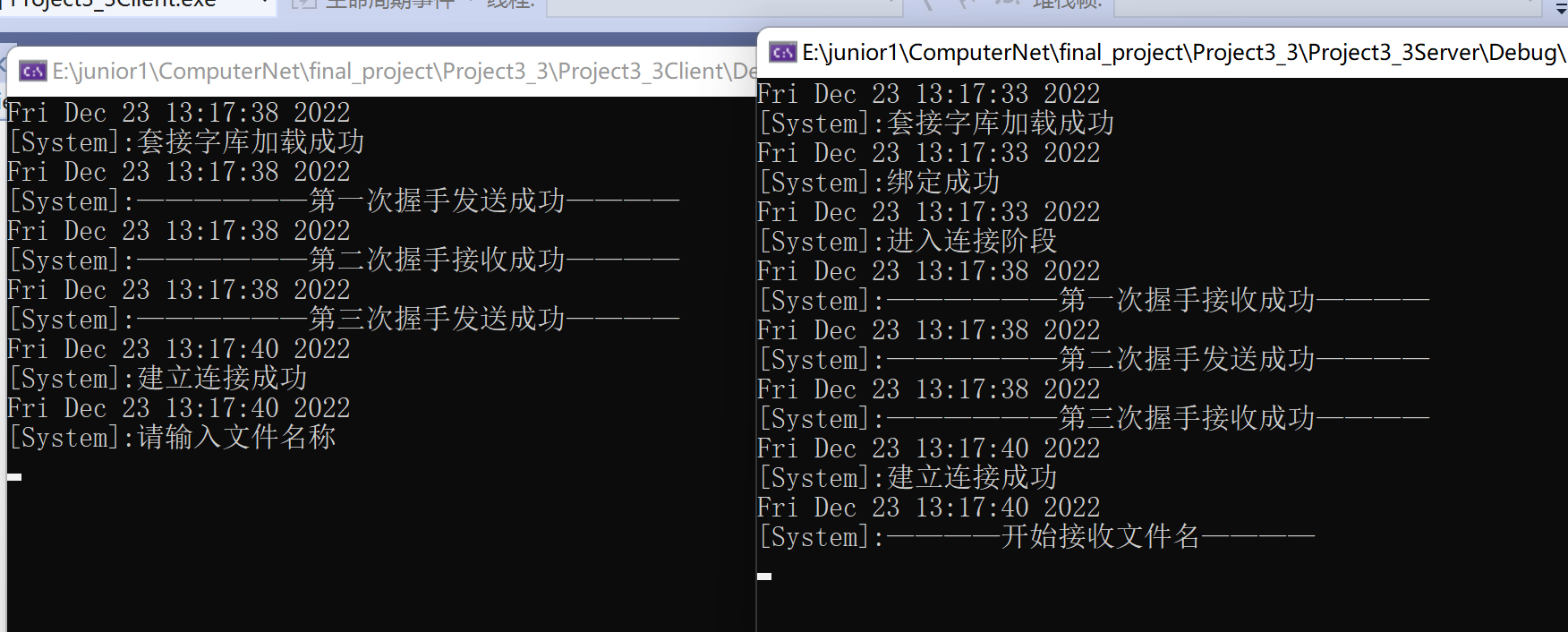
服务器端无需实现双线程，因此代码不做更改。

【拥塞控制窗口】

在发送方客户端设置一个发送窗口，窗口共有四个成员变量，分别为当前窗口的下限floor与上限ceiling，窗口的大小size和记录已发送但未得到确认的最大序列号sentbutnotchecked，此外还记录了一个常量ssthresh作为慢开始门限，定义值为14，如果当前窗口的size小于ssthresh，则处于慢开始阶段，否则处于拥塞避免阶段。发送方依据网络的拥塞程度动态的调整发送窗口的大小。

【结果图】设置丢包率2%，延时为1ms

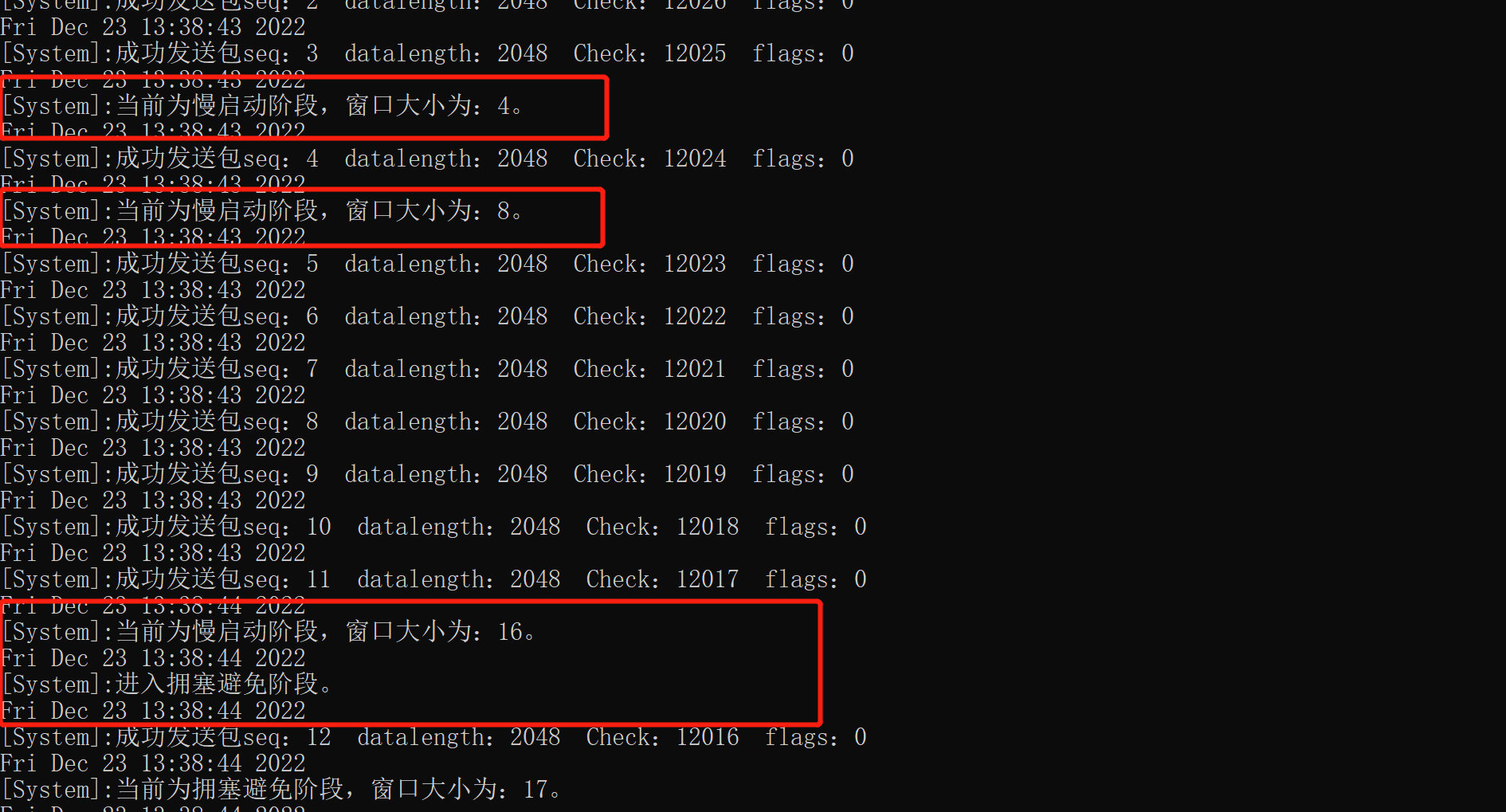
握手结果：



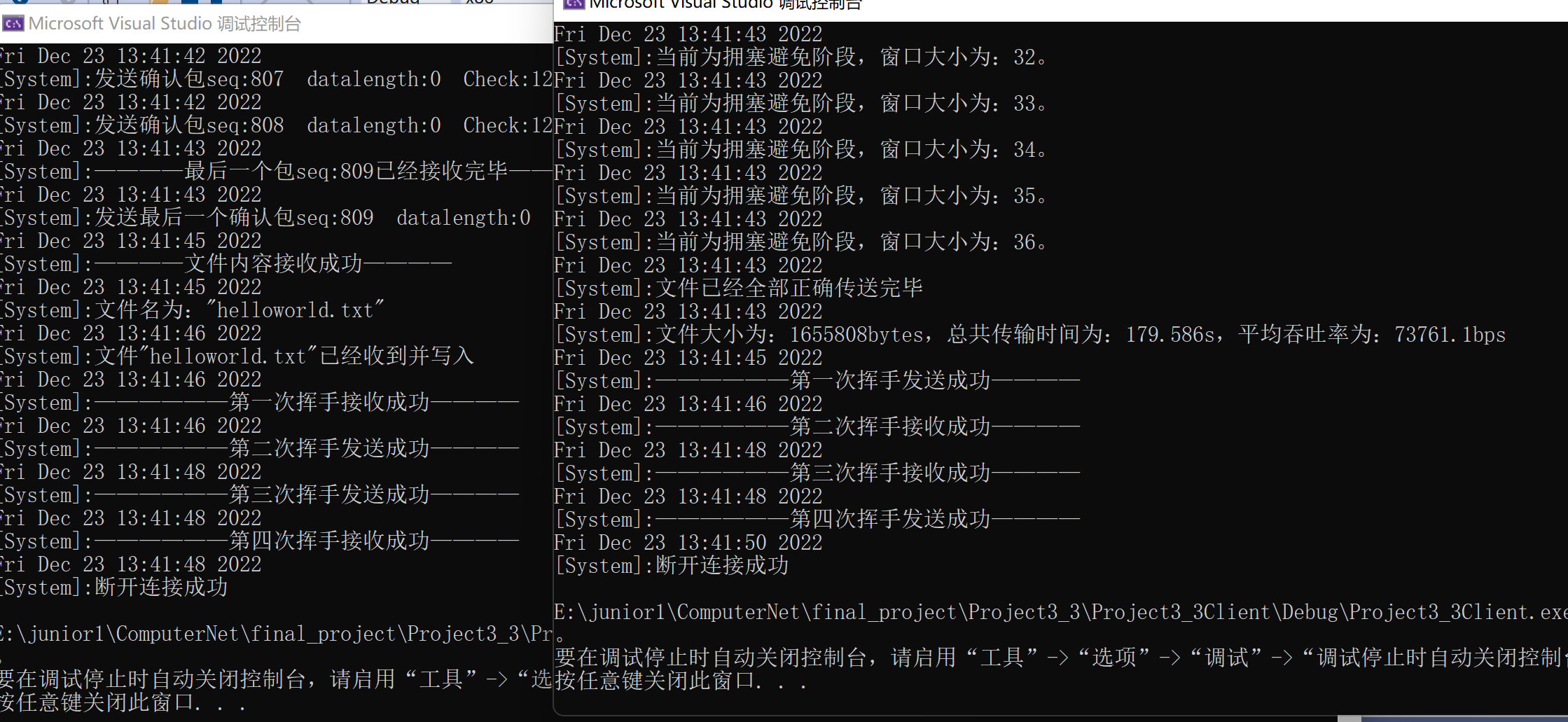
挥手结果：



窗口变更如图：



Helloworld.txt传输结果：共传输时间为：179.586s，平均吞吐率为：73761.1bps



图片一传输结果：共传输177.932s，平均吞吐率83508.4bps



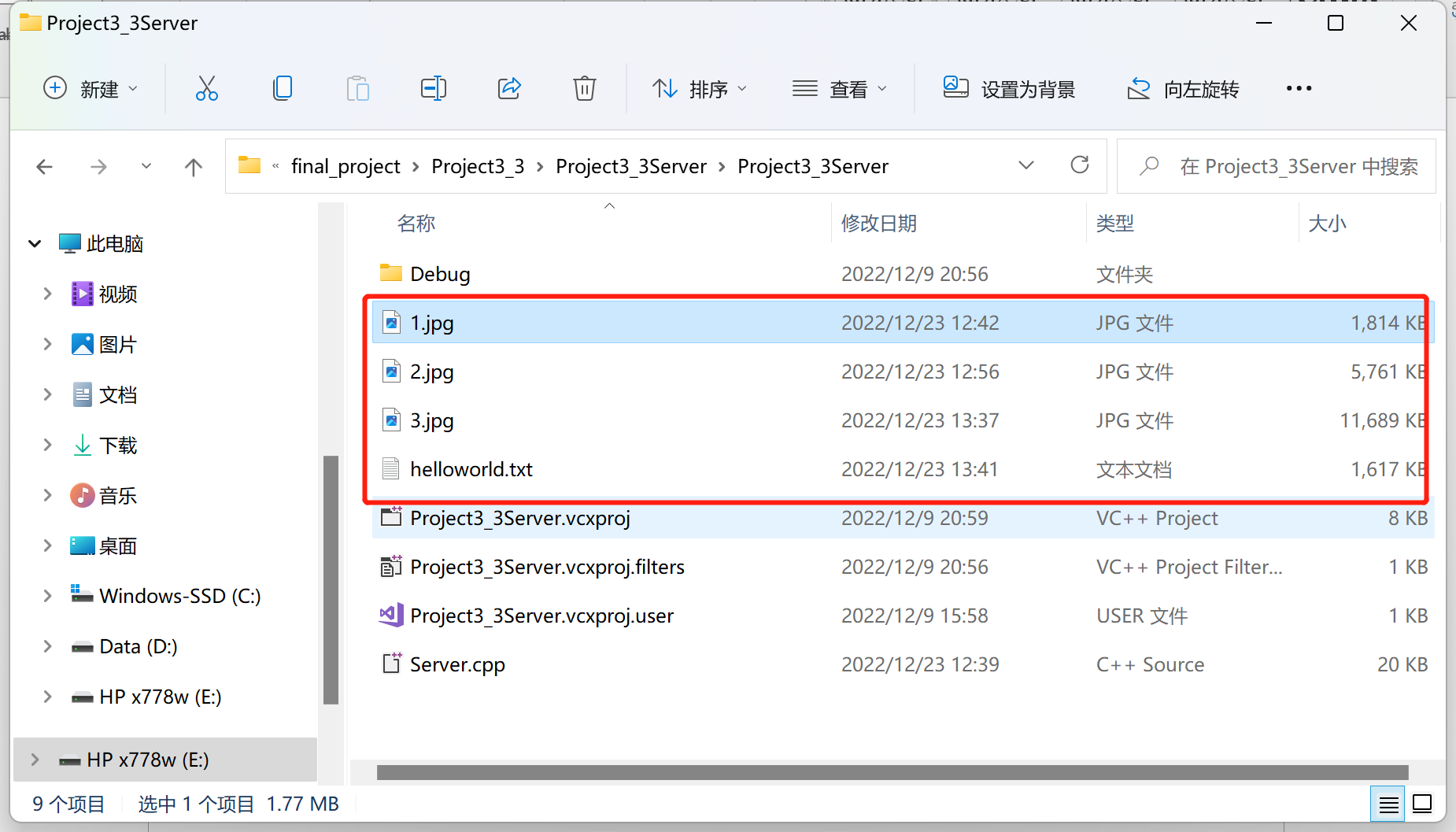
图片二传输结果：共传输534.273s，平均吞吐率88322bps



图片三传输结果：传输时间1120.87s，平均吞吐率为：85426.4bps



总传输结果图：



打开第二张图片为例，测试结果如下，说明文件传输正常。

