Task3 TCP socket programming

1. wireshark抓包：server在主机运行，client在虚拟机中运行，同时需要运行若干次客户端与服务器抓取数据包。

在host os通过python端启动服务器程序，如图1所示：

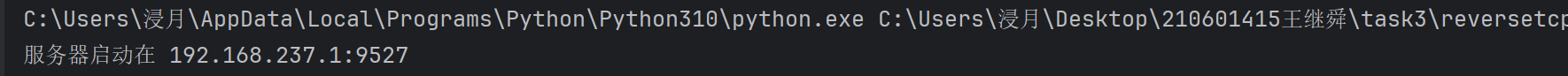


图1 启动服务器

在虚拟机中运行客户端程序，通过命令行终端输入，其中，需要给出服务器IP，目标i端口号，待反转文本路径，最小值，最大值，新反转文件路径，输入 python3 reversetcpclient.py 192.168.237.1 9527 /home/wjs/桌面/test.txt 1 8 /home/wjs/桌面/new1.txt，客户端运行如图2所示：

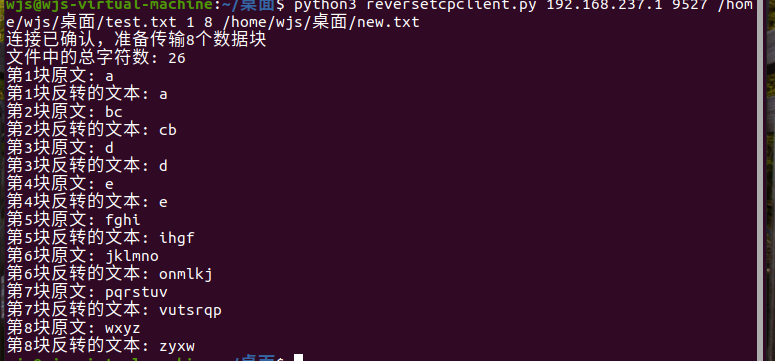


图2 启动客户端程序

在wireshark中捕获的数据包如图3所示，在TCP协议中，三次握手是建立一个TCP连接的过程，而四次挥手则是结束一个TCP连接的过程。

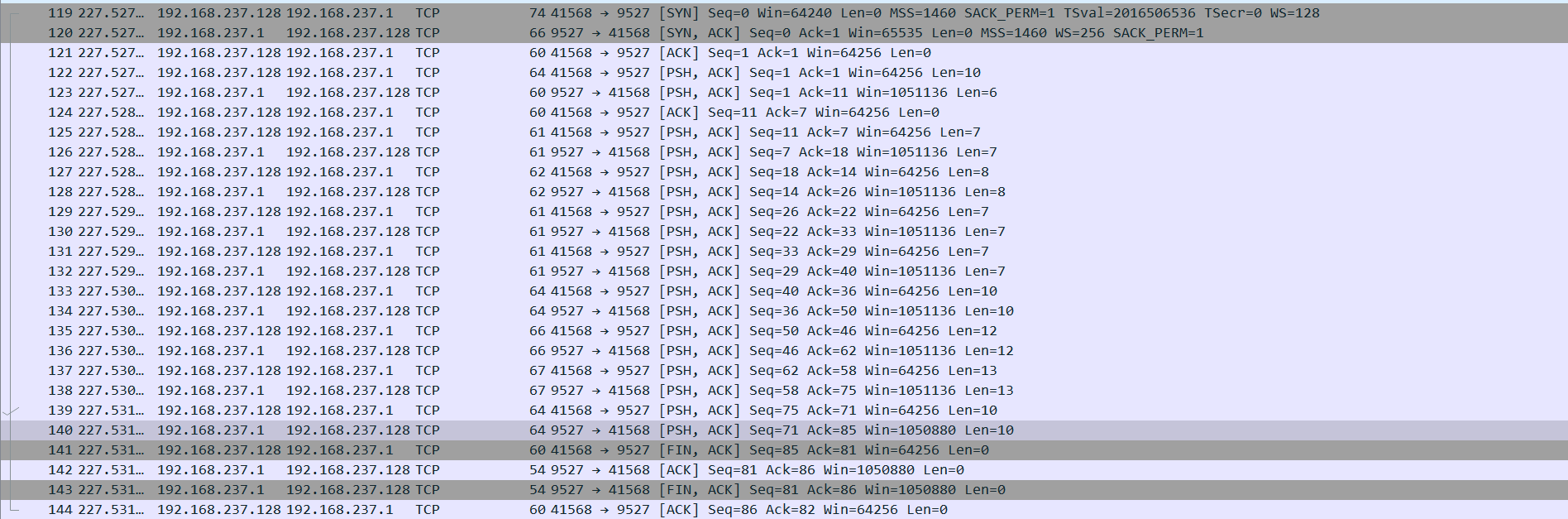


图3 wireshark捕获数据包

其中，No.119-No121是TCP建立连接的三次握手：

No.119报文是第一次握手（SYN）：SYN标志位被设置为1，其他标志位通常为0；Win=64240，表示客户端的接收窗口大小是64240字节；MSS(Maximum Segment Size)=1460，表示TCP一次传输发送的最大数据段长度是1460字节。

No.120报文是第二次握手（SYN+ACK）：YN和ACK都被设置为1；确认号ack=1，是客户端的序列号加1（第一次握手的序列号 + 1）。

No.121报文是第三次握手（ACK）：只有ACK标志位设置为1；序列号seq=1，是第一次握手的确认号（即客户端序列号 + 1）；确认号ack=1：第二次握手的序列号加1（服务器序列号 + 1）。

之后的No.122-No.140 建立客户端与服务器之间的通信：

No.122报文是客户端给服务器发送的initialization报文，该报文包含Type和N字段，N是请求服务器反转的总块数。

No.123报文中显示“This is an ACK to the segment in frame: 122”，表示这是对No.122报文的确认报文。

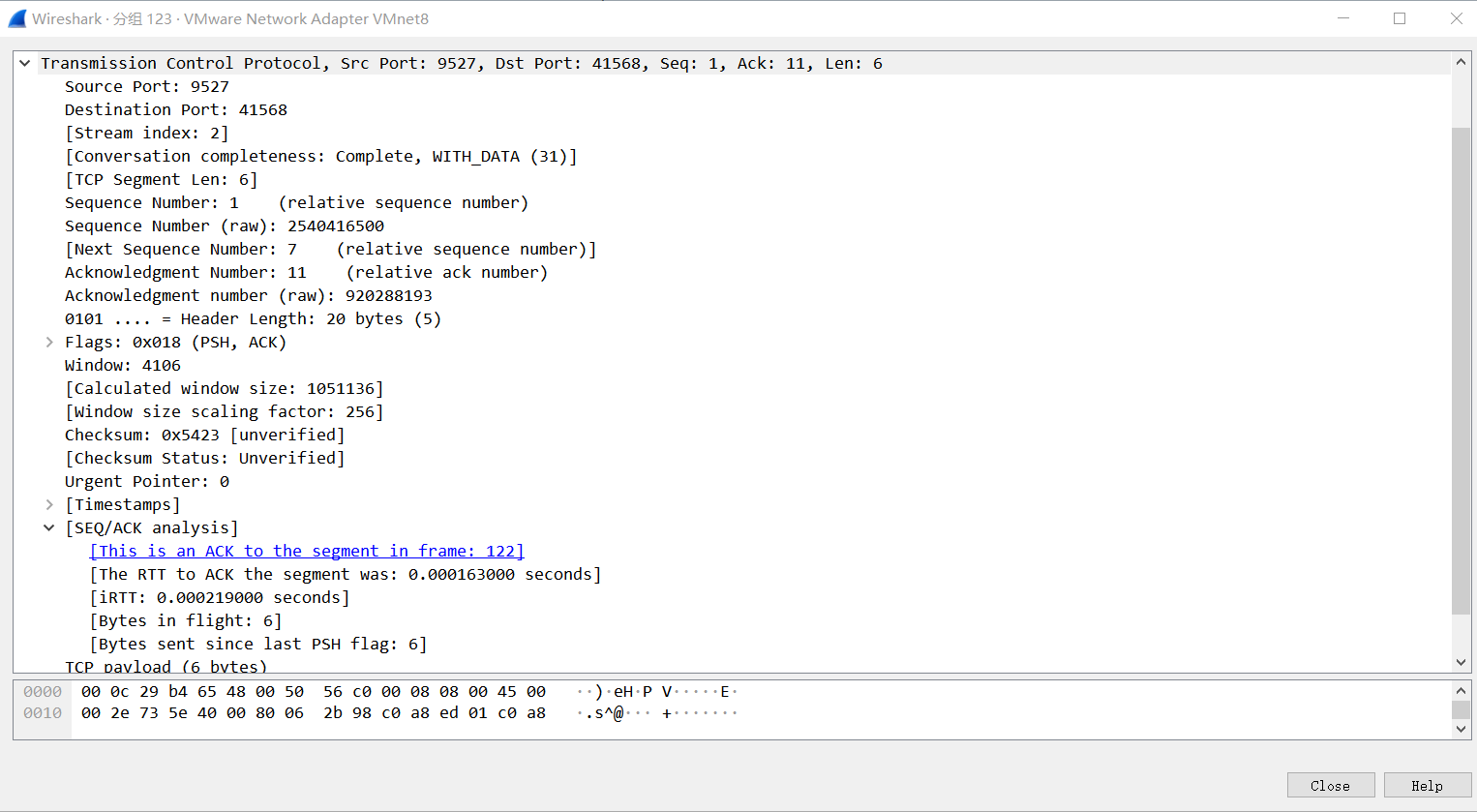


图4 对No.122报文的确认报文

No.124报文是服务器发给客户端的agree报文。

No.125报文中显示“This is an ACK to the segment in frame: 124”，表示这是对No.124报文的确认报文。

No.126报文是客户端发给服务器的reverseRequest报文。

No.127报文中显示“This is an ACK to the segment in frame: 126”，表示这是对No.126报文的确认报文。

No.128报文是服务器发给客户端的reverseAnswer报文。

No.129报文中显示“This is an ACK to the segment in frame: 128”，表示这是对No.128报文的确认报文。

其余数据包同理。

最后的四个数据包No.141-No.144模拟了TCP连接释放的过程，其中：

No.141报文是客户端第一次挥手（FIN）：客户端发送一个FIN报文给服务器，表明客户端已经没有数据要发送了，希望关闭连接。通常，这个FIN报文会有一个序列号，FIN标志位设置为1，seq=85。

No.142报文是服务器第二次挥手（ACK）：服务器收到客户端的FIN报文后，需要发送一个ACK报文作为响应。这个ACK报文的确认号会是客户端的FIN报文序列号加1，ack=86，是第一次挥手的序列号加1，表示服务器已经接收到客户端想要关闭连接的信号，并确认了该请求。

No.143报文服务器是第二次挥手（ACK）：服务器如果还有数据要发送，会继续发送直到数据全部发送完毕。然后，服务器也会发送一个FIN报文给客户端，表明服务器方面的数据也全部发送完毕，希望关闭服务器到客户端的方向的连接

No.144报文是客户端收到服务器的FIN报文后，会发送一个ACK报文给服务器，确认收到了服务器的关闭请求。这个ACK报文的确认号通常是服务器FIN报文的序列号加1。

1. 实现中的关键点：
2. 服务器端：

创建一个TCP socket，绑定到指定的IP地址和端口，并开始监听连接请求。并设置socket为非阻塞模式。服务器需要能够同时处理多个客户端的连接请求和数据处理，而不会因为一个客户端的延迟或阻塞而影响到其他客户端。通过select.select(inputs, outputs, inputs)，服务器可以监控多个输入和输出套接字，并且仅当套接字准备好进行下一步操作时才处理它们。这样可以同时接受新的连接请求并处理现有连接的数据传输。使用select进行非阻塞的I/O，这允许服务器同时处理多个socket连接。对于每个新的连接，服务器接受连接并将其加入到监控列表。如图5-1和5-2所示：



图5-1 创建服务器端

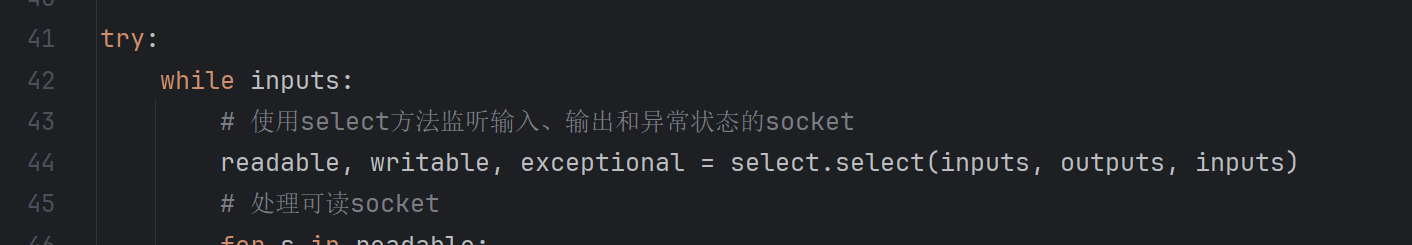


图5-2 设置非阻塞模式

1. 客户端：

客户端使用服务器的IP地址和端口号创建socket并建立连接。

读取本地文件，根据文件大小切分成多个块，并随机确定每块的大小，但保持在用户指定的最小和最大值之间。对于每块数据，发送包含待反转字符串的请求到服务器。

使用一个循环来分割文件。在每次循环中，使用random.randint(lmin, min(lmax, remaining\_size))来随机生成这一块的大小。这里的lmin和lmax是用户指定的最小和最大块大小，而remaining\_size是文件中剩余未分配的部分。通过这种随机分块，还可以增加传输过程的健壮性，避免因单一大块数据传输导致的潜在问题。如图6所示：

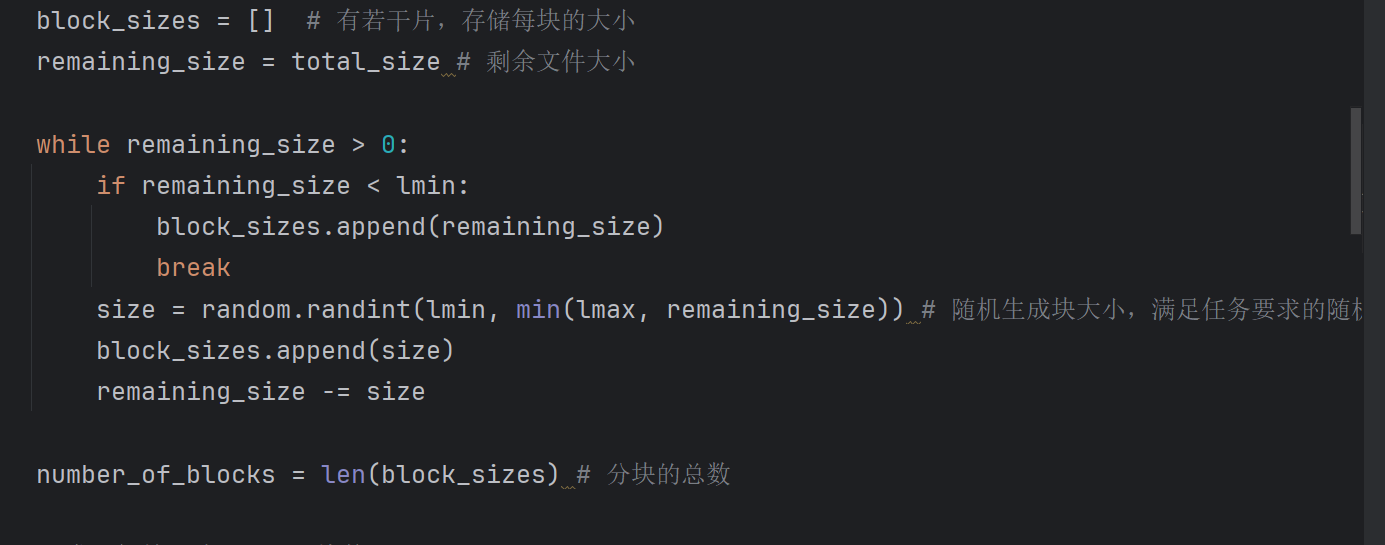


图6 随机数据块处理

1. 程序涉及了一系列网络编程的核心知识点和高级功能。
2. TCP网络编程：TCP是可靠的，面向连接的协议，通过使用 Python 的 socket 库来建立 TCP 连接，涉及到 socket 的创建、连接、发送数据、接收数据和关闭。正确使用IP地址和端口，理解 IP 地址和端口的概念以及它们如何在网络通信中被使用来标识网络上的设备和应用程序。
3. 数据处理和编码：对二进制数据处理，如使用 to\_bytes 和 from\_bytes 方法对整数进行字节转换。网络通信中的编码与解码：理解如何将字符串数据编码为字节以及如何将接收到的字节解码回字符串，基于网络通信与网络编程中常见的操作。
4. 自定义通信协议：创建一个简单的应用层协议，基于报文类型，用于封装发送的数据和解析接收到的数据，包括消息类型、长度和具体内容。
5. 文件操作：打开和读取文件内容，以及将处理结果写回到文件中，涉及到 Python 的文件操作方法。本次实验中更加注重随机性应用：使用 random 模块为文件传输中的数据块大小引入随机性，以适应不同的网络条件，避免由于网络波动导致的数据传输问题。
6. 非阻塞 I/O实现并发处理：通过设置 socket 为非阻塞模式，使得服务器可以在没有数据可读或写时不被阻塞，从而提高效率。通过使用 select 模块：利用 select.select 方法来监控多个套接字的状态，使服务器能同时处理多个连接的数据读写，这是实现高效并发服务器的关键技术之一。
7. Git 链接：