**传输流程：**

Socket\_clientr

socket\_server

bind()

listen()

accept()

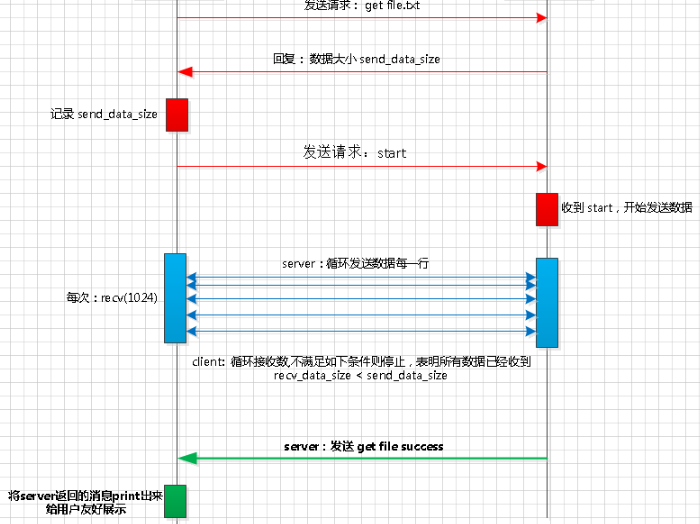
加密文件

解密文件

校验MD5

结束连接

read



close

close

print("server respoense:",server\_response)处粘包问题：

client已经接收了完成了所有文件数据，但且还处于阻塞状态，收不到server\_response。

查看 client 端收到的文件，发现 server\_response这个消息，被追加在了文件末尾。

因为client是通过 client.recv(1024)来进行接受数据，而1024表示，最多每次接受1024字节，

当文件本身所有数据小于1024时候，那么，最后的server\_response友好信息也会被client一次性接受，即此时：

data包涵了文件本身的所有数据+server\_response

而当代码走到print("server respoense:",server\_response)时，便会阻塞（实际上server已经将server\_response发送，只不过因为1024的原因，当做了文件本身数据）

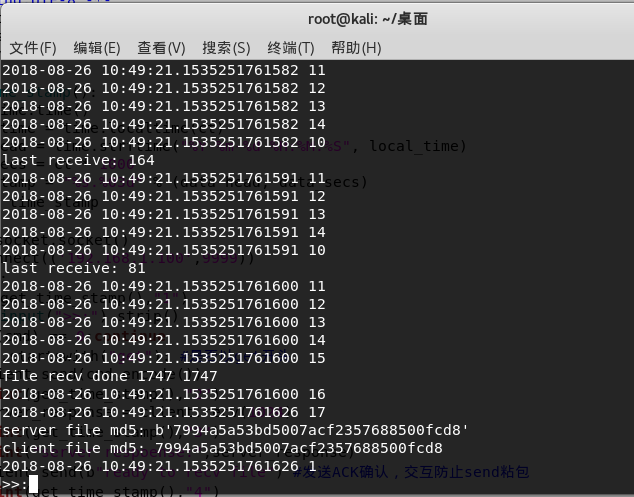
client.send(b"ready to recv file") 处发送ACK确认，可交互防止send粘包。

知识点：

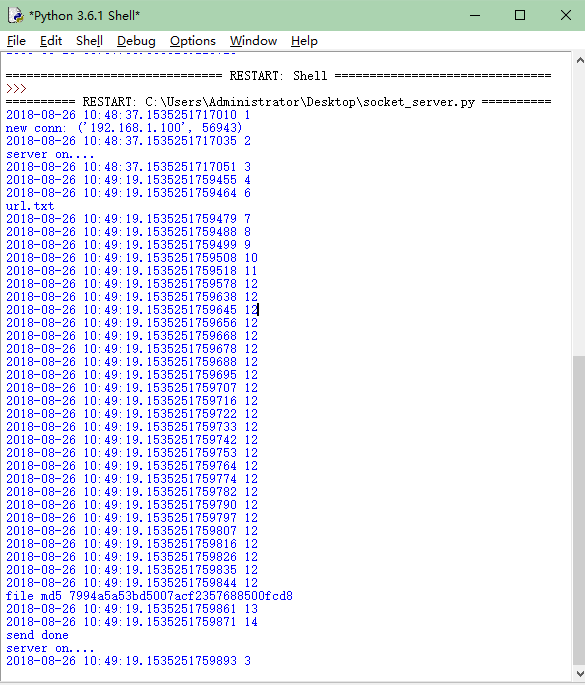
TCP的socket编程，收发两端（客户端和服务器端）都要有成对的socket，因此，发送端为了将多个发往接收端的包，更有效的发到对方，使用了优化方法（Nagle算法），将多次间隔较小、数据量小的数据，合并成一个大的数据块，然后进行封包。这样，接收端，就难于分辨出来了，必须提供科学的拆包机制。

TCP为了保证可靠传输，尽量减少额外开销（每次发包都要验证），因此采用了流式传输，面向流的传输，相对于面向消息的传输，可以减少发送包的数量，从而减少了额外开销。但是，对于数据传输频繁的程序来讲，使用TCP可能会容易粘包。

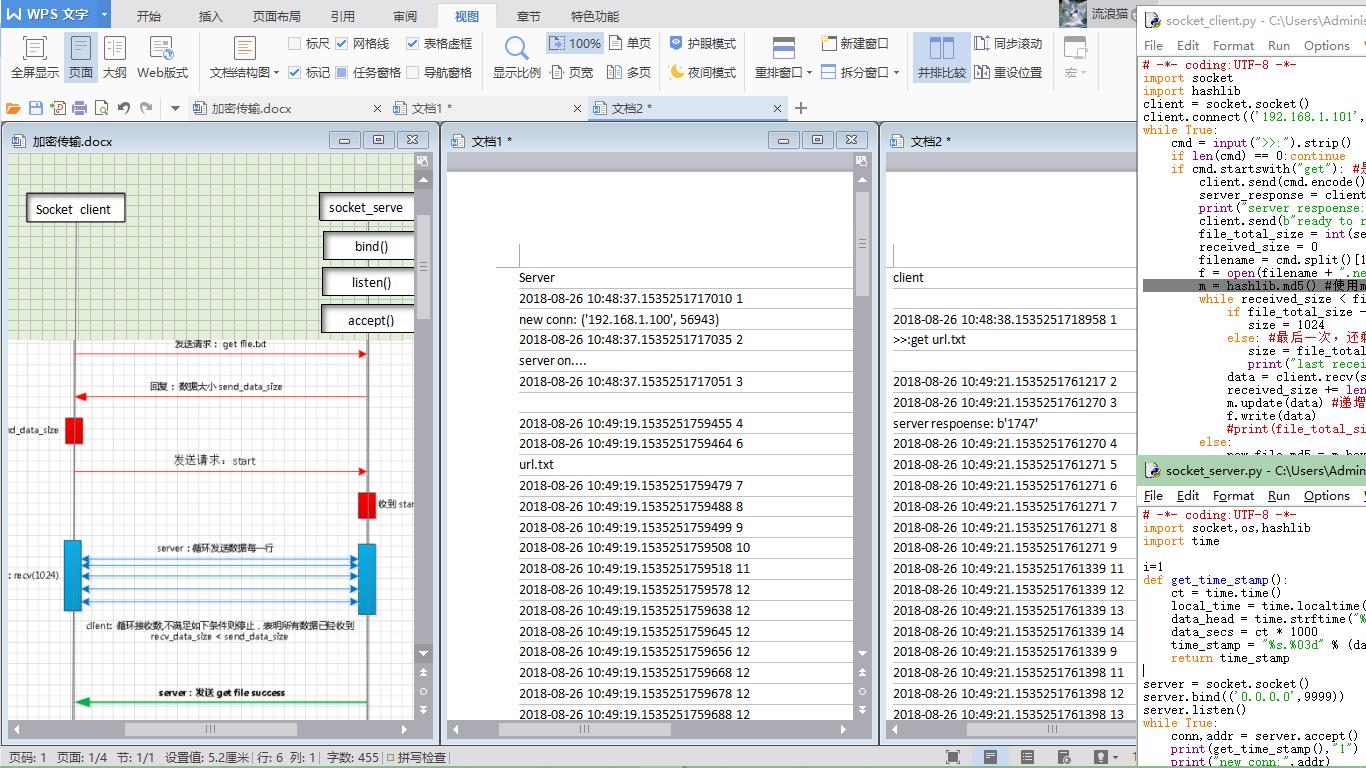
**Client\_log:**



**Servce\_log:**



**Timestamp\_proofreading:**



**加入加解密模块之后：**

**Client:**



**Server:**



报错：



































Get 技能&知识：

1、tcp协议基础

2、socket加解密传输&MD5校验

3、时间戳分析校验

4、文件读写

5、python3默认编码为unicode，由str类型进行表示，utf-8可以看成是unicode的一个扩展集，所以可以通过encode编码成utf-8。二进制数据使用byte类型表示，所以不会将str和byte混在一起。在实际应用中我们经常需要将两者进行互转：

str--->(encode)--->bytes，bytes--->(decode)--->str