计算机网络实验报告

--网络传输协议tcp，udp篇

学校：哈尔滨工业大学

班级：11403202

学号：1140320206

姓名：霍峻杰

实验3.1

一、实验题目:

用UDP实现一个时钟协议：

* 1. 客户端先发送一个消息给服务器，消息内容为客户端当前时间Ts
  2. 服务器收到消息，随机延迟L秒之后，获取当前时间Tn，再延迟L秒，发送给客户端当前时间Tn
  3. 客户端接收到应答时的本地时间为Tr，计算当前时间T：T=Tn+(Tr-Ts)/2

二、实验原理：

Udp是一种面向无连接的传输协议，它是一种有消息边界的协议，所以不存在粘包问题，实验3.2我们会详细讨论tcp的粘包情况和解决办法。Udp传输协议在发送端无法得知自己发送的数据报文是否被接收端顺利接收到，所以被大家称为”不可靠传输“。但是这种方式并没有因为它的”不可靠性”被摒弃，反而在很多方面得到广泛使用（如第二章所讲的dns等），因为他的传输速度非常的快。

Udp是无状态的连接，必然会遇到一些问题：

1. 传输的包可能丢失，没有到达接收端

解决办法：设置超时重传功能。

这里说明一下，超时重传的方法有很多种，至少书上就介绍了三种 方法，这里我选择了使用setsockopt（设置SO\_RCVTIMEO参数）的方法 来进行超时重传。

2. 客户端可能发送多个请求时间的包，不能判断传回的包对应哪个发 送包。

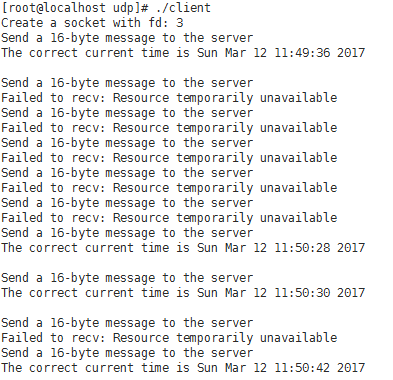
解决办法：为每一个发送包设置唯一标识。

这里我的解决办法是，发送udp包的时候在数据报文中加入一个随 机生成的数（我用了数组结构），这样相当于唯一标识了数据包， 返回时，只需要判断返回的包是否和发送包的随机数相等，即可将 这一对儿包对应起来。

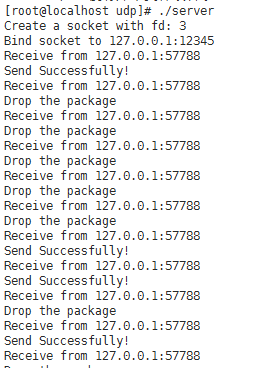
三、实验结果:

1. 这里我人为设置了一个丢包情况，就是当上述所说的随机数大于500（因为我的随机数生成范围是0-999）时，将这个数据包进行丢弃，我设定的超时重传时间为10s，下面看实验结果。

Client端：



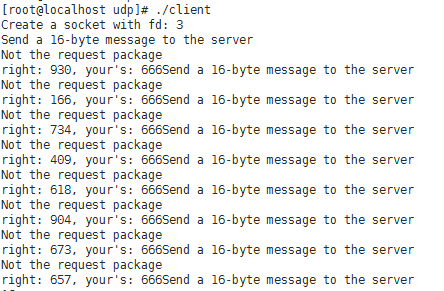
Server端：



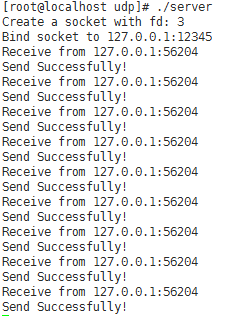
由实验结果可知，在包丢失的情况下，会进行超时重传，间隔为10s。

2. 为了验证客户端多包发送问题，我将server.c的代码中，固定回传的随机数为负数（使之不可能与发送端随机数相等），得到结果为丢弃包并进行下一次发送，直到随机数与发回的数相等（即对应包）。下图为实验结果：

Client端：



Server端：

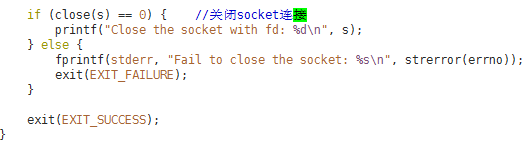


四、实验代码详解

Client.c







Server.c







实验3.2

一、实验题目:

用TCP实现一个文件传输协议，客户端将一个本地文件传输给服务器

需要考虑的问题：

1. 本地文件名和远程文件名可能不同

2. 同一个服务器可能同时接收来自多个客户的传输

3. 不要求传输文件内容之外的其他文件属性

二、实验原理：

Tcp是一种面向连接的“可靠”传输协议（其实也不完全可靠，报告里就不做讨论了），他需要通过三次握手建立起两个通信进程之间的联系（通道）。

Tcp协议被广泛应用于通信中，它有两种连接方式，一种是长连接，一种是短连接。长连接一般用于一对一的通信（如聊天框），短连接用于多个客户端连接一个服务器，连接传输一个文件后立即结束。

如果是长连接通信，会有一个问题----粘包。如何处理粘包问题呢，我思考出了四种解决办法：

1. 将文件名和文件内容打包成一个文件传输，中间加入分隔符，在接收端可以直接分离出文件名和内容，但是这种方法比较一般。

2. 设置参数，使tcp不等待缓存，直接发送。这种方式弊端非常大，传输效率低，违背了tcp设计的初衷。

3. 发送定常包，发送的每个包都是定长的（如512字节），接收方只需要每次接收512字节就可以。缺点很明显，发送了很多不需要的扩充数据，网络流量变大。

4. 根据换行符\n进行分割(readline)，方法类似1方法。

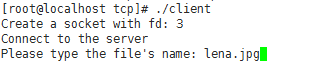
5. 发送每个包的时候都在前面带上发送数据的长度，采用一种结构体的方式，本实验就是采用这种方式，来防止粘包现象产生。

当多个客户端请求服务端时，服务端通过fork，产生一个自己的复制，叫做子进程，在子进程中关闭监听socket，进行操作，这样服务器为每个客户端连接提供了单独的，互不影响的连接。

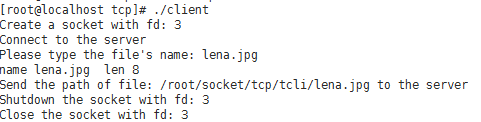
三、实验结果：

这里传输的是图形学经典图片lena，可根据客户端需要自定义传输后保存的文件名，也可以使用默认的（传送的原文件名）文件名，实验结果：

建立连接后，输入要传输的文件名：输入lena.jpg



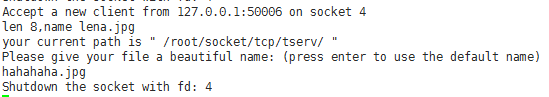
发送成功：



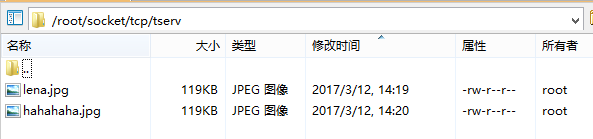
服务端可选自定义文件名，默认为传过来的原文件名：



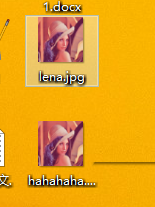
再传输一次，这次起个特殊的名字：



看一下传输是否成功：



把图片拷贝出来（因为我用的纯字符界面）

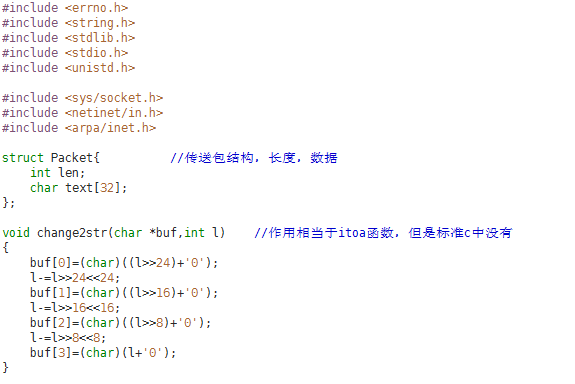


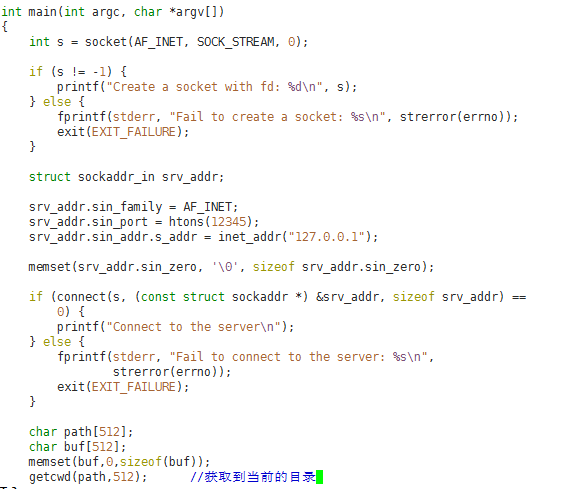
两张图片都是完整的，文件名也和预期的一样，试验成功。

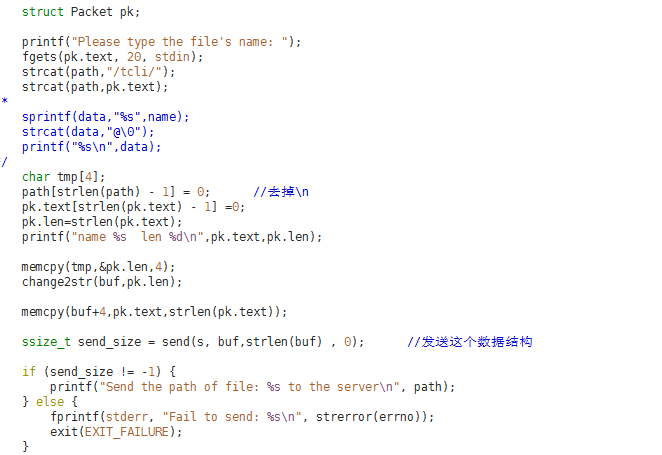
四、实验代码：

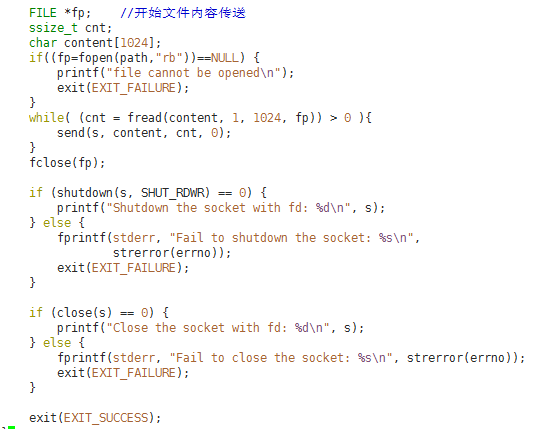
这次的代码详解我还是以注释的形式给出，方便。

Client.c



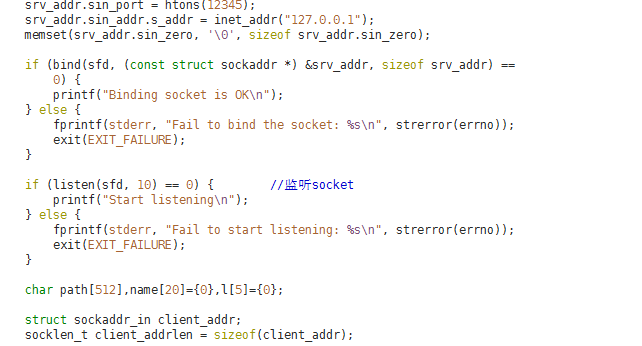






Server.c









实验总结：

这两个实验虽然看起来比较简单，但是实际操作起来，还是有很多需要注意的地方。粘包现象是之前写socket时遇到的情况，这次实验过程中，也深刻的认识了这种问题和解决方法。虽然理论上方法一定可行，但是在实践上，还是遇到了一些小困难，通过讨论和查资料都得以解决。总体来说，这是一次简单而又非常丰富的实验。

这个代码用的老师的模板，应该是简单的编写方式，我还在书中发现一些更加高级的编写方式，应该都可以应用到这两个实验中。

另外，提一句，这个代码用fgets获取shell输入相当的不安全，尝试代码输入一些backspace，会覆盖提示字符，甚至可以对提示字符进行伪造。