EXP3 Socket编程

汪楚文 2018202114 05/21/2020

实验概述

[实验目的]

·熟悉与socket通信相关的系统调用的使用

·通过编写程序理解进程通信机制

[基本要求]

在Linux环境下编写程序（不限编程语言），实现socket通信的基本流程

[具体要求]

·客户端进程和服务端进程可彼此收到对方发送的信息并显示出来

[进一步要求]

·实现多客户端连接（20分）

·程序除了通信之外具有一定其他功能，例：猜数字游戏、通信加密、文档上传与下载等（20分）

[实验报告要求]

详见文件README.txt

【重要】本实现的全部要求都已实现，其中通信在option1聊天室体现，并在聊天的基础上增加了聊天记录存档，管理员等功能。聊天时可多客户端连接，可以看到来自不同的人发来的消息，能达到社交软件的基本功能。option2为文件下载，支持下载服务器上的文件。

实验思路

[实验环境]

Ubuntu X86_64

[实验内容]

根据实验要求，本c++程序的主要设计思路如下。

本c++程序功能主要有聊天室和文件下载。用户可以通过menu中的选项来决定要进行的操作：聊天，文件下载，退出。代码实现主要是通过switch-case结构来调用不同的函数模块。与此同时，在server端初始时，亦有一menu，服务器端可以选择创建聊天房间，开启文件下载通道，或者退出。

主要函数模块如下：分为聊天室模块和文件下载模块。

一.聊天室设计

[client]

思路如下：通过pack的传输来实现与server端的数据传输。基于TCP协议的socket与服务器连接。并通过socket的返回值connectfd来判断是否socket成功。

若socket成功，则检查connect的返回值，判断是否与服务器连接成功。

若连接成功，则对用户输出提示信息，并让用户输入在聊天室中使用的姓名。

通过pid=fork()来创建子进程以支持用户的加入，

（1）若pid>0，则为父进程。经过test的错误检查后，用readn（read函数的包装）来将pack中的数据传送至服务器端。在父进程中，实现的是用户发出消息到服务器。此过程在while循环中，用户可以随时发消息。

（2）若pid=0，则为子进程，此时通过write，用户端可以接收到服务器端传来的提示信息，其中包括其他用户发来的聊天信息。

[server]

思路如下：进入socket监听，处理用户端发来的信息，后与client端设计一样。通过pid=fork()来实现不同功能。

（1）若pid>0，则为父进程。父进程的作用为用readn函数接收用户端传来的信息，并把其逐行保存在聊天记录文件record.txt中，此过程同样在while循环中，以实现随时接收数据。

（2）若pid=0，则为子进程，此时通过write，服务器端可以给用户端发送提示信息和其他用户发言。此过程同样在while循环中，以实现随时向用户端发送数据。

二.文件下载设计

[client]

连接服务器过程与聊天室设计相同基于TCP协议。

连接上以后，输入文件名，并放到缓冲区buffer中等待发送至服务器。

通过send函数向服务器发送储存在buffer里的要下载的文件名。

用户端初始化一个与服务器端同名的文件，等待写入。

通过recv函数从服务器接收数据到buffer中。

每接收到一段数据，便将buffer中的内容写到用户端文件，直至文件写完。

写入结束，fclose关闭文件。

[server]

接受连接请求，返回一个新的socket(描述符)，这个新socket用于同连接的客户端通信。

accept函数会把连接到的客户端信息写到client_addr中。

recv函数接收数据到缓冲区buffer，然后从buffer(缓冲区)拷贝到file_name中。

打开对应的文件并读取文件数据。

每读取一段数据，便将其发送给客户端，循环直到文件读完为止。

写入结束，fclose关闭文件。

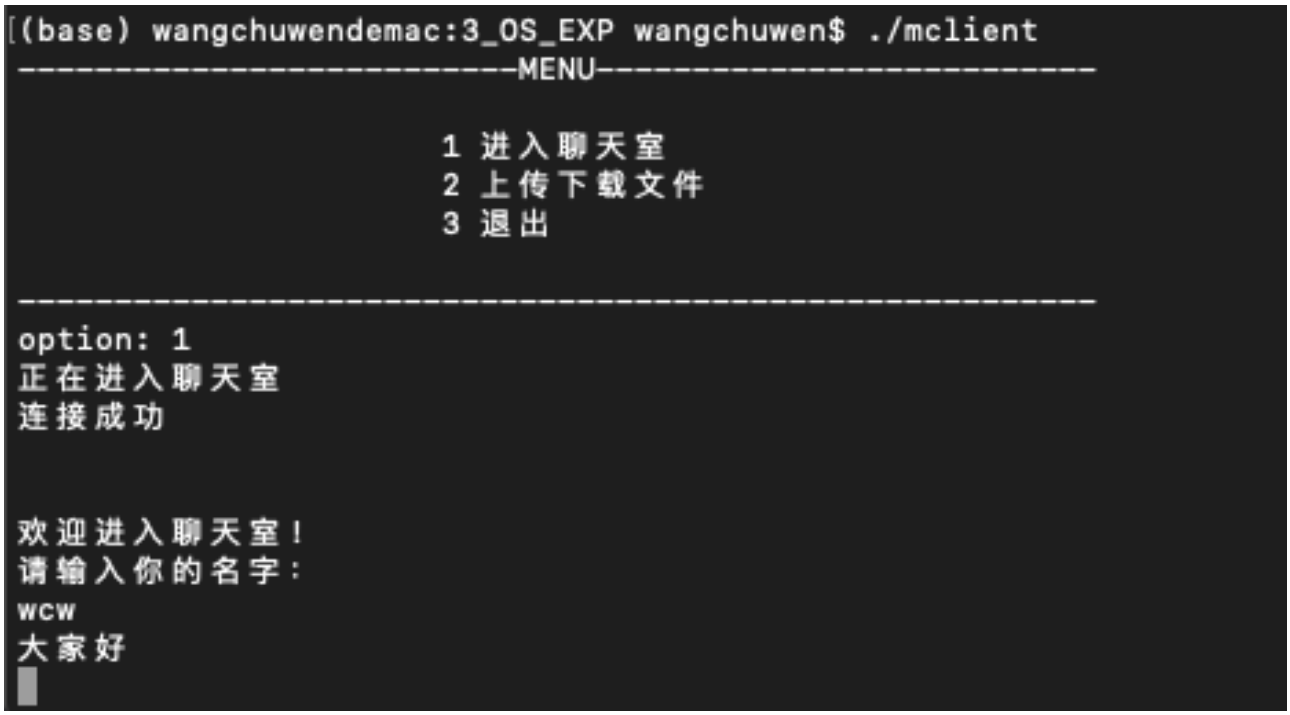
三.主函数

server和client的主函数都是在一个while框架之下，反复调用menu来让用户端和服务器端选择相应的功能。switch-case结构来执行对模块。用户界面详见实验结果。

实验结果

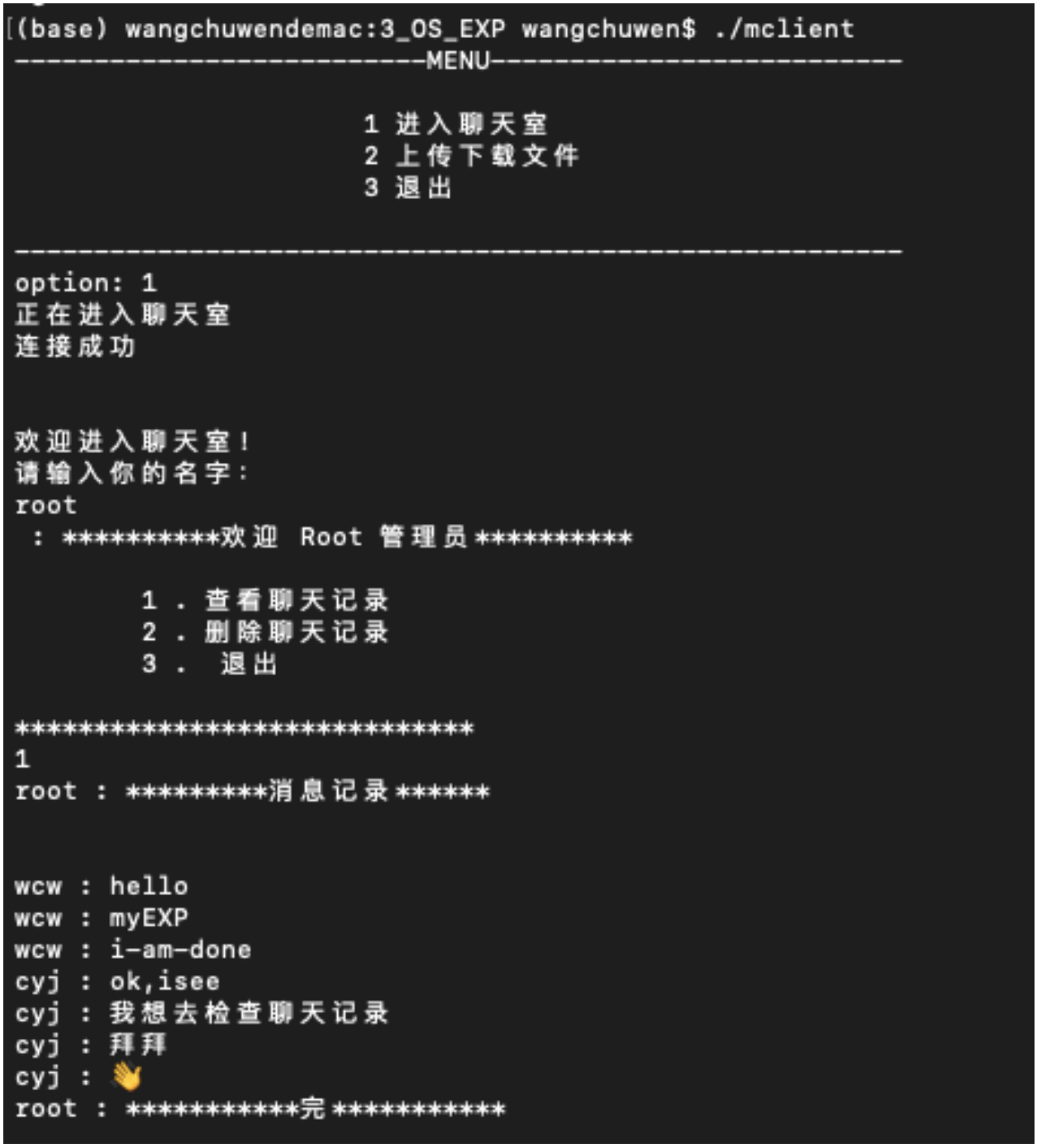
一.聊天室

[client]普通用户



[server]普通用户

[client]管理员root



二.文件下载

[client]

```
(base) wangchuwendemac:3_OS_EXP wangchuwen$ ./mclient
-------------------------MENU-------------------------

                        1 进入聊天室
                        2 上传下载文件
                        3 退出

------------------------------------------------------
option: 2
Please Input File Name On Server:       pipe.c
Receive File:   pipe.c From Server IP Successful!
-----------------------------MENU-----------------------------
```

[server]

```
ubuntu@VM-0-17-ubuntu:~/3_OS_EXP$ ./mserver
-------------------------MENU-------------------------

                        1 创建聊天室
                        2 开启文件下载
                        3 退出

------------------------------------------------------
option: 2
已打开下载通道，等待用户请求...
pipe.c
File:pipe.c Transfer Successful!
```

实验中遇到的问题及解决办法

一.在聊天室的设计过程中，不知道怎么去在接收用户发来的数据的同时又能把其他用户发来的信息转发给用户。

解决方法：通过使用pid=fork（）来创建子进程，父进程用来接收用户端发来的数据并处理数据，子进程用来向用户端发送提示信息以及转发其他用户的消息。子进程和父进程同步运行，检测到不同的pid时便可执行相对应的操作。这样一来，就可以边接受消息边发送消息了。

二.在文件下载的模块中，不知道文件下载是怎么进行的。刚开始是以为把scp命令写进c++代码里，然后通过代码的编译执行来把服务器的文件scp过来。

解决方法：在了解了相关情况后，发现socket文件下载的工作原理并不是scp或者其他文件传输方式，而是在用户端创建一个新的同名文件，然后通过buffer把服务器原始文件的数据一行行的写入用户端的文件之中。

三.在实现多用户接入时，不知道如何下手。错误的理解多客户端连接。

解决方法：由于本程序很多地方都是通过fork来实现，因此不论用户端有多少个，理论上都应该能正常进行，因为每一次fork都能产生一个新的进程对此用户的请求进行处理。于是在相应的地方套上while循环，即可实现多用户端的接入，且互相不干扰。

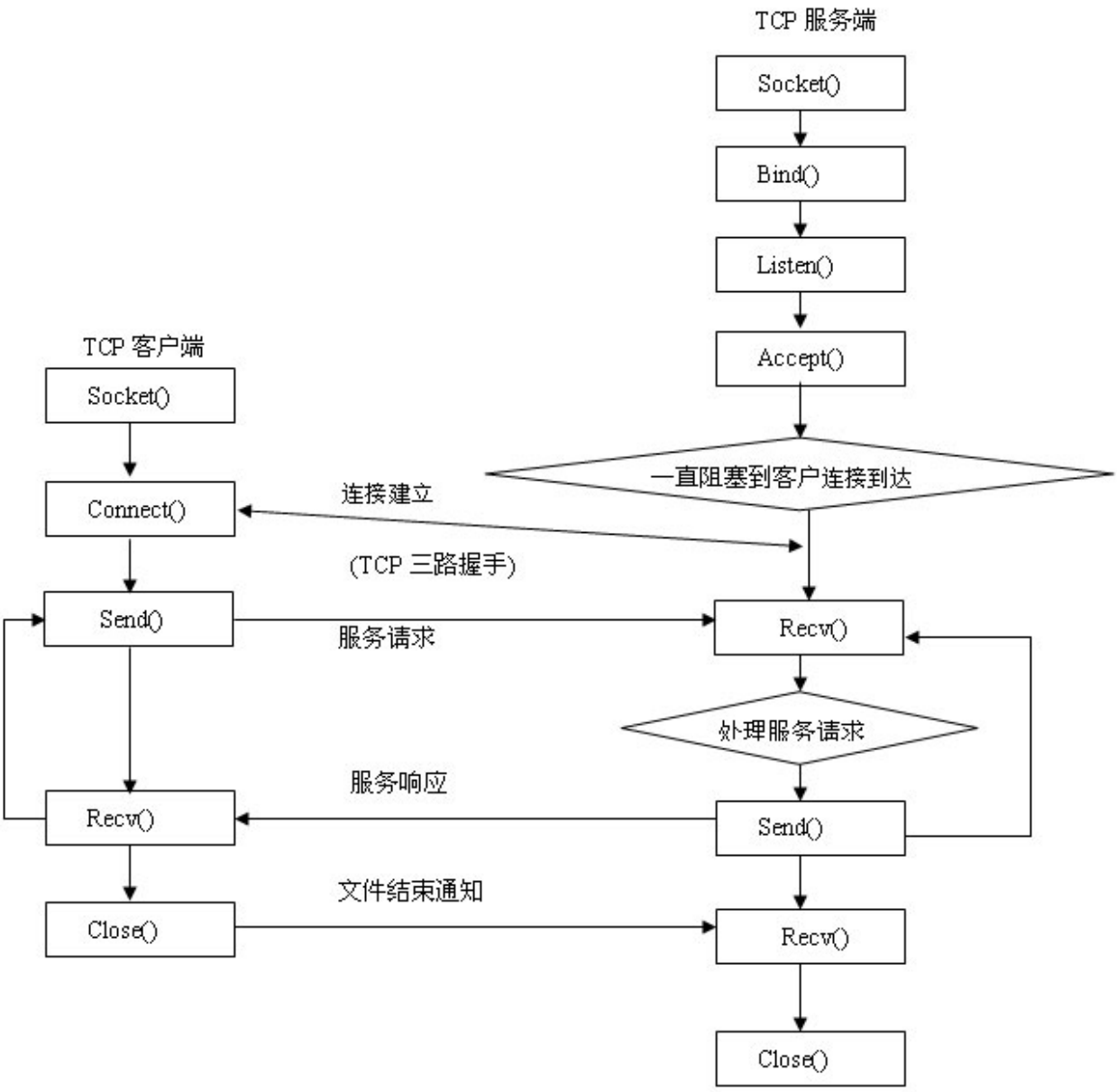
四.由于做了两个模块，聊天室和文件下载。但是这两者对用户发来的数据都要进行不同的处理，之后还要向客户端发送不同的信息。所以如果一开始就同时打开了聊天室和文件下载两个通道，由于监听的过程中难以切换不同的功能，所以势必会造成只有一个功能能用的结果。

解决方法：通过在server端设置一个menu，让服务器的管理员手动进行调整。若有用户想要进入聊天室，则对其打开聊天室的通道，与此同时，服务器进入监听聊天的状态；如果有用户想要文件下载，则对其打开文件下载的通道，与此同时，服务器进入文件下载监听。这样一来，上述问题就可以得到解决。可以进行聊天室和文件下载，且互不干扰。

TCP协议和UDP协议

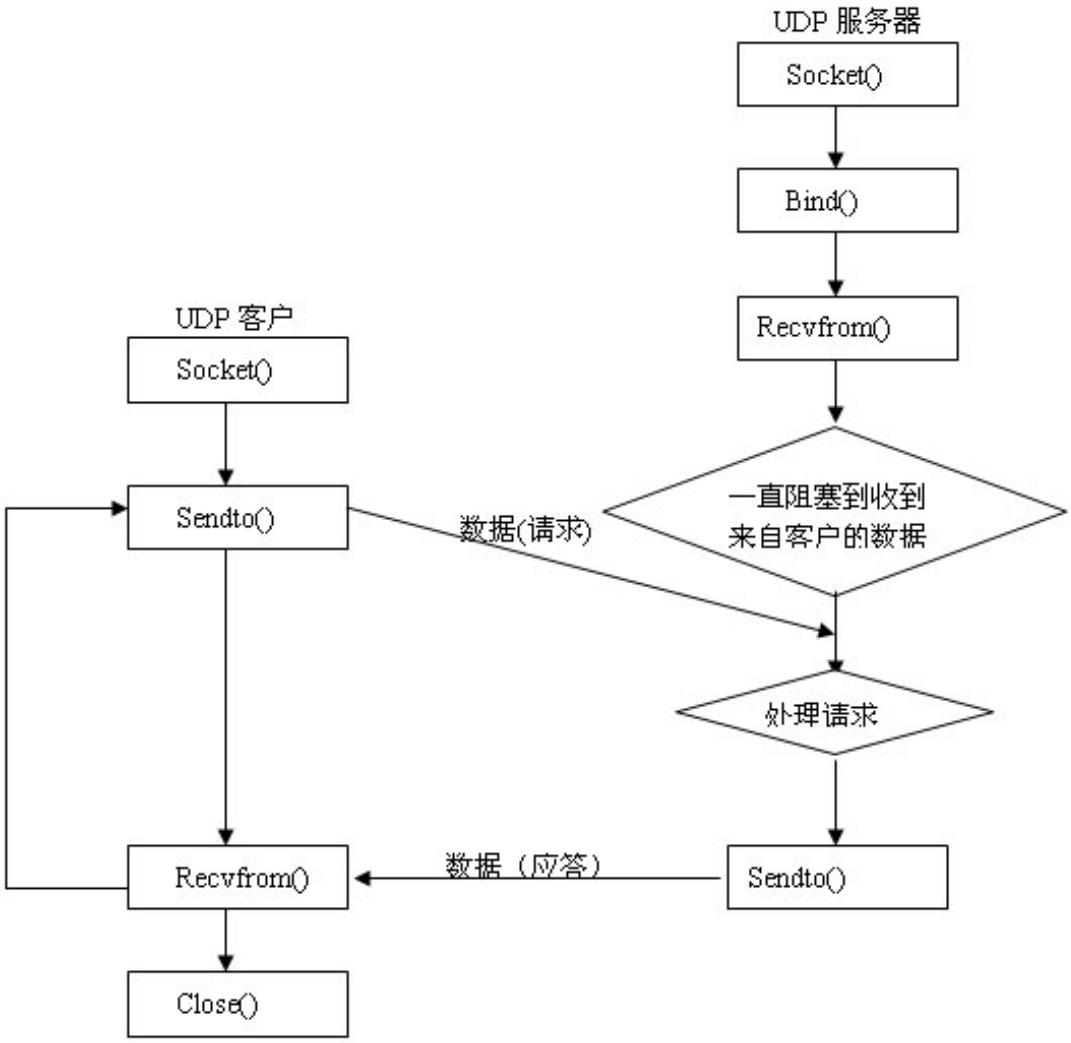
[TCP]

流程图:



[UDP]

流程图：



1.从上面的流程图比较我们可以很明显的看出UDP没有三次握手过程。
2.TCP具有高可靠性，确保传输数据的正确性，不出现丢失或乱序；UDP在传输数据前不建问立连接，不对数据报进行检查与修改，无须等待对方的应答，所以会出现分组丢失、重复、乱序，应用程序需要负责传输可靠性方答面的所有工作；UDP处理的细节比TCP少。
3.UDP不能保证消息被传送到（它也报告消息没有传送到）目的地。UDP也不保证数据包的传送顺序。UDP把数据发出去后只能希望它能够抵达目的地。因此网络开销也小，时效性高
4.TCP是面向连接的传输控制协议，而UDP提供了无连接的数据报服务；UDP通信不关心目的主机是否可达，目的主机是否能接收该数据包，以及数据包是否完整无误的发送给客户端。
5.UDP 不能直接使用read/write来处理数据，因为UDP无法知晓目的主机的ip和端口号，需要使用sendto或sendmsg,调用时已参数的方式列出。

UDP与TCP在编程区别

TCP编程的服务器端一般步骤是：

1、创建一个socket，用函数socket()；
2、设置socket属性，用函数setsockopt(); * 可选
3、绑定IP地址、端口等信息到socket上，用函数bind();
4、开启监听，用函数listen()；
5、接收客户端上来的连接，用函数accept()；
6、收发数据，用函数send()和recv()，或者read()和write();
7、关闭网络连接；
8、关闭监听；

TCP编程的客户端一般步骤是：

1、创建一个socket，用函数socket()；
2、设置socket属性，用函数setsockopt();* 可选
3、绑定IP地址、端口等信息到socket上，用函数bind();* 可选
4、设置要连接的对方的IP地址和端口等属性；
5、连接服务器，用函数connect()；
6、收发数据，用函数send()和recv()，或者read()和write();
7、关闭网络连接；

与之对应的UDP编程步骤要简单许多，分别如下：

UDP编程的服务器端一般步骤是：

1、创建一个socket，用函数socket()；
2、设置socket属性，用函数setsockopt();* 可选
3、绑定IP地址、端口等信息到socket上，用函数bind();
4、循环接收数据，用函数recvfrom();
5、关闭网络连接；

UDP编程的客户端一般步骤是：

1、创建一个socket，用函数socket()；
2、设置socket属性，用函数setsockopt();* 可选
3、绑定IP地址、端口等信息到socket上，用函数bind();* 可选
4、设置对方的IP地址和端口等属性;
5、发送数据，用函数sendto();
6、关闭网络连接；

UDP与TCP调用参数设置区别

（1） UDP Server不需要调用listen和accept
（2） UDP收发数据用sendto/recvfrom函数
（3） TCP：地址信息在connect/accept时确定
（4） UDP：在sendto/recvfrom函数中每次均 需指定地址信息
（5） UDP：shutdown函数无效

项目相关文件说明

OS_EXP3.pdf----------实验报告

README.txt----------实验要求文件

mserver.cpp----------服务器端源代码文件

mclient.cpp----------客户端源代码文件

超链接：点击此处可在github中查看项目：）