《网络与通信》课程实验报告

实验 2：Socket 通信编程

姓名	孔馨怡	院系	计算机学院	学号	22122128
任课教师	何冰		指导教师	何冰	
实验地点	计 708		实验时间	10 月 9 日 周三	
实验课表现	出勤、表现得分(10)		实验报告得分(40)		实验总分	
	操作结果得分(50)					

实验目的：

1. 掌握 Socket 编程过程；
2. 编写简单的网络应用程序。

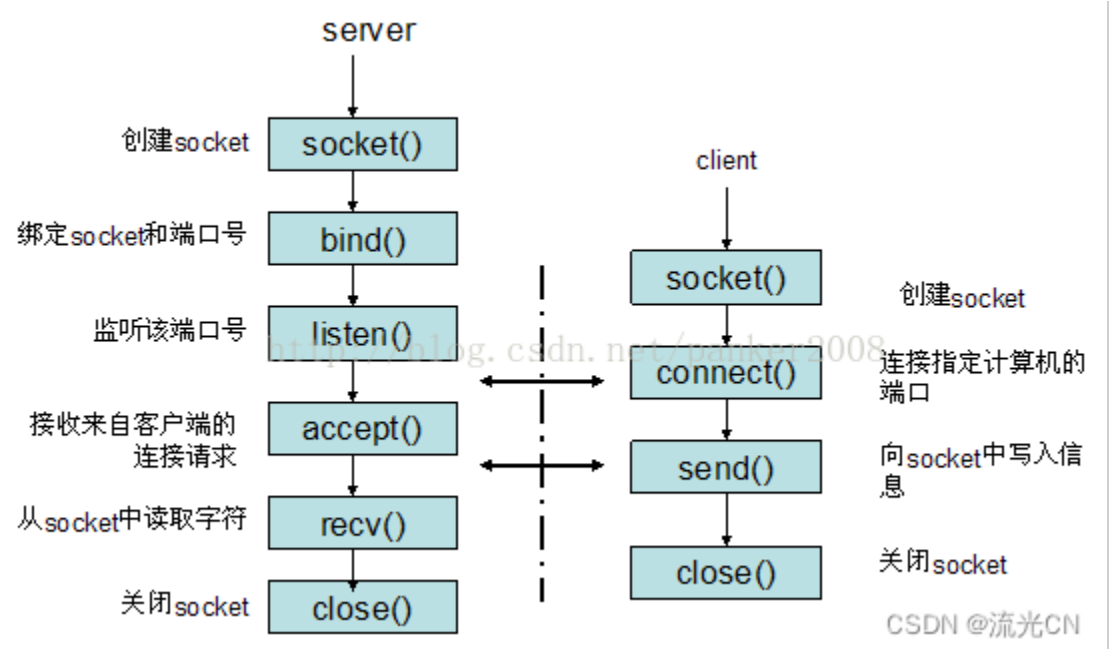
实验内容：

利用你选择的任何一个编程语言，分别基于 TCP 和 UDP 编写一个简单的 Client/Server 网络应用程序。具体程序要求参见《实验指导书》。

要求以附件形式给出：

- 系统概述：运行环境、编译、使用方法、实现环境、程序文件列表等；
- 主要数据结构；
- 主要算法描述；
- 用户使用手册；
- 程序源代码；

实验要求：（学生对预习要求的回答）（10 分）　　得分：



图一 服务器&客户端界面大致流程

- **Socket编程服务器端的主要步骤：**

socket(): 创建一个Socket。服务器端首先需要通过 socket() 函数创建一个套接字，它是通信的基础。套接字会创建一个文件描述符，用于标识该网络通信端点。

```
函数原型int socket(int domain, int type, int protocol);
//domain：通信域，type：指定套接字的类型 ，protocal:给定的通信域和套接字类型选择默认协议
```

bind(): 将Socket绑定到特定的IP地址和端口号。使它成为一个固定的监听点。客户端可以通过该IP和端口连接到服务器。

```
函数原型int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

listen(): 让服务器开始监听来自客户端的连接请求。当客户端发起连接请求时，服务器能够接收到。

```
函数原型int listen(int sockfd, int backlog);
// backlog 用来描述 sockfd 的等待连接队列能够达到的最大值
```

accept(): 接受客户端的连接请求，建立连接。accept() 阻塞并等待客户端发起的连接请求，一旦有客户端请求连接，服务器会返回一个新的Socket，用于和该客户端进行通信。原始的Socket继续监听新的连接。

```
函数原型int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
// 参数 addr 是一个传出参数，参数 addr 用来返回已连接的客户端的 IP 地址与端口号等这些信息。
```

recv(): 从客户端接收数据。这里用到循环监听（后续问题2会说）

```
函数原型ssize_t recv(int sockfd, void *buf, size_t len, int flags);
// 返回值是接受到的字节数
// 参数 buf 指向了一个数据接收缓冲区，参数 len 指定了读取数据的字节大小，参数 flags 可以指定一些标志用于控制如何接收数据。
```

Send(): 发送信息给客户端。（除了send可以用written函数，这里服务器和客户端都是用这些函数发送数据）

```
函数原型ssize_t send(int sockfd, const void *buf, size_t len, int flags);
// 参数 buf 指向了一个数据接收缓冲区，参数 len 指定了读取数据的字节大小，参数 flags 可以指定一些标志用于控制如何发送数据。
```

close()：关闭Socket连接。当通信结束时，服务器需要关闭客户端的Socket和服务器Socket，以释放系统资源。

- **Socket编程客户端的主要步骤：**

socket(): 创建一个Socket。

connect(): 连接到服务器的IP地址和端口号。发起与服务器的连接请求。它指定服务器的IP和端口，尝试与服务器建立通信连接。

send(): 向服务器发送数据。

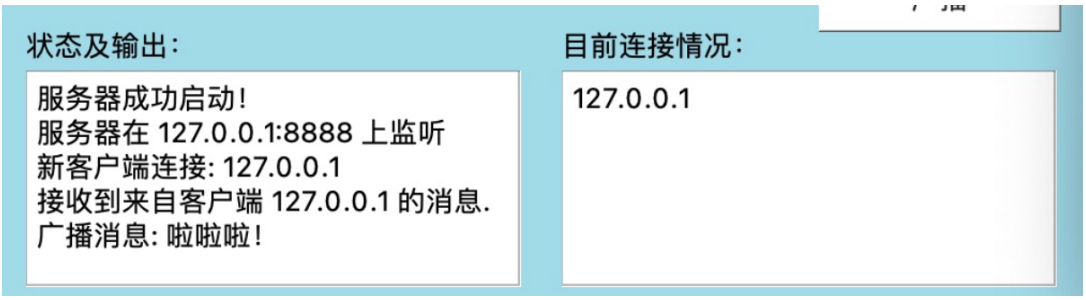
close(): 关闭Socket连接。释放资源。

实验过程中遇到的问题如何解决的？（10 分）	得分：

问题 1：在用 cpp 实现最基本的实现以后，为了让服务器和客户端的界面更加清楚，各个接受消息和发送消息更加灵活、自由，我采取了 Qt Creator +CPP 的实现。在 QT Creator 实现过程中，我为了增加“服务器实时显示在线用户列表”的功能，发现多个客户端连接的时候一直是一个 IP 这样的可视化就体现不出来多个客户端在线服务器：

（1） 回环地址（loopback address）/本地地址

它指向本机，即发送的网络请求会回到自己。当我的服务器和客户端都在我的电脑上的时候他们都是 127.0.0.1 的 IP 地址，这代表本地地址，如果我无法解决用别处 IP 的客户端来登录服务器，则所有客户端 IP 都是和服务器是一样的，得想其他方式标识各客户端来实现我的用户列表功能。及下图，无论多少个本地客户端连接，连接列表只有 127.0.0.1:



图二 服务器界面显示客户端唯一 IP

（我老服务器的截图没有截，代码更新了，反正就是所有客户端都一个 IP 列表只有一个）

在我的服务器界面创建服务器的时候给定 IP 除了可以是 127.0.0.1，也可以是 0.0.0.0，这表示非指定地址/所有网络接口/不确定的地址，这时候客户端依旧可以通过连接 127.0.0.1 来连接服务器；但是相反的，服务器是 127.0.0.1 或者 0.0.0.0，客户端要连接的服务器 IP 不能是 0.0.0.0，这时候会看见客户端显示“连接成功”后立刻断开了，连接的不是我们这个指定的服务器，因为 0.0.0.0 代表所有网络接口。

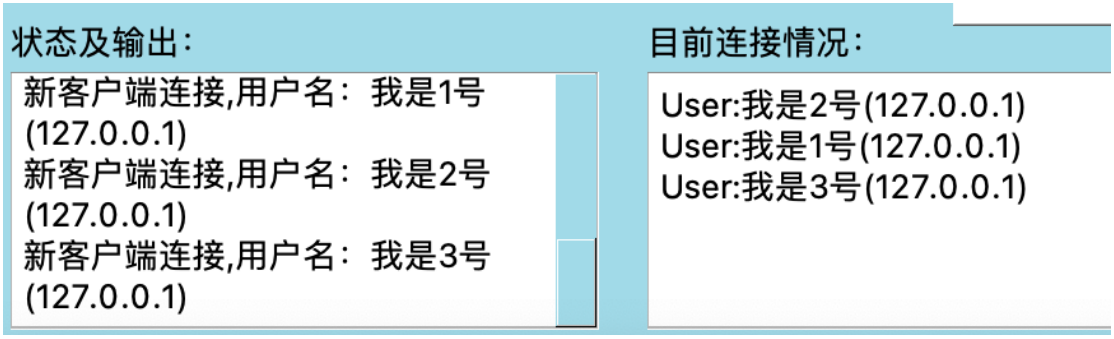
图三 服务器&客户端界面当两边都为 0.0.0.0 的时候情况显示

（2） 用户名 ID 的添加，绑定 socket 指针实现唯一客户端标识和其他功能

如上图二，在本地服务器与客户端共用相同 IP 的情况下，我通过添加用户名 ID 来标识每个客户端。每个客户端的用户名由用户在客户端界面的用户名输入框中填写，并将其与该客户端唯一的 Socket 指针绑定到一个结构体中。通过这种方式，实现了每个客户端的唯一标识与区分。

这种设计不仅确保了客户端之间的有效区分，还实现了“用户连接情况名单实时更新”的功能。相比之下，如果通过 IP 地址发送数据而没有绑定唯一的 Socket 指针，在服务器进行广播时可能会引发混乱：例如，某个客户端可能会收到多条消息，而其他客户端却无法接

收到。这是因为 IP 地址并非唯一标识符，而 Socket 指针在底层实现中具有唯一性。通过绑定每个客户端的 Socket 指针，可以确保消息的准确传递和广播功能的正常实现，而用户名则为用户提供了直观的界面，使其更易于理解和观察客户端连接状态。



图四 服务器界面实时显示有用户名标识的连接用户名单，以及各用户情况

问题 2：如何让多个客户端同时连接服务器并且一直能够接收新连接，**即如何实现多个客户端的并发连接：**

为了让服务器能同时处理多个客户端的连接，需通过创建新线程来处理每个客户端的请求。服务器主线程持续监听新的客户端连接，并在每个连接建立后，创建一个独立的线程来处理该客户端的通信。这样，服务器可以继续接受新的连接请求，而不会被阻塞。

持续监听连接：

服务器通过 accept() 函数接收客户端的连接请求。通过循环结构，服务器能够持续监听新的连接，而不会因为处理某个客户端的请求而阻塞后续连接。

并发处理客户端：

每当有新的客户端连接时，服务器创建一个新的线程，通过线程来单独处理与该客户端的通信，确保服务器主线程不受影响。这种方法使得服务器能够同时处理多个客户端的请求，实现并发处理。

```
// 创建线程来单独处理
        pthread_t tid;
        if (pthread_create(&tid, nullptr, handle_client, client) != 0)
        {
            perror("pthread_create error.");
            close(*client);
            delete client; // 清理内存
            continue;
        }
        pthread_detach(tid); // 分离线程，自动回收资源
```

循环接收数据的必要性：

在每个客户端的通信处理中，服务器需要持续接收和发送数据。通过循环结构，服务器能够不断接收客户端发送的数据，直到客户端断开连接为止。

```
while (true)
    {
        *client = accept(socket_fd, (struct sockaddr *)&client_addr,
&client_addrlen);
        ......
        // 创建线程来单独处理
```

```
        pthread_t tid;
        if (pthread_create(&tid, nullptr, handle_client, client) != 0)
        {
            perror("pthread_create error.");
            close(*client);
            delete client; // 清理内存
            continue;
        }
        pthread_detach(tid); // 分离线程，自动回收资源
    }
```

错误处理和资源管理：

如果在接收过程中发生错误，服务器需要妥善处理这些错误，避免崩溃或资源泄漏。通过 continue; 来跳过错误并继续监听下一个连接，可以提高服务器的健壮性。

使用 pthread_detach() 使线程资源能够在任务结束时自动回收，避免内存泄漏。

问题 3： 无

本次实验的体会（结论）（10 分）	得分：

在本次 Socket 通信实验中，我深入了解了 Socket 的基本原理及其在网络编程中的应用。通过实际编程实践，成功实现了服务器与客户端之间的通信，掌握了使用 Qt 框架中的信号与槽机制来处理网络事件。这一机制不仅简化了代码结构，也提高了代码的可维护性。特别是在动态管理多个客户端连接时，能够及时响应连接和断开事件，使得程序更加高效。

在用 C++实现 Socket 底层代码后，我对 TCP/UDP 协议有了更深入的理解。为了优化设计界面和用户交互体验，我快速掌握了 Qt Creator 的开发流程，虽然封装好的函数非常便捷，但我也意识到底层逻辑的掌握同样重要。实验中我加入了用户名、登录列表等功能设计，并对各种细节问题进行了完善，最终实现了更健全的代码结构和用户体验。这次实验不仅提升了我对 Socket 通信的理解，也增强了编程能力和代码优化的意识。

思考题：（10 分）

思考题 1：（4 分）	得分：

你所用的编程语言在 Socket 通信中用到的主要类及其主要作用。

CPP 底层开发的时候，没有涉及主要类，**主要函数在前面实验要求处已经介绍。**

在 Socket 通信中，使用 C++和 Qt 框架时，主要涉及到的类及其作用如下：

1. QTcpServer

主要作用：用于创建 TCP 服务器，监听来自客户端的连接请求。

listen(const QHostAddress &address, quint16 port)：开始监听指定的 IP 地址和端口。

nextPendingConnection()：获取下一个待处理的连接（返回一个 QTcpSocket 指针）。

2. QTcpSocket

主要作用：用于创建 TCP 客户端和与 TCP 服务器的通信。

connectToHost(const QString &hostName, quint16 port)：连接到指定的主机和端口。

write(const QByteArray &data)：发送数据到连接的服务器。

readAll()：读取接收到的数据。

disconnectFromHost()：断开与服务器的连接。

信号：

readyRead()：当有新数据可读时发射。

disconnected()：当与服务器断开连接时发射。

3. QHostAddress

主要作用：表示一个网络地址（IP 地址）。

QHostAddress::Any：表示任意可用的 IP 地址。

QHostAddress::LocalHost：表示本地回环地址（127.0.0.1）。

QHostAddress(const QString &address)：根据字符串形式的 IP 地址构造。

4. QByteArray

主要作用：用于存储和处理字节数组。

QByteArray::fromStdString(const std::string &str)：从标准字符串创建字节数组。

QByteArray::toStdString()：将字节数组转换为标准字符串。

5. QObject

主要作用：Qt 中所有对象的基类，提供信号和槽机制。

connect()：连接信号和槽。

deleteLater()：标记对象在事件循环的下一个迭代中被删除。

具体看代码文件！

思考题 2：（6 分） 得分：

说明 TCP 和 UDP 编程的主要差异和特点。

TCP（Transmission Control Protocol）

特点：

1. **面向连接**：TCP 是一种面向连接的协议，客户端和服务器之间必须先建立连接（三次握手）才能进行数据传输。在断开连接时，需要通过四次挥手关闭连接。
2. **可靠性高**：TCP 通过确认机制（ACK），确保数据包按序到达且不会丢失或重复传输。如果数据包丢失，发送方会重发，直到收到确认。TCP 还能保证数据的完整性和正确性。
3. **流量控制与拥塞控制**：TCP 有流量控制（如滑动窗口机制）和拥塞控制（如慢启动、拥塞避免等），可以防止网络拥塞并适应不同的网络带宽。
4. **数据传输有序**：TCP 确保数据按发送顺序到达接收端，这对于需要按顺序处理的数据非常重要。
5. **适用场景**：TCP 适用于对可靠性要求较高的应用场景，例如文件传输、电子邮件、网页浏览等。

编程中的特点：

- **socket 类型**：使用 SOCK_STREAM 作为套接字类型。
- **连接管理**：必须使用 connect()（客户端） 和 listen()/accept()（服务器）进行连接的建立与管理。
- **数据收发**：通过 send() 和 recv() 函数进行数据的发送和接收，数据传输过程中会有确认和重发机制。

优点：

- 数据传输可靠性高。
- 确保数据按序到达且没有丢包现象。

缺点：

- TCP 有额外的连接管理开销，效率较低。
- 适应网络带宽变化时，传输速度可能较慢。

UDP（User Datagram Protocol）

特点：

1. **无连接**：UDP 是一种无连接的协议，客户端和服务器之间无需建立连接即可直接发送数据。这也意味着 UDP 没有三次握手和连接断开机制。
2. **不可靠传输**：UDP 不提供数据包的确认、重传、顺序控制等功能，因此数据包可能会丢失、重复或乱序到达。
3. **没有流量控制与拥塞控制**：UDP 不具备 TCP 的流量控制和拥塞控制机制，数据包发送的速度完全由发送方决定，可能导致网络拥塞。
4. **数据传输快速**：由于 UDP 不进行复杂的连接管理和流量控制，传输速度较快，适用于实时性强但不要求高可靠性的应用场景。
5. **适用场景**：UDP 适用于对传输速度和实时性要求高，但对数据可靠性要求不高的场景，例如视频直播、在线游戏、语音通话等。

编程中的特点：

- **socket 类型**：使用 SOCK_DGRAM 作为套接字类型。
- **无需连接管理**：直接使用 sendto()（客户端） 和 recvfrom()（服务器）进行数据传输，不需要建立和断开连接。
- **数据收发**：每个数据包的发送和接收是独立的，没有顺序保证和重传机制。

优点：

- 传输速度快，开销小。
- 适用于对实时性要求高的场景。

缺点：

- 传输不可靠，数据包可能丢失或乱序。
- 不能保证所有数据包按顺序到达。

指导教师评语：

日期：