计算机网络实验报告

警示

1.实验报告如有雷同，雷同各方当次实验成绩均以 0 分计。
2.当次小组成员成绩只计学号、姓名登录在下表中的。
3.在规定时间内未上交实验报告的，不得以其他方式补交，当次成绩按 0 分计。
4.实验报告文件以 PDF 格式提交。

院系	计算机学院	班 级	计科 2 班	组长	林隽哲
学号	21312450	22365043	22302056		
学生	林隽哲	江颢怡	刘彦凤		

编程实验

【实验内容】

（1）完成实验教程实例 3-2 的实验（考虑局域网、互联网两种实验环境），回答实验提出的问题及实验思考。（P103）。

（2）注意实验时简述设计思路。

（3）引起 UDP 丢包的可能原因是什么？

（1）实验设计思路：

本次实验主要用的是虚拟机（Linux）和本机（windows）来模拟两台机器的通信，是一种在局域网内的通信模式。其中本机作为服务器接受客户端发送过来的包，并且打开 wireshark 实时分析是否有丢包现象的产生。而虚拟机作为客户端。

关键代码分析：

1.客户端代码分析

① 需要创建 UDP 套接字来发送消息

```
// 创建一个 UDP 套接字
sockfd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);  // 使用 IPv4，数据报文套接字和 UDP 协议 这里的SOCK_DGRAM是标识UDP的
```

② 通过 snprintf 指定每个包的发送内容为 Package i

char buf[buffer_size]就是每次发送的包数据缓存的地方 而通过宏定义已经指定了缓冲区的大小为 30 字节，所以每次发送包的大小不超过 30 字节

```
char buf[BUFFER_SIZE];           // 定义一个字符数组作为发送缓冲区
snprintf(buf, sizeof(buf), "Package %d", i); // 生成发送内容，格式为 "Package X"
```

③ 通过调用 sendto 来发送数据包，my_socket 中设置了传递包的目标地址

```
// 发送数据包
ssize_t cc = sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr *)&my_socket, sizeof(my_socket));
// sendto 函数用于发送数据包到指定地址
```

④ 通过 sleep 机制，可以有效防止同一时间客户端发送数据包的频率太高，造成网络拥塞

```
usleep(1000); // 暂停 1 毫秒，以控制发送速度
```

⑤ 最后传输完数据包后即时关闭套接字，释放相关资源

```
}

close(sockfd);  // 关闭套接字，释放资源
```

2.服务器代码分析

服务器主要是接受来自客户端发送的数据包，并即时输出是否有包丢失，丢包的序列号。

① 因为服务器代码是运行在 windows 上，所以需要引入 ws2tcpip,winsock2 网络编程库。

```
#include <winsock2.h>
#include <ws2tcpip.h>

#pragma comment(lib, "ws2_32.lib")  // 链接 Winsock 库

#define PORT 10000  // 定义服务器监听的端口
#define BUFFER_SIZE 30  // 定义接收缓冲区的大小
#define TOTAL_PACKETS 500  // 假设客户端发送 100 个数据包
```

②为了方便统计丢包的序列号，引入了数组记录每个对应的包的接受情况，1 就是正常接收，0 就是发生丢包了。

```
int receivedPackets[TOTAL_PACKETS] = {0};  // 数组，用于记录接收到的包
int receivedCount = 0;  // 已接收的包计数
```

③ 同理，服务器也需要创建套接字用来接收消息

```
// 创建 UDP 套接字
sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

④ 且需要把套接字绑定到监听的端口上

```
// 绑定套接字
if (bind(sockfd, (const struct sockaddr*)&my_socket, sizeof(my_socket)) == SOCKET_ERROR) {
    printf("绑定失败. 错误代码: %d\n", WSAGetLastError());
    closesocket(sockfd);  // 关闭套接字
    WSACleanup();  // 清理 Winsock
    return 1;  // 返回错误
}
```

⑤ 定义一个循环来接受客户端连续发送的数据包，调用 recvfrom 函数来接收。同时做好是否有丢包的标记

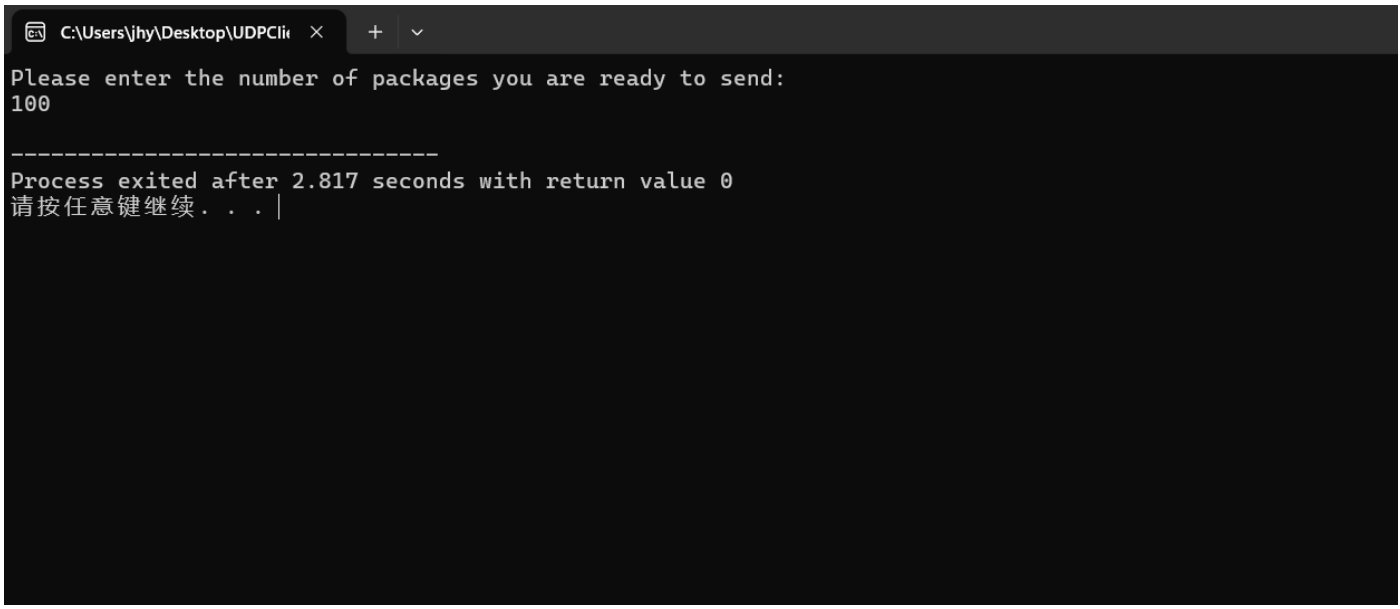
```
// 接收数据包
while (receivedCount < TOTAL_PACKETS) {  // 当接收到的包数量小于总包数时循环
    int cc = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, (struct sockaddr*)&from, &from_len);  // 接收数据包
    if (cc == SOCKET_ERROR) {  // 检查接收是否成功
        printf("recvfrom() 失败; 错误代码: %d\n", WSAGetLastError());
        break;  // 如果接收失败，退出循环
    }

    // 确保缓冲区以空字符结尾
    buffer[cc] = '\0';

    // 从接收到的消息中提取包号
    int packageNum;
    sscanf(buffer, "Package %d", &packageNum);  // 假设接收的格式为 "Package X"
    if (packageNum >= 1 && packageNum <= TOTAL_PACKETS) {  // 检查包号是否在有效范围内
        receivedPackets[packageNum - 1] = 1;  // 标记该包为已接收
        receivedCount++;  // 增加接收到的包计数
        printf("接收到的数据包: %s\n", buffer);  // 打印接收到的包
    } else {
        printf("接收到的包号无效: %s\n", buffer);  // 打印无效的包号
    }
}
```

⑥ 最后统计丢包数量输出并关闭套接字和清理 winsock

```
printf("接收到的包: ");
for (i = 0; i < TOTAL_PACKETS; i++) {  // 遍历所有包
    if (!receivedPackets[i]) {
        lostCount++;  // 增加丢失包计数
        printf("丢失的包: %d\n", i + 1);  // 打印丢失的包号
    }
}

printf("\n接收到的包总数: %d\n", receivedCount);  // 打印接收到的包总数
printf("丢失的包总数: %d\n", lostCount);  // 打印丢失的包总数

// 关闭套接字并清理资源
closesocket(sockfd);  // 关闭套接字
WSACleanup();  // 清理 Winsock
return 0;  // 返回成功
```

（2）实验结果展示

首先启动客户端发送数据包 可以看到最后 100 个数据包一共耗时 2.817ms



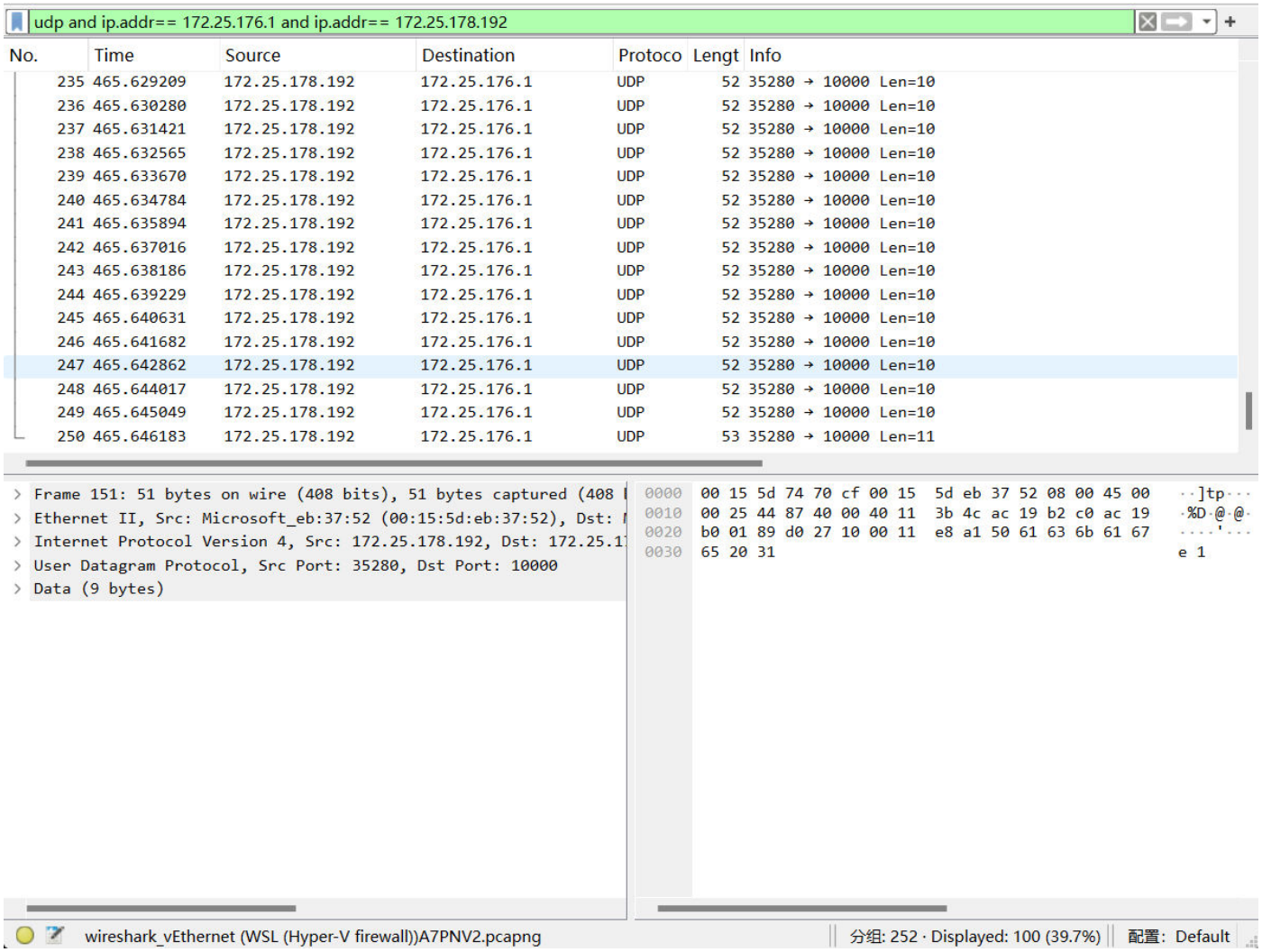
然后按要求在服务器统计 100 个数据包接受情况，最后丢包率是 0

```
接收到的数据包: Package 94
接收到的数据包: Package 95
接收到的数据包: Package 96
接收到的数据包: Package 97
接收到的数据包: Package 98
接收到的数据包: Package 99
接收到的数据包: Package 100

总结:
接收到的包: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
接收到的包总数: 100
丢失的包总数: 0

--------------------------------
Process exited after 16.5 seconds with return value 0
请按任意键继续. . .
```

通过 ipconfig 查看通信双方的 IP

```
以太网适配器 vEthernet (WSL (Hyper-V firewall)):

   连接特定的 DNS 后缀 . . . . . :
   本地链接 IPv6 地址. . . . . : fe80::4f76:42b5:c91b:2f16%39
   IPv4 地址 . . . . . : 172.25.176.1
   子网掩码 . . . . . : 255.255.240.0
   默认网关. . . . . :
```

然后在服务器上运行 wireshark 来抓包，限制 udp and ip.addr== 172.25.176.1 and ip.addr== 172.25.178.192 的过滤条件，最后下方显示一共有 100 个分组 也符合实际情况。



最后再次实验观察 500 个包的传送情况 最后也没有发生包的丢失 说明网络情况良好

```
接收到的数据包： Package 497
接收到的数据包： Package 498
接收到的数据包： Package 499
接收到的数据包： Package 500

总结：
接收到的包：
接收到的包总数： 500
丢失的包总数： 0

--------------------------------
Process exited after 11.86 seconds with return value 0
请按任意键继续. . .
```

（3）引起 UDP 丢包的原因是什么

引起 UDP 丢包的可能原因包括：

1. **网络拥塞**：
   - 网络带宽不足，导致数据包在路由器或交换机处被丢弃。
2. **缓冲区溢出**：
   - 接收方或中间设备的缓冲区容量不够，无法处理所有到达的数据包。
3. **网络设备故障**：
   - 路由器、交换机或网卡等设备故障导致数据包丢失。
4. **线路干扰**：
   - 无线网络中的电磁干扰、物理障碍等导致信号不稳定。
5. **高流量应用**：
   - 短时间内数据流量过大，超出网络或设备的处理能力。
6. **配置错误**：
   - 网络设备配置不当，导致部分数据包无法正确传输。

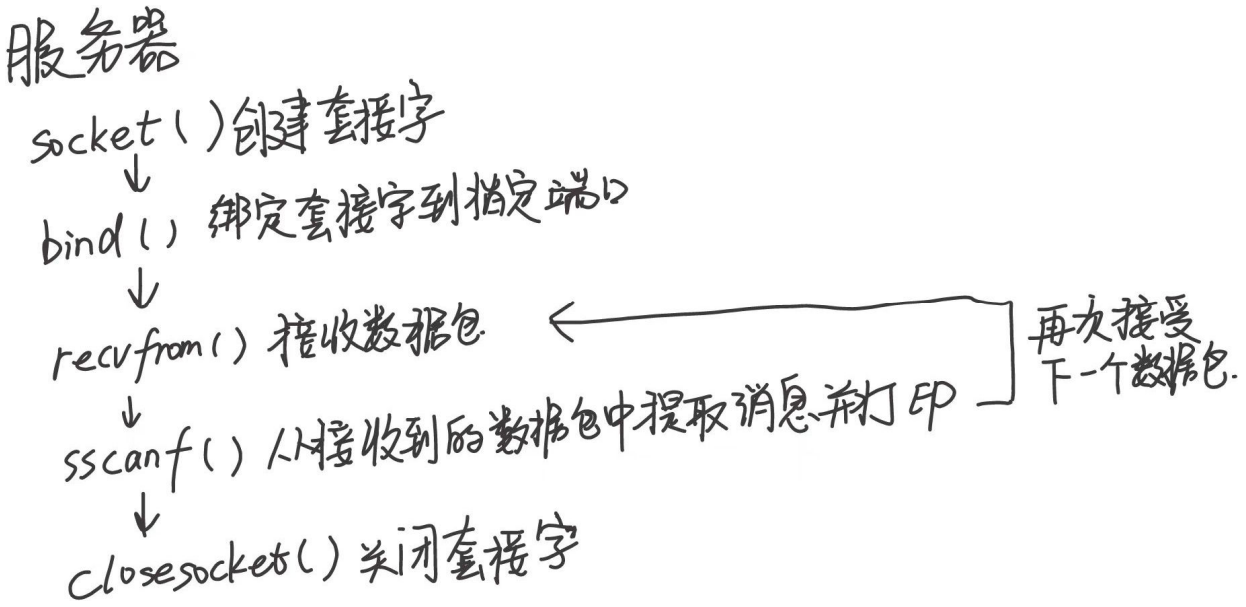
（4）实验思考部分

实验过程遇到的问题和解决方法:

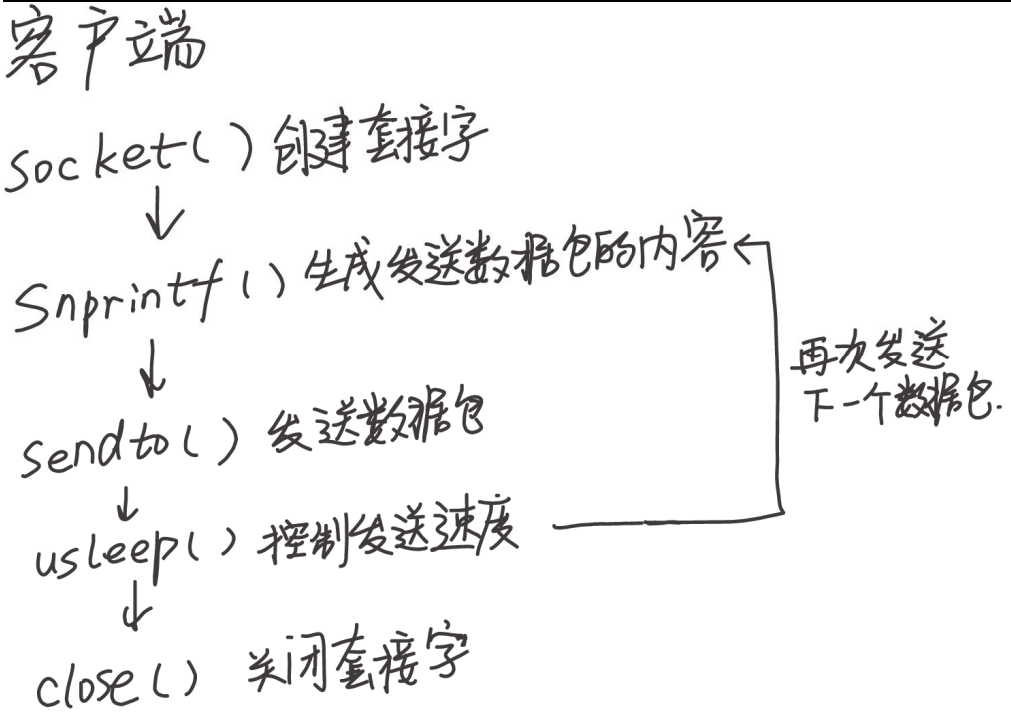
刚开始对局域网和互联网的概念不是很清楚 在询问老师之后 了解了局域网就是在同一个子网下可以直接通信的 而互联网就是与具有公网 ip 的主机取得通信，本次实验我们选择局域网的环境进行考虑

程序详细的流程图和程序关键函数的详细说明:

程序详细的流程图：



计算机网络实验报告



程序关键函数的详细说明：

sendto 和 recvfrom 是用于 UDP 套接字通信的函数，用于在不建立连接的情况下发送和接收数据。

1.ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
const struct sockaddr *dest_addr, socklen_t addrlen);

参数说明：

- sockfd: 套接字描述符，指定要通过其发送数据的套接字。
- buf: 指向要发送数据的缓冲区。
- len: 要发送的数据长度（字节数）。
- flags: 标志位，通常为 0。如果你有特殊需求，可以设置一些控制标志（如 MSG_DONTWAIT 等）。
- dest_addr: 指向 sockaddr 结构体的指针，指定目标主机的地址和端口（例如 sockaddr_in）。
- addrlen: dest_addr 结构的大小，通常使用 sizeof(struct sockaddr_in)。

返回值：

- 成功时返回实际发送的字节数。
- 失败时返回 -1，并设置 errno，可以通过 perror() 查看具体错误信息。

2. ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
struct sockaddr *src_addr, socklen_t *addrlen);

参数说明：

- sockfd: 套接字描述符，指定从哪个套接字接收数据。
- buf: 指向存放接收到的数据的缓冲区。
- len: 指定缓冲区的大小（字节数），即最多可以接收的字节数。
- flags: 标志位，通常为 0。如果有特殊需求，可以使用标志位（如 MSG_PEEK 等）。

- src_addr: 指向 sockaddr 结构体的指针，用于保存发送方的地址信息（如果不需要获取源地址信息，可以设置为 NULL）。
- addrlen: 输入输出参数，指向 socklen_t 变量，表示 src_addr 的大小。函数执行成功后，将返回实际的地址长度。

返回值：

- 成功时返回接收到的字节数。
- 失败时返回 -1，并设置 errno，可以通过 perror() 查看具体错误信息

使用 Socket API 开发通信程序中的客户端程序和服务器程序时，各需要哪些不同的函数?

客户端程序

1. **socket()**
   - 创建一个套接字。
2. **send() / sendto()**
   - 发送数据到服务器。
3. **recv() / recvfrom()**
   - 接收来自服务器的数据。
4. **close()**
   - 关闭套接字。

服务器程序

1. **socket()**
   - 创建一个套接字。
2. **bind()**
   - 将套接字绑定到特定的 IP 地址和端口。
3. **recv() / recvfrom()**
   - 接收来自客户端的数据。
4. **close()**
   - 关闭套接字。

解释 connect()、bind()等函数中 struct sockaddr*addr 参数各个部分的含义,并用具体的数据举例说明。

在使用 connect()、bind() 等函数时，struct sockaddr *addr 参数用于指定地址信息。具体来说，它通常是一个指向 struct sockaddr_in 的指针（用于 IPv4）。

struct sockaddr_in 结构

```
struct sockaddr_in {
    sa_family_t    sin_family;   // 地址族（协议族）
    in_port_t      sin_port;     // 端口号
    struct in_addr sin_addr;     // IP 地址
};
```

sin_family

- 指定地址族。对于 IPv4，通常设置为 AF_INET。

sin_port

- 指定端口号，必须使用 htons() 函数将主机字节序转换为网络字节序。

sin_addr

- 指定 IP 地址，通常使用 inet_addr() 或 inet_pton() 将字符串形式的地址转换为网络字节序。

具体的例子:
假设我们要绑定到 IP 地址 192.168.1.10 和端口 8080：
struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET; // IPv4
server_addr.sin_port = htons(8080); // 端口号，网络字节序
server_addr.sin_addr.s_addr = inet_addr("192.168.1.10"); // IP 地址，网络字节序

说明面向连接的客户端和面向非连接的客户端在建立 Socket 时有什么区别。

连接建立: 面向连接的客户端需要通过 connect() 方法与服务器建立连接，而面向非连接的客户端无需建立连接，可以直接发送数据。

可靠性: TCP 提供可靠的传输服务，UDP 不提供传输保证。

说明面向连接的客户端和面向非连接的客户端在收发数据时有什么区别。面向非连接的客户端又是如何判断数据发送结束的?

面向连接的客户端使用 send() 和 recv()，而面向非连接的客户端使用 sendto() 和 recvfrom()，并在每次发送和接收时指定目标地址。 UDP 本身不提供传输结束的标识。 通常由应用层协议定义结束条件，例如：

- 发送特定的结束标志。
- 计时器超时。
- 预定的数据包数量。

计算机网络实验报告

比较面向连接的通信和无连接通信,它们各有什么优点和缺点?适合在哪种场合下使用?

面向连接的通信（TCP）

优点

- **可靠性**：确保数据按顺序传输且无丢失。
- **流控制和拥塞控制**：自动调整数据流速率，避免网络拥塞。
- **错误检测和校正**：提供错误检测和重传机制。

缺点

- **开销大**：连接建立和维护需要额外的时间和资源。
- **速度较慢**：因为有流控和重传机制，速度可能受限。

适用场合

- **需要可靠传输的场合**：如网页浏览、文件传输和邮件发送。

无连接通信（UDP）

优点

- **效率高**：无需建立连接，数据传输速度更快。
- **开销小**：没有连接管理的开销。

缺点

- **不可靠**：无法保证数据顺序和完整性。
- **无流量控制**：容易导致网络拥塞。

适用场合

- **实时应用**：如视频会议、在线游戏和语音通话，强调速度而非可靠性。

实验过程中使用 Socket 时是工作在阻塞方式还是非阻塞方式?通过网络检索阐述这两种操作方式的不同。

该实验使用的是阻塞方式。

阻塞方式

特点

- 调用 I/O 操作（如 accept()、recv()、send()）时，如果操作无法立即完成，程序会等待直到操作完成。
- 简单易用，适合处理简单网络通信。

非阻塞方式

特点

- I/O 操作不会阻塞程序执行。如果操作无法立即完成，函数会立即返回错误代码（通常是 EWOULDBLOCK 或 EAGAIN）。
- 程序可以继续处理其他任务。

计算机网络实验报告

学号	学生	自评分
21312450	林隽哲	**100**
22365043	江颢怡	**100**
22302056	刘彦凤	**100**