

## 实验四：TCP/UDP通信程序设计

### 实验内容

#### 1. 完成一个TCP回射程序

- Client从标准输入读入一行文本，发送给Server。
- Server接收这行文本，再将其发送回Client。
- Client接收到这行回射的文本，将其显示在标准输出上。

程序代码：

tcp\_server.c

```
#include "net.h"

int main() {

    /* 建立服务端套接字 */
    int server_sockfd = Socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    int enable = 1;
    Setsockopt(server_sockfd, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int));

    /* 绑定端口 */
    struct sockaddr_in server_addr = {0};
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(TCP_SERVER_PORT);
    if (inet_pton(AF_INET, TCP_SERVER_ADDRESS, &server_addr.sin_addr.s_addr) !=
1) {
        perror_and_exit("inet_pton");
    }

    int opt = 1;
    // server_sockfd为需要端口复用的套接字
    setsockopt(server_sockfd, SOL_SOCKET, SO_REUSEADDR, (const void *)&opt,
sizeof(opt));

    Bind(server_sockfd, (struct sockaddr *)&server_addr, sizeof(struct
sockaddr));

    /* 监听端口 */
    Listen(server_sockfd, CONNECTION_NUMBER);
    printf("The server is ready to receive. \n\n");
    /* 建立tcp连接 */
    struct sockaddr_in client_addr;
    unsigned int client_addr_len = sizeof(struct sockaddr_in);
    int client_sockfd = Accept(server_sockfd, (struct sockaddr *)&client_addr,
&client_addr_len);
    printf("Accept client %s:%d\n\n", inet_ntoa(client_addr.sin_addr),
ntohs(client_addr.sin_port));

    /* 接收数据 */
    char buf[TCP_BUF_LENGTH];
    size_t pkt_len = Recv(client_sockfd, buf, TCP_BUF_LENGTH, 0);
```

```

    if (pkt_len > 0) {
        buf[pkt_len] = '\0';
        printf("Message received from client: %s[%zu bytes]\n", buf, pkt_len);
        send(client_sockfd, buf, strlen(buf), 0);
        printf("%ld\n", strlen(buf));
        printf("Send successfully to the client.\n\n");
    } else {
        printf("Connection closed\n");
    }

    /* 关闭套接字 */
    close(client_sockfd);
    close(server_sockfd);

    return 0;
}

```

tcp\_client.c

```

#include "net.h"

int main() {

    /* 建立套接字 */
    int sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    /* 建立tcp连接 */
    struct sockaddr_in server_addr = {0};
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(TCP_SERVER_PORT);
    if (inet_pton(AF_INET, TCP_SERVER_ADDRESS, &server_addr.sin_addr.s_addr) !=
1) {
        perror_and_exit("inet_pton");
    }

    connect(sockfd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr));
    printf("Successfully connected with the server!\n\n");
    /* 发送数据 */
    char msg[] = "default";
    printf("Please enter a sentence!\n");
    scanf("%s", msg);
    // printf("%s",msg);
    printf("Send message to server: %s[%zu bytes]\n", msg, strlen(msg));
    send(sockfd, msg, strlen(msg), 0);
    printf("Successfully send data!\n\n");
    /*接收数据*/
    char buf[TCP_BUF_LENGTH];
    size_t pkt_len = recv(sockfd, buf, TCP_BUF_LENGTH, 0);
    printf("Message received from server: %s[%zu bytes]\n", buf, pkt_len);
    /* 关闭套接字 */
    close(sockfd);

    return 0;
}

```

实验结果:

```
ustc@ustc-lug-linux101:~/Codefield$ ./tcp_server
The server in ready to receive.

Accept client 127.0.0.1:53294

Message received from client: tsvdbje[7 bytes]
Send sucessfully to the client.
```

```
ustc@ustc-lug-linux101:~/Codefield$ ./tcp_client
Sucessfully connected with the server!

Please enter a sentence!
tsvdbje
Send message to server: tsvdbje[7 bytes]
Sucessfully send data!

Message received from server: tsvdbje[7 bytes]
```

## 2. 完成一个UDP通信程序

- Client 创建 10 个 socket, 每个 socket 发送 1 个数据包给 Server, 内容为任意字符串。
- Server 在每次收到数据包时, 将发送端的 IP 地址和端口号显示在标准输出上。
- 要求 Client 使用 connect() 和 send() 实现。

程序代码:

udp\_server.c

```
#include "net.h"
#include <string.h>
# include <arpa/inet.h>
int main() {
    /* 建立套接字 */
    int sockfd = Socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    int enable = 1;
    Setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int));

    /* 绑定端口 */
    struct sockaddr_in server_addr = {0};
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(UDP_SERVER_PORT);
    if (inet_pton(AF_INET, UDP_SERVER_ADDRESS, &server_addr.sin_addr.s_addr) !=
1) {
        perror_and_exit("inet_pton");
    }

    Bind(sockfd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr));

    /* 接收数据 */
    struct sockaddr_in client_addr;
    int client_addr_len = sizeof(client_addr);

    char buf[UDP_BUF_LENGTH];
    char ip[20];
    int i=1;
    printf("Start receiving message: \n");
    while(1){
        // printf("%d", i);
```

```

        // i++;
        size_t pkt_len = Recvfrom(sockfd, buf, UDP_BUF_LENGTH, 0, (struct
sockaddr *)&client_addr, &client_addr_len);
        if(pkt_len != -1){
            buf[pkt_len] = '\0';
            printf("Message received: %s[%zu bytes]\t", buf, pkt_len);
            inet_ntop(AF_INET, &client_addr.sin_addr, ip, sizeof(ip));
            printf("from client IP:%s port: %d\n", ip,
ntohs(client_addr.sin_port));
        }
    }
    /* 关闭套接字 */
    close(sockfd);

    return 0;
}

```

udp\_client.c

```

#include "net.h"

int main() {
    int i;
    for(i=1; i<=10; i++)
    {
        /* 建立套接字 */
        int sockfd = Socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

        /* 发送数据 */
        struct sockaddr_in server_addr = {0};
        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(UDP_SERVER_PORT);
        if (inet_pton(AF_INET, UDP_SERVER_ADDRESS, &server_addr.sin_addr.s_addr)
!= 1) {
            perror_and_exit("inet_pton");
        }

        Connect(sockfd, (struct sockaddr *)&server_addr, sizeof(struct
sockaddr));
        char msg[] = "datagram_dhx" ;
        printf("Send message to server: %s[%zu bytes]\t(socket %d)\n", msg,
strlen(msg), i);
        Send(sockfd, msg, strlen(msg), 0);

        /* 关闭套接字 */
        close(sockfd);
    }
    return 0;
}

```

实验结果:

```
ustc@ustc-lug-linux101:~/Codefield$ ./udp_client
Send message to server: datagram_dxh[12 bytes] (socket 1)
Send message to server: datagram_dxh[12 bytes] (socket 2)
Send message to server: datagram_dxh[12 bytes] (socket 3)
Send message to server: datagram_dxh[12 bytes] (socket 4)
Send message to server: datagram_dxh[12 bytes] (socket 5)
Send message to server: datagram_dxh[12 bytes] (socket 6)
Send message to server: datagram_dxh[12 bytes] (socket 7)
Send message to server: datagram_dxh[12 bytes] (socket 8)
Send message to server: datagram_dxh[12 bytes] (socket 9)
Send message to server: datagram_dxh[12 bytes] (socket 10)
```

```
ustc@ustc-lug-linux101:~/Codefield$ ./udp_server
Start receiving message:
Message received: datagram_dxh[12 bytes] from client IP:127.0.0.1 port: 35938
Message received: datagram_dxh[12 bytes] from client IP:127.0.0.1 port: 34722
Message received: datagram_dxh[12 bytes] from client IP:127.0.0.1 port: 40399
Message received: datagram_dxh[12 bytes] from client IP:127.0.0.1 port: 40774
Message received: datagram_dxh[12 bytes] from client IP:127.0.0.1 port: 45559
Message received: datagram_dxh[12 bytes] from client IP:127.0.0.1 port: 55027
Message received: datagram_dxh[12 bytes] from client IP:127.0.0.1 port: 49155
Message received: datagram_dxh[12 bytes] from client IP:127.0.0.1 port: 34880
Message received: datagram_dxh[12 bytes] from client IP:127.0.0.1 port: 37260
Message received: datagram_dxh[12 bytes] from client IP:127.0.0.1 port: 49705
```

### 3. 可选，完成一个TCP通信程序

- 利用 `fork()`，实现两终端实时通信。

程序代码：

tcp\_rt\_server.c

```
#include "net.h"
void handler(int sig)
{
    printf("recv a sig=%d\n", sig);
    exit(EXIT_SUCCESS);
}
int main()
{
    /* 建立服务端套接字 */
    int server_sockfd = Socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    int enable = 1;
    setsockopt(server_sockfd, SOL_SOCKET, SO_REUSEADDR, &enable,
        sizeof(int));
    /* 绑定端口 */
    struct sockaddr_in server_addr = {0};
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(TCP_SERVER_PORT);
    if (inet_pton(AF_INET, TCP_SERVER_ADDRESS,
        &server_addr.sin_addr.s_addr) != 1)
    {
        perror_and_exit("inet_pton");
    }
    Bind(server_sockfd, (struct sockaddr *)&server_addr, sizeof(struct
        sockaddr));
    /* 监听端口 */
    Listen(server_sockfd, CONNECTION_NUMBER);
    /* 建立 tcp 连接 */
```

```

struct sockaddr_in client_addr;
unsigned int client_addr_len = sizeof(struct sockaddr_in);
int client_sockfd = Accept(server_sockfd, (struct sockaddr
*)&client_addr, &client_addr_len);
printf("Accept client %s:%d\n", inet_ntoa(client_addr.sin_addr),
ntohs(client_addr.sin_port));
pid_t pid;
pid = fork();
if (pid == -1)
printf("fork error\n");
if (pid == 0)
{
//child process
signal(SIGUSR1, handler);
/* send data */
char sendbuf[TCP_BUF_LENGTH] = {0};
while (fgets(sendbuf, sizeof(sendbuf), stdin) != NULL)
{
Send(client_sockfd, sendbuf, strlen(sendbuf), 0);
memset(sendbuf, 0, sizeof(sendbuf));
}
exit(EXIT_SUCCESS);
}
else
{
//parent process
/* 接收数据 */
char recvbuf[TCP_BUF_LENGTH];
while (1)
{
memset(recvbuf, 0, sizeof(recvbuf));
size_t pkt_len = Recv(client_sockfd, recvbuf,
TCP_BUF_LENGTH, 0);
if (pkt_len > 0)
{
recvbuf[pkt_len] = '\0';
printf("Message received: %s[%zu bytes]\n", recvbuf,
pkt_len);
}
else
{
printf("Connection closed\n");
break;
}
}
kill(pid, SIGUSR1);
exit(EXIT_SUCCESS);
}
close(client_sockfd);
close(server_sockfd);
return 0;
}#include "net.h"

void handler(int sig)
{
printf("recv a sig=%d\n", sig);
exit(EXIT_SUCCESS);
}

```

```

int main()
{
    /* 建立服务端套接字 */
    int server_sockfd = Socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    int enable = 1;
    Setsockopt(server_sockfd, SOL_SOCKET, SO_REUSEADDR, &enable,
        sizeof(int));
    /* 绑定端口 */
    struct sockaddr_in server_addr = {0};
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(TCP_SERVER_PORT);
    if (inet_pton(AF_INET, TCP_SERVER_ADDRESS, &server_addr.sin_addr.s_addr) !=
1)
    {
        perror_and_exit("inet_pton");
    }
    Bind(server_sockfd, (struct sockaddr *)&server_addr, sizeof(struct
sockaddr));
    /* 监听端口 */
    Listen(server_sockfd, CONNECTION_NUMBER);
    /* 建立 tcp 连接 */
    struct sockaddr_in client_addr;
    unsigned int client_addr_len = sizeof(struct sockaddr_in);
    int client_sockfd = Accept(server_sockfd, (struct sockaddr*)&client_addr,
&client_addr_len);
    printf("Accept client %s:%d\n", inet_ntoa(client_addr.sin_addr),
ntohs(client_addr.sin_port));
    pid_t pid;
    pid = fork();
    if (pid == -1)
        printf("fork error\n");
    if (pid == 0)
    {
        //child process
        signal(SIGUSR1, handler);
        /* send data */
        char sendbuf[TCP_BUF_LENGTH] = {0};
        while (fgets(sendbuf, sizeof(sendbuf), stdin) != NULL)
        {
            Send(client_sockfd, sendbuf, strlen(sendbuf), 0);
            memset(sendbuf, 0, sizeof(sendbuf));
        }
        exit(EXIT_SUCCESS);
    }
    else
    {
        //parent process
        /* 接收数据 */
        char recvbuf[TCP_BUF_LENGTH];
        while (1)
        {
            memset(recvbuf, 0, sizeof(recvbuf));
            size_t pkt_len = Recv(client_sockfd, recvbuf,
TCP_BUF_LENGTH, 0);
            if (pkt_len > 0)
            {
                recvbuf[pkt_len] = '\0';

```

```

        printf("Message received: %s[%zu bytes]\n", recvbuf,
            pkt_len);
    }
    else
    {
        printf("Connection closed\n");
        break;
    }
}
kill(pid, SIGUSR1);
exit(EXIT_SUCCESS);
}
close(client_sockfd);
close(server_sockfd);
return 0;
}

```

tcp\_rt\_client.c

```

#include "net.h"

void handler(int sig)
{
    printf("recv a sig=%d\n", sig);
    exit(EXIT_SUCCESS);
}

int main()
{
    /* 建立套接字 */
    int sockfd = Socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    /* 建立 tcp 连接 */
    struct sockaddr_in server_addr = {0};
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(TCP_SERVER_PORT);
    if (inet_pton(AF_INET, TCP_SERVER_ADDRESS, &server_addr.sin_addr.s_addr) !=
1)
    {
        perror_and_exit("inet_pton");
    }
    Connect(sockfd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr));
    pid_t pid;
    pid = fork();
    if (pid == -1)
        printf("fork error\n");
    if (pid == 0)
    {
        //child process
        signal(SIGUSR1, handler);
        /* 发送数据 */
        char sendbuf[TCP_BUF_LENGTH] = {0};
        while (fgets(sendbuf, sizeof(sendbuf), stdin) != NULL)
        {
            Send(sockfd, sendbuf, strlen(sendbuf), 0);
            memset(sendbuf, 0, sizeof(sendbuf));
        }
        exit(EXIT_SUCCESS);
    }
}

```

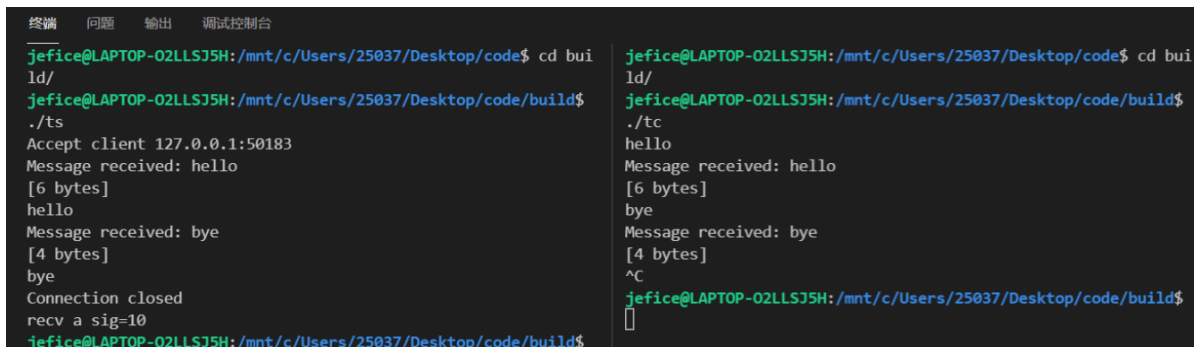


```

    }
    else
    {
        //parent process
        /* receive data */
        char recvbuf[TCP_BUF_LENGTH];
        while (1)
        {
            size_t pkt_len = Recv(sockfd, recvbuf, TCP_BUF_LENGTH, 0);
            if (pkt_len > 0)
            {
                recvbuf[pkt_len] = '\0';
                printf("Message received: %s[%zu bytes]\n", recvbuf, pkt_len);
            }
            else
            {
                printf("Connection closed\n");
                break;
            }
        }
        kill(pid, SIGUSR1);
        exit(EXIT_SUCCESS);
    }
    /* 关闭套接字 */
    close(sockfd);
    return 0;
}

```

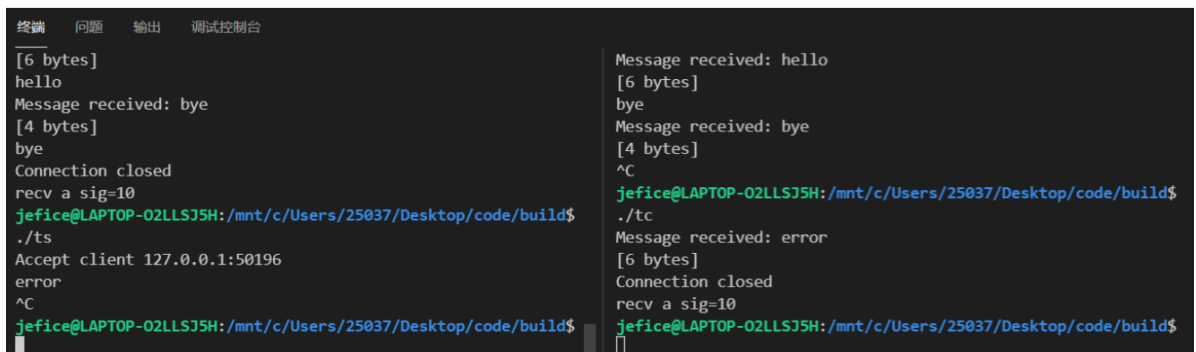
实验结果：



```

终端 问题 输出 调试控制台
jefice@LAPTOP-02LLS35H:/mnt/c/Users/25037/Desktop/code$ cd bui
ld/
jefice@LAPTOP-02LLS35H:/mnt/c/Users/25037/Desktop/code/build$
./ts
Accept client 127.0.0.1:50183
Message received: hello
[6 bytes]
hello
Message received: bye
[4 bytes]
bye
Connection closed
recv a sig=10
jefice@LAPTOP-02LLS35H:/mnt/c/Users/25037/Desktop/code/build$
jefice@LAPTOP-02LLS35H:/mnt/c/Users/25037/Desktop/code$ cd bui
ld/
jefice@LAPTOP-02LLS35H:/mnt/c/Users/25037/Desktop/code/build$
./tc
hello
Message received: hello
[6 bytes]
bye
Message received: bye
[4 bytes]
^C
jefice@LAPTOP-02LLS35H:/mnt/c/Users/25037/Desktop/code/build$

```



```

终端 问题 输出 调试控制台
[6 bytes]
hello
Message received: bye
[4 bytes]
bye
Connection closed
recv a sig=10
jefice@LAPTOP-02LLS35H:/mnt/c/Users/25037/Desktop/code/build$
./ts
Accept client 127.0.0.1:50196
error
^C
jefice@LAPTOP-02LLS35H:/mnt/c/Users/25037/Desktop/code/build$
Message received: hello
[6 bytes]
bye
Message received: bye
[4 bytes]
^C
jefice@LAPTOP-02LLS35H:/mnt/c/Users/25037/Desktop/code/build$
./tc
Message received: error
[6 bytes]
Connection closed
recv a sig=10
jefice@LAPTOP-02LLS35H:/mnt/c/Users/25037/Desktop/code/build$

```

注：此处使用的是 wsl，即 windows 下子系统 linux，界面与之前不一样，但终端仍是 linux 系统的。

## 思考题

1. 解释 `struct sockaddr_in` 结构体各个部分的含义，并用具体的数据举例说明。

(1)描述 socket 地址的结构体 `sockaddr_in`:

- `sin_family`: 协议族  
常见的有 `AF_INET`: 只用于 IPv4; `AF_INET6`: IPv6, 有时向下兼容 IPv4; `AF_UNIX`: 用于本地套接字 (使用特殊的文件系统节点)
- `in_port_t sin_port`: 端口号 (使用网络字节顺序, 16 位) 理论上端口号的取值范围为 0-65536, 但 0-1023 的端口一般由系统分配给特定的服务程序 (如 80 号端口给予 Web 服务, FTP 服务的端口号为 21)
- `struct in_addr sin_addr`: ip 地址 (32 位地址) 其实是一个整数, 需要用 `inet_addr()` 进行转换为 IP 地址 (字符串)
- `sin_zero` 是为了让 `sockaddr` 与 `sockaddr_in` 两个数据结构保持大小相同而保留的空字节, 一般使用 `memset()` 函数填充为 0, 或者初始定义时便定义为 0

(2)举例说明:

- `sin_family=AF_INET`, 用于 IPv4
  - `sin_port=htons(TCP_SERVER_PORT)`, 其中 `TCP_SERVER_PORT` 被定义为 8001, `htons` 函数将这个无符号短整数从主机字节以网络字节顺序
  - `inet_pton` 函数将 IPv4 地址从文本转换为二进制形式, 其中 `AF_INET` 为协议, `TCP_SERVER_ADDRESS` 为点分十进制的本机地址 (字符串), 转换成功返回 1, 结果 `sin_addr` 存入 IP 地址
  - 初始定义时便定义 `sin_zero` 为 0;
- 

2. 说明面向连接的客户端和非连接的客户端在建立 socket 时有什么区别。

(1)TCP编程为socket函数创建一个socket用于TCP通讯, 函数参数通常填为 `SOCK_STREAM`。即

`socket(PF_INET, SOCK_STREAM, 0)`, 这表示建立一个 socket 用于流式网络通讯。

`SOCK_STREAM` 这种的特点是面向连接的, 即每次收发数据之前必须通过 `connect` 建立连接, 也是双向的, 即任何一方都可以收发数据, 协议本身提供了一些保障机制保证它是可靠的、有序的, 即每个包按照发送的顺序到达接收方。

(2)UDP 编程使用 `SOCK_DGRAM` 这种为 User Datagram Protocol 协议的网络通讯, 它是无连接的, 不可靠的, 因为通讯双方发送数据后不知道对方是否已经收到数据, 是否正常收到数据。任何一方建立一个 socket 以后就可以用 `sendto` 发送数据, 也可以用 `recvfrom` 接收数据。

---

3. 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。

(1)TCP 客户端中, 设置要连接的对方的 IP 地址和端口等属性, 之后连接服务器, 用函数

`connect()`, 然后收发数据, 用函数 `send()` 和 `recv()`。

(2)UDP 客户端中, 设置对方的 IP 地址和端口等属性, 然后收发数据, 用函数 `sendto()` 和

`recvfrom()`。

---

4. 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?

(1)面向连接网络用户的通信总要经过建立连接、信息传途、释放连接三个阶段；面向连接网络中的每一个节点为每一个呼叫选路,节点中需要有维持连接的状态表。因此传输可靠性好，但是协议复杂，效率不高。

(2)无连接网络不为用户的通信过程建立和拆除连接。无连接网络中的每一个节点为每一个传递的信息选路，节点中不需要维持连接的状态表。因此可靠性不是很好，但是由于省去了很多复杂的协议过程，因此通信过程较为简单，效率较高。

(3)用户信息较长时，采用面向连接的通信方式的效率高；反之，使用无连接的方式要好一些。

---

5. 实验过程中使用 socket 的时候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。

阻塞方式。阻塞模式和非阻塞模式的主要区别在于无请求来到时，阻塞模式会一直停在接收函数即 accept 函数，直到有请求到来才会继续向下进行处理。而非阻塞模式下，运行接收函数，如果有请求，则会接收请求，如果无请求，会返回一个负值，并继续向下运行。