南开大学

Nankai University

南　　开　　大　　学

计　算　机　学　院

计算机网络实验报告

第一次实验

学号：1911437

姓名：刘浩通

年级：2019 级

专业：计算机科学与技术

2021 年 10 月 22 日

目录

一、 实验说明 **1**

二、 协议设计 **1**

三、 程序设计 **2**

（一） 重要函数介绍 2

（二） 服务器端程序设计 5

（三） 客户端程序设计 5

四、 程序实现 **5**

（一） 服务器端程序实现 5

（二） 客户端程序的实现 9

五、 程序运行 **11**

一、 实验说明

利用 Socket，设计和编写一个聊天程序。基本要求如下：

1. 设计一个两人聊天协议，要求聊天信息带有时间标签。
2. 对聊天程序进行设计。
3. 在 Windows 系统下，利用 C/C++ 中的流式 Socket 对设计的程序进行实现。程序界面可以采用命令行方式，但需要给出使用方法。
4. 对实现的程序进行测试。
5. 撰写实验报告，并将实验报告和源码提交至计算机学院网站。

二、 协议设计

在本次实验，使用的是 TCP 协议来实现客户端和服务器端的连接。

TCP 协议全称是传输控制协议是一种面向连接的、可靠的、基于字节流的传输层通信协议，由 IETF 的 RFC 793 定义。TCP 是面向连接的、可靠的流协议。

建立一个 TCP 连接的过程为（三次握手的过程）：

1. 第一次握手
   客户端向服务端发送连接请求报文段。该报文段中包含自身的数据通讯初始序号。请求发送后，客户端便进入 SYN-SENT 状态。
2. 第二次握手
   服务端收到连接请求报文段后，如果同意连接，则会发送一个应答，该应答中也会包含自身的数据通讯初始序号，发送完成后便进入 SYN-RECEIVED 状态。
3. 第三次握手当客户端收到连接同意的应答后，还要向服务端发送一个确认报文。客户端发完这个报文段后便进入 ESTABLISHED 状态，服务端收到这个应答后也进入 ESTABLISHED 状态，此时连接建立成功。

因为是实现的两人对话协议，是服务器端和用户端进行聊天：

客户端的内容：
功能：

1. 数据发送
2. 数据接收

代码上：

1. 使用 socket 嵌套字，TCP 协议连接，与服务器连接
2. 使用多线程：用于接收和发送消息

服务器的内容：
功能:

1. 数据接收

2. 数据发送

代码实现：

1. 建立一个 serversocket 嵌套字，TCP 协议的。
2. 建立 clientsocket，用于和客户端连接
3. 多线程，用于接收和发送消息

三、 程序设计

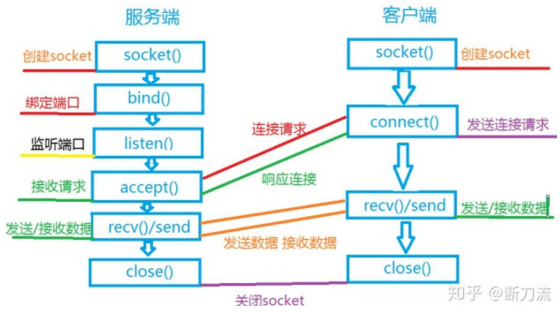

图 1: C/S 架构

(一) 重要函数介绍

1. Socket 函数

描述：描述：根据指定的地址族、数据类型，创建一个特定的套接字。

```
SOCKET WSAAPI socket(
        int af,
        int type,
        int protocol
);
int af
指定地址族规范，当前支持的值为 AF_INET和AF_INET6，它们表示IPv4和IPv6互联网地址族格式。
int type
指定Socket类型，分别有 SOCKSTREAM流式套接字、SOCKDGRAM数据报套接字 、SOCK_RAW原始套接字。
int protocol
地址家族相关协议 IPPROTOTCP、IPPROTOUDP 和 IPPROTO_IP，如果参数为0，系统将会自动分配。
返回值
创建成功返回套接字句柄，失败则返回 INVALID_SOCKET
```

2. bind 函数

描述：描述：将本地地址与 Socket 套接字绑定起来。

```
int WSAAPI bind(
        SOCKET         s,
        const sockaddr *name,
        int            namelen
);
SOCKET s
需绑定的Socket
const sockaddr *name
绑定套接字的本地地址的sockaddr结构体指针。
int namelen
指向的sockaddr结构的长度(以字节为单位)。
返回值
成功返回0，否则返回 SOCKET_ERROR
```

3. listen 函数

描述：将套接字置于侦听传入连接的状态。

```
int WSAAPI listen(
        SOCKET s,
        int    backlog
);
SOCKET s
需要进行监听的 socket句柄
int backlog
连接队列的最大长度。如果设置为SOMAXCONN，则将程序把积压设置为最大合理值。如果
    设置为SOMAXCONN_HINT(N)(其中N是一个数字)，积压值将为N，调整为在范围内
    (200,65535)。请注意，
返回值
成功返回0，否则返回 SOCKET_ERROR
```

4. accept 函数

描述：允许在监听套接字上尝试进入连接。

```
SOCKET WSAAPI accept(
        SOCKET   s,
        sockaddr *addr,
        int      *addrlen
);
SOCKET s
一个描述符，用于标识使用listen函数处于监听状态的套接字。该连接实际上是使用
    accept返回的套接字建立的
sockaddr *addr,
用于接收客户端的地址的sockaddr结构体指针。
int *addrlen
指向的sockaddr结构的长度(以字节为单位)。
返回值
成功返回 与客户端连接的新socket句柄，否则返回INVALID_SOCKET
```

5. send 函数

描述：在连接的套接字上发送数据。

```
int WSAAPI send(
        SOCKET     s,
        const char *buf,
        int        len,
        int        flags
);
SOCKET s
指定接收数据的socket句柄
const char *buf
指定要发送的数据
int len
发送数据的长度
int flags
一般置为0
返回值
成功返回发送的字节数，否则返回 SOCKET_ERROR
```

6. recv 函数

描述：从已连接的套接字或绑定的无连接套接字接收数据。

```
int WSAAPI recv(
        SOCKET s,
        char   *buf,
        int    len,
        int    flags
);
SOCKET s
指定接收那个socket的消息
char *buf
用于接收数据的缓冲区
int len
指定缓冲区的长度
int flags
一般置为0
返回值
成功 返回接收的字节数，如果连接已正常关闭则返回0
```

7. connect 函数

描述：建立到指定套接字的连接。

```
int WSAAPI connect(
SOCKET         s,
const sockaddr *name,
int            namelen
);
参数
```

```
7  SOCKET s
8  需要连接的 socket句柄
9  const sockaddr *name
10 建立连接的sockaddr结构体指针。
11 int namelen
12 指向的sockaddr结构的长度(以字节为单位)。
13 函数返回
14 成功返回零。否则，返回SOCKET_ERROR
```

(二) 服务器端程序设计

首先根据 C/S 结构我们可以得知，在服务器端与客户端连接之前，需要进行创建 socket、绑定端口、监听窗口初始化三个步骤，才能够接受客户端发来的连接请求。

在与客户端建立连接时，创建一个 socket 实例 client，后续的接收与发送聊天信息，都需要 client 参与。

在与客户端建立好连接后，这时候服务器端的主线程就会创建两个子线程来分别进行接收信息和发送信息的操作。

这样做的好处是相较于单线程，不必一直处于某种状态 (监听信息状态或者发送信息状态，亦或者是发送一条信息后等待回复才能继续发送)，对于使用的一方更加便捷。

在主线程创建完两个线程后，主线程便会进入沉睡状态，以免主线程结束杀死两个子线程。

在发送消息的子线程，发送消息时会加上发送消息时的时间标签，已告知客户端。

同理，接收信息时会将消息内容中的时间标签单独的提取出来显示，在打印消息内容。

两个线程并不会干扰，以便让聊天流畅的进行。

然后，还设置了一个标志 is_Quit 判断客户端是否关闭。

(三) 客户端程序设计

相较于服务器端的实现，客户端的实现的功能雷同，但是更加简单。

首先，客户端建立 socket 以便可以与服务器端建立连接。

当 socket 实例建立成功后，便会调用 connect 函数向服务器端发送请求，请求连接。

当服务器端与客户端的连接建立起来了后，客户端主线程也会建立两个线程：发送线程和接收线程，原理与服务器端多线程一样。

在主线程创建完两个线程后，主线程便会进入沉睡状态，以免主线程结束杀死两个子线程。

在发送消息的子线程，发送消息时会加上发送消息时的时间标签，已告知客户端。

同理，接收信息时会将消息内容中的时间标签单独的提取出来显示，在打印消息内容。

两个线程并不会干扰，以便让聊天流畅的进行。

然后，还设置了一个标志 is_Quit 判断客户端是否关闭。

四、 程序实现

(一) 服务器端程序实现

先看主线程的实现 (也就是 main 函数实现)：

首先是初始化网络环境，创建服务器 socket：serversocket

```
WSADATA wsadata;
if (0 != WSAStartup(MAKEWORD(2, 2), &wsadata)) {
        cout << "嵌套字初始化失败" << endl;
        return 0;
}
SOCKET server = INVALID_SOCKET;
SOCKET client = INVALID_SOCKET;

if ((server = socket(AF_INET, SOCK_STREAM, 0)) == SOCKET_ERROR){
        cout << "服务器套接字创建失败! " << endl;
        return 0;
}
```

接着服务器 socket 创建完成了后，进行绑定端口：

首先是要绑定 IP 地址：

```
struct sockaddr_in serverAddr;
memset(&serverAddr, 0, sizeof(sockaddr_in));
serverAddr.sin_family = AF_INET;//指定地址族为IPv4
serverAddr.sin_addr.S_un.S_addr = htonl(INADDR_ANY); //设置绑定的IP
serverAddr.sin_port = htons(10086);//设置监听端口,这里端口号是10086, 客户端连接需要使用此端口号
```

在调用 bind 函数进行绑定端口：

```
if (SOCKET_ERROR == bind(server, (SOCKADDR*)&serverAddr, sizeof(serverAddr))) {
        cout << "嵌套字绑定失败" << endl;
        return 0;
}
else{
        cout << "绑定成功" << endl;
}
```

接着调用 listen 来初始化监听端口：

```
if (SOCKET_ERROR == listen(server, SOMAXCONN)) {
        cout << "监听失败" << endl;
        return 0;
}
else{
        cout << "监听成功" << endl;
}
```

完成上面准备的三个步骤后，接下来就是接受客户连接请求：

```
 sockaddr_in addrClient = { 0 };//创建结构体 用于获取客户端 socketaddr 结构体
int addrsize = sizeof(addrClient);//获取结构体大小
cout << "正在等待客户端连接" << endl;
client = accept(server, (SOCKADDR*)&addrClient, &addrsize);//接受客户端的连接
if (INVALID_SOCKET == client) {
```

```
        cout << "与客户端连接失败" << endl;
        return 0;
}
else{
        cout << "与客户端连接成功" << endl;
}
```

接下来就是创建两个子线程来实现接收和发送消息的操作了，随后主线程会陷入睡眠：

```
_beginthread(Receive, 0, &client);//创建接收线程
_beginthread(Send, 0, &client);//创建发送线程

while (!is_Quit) {//若准备推出，留一定时间关闭线程
        Sleep(1000);
}

closesocket(server);//关闭连接
closesocket(client);

WSACleanup();
cout << "服务器停止了" << endl;
return 0;
```

下面来看服务器发送功能的实现：
为了防止线程执行一次就结束，使用 while(1){} 来防止结束：

```
void Send(void* param)
{
        while (!is_Quit)
        {
                发送的操作
        }
}
```

接下来，因为发送消息要带有时间标签，所以要获取当前的时间：

```
time_t  t;
char buf[20];
memset(buf, 0, sizeof(buf));
struct tm* tmp;
t = time(NULL);
tmp = localtime(&t);
strftime(buf, sizeof(buf), "%Y-%m-%d %H:%M:%S", tmp);
```

将获取的时间保存在了 buf 数组中去，当发送消息时，会将 buf 数组通过 strcpy 加入到发送数组中去。

然后就是获取发送的消息，并且与时间标签融合：
这里还判断，若是“quit”关闭服务器，则将 is_Quit 置为，重新进入 while 判断。

```
char sendbuf[MAXBYTE] = {0};
```

```
cout << "当前时间: " << buf << " 你要发送的内容是: ";
strcpy(sendbuf, buf);
string _tmp;
cin >> _tmp;
if (_tmp == "quit") {
        is_Quit = true;
        continue;
}
strcat(sendbuf, _tmp.c_str());
```

现在要发送给客户端的消息已经准备好了，存储在 sendbuf 数组中，前 19 个元素是时间标签，后面是要发送的内容。

然后调用 send 函数，将信息发送给客户端：

```
if (SOCKET_ERROR == send(revClientSocket, sendbuf, strlen(sendbuf), 0)) {
        is_Quit=true;
        cout << "发送消息失败! " << endl;
        break;
}
else
cout << "发送成功" << endl << endl;
```

这样发送功能就实现了，现在来看接收功能的实现：

同理，为了防止只接受一次消息线程就结束，因此采用 while(1){}

```
void Receive(void* param)
{
        while (!is_Quit){
                接收的操作
        }
}
```

首先调用 recv 函数，来接受来自客户端的消息：

```
SOCKET client = *(SOCKET*)(param);
char recvbuf[MAXBYTE] = {0};
char client_time[20];
memset(client_time, 0, sizeof(client_time));
if (SOCKET_ERROR == recv(client, recvbuf, MAXBYTE, 0))
{
        is_Quit=true;
        cout << "数据接受失败! " << endl;
        break;
}else{
        \\接收成功
}
```

接下来就是分析消息的操作，如发送操作中说的那样，消息的前 19 个元素是时间标签，后面才是内容。

因此，使用一个数组 client_time 来存储时间，分开打印时间和内容：

```
memcpy(client_time, recvbuf, 19 * sizeof(char));
cout << endl << "客户端时间: " << client_time << " 消息内容:";
for (int i = 19; i < sizeof(recvbuf); i++)
cout << recvbuf[i];
cout << endl;
```

(二) 客户端程序的实现

客户端相较于服务器端实现更为简单，而且这是一个两人聊天协议，发送和接收线程的实现一致，相关部分直接贴代码，解释同服务器端实现。

先是建立嵌套字 socket，完成后向服务器端发送请求：

```
cout << "客户端" << endl;
WSADATA wsadata;
if (0 != WSAStartup(MAKEWORD(2, 2), &wsadata)) {
        cout << "嵌套字初始化失败" << endl;
        return 0;
}

SOCKET client = INVALID_SOCKET;

if ((client = socket(AF_INET, SOCK_STREAM, 0)) == SOCKET_ERROR){
        cout << "套接字创建失败!" << endl;
        return 0;
}
else
cout << "嵌套字创建成功" << endl;

struct sockaddr_in serverAddr;
memset(&serverAddr, 0, sizeof(sockaddr_in));
serverAddr.sin_family = AF_INET;
serverAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");  //本机的IP地址
serverAddr.sin_port = htons(10086); //端口号

if (SOCKET_ERROR == connect(client, (SOCKADDR*)&serverAddr, sizeof(serverAddr
    ))) {
        cout << "与服务器端连接失败" << endl;
        return 0;
}
else{
        cout << "与服务器端连接成功" << endl;
}
```

成功与服务器端建立连接后，主线程创建两个线程，接收线程和发送线程，随后主线程就进入睡眠：

```
_beginthread(Receive, 0, &client);
_beginthread(Send, 0, &client);
```

```
3
4  while (!is_Quit) { //判断是否退出
5          Sleep(1000);
6  }
7
8  if (client != INVALID_SOCKET) {
9          closesocket(client);
10         client = INVALID_SOCKET;
11 }
12
13 WSACleanup();
14 cout << "客户端退出! " << endl;
15 return 0;
```

然后发送线程和接收线程的实现与服务器的的两个子线程是一致的，下面展示实现代码，解释同服务器端处：

```
1  void Receive(void* param){
2          while (!is_Quit) {
3                  SOCKET client = *(SOCKET*)(param);
4                  char recvbuf[MAXBYTE] = { 0 };
5                  char server_time[20];
6                  memset(server_time, 0, sizeof(server_time));
7                  if (SOCKET_ERROR == recv(client, recvbuf, MAXBYTE, 0)){
8                          is_Quit=true;
9                          cout << "数据接受失败! " << endl;
10                         break;
11                 }
12                 else {
13                         cout << "接收成功: " << endl;
14                         memcpy(server_time, recvbuf, 19 * sizeof(char));
15                         cout << endl << "服务器端发送时间: " << server_time
                               << " 消息: ";
16                         for (int i = 19; i < sizeof(recvbuf); i++)
17                         cout << recvbuf[i];
18                         cout << endl << endl;
19                         time_t  t;
20                         char buf[20];
21                         memset(buf, 0, sizeof(buf));
22                         struct tm* tmp;
23                         t = time(NULL);
24                         tmp = localtime(&t);
25                         strftime(buf, sizeof(buf), "%Y-%m-%d %H:%M:%S", tmp);
26                         cout << "当前时间: " << buf << " 你要发送的内容是: ";
27                         memset(recvbuf, 0, sizeof(recvbuf));
28                 }
29         }
30 }
31 void Send(void* param){
```

```
32          while (1){
33                  time_t  t;
34                  char buf[20];
35                  memset(buf, 0, sizeof(buf));
36                  struct tm* tmp;
37                  t = time(NULL);
38                  tmp = localtime(&t);
39                  strftime(buf, sizeof(buf), "%Y-%m-%d %H:%M:%S", tmp);
40                  SOCKET revClientSocket = *(SOCKET*)(param);
41                  char sendbuf[MAXBYTE] = { 0 };
42
43                  cout << "当前时间: " << buf << " 你要发送的内容是: ";
44                  strcpy(sendbuf, buf);
45                  string _tmp;
46                  cin >> _tmp;
47                  if (_tmp == "quit") {
48                          is_Quit = true;
49                          continue;
50                  }
51                  strcat(sendbuf, _tmp.c_str());
52
53                  if (SOCKET_ERROR == send(revClientSocket, sendbuf, strlen(
                        sendbuf), 0)) {
54                          is_Quit=true;
55                          cout << "发送消息失败! " << endl;
56                          break;
57                  }
58                  else
59                  cout << "发送成功" << endl << endl;
60          }
61  }
```

五、 程序运行

先启动服务器端，后期的客户端，启动后结果：

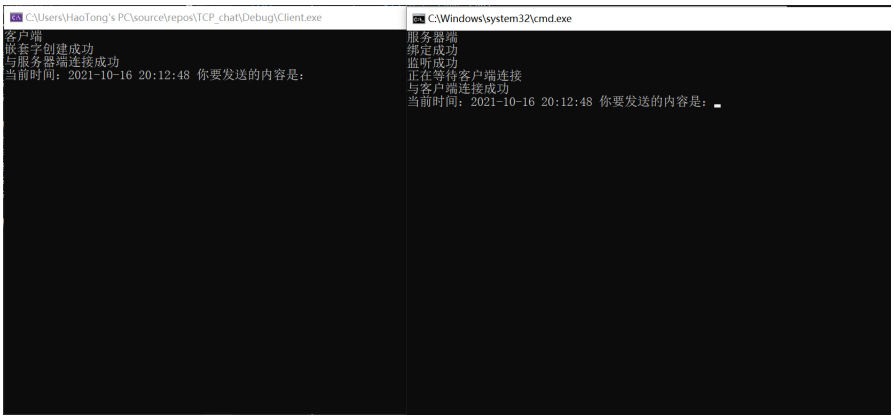

图 2: 左控制台客户端 右控制台服务器

随后进行测试，客户端向服务器端发两条消息，服务器回应一条消息：

可以看到，带有时间标签的聊天功能实现了，发送和接收线程互不冲突。
然后，观察输入“quit”只能，服务器端或者客户端能否正常退出。

可以看到服务器端和客户端的退出功能是正常的。

六、 总结

本次实验，实现的 TCP 协议的两人聊天协议。

了解到了 TCP 连接的代码实现，是可靠稳定的连接。

两人连接协议是实现客户端和服务器端的相互通话。无论是服务器还是客户，都实现多线程(发送消息和接收消息)，使得发送功能和接收功能互不冲突。

本次实验是要实现发送的消息带有时间标签，我的实现方法是用一个定长数组作为消息的发送数组，数组头 (也就是一段固定长度的数组) 存储当前的时间，在此之后则是要发送的消息内容。当接受消息是，接受数组保存的内容也可根据发送消息的格式，先从数组头部固定长度取出时间标签，再接下来的是消息内容。

然后需要一点技巧的是如何实现退出功能，在全局变量中设置一个 is_Quit=false, 在为假的情况下，主线程继续睡眠，两个子线程也可继续执行。当检测到输入的信息为 quit 时，则将 is_quit 设置为真，两个子线程发现为真，则结束，主线程停止睡眠，关闭 socket 嵌套字。