南開大學
Nankai University

南　　开　　大　　学

计　算　机　学　院

计算机网络实验报告

利用 Socket 设计和编写一个聊天程序

马永田

年级：2020 级

专业：计算机科学与技术

指导教师：张建忠 & 徐敬东

2022 年 10 月 22 日

摘要

使用流式 Socket，设计一个两人聊天协议，并实现聊天程序，对聊天协议以及程序的模块进行讲解。

关键字：Socket　C++　TCP/IP　聊天

目录

一、 实验要求

1. 使用流式 Socket，设计一个两人聊天协议，要求聊天信息带有时间标签。请完整地说明交互消息的类型、语法、语义、时序等具体的消息处理方式。
2. 对聊天程序进行设计。给出模块划分说明、模块的功能和模块的流程图。
3. 在 Windows 系统下，利用 C/C++ 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。

二、 实验设计

(一) 协议设计

1. 消息类型

当客户端与服务器连接成功时，服务器会向客户端发送服务器名称，客户端也会向服务器发送客户端名称，此外服务器还会向客户端进行广播，告知上线。之后客户端输入的内容会被当做要发送的消息识别，当有客户端离线时服务器也会进行离线广播。

2. 语法

服务器发送的消息依次分为时间标签段、名称段、以及信息段，若为私发消息，会附带私聊标签，只发送给进行私聊的客户端，此外服务器中会打印日志，不同的日志信息也会有不同的标签；客户端发送的消息也可能会在头部包含退出和私聊区段。其中服务器发送的消息均以空格分割，日志标签与私聊标签以 [ ] 括起。

3. 语义

时间标签段即表示该消息打印的时间，名称段则是消息发出者的名称，信息段则是要发送的信息，私聊标签分为 [private chat] 和 [private chat to XXX](其中 XXX 为接受私聊消息的客户端名称)，分别表示接收到的是私聊消息以及发出的是私聊消息。

客户端在输入时，消息头包含 [XXX]（XXX 为某客户端名称）时表示向 XXX 私发消息，其他客户端无法收到，输入 exit 表示断开连接。

日志中 INFO 表示通知，OK 表示成功执行，GET 表示获取,SEND 为公聊，PSEND 为私聊，SSEND 为服务器广播，JOIN 为客户端加入，LEFT 为客户端离线。

4. 时序

server 与 client 首先均进行 Socket 初始化，之后 server 绑定套接字、IP、端口号后，进入 listen 状态监听客户端请求，client 发出 connect 请求以后 server 对 client 进行 accept 建立连接，之后 server 创建消息接收分发线程，任一 client 对应的线程中 recv 到消息后，都会进行处理并 send 广播给所有的 client；而 client 在成功 connect 后创建消息发送线程和消息接收线程，server 发送后 client 会 recv 到消息并打印，而 client 在完成输入后则会将消息 send 到 serve，当 client 离线或 server 退出时会进行 close。

(二) 聊天程序设计

篇幅原因报告中省略部分打印语句，如下为服务器与客户端的模块讲解以及程序运行流程，其中关键代码均附有注释，用于进一步讲解不同模块的功能以及实现。

1. 服务端

其中服务端分为声明定义、服务器初始化、监听并接受客户端、消息接收与分发、消息发送、断开连接几个模块：

声明定义：定义需要用到的结构体、变量等。

```
#define MAX_CLIENT 50 //最大客户端数
#define MAX_BYTES 512 //消息最大字节数

struct Client {//客户端
    SOCKET clientSocket;//套接字
    char clientName[20];//客户端名称
    bool flag = true;//是否在线标志
};
Client Clients[MAX_CLIENT];//存储连接到服务器的客户端信息
char serverName[20];//服务器名称
int clientNum = 0;//连接到服务器的客户端数量
int port = 1234;//端口号
```

服务器初始化：该模块在 init 函数中，在该阶段进行服务器相关内容的初始化。进行服务器名、服务器端口的设置，初始化 DLL 库，创建流式套接字并绑定。

```
SOCKET init() {
//服务器初始化模块关键内容
    cin >> serverName;
    cin >> port;
    //初始化 DLL
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        return WSAGetLastError();
    }
    //创建流式套接字
    //第一个参数 协议簇（AF_INET，ipv4；AF_INET6，ipv6；AF_UNIX，本机通信）
    //第二个参数 类型（SOCK_STREAM，TCP流；SOCK_DGRAM，UDP数据报；SOCK_RAW，
        原始套接字）
    //第三个参数 一般设置0，当确定套接字使用的协议簇和类型时，这个参数的值就
        为0
    SOCKET servSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (servSock == INVALID_SOCKET) {
        return WSAGetLastError();
    }
    //绑定套接字 端口和ip
    struct sockaddr_in sockAddr;
    memset(&sockAddr, 0, sizeof(sockAddr));  //每个字节都用0填充
    sockAddr.sin_family = AF_INET;  //使用IPv4地址
```

```
    sockAddr.sin_addr.s_addr = inet_addr("0.0.0.0");   //具体的IP地址
    sockAddr.sin_port = htons(port);   //端口
    if (bind(servSock, (SOCKADDR*)&sockAddr, sizeof(SOCKADDR))) {
        return WSAGetLastError();
    }
    return servSock;
}
```

监听并接受客户端：该模块在 listen 函数中在该阶段中服务器进入监听状态，不断监听客户端的请求。每当接收到一个客户端的请求并成功建立连接后，就将对应的套接字存储，并与客户端互相收发存储姓名，之后创建消息接受与分发线程。当第一个客户端加入时，会创建一次消息发送线程，之后不会再创建。

```
void listen(SOCKET servSock) {
//进入监听状态
    listen(servSock, 20);
    //接收客户端请求
    while (true) {
        sockaddr_in  addrClnt;
        int len = sizeof(sockaddr_in);
        //接受成功 返回与Client通讯的Scoket 存储到Clients中
        Clients[clientNum].clientSocket = accept(servSock, (SOCKADDR*)&
            addrClnt, &len);
        if (Clients[clientNum].clientSocket != INVALID_SOCKET) {
            send(Clients[clientNum].clientSocket, serverName, 20, 0);//发送服
                务器名称
            char buf_r[20] = { 0 };
            int ret = recv(Clients[clientNum].clientSocket, buf_r, 20, 0);
            strcpy(Clients[clientNum].clientName, buf_r); //获取客户端名称
            if (clientNum == 0) {//创建服务器广播线程
                HANDLE hThread = CreateThread(NULL, 0, sendThreadFunc, (
                    LPVOID)clientNum, 0, NULL);
                CloseHandle(hThread);
            }
            //为接受的客户端创建收发线程 传入Client索引
            HANDLE hThread = CreateThread(NULL, 0, recvThreadFunc, (LPVOID)
                clientNum, 0, NULL);
            CloseHandle(hThread);
            clientNum++;
            for (int i = 0; i < clientNum; i++) {//广播客户端的加入
                if (!Clients[i].flag) {
                    continue;
                }
                char buf_s[MAX_BYTES] = { 0 };
                strcpy(buf_s, Clients[clientNum - 1].clientName);
                strcat(buf_s," just joins,welcome!");
                send(Clients[i].clientSocket, buf_s, MAX_BYTES, 0);
            }
        }
```

```
        else {//连接失败
            break;
        }
    }
}
```

消息接收与分发：该模块在 recvThreadFunc 函数中，多线程执行，用于接受客户端发送的消息，每个线程都会对应一个客户端，专门接收其消息，之后转发给所有还在线的用户。若连接失败则表示对应客户端已退出聊天室，将其 flag 置 false；若收到的消息中头部为 [XXX]，其中 XXX 为存在的客户端名称，则会将消息单独发送给 XXX，其他客户端不会收到消息。

```
DWORD WINAPI recvThreadFunc(LPVOID clientId) {
//与客户端通讯，接受客户端信息并转发
    int id = (int)clientId;//获取该线程所服务的客户端Id
    while (true) {
        char pClnt[20];//私聊名
        char buf_r[MAX_BYTES] = { 0 };//原消息
        char buf_rt[MAX_BYTES] = { 0 };//私聊消息
        bool pflag = false;//私聊标志
        int ret = recv(Clients[id].clientSocket, buf_r, MAX_BYTES, 0);
        if (ret <= 0) {//客户端断开连接
            Clients[id].flag = false;//在线标志置FALSE
            for (int i = 0; i < clientNum; i++) {//离线广播
                if (!Clients[i].flag) {
                    continue;
                }
                char buf_s[MAX_BYTES] = { 0 };
                strcpy(buf_s, Clients[id].clientName);
                strcat(buf_s, "just left,bye!");
                send(Clients[i].clientSocket, buf_s, MAX_BYTES, 0);
            }
            return 0;
        }
        else if (ret > 0) {
            if (buf_r[0] == '[') {//判断是否私聊
                int index = 1;
                while (buf_r[index] != ']' && index < 20) {
                    pClnt[index - 1] = buf_r[index];
                    index++;
                }
                pClnt[index-1] = '\0';//截取私聊用户名存到pClnt中
                strncpy(buf_rt, buf_r+index+1, 100-index);//截取私聊信息
                for (int i = 0; i < clientNum; i++) {//匹配私聊名称
                    if (strcmp(pClnt, Clients[i].clientName) == 0) {
                        char buf_s[MAX_BYTES] = { 0 };
                        strcpy(buf_s, Clients[id].clientName);
                        strcat(buf_s, " [private chat]: ");
                        strcat(buf_s, buf_rt);
                        send(Clients[i].clientSocket, buf_s, MAX_BYTES, 0);
```

```
39
40                          char buf_s2[MAX_BYTES] = { 0 };
41                          strcpy(buf_s2, "[private chat to ");
42                          strcat(buf_s2, Clients[i].clientName);
43                          strcat(buf_s2, "]: ");
44                          strcat(buf_s2, buf_rt);
45                          send(Clients[id].clientSocket, buf_s2, MAX_BYTES, 0);
46                          pflag = true;//分别发送到私聊双方并将pflag置true
47                      }
48                  }
49              }
50          }
51          if (pflag) {//若为私聊则打印对应日志
52              nowTime = getTime();
53              cout << nowTime << " [ PSEND ] " << Clients[id].clientName << "
                    to " << pClnt << ": " << buf_rt << endl;
54          }
55          else {//否则将消息进行广播
56              char buf_s[MAX_BYTES] = { 0 };
57              for (int j = 0; j < clientNum; j++){
58                  strcpy(buf_s, Clients[id].clientName);
59                  strcat(buf_s, ": ");
60                  strcat(buf_s, buf_r);
61                  if (!Clients[j].flag) {
62                      continue;
63                  }
64                  int ret = send(Clients[j].clientSocket, buf_s, MAX_BYTES, 0);
65              }
66          }
67      }
68      return 0;
69  }
```

消息发送：该模块在 sendThreadFunc 函数中，服务器也可以对所有用户进行广播消息，该部分放在一个线程中执行，当且仅当第一个客户端加入时线程被创建。

```
1   DWORD WINAPI sendThreadFunc(LPVOID clientId) {
2       //服务器向客户端广播消息
3       int id = (int)clientId;
4       while (true) {
5           char str[MAX_BYTES];
6           cin >> str;
7           char buf[MAX_BYTES];
8           strcpy(buf, serverName);
9           strcat(buf, ": ");
10          strcat(buf, str);
11          for(int i = 0; i < clientNum; i++) {
12              if (!Clients[i].flag) {
13                  continue;
```

```
            }
            int ret = send(Clients[i].clientSocket, buf, MAX_BYTES, 0);
        }
        nowTime = getTime();
        cout << nowTime << " [ SSEND ] " << serverName << ": " << str << endl
            ;
    }
    return 0;
}
```

断开连接：该阶段将断开连接。

```
void close(SOCKET servSock) {
    //关闭监听套接字
    closesocket(servSock);
    nowTime = getTime();
    cout << nowTime << " [ INFO  ] Close Socket!" << endl;
    //终止 DLL 的使用
    WSACleanup();
    nowTime = getTime();
    cout << nowTime << " [ INFO  ] Clean up WSA!" << endl;
}
```

2. 客户端

客户端分为变量声明、客户端初始化、连接服务器、消息接收、消息发送、断开连接几个模块:

变量声明：声明要用到的一些变量。

```
#define MAX_BYTES 512//消息最大字节数
char clientName[20];//客户端名称
char serverName[20];//服务器名称
int port = 1234;//端口号
string IP;//要连接的服务器IP地址
bool flag = true;//连接标志
```

客户端初始化：该模块在 init 函数中，在该阶段进行客户端相关内容的初始化。设置客户端名，选择要连接的服务器 IP 地址与端口号，初始化 DLL 库，创建流式套接字。

```
SOCKET init() {
    cin >> clientName;
    cin >> IP;
    cin >> port;
    //初始化 DLL
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {//失败
        flag = false;
        return WSAGetLastError();
    }
    //创建流式套接字
```

```
    SOCKET servSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (servSock == INVALID_SOCKET) {//失败
        flag = false;
        return WSAGetLastError();
    }
    return servSock;
}
```

连接服务器：该模块在 connect 函数中，在该阶段客户端向服务器发起请求，若连接成功则互相收发存储名称，并创建消息接收线程和消息发送线程.

```
void connect(SOCKET servSock) {
    //向服务器发起请求
    struct sockaddr_in sockAddr;
    sockAddr.sin_family = AF_INET;
    sockAddr.sin_addr.s_addr = inet_addr(IP.c_str());
    sockAddr.sin_port = htons(port);
    if (connect(servSock, (SOCKADDR*)&sockAddr, sizeof(SOCKADDR)) ==
        SOCKET_ERROR) {
        flag = false;
        return ;
    }
    //接收服务器传回的数据
    char buf_r[20] = { 0 };
    recv(servSock, buf_r, 20, 0);//接收服务器名
    strcpy(serverName, buf_r);
    send(servSock, clientName, 20, 0);//发送客户端名
    //创建消息接收线程
    HANDLE hThread = CreateThread(NULL, 0, recvThreadFunc, (LPVOID)servSock,
        0, NULL);
    CloseHandle(hThread);
    //创建消息发送线程
    hThread = CreateThread(NULL, 0, sendThreadFunc, (LPVOID)servSock, 0, NULL
        );
    CloseHandle(hThread);
}
```

消息接收：该模块在 recvThreadFunc 函数中，另起线程执行，用于持续接受服务器发送的信息.

```
DWORD WINAPI recvThreadFunc(LPVOID lpThreadParameter)
{//接收服务器的消息
    SOCKET sock = (SOCKET)lpThreadParameter;
    while (true) {
        char buf_r[MAX_BYTES] = { 0 };
        int ret = recv(sock, buf_r, MAX_BYTES, 0);
        if (ret > 0) {
            nowTime = getTime();
            cout << nowTime << " " << buf_r << endl;
        }
```

```
        else {//若连接错误 标志置false 退出连接
            flag = false;
            return 0;
        }
    }
}
```

消息发送：该模块在 sendThreadFunc 函数中，另起线程执行，用于向服务端发送信息，若输入的信息为 exit 则断开连接。

```
DWORD WINAPI sendThreadFunc(LPVOID lpThreadParameter)
{//向服务器发送消息
    SOCKET sock = (SOCKET)lpThreadParameter;
    while (true) {
        char buf[MAX_BYTES];
        cin >> buf;
        if (strcmp(buf, "exit") == 0) {//若输入为exit，连接标志位置false
            flag = false;
            return 0;
        }
        int ret = send(sock, buf, MAX_BYTES, 0);
        if (ret <= 0) {
            return 0;
        }
    }
}
```

断开连接：将主线程用 while 循环挂起，直到标志位为 false 后断开连接。

```
void close(SOCKET servSock) {
    while (flag) {//若flag为false 则跳出循环关闭连接。
    }
    //关闭套接字
    closesocket(servSock);
    //终止使用 DLL
    WSACleanup();
}
```

3. 程序运行流程

首先服务端进行初始化，之后服务端进入监听状态，等待客户端的请求，客户端在初始化后向服务器发起请求，成功建立连接之后，服务端与客户端互相发送名称，服务端创建消息接收与分发线程，而客户端则创建消息接收线程和消息发送线程，当某个客户端发送公屏消息、发送私聊或是请求断开时，服务端的消息接收和分发线程会进行对应的识别与处理。

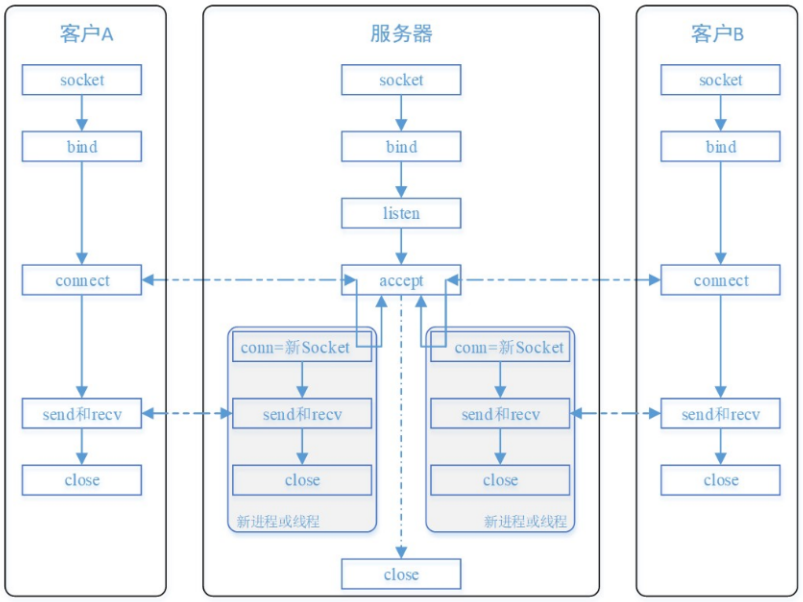

图 1: 流程

三、 实验结果分析

程序运行后如图2所示:

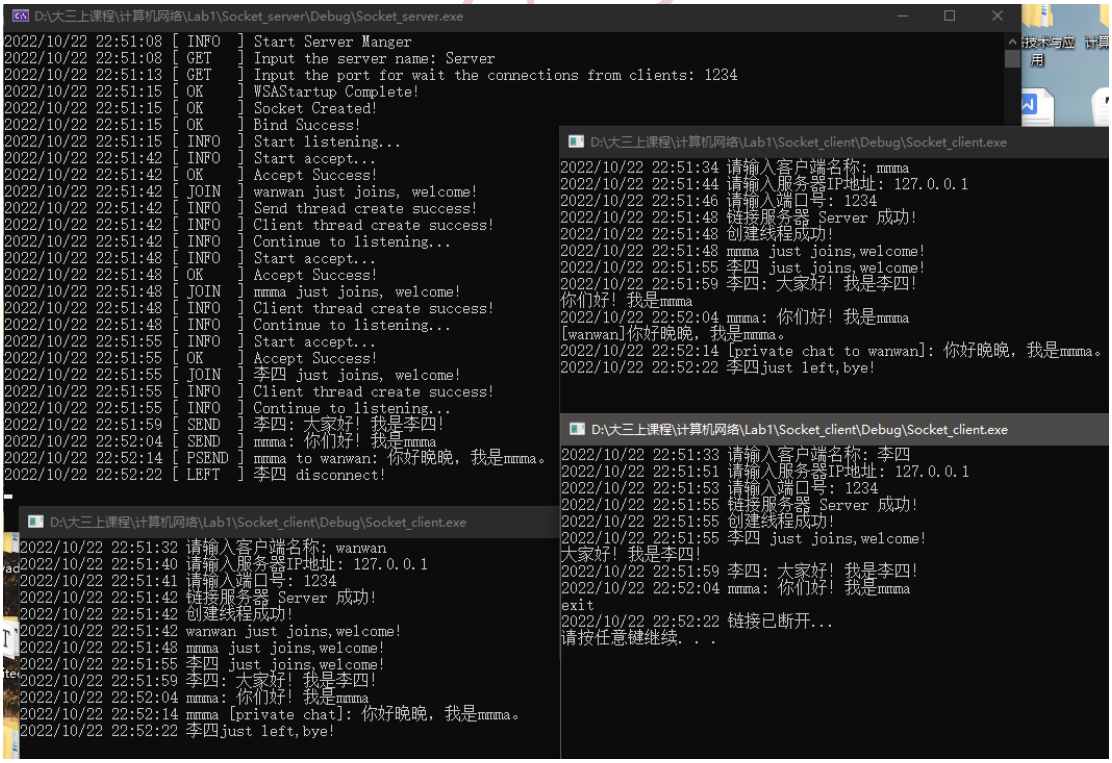

图 2: 程序运行结果

可以看到服务器正确打印日志信息，客户端发送的公聊信息其他的客户端均会收到，而私聊信息则只有相互私聊的两个客户端能够收到，且输入 exit 后客户端成功退出。