**河北农业大学**

**《C语言程序设计》**

**课程设计报告**

设 计 题 目： 多人聊天室

专 业 班 级： 软件工程1905 班

团 队 成 员： 许颖，高俊康，白欣雨

指 导 教 师： 王栋轩

日 期 ： 2019年12月31日

# 设计背景

该课程设计的初衷是想要实现一个游戏，这个游戏我曾使用易语言实现过，它是一个可多人在线对战的规则简单的棋类游戏，我想用C重写它。

与易语言不同，在C中，网络通信和图形界面只能从底层开始实现。

于是，我花了一个星期简单学习了socket接口，将其封装为一个静态库，并提供给小组成员，方便他们调用，大大减少了工作量。

而后，我又花了一个星期去学习win32接口，没想到的是，win32的体系太过庞大，其中很多细节和历史遗留问题盘根错节，而且一个功能常常可以有多个看似相同而在某些方面隐约不同的接口实现。

这时我意识到，实现游戏的工作量太大，于是组内临时改变目标：实现一个聊天室。

相对来讲，聊天室的UI简单，客户端与服务端之间的协议等于无，于是我们就顺着这个路线开发，最终实现了我们的课程设计。

# 设计过程

## 主要内容

### 1.两套封装好的Socket静态库（高俊康 负责）

含有：

（在C中实现面向对象封装的类库）

Gao\_socket\_Class.c

Gao\_socket\_Class.h

Gao\_socket\_Class.lib

（普通库）

Gao\_socket.c

Gao\_socket.h

Gao\_socket.lib

### 2.服务端（Server.c）（白欣雨、许颖 负责）

使用库Gao\_socket.lib

### 3.客户端（Client.c）（高俊康 负责）

使用库Gao\_socket\_Class.lib

## 额外知识

1. win32接口
2. socket接口
3. 回调函数的思想
4. 静态库的封装及头文件的内容
5. 面向对象的程序设计思想

## 其他说明

我们的课程设计提供了两个封装好的socket静态库，其中一个我称之为“类库”，他是另一个库的改进版本。它当然不是真的封装了一个类，而是一个大大的结构体，我通过结构体来实现C++或者JAVA之类支持面向对象的语言的类的功能。

为什么会有两个版本？在最初，我先实现了那个普通的socket库，后来发现了一些问题。通过请教老师（非常感谢吕老师对我的指导），我在原来的基础上重构了它，把它变成了一个“类库”。

因为我考虑到组内成员尚未接触面向对象的思想，所以仍然让他们使用原来的库。

至于我发现了什么问题，这里简单说一下。静态库提供的主要接口仅有两个，一个是创建服务端（Create\_serversock()），一个是创建客户端（Create\_clientsock()），其他接口均在上述两个函数被调用后才能够被调用，如发送数据，取对象，关闭对象等等。两个主要接口其内部封装了从socket()到listen()（或connect()）的所有步骤，并且开始监听端口（或接收数据）。重要的是，程序内部的变量只有一份，也就是说这套接口实际上可以看作是某类的一个实例化，那么它的局限性就凸显出来了：一个进程只能创建和使用仅一个客户端和服务端，试图多次调用创建客户端/服务端函数只会导致变量数据混乱。

于是我通过结构体来实现面向对象的类库。

其中较原来的库多了一个接口，它用来包装和初始化程序员创建的对象，从而所有接口内部使用的变量由程序员提供或是存在于栈空间里，多线程访问时不会发生冲突。

还将原来一些接口的参数删去，包装在类实例的成员中。

C的结构体没有私有成员这个概念。对于这个问题，我在头文件中将重要的成员名均注明private。（可以通过匿名位域解决这个问题，但首先匿名位域成员不存在内存对齐，其次写死了成员长度会使库不具有移植性）

# 难点及解决方案

## socket接口

### ws2\_32.lib链接问题：

由于我使用的IDE为 VS 2019，在最初的阶段，代码总是报错无法编译，多次搜索后找到解决办法：

在VS 2008及以上版本中，使用Socket时需要链接库: Ws2\_32.lib

#pragma comment(lib,"ws2\_32.lib")

### Socket初始化问题：

在学习了一些前置知识后，我开始尝试调用winsock2.h提供的api。

但在最初阶段，我调用socket() 获取文件描述符时总是返回-1（出错）。原因是：在Wndows下使用Socket需要先注册，或者叫绑定Socket api，它过函数WSAStartup()实现。

**MSDN**对**WSAStartup()**给出如下解释：

The WSAStartup function initiates use of the Winsock DLL by a process.

关键部分：

The WSAStartup function must be the first Windows Sockets function called by an application or DLL. It allows an application or DLL to specify the version of Windows Sockets required and retrieve details of the specific Windows Sockets implementation. The application or DLL can only issue further Windows Sockets functions after successfully calling WSAStartup.

在最后还需要调用**WSACleanup()** 函数注销自身：

The WSACleanup function terminates use of the Winsock 2 DLL (Ws2\_32.dll).

网上大部分较系统的讲解Socket的博客基本上都是在linux下的，而在linux下是不需要初始化Socket api的。所以这个问题确实困扰了我一小段时间。

### select() 对阻塞socket实现非阻塞操作

socket分为阻塞和非阻塞两种模式，Windows下所有创建的socket（包括accept()返回的socket描述符）均为阻塞模式，这也就导致我们无法同时干多件事。例如：1. 服务端无法在接收数据时接受用户的连接请求。2. 客户端无法在用户输入时接收数据。

而我们并不知道客户到底什么时候想要干什么，于是程序的设计思路就非常麻烦了。

这时候需要用到select() 轮询所有我想要监视的socket。当服务端开始监听端口后，整个程序就会跑在一个大while中。传入select中一个时间结构timeval作为超时时间，然后他就会在必要的时候提醒我们哪个socket可以操作了，或者是超过超时时间，让我们处理一些我们想要处理的东西。

我为什么不选择使用非阻塞模式的socket，第一是因为麻烦，想要实现非阻塞的socket，需要很多比较麻烦的技巧。第二是因为浪费资源，这种异步的思想，总会让我的程序在函数返回后疯狂检测执行状态，这显然有些浪费资源。第三是因为没必要，select()已经满足了我们课程设计开发的需要，没必要去故弄玄虚（或者说炫技），这是大炮打蚊子的做法。

### 面向对象实现

在上一大节我们提到了那个“类库”，实际上，这套接口的实现并不顺利。主要是在原理上。

最初我认为，每个对象的方法在调用时都是新的，即创建对象时整个类涉及的成员和方法都被copy了一遍。这句话的前半句话是对的，而后半句是错的。这导致我一直在寻找一个办法来copy一个函数在内存中的实现数据。

后来查到：同一个类的实例调用的方法的代码段在内存中均相同，这时我就非常疑惑不解。在支持面向对象的编程语言中，总会有多线程调用同类实例的同一方法的情况，而它却不会出现数据混乱。

再后来，我请教老师，了解到：当线程执行到内存中某个位置需要调用函数时，线程会把函数段copy到栈空间中执行。这时候一切都豁然开朗了。

原理问题解决了，后面就都是一些具体实现的细节问题了。我在头文件中向调用者提供了一个结构体，程序员首先声明这个结构体然后通过将其交给我们提供的包装接口初始化其方法来创建对象。当程序员调用某方法时，当前线程仅使用了这个结构体的成员及一些存在于当前线程栈中的数据。

到这里，就完成了这个面向对象的静态库，程序员可以在一个进程中创建多于一个的客户端及服务端而不会产生数据冲突。

最后，基于对socket的非常浅显的了解，我把它总结在了这篇博客里：

<https://blog.csdn.net/qq_16181837/article/details/103376556>

### 10038和10053错误

**MSDN**对**WSAENOTSOCK（10038）**的解释：

Socket operation on nonsocket.

An operation was attempted on something that is not a socket. Either the socket handle parameter did not reference a valid socket, or for select, a member of an fd\_set was not valid.

这个错误仅出现在客户端服务器共同单步调试的过程中，猜测是断点破坏了socket内核的一些逻辑。

这个错误没什么影响，之后也再没出现过。

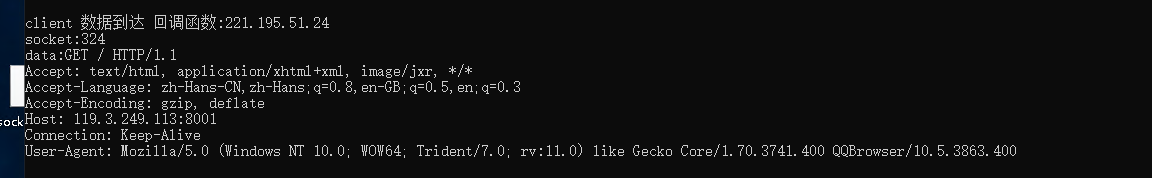
**MSDN**对**WSAECONNABORTED（10053）**的解释：

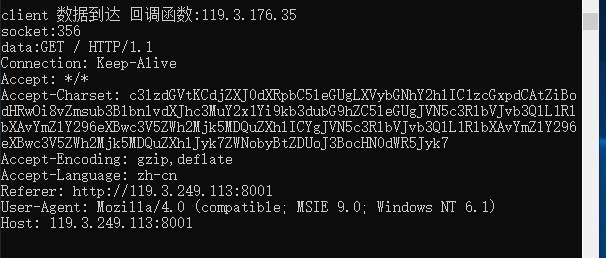
*Software caused connection abort.*

*An established connection was aborted by the software in your host computer, possibly due to a data transmission time-out or protocol error.*

10053错误很难重现，不过出现的概率也很低。

总是在我连接上远程桌面后，看到服务端疯狂被error回调函数刷屏。但奇怪的是，此时客户端仍然可以正常使用。

不知道为什么，服务端监听的端口总是有Http请求，像下面这样：

由于服务端监听的端口总是有Http请求，而且出现错误的周期也很长很长，所以到底是哪个客户的哪个行为导致了这个错误，就很难定位

后来猜测是一些Http请求导致的问题，于是我让服务端记录日志，看看到底是客户端访问导致的还是Http请求导致的，但到目前为止还没触发过10053错误，所以仍然原因不明。

为了尽量避免这个问题，我在最终完成的课设中改了端口。

## win32接口

### 文本框(EDIT)换行

这个问题贯穿着客户端部分的整个开发过程，直到接近完成时才终于解决。

当我改变Windows窗口的尺寸时，文本框中原本已经整整齐齐的文本会突然缩在一起——所有换行符莫名其妙地消失了。

网上查资料查了很久很久都未果，身边也没有任何一个研究过win32的人。开发过程中我一度把窗口风格中的可调尺寸去掉。直到一次偶然，具体细节记不清了，我在单步调试文本拼接的问题时突然发现：在文本框中，换行是“\r\n”而非“\n”。

于是问题就这么解决了，可能看起来我很轻描淡写，而且原因居然如此简单，但这个问题确实困扰了我非常久。

至于为什么在窗口尺寸未改变时，文本框的输出换行是正常的，这个尚未解决，我也没有任何思路。

### 字符集的选择

我经常在别人的代码里见到TEXT()这个宏。根据我对win32的浅显了解，相应的还有\_T() 、\_TEXT()和“在字符串前加一个L”。它们的作用相同，仅有微小的区别。但想要说明白他我甚至需要再写一篇文章，所以不再赘述。大概来讲，它们表示将ANSI字符串转换为Unicode字符串。

我在学习一个新技术时总是喜欢一边写一遍看，于是不管是Socket还是win32，每次在基本结束学习时都会有一个几百到一千行的源文件，里面包含着我调试各种功能的代码。

然而当我新建了一个工程然后真正开始写客户端的界面时，出现了令人百思不得其解的问题：所有地方都出现了乱码问题，而且还是韩文。所有子窗口（窗口控件）都没有创建成功。倒是窗口的消息队列还正常运转着，窗口并没有卡死。

然而我仔细对比了两段代码的调用顺序，看不出任何区别。我甚至把两段代码从头到尾单步运行了一遍，不知道什么原因，客户端在创建子窗口时总是出错，返回一个NULL，而其他一切正常，除了乱码。

后来，我把两段代码交换了位置，用来调试学习的工程完美运行，客户端的工程还是出错乱码。这时候我意识到可能是环境配置的问题，于是我打开两个工程的属性页，一条一条对比，终于发现了问题所在：在客户端工程下，我使用的是Unicode字符集，而在用来调试学习的工程下我使用的是多字节字符集（MBCS），当我把它改成多字节字符集后，一切都正常了。

多字节字符集这种说法实际上是错误的，应该叫“多字节字符系统”。从这里可以看出来，多字节字符系统与Unicode字符集并不是并列关系，它是一种编码类型，而Unicode是一种具体的编码方案。

多字节字符系统（Multi-Byte Chactacter System），指由ANSI的ASCII字符集派生（或扩充）来的一系列字符集。这些字符集种类繁多，不便于在国际上交流时的转换，于是出现了Unicode这样统一的字符集。

在使用win32接口进行图形界面的编程时就需要注意，如果我们使用的是Unicode字符集，那么就必须将所有字符串通过上述提到的宏来处理，否则就会出现令人抓狂的乱码问题。

### 文本框消息

由于Enter键的限制（实际上是因为懒），我的发送文本框设计为不可换行，也就是每次仅可发送一行数据，其显示范围我也让其恰好显示一行文本。

我想在Windows的消息回调函数中判断Enter键的按下状态，然后发送数据。于是我查了窗口按键的消息，有很多，其中一个大概是WM\_KEYDOWN。然而我却发现，我在文本框按键后根本没有触发这些消息。后来我分析了所有传入回调函数的消息，发现全部是针对父窗口的，子窗口的消息在分发消息之前就已经被处理了。

于是我找到了主函数中的Getmessage()，既然它的作用是从消息队列中取一条消息出来，那么他这里的消息一定包含有我想要的消息，于是我尝试直接判断传入实参的MSG结构来判断子窗口的消息。

在MSG结构下的hwnd是当前消息目标窗口的句柄，message是消息，剩下的是该消息的参数，这就很好办了，通过判断该消息的目标窗口句柄和该消息类型及消息的相应参数即可捕捉到我们想要的事件。

键盘消息拦截已经解决，然而又出现了新问题：每次按下Enter后，文本框中总会留下一个换行符。这让非常疑惑，同时也感受到了win32接口其其体系之复杂，细节之繁复。我很难定位这个换行到底是在哪里被发送到文本框的。

经过不断地各种断点调试修改，最终定位了问题所在。

我们知道，就像固定格式一样，在处理Windows窗口消息时，我们通常会写一个while，然后让GetMessage()跑在里面，紧接着下面就是翻译消息（TranslateMessage()）和分派消息给回调函数（DispatchMessage()）。而我的消息拦截则放在了以上三个函数的下面。尝试过各种方法无果后，我尝试改变他们的位置，然后不断地获取文本框的内容，同时监视着msg->message的值。

这次我猜对了，是翻译和消息分发函数的问题。

我发现，如果我拦截消息前后调用了上述两个函数，那么下一次循环取到的消息会是258（WM\_CHAR）

**MSDN**对该消息的解释：

Posted to the window with the keyboard focus when a WM\_KEYDOWN message is translated by the TranslateMessage function. The WM\_CHAR message contains the character code of the key that was pressed.

也就是说，在下一次循环时，该消息会通过分发消息函数被发送到指定窗口。于是在我拦截了键盘消息的后一次循环中，一个换行符被发送到了文本框中。

WM\_KEYDOWN（或WM\_KEYFIRST，两消息等值）是键盘消息的第一个消息，如果在最开始就把他拦截下，跳过TranslateMessage()，那么这个事件就相当于没被触发过。

所以解决方法就是先拦截消息，然后直接continue。

### 创建子窗口失败

这个问题折磨了我一个晚上。

最初，我在学习win32接口时尝试创建了很多控件，一切正常。然而当我准备一次性创建所有我需要用到的控件时，所有控件的创建均失败了。通过GetLastError()我获得了错误码1406，意外的是，百度根本都查不到。

**MSDN**上的解释：

1406 (0x57E)

Cannot create a top-level child window.

这个解释很模糊，无法创建上层子窗口，我确实是没能创建这个子窗口，没有具体原因。

我在Bing，Google上查了很久，确实找不到相关信息。于是只能靠自己了。

我从头到尾单步运行了几遍，检查了每个变量和参数的传递情况，最终发现，当我试图调用CreateWindow()创建控件时，参数hWndParent（父窗口句柄）的实参是NULL。

实参是一个全局变量，它在主函数中由CreateWindow()返回。然而我是在windows的回调函数的WM\_CREAT消息下开始创建控件的，这个消息在窗口创建完毕时被发送到该窗口。这时候我意识到，也许此时主函数中的CreateWindow()尚未返回。

我突然想到了易语言中的窗口创建完毕事件，在那个回调函数下执行代码时窗口还没有显示。其实这也就对应着尚未调用ShowWindow()及UpdateWindow()（这个函数可不写，仅为绕过队列发送一个重画消息）函数，那么很可能此时CreateWindow()确实还未返回。带着这些关键字在百度上查了一下，果然如此。

解决方法很简单，就是使用windows回调函数传入的第一个参数hWnd，它就是该窗口的句柄。

# 总结

这次课程设计，整个小组都学到了很多东西，

了解了网络通信和Windows下的图形界面编程，还有一些编程思想，同时也增强了自身的学习能力和Debug能力。