一对多聊天——Socket编程实验报告

PB20111686 黄瑞轩

本实验的目的在于建立多个用户之间的消息传输。

如何区分不同的用户?

在网络中，使用IP地址+协议+端口可以唯一标识一台终端上的某一进程。本实验采用TCP协议。

如何获取本机IP地址？由于这不是本实验所关心的内容，因此笔者在网上查到了这样一个函数并直接在实验中复用。

```
void getLocalIP(char local_ipv4[], int n) {
    //获得私网IP
    gethostname(local_ipv4, n);
    HOSTENT* host = gethostbyname(local_ipv4);
    in_addr PcAddr;
    for (int i = 0;; i++) {
        char* p = host->h_addr_list[i];
        if (NULL == p) {
            break;
        }
        memcpy(&(PcAddr.S_un.S_addr), p, host->h_length);
        strcpy(local_ipv4, inet_ntoa(PcAddr));
    }
}
```

Socket是什么?

所谓Socket(套接字)，就是对网络中不同主机上的应用进程之间进行双向通信的端点的抽象。一个套接字就是网络上进程通信的一端，提供了应用层进程利用网络协议交换数据的机制。它是应用程序通过网络协议进行通信的接口，也是应用程序与网络协议根进行交互的接口。Socket可以看成是两个网络应用程序进行通信时，各自通信连接中的端点，这是一个逻辑上的概念。它是网络环境中进程间通信的API(应用程序编程接口)，也是可以被命名和寻址的通信端点，使用中的每一个套接字都有其类型和一个与之相连进程。通信时其中一个网络应用程序将要传输的一段信息写入它所在主机的 Socket中，该Socket通过与网络接口卡(NIC)相连的传输介质将这段信息送到另外一台主机的 Socket中，使对方能够接收到这段信息。 Socket是由IP地址和端口结合的，提供向应用层进程传送数据包的机制。

如何使用Socket?

下面是一些Socket函数的介绍。

socket()

socket函数原型：

```
int socket(int domain, int type, int protocol);
```

`socket()` 用于创建一个socket描述符，它唯一标识一个socket。这个socket描述字跟文件描述字一样，后续的操作都有用到它，把它作为参数，通过它来进行一些读写操作。

实验中：

```
SOCKET socket_server = socket(AF_INET, SOCK_STREAM, 0);
SOCKADDR_IN address_server;//服务器端口地址

//初始化
address_server.sin_addr.S_un.S_addr = inet_addr(local_ipv4);//本机电脑(服务器)ip
address_server.sin_family = AF_INET;
address_server.sin_port = htons(port);
```

创建socket的时候，也可以指定不同的参数创建不同的socket描述符，socket函数的三个参数分别为：

- `domain`：协议域。协议族决定了socket的地址类型，在通信中必须采用对应的地址，AF_INET决定了要用ipv4地址（32位的）与端口号（16位的）的组合。
- `type`：socket类型。常用的socket类型有，SOCK_STREAM、SOCK_DGRAM等等，SOCK_STREAM对应于TCP协议模式。
- `protocol`：指定协议。常用的协议有，IPPROTO_TCP、IPPTOTO_UDP等，分别对应TCP传输协议、UDP传输协议等。

当 `protocol` 为0时，会自动选择type类型对应的默认协议。

当我们调用 `socket()` 创建一个socket时，返回的socket描述字存在于协议族（address family，AF_XXX）空间中，但没有一个具体的地址。如果想要给它赋值一个地址，就必须调用 `bind()` 函数，否则当调用 `connect()` 、 `listen()` 时系统会自动随机分配一个端口。

bind()

`bind()` 函数把一个地址族中的特定地址赋给 `socket` 。函数原型：

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

实验中：

```
bind(socket_server, (SOCKADDR*)&address_server, sizeof(SOCKADDR));
```

函数的三个参数分别为：

- `sockfd`：即socket描述字，它是通过`socket()`函数创建的，唯一标识一个socket。`bind()`函数就是将给这个描述字绑定一个名字。
- `addr`：一个`const struct sockaddr*`指针，指向要绑定给`sockfd`的协议地址。这个地址结构根据地址创建`socket`时的地址协议族的不同而不同，如ipv4对应的是：

```
struct sockaddr_in {
    sa_family_t    sin_family;
    in_port_t      sin_port;
    struct in_addr sin_addr;
};
struct in_addr {
    uint32_t       s_addr;
};
```

- `addrlen`：对应的是地址的长度。

在将一个地址绑定到socket的时候，需要先将主机字节序转换成为网络字节序。（`inet_addr()`函数）

```
address_server.sin_addr.S_un.S_addr = inet_addr(local_ipv4);
```

listen()、connect()

如果作为一个服务器，在调用`socket()`、`bind()`之后就会调用`listen()`来监听这个socket，如果客户端这时调用`connect()`发出连接请求，服务器端就会接收到这个请求。

```
int listen(int sockfd, int backlog);
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

listen函数的第一个参数即为要监听的socket描述字，第二个参数为相应socket可以排队的最大连接个数。`socket()`函数创建的socket默认是一个主动类型的，listen函数将socket变为被动类型的，等待客户的连接请求。

connect函数的第一个参数即为客户端的socket描述字，第二参数为服务器的socket地址，第三个参数为socket地址的长度。客户端通过调用connect函数来建立与TCP服务器的连接。

accept()

TCP服务器端依次调用`socket()`、`bind()`、`listen()`之后，就会监听指定的socket地址了。TCP客户端依次调用`socket()`、`connect()`之后就想TCP服务器发送了一个连接请求。TCP服务器监听到这个请求之后，就会调用`accept()`函数取接收请求，这样连接就建立好了。之后就可以开始网络I/O操作了。

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

accept函数的第一个参数为服务器的 `socket` 描述字，第二个参数为指向 `struct sockaddr` 的指针，用于返回客户端的协议地址，第三个参数为协议地址的长度。如果accpet成功，那么其返回值是由内核自动生成的一个全新的描述字，代表与返回客户的TCP连接。

注意：accept的第一个参数为服务器的socket描述字，是服务器开始调用 `socket()` 函数生成的，称为监听socket描述字；而accept函数返回的是已连接的socket描述字。一个服务器通常通常仅仅只创建一个监听socket描述字，它在该服务器的生命周期内一直存在。内核为每个由服务器进程接受的客户连接创建了一个已连接socket描述字，当服务器完成了对某个客户的服务，相应的已连接socket描述字就被关闭。

• send()、recv()

```
int send(SOCKET s,const char FAR* buf, int len,int flags);
```

返回值：0 – 成功拷贝至发送缓冲区的字节数（可能小于len），

-1 – 出错，把错误信息告诉全局变量 `errno` .

其中：

`s` ： 发送端套接字描述符

`buf` ： 应用要发送数据的缓存

`len` ： 实际要发送的数据长度

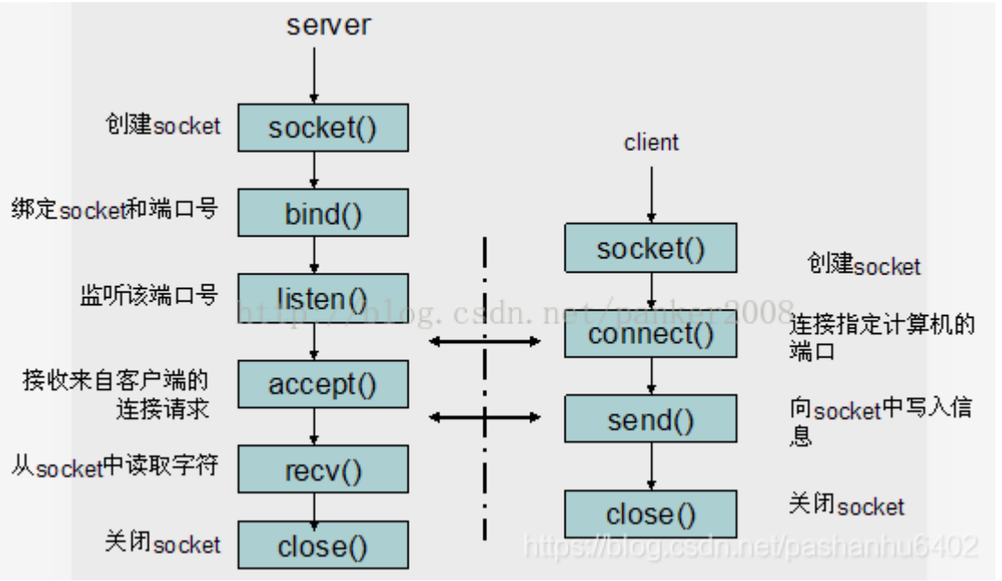
`flag` ： 一般设置为0

每个TCP套接口都有一个发送缓冲区，它的大小可以用 `SO_SNDBUF` 这个选项来改变。调用send函数的过程，实际是内核将用户数据拷贝至TCP套接口的发送缓冲区的过程：若len大于发送缓冲区大小，则返回-1；否则，查看缓冲区剩余空间是否容纳得下要发送的len长度，若不够，则拷贝一部分，并返回拷贝长度（指的是非阻塞send，若为阻塞send，则一定等待所有数据拷贝至缓冲区才返回，因此阻塞send返回值必定与len相等）；若缓冲区满，则等待发送，有剩余空间后拷贝至缓冲区；若在拷贝过程出现错误，则返回-1。关于错误的原因，查看errno的值。

如果send在等待协议发送数据时出现网络断开的情况，则会返回-1。注意：send成功返回并不代表对方已接收到数据，如果后续的协议传输过程中出现网络错误，**下一个 `send` 便会返回-1发送错误。**TCP给对方的数据必须在对方给予确认时，方可删除发送缓冲区的数据。否则，会一直缓存在缓冲区直至发送成功（TCP可靠数据传输决定的）。

• 具体流程图

具体流程图如下，如果需要不间断I/O，用 `while(1)` 循环接受即可。



线程

线程是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务。

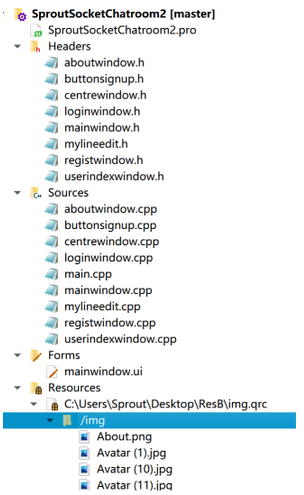
同一进程中的多条线程将共享该进程中的全部系统资源，如虚拟地址空间，文件描述符和信号处理等等。但同一进程中的多个线程有各自的调用栈，自己的寄存器环境，自己的线程本地存储。

一个进程可以有很多线程，每条线程并行执行不同的任务。

本实验使用`DWORD WINAPI Event_Thread(LPVOID lpParameter)`来创建线程，分别负责监听不同的client事件。

图形化

本实验使用QT 5.12版本进行图形化制作，项目文件树如下：



由main.cpp创建`MainWindow`窗口；

`MainWindow`窗口创建用户登录、用户注册、关于、聊天界面四个窗口。避免发送消息指令混杂，实验中用户注册、用户登录、聊天界面各分配一个端口，各由一个服务端进行服务。

用户注册

维护一个存放于 `FILE_PATH` 的txt文件，其格式如下：

```
HistoryUID:XX
HistoryTime:XXXXXXXXXXXXXXX

01
XXXX@mail.ustc.edu.cn
nickname
password

02
XXXY@mail.ustc.edu.cn
nickname2
password2
```

`RegistWindow` 接受用户注册信息输入，由 `RegistWindow` 先行进行单层校验后再发送消息至服务器。

本地作如下校验：检查所有输入信息的格式是否符合要求；

若符合要求，本地将开始联网，向服务端注册端口发送消息，格式如下：

```
email \n nickname \n password \n
```

服务端作如下校验：检查邮箱、昵称是否存在，如果存在，发回一个WINDOWS消息 `ERROR` ；如果不存在，写入注册文件并发回一个 `CORRECT` ，客户端监听回传消息。

用户登录

`LoginWindow` 接受用户登录信息输入，该窗口获得信息后转成如下格式

```
email \n password \n
```

后发给登录端口，登陆端口打开用户注册信息文件查找并核验，如果错误将发回一个WINDOWS消息 `ERROR` ；如果正确，将发回一个WINDOWS消息，直接告知Client该用户的nickname并使 `CentreWindow` 联网。

为何发送WINDOWS消息?

本实验使用WINDOWS API，因此在QT中，如果使用 `send()` 使用到子线程更新主线程UI，会被QT捕获并阻塞。利用 `SendMessage` 函数发出消息、重写的 `nativeEvent` 中 `MSG` 结构体接收消息。

聊天

最终实验界面如下：


Sprout Socket Chatroom(Build Version)
当前在线
黄瑞轩
123456
Sprout Socket Chatroom
123456 6月22日 21:14
大家好
黄瑞轩 6月22日 21:14
你好
[SERVER]服务器小哥哥 6月22日 21:14
大家好！大家作业写完了吗?
123456 6月22日 21:15
?
黄瑞轩
保存本次聊天记录
最多100字
send
Sprout Socket Chatroom
123456 6月22日 21:14
大家好
黄瑞轩 6月22日 21:14
你好
[SERVER]服务器小哥哥 6月22日 21:14
大家好！大家作业写完了吗?
123456 6月22日 21:15
?
123456
最多100字
send