

浙江大学

本科实验报告

课程名称： 计算机网络基础

实验名称： 基于 Socket 接口实现自定义协议通信

姓 名： 张雯琪

学 院： 计算机学院

系：

专 业： 计算机科学与技术

学 号： 3180103770

指导教师： 高艺

2020 年 12 月 13 日

浙江大学实验报告

实验名称： 基于 Socket 接口实现自定义协议通信 实验类型： 编程实验

同组学生： 无 实验地点： 计算机网络实验室

一、 实验目的

- 掌握 Socket 编程接口编写基本的网络应用软件

二、 实验内容

根据自定义的协议规范，使用 Socket 编程接口编写基本的网络应用软件。

- 掌握 C 语言形式的 Socket 编程接口用法，能够正确发送和接收网络数据包
- 开发一个客户端，实现人机交互界面和与服务器的通信
- 开发一个服务端，实现并发处理多个客户端的请求
- 程序界面不做要求，使用命令行或最简单的窗体即可
- 功能要求如下：
 1. 运输层协议采用 TCP
 2. 客户端采用交互菜单形式，用户可以选择以下功能：
 - a) 连接：请求连接到指定地址和端口的服务端
 - b) 断开连接：断开与服务端的连接
 - c) 获取时间：请求服务端给出当前时间
 - d) 获取名字：请求服务端给出其机器的名称
 - e) 活动连接列表：请求服务端给出当前连接的所有客户端信息（编号、IP 地址、端口等）
 - f) 发消息：请求服务端把消息转发给对应编号的客户端，该客户端收到后显示在屏幕上
 - g) 退出：断开连接并退出客户端程序
 3. 服务端接收到客户端请求后，根据客户端传过来的指令完成特定任务：
 - a) 向客户端传送服务端所在机器的当前时间
 - b) 向客户端传送服务端所在机器的名称
 - c) 向客户端传送当前连接的所有客户端信息
 - d) 将某客户端发送过来的内容转发给指定编号的其他客户端
 - e) 采用异步多线程编程模式，正确处理多个客户端同时连接，同时发送消息的情况
- 本实验涉及到网络数据包发送部分不能使用任何的 Socket 封装类，只能使用最底层的 C 语言形式的 Socket API
- 本实验可组成小组，服务端和客户端可由不同人来完成

三、 主要仪器设备

- 联网的 PC 机
- Visual C++、gcc 等 C++集成开发环境。

四、操作方法与实验步骤

- 小组分工：1 人负责编写服务端，1 人负责编写客户端
- 客户端编写步骤（需要采用多线程模式）
 - a) 运行初始化，调用 `socket()`，向操作系统申请 `socket` 句柄
 - b) 编写一个菜单功能，列出 7 个选项
 - c) 等待用户选择
 - d) 根据用户选择，做出相应的动作（未连接时，只能选连接功能和退出功能）
 1. 选择连接功能：请用户输入服务器 IP 和端口，然后调用 `connect()`，等待返回结果并打印。连接成功后设置连接状态为已连接。然后创建一个接收数据的子线程，循环调用 `receive()`，直至收到主线程通知退出。
 2. 选择断开功能：调用 `close()`，并设置连接状态为未连接。通知并等待子线程关闭。
 3. 选择获取时间功能：调用 `send()`将获取时间请求发送给服务器，接着等待接收数据的子线程返回结果，并根据响应数据包的内容，打印时间信息。
 4. 选择获取名字功能：调用 `send()`将获取名字请求发送给服务器，接着等待接收数据的子线程返回结果，并根据响应数据包的内容，打印名字信息。
 5. 选择获取客户端列表功能：调用 `send()`将获取客户端列表信息请求发送给服务器，接着等待接收数据的子线程返回结果，并根据响应数据包的内容，打印客户端列表信息（编号、IP 地址、端口等）。
 6. 选择发送消息功能（选择前需要先获得客户端列表）：请用户输入客户端的列表编号和要发送的内容，然后调用 `send()`将数据发送给服务器，观察另外一个客户端是否收到数据。
 7. 选择退出功能：判断连接状态是否为已连接，是则先调用断开功能，然后再退出程序。否则，直接退出程序。
 8. 主线程除了在等待用户的输入外，还在处理子线程的消息队列，如果有消息到达，则进行处理，如果是响应消息，则打印响应消息的数据内容（比如时间、名字、客户端列表等）；如果是指示消息，则打印指示消息的内容（比如服务器转发的别的客户端的消息内容、发送者编号、IP 地址、端口等）。
- 服务端编写步骤（需要采用多线程模式）
 - a) 运行初始化，调用 `socket()`，向操作系统申请 `socket` 句柄
 - b) 调用 `bind()`，绑定监听端口（请使用学号的后 4 位作为服务器的监听端口），接着调用 `listen()`，设置连接等待队列长度
 - c) 主线程循环调用 `accept()`，直到返回一个有效的 `socket` 句柄，在客户端列表中增加一个新客户端的项目，并记录下该客户端句柄和连接状态、端口。然后创建一个子线程后继续调用 `accept()`。该子线程的主要步骤是（刚获得的句柄要传递给子线程，子线程内部要使用该句柄发送和接收数据）：
 - ✧ 调用 `send()`，发送一个 `hello` 消息给客户端（可选）
 - ✧ 循环调用 `receive()`，如果收到了一个完整的请求数据包，根据请求类型做相应的动作：
 1. 请求类型为获取时间：调用 `time()`获取本地时间，并调用 `send()`发给客户端
 2. 请求类型为获取名字：调用 `GetComputerName` 获取本机名，调用 `send()`发给客户端
 3. 请求类型为获取客户端列表：读取客户端列表数据，将编号、IP 地址、端口等数据通过调用 `send()`发给客户端
 4. 请求类型为发送消息：根据编号读取客户端列表数据，将要转发的消息组

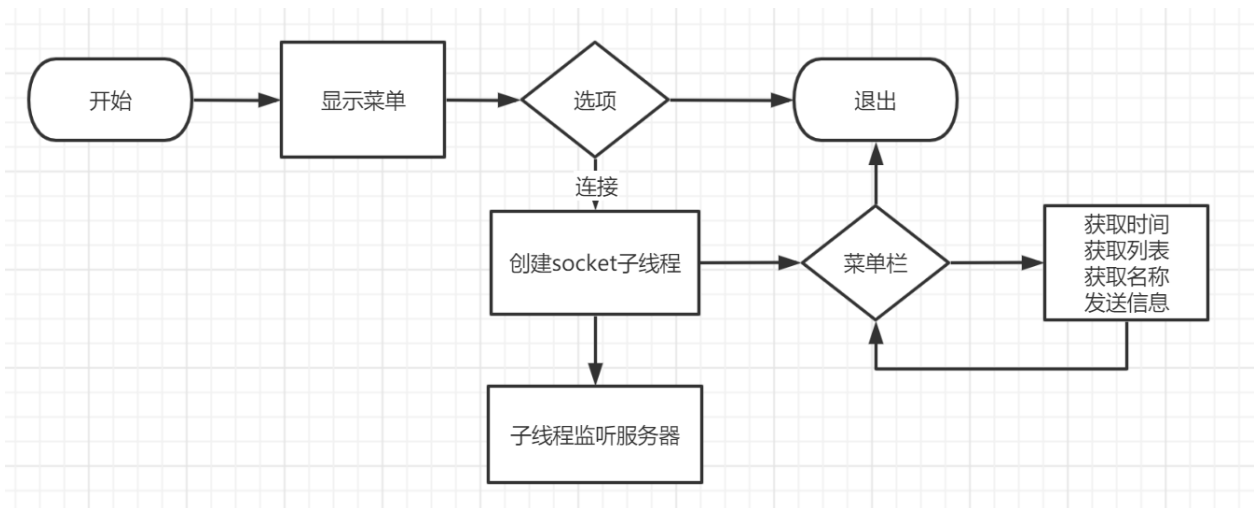
装通过调用 send()发给接收客户端（使用接收客户端的 socket 句柄）。

- 编程结束后，双方程序运行，检查是否实现功能要求，如果有问题，查找原因，并修改，直至满足功能要求
- 使用多个客户端同时连接服务端，检查并发性

五、实验数据记录和处理

请将以下内容和本实验报告一起打包成一个压缩文件上传：

- 源代码：客户端和服务端的代码分别在一个目录
- 可执行文件：可运行的.exe 文件或 Linux 可执行文件，客户端和服务端各一个
- 客户端和服务端框架图（用流程图表示）



- 客户端初始运行后显示的菜单选项

```
ubuntu@ubuntu: ~/Documents/network/socket/3
ubuntu@ubuntu:~/Documents/network/socket/3$ ./client
-----START-----
Please input the command:
[1] Connect [IP] [port]
[2] Close
[3] GetTime
[4] GetServername
[5] GetClients
[6] Send [IP] [port] [content]
[7] Exit
[8] Help
-----END-----
>
```

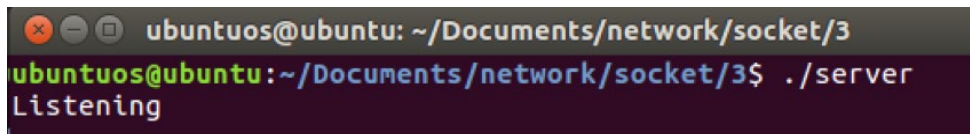
- 客户端的接收数据子线程循环关键代码截图（描述总体，省略细节部分）

```

88     while (true)
89     {
90         cout<<"> ";
91         string line;
92         getline(cin, line);
93         regex whitespace("\\s+");
94         vector<string> words(sregex_token_iterator(line.begin(), line.end(), whitespace, -1),
95                             sregex_token_iterator());
96         if (words[0] == "")
97             continue;
98 >     else if (words[0] == "Connect") ...
109 >     else if (words[0] == "Close") ...
125 >     else if (words[0] == "GetTheTime") ...
135 >     else if (words[0] == "GetTheName") ...
143 >     else if (words[0] == "GetTheClients") ...
176 >     else if (words[0] == "Send") ...
201 >     else if (words[0] == "Receive") ...
208 >     else if (words[0] == "Exit") ...
220 >     else if (words[0] == "Help") ...
224 >     else ...
228 }

```

- 服务器初始运行后显示的界面



A terminal window screenshot showing the command prompt for a user named 'ubuntuos' on a machine named 'ubuntu'. The current directory is '~/Documents/network/socket/3'. The user has entered the command './server', and the output is 'Listening'.

```

ubuntuos@ubuntu: ~/Documents/network/socket/3
ubuntuos@ubuntu:~/Documents/network/socket/3$ ./server
Listening

```

- 服务器的客户端处理子线程循环关键代码截图（描述总体，省略细节部分）

```

while (true)
{
    recv(cfd, buffer_recv, BUFFER_MAX, 0);
    memset(buffer_send, 0, BUFFER_MAX);
    mt.lock();
    switch (buffer_recv[0])
    {
        case 0:
        { ...
        case 1:
        { ...
        case 2:
        {
            cout<<cfd<<"GetTheName: "<<'\\n';
            buffer_send[0] = 12;
            gethostname(buffer_send + strlen(buffer_send), sizeof(buffer_send) - sizeof(char));
            send(cfd, buffer_send, strlen(buffer_send), 0);
            break;
        }
        case 3:
        { ...
        case 4:
        { ...
        case 5:
            break;
        }
    }
    memset(buffer_recv, 0, BUFFER_MAX);
    mt.unlock();
}

```

- 客户端选择连接功能时，客户端和服务端显示内容截图。

```

ubuntu@ubuntu: ~/Documents/network/socket/3
ubuntu@ubuntu:~/Documents/network/socket/3$ ./client
-----START-----
Please input the command:
[1] Connect [IP] [port]
[2] Close
[3] GetTheTime
[4] GetTheName
[5] GetTheClients
[6] Send [IP] [port] [content]
[7] Exit
[8] Help
-----END-----
> Connect 127.0.0.1 3770
>
hello
> >

```

```

ubuntu@ubuntu:~/Documents/network/socket/3$ ./server
Listening
127.0.0.1:39210 connected.

```

- 客户端选择获取时间功能时，客户端和服务端显示内容截图。

```
ubuntu@ubuntu:~/Documents/network/socket/3$ ./client
-----START-----
Please input the command:
[1] Connect [IP] [port]
[2] Close
[3] GetTheTime
[4] GetTheName
[5] GetTheClients
[6] Send [IP] [port] [content]
[7] Exit
[8] Help
-----END-----
> Connect 127.0.0.1 3770
>
hello
> >
> GetTheTime
  Server time: Mon Dec 21 05:47:22 2020
>
```

```
ubuntu@ubuntu:~/Documents/network/socket/3$ ./server
Listening
127.0.0.1:39210 connected.
GetTheTime:
```

- 客户端选择获取名字功能时，客户端和服务端显示内容截图。


```
ubuntuos@ubuntu: ~/Documents/network/socket/3
> GetTheTime
Server time: Mon Dec 21 05:44:57 2020
> ^C
ubuntuos@ubuntu:~/Documents/network/socket/3$ ./client
-----START-----
Please input the command:
[1] Connect [IP] [port]
[2] Close
[3] GetTheTime
[4] GetTheName
[5] GetTheClients
[6] Send [IP] [port] [content]
[7] Exit
[8] Help
-----END-----
> Connect 127.0.0.1 3770
>
hello
> >
> GetTheTime
Server time: Mon Dec 21 05:47:22 2020
> GetTheName
Server host name: ubuntu
>
```

```
ubuntuos@ubuntu:~/Documents/network/socket/3$ ./server
Listening
127.0.0.1:39210 connected.
GetTheTime:
GetTheName:
█
```

相关的服务器的处理代码片段:

```
135  else if (words[0] == "GetTheName")
136  {
137      char buffer = 2;
138      send(sockfd, &buffer, sizeof(buffer), 0);
139      Message namemsg;
140      msgrcv(msqid, &namemsg, MAXBUFFER, 12, 0);
141      cout<<"Server host name: "<<namemsg.text<<'\n'<<NORMAL;
142  }

128  case 2:
129  {
130      cout<<cfd<<"GetTheName: "<<'\n';
131      buffer_send[0] = 12;
132      gethostname(buffer_send + strlen(buffer_send), sizeof(buffer_send) - sizeof(char));
133      send(cfd, buffer_send, strlen(buffer_send), 0);
134      break;
135  }
```


- 客户端选择获取客户端列表功能时，客户端和服务端显示内容截图。

```
ubuntu@ubuntu:~/Documents/network/socket/3$ ./client
-----START-----
Please input the command:
[1] Connect [IP] [port]
[2] Close
[3] GetTheTime
[4] GetTheName
[5] GetTheClients
[6] Send [IP] [port] [content]
[7] Exit
[8] Help
-----END-----
> Connect 127.0.0.1 3770
>
hello
> >
>
> GetTheClients
0 127.0.0.1 39216
```

```
ubuntu@ubuntu:~/Documents/network/socket/3$ ./server
Listening
127.0.0.1:39210 connected.
GetTheTime:
GetTheName:
GetTheClients:
```

相关的服务器的处理代码片段：

```

143     else if (words[0] == "GetTheClients")
144     {
145         char buffer = 3;
146         send(sockfd, &buffer, sizeof(buffer), 0);
147         Message peermsg;
148         msgrcv(msqid, &peermsg, MAXBUFFER, 13, 0);
149
150         char* ptr = peermsg.text;
151         int count = 0, flag = 1;
152         while (*ptr)
153         {
154             if ('^' == *ptr)
155                 cout<<' ';
156             else if ('$' == *ptr)
157             {
158                 cout<<'\\n';
159                 flag = 1;
160             }
161             else
162             {
163                 if (flag)
164                 {
165                     cout<<count++<<' ';
166                     flag = 0;
167                 }
168                 cout<<(*ptr);
169             }
170             ptr++;
171         }
172         cout<<NORMAL;
173     }

```

```

136         case 3:
137         {
138             cout<<cfd<<"GetTheClients: "<<'\\n';
139             buffer_send[0] = 13;
140             for (auto& it: clientList)
141             {
142                 sprintf(buffer_send + strlen(buffer_send), "%s", it.second.first.c_str());
143                 sprintf(buffer_send + strlen(buffer_send), "^");
144                 sprintf(buffer_send + strlen(buffer_send), "%d", it.second.second);
145                 sprintf(buffer_send + strlen(buffer_send), "$");
146             }
147             send(cfd, buffer_send, strlen(buffer_send), 0);
148             break;
149         }

```

- 客户端选择发送消息功能时，两个客户端和服务端（如果有的话）显示内容截图。

发送消息的客户端：

```

ubuntu@ubuntu:~/Documents/network/socket/3$ ./client
-----START-----
Please input the command:
[1] Connect [IP] [port]
[2] Close
[3] GetTheTime
[4] GetTheName
[5] GetTheClients
[6] Send [IP] [port] [content]
[7] Exit
[8] Help
-----END-----
0.1 3770
System Settings
hello
> >
>
> GetTheClients
0 127.0.0.1 39216
> Send 127.0.0.1 39216 hello world
hello world
Success.

```

服务器端：

```

ubuntu@ubuntu:~/Documents/network/socket/3$ ./server
Listening
127.0.0.1:39216 connected.
GetTheTime:
GetTheName:
GetTheClients:
Send message to 127.0.0.1:39216

```

相关的服务器的处理代码片段：

```

150     case 4:
151     {
152         string msg(buffer_recv + 1);
153         size_t pos0 = msg.find("^"), pos1 = msg.find("$");
154         string ip_addr = msg.substr(0, pos0);
155         int port = atoi(msg.substr((pos0 + 1), pos1 - pos0 - 1).c_str());
156         string content = msg.substr(pos1 + 1);
157         int sock = -1;
158         for (auto it = clientList.begin(); it != clientList.end(); ++it)
159         {
160             if (it->second.first == ip_addr && it->second.second == port)
161             {
162                 sock = it->first;
163                 break;
164             }
165         }
166         cout<<cfd<<" Send message to "<<ip_addr<<":"<<port<<'\n';
167         buffer_send[0] = 14;
168         if (-1 == sock)
169             sprintf(buffer_send + 1, "Fail.\n");
170         else
171         {
172             SendMsg(sock, content);
173             sprintf(buffer_send + 1, "Success.\n");
174         }
175         send(cfd, buffer_send, strlen(buffer_send), 0);
176         break;
177     }
178     case 5:
179         break;
180 }

```

相关的客户端（发送和接收消息）处理代码片段：

```

174  ✓ else if (words[0] == "Send")
175      {
176          char buffer[MAXBUFFER] = {0};
177          buffer[0] = 4;
178          sprintf(buffer + strlen(buffer), "%s", words[1].c_str());
179          sprintf(buffer + strlen(buffer), "^");
180          sprintf(buffer + strlen(buffer), "%s", words[2].c_str());
181          sprintf(buffer + strlen(buffer), "$");
182  ✓      for (int i = 3; i < words.size(); ++i)
183          {
184              sprintf(buffer + strlen(buffer), "%s", words[i].c_str());
185  ✓              if (i != words.size())
186                  {
187                      sprintf(buffer + strlen(buffer), " ");
188                  }
189  ✓              else
190                  {
191                      sprintf(buffer + strlen(buffer), "\n");
192                  }
193              }
194          send(sockfd, buffer, MAXBUFFER, 0);
195          Message statusmsg;
196          msgrcv(msqid, &statusmsg, MAXBUFFER, 14, 0);
197          cout<<statusmsg.text<<NORMAL;
198      }

199  ✓ else if (words[0] == "Receive")
200      {
201          char buffer[MAXBUFFER] = {0};
202          buffer[0] = 5;
203          sprintf(buffer + strlen(buffer), "receive");
204          send(sockfd, buffer, MAXBUFFER, 0);
205      }

```

六、 实验结果与分析

- 客户端是否需要调用 bind 操作？它的源端口是如何产生的？每一次调用 connect 时客户端的端口是否都保持不变？

Ans:

不需要调用 bind 操作。当多个客户端在本机同时运行时会出现端口被占用的问题，考虑到本次实验不需要特定端口进行客户端连接，所以不需要调用 bind。不调用 bind，则由操作系统内核分配一个端口号。每一次调用 connect，内核会随机分配一个端口连接，所以不会都保持不变。

- 假设在服务端调用 `listen` 和调用 `accept` 之间设了一个调试断点，暂停在此断点时，此时客户端调用 `connect` 后是否马上能连接成功？

Ans:

不能。`listen` 的作用是服务器开启监听模式，而 `accept` 的作用是服务器等待客户端连接，一般都是阻塞态。如果在 `listen` 和 `accept` 之间设置了一个调试断点，那么当客户端调用 `connect` 之后虽然服务端可以通过 `listen` 函数接受到客户端的信息，但是由于没有执行 `accept` 就无法得到客户端的 `file description`，因此连接不能成功。

- 服务器在同一个端口接收多个客户端的数据，如何能区分数据包是属于哪个客户端的？

Ans:

本次实验采用了多线程的方式来实现多个客户端的同步，也就是说每次有一个新的客户端连接服务端的时候都产生了新的线程。而每个客户端的 `ip` 地址以及 `port` 地址是唯一的，通过一系列数组保存该信息。每次当服务器在同一个接口接收多个客户端的数据时，通过 `socket` 的 `ip` 和 `port` 信息可以区分数据包是哪个客户端的。

- 客户端主动断开连接后，当时的 TCP 连接状态是什么？这个状态保持了多久？（可以使用 `netstat -an` 查看）

Ans:

断开前 `Server` 和 `Client` 处于 `ESTABLISHED` 状态，客户端断开后服务端变为 `TIME_WAIT`，持续大约 2 倍 `MSL` 时长，随后变为 `CLOSED`

- 客户端断网后异常退出，服务器的 TCP 连接状态有什么变化吗？服务器该如何检测连接是否继续有效？

Ans:

断开前 `Server` 和 `Client` 处于 `ESTABLISHED` 状态，断网后 `Server` 变为 `CLOSE_WAIT`，`Client` 变为 `FIN_WAIT2` 随后消失，服务器可通过 `select()` 函数进行检测，`retval`

= select(maxfd+1, &rfd, NULL, NULL, &tv);返回值为-1时表示连接无效

七、 讨论、心得

服务器编写：

开始的难点在于多线程的编程，因为没有接触过，但是了解之后就没有多大问题，不过存在一个缺点就是不知道如何正确处理子线程的资源，尤其是在服务端非正常退出之后，紧接着再次启动出现 bind error 的状况。

再者是对数据包的定义，由于没有和客户端连接好造成了一些困难，后来还是采取最简单的字符串传递方式，但是仍存在一些缺陷。

对于实验安排的最主要的建议是，对子线程的资源管理释放做一些讲解。

客户端编写：

难点在于多线程的编写。一开始使用的是在主线程中直接 recv，只有 sendmessage 时另外开了子线程，因为需要一直 recv 以确保一旦有人发送消息立刻能收到。但是发现 2 个线程都有 recv 会导致线程崩溃。