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：

学　　院：　计算机学院

系：　计算机科学与技术

专　　业：　计算机科学与技术（图灵班）

学　　号：

指导教师：　高艺

2023 年　1 月　4 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：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿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）。

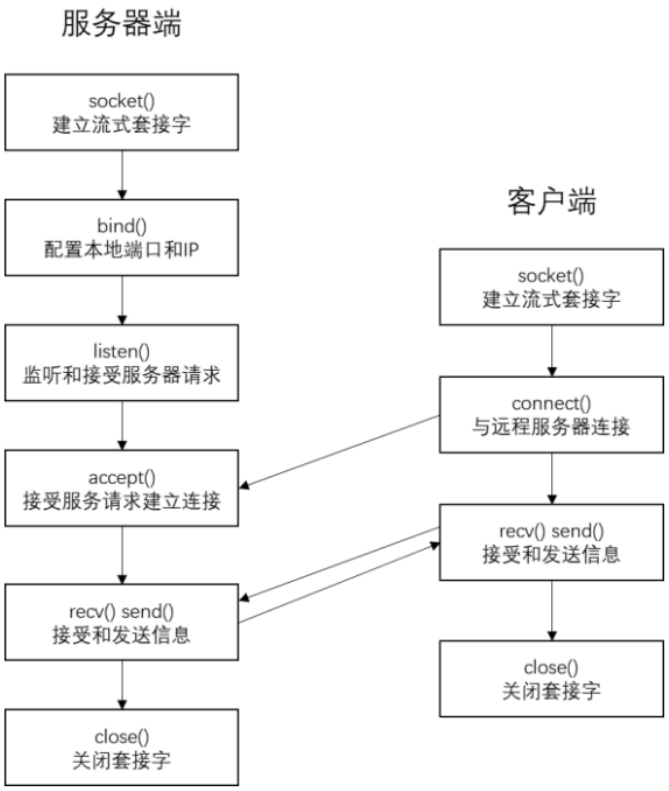
- 编程结束后，双方程序运行，检查是否实现功能要求，如果有问题，查找原因，并修改，直至满足功能要求
- 使用多个客户端同时连接服务端，检查并发性

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**

- 客户端和服务器端框架图（用流程图表示）



- 客户端初始运行后显示的菜单选项

```
> ./client
----------------------------------------Client----------------------------------------
Usage:
1. connect [IP] [PORT]: connect to server
2. exit: shutdown the connection
3. time: show server time
4. name: show client name
5. list: show the clients connected to the server
6. send [ID]: start to send [MSG] to client [ID], press Enter to stop sending message
--------------------------------------------------------------------------------------
>
```

- 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）

```c
static int IsFromClient(char *buf) {
    char buf2[BUF_SIZE];
    strncpy(buf2, buf, BUF_SIZE);
    char *p = strchr(buf2, ' ');
    if (p != NULL) {
        *p = '\0';
        if (strcmp(buf2, "[HOST]") != 0) { // client
            return 1;
        }
    }
    return 0;
}

void *Listen(void *arg) {
    int recv_num;
    char buf_listen[BUF_SIZE];

    while (1) {
        recv_num = read(socket_fd, buf_listen, BUF_SIZE);
        if (IsFromClient(buf_listen)) {
            printf("%s", buf_listen);
        } else { // server
            pthread_mutex_lock(&mtx);
            strncpy(buf, buf_listen, BUF_SIZE);
            pthread_cond_signal(&cond);
            pthread_mutex_unlock(&mtx);
        }
    }
    return NULL;
}

static int Recv() {
    pthread_mutex_lock(&mtx);
    pthread_cond_wait(&cond, &mtx);
    pthread_mutex_unlock(&mtx);
}
```

这里有三个函数：IsFromClient 表示检查某段信息是否从别的 Client 转发过来的，从 Server 返回的消息会以[HOST]开头，其他 Client 转发过来的消息会以[Client <ID>]开头，所以这里就是检查是否是以[HOST]开头；Listen 是客户端子线程函数，里面会监听从服务器发来的消息，如果是服务器转发别的客户端的消息就直接打印，否则说明是服务器发来的回答，客户端此时一定处于 block 状态，所以用条件变量唤醒主线程；Recv 是主线程向服务器发完请求之后用来 block 的函数，这里会 wait 条件变量，直到子线程获得客户端的回应才会继续执行。

- 服务器初始运行后显示的界面

```
❯ ./server 2339
[OK] Socket Success! Socket: 3
[OK] Bind Succcess! IP: 0.0.0.0, Port: 2339
[OK] Listen Success! Waiting for Connection!
```

这里选择的端口是我的学号后四位 2339。

- 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）

```c
while (1) {
    pthread_t thread_id;

    if ((accept_fd = accept(socket_fd, (struct sockaddr *)&client_addr, &client_size)) == -1) {
        printf("[ERROR] Accept Error!\n");
        continue;
    } else {
        struct in_addr ip = client_addr.sin_addr;
        uint16_t port = ntohs(client_addr.sin_port);
        int id = AddClient(ip, port, accept_fd);
        printf("[OK] Client %d Connect! IP: %s, Port: %d\n", id, inet_ntoa(ip), port);
    }

    if(pthread_create(&thread_id, NULL, Handler, (void *)(&accept_fd)) == -1)
    {
        printf("[ERROR] Pthread Create Error!\n");
        continue;;
    }
}
```

```c
static void *Handler(void *accept_fd) {
    int client_fd = *(int*)accept_fd;
    char buf[BUF_SIZE];
    int recv_num = 0;
    int send_num = 0;
    char *args[ARG_NUM] = {0};
    int parse_res = 0;

    while (1) {
        memset(buf, 0, BUF_SIZE);
        recv_num = read(client_fd, buf, BUF_SIZE);
        if (recv_num < 0) {
            printf("[ERROR] Read Error!\n");
            break;
        } else if (recv_num == 0) {
            printf("[INFO] Read Nothing! Client May Close!\n");
            break;
        }
        parse_res = ParseInput(buf, args);
        if (parse_res == 1) {
            printf("[ERROR] Parse Error!\n");
            continue;
        }

        if (strcmp(args[0], "time") == 0) {…
        } else if (strcmp(args[0], "name") == 0) {…
        } else if (strcmp(args[0], "list") == 0) {…
        } else if (strcmp(args[0], "send") == 0) {…
        } else {
            printf("[ERROR] Invalid Instruction!\n");
        }
    }

    CloseClient(client_fd);
    return NULL;
}
```

首先服务器端会用一个循环，不断接受 Client 的连接，每个连接都会使用 Handler 函数来处理。Handler 函数中会对 Client 的输入 Parse，然后根据 Client 不同的请求调用不同的功能。

- 客户端选择连接功能时，客户端和服务端显示内容截图。

客户端：

```
> connect 127.0.0.1 2339
[OK] Connect Success!
>
```

服务端：

```
[OK] Client 1 Connect! IP: 127.0.0.1, Port: 41554
```

- 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。

客户端：

```
> time
[HOST] Wed Jan  4 19:30:11 2023
```

服务端没有输出。

- 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。

客户端：

```
> name
[HOST] LAPTOP-48IILPU2
```

服务端没有输出。

相关的服务器的处理代码片段：

```
} else if (strcmp(args[0], "name") == 0) {
    char hostname[30];
    gethostname(hostname, sizeof(hostname));
    sprintf(buf, "[HOST] %s\n", hostname);
    write(client_fd, buf, BUF_SIZE);
```

- 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。

客户端：

```
> list
[HOST] Client Num: 2
        [ID] 01 [IP] 127.0.0.1 [PORT] 41554
        [ID] 02 [IP] 127.0.0.1 [PORT] 38744
```

服务端没有输出。

相关的服务器的处理代码片段：

```
} else if (strcmp(args[0], "list") == 0) {
    char *p = buf;
    sprintf(p, "[HOST] Client Num: %d\n", cur_client_num);
    p += strlen(p);
    for (int i = 0; i < MAX_QUEUE; i++) {
        if (clients[i].id != 0) {
            sprintf(p, "\t[ID] %02d [IP] %s [PORT] %d\n", clients[i].id, inet_ntoa(clients[i].ip), clients[i].port);
            p += strlen(p);
        }
    }
    write(client_fd, buf, BUF_SIZE);
```

服务器端维护了一个 clients 的列表，其中如果 id 是 0 表示 invalid，所以这里就遍历整个列表，如果 id 非 0 那么就加入到输出中，最后 write 到 client_fd 中。

- 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：我这里发送了 3 条信息，并且最后输入回车结束发送。

```
> send 2
>> hello
>> my name is 0xWe11es.eth
>> nice to meet you!
>>
>
```

服务器端（可选）：没有输出

接收消息的客户端：

```
> [CLIENT 01] hello
[CLIENT 01] my name is 0xWe11es.eth
[CLIENT 01] nice to meet you!
```

相关的服务器的处理代码片段：

```c
} else if (strcmp(args[0], "send") == 0) {
    if (args[1] == NULL) {
        printf("[ERROR] Lack Args!\n");
        continue;
    }
    int cur_id = FindIdByFd(client_fd);
    int target_id = atoi(args[1]);
    if (target_id == 0 || target_id == cur_id || !ExistId(target_id)) {
        sprintf(buf, "[HOST] Invalid Id!\n");
        write(client_fd, buf, BUF_SIZE);
        continue;
    } else {
        sprintf(buf, "[HOST] Communication Established!\n");
        write(client_fd, buf, BUF_SIZE);
    }

    int target_fd = FindFdById(target_id);
    char buf2[BUF_SIZE];
    while (1) {
        recv_num = read(client_fd, buf, BUF_SIZE);
        if (recv_num < 0) {
            printf("[ERROR] Read Error!\n");
            break;
        } else if (recv_num == 0) {
            printf("[INFO] Read Nothing! Client May Close!\n");
            break;
        }
        if (strcmp(buf, "\n") == 0) {
            break;
        } else {
            sprintf(buf2, "[CLIENT %02d] %s", cur_id, buf);
            write(target_fd, buf2, BUF_SIZE);
        }
    }
```

首先通过服务器维护的 clients 列表获得发送方的 id（cur_id）、发送方的 fd（client_fd）、接收方的 id（target_id）以及接收方的 fd（target_fd）；然后判断接收方是否合法，检查其 id 是否存在是否和发送方重合，如果失败就发送给发送方[HOST] Invalid Id!\n，如果成功就进入循环，不断从发送方处 read 消息并 write 给接收方。

相关的客户端（发送和接收消息）处理代码片段：

接收消息的逻辑在上面。

下面是发送的逻辑，首先是发送 send <ID>，然后接收服务器发来的反馈，如果成功建立连接，就可以开始发送消息，当用户单输入回车键的时候停止发送。

```
} else if (strcmp(args[0], "send") == 0) {
    if (args[1] == NULL) {
        printf("[ERROR] Lack Args!\n");
        continue;
    }
    // send instruction
    sprintf(buf, "%s %s", args[0], args[1]);
    write(socket_fd, buf, BUF_SIZE);
    // check validity of communition
    Recv();
    if (strcmp(buf, "[HOST] Invalid Id!\n") == 0) {
        printf("%s", buf);
        continue;
    }

    while (1) {
        printf(">> ");
        fgets(buf, BUF_SIZE, stdin);
        write(socket_fd, buf, BUF_SIZE);
        if (strcmp(buf, "\n") == 0) break;
    }
```

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？

答：不需要。客户端源端口是系统随机分配的，每次调用 connect 时客户端的端口会随机选一个。

- 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？

答：可以连接成功，但是会有一定连接数量限制。accept 的作用是将已经完成握手的 socket 从握手完成队列中取出，如果不调用的话那么完成握手的 socket 达到上限就不能再建立新的连接。

- 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？

答：和服务器连接的客户端 IP 地址和端口不同，服务器可以区分。

- 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）

```
❯ netstat -an | rg 2339
tcp        0      0 0.0.0.0:2339            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:2339          127.0.0.1:48700         ESTABLISHED
tcp        0      0 127.0.0.1:48700         127.0.0.1:2339          ESTABLISHED
```

```
❯ netstat -an | rg 2339
tcp        0      0 0.0.0.0:2339            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:48700         127.0.0.1:2339          TIME_WAIT
```

```
❯ netstat -an | rg 2339
tcp        0      0 0.0.0.0:2339            0.0.0.0:*               LISTEN
```

答：当时的 TCP 连接状态是 TIME_WAIT。大约保持了 1 分钟左右。

- 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？

答：TCP 连接状态不会发生变化，依旧是 ESTABLISED 状态。服务器向客户端发送的数据将得不到回应，此时触发超时重传机制，如果在没有达到重传次数上限前客户端重新联网并回应那就相当于无事发生，否则 TCP 连接就断开了。

七、 讨论、心得

本实验主要遇到的困难就是要实现其他客户端发来的消息立刻显示在本客户端上，即客户端的多线程。我之前对 pthread 没有太多使用经验，尤其是条件变量的使用我比较模糊，最后再和队友的交流中较好地完成了实验。