网络编程实验报告

13061210 何涛
130641 班

2016 年 5 月 31 日

目　　录

一、 引言

1.1 编写目的

1. 使用 POSIX Socket API 构建网络通信程序。
2. 使用 POSIX Thread 实现支持并发连接的 TCP 服务端程序。

1.2 定义

程序整体分为服务端和客户端，支持多客户数据流 Socket 通信，同时添加了 SSL 支持来保障通信过程中数据在传输层的安全和完整。服务端程序创建 SOCK_STREAM 获得套接字描述符，使用 bind/listen 函数监听指定的端口。客户端程序创建 SOCK_STREAM 获得套接字描述符，使用 connect 建立到服务端的连接。对 SSL 的支持通过调用 OpenSSL 的接口完成。

1.3 参考资料

1. Unix 网络编程卷一：套接字联网 API
2. OpenSSL Documentation

二、 软件需求说明

2.1 任务概述

1. 服务器可以接受来自任意 IP 的客户端的 TCP 连接请求。
2. 服务器支持同时与多个客户端建立 TCP 连接并进行通信。
3. 服务器接收客户端发出的文本数据，将数据的内容返回给客户端并在标准输出中显示客户端的基本信息和消息内容。
4. 客户端从标准输入接受用户输入，并将用户输入的文本通过 TCP 连接发送给服务端。
5. 当客户端输入的消息内容为 bye 时，服务端收到消息后断开与客户端的连接。
6. 当客户端无法正常接收输入（输入 EOF）时，客户端会主动与服务端断开 TCP 连接。

2.2 运行环境规定

1. Linux

三、 详细设计说明

3.1 程序系统的结构

程序采用 C/S 结构，服务端运行后根据命令行参数开始监听指定端口。客户端运行后根据命令行参数与指定的服务器建立连接，当服务器收到的消息为 bye 时断开与客户端的连接。服务端对于每一个 TCP 连接采用一个单独的线程进行处理，可以支持同时与多个客户端进行通信。

3.2 服务器监听端口程序设计说明

服务端程序从命令行参数中获得端口参数，使用 IPv4 套接字地址 sockaddr_in，设置好端口和可接受的客户端 IP，通过 bind 函数将地址结构体和套接字描述符绑定，调用 listen 函数设置最大连接数同时开始监听端口。然后开始事件循环，等待客户端的连接请求。每接受一个客户端的连接请求，开启一个线程进行处理。

3.3 客户端服务程序设计说明

客户端程序从命令行参数中获得端口参数，使用 IPv4 套接字地址 `sockaddr_in`，设置好端口和要连接的服务端的 IP 地址，调用 `connect` 函数使用创建好的客户端套接字描述符与指定的套接字地址建立 TCP 连接，然后开始事件循环，不断地从标准输入中获取用户输入，并将消息发送给服务端，然后等待服务端的响应，读取响应的内容，并在标准输入中显示服务端的基本信息和响应内容。

3.4 SSL 通信设计说明

SSL 可靠通信的实现需要使用 OpenSSL 库，并使用 `openssl` 命令在服务端生成私钥和自签名证书。服务端首先初始化 SSL 上下文，加载证书和私钥并检查是否加载成功。对于每一个来自客户端的连接请求，创建一个的新的 SSL 连接，设置对应客户端的套接字描述符，之后使用 `SSL_read` 和 `SSL_write` 与客户端进行可靠通信。

SSL 客户端首先初始化 SSL 上下文，使用 `SSL_connect` 与服务端完成握手，建立 SSL 连接，设置套接字描述符，随后使用 `SSL_write` 和 `SSL_read` 与服务端进行可靠通信。

3.5 SSL 握手过程

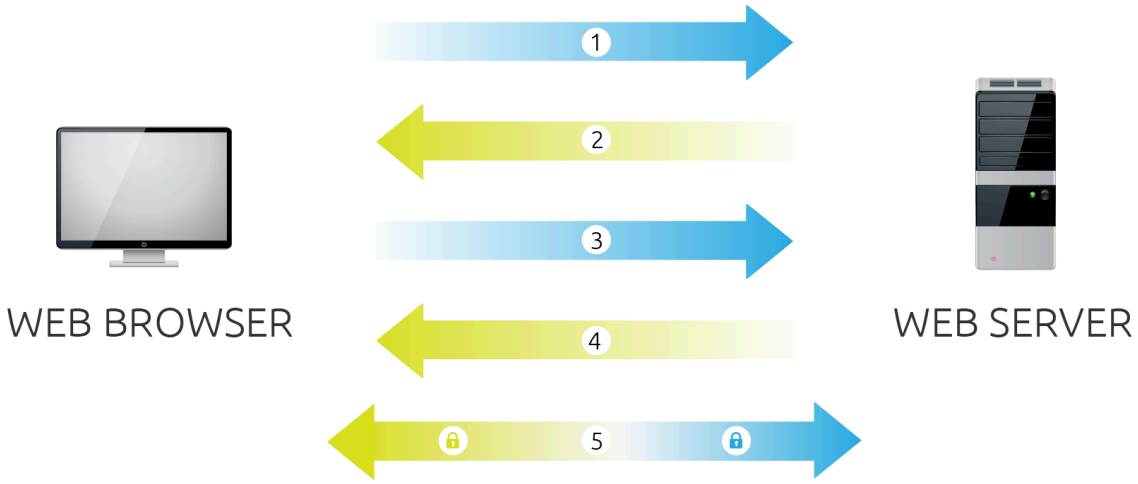

Figure 1: SSL 握手过程示意图

1. 客户端发送 Client Hello 报文给服务器，告诉服务器客户端支持的算法列表以及一个用于生成密钥的随机数。
2. 服务器发送 Server Hello 报文，包含选择的算法、用于生成密钥的随机数、服务端的证书，以及 Server Hello Done 报文，告诉客户端继续进行下一步握手。
3. 客户端验证服务器证书，生成一个新的随机数，使用服务器证书中的公钥加密，发送给服务端，并发送 Finish 消息，结束握手。
4. 服务器生成 Session Key，发送给客户端，完成握手。
5. 客户端和服务端使用 Session Key 进行通信。

四、 编码实现

4.1 服务端编码实现

具体编码实现上，服务端用到了 socket 函数创建套接字描述符，使用 bind 和 listen 函数建立端口监听，使用 accept 接受来自客户端的连接请求，使用 pthread 相关函数来创建线程处理与客户端的连接，之后使用 recv 和 send 函数进行数据的接收和发送。此外还是用了 htons、htonl 和 inet_ntop 来实现本地编码和大小端标准与网络编码和大小端标准之间的转换。具体的代码实现为（仅部分代码，用于说明流程）：

```
void *handle(client_fd, client) {
    while ((read_size = recv(client_fd, buf, BUFFER, 0)) > 0) {
        send(client_fd, buf, read_size, 0);
    }
    close(client_fd);
    pthread_exit(NULL);
}

void build_server(uint16_t port) {
    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    server.sin_family = AF_INET;
    server.sin_port = htons(port);
    server.sin_addr.s_addr = htonl(INADDR_ANY);

    bind(server_fd, server, sizeof(server));
    listen(server_fd, 1024);
    while (1) {
        client_fd = accept(server_fd, client, client_size);
        pthread_create(&tid, NULL, handle, args);
        pthread_detach(tid);
    }
    close(server_fd);
}
```

4.2 客户端编码实现

客户端使用了 inet_pton 函数来进行字符串表示的 IP 地址和网络 IP 编码标准之间的转换，使用 connect 函数来建立连接。具体的代码实现为（仅部分代码，用于说明流程）：

```
void hanlde(server_fd, server) {
    while (scanf(”%s”, buf) != EOF) {
        send(server_fd, buf, strlen(buf)+1, 0);
        recv(server_fd, buf, BUFFER, 0);
    }
    close(server_fd);
}

void build_client(char const *target, uint16_t port) {
    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    server.sin_family = AF_INET;
    server.sin_port = htons(port);
    inet_pton(AF_INET, target, &server.sin_addr.s_addr);
```

```
    connect(server_fd, server, sizeof(server));
    handle(server_fd, server);
}
```

4.3 SSL 握手验证过程

SSL 需要调用 OpenSSL 中的函数，代码实现如下：

- 服务端

```
void handle {
    ssl = SSL_new(ctx);
    SSL_set_fd(ssl, client_fd);
    SSL_accept(ssl);

    while (SSL_read(...)) {
        SSL_write(...);
    }
    SSL_shutdown(ssl);
    SSL_free(ssl);
}

void build_server() {
    // Create SSL context.
    SSL_library_init();
    OpenSSL_add_all_algorithms();
    ctx = SSL_CTX_new(SSLv23_server_method());

    // Load certificate and private_key.
    SSL_CTX_use_certificate_file(ctx, cert, SSL_FILETYPE_PEM)
    SSL_CTX_use_PrivateKey_file(ctx, private_key, SSL_FILETYPE_PEM)

    // Loop forever ...

    SSL_CTX_free(ctx);
    close(server_fd);
}
```

- 客户端：

```
void handle {
    // Create SSL context.
    SSL_library_init();
    OpenSSL_add_all_algorithms();
    ctx = SSL_CTX_new(SSLv23_server_method());

    // Establish SSL connection.
    ssl = SSL_new(ctx);
    SSL_set_fd(ssl, client_fd);
    SSL_connect(ssl);

    while (scanf(...) != EOF) {
```

```
        SSL_write(...);
        SSL_read(...);
    }
    SSL_shutdown(ssl);
    SSL_free(ssl);
    SSL_CTX_free(ctx);
}

void build_client() {
    // Establish TCP connection.
    connect(server_fd, server, sizeof(server));
    handle(server_fd, server);
}
```

五、 系统测试

5.1 服务器同时与多个客户端进行通信

- 服务端

```
vagrant@trusty:~/sock$ ./echo-server.out 8888
Listening start on port 8888....
Establish connection with client 127.0.0.1:45848
Establish connection with client 127.0.0.1:45850
from 127.0.0.1:45848 message a
from 127.0.0.1:45850 message b
Establish connection with client 127.0.0.1:45852
```

- 客户端

  – 客户端 1

  ```
  vagrant@trusty:~/sock$ ./echo-client.out 127.0.0.1 8888
  Establish connection with server 127.0.0.1:8888
  message a
  from server 127.0.0.1:8888 message a
  ```

  – 客户端 2

  ```
  vagrant@trusty:~/sock$ ./echo-client.out 127.0.0.1 8888
  Establish connection with server 127.0.0.1:8888
  message b
  from server 127.0.0.1:8888 message b
  ```

  – 客户端 3

  ```
  vagrant@trusty:~/sock$ ./echo-client.out 127.0.0.1 8888
  Establish connection with server 127.0.0.1:8888
  ```

5.2 客户端发送断开连接命令

- 服务端

```
vagrant@trusty:~/sock$ ./echo-server.out 8888
Listening start on port 8888....
Establish connection with client 127.0.0.1:45848
Establish connection with client 127.0.0.1:45850
from 127.0.0.1:45848 message a
from 127.0.0.1:45850 message b
Establish connection with client 127.0.0.1:45852
Close connection with client 127.0.0.1:45852.
Close connection with client 127.0.0.1:45850.
Close connection with client 127.0.0.1:45848.
```

- 客户端

  – 客户端 1

  ```
  vagrant@trusty:~/sock$ ./echo-client.out 127.0.0.1 8888
  Establish connection with server 127.0.0.1:8888
  message a
  from server 127.0.0.1:8888 message a
  bye
  Client are going to stop connection
  Connection closed.
  ```

  – 客户端 2

  ```
  vagrant@trusty:~/sock$ ./echo-client.out 127.0.0.1 8888
  Establish connection with server 127.0.0.1:8888
  message b
  from server 127.0.0.1:8888 message b
  bye
  Client are going to stop connection
  Connection closed.
  vagrant@trusty:~/sock$
  ```

  – 客户端 3

  ```
  vagrant@trusty:~/sock$ ./echo-client.out 127.0.0.1 8888
  Establish connection with server 127.0.0.1:8888
  bye
  Client are going to stop connection
  Connection closed.
  ```

5.3 SSL 可靠通信

- 服务端

```
vagrant@trusty:~/sock$ sudo ./ssl-server.out 44330
Listening start on port 44330....
Establish connection with client 127.0.0.1:55806.
```

```
Establish SSL connection with AES256-GCM-SHA384 encryption.
No certificate from client.
from 127.0.0.1:55806 message a
```

- 客户端

```
vagrant@trusty:~/sock$ ./ssl-client.out 127.0.0.1 44330
Establish connection with server 127.0.0.1:44330.
Establish SSL connection with AES256-GCM-SHA384 encryption.
Server certificate subject: /C=CN/ST=Some-State/L=Some-Locality/O=Some-Ltd/OU=Some-Organization/
↪  CN=sighingnow/emailAddress=sighingnow@yandex.com.
Server certificate issuer: /C=CN/ST=Some-State/L=Some-Locality/O=Some-Ltd/OU=Some-Organization/CN=sighingnow/
↪  emailAddress=sighingnow@yandex.com.
message a
from SSL server 127.0.0.1:44330 message a
```

六、 总结

采取了多线程技术使得服务端可以支持并发连接，同时增加了对 SSL 的支持保障了通信过程中的数据的安全和可靠。但此外还有不少可以改进的余地，比如使用 `select/epoll/kqueue` 来支持 IO 多路复用和非阻塞连接，以及使用 Linux AIO 提供的异步 IO，提高服务器的吞吐量和性能。