

实验 5 流式套接字编程实践

目录

- 实验 5 流式套接字编程实践 1
 - 实验目的 1
 - 实验要求 1
 - 实验环境 2
 - 实验步骤 3
 - 1. TCP 网络编程的流程 3
 - 2. 编写服务器端实例 4
 - 3. 编写客户端实例 7
 - 实验测试 8
 - 实验体会 9

实验目的

掌握 Socket 编程的基础知识，了解服务器端和客户端编程的模型，掌握基本的 Socket 函数，能够实现 TCP 环境下客户端与服务器端交互的例子。

实验要求

- 1. 了解 TCP 网络编程的流程
- 2. 编写 TCP 网络交互客户端和服务端实例，实现如下功能
 - 客户端读取某个文件并将内容发送给服务端

- 服务器端读取客户端消息之后，将客户端消息、客户端消息长度写入本地文件，同时将这些信息返回给客户端
- 实现 Ctrl+C 信号的捕获并正常退出
- 服务器端接受时使用新的进程处理，实现并发处理

3. 了解服务器端的配置(地址、端口、侦听队列)

实验环境

实验环境采用 **vscode remote + wsl**，操作系统的版本为 **Ubuntu 20.04.2**

LTS，如下图所示

```
> lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.2 LTS
Release:        20.04
Codename:       focal
```

操作系统版本

内核版本如下图所示

```
> uname -a
Linux qiufeng 5.10.16.3-microsoft-standard-WSL2
```

内核版本

gcc 版本为 9.3.0

```
> gcc --version
gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

gcc 版本

实验步骤

1. TCP 网络编程的流程

TCP 网络编程的流程可以用下图来表示。

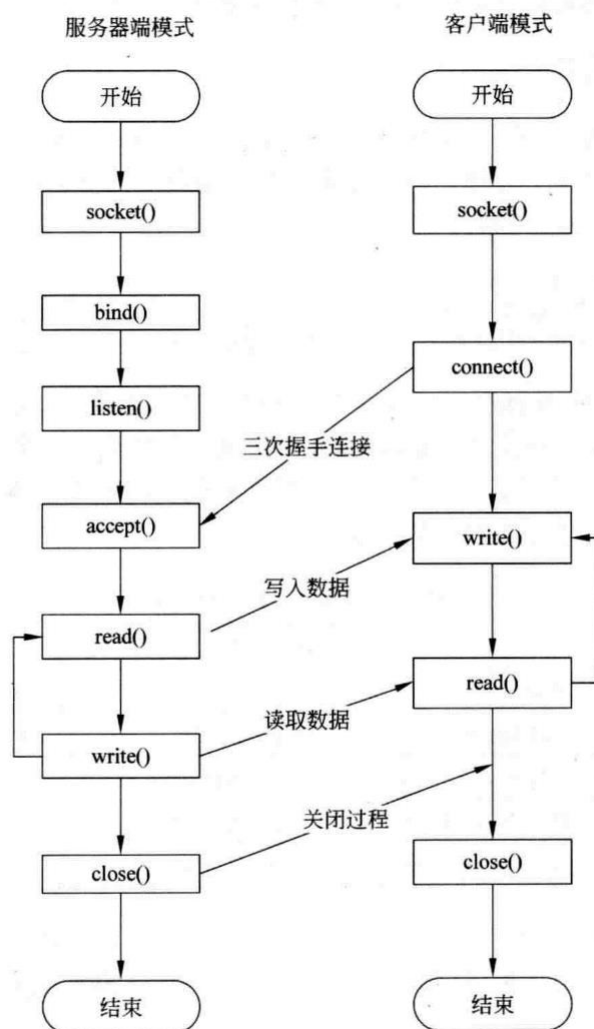


图 7.4 TCP 的服务器端模式

图 7.5 TCP 的客户端模式

TCP 网络编程流程

其中服务器端的流程主要包括：

- 套接字初始化
- 套接字与端口绑定
- 设置排队队列长度并监听

- 等待客户端连接
- 按照预定义的规则对套接字数据进行处理，并将结果发送给客户端
- 当处理完数据，要结束通信过程时，关闭套接字连接

客户端的流程主要包括

- 套接字初始化
- 连接服务器
- 读写网路数据并进行数据处理
- 关闭套接字

2. 编写服务器端实例

2.1 套接字初始化

使用 *socket* 函数并指定参数 *SOCK_STREAM* 创建流式套接字

```
/* 建立一个流式套接字
ss = socket(AF_INET, SOCK_STREAM, 0);
if (ss < 0)
    exit_with_failure("socket error");
```

创建流式套接字

2.2 端口绑定

设置服务器的协议族、地址和端口，并使用 *bind* 函数将地址结构绑定到流式套接字

```

/* 设置服务器地址
bzero(&server_addr, sizeof(server_addr));          /* 清零*/
server_addr.sin_family = AF_INET;                  /* 协议族*/
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);    /* 本地地址*/
server_addr.sin_port = htons(PORT);                 /* 服务器端口*/

/* 绑定地址结构到套接字描述符
err = bind(ss, (struct sockaddr *)&server_addr, sizeof(server_addr));

```

绑定端口

2.3 设置监听

使用 *listen* 函数设置监听，其中 **BACKLOG** 表示侦听队列长度

```

/* 设置侦听
err = listen(ss, BACKLOG);
if (err < 0)
    exit_with_failure("listen error");

```

设置监听

2.4 等待客户端连接

使用 *accept* 函数等待客户端连接

```

sc = accept(ss, (struct sockaddr *)&client_addr, &addrlen);

```

等待客户端连接

2.5 处理客户端连接

当服务器端接收到连接时使用 *fork* 创建子进程

```

/* 接收到新的连接时使用新的进程处理，实现并发处理
pid = fork();

```

fork 函数

在子进程中调用 *process_conn_sever* 函数处理客户端套接字，处理完成后

关闭服务器端套接字

```

if (pid == 0)
{
    /* 处理客户端套接字 */
    process_conn_server(sc);
    close(ss);
}

```

在子进程中处理套接字

其中函数 *process_conn_server* 的流程为

- 读取流式套接字中的数据至缓冲区
- 使用格式化字符串函数 *sprintf* 将字符串长度放置缓冲区末尾
- 将处理后的结果写入文件
- 将处理后的结果返回给客户端

```

void process_conn_server(int s)
{
    ssize_t size = 0;
    char buf[MAX_SIZE]; /* 数据的缓冲区 */
    char final[MAX_SIZE + 30];
    int fd = open(LOG_PATH, O_CREAT | O_WRONLY);
    if (fd == -1)
        exit_with_failure("open file failure");

    size = read(s, buf, MAX_SIZE); /* 从套接字中读取数据放到 */
    sprintf(final, "%s, length: %ld", buf, size);
    printf("receive message from client: %s\n", buf);
    if (size == 0)
    { /* 没有数据 */
        printf("nothing to read\n");
        return;
    }
    write(fd, final, strlen(final));
    close(fd);
    fprintf(stdout, "send message to client: %s\n", final);
    write(s, final, strlen(final));
}

```

服务器端处理函数

3. 编写客户端实例

3.1 套接字初始化

使用 *socket* 函数并指定 *SOCK_STREAM* 参数创建流式套接字

```
// *建立一个流式套接字
s = socket(AF_INET, SOCK_STREAM, 0);
if (s < 0)
    exit_with_failure("socket error");
```

创建流式套接字

3.2 连接服务器

使用 *connect* 函数连接服务器

```
/* 连接服务器
connect(s, (struct sockaddr *)&server_addr, sizeof(struct sockaddr));
```

连接服务器

3.3 读写网路数据并进行数据处理

使用 *process_conn_client* 对数据进行处理

```
/* 数据处理
process_conn_client(s);
```

数据处理

其中 *process_conn_client* 的主要逻辑如下

- 读取文件并存入缓冲区
- 发送数据给服务器
- 接受服务器返回的数据并打印

```

void process_conn_client(int s)
{
    ssize_t size = 0;
    char buf[MAX_SIZE] = { 0 };

    /* 读取指定文件 */
    size = read(fd, buf, MAX_SIZE);
    if (size > 0)
    {
        /* 发送给服务器 */
        write(s, buf, size);
        /* 打印发送的数据 */
        printf("send message to server: %s\n", buf);

        /* 从服务器接受数据 */
        size = read(s, buf, MAX_SIZE);
        /* 打印接收到的数据 */
        fprintf(stdout, "receive from server: %s\n", buf);
    }
}

```

客户端处理函数

实验测试

使用 gcc 分别编译客户端和服务端程序

```

> gcc server.c -o server
> gcc client.c -o client

```

编译程序

运行服务器端程序

```

> sudo ./server

```

运行服务器端程序

查看要传输的文件 *tmp.txt*, 内容为 *hello, qiufeng*

```

5 > tmp.txt
1  hello, qiufeng

```

要传输的内容

运行客户端程序, 成功读取到 *tmp.txt* 文件中的内容, 并成功接收到服务器

端的返回消息 *hello, qiufeng, length: 14*

```
> sudo ./client 127.0.0.1
send message to server: hello, qiufeng
receive from server: hello, qiufeng, length: 14
```

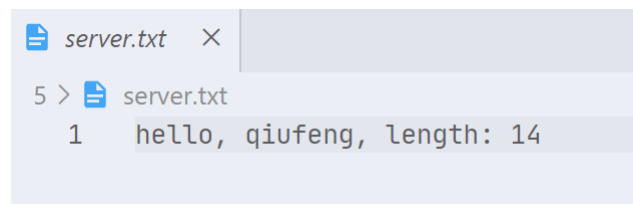
返回消息

服务器端打印消息如下

```
> sudo ./server
receive message from client: hello, qiufeng
send message to client: hello, qiufeng, length: 14
```

服务器端打印消息

服务器成功将消息内容写入 *server.txt* 文件中

A screenshot of a text editor window titled 'server.txt'. The editor shows a single line of text: 'hello, qiufeng, length: 14'. The line number '1' is visible on the left margin. The editor interface includes a tab at the top and a command line at the bottom.

server.txt

实验体会

本次实验使用了**流式套接字**实现了在 linux 中客户端和服务端程序的交互以及基本的文件读写，并通过 *fork* 函数创建子进程对接收到的客户端套接字进行处理。

通过这次试验，加深了我对 *socket* 编程的基础知识的了解，对服务器端和客户端的模型有了进一步的认识，掌握了 *socket* 编程的基本 API 函数。