

中国矿业大学计算机学院



2019-2020(2)本科生 Linux 操作系统课程报告

内容范围 Linux 下的网络编程

指标点 1.2 占 比 30%

学生姓名 袁孝健 学 号 06172151

专业班级 信息安全 2017-01 班

任课教师 杨东平

课程基础理论掌握程度	熟练 <input type="checkbox"/>	较熟练 <input type="checkbox"/>	一般 <input type="checkbox"/>	不熟练 <input type="checkbox"/>
综合知识应用能力	强 <input type="checkbox"/>	较强 <input type="checkbox"/>	一般 <input type="checkbox"/>	差 <input type="checkbox"/>
报告内容	完整 <input type="checkbox"/>	较完整 <input type="checkbox"/>	一般 <input type="checkbox"/>	不完整 <input type="checkbox"/>
报告格式	规范 <input type="checkbox"/>	较规范 <input type="checkbox"/>	一般 <input type="checkbox"/>	不规范 <input type="checkbox"/>
实验完成状况	好 <input type="checkbox"/>	较好 <input type="checkbox"/>	一般 <input type="checkbox"/>	差 <input type="checkbox"/>
工作量	饱满 <input type="checkbox"/>	适中 <input type="checkbox"/>	一般 <input type="checkbox"/>	欠缺 <input type="checkbox"/>
学习、工作态度	好 <input type="checkbox"/>	较好 <input type="checkbox"/>	一般 <input type="checkbox"/>	差 <input type="checkbox"/>
抄袭现象	无 <input type="checkbox"/>	有 <input type="checkbox"/> 姓名:		
存在问题				
总体评价				

综合成绩:

任课教师签字:

年 月

摘 要

在 Linux 的学习过程中，不可避免的要学习网络相关的知识，而随着网络技术的发展，网络编程技术也越来越值得去学习。Linux 系统中，Socket 编程就是要设计一个能够通过网络和另一个进程进行通信的程序。因此，本文先从整体方面介绍了 Linux 下的 Socket 通信以及字节序的相关知识。接着对常用的网络编程系统逐个进行阐述，介绍了它们的原型、功能及用法等。最后，实现了一个基于 TCP 的网络通信程序，其功能包括传递信息、即时通信、文件传入，通过对该程序的实现思路及具体代码进行分析，从而对 Linux 下的 Socket 编程有了一个更加深入和具体的理解。

关键词：网络编程；套接字；TCP 协议

Abstract

Socket programming is to design a program that can communicate with another process through the network. In Linux system, Socket programming is to design a program that can communicate with another process through the network. Therefore, this article first introduces the related knowledge of Socket communication and byte order under Linux from the overall aspect. Next, the commonly used network programming systems are explained one by one, and their prototypes, functions and usages are introduced. Finally, a network communication program based on TCP is implemented. Its functions include transferring information, instant messaging, and file transfer. By analyzing the program's implementation ideas and specific codes, it has a deeper understanding of Socket programming under Linux. And specific understanding.

Keywords: network programming; socket; TCP protocol

目 录

摘 要.....	I
Abstract.....	I
4 Linux 下的网络编程	1
4.1 socket 概述	1
4.1.1 socket 通信	1
4.1.2 TCP/IP 与 UDP.....	1
4.1.3 网络字节序与主机字节序.....	2
4.2 socket 编程系统调用	3
4.2.1 socket.....	3
4.2.2 bind.....	4
4.2.3 connect.....	4
4.2.4 listen.....	5
4.2.5 accept.....	5
4.2.6 read.....	6
4.2.7 write.....	6
4.2.8 recv.....	6
4.2.9 recvfrom.....	7
4.2.10 send.....	8
4.2.11 sendto.....	8
4.2.12 close.....	9
4.3 socket 编程实例	9
4.3.1 实现思路.....	9
4.3.2 服务端代码.....	10
4.3.3 客户端代码.....	15
4.3.4 运行结果.....	19
4.4 收获与体会.....	20
参考文献.....	20

4 Linux 下的网络编程

要求：握 linux 下 socket 编程相关的各种系统调用 socket、bind、connect、listen、accept、read、recvfrom、write、sendto、close。

独立完成一个 linux 下的网络通信程序，要求包括客户端和服务端两部分程序，互相能够通信，传递消息，传送文件，即时聊天等等，要有必要的注释，并将运行过程和结果截图附上，最后阐明体会与收获。

根据所完成的网络通信程序的难度与功能复杂程度给出分值，若无运行截图，扣 10 分。

说明：如果撰写规范不符合《计算机学院考查类课程报告撰写规范》要求的，整体上酌情扣除 1-10 分。

4.1 socket 概述

4.1.1 socket 通信

在之前作业中，我们提到了进程之间的通信，如管道、信号、共享内存等，但是他们都仅限于用在本机进程之间通信。而网间进程通信要解决的是不同主机进程间的相互通信问题，为此首先要解决的是网间进程标识问题。同一主机上，不同进程可用进程号（process ID）唯一标识。但在网络环境下，各主机独立分配的进程号不能唯一标识该进程。其次，操作系统支持的网络协议众多，不同协议的工作方式不同，地址格式也不同。因此，网间进程通信还要解决多重协议的识别问题。

实际上 TCP/IP 协议族已经帮我们解决了这个问题，网络层的“ip 地址”可以唯一标识网络中的主机，而传输层的“协议:端口”可以唯一标识主机中的应用程序（进程）。这样利用（ip 地址，协议，端口）三元组就可以标识网络的进程了，网络中的进程通信就可以利用这个标志与其它进程进行交互。

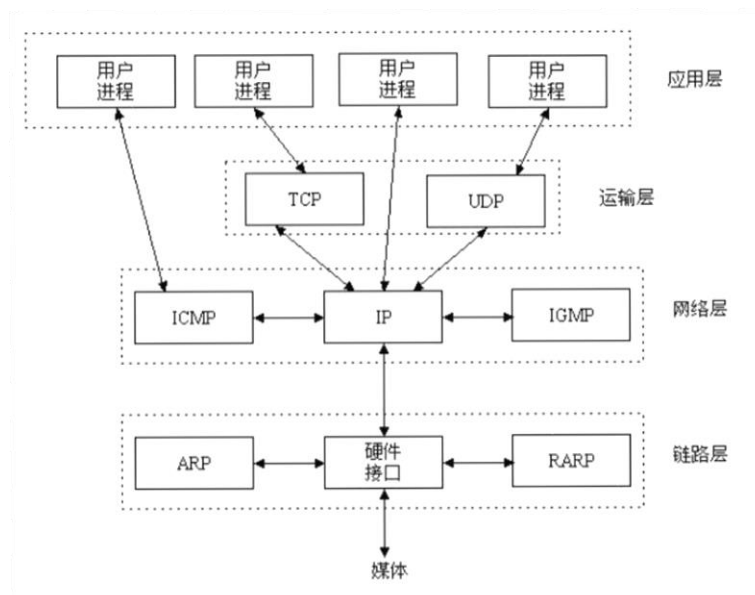
使用 TCP/IP 协议的应用程序通常采用应用编程接口：UNIX BSD 的套接字（socket）和 UNIX System V 的 TLI（已经被淘汰），来实现网络进程之间的通信。而目前而言，几乎所有的应用程序都是采用 socket，而现在又是网络时代，网络中进程通信是无处不在，因此 socket 编程越来越重要。实际上 Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket 其实就是一个门面模式，它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面，对用户来说，一组简单的接口就是全部，让 Socket 去组织数据，以符合指定的协议。

4.1.2 TCP/IP 与 UDP

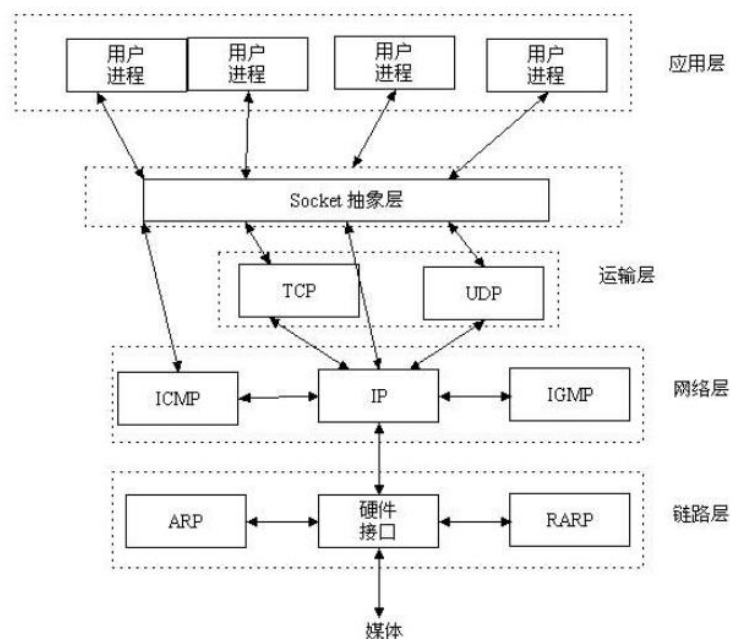
TCP/IP（Transmission Control Protocol/Internet Protocol）即传输控制协议/网间协议，是一个工业标准的协议集，它是为广域网（WANs）设计的。TCP/IP 协议存在于 OS 中，网络服务通过 OS 提供，在 OS 中增加支持 TCP/IP 的系统调用——Berkeley 套接

字，如 Socket, Connect, Send, Recv 等

UDP (User Data Protocol, 用户数据报协议) 是与 TCP 相对应的协议。它是属于 TCP/IP 协议族中的一种。如图：



TCP/IP 协议族包括运输层、网络层、链路层，而 socket 所在位置如图，Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层。



4.1.3 网络字节序与主机字节序

(1) 主机字节序

主机字节序就是我们平常说的大端和小端模式：不同的 CPU 有不同的字节序类型，这些字节序是指整数在内存中保存的顺序，这个叫做主机序。引用标准的 Big-Endian 和 Little-Endian 的定义如下：

- Little-Endian 就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。
- Big-Endian 就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。

(2) 网络字节序

4 个字节的 32 bit 值以下的次序传输：首先是 0~7bit，其次 8~15bit，然后 16~23bit，最后是 24~31bit。这种传输次序称作大端字节序。由于 TCP/IP 首部中所有的二进制整数在网络中传输时都要求以这种次序，因此它又称作网络字节序。字节序，顾名思义字节的顺序，就是大于一个字节类型的数据在内存中的存放顺序，一个字节的数没有顺序的问题了。

(3) 字节序的转换

在将一个地址绑定到 socket 的时候，请先将主机字节序转换为网络字节序，而不要假定主机字节序跟网络字节序一样使用的是 Big-Endian，接收数据包时需要将网络字节序转换为主机字节。转换时，需要引入头文件 `arpa/inet.h`：

- 主机字节序转网络字节序：`uint32_t htonl(uint32_t hostlong)` 或 `uint16_t htons(uint16_t hostshort)`
- 网络字节序转主机字节序：`uint32_t ntohl(uint32_t netlong)` 或 `uint16_t ntohs(uint16_t netshort)`

4.2 socket 编程系统调用

4.2.1 socket

(1) 原型

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

(2) 头文件

```
#include <sys/types.h>
#include <sys/socket.h>
```

(3) 返回值

成功返回 Socket 描述符；失败返回 -1，可用 `errno` 查看出错的详细情况

(4) 参数

① domain：程序采用的通讯协族。

- AF_UNIX：只用于单一的 Unix 系统进程间通信。
- AF_INET：用于 Internet 通信。

② type：采用的通讯协议，SOCK_STREAM 表示 TCP，SOCK_DGRAM 表示 UDP。

③ protocol：由于指定了 type，此值一般为 0。

(5) 功能

Socket 系统调用可用于创建一个 socket 描述符，socket 描述符可唯一标识一个 socket，类似于普通文件的文件描述符。后续的操作都有用到 socket 描述符，把它作为参数，通过它来进行一些读写操作。

4.2.2 bind

(1) 原型

```
int bind(int sockfd, struct sockaddr* servaddr, int addrlen);
```

(2) 头文件

```
#include <sys/types.h>
#include <sys/socket.h>
```

(3) 返回值

成功返回 0，失败返回-1，相应地设定全局变量 `errno`，最常见的错误是该端口已经被其他程序绑定。

(4) 参数

- ① `sockfd`: 由 `socket` 调用返回的文件描述符。
- ② `servaddr`: 出于兼容性，一般使用 `sockaddr_in` 结构。
- ③ `addrlen`: `servaddr` 结构的长度。

(5) 功能

用于 Server 程序，绑定被侦听的端口。当调用 `socket` 创建一个 socket 时，返回的 socket 描述字它存在于协议族空间中，不过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` 系统调用。

4.2.3 connect

(1) 原型

```
int connect(int sockfd, struct sockaddr* servaddr, int addrlen);
```

(2) 头文件

```
#include <sys/types.h>
#include <sys/socket.h>
```

(3) 返回值

成功，返回 0；失败，返回-1，相应地设定全局变量 `errno`。

(4) 参数

- ① `sockfd`: `socket` 返回的文件描述符。
- ② `servaddr`: 被连接的服务器端地址和端口信息，出于兼容性，一般使用 `sockaddr_in` 结构。

③ `addrlen`: `servaddr` 的长度。

(5) 功能

用于 Client 程序，连接到某个 Server。服务端在监听状态时，客户端通过调用 `connect` 可发出连接请求，服务端会收到这个请求。

4.2.4 `listen`

(1) 原型

```
int listen(int sockfd, int backlog);
```

(2) 头文件

```
#include <sys/types.h>
#include <sys/socket.h>
```

(3) 返回值

```
int listen(int sockfd, int backlog);
```

(4) 参数

① `sockfd`: 被 `bind` 的文件描述符(`socket()`建立的)。

② `backlog`: 设置 Server 端请求队列的最大长度。

(5) 功能

用于 Server 程序，侦听 `bind` 绑定的套接字。

4.2.5 `accept`

(1) 原型

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

(2) 头文件

```
#include <sys/types.h>
#include <sys/socket.h>
```

(3) 返回值

成功，则返回 Server 用于与 Client 进行数据传输的文件描述符；失败，则返回-1，相应地设定全局变量 `errno`。

(4) 参数

① `sockfd`: `listen` 后的文件描述符(`socket()`建立的)。

② `addr`: 返回 Client 的 IP、端口等信息，确切格式由套接字的地址类别(如 TCP 或 UDP)决定；若 `addr` 为 `NULL`，则 `addrlen` 应置为 `NULL`。

③ `addrlen`: 返回真实的 `addr` 所指向结构的大小，只要传递指针就可以，但必须先初始化为 `addr` 所指向结构的大小。

(5) 功能

Server 用它响应连接请求，建立与 Client 连接。`accept` 是阻塞函数，服务器端会

一直阻塞到有一个客户程序发出了连接。

4.2.6 read

(1) 原型

```
ssize_t read (int fd, void *buf, size_t count);
```

(2) 头文件

```
#include <unistd.h>
```

(3) 返回值

成功，返回实际所读取的字节数，如果已经读取到文件的结尾则返回 0；失败，则返回小于 0 的数。

(4) 参数

- ① fd: 文件描述符。
- ② buf: 指向内存块的指针，存放从文件中读出的字节。
- ③ count: 写入到 buf 中的字节数。

(5) 功能

负责从 socket 描述符对应 socket 中读取内容。

4.2.7 write

(1) 原型

```
ssize_t write(int fd, const void *buf, size_t count);
```

(2) 头文件

```
#include <unistd.h>
```

(3) 返回值

write 并非真正写入磁盘,而是先写入内存缓冲区,待缓冲区满或进行刷新操作后才真正写入磁盘,如果出现错误,返回-1。

(4) 参数

- ① fd: 文件描述符。
- ② buf: 指向内存块的指针，从该内存块读出数据写入文件。
- ③ count: 写入到文件中的字节数。

(5) 功能

用于向发送缓冲区中写入数据。

4.2.8 recv

(1) 原型

```
ssize_t recv(int sockfd, void *buf, size_t nbytes, int flags);
```

(2) 头文件

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

(3) 返回值

成功, 则返回实际接收的字节数; 失败, 则返回-1, 相应地设定全局变量 `errno`; 为 0, 则表示对端已经关闭。

(4) 参数

- ① `sockfd`: 接收端套接字描述符。
- ② `buf`: 指向容纳接收信息的缓冲区的指针。
- ③ `nbytes`: `buf` 缓冲区的大小。
- ③ `flags`: 接收标志, 一般为 0 或:

flags	说明
MSG_DONTWAIT	仅本操作非阻塞
MSG_OOB	发送或接收带外数据
MSG_PEEK	窥看外来消息
MSG_WAITALL	等待所有数据

(5) 功能

用于 TCP 协议中接收信息, `recv` 缺省是阻塞函数, 直到接收到信息或出错。

4.2.9 recvfrom

(1) 原型

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct
sockaddr *from, socklen_t *fromlen);
```

(2) 头文件

```
#include <sys/types.h>
#include <sys/socket.h>
```

(3) 返回值

成功, 返回实际接收的字节数; 失败, 返回-1, 错误原因存于 `errno` 中; 返回 0, 则表示对端已经关闭。

(4) 参数

- ① `sockfd`: `socket` 描述符。
- ② `buf`: 指向容纳接收 UDP 数据报的缓冲区的指针。
- ③ `len`: `buf` 缓冲区的大小。
- ④ `flags`: 接收标志, 一般为 0。
- ⑤ `from`: 指明数据的来源。
- ⑥ `fromlen`: 传入函数之前初始化为 `from` 的大小, 返回之后存放 `from` 实际大小。

(5) 功能

用于 UDP 协议中接收信息，recvfrom 是阻塞函数，直到接收到信息或出错。

4.2.10 send

(1) 原型

```
ssize_t send(int sockfd, const void *buf, size_t nbytes, int flags);
```

(2) 头文件

```
#include <sys/types.h>
#include <sys/socket.h>
```

(3) 返回值

成功，则返回已发送的字节数；失败，则返回-1，相应地设定全局变量 errno。

(4) 参数

- ① sockfd: 指定发送端套接字描述符。
- ② buf: 存放要发送数据的缓冲区。
- ③ nbytes: 实际要发送的数据的字节数。
- ③ flags: 一般设置为 0 或:

flags	说明
MSG_DONTROUTE	绕过路由表查找
MSG_DONTWAIT	仅本操作非阻塞
MSG_OOB	发送或接收带外数据

(5) 功能

用于 TCP 协议中发送信息，send 缺省是阻塞函数，直到发送完毕或出错。

4.2.11 sendto

(1) 原型

```
int sendto(int sockfd, const void *msg, int len, unsigned int flags,
const struct sockaddr *to, int tolen);
```

(2) 头文件

```
#include <sys/types.h>
#include <sys/socket.h>
```

(3) 返回值

成功，返回实际传送出去的字符数；失败，-1，错误原因存于 errno 中。

(4) 参数

- ① sock: 将要从其发送数据的套接字。
- ② buf: 指向将要发送数据的缓冲区。
- ③ len: 数据缓冲区的长度。
- ④ flags: 一般是 0。

- ⑤ to: 指明数据的目的地。
- ⑥ tolen: to 内存区的长度。

(5) 功能

用于 UDP 协议中发送信息，send 缺省是阻塞函数，直到发送完毕或出错。

4.2.12 close

(1) 原型

```
int close(int sockfd);
```

(2) 头文件

```
#include <sys/types.h>
#include <sys/socket.h>
```

(3) 返回值

成功，返回 0；失败。返回-1，并置错误码 errno。

(4) 参数

- ① sockfd: 要关闭的 socket 描述符。

(5) 功能

用于 TCP，关闭特定的 socket 连接。close 操作只是使相应 socket 描述字的引用计数-1，只有当引用计数为 0 的时候，才会触发 TCP 客户端向服务器发送终止连接请求。

4.3 socket 编程实例

4.3.1 实现思路

要实现一个基于 TCP 的通信程序，且要有即时通讯和传输文件的功能，因此我采取 C/S 模式，编写了两个程序 server.c 和 client.c 分别作为 C/S 模式中的服务端和客户端。

(1) 服务端

- ① 在服务端运行时，要求用户输入一个 name 作为聊天时的用户名。
- ② 随后是等待与客户端建立 TCP 连接，依次创建 socket 描述符和 sockaddr_in 结构体变量，然后 bind 系统调用绑定端口，listen 系统调用进行监听，最后用 accept 系统调用接收客户端的连接请求。
- ③ 建立 TCP 连接后，用 fork 系统调用创建一个子进程用来接收并显示客户端发来的消息。
- ④ 在父进程中，分为下面三种情况：
 - 服务端输入 quit: 表示服务端退出聊天，程序在杀掉子进程后将终止。
 - 服务端输入 file: 表示服务端要向客户端传输文件，采取分时传输的方式，每次传输 MAXSIZE 字节的文件，在文件传输完成后向客户端发送\END 标识符，方便客户端停止接收文件。

- 服务端输入其他字符：表示服务端向客户端发送聊天消息。

(2) 客户端

- ① 在客户端运行时，同样要求用户输入一个 name 作为聊天时的用户名。
- ② 随后是请求与服务端建立 TCP 连接，依次创建 socket 描述符和 sockaddr_in 结构体变量，然后调用 connect 系统调用请求与服务端建立连接。
- ③ 建立 TCP 连接后，用 fork 系统调用创建一个子进程用来接收服务端发来的消息或服务端传输来的文件，在接收文件时以收到的 \END 标识符来确定文件已接收完。
- ④ 在父进程中，主要用于向服务端发送消息，同样若发送的消息为 quit 指令，表示客户端退出聊天，程序终止。

(3) 其他

- ① 服务端和客户端的用户名以及通信过程中的消息，均通过 trim 函数来删除前后的空白字符。
- ② 服务端和客户端在输入 quit 指令退出聊天后，都会向对方发送一条消息声明自己已退出聊天，方便程序结束。
- ③ 在服务端的 bind 调用前使用了 setsockopt 函数来避免 TCP 套接字状态 TIME_WAIT 导致的 Address already in use 报错。

4.3.2 服务端代码

服务端代码 server.c 如下，具体解释可见代码注释：

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <signal.h>
#include <sys/prctl.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <ctype.h>

#define MY_PORT 8888
#define BACKLOG 10
#define MAXSIZE 1000

//去除首尾的空格
void trim(char *str)
{
    int i, j;
    i = 0;
```

```

    j = strlen(str) - 1;
    while (str[i] != '\0' && isspace(str[i]))
        ++i;
    while (str[j] != '\0' && isspace(str[j]))
        --j;
    strncpy(str, str + i, j - i + 1);
    str[j - i + 1] = '\0';
}

int main()
{
    int serverfd, clientfd;
    char name[MAXSIZE], msg[MAXSIZE], buf[MAXSIZE];
    //设置服务端的用户名
    printf("请输入用户名:\n");
    while (1)
    {
        //输入用户名并去除空格
        fgets(name, MAXSIZE, stdin);
        trim(name);
        if (name[0] != '\0')
        {
            //增加标识
            strcat(name, "(server)");
            printf("欢迎进入聊天室!\n");
            break;
        }
        printf("用户名不能为空!\n");
    }

    //创建服务端 socket,并取得描述符
    serverfd = socket(AF_INET, SOCK_STREAM, 0);
    if (serverfd == -1)
    {
        perror("Server socket failed!");
        exit(1);
    }

    //创建 sockaddr_in 结构体变量
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET; //使用 Internet 地质
    server_addr.sin_port = htons(MY_PORT); //侦听端口号
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY); //侦听 IP 地址
    bzero(&(server_addr.sin_zero), 8); //其余部分置 0

```

```
//使用 bind 进行绑定
int opt = 1;
if (setsockopt(serverfd, SOL_SOCKET, SO_REUSEADDR, &opt,
sizeof(opt)) != 0)
{
    perror("Server setsockopt failed!");
    return 1;
}
if (bind(serverfd, (struct sockaddr *)&server_addr,
sizeof(server_addr)) == -1)
{
    perror("Server bind failed!");
    exit(1);
}

//使用 listen 进行监听
if (listen(serverfd, BACKLOG) == -1)
{
    perror("Server listen failed!");
    exit(1);
}

//使用 accept 接受客户端的连接请求
clientfd = accept(serverfd, NULL, NULL);
if (clientfd == -1)
{
    perror("Server accept failed!");
    exit(1);
}

pid_t pid = fork();

while (1)
{
    if (pid < 0)
    {
        printf("Server fork failed!\n");
        exit(1);
    }
    //父进程用于发送消息及传输文件
    if (pid)
    {
        fgets(msg, MAXSIZE, stdin);
    }
}
```

```

trim(msg);
//输入 quit 关闭聊天
if (!strcmp(msg, "quit"))
{
    //向客户端发送关闭通知
    strcpy(buf, "服务端退出, 聊天室关闭!");
    if (send(clientfd, buf, sizeof(buf), 0) == -1)
    {
        perror("Server send failed!");
    }
    //保存父进程退出前, 杀死子进程
    kill(pid, SIGTERM);
    close(clientfd);
    close(serverfd);
    sleep(0.5);
    exit(0);
}
//向客户端传输文件
else if (!strncmp(msg, "file", 4))
{
    char filename[MAXSIZE];
    printf("请输入要传输的文件路径: \n");
    fgets(filename, MAXSIZE, stdin);
    trim(filename);
    //创建文件描述符号
    FILE *fp = fopen(filename, "r");
    if (fp == NULL)
    {
        printf("File Not Found!\n");
        continue;
    }
    strcpy(msg, "服务端正在向你传输文件...");
    if (send(clientfd, msg, sizeof(msg), 0) < 0)
    {
        perror("Server send failed!");
        continue;
    }
    printf("正在发送文件...\n");
    //清空缓冲区
    bzero(buf, MAXSIZE);
    int len = 0;
    //循环分段发送文件, 每次发送 MAXSIZE 字节
    while ((len = fread(buf, sizeof(char), MAXSIZE, fp)) > 0)
    {

```



```
        if (send(clientfd, buf, sizeof(buf), 0) < 0)
        {
            printf("文件%s 发送出错!\n", filename);
            break;
        }
        bzero(buf, MAXSIZE);
    }
    //文件结束确认\END
    strcpy(buf, "\\END");
    if (send(clientfd, buf, sizeof(buf), 0) > 0)
    {
        fclose(fp);
        printf("文件发送成功!\n");
    }
    else
    {
        perror("文件发送失败!");
    }
}
else
{
    sprintf(buf, "%s: %s", name, msg);
    if (send(clientfd, buf, sizeof(buf), 0) == -1)
    {
        perror("Server send failed!");
    }
}
}
//子进程用来接受并显示客户端发来的消息
else
{
    if (getppid() == 1)
        exit(0);
    if (recv(clientfd, buf, sizeof(buf), 0) > 0)
    {
        printf("%s\n", buf);
        if (!strcmp(buf, "客户端退出聊天!"))
        {
            kill(getppid(), SIGTERM);
            close(clientfd);
            close(serverfd);
            sleep(0.5);
            exit(0);
        }
    }
}
```

```

    }
    else
    {
        perror("Server recv failed!");
    }
}
}
return 0;
}

```

4.3.3 客户端代码

客户端代码 client.c 如下，具体解释可见代码注释：

```

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <ctype.h>

#define SERVER_PORT 8888
#define MAXSIZE 1000

//去除首尾的空格
void trim(char *str)
{
    int i, j;
    i = 0;
    j = strlen(str) - 1;
    while (str[i] != '\0' && isspace(str[i]))
        ++i;
    while (str[j] != '\0' && isspace(str[j]))
        --j;
    strncpy(str, str + i, j - i + 1);
    str[j - i + 1] = '\0';
}

int main()
{
    int sockfd;
    char name[MAXSIZE], msg[MAXSIZE], buf[MAXSIZE];

    //设置客户端的用户名

```

```

printf("请输入用户名:\n");
while (1)
{
    //输入用户名并去除空格
    fgets(name, MAXSIZE, stdin);
    trim(name);
    if (name[0] != '\0')
    {
        //增加标识
        strcat(name, "(client)");
        printf("欢迎进入聊天室!\n");
        break;
    }
    printf("用户名不能为空!\n");
}

//创建客户端 socket,并取得描述符
sockfd = socket(AF_INET, SOCK_STREAM, 0);

if (sockfd == -1)
{
    perror("Client socket failed!");
    exit(1);
}

//创建 sockaddr_in 结构体变量
struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET; //使用 Internet 地
质

server_addr.sin_port = htons(SERVER_PORT); //侦听端口号
server_addr.sin_addr.s_addr = inet_addr("127.0.0.1"); //侦听 IP 地址
bzero(&(server_addr.sin_zero), 8); //其余部分置 0

//用 connect 与服务端进行连接
if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(struct
sockaddr_in)) == -1)
{
    perror("Client connect failed!");
    exit(1);
}

sprintf(msg, "用户%s 加入聊天!\n", name);
if (send(sockfd, msg, sizeof(msg), 0) == -1)
{

```

```
perror("Client send failed!");
exit(1);
}
bzero(msg, MAXSIZE);

pid_t pid = fork();
while (1)
{
    if (pid < 0)
    {
        printf("Client foek failed!\n");
        exit(1);
    }

    //父进程用于发送消息
    if (pid)
    {
        fgets(msg, MAXSIZE, stdin);
        trim(msg);
        //输入 quit 推出聊天
        if (!strcmp(msg, "quit"))
        {
            sprintf(buf, "客户端退出聊天!", name);
            if (send(sockfd, buf, sizeof(buf), 0) == -1)
            {
                perror("Client send falied!");
            }
            printf("您已退出聊天!\n");
            kill(pid, SIGTERM);
            close(sockfd);
            sleep(0.5);
            exit(0);
        }
        sprintf(buf, "%s: %s", name, msg);
        if (send(sockfd, buf, sizeof(buf), 0) == -1)
        {
            perror("Client send failed!");
        }
    }

    //子进程用来接受消息
    else
    {
        if (getppid() == 1)
            exit(0);
    }
}
```

```

if (recv(sockfd, buf, sizeof(buf), 0) > 0)
{
    printf("%s\n", buf);
    //接收服务端发来的文件
    if (!strcmp(buf, "服务端正在向你传输文件..."))
    {
        //保存为 recv.txt
        FILE *fp = fopen("recv.txt", "w");
        printf("正在传输文件...\n");
        bzero(buf, MAXSIZE);
        int len = 0;
        //循环接受并写入文件,以\END 为文件结束标志
        while ((len = recv(sockfd, buf, MAXSIZE, 0)) > 0 &&
            strcmp(buf, "\\END"))
        {
            int l = strlen(buf);
            if (fwrite(buf, sizeof(char), l, fp) < l)
            {
                puts("写入文件失败! ");
                continue;
            }
            bzero(buf, MAXSIZE);
        }
        fclose(fp);
        printf("文件传输完成, 保存在 recv.txt!\n");
    }
    //服务端退出聊天
    else if (!strcmp(buf, "服务端退出, 聊天室关闭!"))
    {
        kill(getppid(), SIGTERM);
        sleep(0.5);
        exit(0);
    }
}
else
{
    perror("Server recv falied!");
}
}
close(sockfd);
return 0;
}

```

4.3.4 运行结果

(1) 聊天通信

① 服务端：

```
lethe@ubuntu-vm: ~/MyCode/cpp
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
lethe@ubuntu-vm:~/MyCode/cpp$ ./server
请输入用户名：
admin
欢迎进入聊天室！
用户YuanXiaoJian(client)加入聊天！

Hello, I am admin!
YuanXiaoJian(client): Hi, My name is YuanXiaoJian.
How are u.
YuanXiaoJian(client): I am fine, thanks!
客户端退出聊天！
已终止
```

② 客户端：

```
lethe@ubuntu-vm: ~/MyCode/cpp
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
lethe@ubuntu-vm:~/MyCode/cpp$ ./client
请输入用户名：
YuanXiaoJian
欢迎进入聊天室！
admin(server): Hello, I am admin!
Hi, My name is YuanXiaoJian.
admin(server): How are u.
I am fine, thanks!
quit
您已退出聊天！
```

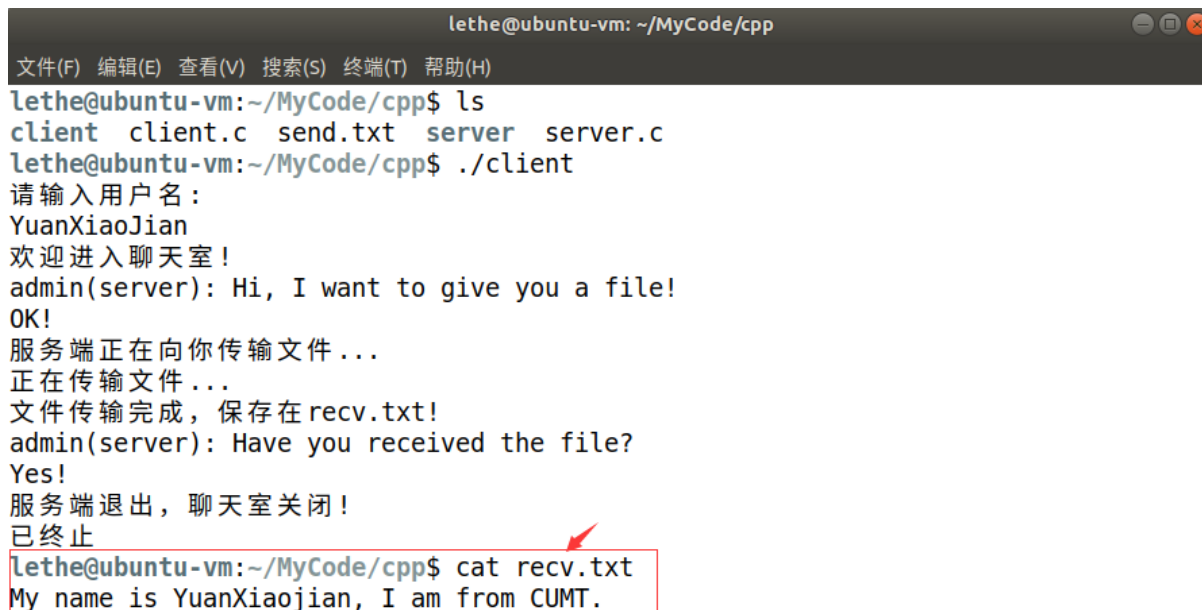
(2) 文件传输

① 服务端：

```
lethe@ubuntu-vm: ~/MyCode/cpp
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
lethe@ubuntu-vm:~/MyCode/cpp$ cat send.txt
My name is YuanXiaojian, I am from CUMT.
lethe@ubuntu-vm:~/MyCode/cpp$ ./server
请输入用户名：
admin
欢迎进入聊天室！
用户YuanXiaoJian(client)加入聊天！

Hi, I want to give you a file!
YuanXiaoJian(client): OK!
file
请输入要传输的文件路径：
send.txt
正在发送文件...
文件发送成功！
Have you received the file?
YuanXiaoJian(client): Yes!
quit
```

② 客户端:



```
lethe@ubuntu-vm: ~/MyCode/cpp
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
lethe@ubuntu-vm:~/MyCode/cpp$ ls
client client.c send.txt server server.c
lethe@ubuntu-vm:~/MyCode/cpp$ ./client
请输入用户名:
YuanXiaoJian
欢迎进入聊天室!
admin(server): Hi, I want to give you a file!
OK!
服务端正在向你传输文件...
正在传输文件...
文件传输完成, 保存在recv.txt!
admin(server): Have you received the file?
Yes!
服务端退出, 聊天室关闭!
已终止
lethe@ubuntu-vm:~/MyCode/cpp$ cat recv.txt
My name is YuanXiaoJian, I am from CUMT.
```

4.4 收获与体会

通过本章的理论学习以及此次实验的课后巩固, 我对 Linux 下的进程通信有了更加深入的理解与认识。尤其是在这一次的实验过程中, 我进行了大量 Linux C 的 socket 编程, 对课上所讲的 socket、bind、listen、accept、connect、recv、send 等系统调用有了更加具体的掌握, 知道了它们之间相互的关联与应用。

在之前的学习过程中, 我尝试用 Python 写过 TCP 通信以及文件传输的程序, 但是由于 Python 的封装性较好, 并没有从底层了解到服务端与客户端的 TCP 连接是如何一步一步建立的。而这次首度用 C 语言写, 明显感觉到代码量与复杂程度远远大于 Python, 但是在用心完成后, 对 TCP 通信的整个流程都有了一个认识, 实际上这样对我们的学习才是更有利的, 不仅要知其然, 更要知其所以然。

由于之前没有用 C 语言写过具有实际工程意义的代码, 所以虽然完成了此次实验, 但在过程中仍然存在许多的问题。C 语言实际上更考验编程人员能否以计算机的思维解决问题, 在这一点上我还有待提高, 在日后的学习生活中, 我更会多加练习, 培养自己的这种思维方式。

参考文献

- [1] 鸟哥. 鸟哥的 Linux 私房菜: 基础学习篇[M]. 人民邮电出版社, 2010.
- [2] 宋敬彬. Linux 网络编程. 清华大学出版社, 2014.
- [3] 宋劲杉. Linux C 编程一站式学习. 电子工业出版社, 2009.