

# 《Linux 操作系统设计实践》

## 实验报告

### 实验 3：网络编程

院    系： 数学与计算机科学学院

专    业： 计算机科学与技术

年    级： 2016 级计算机 5 班

学    号： 031602507

姓    名： 陈俞辛

## 一、实验环境：Ubuntu Kylin 14.04

## 二、实验内容：

### （一）实验代码

server 端代码：

```
#include <netinet/in.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
    int server_sockfd, client_sockfd;
    socklen_t client_len = 0;

    struct sockaddr_in server_addr; //定义服务器端套接口数据结构server_addr
    struct sockaddr_in client_addr; //定义客户端套接口数据结构client_addr

    // 服务器端开始建立 socket 描述符
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);

    // 设置服务器接收的连接地址和监听的端口
    server_addr.sin_family = AF_INET; // 指定网络套接字为TCP/IP 地址族
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY); // 接受本机地址的连接
    server_addr.sin_port = htons(6666); // 绑定到 6666 端口

    // 设为TCP/IP地址族
    bind(server_sockfd, (struct sockaddr *) &server_addr, sizeof(server_addr));

    // 创建套接字队列，监听套接字，同时处理的最大连接请求数为5
    listen(server_sockfd, 5);

    // 忽略子进程停止或退出信号
    signal(SIGCHLD, SIG_IGN);

    while (1)
    {
        char ch = '\0';
        client_len = sizeof(client_addr);
        printf("Server waiting\n");

        // 调用 accept 接受一个连接请求
        client_sockfd = accept(server_sockfd, (struct sockaddr *)&client_addr, &client_len);
```

```

if (fork() == 0)
{
    // 子进程中，读取客户端发过来的信息，处理信息，再发送给客户端
    read(client_sockfd, &ch, 1);
    printf("char from client = %c\n",ch);
    sleep(1);
    ch++;
    write(client_sockfd, &ch, 1);
    close(client_sockfd);
    exit(0);
}
else
{
    // 父进程中，关闭套接字
    close(client_sockfd);
}
}
return 0;
}

```

client 端代码:

```

#include <unistd.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    int sockfd = -1;
    struct sockaddr_in address;
    int result;
    char ch = 'A';

    // 客户程序开始建立 socket 描述符
    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    // 设置要连接的服务器的信息
    address.sin_family = AF_INET; // 指定网络套接字为 TCP/IP 地址族
    address.sin_addr.s_addr = inet_addr("127.0.0.1"); // 服务器地址
    address.sin_port = htons(6666); // 服务器所监听的端口

    // 连接到服务器
    if (connect(sockfd, (struct sockaddr *) &address, (socklen_t) sizeof(address)) == -1)
    {
        perror("ops: client\n");
        exit(1);
    }

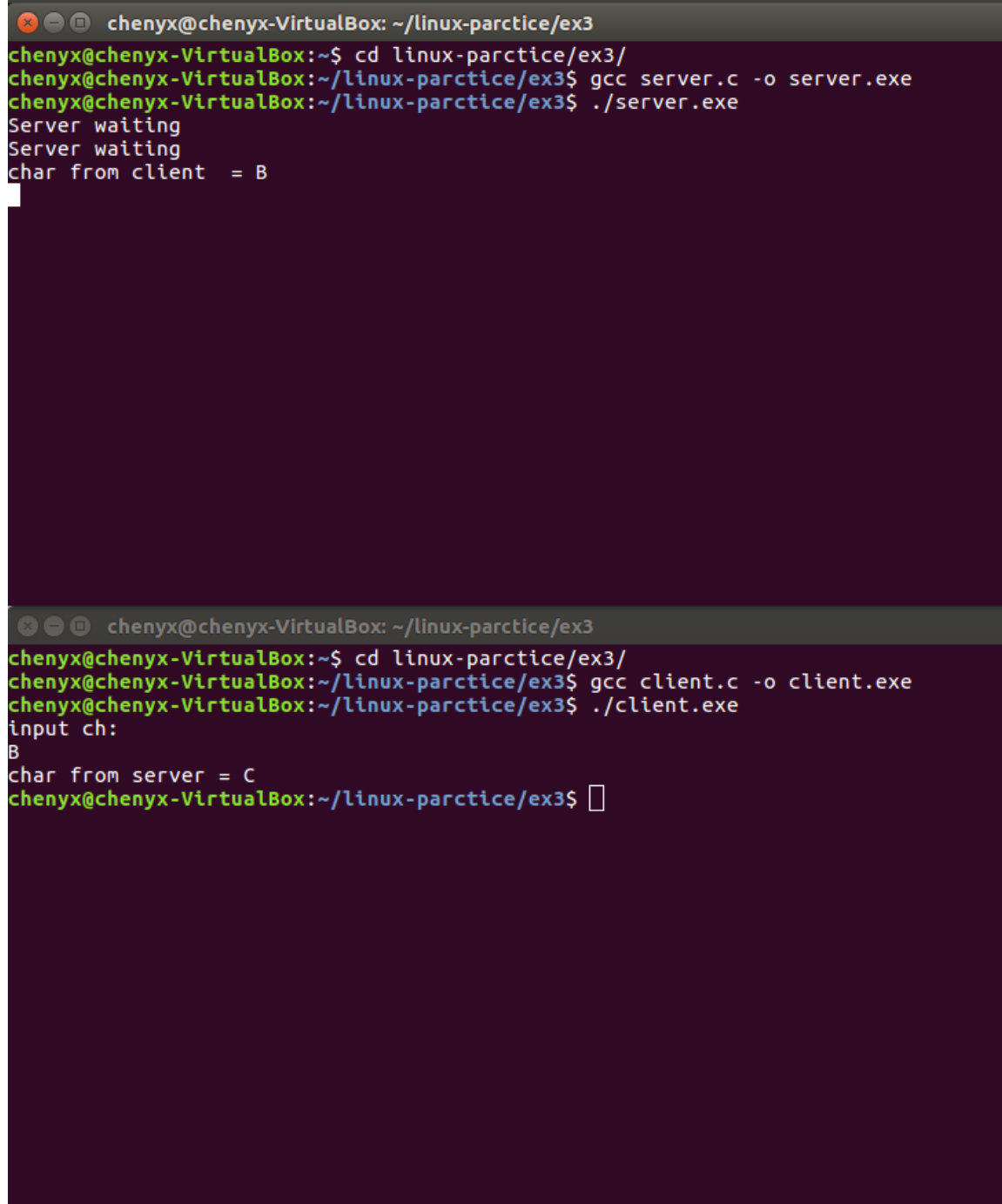
    printf("input ch:\n");
    scanf("%c",&ch);
    // 发送数据给服务器
    write(sockfd, &ch, 1);
}

```

```
// 从服务器获取数据
read(sockfd, &ch, 1);

printf("char from server = %c\n", ch);
close(sockfd);
exit(0);
}
```

## （二）实验结果

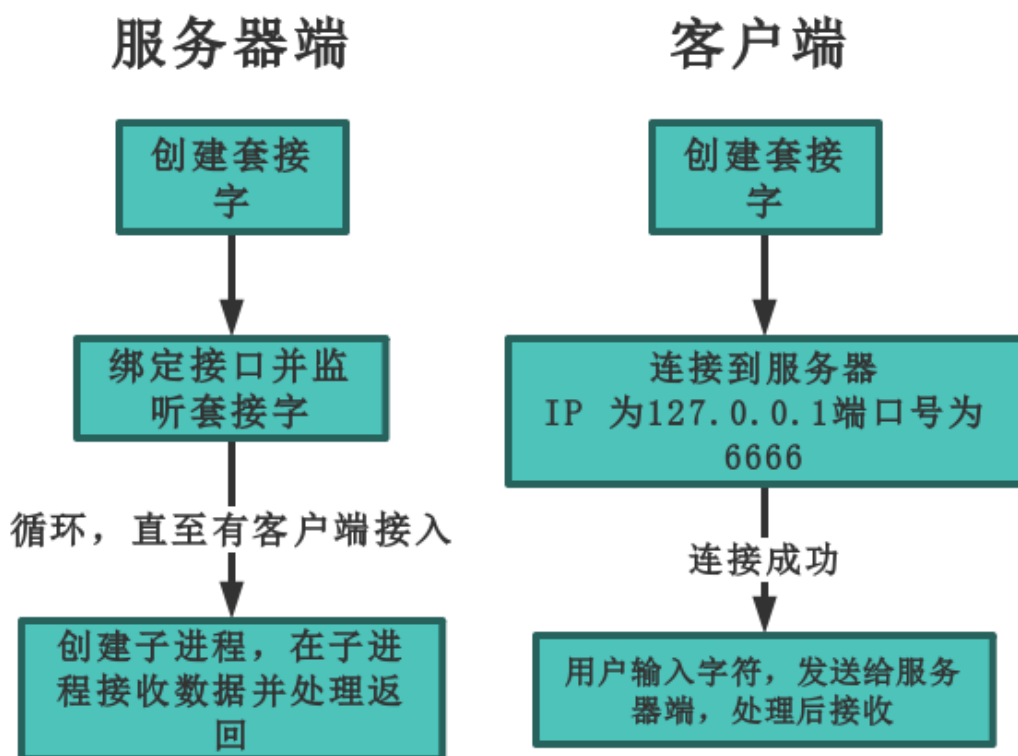


```
chenyx@chenyx-VirtualBox: ~/linux-parctice/ex3
chenyx@chenyx-VirtualBox:~$ cd linux-parctice/ex3/
chenyx@chenyx-VirtualBox:~/linux-parctice/ex3$ gcc server.c -o server.exe
chenyx@chenyx-VirtualBox:~/linux-parctice/ex3$ ./server.exe
Server waiting
Server waiting
char from client = B

chenyx@chenyx-VirtualBox: ~/linux-parctice/ex3
chenyx@chenyx-VirtualBox:~$ cd linux-parctice/ex3/
chenyx@chenyx-VirtualBox:~/linux-parctice/ex3$ gcc client.c -o client.exe
chenyx@chenyx-VirtualBox:~/linux-parctice/ex3$ ./client.exe
input ch:
B
char from server = C
chenyx@chenyx-VirtualBox:~/linux-parctice/ex3$
```

### 三、实验总结

#### (一) 编程思想



server.c 是一个服务器端，设置的 IP 地址为 127.0.0.1，端口号为 6666。它处理数据十分简单，就是把客户端发过来的字符+1 然后发送回给客户端。其中收发数据分别通过 read 函数和 write 函数。

client.c 是一个客户端，用户可以输入一个字符，然后发送到服务器端，在服务器端处理之后再接收。收发数据同样分别通过 read 函数和 write 函数。

#### (二) 相关知识

0、

socket 的接口函数声明在头文件 sys/types.h 和 sys/socket.h 中。

##### 1、socket()

该函数用来创建一个套接字，并返回一个描述符，该描述符可以用来访问该套接字。

函数原型如下：

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

函数中的三个参数分别对应前面所说的三个套接字属性。protocol 参数设置为 0 表示使用默认协议。

## 2、bind()

该函数把通过 socket 调用创建的套接字命名，从而让它可以被其他进程使用。对于 AF\_UNIX，调用该函数后套接字就会关联到一个文件系统路径名，对于 AF\_INET，则会关联到一个 IP 端口号。成功时返回 0，失败时返回-1；

函数原型如下：

```
int bind( int socket, const struct sockaddr *address, size_t address_len);
```

## 3、listen()

该函数用来创建一个队列来保存未处理的请求。成功时返回 0，失败时返回-1。backlog 用于指定队列的长度，等待处理的进入连接的个数最多不能超过这个数字，否则往后的连接将被拒绝，导致客户的连接请求失败。调用后，程序一直会监听这个 IP 端口，如果有连接请求，就把它加入到这个队列中。

函数原型如下：

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

## 4、accept()

该函数用来等待客户建立对该套接字的连接。accept 系统调用只有当客户程序试图连接到由 socket 参数指定的套接字上时才返回，也就是说，如果套接字队列中没有未处理的连接，accept 将阻塞直到有客户建立连接为止。accept 函数将创建一个新套接字来与该客户进行通信，并且返回新套接字的描述符，新套接字的类型和服务器监听套接字类型是一样的。address 为连接客户端的地址，参数 address\_len 指定客户结构的长度，如果客户地址的长度超过这个值，会被截断。

函数原型如下：

```
int accept(int socket, struct sockaddr *address, size_t *address_len);
```

## 5、connect()

该函数用来让客户程序通过在一个未命名套接字和服务器监听套接字之间建立连接的方法来连接到服务器。参数 socket 指定的套接字连接到参数 address 指定的服务器套接字。成功时返回 0，失败时返回-1。

函数原型如下：

```
int connect(int socket, const struct sockaddr *address, size_t address_len);
```

## 6、close()

该函数用来终止服务器和客户上的套接字连接，我们应该总是在连接的两端（服务器和客户）关闭套接字。

函数原型如下：

```
int close(int fd);
```

