

基于 linux 的多线程服务器的开发

Development of a Multi-thread Server based on Linux Operating System

Abstract

This project is to develop a multi-thread server, which can respond to several requests at the same time. The concrete method is to create threads by calling operating system API (Application Programming Interface). The 3 users send message “Hi,God!” to the server respectively, and display the answer after receiving the the respond from the server. The system is test with TCP and UDP protocol , with user defined port number.

英才学院 2011 级 1 班

刘北北

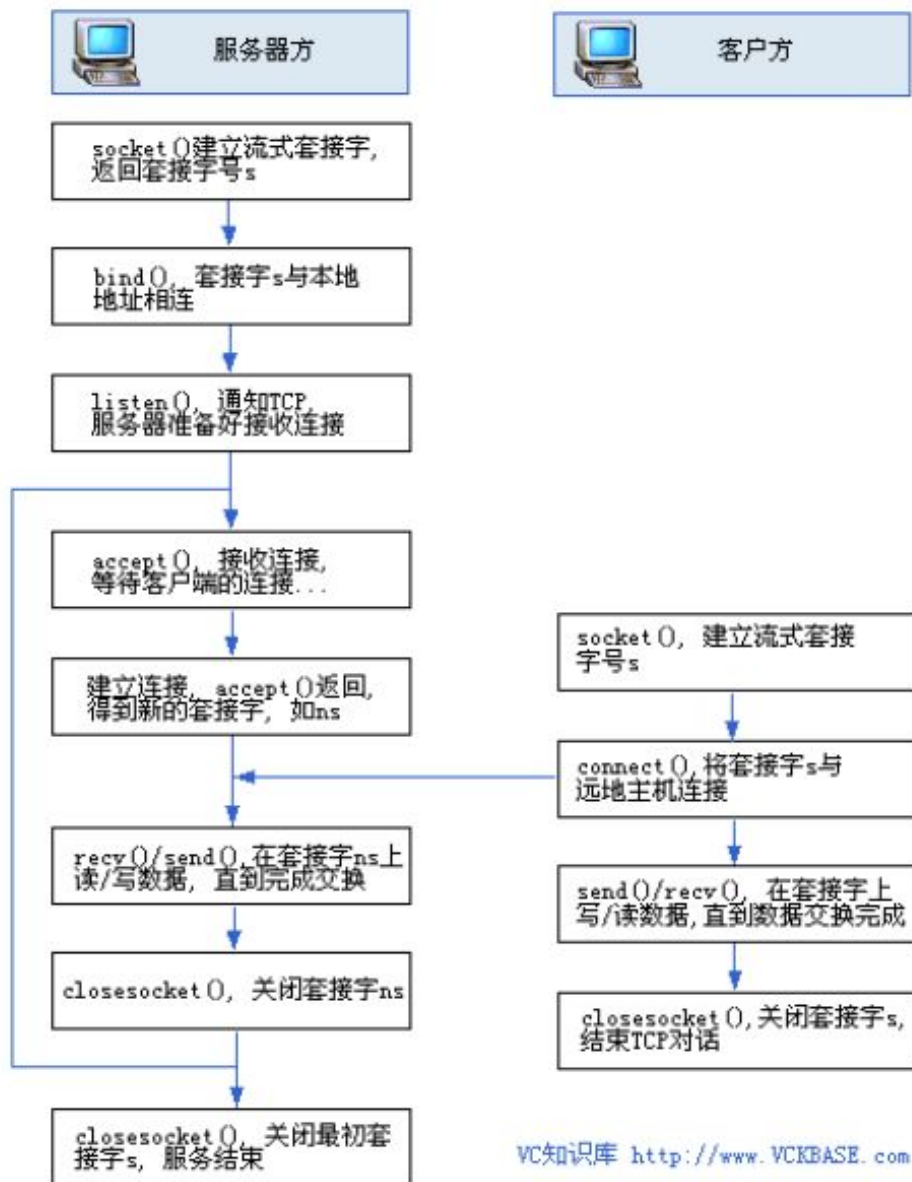
学号 : 2011019060027

摘要：

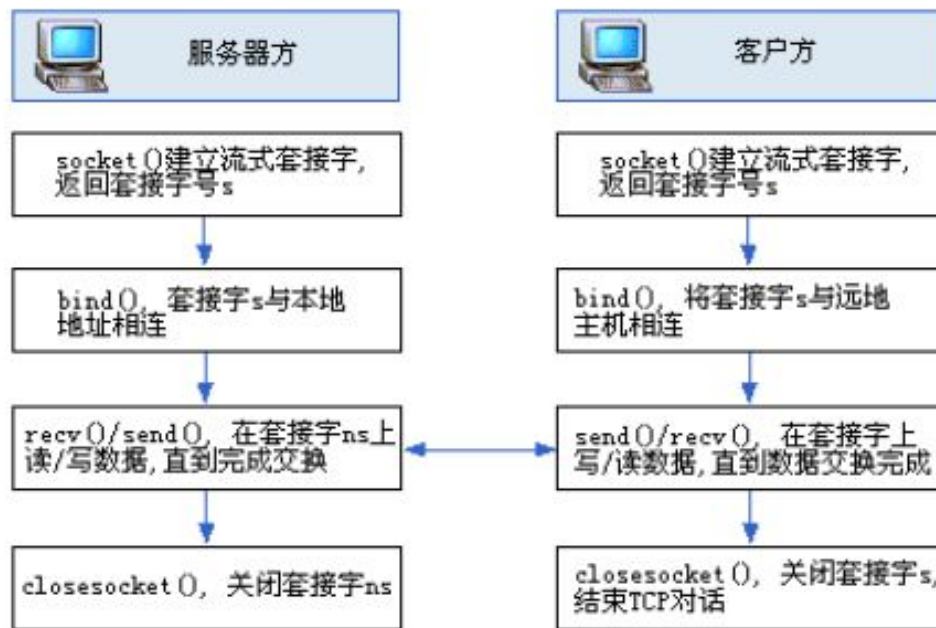
本设计开发一个多线程服务器，能并行服务于多个请求。具体方法是通过调用操作系统 API 接口创建线程。

3 个客户端分别向服务器发送 “Hi,God!”，并在收到服务器回答之后显示出其回答。用 tcp 和 udp 两个协议测试，端口号自定义。

一、明确客户端和服务端使用 tcp 协议通信的过程：



二、明确客户端和服务端使用 udp 协议通信的过程：



三、用编程语言具体实现 tcp 通信用程（先写单线程）：

Java 语言有一些封装好的类实现套接字连接、监听等过程，在网络编程方面有较大优势，但由于我刚接触 java，还不熟，暂时用 c 语言写。学长借给了我一本书是基于 Linux 操作系统的，正好顺便了解一下 Linux 操作系统，于是本次作业就采用 c 语言在 Linux 操作系统上实现。

基本定义：

struct sockaddr_in 结构类型是用来保存 socket 信息的：

```

struct sockaddr_in
{
    short int sin_family; 用于指定地址族
    unsigned short int sin_port; 套接字通信的端口号
    struct in_addr sin_addr; 通信的 IP 地址
    unsigned char sin_zero[8]; 填充 0，保持与 struct sockaddr 同样大小，方便类型转换};
  
```

头文件：

```

#include <stdlib.h> 提供 exit()等工具函数
#include <stdio.h> 标准输入输出
#include <sys/types.h> 定义了 size_t（反映内存中对象的大小）等系统数据类型
#include <sys/socket.h> 定义 socket 函数及数据类型
#include <netinet/in.h> 定义数据结构 sockaddr_in
#include <string.h> 提供 bezero 函数
#include <netdb.h> 提供网络字节顺序的转换函数
#include <arpa/inet.h> 提供 ip 地址转换函数
#include <unistd.h> 提供通用的文件、目录、程序及进程操作的函数

#include <pthread.h> 提供多线程操作的函数，定义结构 pthread_t 和 pthread_mutex_t 等
#include <sys/poll.h> 提供 socket 等待测试进制函数
  
```

服务器：

```
sfd = socket(AF_INET, SOCK_STREAM, 0); //建立监听套接字，描述符为 sfd
```

```

bzero(&s_add,sizeof(struct sockaddr_in)); //将存储服务器端信息的结构体赋值
s_add.sin_family=AF_INET; //服务的类型: IPV4
s_add.sin_addr.s_addr=htonl(INADDR_ANY); //将本地地址转为网络地址
s_add.sin_port=htons(portnum); //将本地端口号转为网络端口号
bind(sfd,(struct sockaddr *)&s_add, sizeof(struct sockaddr))
//将服务器的地址及端口和监听套接字绑定
listen(sfd,5) //监听, 等待客户端出现连接
nfd = accept(sfd, (struct sockaddr *)&c_add, &sin_size);
//如果有用户连接, 则建立连接套接字 (描述符 nfd), 并与客户端 c_add 绑定
write(nfd,"hello,welcome to my server \r\n",32) //向连接套接字 nfd 写入数据
read(nfd, buf, sizeof(buf)) //读客户端发过来的信息
close(nfd); //关闭连接套接字

```

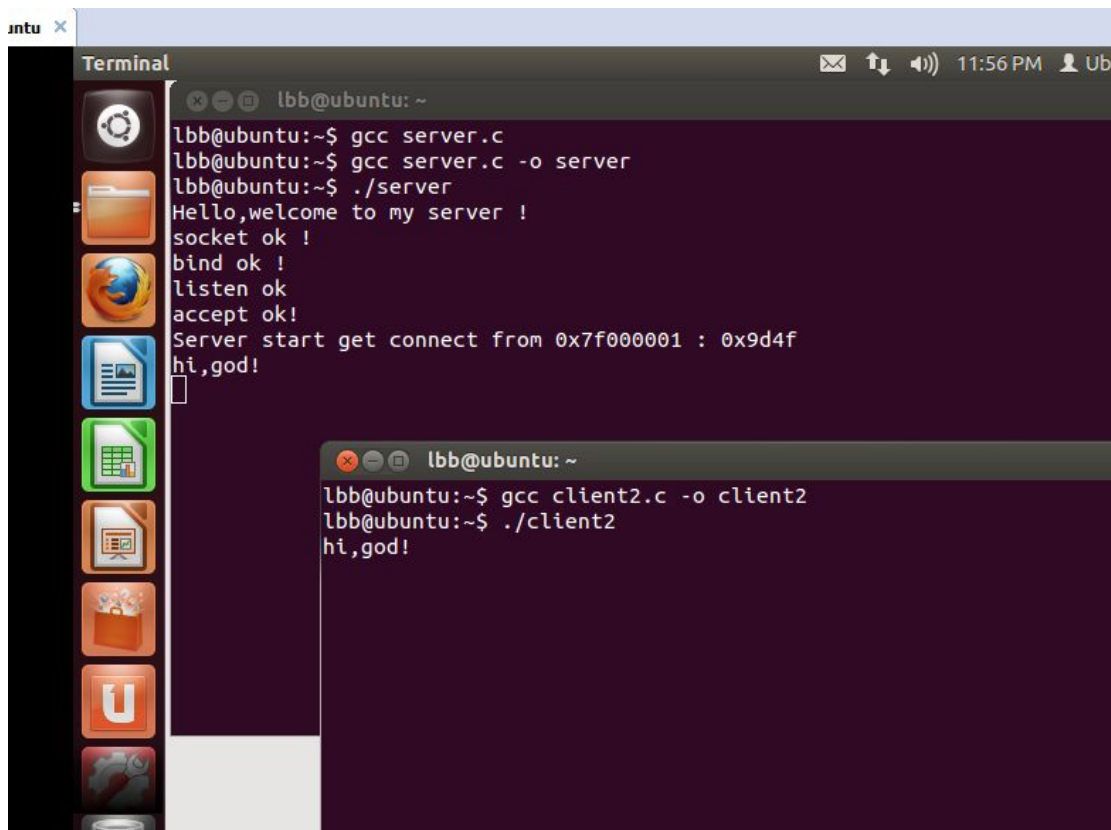
客户机：

```

cfd = socket(AF_INET, SOCK_STREAM, 0); //建立连接套接字, 描述符为 cfd
bzero(&s_add,sizeof(struct sockaddr_in)); //将存储服务器端信息的结构体赋值
s_add.sin_family=AF_INET; //服务的类型: IPV4
s_add.sin_addr.s_addr= inet_addr(host); //将 ip 地址转换形式
s_add.sin_port=htons(portnum); //将本地端口号转为网络端口号
connect(cfd,(struct sockaddr *)&s_add, sizeof(s_add))
//客户端 x 向服务器 s_add 发送服务请求
read(cfd,buffer,1024) //读服务器发过来的信息
close(cfd); //关闭连接套接字

```

在 ubuntu 上运行的结果：



```

Terminal
lbb@ubuntu: ~
lbb@ubuntu:~$ gcc server.c
lbb@ubuntu:~$ gcc server.c -o server
lbb@ubuntu:~$ ./server
Hello,welcome to my server !
socket ok !
bind ok !
listen ok
accept ok!
Server start get connect from 0x7f000001 : 0x9d4f
hi,god!

lbb@ubuntu: ~
lbb@ubuntu:~$ gcc client2.c -o client2
lbb@ubuntu:~$ ./client2
hi,god!

```

四、多线程通信的原理：

我们可以把线程看成是一个进程（执行程序）中的一个执行点，每个进程在任何给定时刻可能有若干个线程在运行。一个进程中的所有线程共享该进程中同样的地址空间，同样的数据和代码，以及同样的资源。进程中每个线程都有自己独立的栈空间，和其它线程分离，并且不可互相访问。每个线程在本进程所占的 CPU 时间内，要么以时间片轮换方式，要么以优先级方式运行。如果以时间片轮换方式运行，则每个线程获得同样的时间量；如果以优先级方式运行，则优先级高的线程将得到较多的时间量，而优先级低的线程只能得到较少的时间量。方式的选择主要取决于系统时间调度器的机制以及程序的实时性要求。

四、具体代码实现多线程通信：

//服务器端的 socket,bind,listen 等步骤与单线程相同

```
pthread_mutex_init(&mut,NULL);    // 用默认属性初始化互斥锁
```

调用函数 thread_create(...);

```
memset(&thread, 0, sizeof(thread));    //创建线程
```

```
pthread_create(&thread[0], NULL, thread1, NULL)) //依次创建 3 个线程
```

```
pthread_create(&thread[1], NULL, thread2,NULL))
```

```
pthread_create(&thread[2], NULL, thread3, NULL))
```

调用函数 thread_wait();

```
pthread_join(thread[0],NULL);    //等待一个线程的结束
```

```
pthread_join(thread[1],NULL);
```

```
pthread_join(thread[2],NULL);
```

函数 thread()

```
pthread_mutex_lock(&mut);    // 声明开始用互斥锁上锁
```

```
number++;    //流水计数
```

```
write(nfd,"hello,welcome to my server \r\n",32) //向客户端发信息
```

```
close(nfd);    //关闭连接套接字
```

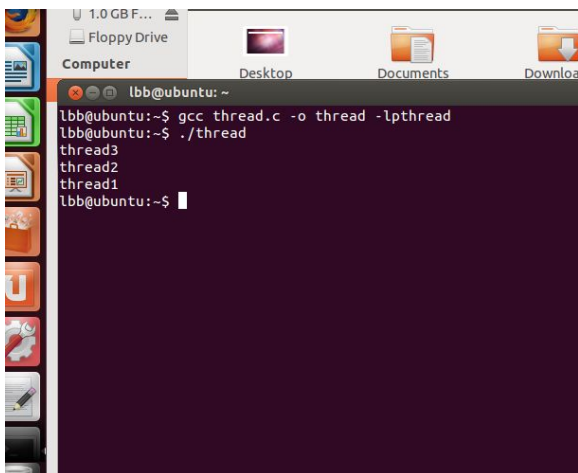
```
pthread_mutex_unlock(&mut);    //解除互斥锁
```

```
sleep(2);    //等待一段时间（三个线程分别设为 sleep（2）（3）（4））
```

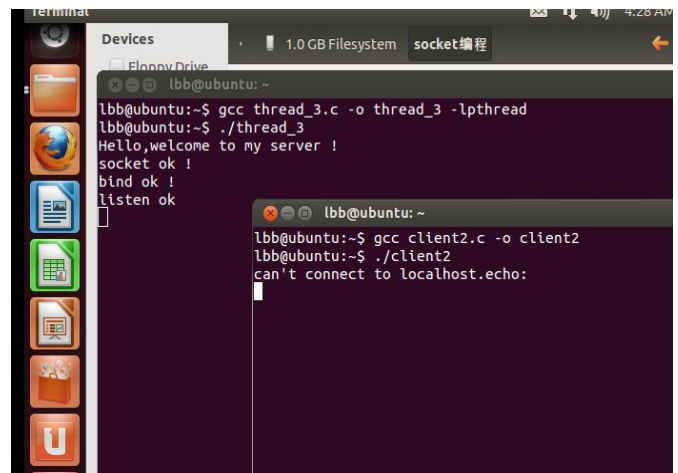
```
pthread_exit(NULL);    //终止线程
```

在 ubuntu 上运行的结果：

1) 单独测试建立线程：



2) 多线程的整体程序：



【遇到的问题】：

1. 单独建立多线程的程序运行正常，但在线程中加了 accept 和 write 后出现了不能连接到 localhost 的情况，我用 printf 函数打印了很多“test”字符，确定问题出在 accept 函数，那里就阻塞住了，它没有返回连接

成功建立的结果，而服务器端也因为没有客户端的请求到达，一直阻塞在 `listen` 那里。还没有解决，有待进一步调试。

2. 定义在头文件 `pthread.h` 里的 `pcreate` 函数参数列表是 `void`,即规定不能传参数，我把 `c_add,nfd,sfd` 传进线程里，发现程序报错，但如果让线程里完成于客户端的通信必须要传递参数。于是把这些参数设为全局变量，就解决了。不过不知道这样做安全性是否有影响。

3. 按照一般的编译命令 `gcc thread.c -o thread` 编译时，会出现 `pthread_create` 和 `pthread_join` 未被定义的报错，上网查的解决办法是在命令之后加 `-lpthread` ,相当与调用另一个文件，我想不知道为什么不把它写在 `include` 的头文件里。

4. 由于对命令行运行程序原理不明白，开始时在同一个命令窗口里敲服务器和客户端的程序，一直不知道哪错了，其实应该用不同的窗口收发数据。

【感受与收获】：

1. 从通信过程的流程图到具体的代码实现有较大的鸿沟，需要参考别人写过的代码。但经过实验，对网络编程有了很具体的认识，能读懂很多代码了。

2. 很多头文件里定义的结构体、函数等对我是透明的，只通过接口传参数，这个抽象的过程比较难掌握，需要反复熟悉。

3. 通过使用虚拟机和 `ubuntu` 软件的 `linux` 系统，对以前没接触过的 `linux` 系统有了初步了解，学会了使用命令行操作编译和运行程序。不过在 `ubuntu` 上我没有使用合适的编辑器和调试器，使得程序改起来特别不方便。老师建议将来使用 `codeblock` 或 `elipse`。

4. 简单学习了一点 `java` 的基本知识，通过配置 `java` 环境，改 `path` 变量，对计算机的运行过程有了了解。而且学习了 `java` 的面向对象的继承、重载等后，觉得 `java` 确实在网络编程方面有优势，不过掌握得还不好，这次先用 `c` 写。

【资料和帮助来源】：

注：因为初次接触需要一个学习的过程，所以从以下资料借鉴。程序经多次修改调试，不完全是抄的哈。

1. `linux` 下 `socket` 编程实例 http://blog.sina.com.cn/s/blog_4ad7c25401019qqb.html

来自：新浪博客 流星的 BLOG <http://blog.sina.com.cn/staratsky>

2. `linux` 下 `C` 语言多线程编程实例 <http://wenku.baidu.com/view/df137b3231126edb6f1a101e.html>

来自：百度文库

附：程序代码：

1.单线程客户端：

//本程序使用 `tcp` 协议进行通信，服务端进行监听，在收到客户端的连接后，发送数据给客户端；

//客户端在接受到数据后打印出来，然后关闭。

```
#include <stdlib.h>
```

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

```
#include <stdio.h>
```

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

```
#include <netinet/in.h>//定义了网络字节顺序的转换函数
```

```
#include <string.h>//定义了 bezero 函数
```

```
int main()
```

```
{
```

```
    int cfd;//cfd 为 socket 文件描述符
```

```
    int recbytes;
```

```
    int sin_size;
```

```
    char buffer[1024]={0};//设定 buffer 长度为 1024
```

```
    struct sockaddr_in s_add,c_add;//s_add 为服务器套接字,c_add 为客户机套接字
```

```
    /*struct sockaddr_in 结构类型是用来保存 socket 信息的：
```

```

    struct sockaddr_in
    {
        short int sin_family;
        unsigned short int sin_port;
        struct in_addr sin_addr;
        unsigned char sin_zero[8];    };
*/
unsigned short portnum=0x8888;
printf("Hello,welcome to client !\r\n");
cfd = socket(AF_INET, SOCK_STREAM, 0);
//调用成功，返回 socket 文件描述符；失败，返回-1，并设置 errno
//AF_INET 指明所使用的协议族，通常为 PF_INET，表示 Ipv4 网络协议
//SOCK_STREAM 指定 socket 的类型，基本上有三种：数据流套接字、数据报套接字、原始套接字
//protocol 通常赋值"0"
if(-1 == cfd){    printf("socket fail ! \r\n");
                return -1;}

printf("socket ok !\r\n");
bzero(&s_add,sizeof(struct sockaddr_in));//bzero 将套接字地址清 0
s_add.sin_family=AF_INET;
s_add.sin_addr.s_addr= inet_addr("192.168.1.2");
/*inet_addr 把一个用数字和点表示的 IP 地址的字符串转换成一个无符号长整型，成功时：返回转换结果，
失败时返回常量 INADDR_NONE，该常量=-1，二进制的无符号整数-1 相当于 255.255.255.255，这是一个广播地址 */
s_add.sin_port=htons(portnum);
//htons 将主机字节顺序转换成网络字节顺序，对无符号短型进行操作 4bytes
printf("s_addr = %#x ,port : %#x\r\n",s_add.sin_addr.s_addr,s_add.sin_port);
if(-1 == connect(cfd,(struct sockaddr *)&s_add, sizeof(struct sockaddr)))
// connect 用于客户端发送服务请求。成功返回 0，否则返回-1，并置 errno
//cfd 是 socket 函数返回的 socket 描述符，&s_add 是包含远端主机 IP 地址和端口号的指针，结构
sockaddr_in 的长度
{
    printf("connect fail !\r\n");
    return -1;}
printf("connect ok !\r\n");
if(-1 == (recbytes = read(cfd,buffer,1024)))
//read 从 cfd 中读取内容,当读成功时,read 返回实际所读的字节数,返回的值是 0 表示已经读到文件的结束了,小于 0 表示出现了错误.
{
    printf("read data fail !\r\n");
    return -1;}
printf("read ok\r\nREC:\r\n");
buffer[recbytes]='\0';
printf("%s\r\n",buffer);
getchar();
close(cfd);//close 释放该 socket,函数运行成功返回 0，否则返回-1
return 0;
}

```

2.单线程服务器:

//本程序只注释和 client 不同的地方

```
#include <stdlib.h>
```

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

```
#include <stdio.h>
```

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

```
#include <netinet/in.h>
```

```
#include <string.h>
```

```
int main(){
```

```
    int sfp,nfp;
```

```
    struct sockaddr_in s_add,c_add;
```

```
    int sin_size;
```

```
    unsigned short portnum=0x8888;
```

```
    printf("Hello,welcome to my server !\r\n");
```

```
    sfp = socket(AF_INET, SOCK_STREAM, 0);
```

```
    if(-1 == sfp)
```

```
    {    printf("socket fail ! \r\n");
```

```
        return -1;}
```

```
    printf("socket ok !\r\n");
```

```
    bzero(&s_add,sizeof(struct sockaddr_in));
```

```
    s_add.sin_family=AF_INET;
```

```
    s_add.sin_addr.s_addr=htonl(INADDR_ANY);
```

```
    s_add.sin_port=htons(portnum);
```

```
    if(-1 == bind(sfp,(struct sockaddr *)&s_add, sizeof(struct sockaddr)))
```

//bind 将套接字和指定的端口相连。成功返回 0，否则，返回-1，并置 errno

//sfp 是调用 socket 函数返回值，&s_add 是一个指向包含有本机 IP 地址及端口号等信息的 sockaddr 类型的指针

```
    {    printf("bind fail !\r\n");
```

```
        return -1;}
```

```
    printf("bind ok !\r\n");
```

```
    if(-1 == listen(sfp,5))
```

// listen 等待指定的端口的出现客户端连接，调用成功返回 0，否则，返回-1，并置 errno

//sfp 是 socket()函数返回值,5 指定在请求队列中允许的最大请求数

```
    {    printf("listen fail !\r\n");
```

```
        return -1;}
```

```
    printf("listen ok\r\n");
```

```
    while(1){sin_size = sizeof(struct sockaddr_in);
```

```
    nfp = accept(sfp, (struct sockaddr *)&c_add, &sin_size);
```

// accept 用于接受客户端的服务请求，成功返回新的套接字描述符，失败返回-1，并置 errno

//sfp 是被监听的 socket 描述符,&c_add 是一个指向客户机套接字的指针,&sin_size 是结构 sockaddr_in 的长度

```
    if(-1 == nfp)
```

```
    {    printf("accept fail !\r\n");
```

```
        return -1;}
```

```
    printf("accept      ok!\r\nServer      start      get      connect      from      %#x      :
```

```
%#x\r\n",ntohl(c_add.sin_addr.s_addr),ntohs(c_add.sin_port));
```

//ntohl 网络字节顺序转换成主机字节顺序，对无符号长型进行操作 8bytes,ntohs 对无符号短型进行

操作 4bytes

```
        if(-1 == write(nfp,"hello,welcome to my server \r\n",32))
            //write 函数将 buf 中的 nbytes 字节内容写入文件描述符 fd,成功时返回写的字节数,失败时返回-1, 并
            设置 errno 变量
        {
            printf("write fail!\r\n");
            return -1;}
        printf("write ok!\r\n");
        close(nfp);
    }
    close(sfp);
    return 0;
}
```

3.多线程服务器:

```
#include <pthread.h>
#include <stdio.h>
#include <sys/time.h>
#include <string.h>
#include <netinet/in.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
```

```
#define MAX 10
```

```
pthread_t thread[3];
pthread_mutex_t mut;
int number=0, i;
```

```
void *thread1(struct sockaddr_in c_add,int sfd,int nfd)
{
    printf("thread1 :Hi!God! I'm thread 1\n");
    for (i = 0; i < MAX; i++)
    {
        printf("thread1 : number = %d\n",number);
        pthread_mutex_lock(&mut);
        number++;

        if(-1 == write(nfd,"hello,welcome to my server \r\n",32))
        {
            printf("write fail!\r\n");
            pthread_exit(NULL);}
        printf("write ok!\r\n");

        close(nfd);
        pthread_mutex_unlock(&mut);
        sleep(2);
    }
}
```

```

        printf("thread1 :已完成! \n");
        pthread_exit(NULL);
    }

void *thread2(struct sockaddr_in c_add,int sfd,int nfd)
{
    printf("thread2 :Hi!God! I'm thread 2\n");
    for (i = 0; i < MAX; i++)
    {
        printf("thread2 : number = %d\n",number);
        pthread_mutex_lock(&mut);
            number++;
        if(-1 == write(nfd,"hello,welcome to my server \r\n",32))
        {
            printf("write fail!\r\n");
            pthread_exit(NULL);}
        printf("write ok!\r\n");

        close(nfd);
        pthread_mutex_unlock(&mut);
        sleep(3);
    }
    printf("thread2 :已完成! \n");
    pthread_exit(NULL);
}

void *thread3(struct sockaddr_in c_add,int sfd,int nfd)
{
    printf("thread3 :Hi!God! I'm thread 3\n");

    for (i = 0; i < MAX; i++)
    {
        printf("thread3 : number = %d\n",number);
        pthread_mutex_lock(&mut);
            number++;
        if(-1 == write(nfd,"hello,welcome to my server \r\n",32))
        {
            printf("write fail!\r\n");
            pthread_exit(NULL);}
        printf("write ok!\r\n");

        close(nfd);
        pthread_mutex_unlock(&mut);
        sleep(4);
    }
    printf("thread3 :已完成! \n");
    pthread_exit(NULL);
}

void thread_create(sockaddr_in c_add,int sfd,int nfd)

```

```

{
    int temp,sin_size;
    sin_size = sizeof(struct sockaddr_in);
    memset(&thread, 0, sizeof(thread));

    if((temp = pthread_create(&thread[0], NULL, thread1, NULL))
        printf("线程 1 创建失败");
    else
        printf("线程 1 被创建\n");

    if((temp = pthread_create(&thread[1], NULL, thread1,NULL))
        printf("线程 2 创建失败");
    else
        printf("线程 2 被创建\n");

    if((temp = pthread_create(&thread[2], NULL, thread3,NULL))
        printf("线程 3 创建失败");
    else
        printf("线程 3 被创建\n");
}

```

```

void thread_wait(pthread_t* thread)
{
    if(thread[0] !=0) {
        pthread_join(thread[0],NULL);

        printf("线程 1 已经结束\n");
    }
    if(thread[1] !=0) {
        pthread_join(thread[1],NULL);
        printf("线程 2 已经结束\n");
    }
    if(thread[2] !=0) {
        pthread_join(thread[2],NULL);
        printf("线程 3 已经结束\n");
    }
}

```

```

int main()
{
    int sfd,nfd,temp;
    struct sockaddr_in s_add,c_add;
    int sin_size;
    unsigned short portnum=0x8888;
    printf("Hello,welcome to my server !\r\n");

    sfd = socket(AF_INET, SOCK_STREAM, 0);
    if(-1 == sfd)

```

```

{    printf("socket fail ! \r\n");
    return -1;}
printf("socket ok !\r\n");

bzero(&s_add,sizeof(struct sockaddr_in));
s_add.sin_family=AF_INET;
s_add.sin_addr.s_addr=htonl(INADDR_ANY);
s_add.sin_port=htons(portnum);

if(-1 == bind(sfd,(struct sockaddr *)&s_add, sizeof(struct sockaddr_in)))
{    printf("bind fail !\r\n");
    return -1;}
printf("bind ok !\r\n");

if(-1 == listen(sfd,5))
{    printf("listen fail !\r\n");
    return -1;}
printf("listen ok\r\n");

pthread_mutex_init(&mut,NULL);

while(1){
    if(-1!=(nfd = accept(sfd, (struct sockaddr *)&c_add, &sin_size)))
    {
        temp= thread_create(c_add,sfd,nfd);
        printf("accept    ok!\r\nServer    start    get    connect    from    %#x    :
%#x\r\n",ntohl(c_add.sin_addr.s_addr),ntohs(c_add.sin_port));
    }
    else    printf("accept fail !\r\n");

    if(temp!= 0)
        printf("线程创建失败!\n");
    else
        printf("线程被创建\n");

    thread_wait(thread);

}
close(sfd);
return 0;
}

```