Linux 网络编程一步一步学

编辑：刘杰

日期：2009-08-12

于：无锡矽太恒科

Linux 网络编程一步一步学-简单客户端编写

关键词： Linux 网络 socket 编程 program

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/socket.h>
#include <resolv.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define MAXBUF 1024
/************关于本文档********************************************
*filename: simple-socket.c
*purpose: 演示最基本的网络编程步骤，这是个客户端程序
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-23 19:41:54
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*******************************************************************/
int main(int argc, char **argv)
{
int sockfd;
struct sockaddr_in dest;
char buffer[MAXBUF];

if (argc != 3) {
printf
("参数格式错误！正确用法如下：\n\t\t%s IP 地址 端口\n\t 比如:\t%s 127.0.0.1 80\n 此程序
用来从某个 IP 地址的服务器某个端口接收最多 MAXBUF 个字节的消息",
argv[0], argv[0]);
exit(0);
}
/* 创建一个 socket 用于 tcp 通信 */
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
perror("Socket");
exit(errno);
```

```
}

/* 初始化服务器端（对方）的地址和端口信息 */
bzero(&dest, sizeof(dest));
dest.sin_family = AF_INET;
dest.sin_port = htons(atoi(argv[2]));
if (inet_aton(argv[1], (struct in_addr *) &dest.sin_addr.s_addr) == 0) {
perror(argv[1]);
exit(errno);
}

/* 连接服务器 */
if (connect(sockfd, (struct sockaddr *) &dest, sizeof(dest)) != 0) {
perror("Connect ");
exit(errno);
}

/* 接收对方发过来的消息，最多接收 MAXBUF 个字节 */
bzero(buffer, MAXBUF);
recv(sockfd, buffer, sizeof(buffer), 0);
printf("%s", buffer);

/* 关闭连接 */
close(sockfd);
return 0;
}
```

编译此程序使用如下命令：

```
gcc -Wall simple-socket.c
```

运行此程序使用如下命令(假设你的主机上开启了 ssh 服务)：

```
./a.out 127.0.0.1 22
```

Linux 网络编程一步一步学-绑定 IP 和端口

关键词： socket　　bind　　port　　端口　　绑定

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/socket.h>
#include <resolv.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define MAXBUF 1024
/***********关于本文档*****************************************
*filename: simple-bind.c
*purpose: 演示最基本的网络编程步骤，这是个客户端程序以固定 IP 和端口连接服务器
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-23 19:51:54
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*********************************************************************/
int main(int argc, char **argv)
{
int sockfd;
struct sockaddr_in dest, mine;
char buffer[MAXBUF];

if (argc != 5) {
printf
("参数格式错误！正确用法如下：\n\t\t%s 对方 IP 地址 对方端口 本机 IP 地址 本机端口\n\t 比如:
\t%s 127.0.0.1 80\n 此程序用来以本机固定的端口从某个 IP 地址的服务器某个端口接收最多
MAXBUF 个字节的消息",
argv[0], argv[0]);
exit(0);
}
/* 创建一个 socket 用于 tcp 通信 */
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
perror("Socket");
```

```
exit(errno);
}

/* 初始化服务器端（对方）的地址和端口信息 */
bzero(&dest, sizeof(dest));
dest.sin_family = AF_INET;
dest.sin_port = htons(atoi(argv[2]));
if (inet_aton(argv[1], (struct in_addr *) &dest.sin_addr.s_addr) == 0) {
perror(argv[1]);
exit(errno);
}

/* 初始化自己的地址和端口信息 */
bzero(&mine, sizeof(mine));
mine.sin_family = AF_INET;
mine.sin_port = htons(atoi(argv[4]));
if (inet_aton(argv[3], (struct in_addr *) &mine.sin_addr.s_addr) == 0) {
perror(argv[3]);
exit(errno);
}

/* 把自己的 IP 地址信息和端口与 socket 绑定 */
if (bind(sockfd, (struct sockaddr *) &mine, sizeof(struct sockaddr)) == -1) {
perror(argv[3]);
exit(errno);
}

/* 连接服务器 */
if (connect(sockfd, (struct sockaddr *) &dest, sizeof(dest)) != 0) {
perror("Connect ");
exit(errno);
}

/* 接收对方发过来的消息，最多接收 MAXBUF 个字节 */
bzero(buffer, MAXBUF);
recv(sockfd, buffer, sizeof(buffer), 0);
printf("%s", buffer);
sleep(10);
/* 关闭连接 */
close(sockfd);
return 0;
}
```

编译程序用此命令：

```
gcc -Wall simple-bind.c
```

运行程序用此命令：

```
./a.out 127.0.0.1 22 127.0.0.1 3000
```

同时可以用下列 netstat 命令查看网络连接状态：

```
netstat -an|grep 3000
```

查看到如下信息：
tcp 0 0 127.0.0.1:3000 127.0.0.1:22 ESTABLISHED

Linux 网络编程一步一步学-循环读取服务器上的数据

关键词： Linux 网络 socket 编程 循环

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/socket.h>
#include <resolv.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define MAXBUF 10
/************关于本文档*********************************************
*filename: simple-readall.c
*purpose: 演示最基本的网络编程，循环读取服务器上发过来的内容，直到读完为止
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-23 20:16:54
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*********************************************************************/
int main(int argc, char **argv)
{
int sockfd, ret;
struct sockaddr_in dest, mine;
char buffer[MAXBUF + 1];

if (argc != 5) {
printf
("参数格式错误！正确用法如下：\n\t\t%s 对方 IP 地址 对方端口 本机 IP 地址 本机端口\n\t 比如:
\t%s 127.0.0.1 80\n 此程序用来以本机固定的端口从某个 IP 地址的服务器某个端口接收最多
MAXBUF 个字节的消息",
argv[0], argv[0]);
exit(0);
}
/* 创建一个 socket 用于 tcp 通信 */
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
perror("Socket");
exit(errno);
```

```
}

/* 初始化服务器端（对方）的地址和端口信息 */
bzero(&dest, sizeof(dest));
dest.sin_family = AF_INET;
dest.sin_port = htons(atoi(argv[2]));
if (inet_aton(argv[1], (struct in_addr *) &dest.sin_addr.s_addr) == 0) {
perror(argv[1]);
exit(errno);
}

/* 初始化自己的地址和端口信息 */
bzero(&mine, sizeof(mine));
mine.sin_family = AF_INET;
mine.sin_port = htons(atoi(argv[4]));
if (inet_aton(argv[3], (struct in_addr *) &mine.sin_addr.s_addr) == 0) {
perror(argv[3]);
exit(errno);
}

/* 把自己的 IP 地址信息和端口与 socket 绑定 */
if (bind(sockfd, (struct sockaddr *) &mine, sizeof(struct sockaddr)) == -1) {
perror(argv[3]);
exit(errno);
}

/* 连接服务器 */
if (connect(sockfd, (struct sockaddr *) &dest, sizeof(dest)) != 0) {
perror("Connect ");
exit(errno);
}

/* 接收对方发过来的消息，每次最多接收 MAXBUF 个字节，直到把对方发过来的所有消息接收完毕为止 */
do {
bzero(buffer, MAXBUF + 1);
ret = recv(sockfd, buffer, MAXBUF, 0);
printf("读到%d 个字节，它们是:'%s'\n", ret, buffer);
}while(ret==MAXBUF);

/* 关闭连接 */
close(sockfd);
return 0;
}
```

编译程序使用如下命令：

```
gcc -Wall simple-readall.c
```

运行程序用如下命令：

```
./a.out 127.0.0.1 22 127.0.0.1 3000
```

此程序运行结果如下：

```
读到 10 个字节，它们是:'SSH-2.0-Op'
读到 10 个字节，它们是:'enSSH_4.3p'
读到 10 个字节，它们是:'2 Debian-5'
读到 8 个字节，它们是:'ubuntu1
'
```

注意：如果你运行时程序长久没有退出，请把程序的第一行：
#define MAXBUF 10
稍微改一下，比如改成下面的：
#define MAXBUF 9
再编译运行试试。
为什么？请继续看下一篇文章“设置非阻塞方式”。

Linux 网络编程一步一步学-设置非阻塞方式

关键词： Linux 网络 socket 非阻塞 nonblock

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/socket.h>
#include <resolv.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <fcntl.h>

#define MAXBUF 10
/************关于本文档*********************************************
*filename: simple-nonblock.c
*purpose: 演示最基本的网络编程，循环读取服务器上发过来的内容，直到读完为止
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-23 20:46:54
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*********************************************************************/
int main(int argc, char **argv)
{
int sockfd, ret, rcvtm = 0;
struct sockaddr_in dest, mine;
char buffer[MAXBUF + 1];

if (argc != 5) {
printf
("参数格式错误！正确用法如下：\n\t\t%s 对方 IP 地址 对方端口 本机 IP 地址 本机端口\n\t 比如:
\t%s 127.0.0.1 80\n 此程序用来以本机固定的端口从某个 IP 地址的服务器某个端口接收最多
MAXBUF 个字节的消息",
argv[0], argv[0]);
exit(0);
}

/* 创建一个 socket 用于 tcp 通信 */
```

```
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
perror("Socket");
exit(errno);
}

/* 初始化服务器端（对方）的地址和端口信息 */
bzero(&dest, sizeof(dest));
dest.sin_family = AF_INET;
dest.sin_port = htons(atoi(argv[2]));
if (inet_aton(argv[1], (struct in_addr *) &dest.sin_addr.s_addr) == 0) {
perror(argv[1]);
exit(errno);
}

/* 初始化自己的地址和端口信息 */
bzero(&mine, sizeof(mine));
mine.sin_family = AF_INET;
mine.sin_port = htons(atoi(argv[4]));
if (inet_aton(argv[3], (struct in_addr *) &mine.sin_addr.s_addr) == 0) {
perror(argv[3]);
exit(errno);
}

/* 把自己的 IP 地址信息和端口与 socket 绑定 */
if (bind(sockfd, (struct sockaddr *) &mine, sizeof(struct sockaddr)) == -1) {
perror(argv[3]);
exit(errno);
}

/* 连接服务器 */
if (connect(sockfd, (struct sockaddr *) &dest, sizeof(dest)) != 0) {
perror("Connect ");
exit(errno);
}

/* 设置 socket 属性为非阻塞方式 */
if(fcntl(sockfd, F_SETFL, O_NONBLOCK) == -1) {
perror("fcntl");
exit(errno);
}

/* 接收对方发过来的消息，每次最多接收 MAXBUF 个字节，直到把对方发过来的所有消息接收完毕为
止 */
do {
```

```
_retry:
bzero(buffer, MAXBUF + 1);
ret = recv(sockfd, buffer, MAXBUF, 0);
if(ret > 0)
printf("读到%d 个字节, 它们是:'%s'\n", ret, buffer);

if(ret < 0) {
if(errno == EAGAIN) {
if(rcvtm)
break;
else {
printf("数据还未到达! \n");
usleep(100000);
goto _retry;
};
};
printf("接收出错了! \n");
perror("recv");
}
rcvtm++;
}while(ret==MAXBUF);

/* 关闭连接 */
close(sockfd);
return 0;
}
```

编译程序用下列命令:

```
gcc -Wall simple-nonblock.c
```

运行程序用下列命令:

```
./a.out 127.0.0.1 21 127.0.0.1 3000
```

程序运行输出结果如下:

```
数据还未到达!
读到 10 个字节, 它们是:'220 (vsFTP'
读到 10 个字节, 它们是:'d 2.0.4)
'
```

问题：

1、非阻塞是什么?
网络通信有阻塞和非阻塞之分，例如对于接收数据的函数recv：在阻塞方式下，没有数据到达时，即接收不到数据时，程序会停在recv函数这里等待数据的到来；而在非阻塞方式下就不会等，如果没有数据可接收就立即返回-1表示接收失败。
2、什么是errno?
errno是Linux系统下保存当前状态的一个公共变量，当前程序运行时进行系统调用如果出错，则会设置errno为某个值告诉用户出了什么错误。可以用printf("%d %s\n", errno, strerror(errno));得到具体信息。
3、什么是EAGAIN?
man recv
当recv系统调用返回这个值时表示recv读数据时，对方没有发送数据过来。

Linux 网络编程一步一步学-开启网络监听服务

关键词： Linux 网络 socket listen 监听

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <arpa/inet.h>

/************关于本文档*********************************************
*filename: simple-listen.c
*purpose: 演示最基本的网络编程步骤，开启服务端的监听，等待客户端来连接
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-24 12:31:00
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*********************************************************************/

int main(int argc, char **argv)
{
int sockfd;
struct sockaddr_in my_addr;
unsigned int myport, lisnum;

if (argv[1])
myport = atoi(argv[1]);
else
myport = 7838;

if (argv[2])
lisnum = atoi(argv[2]);
else
lisnum = 2;
```

```
if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
perror("socket");
exit(1);
}
else printf("socket created\n");

bzero(&my_addr, sizeof(my_addr));
my_addr.sin_family = PF_INET;
my_addr.sin_port = htons(myport);
if(argv[3]) my_addr.sin_addr.s_addr = inet_addr(argv[3]);
else my_addr.sin_addr.s_addr = INADDR_ANY;

if (bind(sockfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr)) == -1)
{
perror("bind");
exit(1);
}
else printf("binded\n");

if (listen(sockfd, lisnum) == -1) {
perror("listen");
exit(1);
}
else printf("begin listen\n");

sleep(100);
close(sockfd);
return 0;
}
```

编译程序用下列命令：

```
gcc -Wall simple-listen.c
```

运行程序用如下命令：

```
./a.out 7838 2
```

这将在你自己主机的所有 IP 地址是等待客户端连接。比如你的网卡 lo 的 IP 地址 127.0.0.1，又比如你的网卡 eth0 的 IP 地址 192.168.0.100
如果要指定只在某个地址是开启监听服务，可以用下面的命令：

```
./a.out 7838 2 127.0.0.1
```

这样，客户端只能通过 127.0.0.1 的 7838 端口连接你的程序。
你可以开启一个终端输入 telnet 127.0.0.1 7838 来测试是否能连接成功。
同时可以用 netstat -an|grep 7838 命令来查看网络是否连接正常

Linux 网络编程一步一步学-接受客户端连接请求

关键词： Linux 网络 socket accept 接受

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <arpa/inet.h>

/***********关于本文档*****************************************
*filename: simple-accept.c
*purpose: 演示最基本的网络编程步骤，开启服务端的监听，并接收每个客户端的连接请求
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-24 12:41
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*********************************************************************/

int main(int argc, char **argv)
{
int sockfd, new_fd;
socklen_t len;
struct sockaddr_in my_addr, their_addr;
unsigned int myport, lisnum;

if (argv[1])
myport = atoi(argv[1]);
else
myport = 7838;

if (argv[2])
lisnum = atoi(argv[2]);
else
lisnum = 2;
```

```
if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
perror("socket");
exit(1);
}
else printf("socket created\n");

bzero(&my_addr, sizeof(my_addr));
my_addr.sin_family = PF_INET;
my_addr.sin_port = htons(myport);
if(argv[3]) my_addr.sin_addr.s_addr = inet_addr(argv[3]);
else my_addr.sin_addr.s_addr = INADDR_ANY;

if (bind(sockfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr)) == -1)
{
perror("bind");
exit(1);
}
else printf("binded\n");

if (listen(sockfd, lisnum) == -1) {
perror("listen");
exit(1);
}
else printf("begin listen\n");

while(1) {
len = sizeof(struct sockaddr);
if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &len)) == -1) {
perror("accept");
exit(errno);
}
else printf("server: got connection from %s, port %d, socket
%d\n",inet_ntoa(their_addr.sin_addr), ntohs(their_addr.sin_port), new_fd);
}

close(sockfd);
return 0;
}
```

编译程序用下列命令:

```
gcc -Wall simple-accept.c
```

运行程序用如下命令：

```
./a.out 7838 1 127.0.0.1
```

另外开多几个客户端程序连接上来，
程序输出结果如下：

```
server: got connection from 127.0.0.1, port 15949, socket 4
server: got connection from 127.0.0.1, port 15950, socket 5
server: got connection from 127.0.0.1, port 15951, socket 6
server: got connection from 127.0.0.1, port 15952, socket 7
```

Linux 网络编程一步一步学-向客户端发送消息

关键词： Linux 网络 socket send 发送消息

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MAXBUF 1024
/***********关于本文档***********************************************
*filename: simple-send.c
*purpose: 演示最基本的网络编程步骤，开启服务接收客户端连接并向客户端发送消息
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-24 13:00
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*********************************************************************/

int main(int argc, char **argv)
{
int sockfd, new_fd;
socklen_t len;
struct sockaddr_in my_addr, their_addr;
unsigned int myport, lisnum;
char buf[MAXBUF + 1];

if (argv[1])
myport = atoi(argv[1]);
else
myport = 7838;

if (argv[2])
lisnum = atoi(argv[2]);
```

```
else
lisnum = 2;

if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
perror("socket");
exit(1);
}
else printf("socket created\n");

bzero(&my_addr, sizeof(my_addr));
my_addr.sin_family = PF_INET;
my_addr.sin_port = htons(myport);
if(argv[3]) my_addr.sin_addr.s_addr = inet_addr(argv[3]);
else my_addr.sin_addr.s_addr = INADDR_ANY;

if (bind(sockfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr)) == -1)
{
perror("bind");
exit(1);
}
else printf("binded\n");

if (listen(sockfd, lisnum) == -1) {
perror("listen");
exit(1);
}
else printf("begin listen\n");

while(1) {
len = sizeof(struct sockaddr);
if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &len)) == -1) {
perror("accept");
exit(errno);
}
else printf("server: got connection from %s, port %d, socket
%d\n",inet_ntoa(their_addr.sin_addr), ntohs(their_addr.sin_port), new_fd);

/* 开始处理每个新连接上的数据收发 */
bzero(buf, MAXBUF + 1);
strcpy(buf, "这是在连接建立成功后向客户端发送的第一个消息\n 只能向 new_fd 这个用 accept 函
数新建立的 socket 发消息，不能向 sockfd 这个监听 socket 发送消息，监听 socket 不能用来接收或
发送消息\n");
len = send(new_fd, buf, strlen(buf), 0);
if(len < 0) {
```

```
printf("消息'%s'发送失败！错误代码是%d，错误消息是'%s'\n", buf, errno,
strerror(errno));
}
else printf("消息'%s'发送成功，共发送了%d个字节！\n", buf, len);
/* 处理每个新连接上的数据收发结束 */
}

close(sockfd);
return 0;
}
```

编译程序用下列命令：

```
gcc -Wall simple-send.c
```

运行程序用如下命令：

```
./a.out 127.0.0.1 7838 1
```

程序输出结果如下：

```
socket created
binded
begin listen
server: got connection from 127.0.0.1, port 13097, socket 4
消息'这是在连接建立成功后向客户端发送的第一个消息
只能向new_fd这个用accept函数新建立的socket发消息，不能向sockfd这个监听socket发送消息，监听socket不能用来接收或发送消息
'发送成功，共发送了227个字节！
server: got connection from 127.0.0.1, port 13099, socket 5
消息'这是在连接建立成功后向客户端发送的第一个消息
只能向new_fd这个用accept函数新建立的socket发消息，不能向sockfd这个监听socket发送消息，监听socket不能用来接收或发送消息
'发送成功，共发送了227个字节！
```

客户端运行如下命令：

```
telnet 127.0.0.1 7838
```

客户端得到输出如下：

```
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
```

这是在连接建立成功后向客户端发送的第一个消息
只能向 new_fd 这个用 accept 函数新建立的 socket 发消息，不能向 sockfd 这个监听 socket 发送消息，监听 socket 不能用来接收或发送消息
Connection closed by foreign host.

Linux 网络编程一步一步学-客户端和服务器端互相收发消息

关键词： Linux 网络 socket send recv

服务器端源代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MAXBUF 1024
/************关于本文档********************************************
*filename: simple-accept.c
*purpose: 演示最基本的网络编程步骤，开启服务接收客户端连接并和客户端通信，互相收发消息
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-24 13:26
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*********************************************************************/

int main(int argc, char **argv)
{
int sockfd, new_fd;
socklen_t len;
struct sockaddr_in my_addr, their_addr;
unsigned int myport, lisnum;
char buf[MAXBUF + 1];

if (argv[1])
myport = atoi(argv[1]);
else
myport = 7838;
```

```
if (argv[2])
lisnum = atoi(argv[2]);
else
lisnum = 2;

if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
perror("socket");
exit(1);
}
else printf("socket created\n");

bzero(&my_addr, sizeof(my_addr));
my_addr.sin_family = PF_INET;
my_addr.sin_port = htons(myport);
if(argv[3]) my_addr.sin_addr.s_addr = inet_addr(argv[3]);
else my_addr.sin_addr.s_addr = INADDR_ANY;

if (bind(sockfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr)) == -1)
{
perror("bind");
exit(1);
}
else printf("binded\n");

if (listen(sockfd, lisnum) == -1) {
perror("listen");
exit(1);
}
else printf("begin listen\n");

while(1) {
len = sizeof(struct sockaddr);
if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &len)) == -1) {
perror("accept");
exit(errno);
}
else printf("server: got connection from %s, port %d, socket
%d\n",inet_ntoa(their_addr.sin_addr), ntohs(their_addr.sin_port), new_fd);

/* 开始处理每个新连接上的数据收发 */
bzero(buf, MAXBUF + 1);
strcpy(buf, "这是在连接建立成功后向客户端发送的第一个消息\n 只能向 new_fd 这个用 accept 函
数新建立的 socket 发消息，不能向 sockfd 这个监听 socket 发送消息，监听 socket 不能用来接收或
```

```
发送消息\n");
/* 发消息给客户端 */
len = send(new_fd, buf, strlen(buf), 0);
if(len < 0) {
printf("消息'%s'发送失败！错误代码是%d，错误信息是'%s'\n", buf, errno,
strerror(errno));
}
else printf("消息'%s'发送成功，共发送了%d 个字节！\n", buf, len);

bzero(buf, MAXBUF + 1);
/* 接收客户端的消息 */
len = recv(new_fd, buf, MAXBUF, 0);
if(len > 0) printf("接收消息成功:'%s'，共%d 个字节的数据\n", buf, len);
else printf("消息接收失败！错误代码是%d，错误信息是'%s'\n", errno, strerror(errno));
/* 处理每个新连接上的数据收发结束 */
}

close(sockfd);
return 0;
}
```

客户端源代码如下：

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/socket.h>
#include <resolv.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define MAXBUF 1024
/************关于本文档*********************************************
*filename: simple-socket.c
*purpose: 演示最基本的网络编程步骤，这是个客户端程序，与服务器互相收发消息
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-24 13:26
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
```

```
**********************************************************************/
int main(int argc, char **argv)
{
int sockfd, len;
struct sockaddr_in dest;
char buffer[MAXBUF + 1];

if (argc != 3) {
printf
("参数格式错误！正确用法如下：\n\t\t%s IP地址 端口\n\t 比如:\t%s 127.0.0.1 80\n此程序
用来从某个 IP 地址的服务器某个端口接收最多 MAXBUF 个字节的消息",
argv[0], argv[0]);
exit(0);
}
/* 创建一个 socket 用于 tcp 通信 */
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
perror("Socket");
exit(errno);
}
printf("socket created\n");

/* 初始化服务器端（对方）的地址和端口信息 */
bzero(&dest, sizeof(dest));
dest.sin_family = AF_INET;
dest.sin_port = htons(atoi(argv[2]));
if (inet_aton(argv[1], (struct in_addr *) &dest.sin_addr.s_addr) == 0) {
perror(argv[1]);
exit(errno);
}
printf("address created\n");

/* 连接服务器 */
if (connect(sockfd, (struct sockaddr *) &dest, sizeof(dest)) != 0) {
perror("Connect ");
exit(errno);
}
printf("server connected\n");

/* 接收对方发过来的消息，最多接收 MAXBUF 个字节 */
bzero(buffer, MAXBUF + 1);
/* 接收服务器来的消息 */
len = recv(sockfd, buffer, MAXBUF, 0);
if(len > 0) printf("接收消息成功:'%s'，共%d个字节的数据\n", buffer, len);
else printf("消息接收失败！错误代码是%d，错误信息是'%s'\n", errno, strerror(errno));
```

```
bzero(buffer, MAXBUF + 1);
strcpy(buffer, "这是客户端发给服务器端的消息\n");
/* 发消息给服务器 */
len = send(sockfd, buffer, strlen(buffer), 0);
if(len < 0) printf("消息'%s'发送失败！错误代码是%d，错误信息是'%s'\n", buffer,
errno, strerror(errno));
else printf("消息'%s'发送成功，共发送了%d 个字节！\n", buffer, len);

/* 关闭连接 */
close(sockfd);
return 0;
}
```

编译两个程序用下列命令：

```
gcc -Wall simple-server.c -o server
gcc -Wall simple-client.c -o client
```

启动服务端程序用如下命令：

```
./server 7838 1
```

启动客户端程序用如下命令：

```
./client 127.0.0.1 7838
```

服务端程序运行屏幕输出如下：

```
socket created
binded
begin listen
server: got connection from 127.0.0.1, port 32537, socket 4
消息'这是在连接建立成功后向客户端发送的第一个消息
只能向 new_fd 这个用 accept 函数新建立的 socket 发消息，不能向 sockfd 这个监听 socket 发送消
息，监听 socket 不能用来接收或发送消息
'发送成功，共发送了 227 个字节！
接收消息成功:'这是客户端发给服务器端的消息
'，共 43 个字节的数据
```

客户端程序运行屏幕输出如下：

```
socket created
address created
```

server connected
接收消息成功:'这是在连接建立成功后向客户端发送的第一个消息
只能向 new_fd 这个用 accept 函数新建立的 socket 发消息，不能向 sockfd 这个监听 socket 发送消息，监听 socket 不能用来接收或发送消息
'，共 227 个字节的数据
消息'这是客户端发给服务器端的消息
'发送成功，共发送了 43 个字节！

Linux 网络编程一步一步学-UDP 编程介绍

关键词： Linux UDP socket recvfrom sendto

通常我们在说到网络编程时默认是指 TCP 编程，即用前面提到的 socket 函数创建一个 socket 用于 TCP 通讯，函数参数我们通常填为 SOCK_STREAM。即 socket(PF_INET, SOCK_STREAM, 0)，这表示建立一个 socket 用于流式网络通讯。
通过查看 socket 的 man 手册可以看到 socket 函数的第一个参数的值可以为下面这些值：

Name	Purpose
PF_UNIX, PF_LOCAL	Local communication
PF_INET	IPv4 Internet protocols
PF_INET6	IPv6 Internet protocols
PF_IPX	IPX - Novell protocols
PF_NETLINK	Kernel user interface device
PF_X25	ITU-T X.25 / ISO-8208 protocol
PF_AX25	Amateur radio AX.25 protocol
PF_ATMPVC	Access to raw ATM PVCs
PF_APPLETALK	Appletalk
PF_PACKET	Low level packet interface

第二个参数支持下列几种值：

SOCK_STREAM
Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.

SOCK_DGRAM
Supports datagrams (connectionless, unreliable messages of a fixed maximum length).

SOCK_SEQPACKET
Provides a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer is required to read an entire packet with each read system call.

SOCK_RAW
Provides raw network protocol access.

SOCK_RDM
Provides a reliable datagram layer that does not guarantee ordering.

SOCK_PACKET

Obsolete and should not be used in new programs; see packet(7).

从这里可以看出，SOCK_STREAM 这种的特点是面向连接的，即每次收发数据之前必须通过 connect 建立连接，也是双向的，即任何一方都可以收发数据，协议本身提供了一些保障机制保证它是可靠的、有序的，即每个包按照发送的顺序到达接收方。
而 SOCK_DGRAM 这种是 User Datagram Protocol 协议的网络通讯，它是无连接的，不可靠的，因为通讯双方发送数据后不知道对方是否已经收到数据，是否正常收到数据。任何一方建立一个 socket 以后就可以用 sendto 发送数据，也可以用 recvfrom 接收数据。根本不关心对方是否存在，是否发送了数据。它的特点是通讯速度比较 快。大家都知道 TCP 是要经过三次握手的，而 UDP 没有。
基于上述不同，UDP 和 TCP 编程步骤也有些不同，如下：

```
/*****************************************************************
*filename: UDP 编程介绍
*purpose: 通过比较 UDP 和 TCP 编程对二者编程步骤进行总结说明
*tidied by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-24 20:12:00
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
*****************************************************************/
```

TCP 编程的服务器端一般步骤是：

1、创建一个 socket，用函数 socket()；
2、设置 socket 属性，用函数 setsockopt()； * 可选
3、绑定 IP 地址、端口等信息到 socket 上，用函数 bind()；
4、开启监听，用函数 listen()；
5、接收客户端上来的连接，用函数 accept()；
6、收发数据，用函数 send()和 recv()，者 read()和
write()；
7、关闭网络连接；
8、关闭监听；

TCP 编程的客户端一般步骤是：

1、创建一个 socket，用函数 socket()；
2、设置 socket 属性，用函数 setsockopt()；* 可选
3、绑定 IP 地址、端口等信息到 socket 上，用函数 bind()；* 可
选
4、设置要连接的对方的 IP 地址和端口等属性；
5、连接服务器，用函数 connect()；
6、收发数据，用函数 send()和 recv()，或者 read()和

write();
7、关闭网络连接;

与之对应的UDP编程步骤要简单许多，分别如下：

UDP编程的服务器端一般步骤是：

1、创建一个socket，用函数socket();
2、设置socket属性，用函数setsockopt();* 可选
3、绑定IP地址、端口等信息到socket上，用函数
bind();
4、循环接收数据，用函数recvfrom();
5、关闭网络连接;

UDP编程的客户端一般步骤是：

1、创建一个socket，用函数socket();
2、设置socket属性，用函数setsockopt();* 可选
3、绑定IP地址、端口等信息到socket上，用函数bind();* 可
选
4、设置对方的IP地址和端口等属性;
5、发送数据，用函数sendto();
6、关闭网络连接;

Linux 网络编程一步一步学-UDP 方式点对点通讯

关键词： Linux　　UDP　　单播　　recvfrom　　sendto

UDP 通讯服务器端源代码如下：

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <errno.h>
#include <stdlib.h>
#include <arpa/inet.h>

/*************************************************************************
*filename: simple-udpserver.c
*purpose: 基本编程步骤说明，演示了 UDP 编程的服务器端编程步骤
*tidied by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-24 20:22:00
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*************************************************************************/

int main(int argc, char **argv)
{
struct sockaddr_in s_addr;
struct sockaddr_in c_addr;
int sock;
socklen_t addr_len;
int len;
char buff[128];

/* 创建 socket ，关键在于这个 SOCK_DGRAM */
if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
perror("socket");
exit(errno);
} else
printf("create socket.\n\r");
```

```
memset(&s_addr, 0, sizeof(struct sockaddr_in));
/* 设置地址和端口信息 */
s_addr.sin_family = AF_INET;
if (argv[2])
s_addr.sin_port = htons(atoi(argv[2]));
else
s_addr.sin_port = htons(7838);
if (argv[1])
s_addr.sin_addr.s_addr = inet_addr(argv[1]);
else
s_addr.sin_addr.s_addr = INADDR_ANY;

/* 绑定地址和端口信息 */
if ((bind(sock, (struct sockaddr *) &s_addr, sizeof(s_addr))) == -1) {
perror("bind");
exit(errno);
} else
printf("bind address to socket.\n\r");

/* 循环接收数据 */
addr_len = sizeof(c_addr);
while (1) {
len = recvfrom(sock, buff, sizeof(buff) - 1, 0,
(struct sockaddr *) &c_addr, &addr_len);
if (len < 0) {
perror("recvfrom");
exit(errno);
}

buff[len] = '\0';
printf("收到来自%s:%d 的消息:%s\n\r",
inet_ntoa(c_addr.sin_addr), ntohs(c_addr.sin_port), buff);
}
return 0;
}
```

客户端源代码如下：

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <errno.h>
```

```
#include <stdlib.h>
#include <arpa/inet.h>

/*************************************************************************
*filename: simple-udpclient.c
*purpose: 基本编程步骤说明，演示了UDP编程的客户端编程步骤
*tidied by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux爱好者 Linux知识传播者 SOHO族 开发者 最擅长C语言
*date time:2007-01-24 21:22:00
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*************************************************************************/
int main(int argc, char **argv)
{
struct sockaddr_in s_addr;
int sock;
int addr_len;
int len;
char buff[128];

/* 创建 socket ，关键在于这个 SOCK_DGRAM */
if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
perror("socket");
exit(errno);
} else
printf("create socket.\n\r");

/* 设置对方地址和端口信息 */
s_addr.sin_family = AF_INET;
if (argv[2])
s_addr.sin_port = htons(atoi(argv[2]));
else
s_addr.sin_port = htons(7838);
if (argv[1])
s_addr.sin_addr.s_addr = inet_addr(argv[1]);
else {
printf("消息必须有一个接收者！\n");
exit(0);
}

/* 发送UDP消息 */
```

```
addr_len = sizeof(s_addr);
strcpy(buff, "hello i'm here");
len = sendto(sock, buff, strlen(buff), 0,
(struct sockaddr *) &s_addr, addr_len);
if (len < 0) {
printf("\n\rsend error.\n\r");
return 3;
}

printf("send success.\n\r");
return 0;
}
```

编译程序用下列命令：

```
gcc -Wall simple-udpserver.c -o server
gcc -Wall simple-udpclient.c -o client
```

运行程序用下列命令：

```
./server 127.0.0.1 7838

./client 127.0.0.1 7838
```

Linux 网络编程一步一步学-UDP 方式广播通讯

关键词： broadcast　UDP　广播　recvfrom　sendto

和前一篇文章＜Linux 网络编程一步一步学-UDP 方式点对点通讯＞一样，只是在客户端源代码里加一行设置 socket 属性为广播方式即可。

需要加的一句是：

```
setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &yes, sizeof(yes));
```

源代码变成下面的：

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <errno.h>
#include <stdlib.h>
#include <arpa/inet.h>

/*****************************************************************
*filename: broadc-udpclient.c
*purpose: 基本编程步骤说明，演示了 UDP 编程的广播客户端编程步骤
*tidied by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-24 21:30:00
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*****************************************************************/
int main(int argc, char **argv)
{
struct sockaddr_in s_addr;
int sock;
int addr_len;
int len;
char buff[128];
int yes;

/* 创建 socket */
if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
```

```
perror("socket");
exit(errno);
} else
printf("create socket.\n\r");

/* 设置通讯方式对广播，即本程序发送的一个消息，网络上所有主机均可以收到 */
yes = 1;
setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &yes, sizeof(yes));
/* 唯一变化就是这一点了 */

/* 设置对方地址和端口信息 */
s_addr.sin_family = AF_INET;
if (argv[2])
s_addr.sin_port = htons(atoi(argv[2]));
else
s_addr.sin_port = htons(7838);
if (argv[1])
s_addr.sin_addr.s_addr = inet_addr(argv[1]);
else {
printf("消息必须有一个接收者！\n");
exit(0);
}

/* 发送UDP消息 */
addr_len = sizeof(s_addr);
strcpy(buff, "hello i'm here");
len = sendto(sock, buff, strlen(buff), 0,
(struct sockaddr *) &s_addr, addr_len);
if (len < 0) {
printf("\n\rsend error.\n\r");
return 3;
}

printf("send success.\n\r");
return 0;
}
```

编译这个程序用下列命令：

```
gcc -Wall broadc-udpclient.c -o client
```

运行程序用下列命令：

```
./client 192.168.0.255 7838
```

就会往192.168.0网络内所有主机发消息。

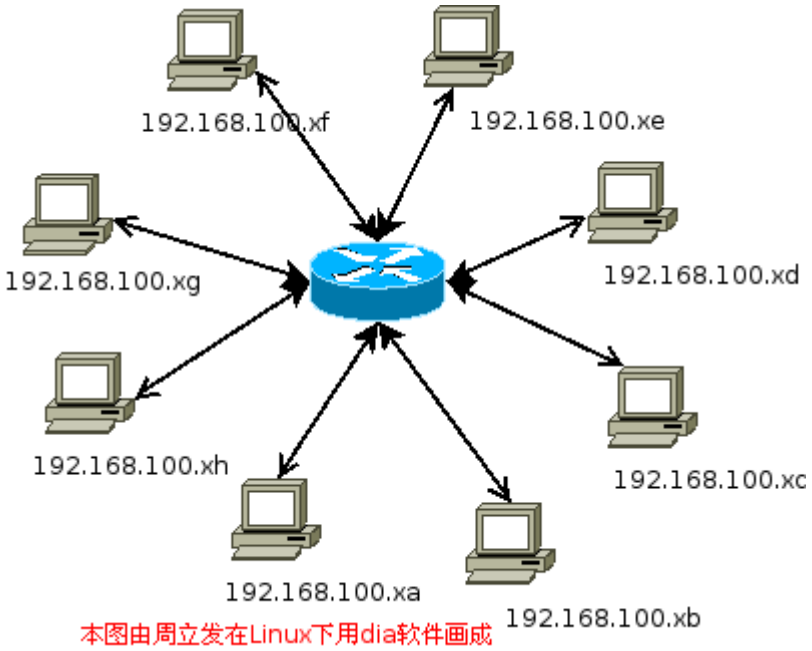
其它主机如果运行了服务端：

./server 自己的IP地址 7838 则都会收到上述客户端发的消息了。

Linux 网络编程一步一步学-网络广播、组播与单播

关键词： Linux　　网络　　单播　　广播　　组播

这里以下图所示的网络为基础来说明网络通讯的各种方式：



什么是广播?

以前面的文章<Linux 网络编程一步一步学-UDP 方式广播通讯>为例：就是用下列命令在上图所示的主机 192.168.100.xa 上运行客户端程序：
./client 192.168.100.255 7838
则 上图所示网络上的所有主机，只要其 IP 地址 192.168.100.*与网络掩码（比如 255.255.255.0）运算得到的子网（比如 192.168.100.0）与 192.168.100.xa 主机所在的子网是一样的，都会在自己的 7838 端口收到 192.168.xa 主机发出来的 UDP 消息。消息会被复制并发到每个主机的网卡上去，网卡收到消息后提交给操作系统去处理，操作系统发现有程序在 7838 端口接收 UDP 数据则把消息转给 相应的程序去处理，如果没有程序接收来自 7838 端口的 UDP 消息，则操作系统丢弃该消息。

```
/*************************************************************************
*filename: Linux 网络编程一步一步学-网络广播、组播与单播
*purpose: 说明网络广播、组播与单播
*tidied by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-25 13:10:00
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
*  但请遵循 GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
```

* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
**********/
广播的缺点：不管主机是否有程序接收广播消息，广播消息一定会被网卡收到并提交给操作系统去处理，所以会造成网络上流量增大，对不接收广播消息的主机造成一定的负担。

什么是单播?

以前面的文章<Linux 网络编程一步一步学-客户端和服务器端互相收发消息>为例：就是用下列命令在上图所示的主机 192.168.100.xa 上运行客户端程序：

```
./client 192.168.100.xf 7838
```

则消息只会从 192.168.100.xa 主机发到 192.168.100.xf 主机上，192.168.100.xf 主机的网卡收到消息后转给操作系统去处理，操作系统再把此消息转给相应程序去处理，如果没有程序处理就丢弃该包。

TCP 方式和 UDP 方式都可以实现单播。也是大多数情况下网络通讯所采取的方式。

什么是组播?

以后面的文章<Linux 网络编程一步一步学-UDP 组播>为例：就是用下列命令在上图所示的主机 192.168.100.xa 上运行客户端程序：

```
./mcastclient 230.1.1.1 7838
```

则 消息只会从 192.168.100.xa 主机发到加入了组 230.1.1.1 的主机的 7838 端口。象广播一样，组播消息一样会被复制发到网络所有主机的 网卡上，但只有宣布加入 230.1.1.1 这个组的主机的网卡才会把数据提交给操作系统去处理。如果没有加入组，则网卡直接将数据丢弃。
要想接收组播消息的主机必须运行命令加入组，如下方式：

```
./mcastserver 230.1.1.1 7838
```

注意：按照 RFC 规定，组播地址范围是 D 类 IP 地址，即 224.0.0.1-239.255.255.255。
组播 IP 地址不能用我们平时所有的 C 类 IP 地址。

Linux 网络编程一步一步学-UDP 组播

关键词： Linux UDP multicast recvfrom sendto

组播客户端代码如下：

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUFLEN 255
/*****************************************************************
*filename: mcastclient.c
*purpose: 演示组播编程的基本步骤，其实这就是一个基本的 UDP 客户端程序
*tidied by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-25 13:10:00
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*****************************************************************/
int main(int argc, char **argv)
{
struct sockaddr_in peeraddr, myaddr;

int sockfd;
char recmsg[BUFLEN + 1];
unsigned int socklen;

/* 创建 socket 用于 UDP 通讯 */
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd < 0) {
printf("socket creating error\n");
exit(1);
}
socklen = sizeof(struct sockaddr_in);

/* 设置对方的端口和 IP 信息 */
```

```
memset(&peeraddr, 0, socklen);
peeraddr.sin_family = AF_INET;
if (argv[2])
peeraddr.sin_port = htons(atoi(argv[2]));
else
peeraddr.sin_port = htons(7838);
if (argv[1]) {
/* 注意这里设置的对方地址是指组播地址，而不是对方的实际IP地址 */
if (inet_pton(AF_INET, argv[1], &peeraddr.sin_addr) <= 0) {
printf("wrong group address!\n");
exit(0);
}
} else {
printf("no group address!\n");
exit(0);
}

/* 设置自己的端口和IP信息 */
memset(&myaddr, 0, socklen);
myaddr.sin_family = AF_INET;
if (argv[4])
myaddr.sin_port = htons(atoi(argv[4]));
else
myaddr.sin_port = htons(23456);

if (argv[3]) {
if (inet_pton(AF_INET, argv[3], &myaddr.sin_addr) <= 0) {
printf("self ip address error!\n");
exit(0);
}
} else
myaddr.sin_addr.s_addr = INADDR_ANY;

/* 绑定自己的端口和IP信息到socket上 */
if (bind
(sockfd, (struct sockaddr *) &myaddr,
sizeof(struct sockaddr_in)) == -1) {
printf("Bind error\n");
exit(0);
}

/* 循环接受用户输入的消息发送组播消息 */
for (;;) {
/* 接受用户输入 */
```

```
bzero(recmsg, BUFLEN + 1);
if (fgets(recmsg, BUFLEN, stdin) == (char *) EOF)
exit(0);
/* 发送消息 */
if (sendto
(sockfd, recmsg, strlen(recmsg), 0,
(struct sockaddr *) &peeraddr,
sizeof(struct sockaddr_in)) < 0) {
printf("sendto error!\n");
exit(3);
}
printf("'%s' send ok\n", recmsg);
}
}
```

组播服务器端程序源代码为:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <errno.h>

#define BUFLEN 255
/*****************************************************************
*filename: mcastserver.c
*purpose: 演示组播编程的基本步骤，组播服务器端，关键在于加入组
*tidied by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux爱好者 Linux知识传播者 SOHO族 开发者 最擅长C语言
*date time:2007-01-25 13:20:00
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*****************************************************************/
int main(int argc, char **argv)
{
struct sockaddr_in peeraddr;
struct in_addr ia;
```

```
int sockfd;
char recmsg[BUFLEN + 1];
unsigne int socklen, n;
struct hostent *group;
struct ip_mreq mreq;

/* 创建 socket 用于UDP通讯 */
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd < 0) {
printf("socket creating err in udptalk\n");
exit(1);
}

/* 设置要加入组播的地址 */
bzero(&mreq, sizeof(struct ip_mreq));
if (argv[1]) {
if ((group = gethostbyname(argv[1])) == (struct hostent *) 0) {
perror("gethostbyname");
exit(errno);
}
} else {
printf
("you should give me a group address, 224.0.0.0-239.255.255.255\n");
exit(errno);
}

bcopy((void *) group->h_addr, (void *) &ia, group->h_length);
/* 设置组地址 */
bcopy(&ia, &mreq.imr_multiaddr.s_addr, sizeof(struct in_addr));

/* 设置发送组播消息的源主机的地址信息 */
mreq.imr_interface.s_addr = htonl(INADDR_ANY);

/* 把本机加入组播地址，即本机网卡作为组播成员，只有加入组才能收到组播消息 */
if (setsockopt
(sockfd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq,
sizeof(struct ip_mreq)) == -1) {
perror("setsockopt");
exit(-1);
}

socklen = sizeof(struct sockaddr_in);
memset(&peeraddr, 0, socklen);
peeraddr.sin_family = AF_INET;
```

```
if (argv[2])
peeraddr.sin_port = htons(atoi(argv[2]));
else
peeraddr.sin_port = htons(7838);
if (argv[1]) {
if (inet_pton(AF_INET, argv[1], &peeraddr.sin_addr) <= 0) {
printf("Wrong dest IP address!\n");
exit(0);
}
} else {
printf("no group address given, 224.0.0.0-239.255.255.255\n");
exit(errno);
}

/* 绑定自己的端口和IP信息到socket上 */
if (bind
(sockfd, (struct sockaddr *) &peeraddr,
sizeof(struct sockaddr_in)) == -1) {
printf("Bind error\n");
exit(0);
}

/* 循环接收网络上来的组播消息 */
for (;;) {
bzero(recmsg, BUFLEN + 1);
n = recvfrom(sockfd, recmsg, BUFLEN, 0,
(struct sockaddr *) &peeraddr, &socklen);
if (n < 0) {
printf("recvfrom err in udptalk!\n");
exit(4);
} else {
/* 成功接收到数据报 */
recmsg[n] = 0;
printf("peer:%s", recmsg);
}
}
}
```

编译程序用下列命令：

```
gcc -Wall mcastclient.c -o mcastclient
gcc -Wall mcastserver.c -o mcastserver
```

运行程序用如下命令：

```
./mcastserver 230.1.1.1 7838
```

客户端程序运行命令为：./mcastclient 230.1.1.1 7838 192.168.100.1 12345

Linux 网络编程一步一步学-同步聊天程序

关键词： Linux synchronous socket accept chat

服务器端源代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MAXBUF 1024
/************关于本文档*********************************************
*filename: sync-server.c
*purpose: 演示网络同步通讯，这是服务器端程序
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-25 20:26
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
********************************************************************/

int main(int argc, char **argv)
{
int sockfd, new_fd;
socklen_t len;
struct sockaddr_in my_addr, their_addr;
unsigned int myport, lisnum;
char buf[MAXBUF + 1];

if (argv[1])
myport = atoi(argv[1]);
else
myport = 7838;
```

```
if (argv[2])
lisnum = atoi(argv[2]);
else
lisnum = 2;

if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
perror("socket");
exit(1);
}

bzero(&my_addr, sizeof(my_addr));
my_addr.sin_family = PF_INET;
my_addr.sin_port = htons(myport);
if (argv[3])
my_addr.sin_addr.s_addr = inet_addr(argv[3]);
else
my_addr.sin_addr.s_addr = INADDR_ANY;

if (bind(sockfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr))
== -1) {
perror("bind");
exit(1);
}

if (listen(sockfd, lisnum) == -1) {
perror("listen");
exit(1);
}

while (1) {
printf
("\n----等待新的连接到来开始新一轮聊天……\n");
len = sizeof(struct sockaddr);
if ((new_fd =
accept(sockfd, (struct sockaddr *) &their_addr,
&len)) == -1) {
perror("accept");
exit(errno);
} else
printf("server: got connection from %s, port %d, socket %d\n",
inet_ntoa(their_addr.sin_addr),
ntohs(their_addr.sin_port), new_fd);
```

```
/* 开始处理每个新连接上的数据收发 */
while (1) {
bzero(buf, MAXBUF + 1);
printf("请输入要发送给对方的消息: ");
fgets(buf, MAXBUF, stdin);
if (!strncasecmp(buf, "quit", 4)) {
printf("自己请求终止聊天! \n");
break;
}
len = send(new_fd, buf, strlen(buf) - 1, 0);
if (len > 0)
printf
("消息:%s\t 发送成功, 共发送了%d 个字节! \n",
buf, len);
else {
printf
("消息'%s'发送失败! 错误代码是%d, 错误信息是'%s'\n",
buf, errno, strerror(errno));
break;
}
bzero(buf, MAXBUF + 1);
/* 接收客户端的消息 */
len = recv(new_fd, buf, MAXBUF, 0);
if (len > 0)
printf
("接收消息成功:'%s', 共%d 个字节的数据\n",
buf, len);
else {
if (len < 0)
printf
("消息接收失败! 错误代码是%d, 错误信息是'%s'\n",
errno, strerror(errno));
else
printf("对方退出了, 聊天终止\n");
break;
}
}
close(new_fd);
/* 处理每个新连接上的数据收发结束 */
}

close(sockfd);
return 0;
}
```

客户端源代码如下：

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/socket.h>
#include <resolv.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define MAXBUF 1024
/************关于本文档*********************************************
// *filename: sync-client.c
*purpose: 演示网络同步通讯，这是客户端程序
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-25 20:32
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
********************************************************************/
int main(int argc, char **argv)
{
int sockfd, len;
struct sockaddr_in dest;
char buffer[MAXBUF + 1];

if (argc != 3) {
printf
("参数格式错误！正确用法如下：\n\t\t%s IP 地址 端口\n\t 比如:\t%s 127.0.0.1 80\n 此程序
用来从某个 IP 地址的服务器某个端口接收最多 MAXBUF 个字节的消息",
argv[0], argv[0]);
exit(0);
}
/* 创建一个 socket 用于 tcp 通信 */
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
perror("Socket");
exit(errno);
}
```

```
printf("socket created\n");

/* 初始化服务器端（对方）的地址和端口信息 */
bzero(&dest, sizeof(dest));
dest.sin_family = AF_INET;
dest.sin_port = htons(atoi(argv[2]));
if (inet_aton(argv[1], (struct in_addr *) &dest.sin_addr.s_addr) == 0) {
perror(argv[1]);
exit(errno);
}
printf("address created\n");

/* 连接服务器 */
if (connect(sockfd, (struct sockaddr *) &dest, sizeof(dest)) != 0) {
perror("Connect ");
exit(errno);
}
printf("server connected\n");

while (1) {
/* 接收对方发过来的消息，最多接收 MAXBUF 个字节 */
bzero(buffer, MAXBUF + 1);
/* 接收服务器来的消息 */
len = recv(sockfd, buffer, MAXBUF, 0);
if (len > 0)
printf("接收消息成功:'%s'，共%d 个字节的数据\n",
buffer, len);
else {
if (len < 0)
printf
("消息接收失败！错误代码是%d，错误信息是'%s'\n",
errno, strerror(errno));
else
printf("对方退出了，聊天终止！\n");
break;
}
bzero(buffer, MAXBUF + 1);
printf("请输入要发送给对方的消息：");
fgets(buffer, MAXBUF, stdin);
if (!strncasecmp(buffer, "quit", 4)) {
printf("自己请求终止聊天！\n");
break;
}
/* 发消息给服务器 */
```

```
len = send(sockfd, buffer, strlen(buffer) - 1, 0);
if (len < 0) {
printf
("消息'%s'发送失败！错误代码是%d，错误信息是'%s'\n",
buffer, errno, strerror(errno));
break;
} else
printf
("消息：%s\t 发送成功，共发送了%d 个字节！\n",
buffer, len);
}
/* 关闭连接 */
close(sockfd);
return 0;
}
```

编译程序用下列命令：

```
gcc -Wall sync-server.c -o server
gcc -Wall sync-client.c -o client
```

分别运行这两个程序：

```
./server 7838 1
./client 127.0.0.1 7838
```

同步聊天过程如下：

1、服务端程序开启
2、客户端程序开启（即客户端连接服务器）
3、服务器端用户输入消息并发送给客户端
4、客户端收到消息，客户端用户输入消息并发送给服务器端
5、服务器端收到消息，服务器端用户输入消息并发送给客户端
6、任何一方退出当前聊天即终止一次聊天过程
7、服务器端继续等待下一个人连接上来开始新的聊天过程

同步通讯，要求任何一方都必须按照顺序操作，一收一发或一发一收

Linux 网络编程一步一步学-异步通讯聊天程序 select

关键词： Linux asynchronous socket accept chat

什么是异步通讯?
就是通讯任意一方可以任意发送消息，有消息来到时会收到系统提示去接收消息。

这里要用到 select 函数。使用步骤如下：
1、设置一个集合变量，用来存放所有要判断的句柄（file descriptors：即我们建立的每个 socket、用 open 打开的每个文件等）
2、把需要判断的句柄加入到集合里
3、设置判断时间
4、开始等待，即 select
5、如果在设定的时间内有任何句柄状态变化了就马上返回，并把句柄设置到集合里

服务器端源代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/time.h>
#include <sys/types.h>

#define MAXBUF 1024
/***********关于本文档*********************************************
*filename: async-server.c
*purpose: 演示网络异步通讯，这是服务器端程序
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-25 21:22
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*****************************************************************/

int main(int argc, char **argv)
```

```
{
    int sockfd, new_fd;
    socklen_t len;
    struct sockaddr_in my_addr, their_addr;
    unsigned int myport, lisnum;
    char buf[MAXBUF + 1];
    fd_set rfds;
    struct timeval tv;
    int retval, maxfd = -1;

    if (argv[1])
        myport = atoi(argv[1]);
    else
        myport = 7838;

    if (argv[2])
        lisnum = atoi(argv[2]);
    else
        lisnum = 2;

    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    bzero(&my_addr, sizeof(my_addr));
    my_addr.sin_family = PF_INET;
    my_addr.sin_port = htons(myport);
    if (argv[3])
        my_addr.sin_addr.s_addr = inet_addr(argv[3]);
    else
        my_addr.sin_addr.s_addr = INADDR_ANY;

    if (bind(sockfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr))
        == -1) {
        perror("bind");
        exit(1);
    }

    if (listen(sockfd, lisnum) == -1) {
        perror("listen");
        exit(1);
    }
```

```
while (1) {
    printf
        ("\n----等待新的连接到来开始新一轮聊天……\n");
    len = sizeof(struct sockaddr);
    if ((new_fd =
         accept(sockfd, (struct sockaddr *) &their_addr,
                &len)) == -1) {
        perror("accept");
        exit(errno);
    } else
        printf("server: got connection from %s, port %d, socket %d\n",
               inet_ntoa(their_addr.sin_addr),
               ntohs(their_addr.sin_port), new_fd);

    /* 开始处理每个新连接上的数据收发 */
    printf
        ("\n 准备就绪, 可以开始聊天了……直接输入消息回车即可发信息给对方\n");
    while (1) {
        /* 把集合清空 */
        FD_ZERO(&rfds);
        /* 把标准输入句柄 0 加入到集合中 */
        FD_SET(0, &rfds);
        maxfd = 0;
        /* 把当前连接句柄 new_fd 加入到集合中 */
        FD_SET(new_fd, &rfds);
        if (new_fd > maxfd)
            maxfd = new_fd;
        /* 设置最大等待时间 */
        tv.tv_sec = 1;
        tv.tv_usec = 0;
        /* 开始等待 */
        retval = select(maxfd + 1, &rfds, NULL, NULL, &tv);
        if (retval == -1) {
            printf("将退出, select 出错! %s", strerror(errno));
            break;
        } else if (retval == 0) {
            /* printf
               ("没有任何消息到来, 用户也没有按键, 继续等待……\n"); */
            continue;
        } else {
            if (FD_ISSET(0, &rfds)) {
                /* 用户按键了, 则读取用户输入的内容发送出去 */
                bzero(buf, MAXBUF + 1);
                fgets(buf, MAXBUF, stdin);
```

```
                if (!strncasecmp(buf, "quit", 4)) {
                    printf("自己请求终止聊天！\n");
                    break;
                }
                len = send(new_fd, buf, strlen(buf) - 1, 0);
                if (len > 0)
                    printf
                        ("消息:%s\t 发送成功，共发送了%d 个字节！\n",
                         buf, len);
                else {
                    printf
                        ("消息'%s'发送失败！错误代码是%d，错误信息是'%s'\n",
                         buf, errno, strerror(errno));
                    break;
                }
            }
            if (FD_ISSET(new_fd, &rfds)) {
                /* 当前连接的 socket 上有消息到来则接收对方发过来的消息并显示 */
                bzero(buf, MAXBUF + 1);
                /* 接收客户端的消息 */
                len = recv(new_fd, buf, MAXBUF, 0);
                if (len > 0)
                    printf
                        ("接收消息成功:'%s'，共%d 个字节的数据\n",
                         buf, len);
                else {
                    if (len < 0)
                        printf
                            ("消息接收失败！错误代码是%d，错误信息是'%s'\n",
                             errno, strerror(errno));
                    else
                        printf("对方退出了，聊天终止\n");
                    break;
                }
            }
        }
    }
    close(new_fd);
    /* 处理每个新连接上的数据收发结束 */
    printf("还要和其它连接聊天吗？(no->退出)");
    fflush(stdout);
    bzero(buf, MAXBUF + 1);
    fgets(buf, MAXBUF, stdin);
    if (!strncasecmp(buf, "no", 2)) {
```

```
            printf("终止聊天！\n");
            break;
        }
    }

    close(sockfd);
    return 0;
}
```

客户端源代码如下：

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/socket.h>
#include <resolv.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/types.h>

#define MAXBUF 1024
/************关于本文档*********************************************
// *filename: ssync-client.c
*purpose: 演示网络异步通讯，这是客户端程序
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux爱好者 Linux知识传播者 SOHO族 开发者 最擅长C语言
*date time:2007-01-25 21:32
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*********************************************************************/
int main(int argc, char **argv)
{
    int sockfd, len;
    struct sockaddr_in dest;
    char buffer[MAXBUF + 1];
    fd_set rfds;
    struct timeval tv;
    int retval, maxfd = -1;
```

```
    if (argc != 3) {
        printf
            ("参数格式错误！正确用法如下：\n\t\t%s IP 地址 端口\n\t 比如:\t%s
127.0.0.1 80\n 此程序用来从某个 IP 地址的服务器某个端口接收最多 MAXBUF 个字节的消息",
             argv[0], argv[0]);
        exit(0);
    }
    /* 创建一个 socket 用于 tcp 通信 */
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket");
        exit(errno);
    }

    /* 初始化服务器端（对方）的地址和端口信息 */
    bzero(&dest, sizeof(dest));
    dest.sin_family = AF_INET;
    dest.sin_port = htons(atoi(argv[2]));
    if (inet_aton(argv[1], (struct in_addr *) &dest.sin_addr.s_addr) == 0) {
        perror(argv[1]);
        exit(errno);
    }

    /* 连接服务器 */
    if (connect(sockfd, (struct sockaddr *) &dest, sizeof(dest)) != 0) {
        perror("Connect ");
        exit(errno);
    }

    printf
        ("\n 准备就绪，可以开始聊天了……直接输入消息回车即可发信息给对方\n");
    while (1) {
        /* 把集合清空 */
        FD_ZERO(&rfds);
        /* 把标准输入句柄 0 加入到集合中 */
        FD_SET(0, &rfds);
        maxfd = 0;
        /* 把当前连接句柄 sockfd 加入到集合中 */
        FD_SET(sockfd, &rfds);
        if (sockfd > maxfd)
            maxfd = sockfd;
        /* 设置最大等待时间 */
        tv.tv_sec = 1;
        tv.tv_usec = 0;
        /* 开始等待 */
```

```
        retval = select(maxfd + 1, &rfds, NULL, NULL, &tv);
        if (retval == -1) {
            printf("将退出, select 出错! %s", strerror(errno));
            break;
        } else if (retval == 0) {
            /* printf
               ("没有任何消息到来, 用户也没有按键, 继续等待……\n"); */
            continue;
        } else {
            if (FD_ISSET(sockfd, &rfds)) {
                /* 连接的 socket 上有消息到来则接收对方发过来的消息并显示 */
                bzero(buffer, MAXBUF + 1);
                /* 接收对方发过来的消息, 最多接收 MAXBUF 个字节 */
                len = recv(sockfd, buffer, MAXBUF, 0);
                if (len > 0)
                    printf
                        ("接收消息成功:'%s', 共%d 个字节的数据\n",
                         buffer, len);
                else {
                    if (len < 0)
                        printf
                            ("消息接收失败! 错误代码是%d, 错误信息是'%s'\n",
                             errno, strerror(errno));
                    else
                        printf("对方退出了, 聊天终止! \n");
                    break;
                }
            }
            if (FD_ISSET(0, &rfds)) {
                /* 用户按键了, 则读取用户输入的内容发送出去 */
                bzero(buffer, MAXBUF + 1);
                fgets(buffer, MAXBUF, stdin);
                if (!strncasecmp(buffer, "quit", 4)) {
                    printf("自己请求终止聊天! \n");
                    break;
                }
                /* 发消息给服务器 */
                len = send(sockfd, buffer, strlen(buffer) - 1, 0);
                if (len < 0) {
                    printf
                        ("消息'%s'发送失败! 错误代码是%d, 错误信息是'%s'\n",
                         buffer, errno, strerror(errno));
                    break;
                } else
```

```
                    printf
                        ("消息：%s\t 发送成功，共发送了%d 个字节！\n",
                         buffer, len);
                }
            }
        }
    /* 关闭连接 */
    close(sockfd);
    return 0;
}
```

编译用如下命令：

```
gcc -Wall async-server.c -o server
gcc -Wall async-client.c -o client
```

运行用如下命令：

```
./server 7838 1
./client 127.0.0.1 7838
```

Linux 网络编程一步一步学-编写一个 HTTP 协议的目录浏览和文件下载服务器

关键词： Linux 网络 socket HTTP 下载

服务器源代码如下：

```
#include <stdarg.h>
#include <errno.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <errno.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <resolv.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <signal.h>
#include <getopt.h>

#define DEFAULTIP "127.0.0.1"
#define DEFAULTPORT "80"
#define DEFAULTBACK "10"
#define DEFAULTDIR "/home"
#define DEFAULTLOG "/tmp/das-server.log"

void prterrmsg(char *msg);
#define prterrmsg(msg)        { perror(msg); abort(); }
void wrterrmsg(char *msg);
#define wrterrmsg(msg)        { fputs(msg, logfp); fputs(strerror(errno),
logfp);fflush(logfp); abort(); }

void prtinfomsg(char *msg);
#define prtinfomsg(msg)        { fputs(msg, stdout);  }
void wrtinfomsg(char *msg);
#define wrtinfomsg(msg)        {  fputs(msg, logfp); fflush(logfp);}

#define MAXBUF        1024
```

```
char buffer[MAXBUF + 1];
char *host = 0;
char *port = 0;
char *back = 0;
char *dirroot = 0;
char *logdir = 0;
unsigned char daemon_y_n = 0;
FILE *logfp;

#define MAXPATH        150

/*---------------------------------------
 *--- dir_up - 查找dirpath所指目录的上一级目录
 *---------------------------------------
 */
char *dir_up(char *dirpath)
{
    static char Path[MAXPATH];
    int len;

    strcpy(Path, dirpath);
    len = strlen(Path);
    if (len > 1 && Path[len - 1] == '/')
        len--;
    while (Path[len - 1] != '/' && len > 1)
        len--;
    Path[len] = 0;
    return Path;
}

/*-------------------------------------------------------
 *--- AllocateMemory - 分配空间并把d所指的内容复制
 *-------------------------------------------------------
 */
void AllocateMemory(char **s, int l, char *d)
{
    *s = malloc(l + 1);
    bzero(*s, l + 1);
    memcpy(*s, d, l);
}
/***********关于本文档*****************************************
*filename: das-server.c
*purpose: 这是在Linux下用C语言写的目录访问服务器，支持目录浏览和文件下载
```

```
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux爱好者 Linux知识传播者 SOHO族 开发者 最擅长C语言
*date time:2007-01-26 19:32
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
********************************************************************/
/*-------------------------------------------------------
 *--- GiveResponse - 把Path所指的内容发送到client_sock去
 *------------------如果Path是一个目录，则列出目录内容
 *------------------如果Path是一个文件，则下载文件
 *-------------------------------------------------------
 */
void GiveResponse(FILE * client_sock, char *Path)
{
    struct dirent *dirent;
    struct stat info;
    char Filename[MAXPATH];
    DIR *dir;
    int fd, len, ret;
    char *p, *realPath, *realFilename, *nport;

    /* 获得实际工作目录或文件 */
    len = strlen(dirroot) + strlen(Path) + 1;
    realPath = malloc(len + 1);
    bzero(realPath, len + 1);
    sprintf(realPath, "%s/%s", dirroot, Path);

    /* 获得实际工作端口 */
    len = strlen(port) + 1;
    nport = malloc(len + 1);
    bzero(nport, len + 1);
#160;   sprintf(nport, ":%s", port);

    /* 获得实际工作目录或文件的信息以判断是文件还是目录 */
    if (stat(realPath, &info)) {
        fprintf(client_sock,
                "HTTP/1.1 200 OK\r\nServer: DAS by ZhouLifa\r\nConnection:
close\r\n\r\n<html><head><title>%d - %s</title></head>"
                "<body><font size=+4>Linux 下目录访问服务器</font><br><hr
width=\"100%%\"><br><center>"
                "<table border cols=3 width=\"100%%\">", errno,
```

```
                strerror(errno));
        fprintf(client_sock,
                "</table><font color=\"CC0000\" size=+2>请向管理员咨询为何出现如下
错误提示: \n%s %s</font></body></html>",
                Path, strerror(errno));
        goto out;
    }
    /* 处理浏览文件请求, 即下载文件 */
    if (S_ISREG(info.st_mode)) {
        fd = open(realPath, O_RDONLY);
        len = lseek(fd, 0, SEEK_END);
        p = (char *) malloc(len + 1);
        bzero(p, len + 1);
        lseek(fd, 0, SEEK_SET);
        ret = read(fd, p, len);
        close(fd);
        fprintf(client_sock,
                "HTTP/1.1 200 OK\r\nServer: DAS by ZhouLifa\r\nConnection:
keep-alive\r\nContent-type: application/*\r\nContent-Length:%d\r\n\r\n",
                len);
        fwrite(p, len, 1, client_sock);
        free(p);
    } else if (S_ISDIR(info.st_mode)) {
        /* 处理浏览目录请求 */
        dir = opendir(realPath);
        fprintf(client_sock,
                "HTTP/1.1 200 OK\r\nServer: DAS by ZhouLifa\r\nConnection:
close\r\n\r\n<html><head><title>%s</title></head>"
                "<body><font size=+4>Linux 下目录访问服务器</font><br><hr
width=\"100%%\"><br><center>"
                "<table border cols=3 width=\"100%%\">", Path);
        fprintf(client_sock,
                "<caption><font size=+3>目录 %s</font></caption>\n",
                Path);
        fprintf(client_sock,
                "<tr><td>名称</td><td>大小</td><td>修改时间</td></tr>\n");
        if (dir == 0) {
            fprintf(client_sock,
                    "</table><font color=\"CC0000\" size=+2>
%s</font></body></html>",
                    strerror(errno));
            return;
        }
        /* 读取目录里的所有内容 */
```

```
        while ((dirent = readdir(dir)) != 0) {
            if (strcmp(Path, "/") == 0)
                sprintf(Filename, "/%s", dirent->d_name);
            else
                sprintf(Filename, "%s/%s", Path, dirent->d_name);
            fprintf(client_sock, "<tr>");
            len = strlen(dirroot) + strlen(Filename) + 1;
            realFilename = malloc(len + 1);
            bzero(realFilename, len + 1);
            sprintf(realFilename, "%s/%s", dirroot, Filename);
            if (stat(realFilename, &info) == 0) {
                if (strcmp(dirent->d_name, "..") == 0)
                    fprintf(client_sock,
                            "<td><a href=\"http://%s%s%s\">(parent)</a></td>",
                            host, atoi(port) == 80 ? "" : nport,
                            dir_up(Path));
                else
                    fprintf(client_sock,
                            "<td><a href=\"http://%s%s%s\">%s</a></td>",
                            host, atoi(port) == 80 ? "" : nport, Filename,
                            dirent->d_name);
#160;               if (S_ISDIR(info.st_mode))
                    fprintf(client_sock, "<td>目录</td>");
                else if (S_ISREG(info.st_mode))
                    fprintf(client_sock, "<td>%d</td>", info.st_size);
                else if (S_ISLNK(info.st_mode))
                    fprintf(client_sock, "<td>链接</td>");
                else if (S_ISCHR(info.st_mode))
                    fprintf(client_sock, "<td>字符设备</td>");
                else if (S_ISBLK(info.st_mode))
                    fprintf(client_sock, "<td>块设备</td>");
                else if (S_ISFIFO(info.st_mode))
                    fprintf(client_sock, "<td>FIFO</td>");
                else if (S_ISSOCK(info.st_mode))
                    fprintf(client_sock, "<td>Socket</td>");
                else
                    fprintf(client_sock, "<td>(未知)</td>");
                fprintf(client_sock, "<td>%s</td>", ctime(&info.st_ctime));
            }
            fprintf(client_sock, "</tr>\n");
            free(realFilename);
        }
        fprintf(client_sock, "</table></center></body></html>");
    } else {
```

```
        /* 既非常规文件又非目录，禁止访问 */
        fprintf(client_sock,
                "HTTP/1.1 200 OK\r\nServer: DAS by ZhouLifa\r\nConnection:
close\r\n\r\n<html><head><title>permission denied</title></head>"
                "<body><font size=+4>Linux 下目录访问服务器</font><br><hr
width=\"100%%\"><br><center>"
                "<table border cols=3 width=\"100%%\">");
        fprintf(client_sock,
                "</table><font color=\"CC0000\" size=+2>你访问的资源'%s'被禁止访问,
请联系管理员解决！</font></body></html& gt;",
                Path);
    }
  out:
    free(realPath);
    free(nport);
}

/*---------------------------------------------------------
 *--- getoption - 分析取出程序的参数
 *---------------------------------------------------------
 */
void getoption(int argc, char **argv)
{
    int c, len;
    char *p = 0;

    opterr = 0;
    while (1) {
        int option_index = 0;
        static struct option long_options[] = {
            {"host", 1, 0, 0},
            {"port", 1, 0, 0},
            {"back", 1, 0, 0},
            {"dir", 1, 0, 0},
            {"log", 1, 0, 0},
            {"daemon", 0, 0, 0},
            {0, 0, 0, 0}
        };
        /* 本程序支持如一些参数:
         * --host IP地址 或者 -H IP地址
         * --port 端口 或者 -P 端口
         * --back 监听数量 或者 -B 监听数量
         * --dir 网站根目录 或者 -D 网站根目录
         * --log 日志存放路径 或者 -L 日志存放路径
```

```
         * --daemon 使程序进入后台运行模式
         */
        c = getopt_long(argc, argv, "H:P:B:D:L",
                        long_options, &option_index);
        if (c == -1 || c == '?')
            break;

        if(optarg)        len = strlen(optarg);
        else        len = 0;

        if ((!c && !(strcasecmp(long_options[option_index].name, "host")))
            || c == 'H')
            p = host = malloc(len + 1);
        else if ((!c
                  &&
                  !(strcasecmp(long_options[option_index].name, "port")))
                 || c == 'P')
            p = port = malloc(len + 1);
        else if ((!c
                  &&
                  !(strcasecmp(long_options[option_index].name, "back")))
                 || c == 'B')
            p = back = malloc(len + 1);
        else if ((!c
                  && !(strcasecmp(long_options[option_index].name, "dir")))
                 || c == 'D')
            p = dirroot = malloc(len + 1);
        else if ((!c
                  && !(strcasecmp(long_options[option_index].name, "log")))
                 || c == 'L')
            p = logdir = malloc(len + 1);
        else if ((!c
                  &&
                  !(strcasecmp
                    (long_options[option_index].name, "daemon")))) {
            daemon_y_n = 1;
            continue;
        }
        else
            break;
        bzero(p, len + 1);
        memcpy(p, optarg, len);
    }
}
```

```
int main(int argc, char **argv)
{
    struct sockaddr_in addr;
    int sock_fd, addr1en;

    /* 获得程序工作的参数，如 IP 、端口、监听数、网页根目录、目录存放位置等 */
    getoption(argc, argv);

    if (!host) {
        addr1en = str1en(DEFAULTIP);
        A11ocateMemory(&host, addr1en, DEFAULTIP);
    }
    if (!port) {
        addr1en = str1en(DEFAULTPORT);
        A11ocateMemory(&port, addr1en, DEFAULTPORT);
    }
    if (!back) {
        addr1en = str1en(DEFAULTBACK);
        A11ocateMemory(&back, addr1en, DEFAULTBACK);
    }
    if (!dirroot) {
        addr1en = str1en(DEFAULTDIR);
        A11ocateMemory(&dirroot, addr1en, DEFAULTDIR);
    }
    if (!1ogdir) {
        addr1en = str1en(DEFAULTLOG);
        A11ocateMemory(&1ogdir, addr1en, DEFAULTLOG);
    }

    printf
        ("host=%s port=%s back=%s dirroot=%s 1ogdir=%s %s是后台工作模式(进程ID:
%d)\n",
         host, port, back, dirroot, 1ogdir, daemon_y_n?"":"不", getpid());

    /* fork() 两次处于后台工作模式下 */
    if (daemon_y_n) {
        if (fork())
            exit(0);
        if (fork())
            exit(0);
        c1ose(0), c1ose(1), c1ose(2);
        1ogfp = fopen(1ogdir, "a+");
        if (!1ogfp)
```

```
        exit(0);
    }

    /* 处理子进程退出以免产生僵尸进程 */
    signal(SIGCHLD, SIG_IGN);

    /* 创建 socket */
    if ((sock_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        if (!daemon_y_n) {
            prterrmsg("socket()");
        } else {
            wrterrmsg("socket()");
        }
    }

    /* 设置端口快速重用 */
    addrlen = 1;
    setsockopt(sock_fd, SOL_SOCKET, SO_REUSEADDR, &addrlen,
               sizeof(addrlen));

    addr.sin_family = AF_INET;
    addr.sin_port = htons(atoi(port));
    addr.sin_addr.s_addr = inet_addr(host);
    addrlen = sizeof(struct sockaddr_in);
    /* 绑定地址、端口等信息 */
    if (bind(sock_fd, (struct sockaddr *) &addr, addrlen) < 0) {
        if (!daemon_y_n) {
            prterrmsg("bind()");
        } else {
            wrterrmsg("bind()");
        }
    }

    /* 开启临听 */
    if (listen(sock_fd, atoi(back)) < 0) {
        if (!daemon_y_n) {
            prterrmsg("listen()");
        } else {
            wrterrmsg("listen()");
        }
    }
    while (1) {
        int len;
        int new_fd;
```

```
        addr1en = sizeof(struct sockaddr_in);
        /* 接受新连接请求 */
        new_fd = accept(sock_fd, (struct sockaddr *) &addr, &addr1en);
        if (new_fd < 0) {
            if (!daemon_y_n) {
                prterrmsg("accept()");
            } e1se {
                wrterrmsg("accept()");
            }
            break;
        }
        bzero(buffer, MAXBUF + 1);
        sprintf(buffer, "连接来自于: %s:%d\n",
                inet_ntoa(addr.sin_addr), ntohs(addr.sin_port));
        if (!daemon_y_n) {
            prtinfomsg(buffer);
        } e1se {
            wrtinfomsg(buffer);
        }
        /* 产生一个子进程去处理请求, 当前进程继续等待新的连接到来 */
        if (!fork()) {
            bzero(buffer, MAXBUF + 1);
            if ((1en = recv(new_fd, buffer, MAXBUF, 0)) > 0) {
                FILE *C1ientFP = fdopen(new_fd, "w");
                if (C1ientFP == NULL) {
                    if (!daemon_y_n) {
                        prterrmsg("fdopen()");
                    } e1se {
                        prterrmsg("fdopen()");
                    }
                } e1se {
                    char Req[MAXPATH + 1] = "";
                    sscanf(buffer, "GET %s HTTP", Req);
                    bzero(buffer, MAXBUF + 1);
                    sprintf(buffer, "请求取文件: \"%s\"\n", Req);
                    if (!daemon_y_n) {
                        prtinfomsg(buffer);
                    } e1se {
                        wrtinfomsg(buffer);
                    }
                    /* 处理用户请求 */
                    GiveResponse(C1ientFP, Req);
                    fc1ose(C1ientFP);
                }
```

```
            }
            exit(0);
        }
        close(new_fd);
    }
    close(sock_fd);
    return 0;
}
```

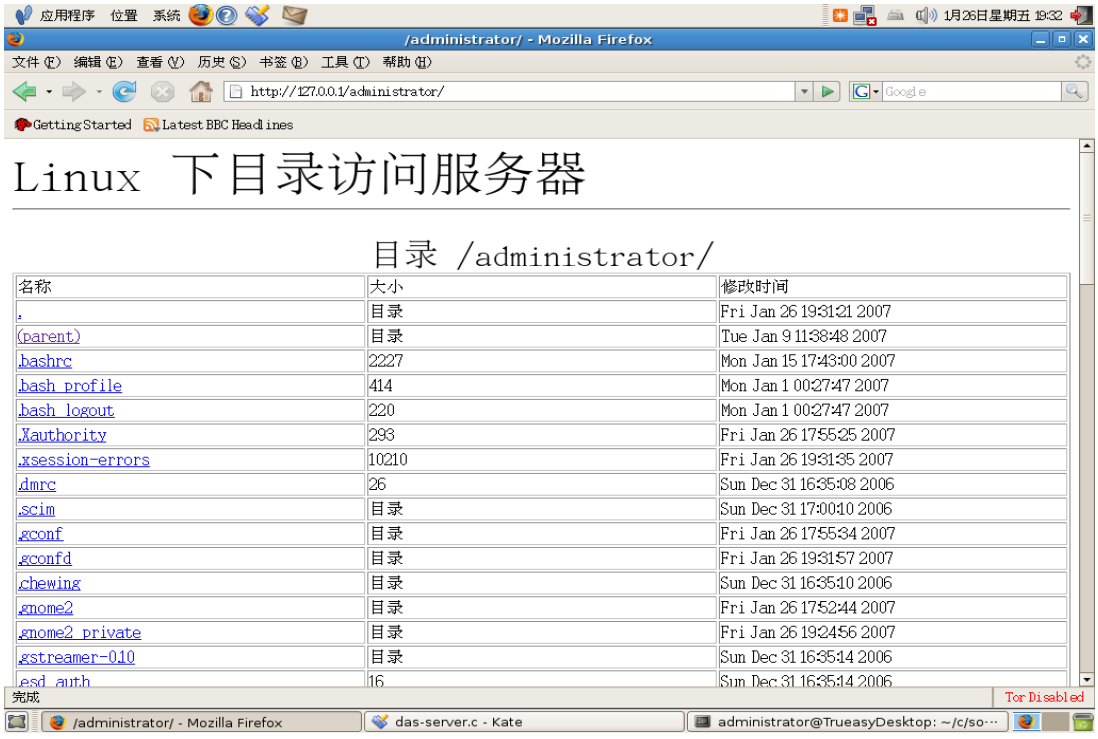
编译程序用下列命令：
gcc -Wall das-server.c -o das-server
注：das 即 Dictory Access Server

以 root 用户启动服务程序用下列命令：
./das-server
或以普通用户启动服务程序用下列命令：
./das-server --port 7838
或
./das-server -P 7838

注：只有 root 用户才有权限启动 1024 以下的端口，所以如果想用默认的 80 端口就得用 root 来运行。

如果要想让程序在后台自动运行，即处理精灵模式下工作，在命令后面加上--daemon 参数即可。

打开一个网络浏览器输入服务地址开始浏览，如下图：



下载文件如下图：

注：请不要下载较大的文件，比如文件大小超过 10M 的，因为程序是一次分配内存，会占用系统内存较大导致系统死掉！

Linux 网络编程一步一步学-用 C 自己编写一个 telnet 服务器

关键词： Linux 网络 socket telnet server

服务器源代码如下：

```
#include <stdarg.h>
#include <errno.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <errno.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <resolv.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <signal.h>
#include <getopt.h>

#define DEFAULTIP "127.0.0.1"
#define DEFAULTPORT "23"
#define DEFAULTBACK "10"
#define DEFAULTDIR "/tmp"
#define DEFAULTLOG "/tmp/telnet-server.log"

void prterrmsg(char *msg);
#define prterrmsg(msg)         { perror(msg); abort(); }
void wrterrmsg(char *msg);
#define wrterrmsg(msg)         { fputs(msg, logfp); fputs(strerror(errno),
logfp);fflush(logfp); abort(); }

void prtinfomsg(char *msg);
#define prtinfomsg(msg)         { fputs(msg, stdout);  }
void wrtinfomsg(char *msg);
#define wrtinfomsg(msg)         {  fputs(msg, logfp); fflush(logfp);}

#define MAXBUF        1024
```

```
char buffer[MAXBUF + 1];
char *host = 0;
char *port = 0;
char *back = 0;
char *dirroot = 0;
char *logdir = 0;
unsigned char daemon_y_n = 0;
FILE *logfp;

#define MAXPATH        150

/*---------------------------------------------------
 *--- AllocateMemory - 分配空间并把 d 所指的内容复制
 *---------------------------------------------------
 */
void AllocateMemory(char **s, int l, char *d)
{
    *s = malloc(l + 1);
    bzero(*s, l + 1);
    memcpy(*s, d, l);
}
/***********关于本文档***********************************************************
*filename: telnet-server.c
*purpose: 这是在Linux下用C语言写的telnet服务器，没有用户名和密码，直接以开启服务者的身份登
录系统
*wrote by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux爱好者 Linux知识传播者 SOHO族 开发者 最擅长C语言
*date time:2007-01-27 17:02
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
*******************************************************************************/

/*---------------------------------------------------
 *--- getoption - 分析取出程序的参数
 *---------------------------------------------------
 */
void getoption(int argc, char **argv)
{
    int c, len;
    char *p = 0;
```

```
opterr = 0;
while (1) {
    int option_index = 0;
    static struct option long_options[] = {
        {"host", 1, 0, 0},
        {"port", 1, 0, 0},
        {"back", 1, 0, 0},
        {"dir", 1, 0, 0},
        {"log", 1, 0, 0},
        {"daemon", 0, 0, 0},
        {0, 0, 0, 0}
    };
    /* 本程序支持如一些参数:
     * --host IP地址 或者 -H IP地址
     * --port 端口 或者 -P 端口
     * --back 监听数量 或者 -B 监听数量
     * --dir 服务默认目录 或者 -D 服务默认目录
     * --log 日志存放路径 或者 -L 日志存放路径
     * --daemon 使程序进入后台运行模式
     */
    c = getopt_long(argc, argv, "H:P:B:D:L",
                    long_options, &option_index);
    if (c == -1 || c == '?')
        break;

    if(optarg)        len = strlen(optarg);
    else        len = 0;

    if ((!c && !(strcasecmp(long_options[option_index].name, "host")))
        || c == 'H')
        p = host = malloc(len + 1);
    else if ((!c
              &&
              !(strcasecmp(long_options[option_index].name, "port")))
             || c == 'P')
        p = port = malloc(len + 1);
    else if ((!c
              &&
              !(strcasecmp(long_options[option_index].name, "back")))
             || c == 'B')
        p = back = malloc(len + 1);
    else if ((!c
              && !(strcasecmp(long_options[option_index].name, "dir")))
```

```
                || c == 'D')
            p = dirroot = malloc(len + 1);
        else if ((!c
                  && !(strcasecmp(long_options[option_index].name, "log")))
                 || c == 'L')
            p = logdir = malloc(len + 1);
        else if ((!c
                  &&
                  !(strcasecmp
                    (long_options[option_index].name, "daemon")))) {
            daemon_y_n = 1;
            continue;
        }
        else
            break;
        bzero(p, len + 1);
        memcpy(p, optarg, len);
    }
}

int main(int argc, char **argv)
{
    struct sockaddr_in addr;
    int sock_fd, addrlen;

    /* 获得程序工作的参数，如 IP 、端口、监听数、网页根目录、目录存放位置等 */
    getoption(argc, argv);

    if (!host) {
        addrlen = strlen(DEFAULTIP);
        AllocateMemory(&host, addrlen, DEFAULTIP);
    }
    if (!port) {
        addrlen = strlen(DEFAULTPORT);
        AllocateMemory(&port, addrlen, DEFAULTPORT);
    }
    if (!back) {
        addrlen = strlen(DEFAULTBACK);
        AllocateMemory(&back, addrlen, DEFAULTBACK);
    }
    if (!dirroot) {
        addrlen = strlen(DEFAULTDIR);
        AllocateMemory(&dirroot, addrlen, DEFAULTDIR);
    }
```

```
    if (!logdir) {
        addrlen = strlen(DEFAULTLOG);
        AllocateMemory(&logdir, addrlen, DEFAULTLOG);
    }

    printf
        ("host=%s port=%s back=%s dirroot=%s logdir=%s %s是后台工作模式(进程ID:
%d)\n",
         host, port, back, dirroot, logdir, daemon_y_n?"":"不", getpid());

    /* fork() 两次处于后台工作模式下 */
    if (daemon_y_n) {
        if (fork())
            exit(0);
        if (fork())
            exit(0);
        close(0), close(1), close(2);
        logfp = fopen(logdir, "a+");
        if (!logfp)
            exit(0);
    }

    /* 处理子进程退出以免产生僵尸进程 */
    signal(SIGCHLD, SIG_IGN);

    /* 创建 socket */
    if ((sock_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        if (!daemon_y_n) {
            prterrmsg("socket()");
        } else {
            wrterrmsg("socket()");
        }
    }

    /* 设置端口快速重用 */
    addrlen = 1;
    setsockopt(sock_fd, SOL_SOCKET, SO_REUSEADDR, &addrlen,
               sizeof(addrlen));

    addr.sin_family = AF_INET;
    addr.sin_port = htons(atoi(port));
    addr.sin_addr.s_addr = inet_addr(host);
    addrlen = sizeof(struct sockaddr_in);
    /* 绑定地址、端口等信息 */
```

```
if (bind(sock_fd, (struct sockaddr *) &addr, addrlen) < 0) {
    if (!daemon_y_n) {
        prterrmsg("bind()");
    } else {
        wrterrmsg("bind()");
    }
}

/* 开启临听 */
if (listen(sock_fd, atoi(back)) < 0) {
    if (!daemon_y_n) {
        prterrmsg("listen()");
    } else {
        wrterrmsg("listen()");
    }
}
while (1) {
    int new_fd;
    addrlen = sizeof(struct sockaddr_in);
    /* 接受新连接请求 */
    new_fd = accept(sock_fd, (struct sockaddr *) &addr, &addrlen);
    if (new_fd < 0) {
        if (!daemon_y_n) {
            prterrmsg("accept()");
        } else {
            wrterrmsg("accept()");
        }
        break;
    }
    bzero(buffer, MAXBUF + 1);
    sprintf(buffer, "连接来自于: %s:%d\n",
            inet_ntoa(addr.sin_addr), ntohs(addr.sin_port));
    if (!daemon_y_n) {
        prtinfomsg(buffer);
    } else {
        wrtinfomsg(buffer);
    }
    /* 产生一个子进程去处理请求, 当前进程继续等待新的连接到来 */
    if (!fork()) {
        /* 把 socket 连接作为标准输入、输出、出错句柄来用 */
        dup2(new_fd, 0);
        dup2(new_fd, 1);
        dup2(new_fd, 2);
        /* 切换到指定目录工作 */
```

```
            chdir(dirroot);
            /* 交互式执行 shell */
            execl("/bin/bash", "-l", "--login", "-i", "-r", "-s", (char *)NULL);
        }
        close(new_fd);
    }
    close(sock_fd);
    return 0;
}
```

用下列命令编译程序:
gcc -Wall telnet-server -o telnetd

启动 telnet 服务:
./telnetd --daemon #以 root 用户身份在 23 端口（即 telnet 默认端口服务）
或
./telnetd -P 7838 #以非 root 用户身份

然后开启一个新终端，telnet 连接自己的服务器试试，如:
telnet 127.0.0.1
或
telnet 127.0.0.1 7838

不需要输入用户名和密码，直接以启动 telnet 服务的用户的身份登录系统了。
输入系统命令体验一下吧！

Linux 网络编程一步一步学-网络编程函数说明-来自“永远的 UNIX”

关键词： Linux 网络 socket 永远的 UNIX www.fanqiang.co

在 www.fanqiang.com (永远的 UNIX)网站上也有一系统文章，比较详细地介绍了网络编程的各函数，大家可以去那边看看，我就不复制过来了。

那边系列文章的目录和链接如下：

```
/***********关于本文档***********************************************************
***
*filename: Linux 网络编程一步一步学-网络编程函数说明-来自“永远的 UNIX”
*purpose: 详细说明 Linux 下网络编程各函数的具体用法，当然最好的是各位自己查看在线手册 man
*tidied by: zhoulifa(zhoulifa@163.com) 周立发(http://zhoulifa.bokee.com)
Linux 爱好者 Linux 知识传播者 SOHO 族 开发者 最擅长 C 语言
*date time:2007-01-27 19:22
*Note: 任何人可以任意复制代码并运用这些文档，当然包括你的商业用途
* 但请遵循 GPL
*Thanks to: Google.com
*Hope:希望越来越多的人贡献自己的力量，为科学技术发展出力
* 科技站在巨人的肩膀上进步更快！感谢有开源前辈的贡献！
**************************************************************************
**/
```

Linux 网络编程--1.Linux 网络知识介绍

Linux 网络编程--2.初等网络函数介绍（TCP）

Linux 网络编程--3. 服务器和客户机的信息函数

Linux 网络编程--4. 完整的读写函数

Linux 网络编程--5. 用户数据报发送

Linux 网络编程--6. 高级套接字函数

Linux 网络编程--7. TCP/IP 协议

Linux 网络编程--8．套接字选项

Linux 网络编程--9．服务器模型

Linux 网络编程--10．原始套接字 --11．后记

学习任何知识都不能光看不练。必须动手练习，对于这些函数，自己写个小程序测试一下其用法就会很明了了。