# 实验八　网络程序设计

## 基本信息

**实验课程：**Linux程序设计 **设课形式：**非独立

**课程学分：**2 **实验项目：**网络程序设计（1）

**项目类型：**验证 **项目学时：**2

## 实验预习

**实验目的和要求：**

Socket的基本概念如何实现面向连接的TCP编程如何实现非连接的TUDP编程复杂网络程序的实现

网络程序高级编程技术

**实验条件：**

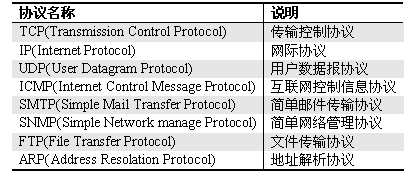
1、装有Linux操作系统的微型计算机；

## 实验过程

### TCP/IP协议

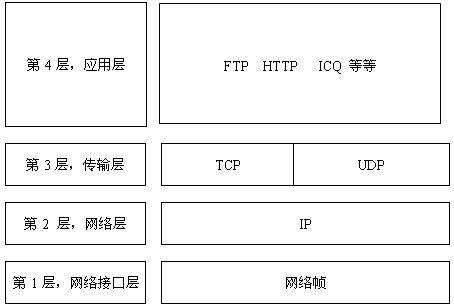
**8.1.1 TCP/IP概述**

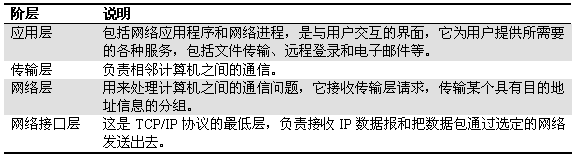
* TCP/IP协议叫做传输控制/网际协议，又叫网络通讯协议。
* TCP/IP是70年代中期美国国防部为其ARPANET广域网开发的网络体系结构和协议标准，以它为基础组建的INTERNET是目前国际上规模最大的计算机网络，正因为INTERNET的广泛使用，使得TCP/IP成了事实上的标准。
* 各种协议：



**8.1.2 TCP/IP模块结构**

* TCP/IP从协议分层模型方面来看，由四个层次组成：网络接口层、网络层、传输层、应用层 。





### Socket基本概念

**8.2.1 socket简介**

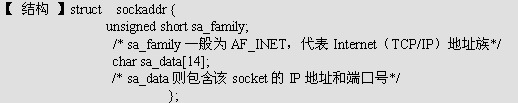
* socket是网络编程的一种接口，它是一种特殊的I/O，用socket函数建立一个Socket连接，此函数返回一个整型的socket描述符，随后进行数据传输。
* 通常socket分为三种类型：流式socket、数据报socket、原始socket。
* **注意：**一个完整的socket有一个本地唯一的socket号，由操作系统分配。最重要的是，socket 是面向客户/服务器模型而设计的。

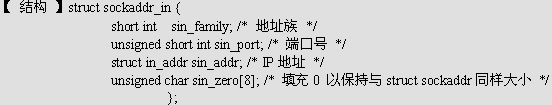
**8.2.2 socket套接口简介**

* 一个IP地址，一个通讯端口，就能确定一个通讯程序的位置。为此开发人员专门设计了一个套接结构，就是把网络程序中所用到的网络地址和端口信息放在一个结构体中。
* 一般套接口地址结构都以“sockaddr”开头。socket根据所使用的协议的不同可以分TCP套接口和UDP套接口，又称为流式套接口和数据套接口。
* UDP是一个无连接协议，TCP是个可靠的端对端协议。传输UDP数据包时，LINUX不知道也不关心它们是否已经安全到达目的地，而传输TCP数据包时，则应先建立连接以保证传输的数据被正确接收。

**8.2.3 socket套接口的数据结构**

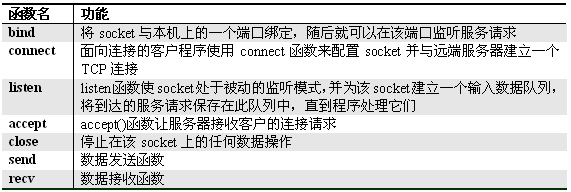
* 两个重要的数据类型：sockadd和sockaddr\_in，这两个结构类型都是用来保存socket信息的，如IP地址、通信端口等。



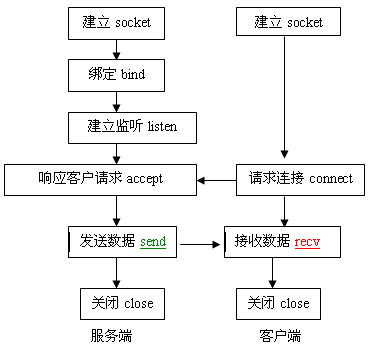


**8.2.4 TCP编程**

* 基于TCP协议的编程，其最主要的特点是建立完连接后，才进行通信。
* 常用的基于TCP网络编程函数及功能



**例8.1：**服务器通过socket连接后，向客户端发送字符串“连接上了”。在服务器上显示客户端的IP地址或域名。



**程序中的主要语句说明：**

**服务器端：**

* 1. 建立socket：socket(AF\_INET, SOCK\_STREAM, 0);
  2. 绑定bind：bind(sockfd,(struct sockaddr \*)&my\_addr,sizeof(struct sockaddr);
  3. 建立监听listen：listen(sockfd, BACKLOG);
  4. 响应客户请求：accept(sockfd,(struct sockaddr \*)&remote\_addr, &sin\_size);
  5. 发送数据send：send(client\_fd, "连接上了 \n", 26, 0);
  6. 关闭close：close(client\_fd);

**客户端：**

* 1. 建立socket：socket(AF\_INET, SOCK\_STREAM, 0);
  2. 请求连接connect：connect(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(struct sockaddr))；
  3. 接收数据recv：recv(sockfd, buf, MAXDATASIZE, 0)；
  4. 关闭close：close(sockfd);

**客户端**

#include<stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <netdb.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <sys/socket.h>

#define SERVPORT 3333

#define MAXDATASIZE 100 /\*每次最大数据传输量 \*/

int main(int argc, char \*argv[]){

int sockfd, recvbytes;

char buf[MAXDATASIZE];

struct hostent \*host;

struct sockaddr\_in serv\_addr;

if (argc < 2) {

fprintf(stderr,"Please enter the server's hostname!\n");

exit(1);

}

if((host=gethostbyname(argv[1]))==NULL) {

herror("gethostbyname error！");

exit(1);

}

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1){

perror("socket create error！");

exit(1);

}

serv\_addr.sin\_family=AF\_INET;

serv\_addr.sin\_port=htons(SERVPORT);

serv\_addr.sin\_addr = \*((struct in\_addr \*)host->h\_addr);

bzero(&(serv\_addr.sin\_zero),8);

if (connect(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(struct sockaddr)) == -1) {

perror("connect error！");

exit(1);

}

if ((recvbytes=recv(sockfd, buf, MAXDATASIZE, 0)) ==-1) {

perror("connect 出错！");

exit(1);

}

buf[recvbytes] = '\0';

printf("收到: %s",buf);

close(sockfd);

}

**服务器端**

#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>

#define SERVPORT 3333 /\*服务器监听端口号 \*/

#define BACKLOG 10 /\* 最大同时连接请求数 \*/

int main()

{

int sockfd,client\_fd; /\*sock\_fd：监听socket；client\_fd：数据传输socket \*/

struct sockaddr\_in my\_addr; /\* 本机地址信息 \*/

struct sockaddr\_in remote\_addr; /\* 客户端地址信息 \*/

int sin\_size;

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)

{

perror("socket 创建失败！"); exit(1);

}

my\_addr.sin\_family=AF\_INET;

my\_addr.sin\_port=htons(SERVPORT);/\* htons()把16位值从主机字节序转换成网络字节序\*/

my\_addr.sin\_addr.s\_addr = INADDR\_ANY;

bzero(&(my\_addr.sin\_zero),8); /\*保持与struct sockaddr同样大小 \*/

if (bind(sockfd, (struct sockaddr \*)&my\_addr, sizeof(struct sockaddr)) == -1)

{

perror("bind 出错！");

exit(1);

}

if (listen(sockfd, BACKLOG) == -1) {

perror("listen 出错！");

exit(1);

}

while(1) {

sin\_size = sizeof(struct sockaddr\_in);

if ((client\_fd = accept(sockfd, (struct sockaddr \*)&remote\_addr, &sin\_size)) == -1) {

perror("accept error");

continue;

}

printf("收到一个连接来自： %s\n", inet\_ntoa(remote\_addr.sin\_addr));

if (!fork()) { /\* 子进程代码段 \*/

if (send(client\_fd, "连接上了 \n", 26, 0) == -1)

perror("send 出错！");

close(client\_fd);

exit(0);

}

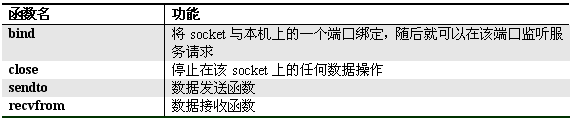
close(client\_fd);

}

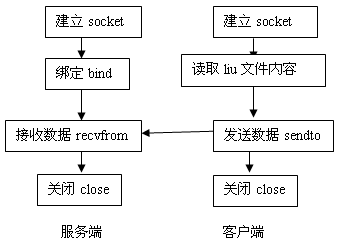
}

**8.2.5 UDP编程**

* 基于UDP协议的编程，其最主要的特点是不需要用函数bind把本地IP地址与端口号进行绑定，也能进行通信。
* 常用的基UDP网络编程函数及功能：



**例8.2：**服务器端接受客户端发送的字符串。客户端将打开liu文件，读取文件中的3个字符串，传送给服务器端，当传送给服务端的字符串为“stop”时，终止数据传送并断开连接。



* **主要语句说明**：

**服务器端：**

* 1. 建立socket：socket(AF\_INET,SOCK\_DGRAM,0)
  2. 绑定bind：bind(sockfd,(struct sockaddr \*)&adr\_inet,sizeof(adr\_inet));
  3. 接收数据recvfrom：recvfrom(sockfd,buf,sizeof(buf),0,(struct sockaddr \*)&adr\_clnt,&len);
  4. 关闭close：close(sockfd);

**客户端：**

* 1. 建立socket：socket(AF\_INET, SOCK\_STREAM, 0);
  2. 读取liu文件：fopen("liu","r");
  3. 发送数据sendto：sendto(sockfd,buf,sizeof(buf),0,(struct sockaddr \*)&adr\_srvr,sizeof(adr\_srvr));
  4. 关闭close：close(sockfd);

**客户端**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<netdb.h>

#include<errno.h>

#include<sys/types.h>

int port=8888;

int main()

{

int sockfd;

int i=0;

int z;

char buf[80],str1[80];

struct sockaddr\_in adr\_srvr;

FILE \*fp;

printf("打开文件......\n");

/\*以只读的方式打开liu文件\*/

fp=fopen("liu","r");

if(fp==NULL)

{

perror("打开文件失败");

exit(1);

}

printf("连接服务端...\n");

/\* 建立IP地址 \*/

adr\_srvr.sin\_family=AF\_INET;

adr\_srvr.sin\_port=htons(port);

adr\_srvr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

bzero(&(adr\_srvr.sin\_zero),8);

sockfd=socket(AF\_INET,SOCK\_DGRAM,0);

if(sockfd==-1)

{

perror("socket 出错");

exit(1);

}

printf("发送文件 ....\n");

/\* 读取三行数据，传给udpserver\*/

for(i=0;i<3;i++)

{

fgets(str1,80,fp);

printf("%d:%s",i,str1);

sprintf(buf,"%d:%s",i,str1);

z=sendto(sockfd,buf,sizeof(buf),0,(struct sockaddr \*)&adr\_srvr,

sizeof(adr\_srvr));

if(z<0)

{

perror("recvfrom 出错");

exit(1);

}

}

printf("发送.....\n");

sprintf(buf,"stop\n");

z=sendto(sockfd,buf,sizeof(buf),0,(struct sockaddr \*)&adr\_srvr,

sizeof(adr\_srvr));

if(z<0)

{

perror("sendto 出错");

exit(1);

}

fclose(fp);

close(sockfd);

exit(0);

}

**服务器端**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<netdb.h>

#include<errno.h>

#include<sys/types.h>

int port=8888;

int main()

{

int sockfd;

int len;

int z;

char buf[256];

struct sockaddr\_in adr\_inet;

struct sockaddr\_in adr\_clnt;

printf("等待客户端....\n");

/\* 建立IP地址 \*/

adr\_inet.sin\_family=AF\_INET;

adr\_inet.sin\_port=htons(port);

adr\_inet.sin\_addr.s\_addr =htonl(INADDR\_ANY);

bzero(&(adr\_inet.sin\_zero),8);

len=sizeof(adr\_clnt);

/\* 建立socket \*/

sockfd=socket(AF\_INET,SOCK\_DGRAM,0);

if(sockfd==-1)

{

perror("socket 出错");

exit(1);

}

/\* 绑定socket \*/

z=bind(sockfd,(struct sockaddr \*)&adr\_inet,sizeof(adr\_inet));

if(z==-1)

{

perror("bind 出错");

exit(1);

}

while(1)

{

/\* 接受传来的信息 \*/

z=recvfrom(sockfd,buf,sizeof(buf),0,(struct sockaddr \*)&adr\_clnt,&len);

if(z<0)

{

perror("recvfrom 出错");

exit(1);

}

buf[z]=0;

printf("接收:%s",buf);

/\* 收到stop字符串，终止连接\*/

if(strncmp(buf,"stop",4)==0)

{

printf("结束....\n");

break;

}

}

close(sockfd);

exit(0);

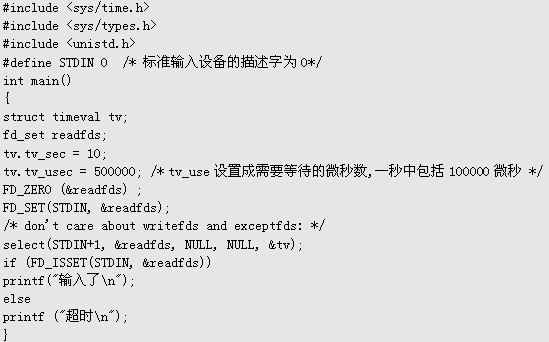
}

**8.3 网络高级编程**

* 在socket应用中，有一个很重要的特性，那就是如何处理阻塞，解决I/O多路利用问题。
* 在数据通信中，当服务器运行函数accept() 时，假设没有客户机连接请求到来，那么服务器就一直会停止在accept（）语句上，等待客户机连接请求到来，出现这样的情况就称为阻塞。

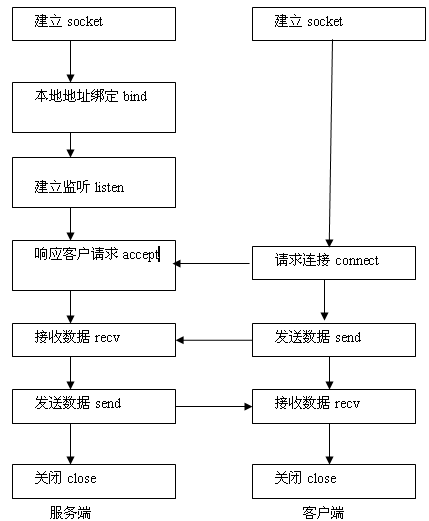
**例8.3：**程序运行当中，在10.5秒判断有没有按回车键，有则返回“输入了”，否则返回“超时”。

* 源程序代码：



**程序通过select函数在指定的时间内唤醒或结束进程，是处理阻塞的一种好方法。**

* **例8.4：**编写一个网络聊天程序。



**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*客户端**

#include <stdlib.h>

#include <stdio.h>

#include <netdb.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <string.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <fcntl.h>

#define SERVPORT 4444 /\*服务器监听端口号\*/

#define MAXDATASIZE 256 /\*最大同时连接请求数\*/

#define STDIN 0 /\*标准输入文件描述符\*/

int main(void)

{

int sockfd; /\*套接字描述符\*/

int recvbytes;

char buf[MAXDATASIZE]; /\*用于处理输入的缓冲区\*/

char \*str;

char name[MAXDATASIZE]; /\*定义用户名\*/

char send\_str[MAXDATASIZE]; /\*最多发出的字符不能超过MAXDATASIZE\*/

struct sockaddr\_in serv\_addr; /\*Internet套接字地址结构\*/

fd\_set rfd\_set, wfd\_set, efd\_set;

/\*被select()监视的读、写、异常处理的文件描述符集合\*/

struct timeval timeout;/\*本次select的超时结束时间\*/

int ret; /\*与server连接的结果\*/

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) { /\*错误检测\*/

perror("socket");

exit(1);

}

/\* 填充 sockaddr结构 \*/

bzero(&serv\_addr, sizeof(struct sockaddr\_in));

serv\_addr.sin\_family=AF\_INET;

serv\_addr.sin\_port=htons(SERVPORT);

inet\_aton("127.0.0.1", &serv\_addr.sin\_addr);

/\*serv\_addr.sin\_addr.s\_addr=inet\_addr("192.168.0.101");\*/

if (connect(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(struct sockaddr)) == -1)

{

/\*错误检测\*/

perror("connect");

exit(1);

}

fcntl(sockfd, F\_SETFD, O\_NONBLOCK);

printf("要聊天首先输入你的名字:");

scanf("%s",name);

name[strlen(name)] = '\0';

printf("%s: ",name);

fflush(stdout);

send(sockfd, name, strlen(name), 0);

/\*发送用户名到sockfd\*/

while (1)

{

FD\_ZERO(&rfd\_set);/\*将select()监视的读的文件描述符集合清除\*/

FD\_ZERO(&wfd\_set);/\*将select()监视的写的文件描述符集合清除\*/

FD\_ZERO(&efd\_set);/\*将select()监视的异常的文件描述符集合清除\*/

FD\_SET(STDIN, &rfd\_set);

/\*将标准输入文件描述符加到seletct()监视的读的文件描述符集合中\*/

FD\_SET(sockfd, &rfd\_set);

/\*将新建的描述符加到seletct()监视的读的文件描述符集合中\*/

FD\_SET(sockfd, &efd\_set);

/\*将新建的描述符加到seletct()监视的异常的文件描述符集合中\*/

timeout.tv\_sec = 10;/\*select在被监视窗口等待的秒数\*/

timeout.tv\_usec = 0;/\*select在被监视窗口等待的微秒数\*/

ret = select(sockfd + 1, &rfd\_set, &wfd\_set, &efd\_set, &timeout);

if (ret == 0) {

continue;

}

if (ret < 0) {

perror("select error: ");

exit(-1);

}

/\*判断是否已将标准输入文件描述符加到seletct()监视的读的文件描述符集合中\*/

if (FD\_ISSET(STDIN, &rfd\_set))

{

fgets(send\_str, 256, stdin);

send\_str[strlen(send\_str)-1] = '\0';

if (strncmp("quit", send\_str, 4) == 0) { /\*退出程序\*/

close(sockfd);

exit(0);

}

send(sockfd, send\_str, strlen(send\_str), 0);

}

/\*判断是否已将新建的描述符加到seletct()监视的读的文件描述符集合中\*/

if (FD\_ISSET(sockfd, &rfd\_set))

{

recvbytes=recv(sockfd, buf, MAXDATASIZE, 0);

if (recvbytes == 0)

{

close(sockfd);

exit(0);

}

buf[recvbytes] = '\0';

printf("Server: %s\n", buf);

printf("%s: ",name);

fflush(stdout);

}

/\*判断是否已将新建的描述符加到seletct()监视的异常的文件描述符集合中\*/

if (FD\_ISSET(sockfd, &efd\_set))

{

close(sockfd);

exit(0);

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*服务器端

#include <stdlib.h>

#include <stdio.h>

#include <netdb.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <string.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <fcntl.h>

#define MAXDATASIZE 256

#define SERVPORT 4444 /\*服务器监听端口号\*/

#define BACKLOG 1 /\*最大同时连接请求数\*/

#define STDIN 0 /\*标准输入文件描述符\*/

int main(void)

{

FILE \*fp; /\*定义文件类型指针fp\*/

int sockfd,client\_fd; /\*监听socket.sock\_fd,数据传输socket.new\_fd\*/

int sin\_size;

struct sockaddr\_in my\_addr, remote\_addr;/\*本机地址信息,客户地址信息\*/

char buf[256]; /\*用于聊天的缓冲区\*/

char buff[256]; /\*用于输入用户名的缓冲区\*/

char send\_str[256]; /\*最多发出的字符不能超过256\*/

int recvbytes;

fd\_set rfd\_set, wfd\_set, efd\_set;/\*被select()监视的读、写、异常处理的文件描述符集合\*/

struct timeval timeout; /\*本次select的超时结束时间\*/

int ret; /\*与client连接的结果\*/

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) { /\*错误检测\*/

perror("socket");

exit(1);

}

/\* 端填充 sockaddr结构 \*/

bzero(&my\_addr, sizeof(struct sockaddr\_in));

my\_addr.sin\_family=AF\_INET; /\*地址族\*/

my\_addr.sin\_port=htons(SERVPORT); /\*端口号为4444\*/

inet\_aton("127.0.0.1", &my\_addr.sin\_addr);

if (bind(sockfd, (struct sockaddr \*)&my\_addr, sizeof(struct sockaddr)) == -1) { /\*错误检测\*/

perror("bind");

exit(1);

}

if (listen(sockfd, BACKLOG) == -1) { /\*错误检测\*/

perror("listen");

exit(1);

}

sin\_size = sizeof(struct sockaddr\_in);

if ((client\_fd = accept(sockfd, (struct sockaddr \*)&remote\_addr, &sin\_size)) == -1) {

/\*错误检测\*/

perror("accept");

exit(1);

}

fcntl(client\_fd, F\_SETFD, O\_NONBLOCK);/\* 服务器设为非阻塞\*/

recvbytes=recv(client\_fd, buff, MAXDATASIZE, 0);

/\*接收从客户端传来的用户名\*/

buff[recvbytes] = '\0';

fflush(stdout);

/\*强制立即内容\*/

if((fp=fopen("name.txt","a+"))==NULL)

{

printf("can not open file,exit...\n");

return -1;

}

fprintf(fp,"%s\n",buff);

/\*将用户名写入name.txt中\*/

while (1) {

FD\_ZERO(&rfd\_set);/\*将select()监视的读的文件描述符集合清除\*/

FD\_ZERO(&wfd\_set);/\*将select()监视的写的文件描述符集合清除\*/

FD\_ZERO(&efd\_set);/\*将select()监视的异常的文件描述符集合清除\*/

FD\_SET(STDIN, &rfd\_set);

/\*将标准输入文件描述符加到seletct()监视的读的文件描述符集合中\*/

FD\_SET(client\_fd, &rfd\_set);

/\*将新建的描述符加到seletct()监视的读的文件描述符集合中\*/

FD\_SET(client\_fd, &wfd\_set);

/\*将新建的描述符加到seletct()监视的写的文件描述符集合中\*/

FD\_SET(client\_fd, &efd\_set);

/\*将新建的描述符加到seletct()监视的异常的文件描述符集合中\*/

timeout.tv\_sec = 10;/\*select在被监视窗口等待的秒数\*/

timeout.tv\_usec = 0;/\*select在被监视窗口等待的微秒数\*/

ret = select(client\_fd + 1, &rfd\_set, &wfd\_set, &efd\_set, &timeout);

if (ret == 0) {

continue;

}

if (ret < 0) {

perror("select error: ");

exit(-1);

}

/\*判断是否已将标准输入文件描述符加到seletct()监视的读的文件描述符集合中\*/

if(FD\_ISSET(STDIN, &rfd\_set))

{

fgets(send\_str, 256, stdin);/\*取从输入输入的内容\*/

send\_str[strlen(send\_str)-1] = '\0';

if (strncmp("quit", send\_str, 4) == 0) { /\*退出程序\*/

close(client\_fd);

close(sockfd); /\*关闭套接字\*/

exit(0);

}

send(client\_fd, send\_str, strlen(send\_str), 0);

}

/\*判断是否已将新建的描述符加到seletct()监视的读的文件描述符集合中\*/

if (FD\_ISSET(client\_fd, &rfd\_set))

{

recvbytes=recv(client\_fd, buf, MAXDATASIZE, 0);/\*接收从客户端传来的聊天内容\*/

if (recvbytes == 0) {

close(client\_fd);

close(sockfd); /\*关闭套接字\*/

exit(0);

}

buf[recvbytes] = '\0';

printf("%s:%s\n",buff,buf);

printf("Server: ");

fflush(stdout);

}

/\*判断是否已将新建的描述符加到seletct()监视的异常的文件描述符集合中\*/

if (FD\_ISSET(client\_fd, &efd\_set)) {

close(client\_fd); /\*关闭套接字\*/

exit(0);

}

}

}

## 实验结果分析

1、熟悉并记录程序执行结果。

2、写出自己的心得体会。

## 思考与实验

1. 在Linux系统下编写一个socket程序，要求服务端等待客户的连接请求.一旦有客户连接,服务器端打印出客户端的IP地址和端口,并且向服务器端发送欢迎信息和时间。
2. 编写一个基于TCP协议的网络通信程序，要求服务器通过socket连接后，并要求输入用户，判断为liu时，才向客户端发送字符串"Hello, you are connected!"。在服务器上显示客户端的IP地址或域名。
3. 一个以客户机/服务器模式工作，要求在客户端读取系统文件/etc/passwd内容，传送到服务端，服务器端接受字符串，并在显示器显示出来。