# Linux实现简单聊天程序

## 实验目的：

在看网络编程知识的时候看见一个用Windows实现的一个聊天程序，就选择用Linux试试能不能实现，在Linux系统下使用C语言与软件工程思想，实现一个简单的聊天程序，包含服务器端和客户端，用户名/密码注册和登录，支持版面聊天，支持点对点私聊，可以选择用户进行私聊。来熟悉在Linux系统所提供的网络通信接口及该系统下的编程思想，深入对计算机系统的理解，并切实提高软件设计开发的能力

## 实验原理：

简单的聊天室，所实现的就是当这个聊天室里面的某一个用户输入一段字符后，其他用户可以看到，因此聊天程序包括客户端和服务器端。每一个客户端就是一个客户，从终端采集信息传递到服务器并将服务器接收到的内容显示出来。

### 服务器结构：

首先是初始化服务器，使服务器进入监听状态：

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

bind(sockfd, (struct sockaddr \*)&servaddr,        sizeof(serv\_addr));

listen（socket， MAX\_CLIENT）;

服务器进入监听状态后，等待客户建立连接。若客户需要连接，也需要先进行网络部分 的初始化工作：

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

connect(sockfd, (struct sockaddr \*)

&serv\_addr, sizeof(serv\_addr))

当客户建立新连接的时候，服务器使用accept来接收该连接：

accept(sockfd, (struct sockaddr \*)&cli\_addr, &cli\_len);

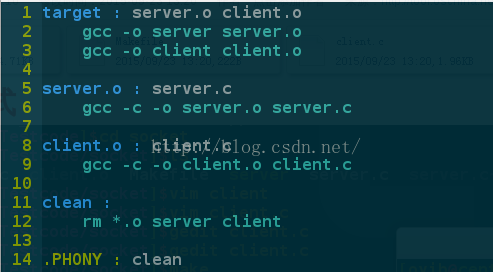
在服务器进入监听状态以后，使用select方法实现，该方法中的所有描述符都是阻 塞的。使用select判断一组文件描述符中时候有一个可读（写），如果没有就阻塞。直到有一个的时候被唤醒。

## 实验环境：

操作系统环境：Ubuntu-10.04 Linux

集成开发环境：Eclipse

### MakeFile文件：



## 代码实现：

服务器端：

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h> //数据类型定义

#include <sys/stat.h> //文件属性

#include <netinet/in.h> //定义数据结构sockaddr\_in

#include <sys/socket.h> //提供socket函数和数据结构

#include <string.h>

#include <unistd.h>

#include <signal.h>

#include <sys/ipc.h>

#include <errno.h>

#include <sys/shm.h> //共享内存

#include <time.h>

#define PERM S\_IRUSR | S\_IWUSR //用户读写

#define MYPORT 3490 //通信端口

#define BACKLOG 10 //定义服务器段可以连接的最大客户数

#define WELCOME "|---------------Welcome to the chat room!----------------|"//当客户端连接服务端时，向客户端发送此字符串

//将int类型转换成char\*类型

void itoa(int i, char \*string)

{

int mask = 1;

while (i / mask >= 10)

mask \*= 10;

while (mask > 0)

{

\*string++ = i / mask + '0';

i %= mask;

mask /= 10;

}

\*string = '\0';

}

//得到当前系统的时间

void get\_cur\_time(char \*time\_str)

{

struct timeval now;

gettimeofday(&now, NULL);

strcpy(time\_str, ctime(&now.tv\_sec));

}

//创建共享存储区

int shm\_create()

{

int shmid;

//shmid = shmget(IPC\_PRIVATE, 1024, PERM);

if ((shmid = shmget(IPC\_PRIVATE, 1024, PERM)) == -1)

{

fprintf(stderr, "Create Share Memory Error:%s\n\a", strerror(errno));

exit(1);

}

return shmid;

}

//端口绑定函数。创建套接字，并绑定到指定端口

int bindPort(unsigned short int port)

{

int sockfd;

struct sockaddr\_in my\_addr;

sockfd = socket(AF\_INET, SOCK\_STREAM, 0); //创建基于流套接字

//bzero(&(my\_addr.sin\_zero),0);

bzero(&my\_addr, sizeof(my\_addr));

my\_addr.sin\_family = AF\_INET; //IPV4协议族

my\_addr.sin\_port = htons(port); //转换端口为网络字节序

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

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

{

perror("fail to bind");

exit(1);

}

printf("bind success!\n");

return sockfd;

}

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

{

int sockfd, clientfd; //监听套接字、客户套接字

int sin\_size, recvbytes;

pid\_t pid, ppid; //定义父子进程标记

char \*buf, \*read\_addr, \*write\_addr, \*temp, \*time\_str; //需要用到的缓冲区

struct sockaddr\_in their\_addr; //定义地址结构

int shmid;

shmid = shm\_create(); //创建共享存储区

temp = (char \*)malloc(255);

time\_str = (char \*)malloc(50);

sockfd = bindPort(MYPORT); //绑定端口

get\_cur\_time(time\_str);

printf("Time is : %s\n", time\_str);

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

{

//在指定端口上监听

perror("fail to listen");

exit(1);

}

printf("listen....\n");

while (1)

{

/\*

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

{

perror("fail to listen");

exit(1);

}

\*/

//接受一个客户端的连接请求

if ((clientfd = accept(sockfd, (struct sockaddr \*)&their\_addr, &sin\_size)) == -1)

{

perror("fail to accept");

exit(1);

}

//得到客户端的IP地址输出

char address[20];

inet\_ntop(AF\_INET, &their\_addr.sin\_addr, address, sizeof(address));

printf("accept from %s\n", address);

send(clientfd, WELCOME, strlen(WELCOME), 0); //发送问候信息

buf = (char \*)malloc(255);

ppid = fork(); //创建子进程

if (ppid == 0) //子进程

{

pid = fork(); //子进程创建子进程

while (1)

{

if (pid > 0)

{

//buf = (char \*)malloc(255);

//父进程用于接收信息

memset(buf, 0, 255);

printf("OK\n");

if ((recvbytes = recv(clientfd, buf, 255, 0)) <= 0)

{

perror("fail to recv");

close(clientfd);

raise(SIGKILL);

exit(1);

}

write\_addr = shmat(shmid, 0, 0);

//shmat将shmid所代表的全局的共享存储区关联到本进程的进程空 间

memset(write\_addr, '\0', 1024);

//把接收到的消息存入共享存储区中

strncpy(write\_addr, buf, 1024);

//把接收到的消息连接此刻的时间字符串输出到标准输出

get\_cur\_time(time\_str);

strcat(buf, time\_str);

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

}

else if (pid == 0)

{

//子进程用于发送消息

sleep(1); //子进程先等待父进程把接收到的信息存入共享存储区

read\_addr = shmat(shmid, 0, 0); //读取共享存储区的内容

//temp存储上次读取过的内容,每次先判断是否已经读取过该消息

if (strcmp(temp, read\_addr) != 0)

{

strcpy(temp, read\_addr); //更新temp，表示已经读取过该消息

get\_cur\_time(time\_str);

strcat(read\_addr, time\_str);

if (send(clientfd, read\_addr, strlen(read\_addr), 0) == -1)

{

perror("fail to send");

exit(1);

}

memset(read\_addr, '\0', 1024);

strcpy(read\_addr, temp);

}

}

else

perror("fai to fork");

}

}

}

printf("------------------------------------\n");

free(buf);

close(sockfd);

close(clientfd);

return 0;

}

客户端：

#include <stdio.h>

#include <netinet/in.h> //定义数据结构sockaddr\_in

#include <sys/socket.h> //定义socket函数以及数据结构

#include <sys/types.h>

#include <string.h>

#include <stdlib.h>

#include <netdb.h>

#include <unistd.h>

#include <signal.h>

#include <time.h>

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

{

struct sockaddr\_in clientaddr; //定义地址结构

pid\_t pid;

int clientfd, sendbytes, recvbytes;

struct hostent \*host; //主机信息数据结构

char \*buf, \*buf\_read;

if (argc < 4)

{

printf("wrong usage");

printf("%s host port name\n", argv[0]);

exit(1);

}

host = gethostbyname(argv[1]);

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

{

perror("fail to create socket");

exit(1);

}

bzero(&clientaddr, sizeof(clientaddr));

clientaddr.sin\_family = AF\_INET;

clientaddr.sin\_port = htons((uint16\_t)atoi(argv[2]));

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

//客户端连接服务端

if (connect(clientfd, (struct sockaddr \*)&clientaddr, sizeof(struct sockaddr)) == -1)

{

perror("fail to connect");

exit(1);

}

buf = (char \*)malloc(120);

memset(buf, 0, 120);

buf\_read = (char \*)malloc(100);

if (recv(clientfd, buf, 100, 0) == -1)

{

perror("fail to recv");

exit(1);

}

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

pid = fork();

while (1)

{

if (pid > 0)

{

//父进程发送消息

strcpy(buf, argv[3]);

strcat(buf, ":");

memset(buf\_read, 0, 100);

fgets(buf\_read, 100, stdin);

strncat(buf, buf\_read, strlen(buf\_read) - 1);

if ((sendbytes = send(clientfd, buf, strlen(buf), 0)) == -1)

{

perror("fail to send");

exit(1);

}

}

else if (pid == 0)

{

//子进程接受消息

memset(buf, 0, 100);

if (recv(clientfd, buf, 100, 0) <= 0)

{

perror("fail to recv");

close(clientfd);

raise(SIGSTOP);

exit(1);

}

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

}

else

perror("fork error");

}

close(clientfd);

return 0;

}