IRC 聊天系统技术报告

linux 大型实验

软件工程 0903

徐飞黎

200926630322

2012/12/29

目录

一. 实验介绍 3
二. 实验原理 3
2.1 C/S 架构 3
2.2 套接字 3
2.2.1 socket 配置 3
2.2.2 socket 相关的数据结构 3
2.2.3 发送消息方式 4
2.3 服务端接受数据方式 4
2.3.1 fcnl 4
2.3.2 select 机制 4
2.3.3 比较 6
三. 开发环境 6
四. 开发设计 6
4.1 系统架构流程 6
4.2 运行流程 7
4.2.1 服务端 7
4.2.2 客户端 8
4.3 客户端和服务端的字节交流 9
4.3.1 服务端接受客户端字节流 9
4.3.2 客户端发送给服务端字节流 10
4.4 配置文件 11
五. 代码 11
5.1 客户端： 11
5.1.1 注册： 11
5.1.2 登录 13
5.1.3 私聊 14
5.1.4 初始化链接地址 端口 15
5.2 服务端 16
5.2.1 注册函数—写入文件 16
5.2.2 登录检验 17
5.2.3 初始化服务端口 17
5.2.4 来自客户端的字节流分析 18
六. 运行演示 22
七. 实验总结 22

一. 实验介绍

本实验实现了简单的 IRC 聊天功能。在客户端和服务端建立字符流套接字连接，遵循协议为 TCP/IP 协议。

该程序分为客户端和服务端，各在终端字符界面执行。

功能为：1. 支持用户管理，用户名，密码注册和登录。

2. 支持版面聊天，版面聊天类似于聊天室，某客户发言，可被其他用户看到。
3. 支持点对点私聊。
4. 支持在线留言。
5. 支持聊天记录的查看（包括版面聊天和私聊）。

二. 实验原理

2.1 C/S 架构

聊天程序是一个 C/S 结构的程序，首先启动服务器，然后用户使用帐号密码进行登陆连接。它的优点是速度快，缺点是当服务器进行更新时，客户端必须更新。

客户端：包含了程序启动初始化的信息，这里包括将连接的服务端 IP 地址 ：127.0.0.1 ，端口号为 9999

服务端:包含了程序启动初始化的信息,服务器的 IP 地址 127.0.0.1 端口号为 9999，最大客户连接数为 10.

以上的数据写入代码,方便直接运行程序,简单。不足之处是当要修改 IP 地址，端口号时需要在代码中修改。

2.2 套接字

2.2.1 socket 配置

客户端：socket 保存链接的服务器信息，包括 IP 地址，端口号等，用 connect()进行连接。

服务端 ：需要保存来自客户端的 socket 信息，包括 IP 地址，端口号等，同时也要保存自身的 socket 信息，用 bind()绑定。

2.2.2 socket 相关的数据结构

```
struct sockaddr_in
 {
  shortint sin_family;/*通信协议*/
  unsigned shortint sin_port ;/*端口号*/
  struct in_addr sin_addr ; /*Internet 地址*/
  unsigned char sin_zero[8] ;/*暂不用*/
 }

struct in_addr
 {
```

```
    in_addr_t s_addr ;/*存储 32 位的 IP 地址*/
}
```

2.2.3 发送消息方式

实验采用 TCP/IP 面向连接，所有客户的数据都需要经过服务端的识别，处理，转发。所以说道是悄悄话（私聊），其实服务端还是知道消息内容的。

2.3 服务端接受数据方式

服务端接受客户端的数据有堵塞和非堵塞两种。

对一个文件描述符指定的文件或者设备，有 2 中工作方式：阻塞和非堵塞，在缺省的情况下文件描述符处于堵塞状态。

在本简易聊天实验中，服务器轮流查询与客户端建立 socket，一旦可读就将该 socket 中的字符读出来，向其他客户端转发。并且，服务端还要随时查看是否有新的客户试图进行连接，这样，如果服务端在任何一个地方堵塞了，其他客户端发送的内容就会受到影响，新的客户端试图建立连接也会受到忽视。所以，可以采用 fcnl 将该文件描述符变为非堵塞的。

2.3.1 fcnl

无论是服务端还是客户端，他们都不停的轮流查询各个文件描述符，一旦可读就读入并进行处理。这样的程序，不停的执行，只要有 cpu，就不会放过。这对系统消耗非常大。

2.3.2 select 机制

select 方法中，所有文件描述符都是堵塞的。使用 select 判断一组文件描述符中是否有一个可读（写），如果没有就堵塞，直到有一个的时候被唤醒。

客户端

只需处理两个文件描述符(一个是来自服务端的消息,一个是自己输入),因此，需要判断是否有可读写的文件描述符只需要加入两项：

```
FD_ZERO(sockset) ; //清空 socket
FD_SET(sockfd, sockset); //把 sockfd 加入到 sockse 集合中
FD_SET(0,sockset); // 0 表述消息来自于自己，把标准输入加入到 sockset
                   集中
```

客户端的处理

```
   while (! exit){
   select(sockfd+1 , &sockfdset ,NULL ,NULL,NULL);
  //此时该函数将堵塞，知道标准输入或者 sockfd 中有一个可读为止
  //第一个参数是 0 和 sockfd 中的最大值+1
 //第二个参数是读集，也就是 sockset
 //第三，四个参数表述写集 和异常集
 //第五个参数表示超时时间，NULL 表示永不超时。
 //当 select 因为可读返回时候，sockset 包含的只是可读的那些文件描述符

if(FD_ISSET(sockfd ,&sockset)){
    //FD_ISSET 宏判断 sockfd 是否属于可读的文件描述符

  从 sockfd 中读入，输出到标准输出
   }

if(FD_ISSET(0 ,&sockset)){
    //FD_ISSET 宏判断 sockfd 是否属于可读的文件描述符
  从标准输入中读入，输出到 sockfd
  }
   重设 sockset。(将 sockset 清空，并将 sockfd 和 0 加入)
}
```

服务端

```
 FD_ZERO(sockset) ;//清空 socket
 FD_SET(sockfd, sockset); //把 sockfd 加入到 sockse 集合中
 for(所有有效连接) {
    FD_SET(userfd[i],sockset); // 来自客户端的连接
 }
  maxfd = 最大的文件描述符 +1 ;
```

服务端的处理

```
  while (1){
    select(maxfd , &sockfdset ,NULL ,NULL,NULL);
   if(FD_ISSET(sockfd ,&sockset)){
    //有新链接
     建立新连接，并将该连接描述符加入到 sockset 中
   }
   for(所有有效连接) {
    if(FD_ISSET(userfd[i] ,&sockset)){
      //连接中有字符可读
      相应的转发或者处理。
       }
   }
   重设 sockset，将所有有效客户连接加入 sockset 集合中。
```

}

2.3.3 比较

select 机制，当没有字符可读时，程序处于堵塞状态，最小程度的占用 CPU 资源，在同一台机器上执行服务器和若干个客户时候，系统负载只有 0.1 左右，而采用 fcntl 方法，只运行一个服务器，系统负载就可以达到 1.5 左右。

基于以上原因 ，本实验中采用 select 堵塞机制。

三. 开发环境

操作系统：ubuntu 10.04
开发工具：vim gdb eclipse+cdt 集成环境 。
版本控制器：git

四. 开发设计

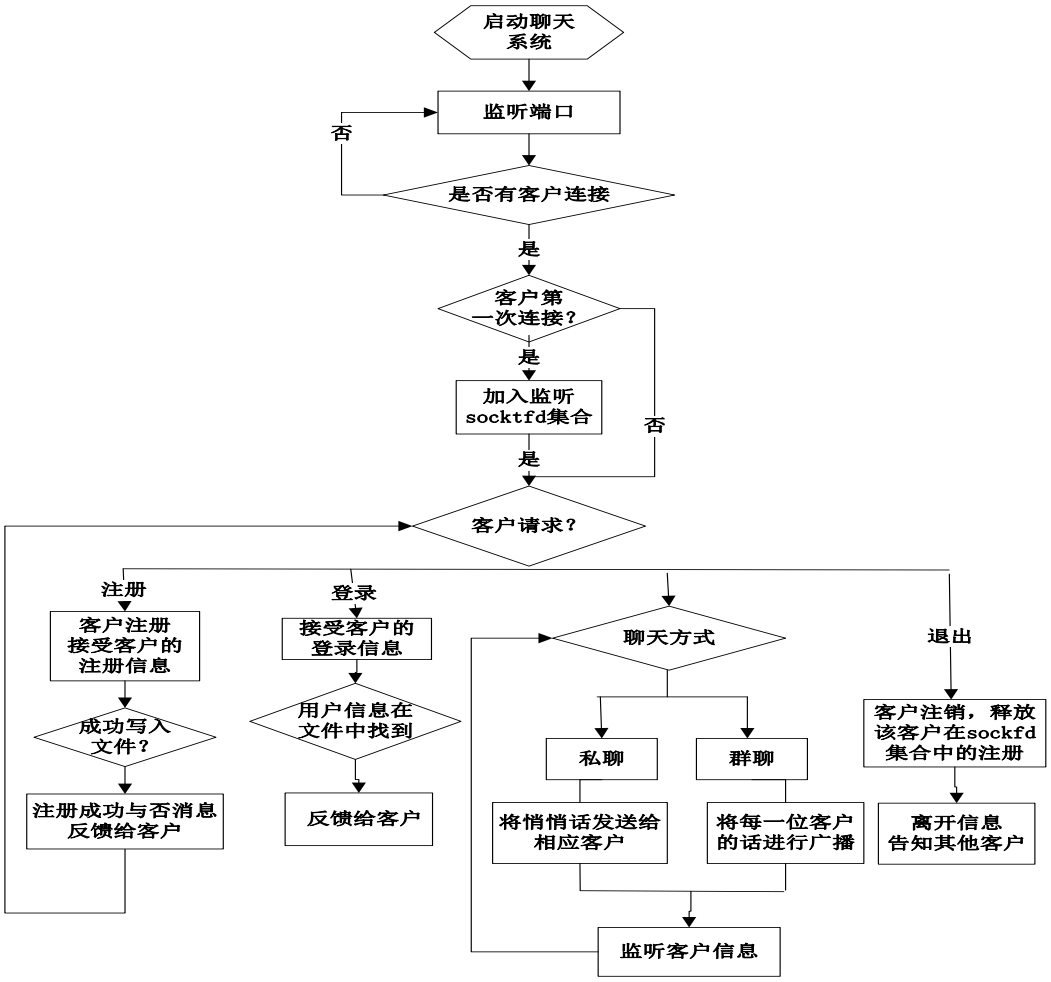
4.1 系统架构流程

服务端	客户端
初始化服务器地址，端口，最大连接数	网络部分初始化
sockfd = socket(AF_INET,SOCK_STREAM,0) 首先建立一个 socket，AF_INET 即使用 TCP/IP 协议族， SOCK_STREAM 类型提供了顺序的，可靠的，基于自节流的全双工共连接。	sockfd = socket(AF_INET,SOCK_STREAM,0) //客户端建立一个 sockfd，其参数与服务器相同。
bind(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr)); 将这个 socket 与某个地址进行绑定，在这里默认为本地服务器,即 127.0.0.1	connect=(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr)); 客户使用 connect 建立一个连接。 serv_addr 中的变量分别设置为： 1. sin_family = AF_INEF 协议族同 socket 2. sin_addr = inet_addr (SERV_HOST_ADDR) 地址为 server 所在计算机的地址。 3. sin_port = htons (SERV_HOST_PORT) 端口为服务器监听的端口。
listen(sockfd,MAX_LINK); 绑定地址后，服务器进入监听状态，MAX_LINK 为最大连接数。 然后等待客户建立连接。	
服务器用 accept 来接收客户的连接 accept(sockfd,(struct sockaddr*) & cli_addr,&cli_len); 函数返回时，cli_addr 中保留的是该连	

接对方的信息，包括对方的 IP 地址和对方使用的端口。 accept 返回一个新的文件描述符	

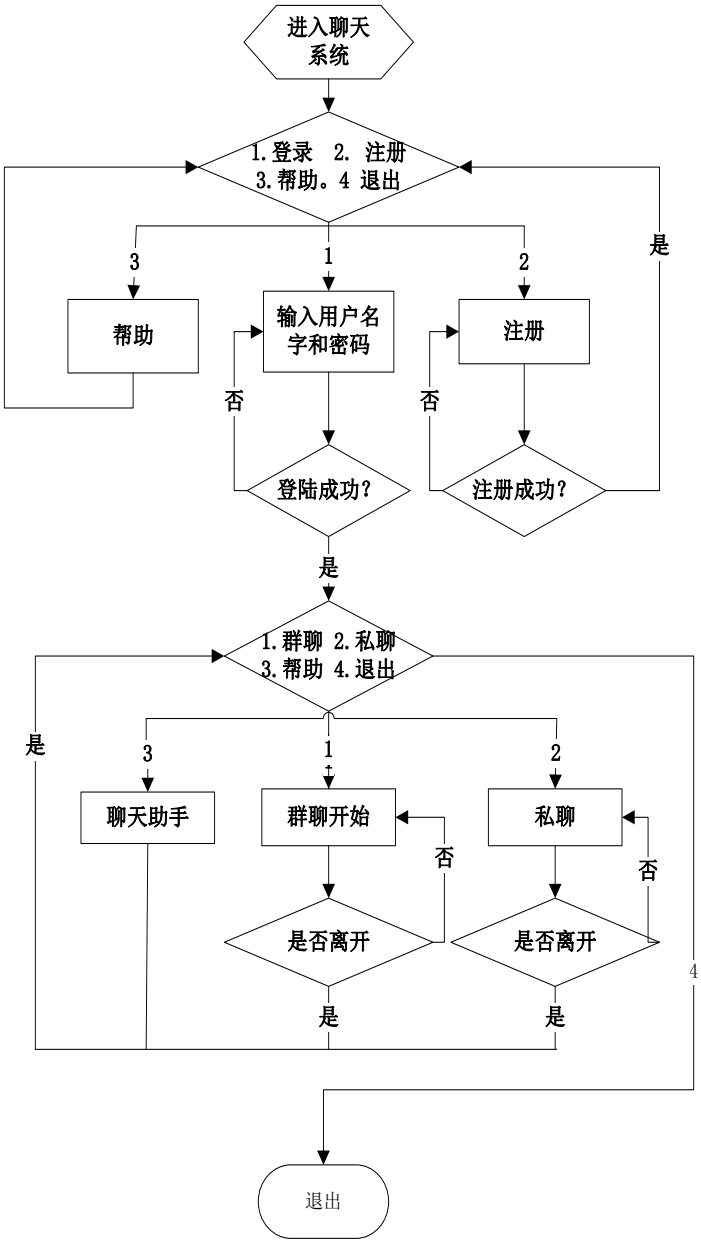
4.2 运行流程

4.2.1 服务端



服务端自开启服务后，一直处于无限循环状态，不停监听端口，对客户端的请求信息进行处理。

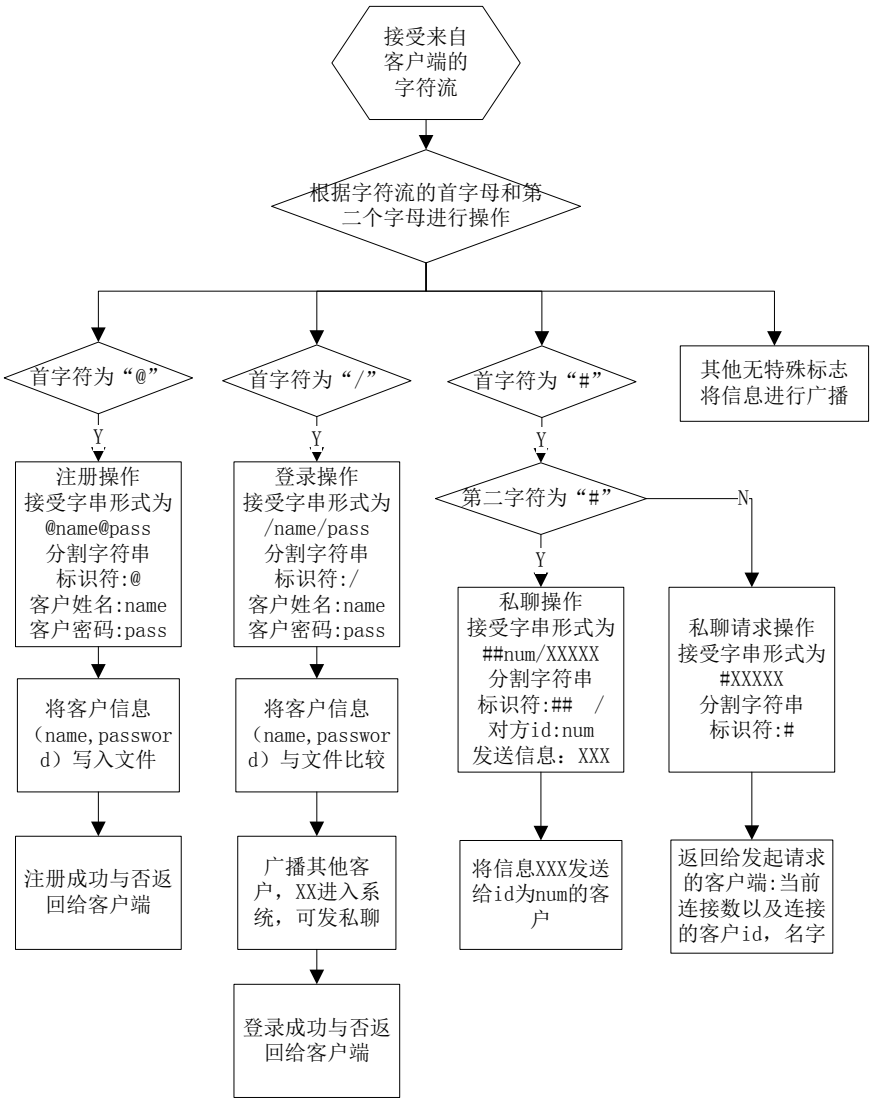
4.2.2 客户端



客户端只有登录接收到服务端的认证许可后，才能进行聊天。若帐号不存在，必须提前进行注册。

4.3 客户端和服务端的字节交流

4.3.1 服务端接受客户端字节流



服务端接受到字串分为下列一种情况：

（XXXX, YYYY, ZZZZ 表示客户发来的局部信息表示）

1. @XXXX@YYYYYY@ZZZZZZ 首字符为“@”，表示客户注册帐号

   服务端的处理：将字串根据@进行分割，存储为 name（XXXX 部分），password（YYYY）两部分，并写入文件，将注册成功与否的信息反馈给客户。

2. /XXXX/YYYY/ZZZZZ 首字符为“/”，表示客户申请登录帐号检验

   服务端处理：将字串根据/进行分割，存储为 name（XXXX 部分），password（YYYY）两部分，从文件中连续读取两个字段，与 name 和 password 进行比较，如果找到完全相等的，表示账号存在并且登录正确，否则表示帐号不存在或者密码错误。并且将登录成功与否的信息反馈给客户。

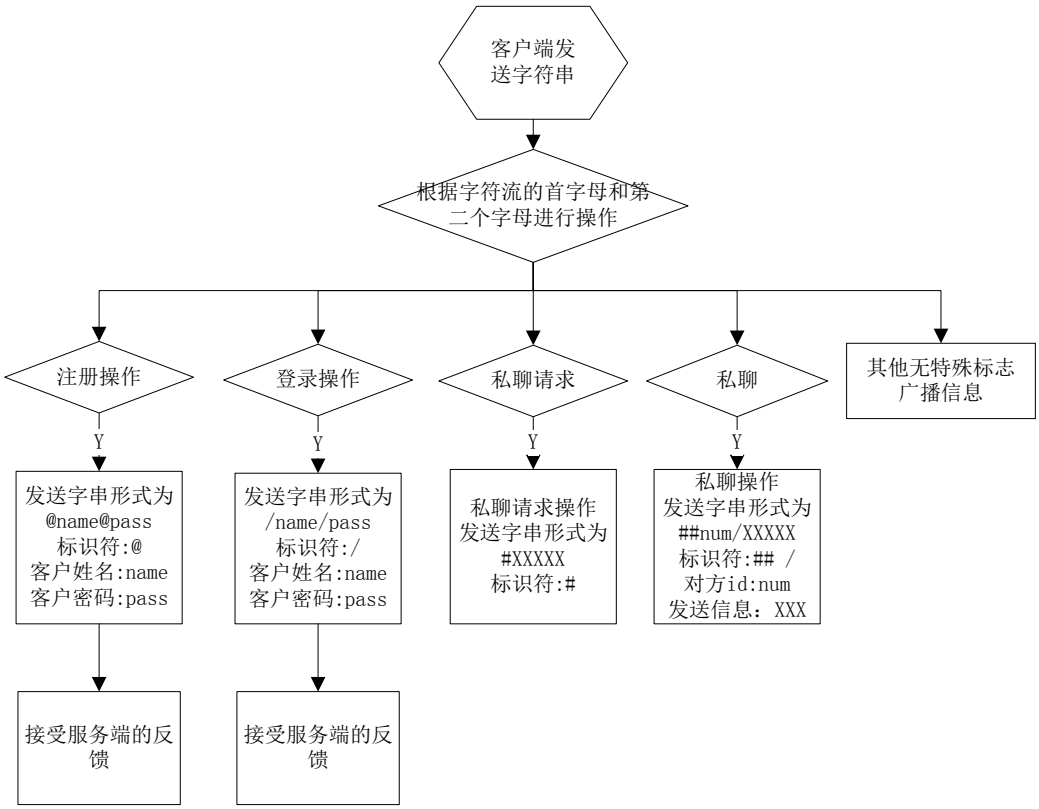
   如果登录成功，把该客户登录的信息广播给在线的用户，允许他们对新客户的欢迎及留言。

3. #XXXXXX 首字符为“#”，并且第二个字符不为“#”，表示客户发起私聊请求

服务端处理：将当前在线客户数，在线客户的 id 号，姓名均发送给该客户，如果当前在线只有该客户一个人，则不能满足该客户的私聊请求。

4. ##X/YYYYY 首字符和第二个字符均为“#”， 表示客户发送悄悄话。
   服务端处理：先将字串根据#和/进行分割 ，提取客户发送悄悄话的 id 号（X 部分），然后将消息 YYYY 发送给 Xsocked 描述符的客户。
5. 若没有其他特殊标志，表示发送的消息为广播消息
   服务端处理：只需将接收到的字符流转发给其他在线客户

4.3.2 客户端发送给服务端字节流



客户端端发送到字串分为下列一种情况：

（XXXX, YYYY, ZZZZ 表示客户发送的局部信息表示）

1. 客户注册帐号
   输入用户名 XXXX 和密码 YYYY（密码需要输入两遍，用来确认），程序自动将组装成字符串 @XXXX@YYYYYY@ZZZZZZ
   然后接受来自服务端的反馈消息，进行下一步动作。
2. 客户申请登录帐号检验
   输入用户名 XXXX 和密码 YYYY，程序自动将组装成字符串 /XXXX/YYYY/ZZZZZ
   然后接受来自服务端的反馈消息。只有接受到登录信息，才可以进入聊天程序。
3. 客户发起私聊请求
   客户端程序自动发送给服务端 #XXXXXX

接受到来自服务端的信息：当前在线客户数，在线客户的 id 号，姓名
如果当前在线只有该客户一个人，则不能进行私聊。

4. 客户私聊
输入聊天对象的 id 号，id 号即在发送请求的时候可以获得。
输入需要发送的内容 YYYYY，客户端程序自动组装成 ##X/YYYYY，（X 表示客户的 id 号，YYYY 表示客户发送悄悄话内容。）
5. 若没有其他特殊标志，表示发送的消息为广播消息
该消息所有在线用户都可看到。

4.4 配置文件

客户端

保留每个用户的聊天记录文件。 即在登陆成功之后，客户端自动生成一个以用文件名字的文件。每一个用户保留一个聊天记录文件，内容包括私聊和版面聊天的息。

文件保存在客户端。

服务端

client.txt 用来保存客户的帐号和密码信息。
文件保存在服务端。

五. 代码

因为代码较长，所以只截取了部分。

5.1 客户端：

5.1.1 注册：

```
/*注册函数*/
int regist(int sockfd) {
    char name[100];
    char password[100];
    char password2[100];
    char buffer[100];

    int flag;
    char str[100];

    /**
     * 假设注册帐号为Jim,密码为 123456
     * 发送给服务端的识别为  @Jim@123456
     * */
    system("clear");
```

```
printf("\t*********************************************\n");
printf("\t**              ....注册新帐号....   \t **\n");
printf("\t*********************************************\n");
printf("\t**\t            请输入帐号              \t **\n\t--> ");
scanf("%s", name);
printf("\t**\t            请输入密码              \t **\n\t-->");
scanf("%s", password);
printf("\t**\t            请确认密码              \t **\n\t-->");
scanf("%s", password2);
while (strcmp(password, password2) != 0) {
    printf("\t**\t  两次输入密码不一致，请重试\t **\n");
    printf("\t**\t            请输入密码              \t **\n\t-->");
    scanf("%s", password);
    printf("\t**\t            请确认密码              \t **\n\t-->");
    scanf("%s", password2);
}
//拼接字符串，发送到服务器
strcpy(buffer, "@");
strcat(buffer, name);
strcat(buffer, "@");
strcat(buffer, password); //字符串拼接
strcat(buffer, "@");
flag = write(sockfd, buffer, strlen(buffer)); //发送给服务器

if (flag <= 0) {
    printf("\t*********************************************\n");
    printf("\t**          连接超时，请重试.....    \t **\n");
    printf("\t*********************************************\n");
    fflush(stdout);
    return 0;
}

flag = recv(sockfd, str, 1024, 0);
str[flag] = '\0';
if (flag <= 0) {
    printf("\t*********************************************\n");
    printf("\t**          连接超时，请重试.....    \t **\n");
    printf("\t*********************************************\n");
    fflush(stdout);
    return 0;
} else if (strcmp("0", str) == 0) {
    printf("\t*********************************************\n");
    printf("\t**\t  注册失败，请重试.....\t **\n");
    printf("\t*********************************************\n");
```

```
            sleep(2);
            fflush(stdout);
            return 0;
        } else {

            printf("\t*********************************************\n");
            printf("\t**\t  ......注册成功.....          \t **\n");
            printf("\t*********************************************\n");
            sleep(2);
            fflush(stdout);
            return 1;
        }
}
```


5.1.2 登录

```
        /*登录函数*/
int login(int sockfd, char * name) {
        char password[100];
        char buffer[100];
        int flag;
        char str[100];

        /**
         * 假设帐号为Jim,密码为 123456
         * 发送给服务端的识别为  /Jim/123456
         * */

        printf("请输入您的密码 :");

        //fgets(password, MAXBUF, stdin);
        scanf("%s", password);
        strcpy(buffer, "/");
        strcat(buffer, name);
        strcat(buffer, "/");
        strcat(buffer, password); //字符串拼接
        strcat(buffer, "/");
        flag = write(sockfd, buffer, strlen(buffer)); //发送给服务器
        if (flag <= 0) {
            printf("连接超时，请重试.....\n");
            fflush(stdout);
            return 0;
        }
```

```
        flag = recv(sockfd, str, 1024, 0);
        str[flag] = '\0';

        if (flag <= 0) {
                printf("服务器忙碌，请重试.....");
                fflush(stdout);
                return 0;
        } else if (strcmp("0", str) == 0) {
                printf("登录失败，请检查用户名或者密码.....\n");
                fflush(stdout);
                return 0;
        } else {

                printf("登录成功.....\n");
                fflush(stdout);
                return 1;
        }
}
```

5.1.3 私聊

```
/*私聊启动函数*/
int startP2P(int sockfd) {
        char temp[MAX_BUF];
        int flag;

        char str[MAX_BUF];
        bzero(temp, MAX_BUF);
        strcpy(temp, "#");

        /*
         * 提出私聊请求的标志 ：      #2/XXXXXXXXX
         * 发送私聊信息的标志 ：     ##2/XXXXXXXX
         * */

        flag = write(sockfd, temp, strlen(temp)); //发送给服务器

        if (flag <= 0) {
                printf("\t*********************************************\n");
                printf("\t**          连接超时，请重试.....    \t **\n");
                printf("\t*********************************************\n");
                fflush(stdout);
```

```
            return 0;
        }

        bzero(temp, MAX_BUF);
        flag = read(sockfd, temp, MAX_BUF);
        str[flag] = '\0';
        fflush(stdout);
        if (flag <= 0) {
            printf("\t**********************************************\n");
            printf("\t**          连接超时，请重试.....    \t **\n");
            printf("\t**********************************************\n");
            fflush(stdout);
        } else if (strcmp(temp, "0") == 0) {
            system("clear");
            printf("\t**********************************************\n");
            printf("\t**    当前在线人数为 0，无法进行私聊...\t **\n");
            printf("\t**********************************************\n");
            sleep(2);
            return 0;
        }
        system("clear");
        printf("\t**             私聊频道启动成功.....       \t**\n");
        printf("\t**  当前在线好友为: %s\t**\n", temp);
        fflush(stdout);
        return 1;
}
```

5.1.4 初始化链接地址 端口

```
int main(int argc, char **argv) {
        int sockfd, len;
        struct sockaddr_in dest; //服务器地址，包括端口
        char str[MAX_BUF]; //发送的字符缓冲区
        fd_set sockset;
        struct timeval tv; //超时限制
        int recvlen;
        int myport = 9999;
        char name[MAX_NAME], temp[MAX_BUF];
        int loginFlag, mychoice = 0;
        char friend[MAX_BUF]; //存储私聊对象的socket描述符号

        /* 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(myport);
        char date[] = "127.0.0.1";
        if (inet_aton(date, (struct in_addr *) &dest.sin_addr.s_addr) == 0) {
            perror(date);
            exit(errno);
        }

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

        /* 把集合清空 */
        FD_ZERO(&sockset);
        /* 把标准输入句柄 0 加入到集合中 */
        FD_SET(0, &sockset);
        /* 把当前连接句柄sockfd加入到集合中 */
        FD_SET(sockfd, &sockset);
            /* 开始进入聊天程序*/
```

5.2 服务端

5.2.1 注册函数—写入文件

```
/*客户注册写入文件函数*/
int regist(char username[30], char password[20]) {

    FILE *cfptr;//文件指针
    if ((cfptr = fopen("client.txt", "a+")) == NULL) {
        printf("File client.txt could not be opened\n");
        fclose(cfptr);
        return 0;
    } else {
        fprintf(cfptr, "%s %s\n", username, password);
```

```
            fclose(cfptr);
            return 1;
        }
        fclose(cfptr);
        return 0;
}
```


5.2.2 登录检验

```
/*客户名字密码检验，是否已经注册（与文件内数据比较）*/
int login(char username[30], char password[20]) {
        char user[30];
        char pass[20];
        FILE *cfptr;//文件指针
        if ((cfptr = fopen("client.txt", "r")) == NULL) {
                printf("File client.txt could not be opened\n");
                fclose(cfptr);
                return 0;
        } else {
                while (!feof(cfptr)) {

                        fscanf(cfptr, "%s%s", user, pass);

                        if ((strcmp(username, user) == 0) && (strcmp(password, pass) == 0)) {

                                fclose(cfptr);
                                return 1;
                        }
                }
        }

        fclose(cfptr);
        return 0;
}
```

5.2.3 初始化服务端口

```
    char *notice;
        int sockfd, new_fd;
        //int socketNum[MAX_LINK];    //保存客户端连接数
        int user_link[MAX_LINK];

        int userfd[MAX_LINK]; //保存连接客户端的 socket 描述符号
        char clientName[MAX_LINK][MAX_NAME];
```

```
char line[MAX_BUF];
char temp[MAX_BUF];

int userCount, i, j;
unsigned int cli_len;
struct sockaddr_in server_addr, client_addr;//网络地址结构体，包括端口号，IP 地址
int port = 9999; //服务器端口号

int flag;
char strfd[10]; //将 int 型的 userfd 转化为 字符串

int length; //标志符，用来判断当前读写是否成功
fd_set sockset;
int maxfd = -1; //用来标志当前连接的最大描述符

char *nam;              //分割字符串得到姓名和密码，临时保存变量
char *pass;
char *tokenPtr;

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

bzero(&server_addr, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(port);
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);

//绑定端口号
if (bind(sockfd, (struct sockaddr *) &server_addr, sizeof(struct sockaddr))
                < 0) {
        perror("bind");
        exit(1);
}
```

5.2.4 来自客户端的字节流分析

```
while (1) {
        select(maxfd, &sockset, NULL, NULL, NULL);
        //如果该 client 已经连接过服务器
        if (FD_ISSET(sockfd,&sockset) && (userCount = clinkNumber(user_link))
                        >= 0) {
                new_fd = accept(sockfd, (struct sockaddr*) &client_addr, &cli_len);
```

```
        if (new_fd < 0) {
            user_link[userCount] = 0;
            printf("%d 连接失败\n", new_fd);
            fflush(stdout);
        } else {

            user_link[userCount] = 1; //标志端口存在
            userfd[userCount] = new_fd;
            FD_SET(new_fd ,&sockset); //加入端口集合
            if (maxfd < (new_fd + 1))
                maxfd = new_fd + 1;
            printf("\n----------用户进程描述符号为 %d 连接服务器----------", new_fd);
            fflush(stdout);
        }
    }
    //监听已注册的端口是否有数据发送
    for (i = 0; i < MAX_LINK; ++i) {
        if ((user_link[i] == 1) && (FD_ISSET(userfd[i],&sockset))) {
            length = read(userfd[i], line, MAX_BUF);
            if (length == 0) //client 的 socket 已经关闭
            {
                /*注销已经关闭的端口*/
                printf("\n----------%s 已经注销----------\n", clientName[i]);
                fflush(stdout);
                user_link[i] = 0;
                clientName[i][0] = '\0';
                FD_CLR(userfd[i],&sockset);
            } else if (length > 0) {
                strcpy(temp, line);
                line[length] = '\0';
                temp[length] = '\0';
                /*根据不同的字符串提示头进行操作*/

                /*注册*/
                if (line[0] == '@') {
                    nam = strtok(temp, "@");
                    pass = strtok(NULL, "@");
                    flag = regist(nam, pass);
                    if (flag == 1) {
                        printf("\n 注册成功\n ");
                        fflush(stdout);
                        notice = "1";
                        write(userfd[i], notice, strlen(notice));
```

```
        } else {
            printf("\n 注册失败\n ");
            fflush(stdout);
            notice = "0";
            write(userfd[i], notice, strlen(notice));
        }
    }

    /*登录*/
    else if (line[0] == '/') {
        nam = strtok(temp, "/");
        pass = strtok(NULL, "/");

        /*检测用户名和密码是否正确*/

        flag = login(nam, pass);

        if (flag == 1) {
            printf("\n 在文件夹中找到\n ");
            fflush(stdout);
            notice = "1";
            write(userfd[i], notice, strlen(notice));
        } else {
            printf("\n 用户名或者密码无法在文件中找到\n ");
            fflush(stdout);
            notice = "0";
            write(userfd[i], notice, strlen(notice));
        }
        //第一次进入聊天室&& (clientName[i][0] =="\0")
        if ((line[0] == '/')) {
            strcpy(clientName[i], nam);
            enterRoom(line, clientName[i]);
        }
        //播放 XX 进入聊天室，进行广播
        if(flag==1){
            printf("%s\n", line);
            fflush(stdout);
        }
        for (j = 0; j < MAX_LINK && (flag == 1); ++j) {
            if ((j != i) && (user_link[j] == 1)) {
                write(userfd[j], line, strlen(line));
            }
        }
    }/*私聊*/
```

```
        else if (line[0] == '#') {
            /*若以 2 个##开始的字串，表示开始进行私聊*/
            if (line[1] == '#') {
                tokenPtr = strtok(temp, "/");
                tokenPtr = strtok(NULL, "/");
                strcpy(temp, tokenPtr);
                printf("%s\n", temp);
                fflush(stdout);
                strfd[0] = line[2];
                strfd[1] = '\0';
                write(atoi(strfd), temp, strlen(temp));
            }
            /*只有一个#开始的字符串，表示第一次连接私聊
              判断当前活动用户数是否为 0
             */
            else {
                bzero(temp, MAX_BUF);
                if (userCount < 1) {
                    write(userfd[i], "0", strlen("0"));
                } else {
                    for (j = 0; j < MAX_LINK; ++j) {
                        if ((j != i) && (user_link[j] == 1)) {
                            sprintf(strfd, "%d", userfd[j]);
                            strcat(temp, strfd);
                            strcat(temp, "-->");
                            strcat(temp, clientName[j]);
                            strcat(temp, "\n");
                        }
                    }
                    write(userfd[i], temp, strlen(temp));
                }
            }
        }
        /*版面聊天*/
        else {
            printf("%s\n",line);
            fflush(stdout);
            for (j = 0; j < MAX_LINK; ++j) {
                if ((j != i) && (user_link[j] == 1)) {
                    write(userfd[j], line, strlen(line));
                }
            }
        }
    }
```

```
        }
    }//結束数据传送的监听
```

六. 运行演示

见软件使用手册。

七. 实验总结

实验在第十五周就做完了，报告一直拖着没写，所以也没急着交。实验演示的截图比较困难，因为是多个客户端，所以聊天聊着自己也跟着混乱了，想想毕竟也不会有人开多个客户端，自己跟自己聊天的吧。

刚开始做这次实验的时候，简直没有一点头绪，后来在图书馆借了一本书，慢慢研究 socket 网络编程。开始实现了群聊，然后这个基础上实现私聊，注册登录。最后还加上了时间戳和聊天记录。全部做完，大概用了一个星期时间，当然加上前期 linux 的摸索，时间就不仅仅是一个星期的了。

以前学习 linux,抱着是一种玩玩看的心态,想接触一下除了 windows 以外的 os，随着后来的深入学习，linux 果然不仅仅是好玩……因为是以 C 为基础，所以我又重抄旧业，把 C 复习了一遍，发现学 C 比 Java 等高级语言要有意思的多。比如字符串处理，JAVA 只要调用库函数即可解决这个问题,C 虽然也能解决某些字符串问题,比如 strcat，strcmp 等但都是比较底层的，能更促进我们的思考，让我们自己学会解决问题。

不是它不伟大，而是我们不了解它。当然学习 linux，我们只是皮毛，还未涉及到真正的核心部分 ，兴趣么，总是最好的老师~