

《计算机网络协议开发》实验报告

第三次实验 即时通信软件的编程

姓名：元玉慧

学号：101220151

10 级 计算机 系 4 班

邮箱：njucsyyh@gmail.com

时间：2013/03/30

一、实验目的

本次实验是让大家熟悉网络应用层的设计，实现，掌握服务器套接字编程。

二、实验设计要点

【1】及时通讯 IM 的设计分析：

主要实现用户登录，维护好友列表，单对单通信聊天，单对多的聊天。

【2】并发/线程：

同一进程内的所有线程共享：进程指令，大多数数据，打开的文件即文件描述符，信号处理函数和信号处置，当前工作目录；

不同线程有各自的线程 ID, 寄存器集合，栈，优先级等。

【2.1】基本线程函数：

一个程序由 `exec` 启动执行时，称为初始线程；

`pthread_create` 函数来创建线程

<code>#include <pthread.h></code>
<code>int pthread_create(pthread_t *tid, const pthread_attr_t *attr, void *(*func) (void *), void *arg);</code>
Returns: 0 if OK, positive Exxx value on error

`pthread_join` 函数等待一个给定的线程终止。

<code>#include <pthread.h></code>
<code>int pthread_join (pthread_t tid, void ** status);</code>
Returns: 0 if OK, positive Exxx value on error

`pthread_self` 函数获取自身的 ID

<code>#include <pthread.h></code>
<code>pthread_t pthread_self (void);</code>
Returns: thread ID of calling thread

`pthread_detach` 把指定的线程设定为脱离状态：

<code>#include <pthread.h></code>
<code>int pthread_detach (pthread_t tid);</code>
Returns: 0 if OK, positive Exxx value on error

`Pthread_exit` 让一个线程终止

【3】设计模式功能

服务器的操作：

- 1、监控 IM 客户加入和离开 IM 网络；
- 2、为所有在线 IM 客户维护一个一致性视图，并将这些状态信息提供给所有在线客户；
- 3、促成 IM 客户之间的信息交换；

客户端的操作：

- 1、显示当前在线 IM 客户；
- 2、允许用户发送消息给这些 IM 客户，并接受他们发送过来的消息；

【4】需要提供一个对所设计协议的简要说明文档，协议文档内容包括：

【4.1】对系统中主要参与者的概述，说明每个参与者在系统中的作用，以及他们是如何结合起来产生所需的系统行为。

【4.2】定义参与者之间的协议，并给出协议运行的图例。

【4.3】提供了一些重要设计决策背后的原因。

【5】简化设计：

- 1- 用户接口可以为命令行接口。
- 2- 不需要为每一个 IM 客户维护一个好友列表，可以假设 IM 网络中所有 IM 客户均属于一个全局的好友列表，当有 IM 客户加入或离开 IM 网络时，需要通知所有其他的 IM 客户，所有的 IM 客户互相之间可以交换信息；
- 3- 只是用一个 IM 服务器。

二、实验内容

-1-客户端设计：

收消息封装为线程，而发消息是客户端的主线程，具体关键代码如下：

```
void *Recv(void *arg)
{
    int sockfd = (int)arg;
    char recvline[MAXLINE];
    memset(recvline, 0, MAXLINE);
    while(recv(sockfd, recvline, MAXLINE, 0) > 0){
        printf("%s", recvline);
        fflush(stdout);
        memset(recvline, 0, MAXLINE);
    }
}
```

```

- pthread_create(&tid, NULL, Recv, (void *)sockfd);
- while(fgets(sendline, MAXLINE, stdin) != NULL){
-     send(sockfd, sendline, strlen(sendline), 0);
-     memset(sendline, 0, MAXLINE);
- }

```

-2-服务器的设计部分

【1】定义了维护客户端连接状态的结构体：

```

static struct Online_list{
    int port_id;
    char username[10];
    char passwd[10];
    bool tag;
    bool logon;
}List[MAXONLINE];

```

【2】关于结构体的操作函数设计有

```

Void init_list ()
Void add_list ()
Void del_list ()
Void logon (int , char * , char *)
Void logoff(int id)
Bool check_log( char *)
Int get_port(char *)
Char * get_username(int )

```

【3】服务器端主线程函数是：

```
Void * doit( void * );
```

函数说明：

根据接收到的 buf 字符串中的第一个分割字段判断报文类型；设置服务编号
设置 serv_type

```

SER_TYPE = strtok(buf, ":");
//change the serv_type to int as switch can only rec
if(strncmp(SER_TYPE, "LOGON", 5)==0) serv_type = 1;
if(strncmp(SER_TYPE, "LOGOFF", 6)==0) serv_type = 2;
if(strncmp(SER_TYPE, "TO", 2)==0) serv_type = 3;
if(strncmp(SER_TYPE, "ALL", 3)==0) serv_type = 4;
if(strncmp(SER_TYPE, "SHOW", 4)==0) serv_type = 5;

```

然后是在 switch 语句中处理所有不同的情况

【3.1】LOGON 部分实现比较简单，主要通过调用 2 中所述的函数来实现的，前面代码部分有鲁棒处理，处理错误的输入，后面的关键代码如下

```
182         //logon OK! And notice the log user
183         logon(connfd,cname,password);
184         int i;
185         char *temp = "Welcome log on the server successful !\n";
186         memset(buf, 0, MAXLINE);
187         strcpy(buf,temp);
188         send(connfd,buf,100,0);
189         memset(buf, 0, MAXLINE);
190         //notify all the online user
191         strcpy(buf,cname);
192         char *message = " has log on the server\n";
193         strcat(buf,message);
194         for(i=0; i<MAXONLINE; i++){
195             if(List[i].logon && List[i].port_id != connfd)    send
(List[i].port_id,buf,100,0);
196         }
197         memset(buf, 0, MAXLINE);
```

【3.2】LOGOFF 部分实现关键代码：

```
200         case 2: { //LOGOFF
201             logoff(connfd);
202             int i;
203             //notice the logoff user
204             char cname[10];
205             char *leaveinfo = "Leave the server successful !\n";
206             memset(buf, 0, MAXLINE);
207             strcpy(buf,leaveinfo);
208             send(connfd,buf,100,0);
209             memset(buf, 0, MAXLINE);
210             strcpy(cname,get_username(connfd));
211             //notice all the online user
212             strcpy(buf,cname);
213             char *message = " has left the server\n";
214             strcat(buf,message);
215             for(i=0; i<MAXONLINE; i++){
216                 if(List[i].logon && List[i].port_id != connfd)    send
(List[i].port_id,buf,100,0);
217             }
218             memset(buf, 0, MAXLINE);
219             break;
220         }
```

【3.3】TO 功能部分实现关键代码：

```

221         case 3:{//T0
222             char dest_user[10];
223             char message[100];
224             //check the input format is OK!
225             char *t1 = strtok(NULL, ":");
226             if(t1==NULL){
227                 memset(buf, 0, MAXLINE);
228                 strcpy(buf,"The correct format should be:
T0:username:message !\n");
229                 send(connfd,buf,100,0);
230                 memset(buf, 0, MAXLINE);
231                 break;
232             }
233             char *t2 = strtok(NULL, ":");
234             if(t2==NULL){
235                 memset(buf, 0, MAXLINE);
236                 strcpy(buf,"The correct format should be:
T0:username:message !\n");
237                 send(connfd,buf,100,0);
238                 memset(buf, 0, MAXLINE);
239                 break;
240             }

```

```

241             //
242             strcpy(dest_user,t1 );
243             strcpy(message,t2 );
244             memset(buf, 0, MAXLINE);
245             char cname[10];
246             strcpy(cname,get_username(connfd) );
247             char *split = ": ";
248             strcpy(buf,cname);
249             strcat(buf,split);
250             strcat(buf,message);
251             int port = get_port(dest_user);
252             if(port==100){
253                 memset(buf, 0, MAXLINE);
254                 strcpy(buf,"The user input is not online!\n");
255                 send(connfd,buf,100,0);
256                 memset(buf, 0, MAXLINE);
257                 break;
258             }
259             send(port,buf,100,0);
260             memset(buf, 0, MAXLINE);
261             break;
262         }

```

【3.4】ALL 功能部分实现关键代码：

```

263         case 4:{//ALL
264             char message[100];
265             strcpy(message,strtok(NULL, ":") );
266             if(message==NULL){
267                 memset(buf, 0, MAXLINE);
268                 strcpy(buf,"The message can not be NULL !\n");
269                 send(connfd,buf,100,0);
270                 memset(buf, 0, MAXLINE);
271                 break;
272             }
273             int i;
274             char *cname = get_username(connfd);
275             char *split = ": ";
276             char *temp = "[ALL]";
277             memset(buf, 0, MAXLINE);

```

```

278         strcpy(buf,temp);
279         strcat(buf,cname);|
280         strcat(buf,split);
281         strcat(buf,message);
282         //send the message to all the online user
283         for(i=0; i<MAXONLINE; i++){
284             if(List[i].logon && List[i].port_id != connfd)    send
(List[i].port_id,buf,100,0);
285         }
286         memset(buf, 0, MAXLINE);
287         break;
288     }

```

【3.5】SHOW 功能部分实现关键代码：

```

289         case 5:{//SHOW
290             int i;
291             char *split = " # ";
292             memset(buf, 0, MAXLINE);
293             char *temp = "Online list is as list :\n";
294             strcpy(buf,temp);
295             for(i=0; i<MAXONLINE; i++)
296             {
297                 if(List[i].logon)
298                 {
299                     strcat(buf,List[i].username);
300                     strcat(buf,split);
301                 }
302             }
303             strcat(buf,"\n");
304             send(connfd,buf,100,0);
305             memset(buf, 0, MAXLINE);
306             break;
307         }

```

互斥信号量的设置

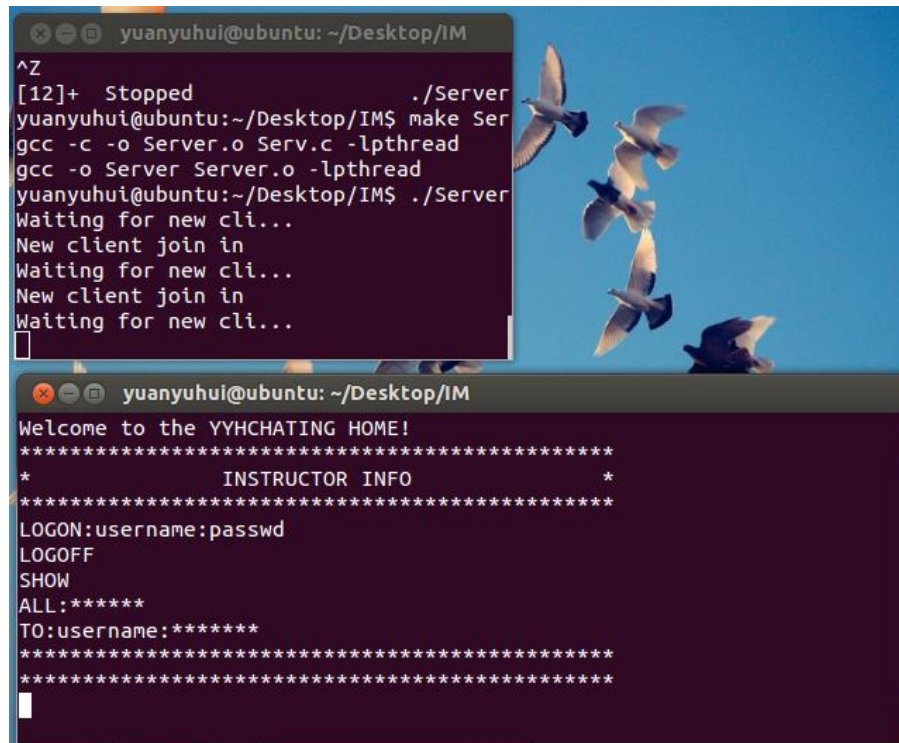
```

36 void
37 add_list(int id)
38 {
39     int i=0;
40     if(Online_num > MAXONLINE){
41         printf("The chatting room is full, please connect later!\n");
42         return;
43     }
44     while(List[i].tag == true) i++;
45     List[i].tag = true;
46     List[i].port_id = id;
47     //lock and unlock the mutex|
48     pthread_mutex_lock(&mutexsum);
49     Online_num++;
50     pthread_mutex_unlock(&mutexsum);
51 }

```

三、实验结果

初始时，有客户端连上服务器

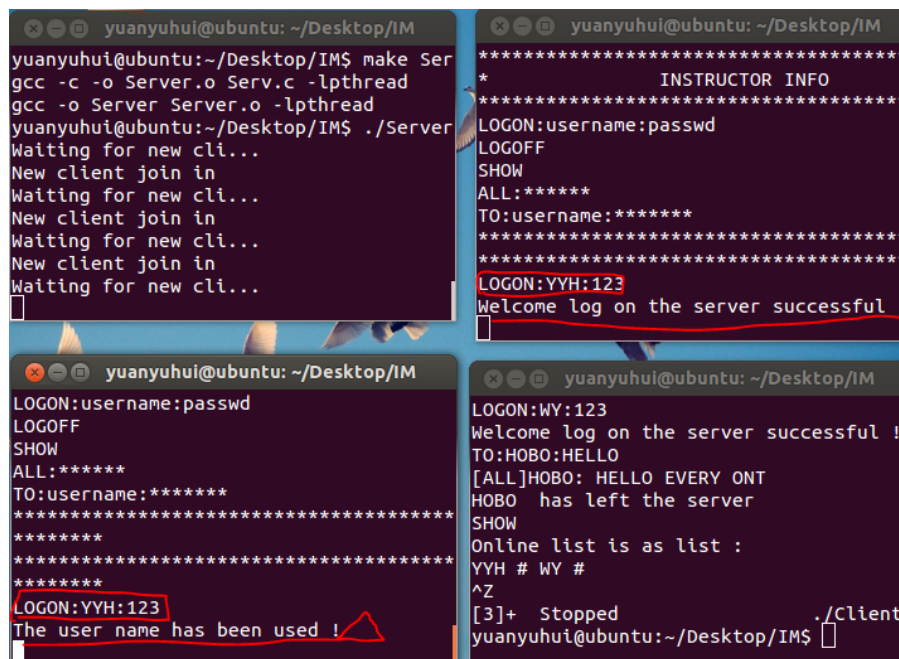


The image shows two terminal windows. The top window shows the server being started with the command `./Server`. It displays messages like "Waiting for new cli...", "New client join in", and "Waiting for new cli...". The bottom window shows the server's main menu with options: "LOGON:username:passwd", "LOGOFF", "SHOW", and "ALL:*****". It also shows a client logging in with the command `TO:username:*****`.

```
yuanyuhui@ubuntu: ~/Desktop/IM
^Z
[12]+  Stopped                  ./Server
yuanyuhui@ubuntu:~/Desktop/IM$ make Ser
gcc -c -o Server.o Serv.c -lpthread
gcc -o Server Server.o -lpthread
yuanyuhui@ubuntu:~/Desktop/IM$ ./Server
Waiting for new cli...
New client join in
Waiting for new cli...
New client join in
Waiting for new cli...

yuanyuhui@ubuntu: ~/Desktop/IM
Welcome to the YYHCHATING HOME!
*****
*                INSTRUCTOR INFO                *
*****
LOGON:username:passwd
LOGOFF
SHOW
ALL:*****
TO:username:*****
*****
```

若新的客户端用户名重复，则会收到通知



The image shows four terminal windows. The top-left window shows the server startup. The top-right window shows the server's main menu. The bottom-left window shows a client logging in with the command `LOGON:YYH:123`, which is highlighted with a red box. The bottom-right window shows the server's response to the login attempt, including the message "The user name has been used !".

```
yuanyuhui@ubuntu: ~/Desktop/IM
yuanyuhui@ubuntu:~/Desktop/IM$ make Ser
gcc -c -o Server.o Serv.c -lpthread
gcc -o Server Server.o -lpthread
yuanyuhui@ubuntu:~/Desktop/IM$ ./Server
Waiting for new cli...
New client join in
Waiting for new cli...
New client join in
Waiting for new cli...
New client join in
Waiting for new cli...

yuanyuhui@ubuntu: ~/Desktop/IM
*****
*                INSTRUCTOR INFO                *
*****
LOGON:username:passwd
LOGOFF
SHOW
ALL:*****
TO:username:*****
*****
LOGON:YYH:123
Welcome log on the server successful !

yuanyuhui@ubuntu: ~/Desktop/IM
LOGON:username:passwd
LOGOFF
SHOW
ALL:*****
TO:username:*****
*****
LOGON:YYH:123
The user name has been used !

yuanyuhui@ubuntu: ~/Desktop/IM
LOGON:WY:123
Welcome log on the server successful !
TO:HOB0:HELLO
[ALL]HOB0: HELLO EVERY ONT
HOB0 has left the server
SHOW
Online list is as list :
YYH # WY #
^Z
[3]+  Stopped                  ./client
yuanyuhui@ubuntu:~/Desktop/IM$
```


当有新的客户端登录时，在线用户会受到通知

The image displays four terminal windows showing the execution of a C program for a simple chat server. The top-left terminal shows the compilation of 'Server.o' and the server running, waiting for clients. The top-right terminal shows the server's output for a client named 'YY' logging in successfully. The bottom-left terminal shows the server's output for a client named 'YY' logging in successfully, with a red circle around the success message. The bottom-right terminal shows the server's output for a client named 'WY' logging in successfully, and then 'Hobo' logging in and leaving the server.

```

yuanyuhui@ubuntu: ~/Desktop/IM
yuanyuhui@ubuntu:~/Desktop/IM$ make Server
gcc -c -o Server.o Serv.c -lpthread
gcc -o Server Server.o -lpthread
yuanyuhui@ubuntu:~/Desktop/IM$ ./Server
Waiting for new cli...
New client join in
Waiting for new cli...
New client join in
Waiting for new cli...
New client join in
Waiting for new cli...

yuanyuhui@ubuntu: ~/Desktop/IM
*                               INSTRUCTOR INFO
*****
*****
LOGON:username:passwd
LOGOFF
SHOW
ALL:*****
TO:username:*****
*****
LOGON:YYH:123
Welcome log on the server successful !
YY has log on the server

yuanyuhui@ubuntu: ~/Desktop/IM
SHOW
ALL:*****
TO:username:*****
*****
*****
*****
LOGON:YYH:123
The user name has been used !
LOGON:YY:123
Welcome log on the server successful !

yuanyuhui@ubuntu: ~/Desktop/IM
LOGON:WY:123
Welcome log on the server successful !
TO:Hobo:HELLO
[ALL]Hobo: HELLO EVERY ONT
Hobo has left the server
SHOW
Online list is as list :
YYH # WY #
^Z
[3]+  Stopped                  ./Client
yuanyuhui@ubuntu:~/Desktop/IM$

```

不同的客户端之间使用 TO 命令来互发消息

The image displays four terminal windows from a user named 'yuanyuhui' on an 'ubuntu' machine, located in the directory '~/Desktop/IM'. The windows show the execution of a C++ server and two client programs.

Terminal 1 (Top Left): Shows the server program 'Server.o' running. It prints 'Waiting for new cli...' and then 'New client join in' four times, indicating successful connections from four clients. It then enters a loop 'Waiting for new cli...' and waits for input.

Terminal 2 (Top Right): Shows a client program running. It prompts 'TO:username:*****' and receives input. It then prints 'LOGON:YYH:123' and 'Welcome log on the server successful !'. It then prompts 'YY has log on the server' and 'ZZ has log on the server'. It then prompts 'ZZ: 123' and 'TO:YY:HELLO'.

Terminal 3 (Bottom Left): Shows a client program running. It prompts 'TO:username:*****' and receives input. It then prints 'LOGON:YYH:123' and 'The user name has been used !'. It then prompts 'LOGON:YY:123' and 'Welcome log on the server successful !'. It then prompts 'ZZ has log on the server' and 'YYH: HELLO'.

Terminal 4 (Bottom Right): Shows a client program running. It prompts 'SHOW' and 'ALL:*****'. It then prompts 'TO:username:*****' and receives input. It then prints 'LOGON:ZZ:123' and 'Welcome log on the server successful !'. It then prompts 'TO:YYH:123'.

Red circles and arrows are drawn on the screenshots to highlight specific parts of the output. In the top-left terminal, a red circle is drawn around the first 'New client join in' message. In the top-right terminal, a red circle is drawn around the 'TO:YY:HELLO' message. In the bottom-left terminal, a red circle is drawn around the 'YYH: HELLO' message. In the bottom-right terminal, a red circle is drawn around the 'TO:YYH:123' message. Red arrows point from the first 'New client join in' message to the 'TO:YY:HELLO' message, and from the 'YYH: HELLO' message to the 'TO:YYH:123' message.

当有客户端请求下线时，其他在线好友会收到他的下线通知


```
yuanyuhui@ubuntu: ~/Desktop/IM
ver
gcc -c -o Server.o Serv.c -lpthread
gcc -o Server Server.o -lpthread
yuanyuhui@ubuntu:~/Desktop/IM$ ./Server
Waiting for new cli...
New client join in
Waiting for new cli...
New client join in
Waiting for new cli...
New client join in
Waiting for new cli...

yuanyuhui@ubuntu: ~/Desktop/IM
LOGOFF
SHOW
ALL:*****
TO:username:*****
*****
*****
LOGON:YYH:123
Welcome log on the server successful !
ZZ has log on the server
JJ has log on the server

yuanyuhui@ubuntu: ~/Desktop/IM
LOGOFF
SHOW
ALL:*****
TO:username:*****
*****
*****
LOGON:ZZ:JJ
Welcome log on the server successful !
JJ has log on the server

yuanyuhui@ubuntu: ~/Desktop/IM
TO:username:*****
*****
*****
LOGON:JJ:123
Welcome log on the server successful !
SHOW
Online list is as list :
YYH # ZZ # JJ #
```

四、实验中遇到的问题及解决方案

【1】这次实验开始的时候遇到了和第二次实验同样的问题，就是会报 Segmentation default (core dumped); 发现主要是由于存在未初始化的指针变量或者是其他潜在的访问越界操作；

【2】实验开始的时候一直纠结与多线程编程，后来参考了很多资料后，初步确定了多线程编程的实现方法，和 java 课程中的线程实现方式很类似；

【3】遇到过发送的消息没有及时显示在客户端，而是等待很久或者是客户端发送一条明后后，才会收到回复，后来发现时 recv 函数调用了 2 次的缘故；

【4】实验中发现很多错误都是跟结构体内部的 char 指针结构有关的，发现 char 指针的使用时很容易出错的，必须初始化申请一定大小的空间，后来采用了 char 数组来实现，纠正了很多错误。还有 strcmp 等基于 char* 的函数对于 char 数组是同样适用的，由于不对 char 指针分配空间的话，他可能指向栈中的内容，而栈中的内容很容易由于程序的执行被冲掉，所有会发生各种莫名其妙的错误额！

使用 char* 时，一定要慎重仔细

【5】switch 的 case 语句的分支只支持整型变量；

【6】strtok(buf, ":");

strtok(NULL, ":");

上述 2 个函数时处理用 “:” 分割的字符串，将其分割成多个独立的字符串。

【7】在后期调试的时候发现每次出现问题，要重启服务器，但是重启服务器后，客户端都无法连上服务器了！所以当时很蛋疼，没隔 10 分钟才能调试一次 !!!后来采用了每次都重启的方法，再后来跟同学讨论后，发现是 linux 后台关于服务器开设的端口号的程序可能仍

在运行,所有后来采用了每次都更改头文件中的服务器的端口号的方法来避免每次重启虚拟机!!!

【8】

```
pthread_create(&tid, NULL, Recv, (void *)sockfd);  
while(fgets(sendline, MAXLINE, stdin) != NULL){  
    send(sockfd, sendline, strlen(sendline), 0);  
    memset(sendline, 0, MAXLINE);  
}
```

Handwritten notes: 子 (child) and 主 (main) with arrows pointing to the thread creation and the while loop respectively.

开始时发现线程放在了主线程的后面是永远都执行不到的,将接收服务器信息的线程的创建放在主线程之前

五、实验的启示/意见和建议

通过此次实验,主要熟悉了 C 语言的多线程编程方法,以及更多的 char* 的操作,发现了很多细节问题,当然最重要的是熟悉了应用层协议的设计的简单方法,以及较好的简化问题的方法。

此次实验大约花费了 24+小时吧!包括开始思考设计协议+写代码+调代码+写报告

感觉花的时间太多了!!! 所以希望陈老师能够不要这么频繁的布置实验,很多其他课程的学习时间都没有很好的保证了!

实验心得:这次实验的 95+%都是自己独立完成的,实验过程中,关于某些 BUG 调试时,跟同学讨论了一些结论,还有也学习了多一点的 makefile 文件的写法,以及可以使用多线程编程,并且最终实现了一个很简单的及时通讯功能,虽然实现的功能并不强大,但是还是很有成就感的