



浙江工业大学

嵌入式系统课程设计报告

题目：_____多人简易聊天室设计_____

作者 1 姓名 _____凌智城_____

作者 1 学号 _____201806061211_____

作者 1 专业班级 _____通信工程 1803 班_____

作者 2 姓名 _____夏炅_____

作者 2 学号 _____201806062219_____

作者 2 专业班级 _____通信工程 1803 班_____

提交日期 2021 年 9 月 24 日

一、设计目的

1. 设计开发一个基于命令行的聊天软件,能够实现单人、多人聊天等功能;
2. 编写基于 TCP/UDP 协议的多用户之间的聊天工具,包括服务端、客户端两部分。支持多人,正常进入、退出聊天,允许发送消息给所有人;
3. 功能包括文字聊天、传输文件、添加/删除用户等;
4. 允许发送消息给指定的一个或者多个人。

二、项目要求及分工

2.1 设计要求

选取项目基本要求作为课程设计目标:编写基于 TCP 协议的多用户聊天工具,包含客户端和服务端。首先完全在虚拟机上实现服务端和客户端的本地回环,实现后将服务端放到实验箱,设置 socket 的 IP 为实验箱的地址即可。实验箱作为服务端,交叉编译完成后通过挂载文件夹将虚拟机上生成的可执行文件拷贝到实验箱,实验箱启动服务监听聊天,虚拟机开启多个命令行终端运行客户端程序模拟多用户连接服务端,实现多用户聊天室的用户正常进入、退出、信息收发及时间显示、发送消息给所有人、发送消息给指定一个或多个人等功能。

2.2 项目分工

我组各阶段需求分析、模块设计、数据结构、函数接口等详细设计均共同完成。

三、设计原理

3.1 TCP Socket 原理

在一般的网络编程中,一般不会直接去操作 TCP/IP 协议的底层,而是用过 socket 套接字进行编程,相当于一个 TCP/IP 与应用程序的中间层,通过套接字中提供的接口函数进行网络开发。

struct sockaddr 是一个通用地址结构,是为了统一地址结构的表示方法,统一接口函数,使不同的地址结构可以被 bind(), connect() 等函数调用,但是它把目标地址和端口信息混在一起了; struct sockaddr_in 中的 in 表示 internet,是比较常用的地址结构,属于 AF_INET 地址族,解决了 sockaddr 的缺陷,把 port 和 addr 分开储存在两个变量中; sockaddr_in 在 <netinet/in.h>中。

`socket(AF_INET, SOCK_STREAM, 0)`; 来自<sys/socket.h>中的 `int socket(int af, int type, int protocol)`; 套接字创建函数, `af` 为地址族 (Address Family), 也就是 IP 地址类型, 常用的有 `AF_INET` 和 `AF_INET6`, 即常用的 IPv4 和 IPv6; `type` 为数据传输方式/套接字类型, 常用的有 `SOCK_STREAM` (流格式套接字/面向连接的套接字) 和 `SOCK_DGRAM` (数据报套接字/无连接的套接字); `protocol` 表示传输协议, 常用的有 `IPPROTO_TCP` 和 `IPPROTO_UDP`, 分别表示 TCP 传输协议和 UDP 传输协议, 但 TCP 和 UDP 两种情况都只有一种协议满足条件, 可以将 `protocol` 的值设为 0, 系统会自动推演出应该使用什么协议。

`setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))`; 来自<sys/types.h> 和<sys/socket.h>中的 `int setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen)`; `setsockopt()` 用来设置参数 `s` 所指定的 socket 状态. 参数 `level` 代表欲设置的网络层, 一般设成 `SOL_SOCKET` 以存取 socket 层. 此处的 `opt` 是 `SO_REUSEADDR` 表示允许在 `bind()` 过程中本地地址可重复使用。

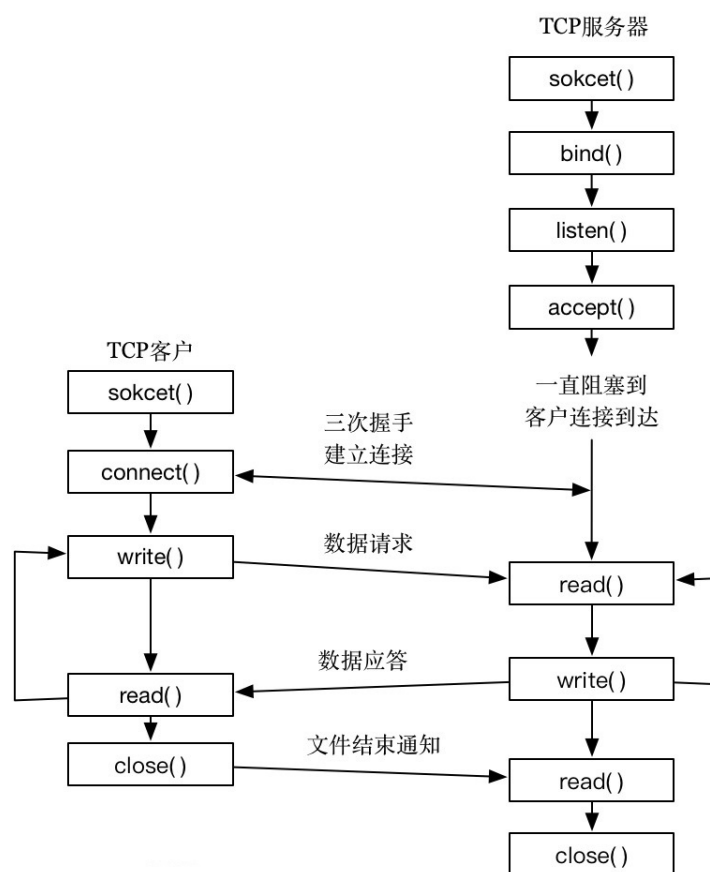


图 1.TCP 连接过程

常用 socket 函数:

创建: `int socket(int family, int type, int protocol)`; 在执行网络 I/O 之前必须调用的函数, 用于指定期望的通信协议类型。

连接: `int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen)`; TCP 客户端用 `connect` 函数来建立与 TCP 服务器的连接。

绑定: `int bind(int sockfd, const struct sockaddr *myaddr, socklen_t addrlen)`; 用于将一个本地协议地址赋予一个套接字。

监听: `int listen(int sockfd, int backlog);` 仅由 TCP 服务器调用, `listen` 函数将一个未连接的套接字转换成一个被动套接字, 指示内核应接受指向该套接字的连接请求。

接收已建立的连接: `int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);` 由 TCP 服务器调用, 用于从已完成连接队列头返回下一个已完成连接。

关闭: `int close(int sockfd);` 用于关闭套接字, 并终止 TCP 连接。

3.2 交叉编译原理

本实验需要将实验箱和 PC 相连通, 运行在不同的平台, 显然需要交叉编译。本地编译可以理解为, 在当前编译平台下, 编译出来的程序只能放到当前平台下运行。平时我们常见的软件开发, 都是属于本地编译: 比如, 我们在 x86 平台上, 编写程序并编译成可执行程序。这种方式下, 我们使用 x86 平台上的工具, 开发针对 x86 平台本身的可执行程序, 这个编译过程称为本地编译。交叉编译可以理解为, 在当前编译平台下, 编译出来的程序能运行在体系结构不同的另一种目标平台上, 但是编译平台本身却不能运行该程序: 比如, 我们在 x86 平台上, 编写程序并编译成能运行在 ARM 平台的程序, 编译得到的程序在 x86 平台上是不能运行的, 必须放到 ARM 平台上才能运行。而此次我们所用的实验箱是 ARM 平台, 需要用 `arm_v5t_le-gcc` 指令进行交叉编译后产生可执行文件才能运行。

四、系统设计架构

4.1 设计开发环境

本地调试环境: Win10 21H1、VMware 16.0 Pro、Ubuntu-20.04.3-desktop-amd64

实验室测试环境: Win10 1903、VMware 15.0 Pro、Ubuntu-14.04.5-desktop-amd64、4G 移动视频创新开发平台 (型号: PN-IP4GC03)、普诺科技 37-01-PN4GE-02

4.2 软件系统架构

1. 设计思路

先完全在虚拟机上实现客户端和服务端, 完成基本的功能后, 将服务端移植到实验箱的板子上, 进一步解决问题, 然后逐渐完善功能, 再虚拟机上调试, 基本无问题后最后再次放到实验箱上测试。

在 server 端, server 首先启动, 调用 `socket()` 创建 TCP 套接字; 然后调用 `bind()` 绑定 server 的地址(REMODE_IP+PORT); 再调用 `listen()` 让 server 做好侦听准备, 并规定好请求队列长度, 然后 server 进入阻塞状态, 等待 client 的连接请求; 最后通过 `accept()` 来接收连接请求, 并获得 client 的地址。当 `accept` 接收到一个 client 发来的 `connect` 请求时, 将生成一个新的 `socket`, 用于传输数据。

在 client 端, client 在创建套接字并指定 client 的 `socket` 地址, 然后就调用 `connect()` 和 server 建立连接。一旦连接建立成功, client 和 server 之间就可以通过调用 `recv()` 和 `send()` 来接收和发送数据。一旦数据传输结束, server 和 client 通过调用 `close()` 来关闭套接字。

首先需要服务端开启，经过一系列的 socket 创建套接字，绑定地址等，开始等待客户端启动。客户端启动后通过 connect 函数与服务端建立连接，传输的数据结构体 message 中包含发送者姓名，接收者姓名，消息，时间戳和一个标志位，通过 message 结构体中的两个 name 字段来匹配发送和接受的信息，分为全体成员都能接收和选定部分成员接收即实现私聊。将客户端主要分成五部分，全局变量和结构体的声明，数据接收和显示函数，群聊函数，私聊函数和主函数，在主函数中选择功能，每个功能可以实现输入 exit 退出，客户端使用 send 发送数据到服务端后，服务端根据数据内各个字段判定是否要发送给其他客户端，并且同时判断是否要有选择性的发送给某一个或某几个用户即私聊，并且服务端实时显示所有客户端进行的操作。宏定义连接的服务端地址，方便修改挂载到实验箱上时物理地址的改变；设置了客户端连接上限值，超过上限值无法连接。

2. 源代码

见附件

五、测试数据与分析

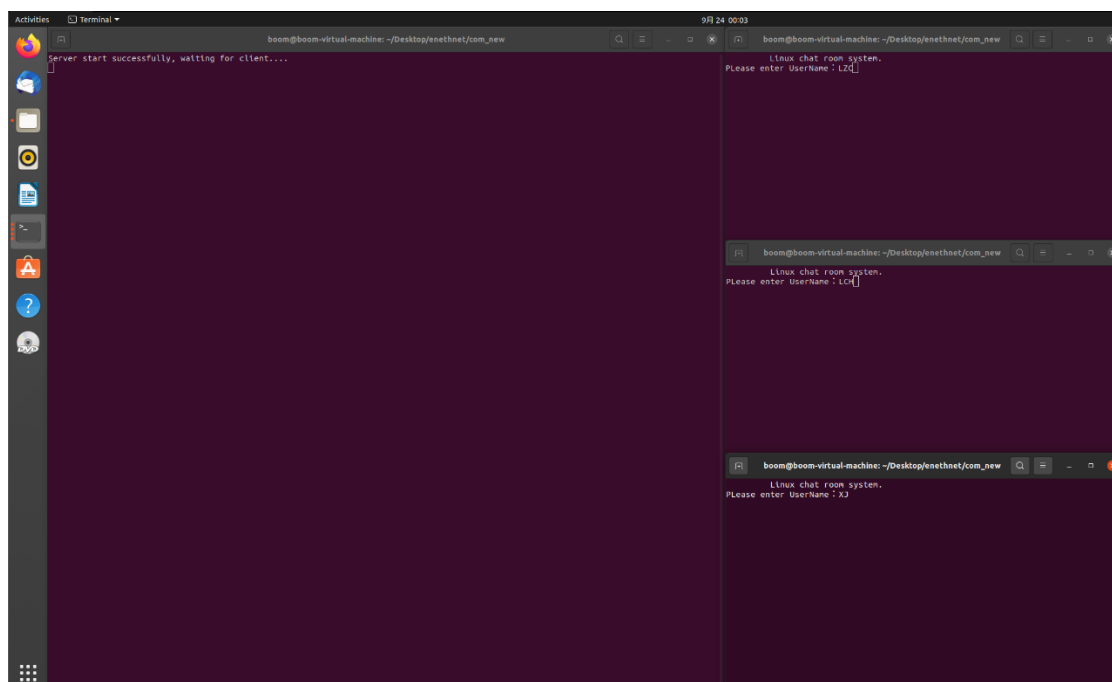


图 2.本地测试登录



图 6.虚拟机 IP 配置

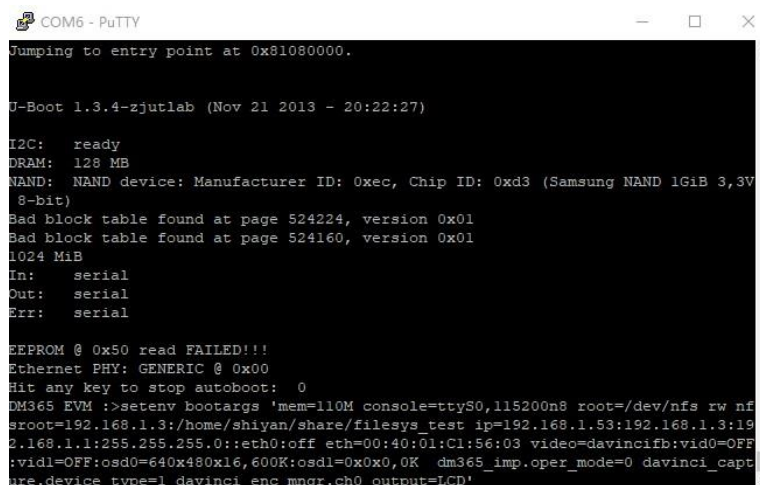


图 7.输入启动参数

VMware IP: 192.168.1.3

Hardware IP: 192.168.1.53

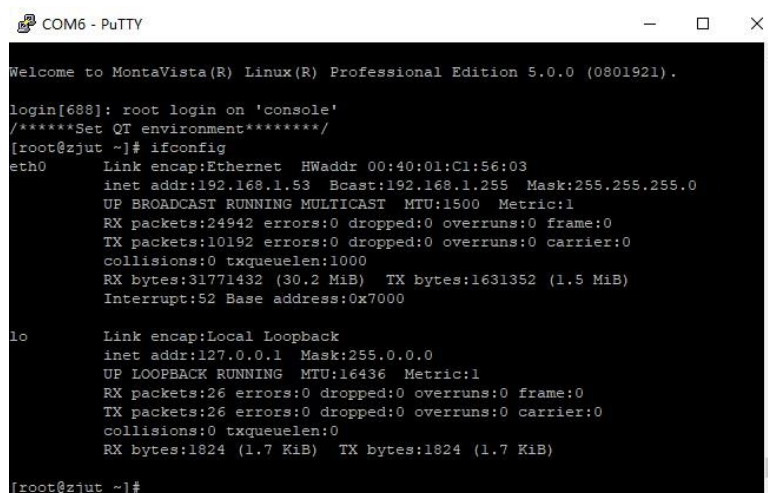


图 8.登录实验箱

```
< -o nolock 192.168.1.3:/home/shiyan/share/filesys_test /mnt/mtd/
[root@zjut ~]# df
Filesystem            1k-blocks      Used Available Use% Mounted on
tmpfs                  53868          92      53776   0% /tmp
tmpfs                  10240          32      10208   0% /dev
tmpfs                  53868          0      53868   0% /dev/shm
192.168.1.3:/home/shiyan/share/filesys_test 43218944 10397696 30602240 25% /mnt/mtd
[root@zjut ~]#
```

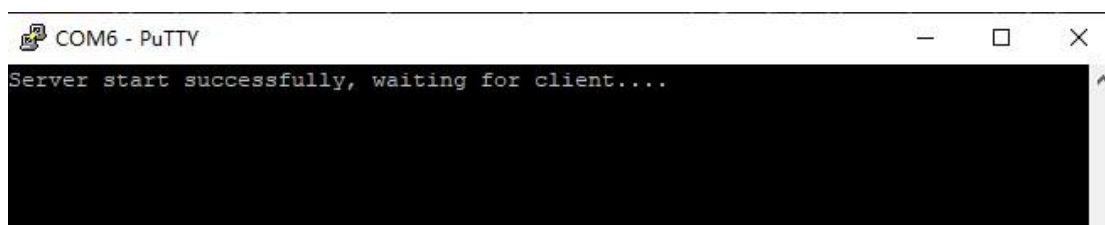
图 9.挂载虚拟机文件夹

```
mount -t nfs -o nolock 192.168.1.3:/home/shiyan/share/filesys_test
/mnt/mtd/
```

```
shiyan@ubuntu:~/share/filesys_test$ gcc client.c -lpthread -o client-gcc
client.c: In function 'multi_chat':
client.c:71:9: warning: passing argument 3 of 'pthread_create' from incompatible pointer type [enabled by default]
pthread_create(&tid,NULL,pthread_recv,NULL);count++;
^
In file included from client.c:18:0:
/usr/include/pthread.h:244:12: note: expected 'void * (*)(void *)' but argument is of type 'void (*)(void *)'
extern int pthread_create (pthread_t *__restrict __newthread,
^
client.c:87:9: warning: 'gets' is deprecated (declared at /usr/include/stdio.h:638) [-Wdeprecated-declarations]
gets(send_buf.mess);
^
client.c: In function 'private_chat':
client.c:126:9: warning: 'gets' is deprecated (declared at /usr/include/stdio.h:638) [-Wdeprecated-declarations]
gets(send_buf.mess);
^
/tmp/ccuGLhgX.o: In function 'multi_chat':
client.c:(.text+0x1d0): warning: the 'gets' function is dangerous and should not be used.
shiyan@ubuntu:~/share/filesys_test$ arm_v5t_le-gcc -lpthread server.c -o server-arm
shiyan@ubuntu:~/share/filesys_test$
```

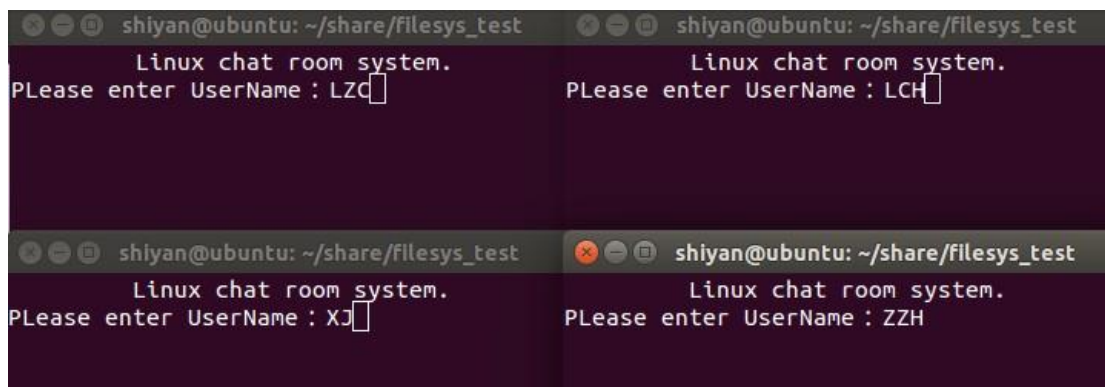
图 12.交叉编译

```
arm_v5t_le-gcc -lpthread server.c -o server_arm
gcc client.c -lpthread -o client-gcc
```



```
COM6 - PuTTY
Server start successfully, waiting for client....
```

图 13.服务端启动成功，监听客户端接入



```
shiyan@ubuntu: ~/share/filesys_test
Linux chat room system.
Please enter UserName : LZC

shiyan@ubuntu: ~/share/filesys_test
Linux chat room system.
Please enter UserName : LCH

shiyan@ubuntu: ~/share/filesys_test
Linux chat room system.
Please enter UserName : XJ

shiyan@ubuntu: ~/share/filesys_test
Linux chat room system.
Please enter UserName : ZZH
```

图 14.分别启动四个客户端接入服务

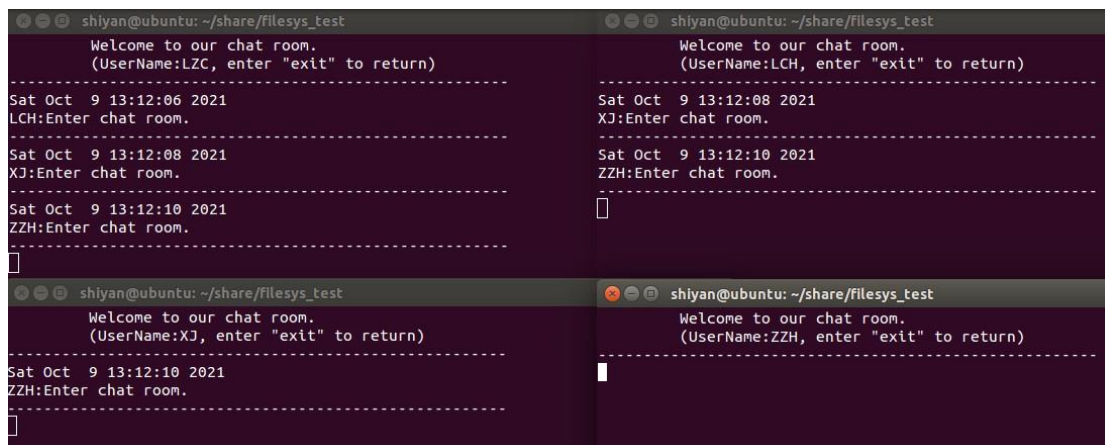


图 15.分别进入聊天室，互相显示进入信息

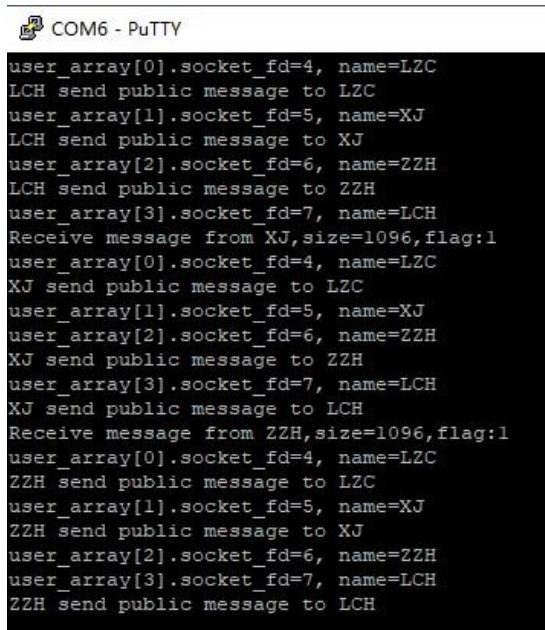


图 16.服务端监听群聊数据

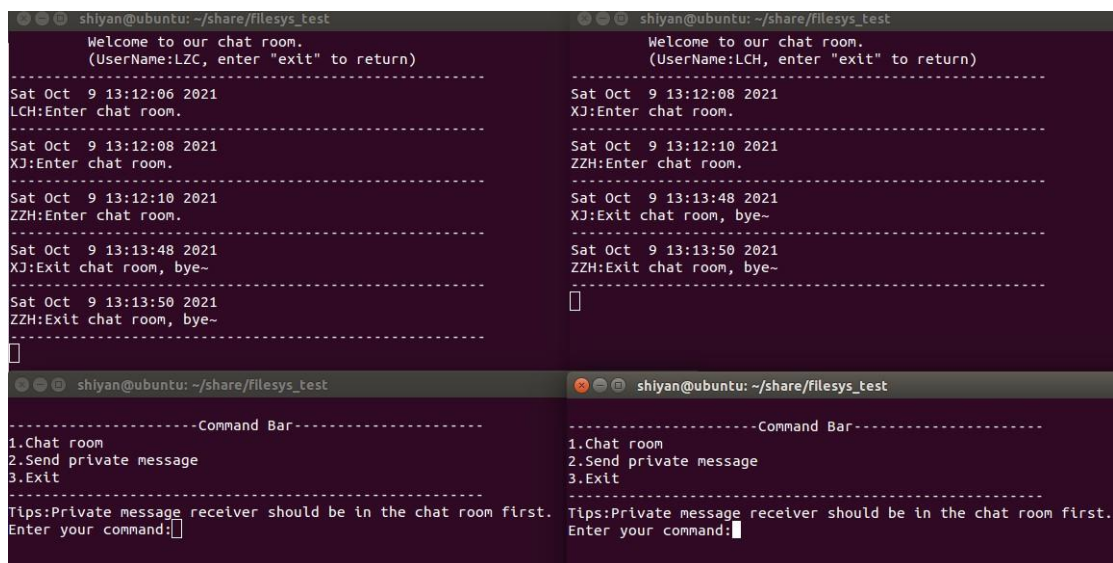


图 17.若其中有成员退出，其他成员接收到退出信息

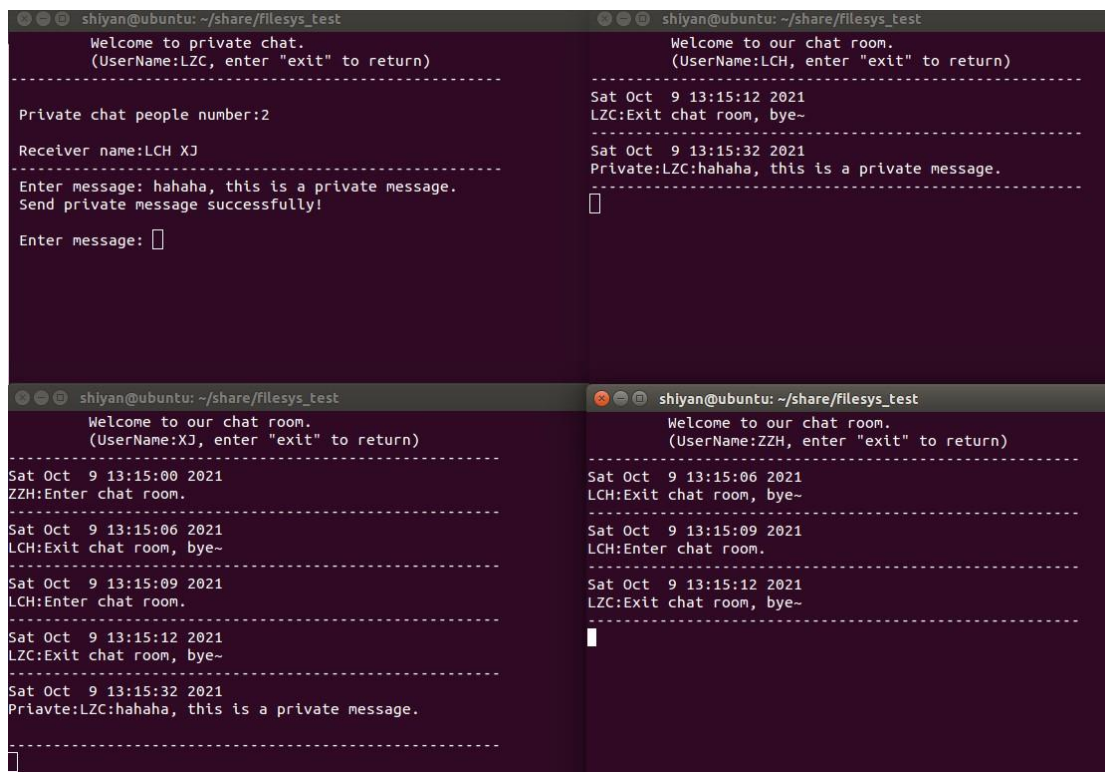


图 18.选定一个或多个用户聊天, 未被选中的不会收到消息

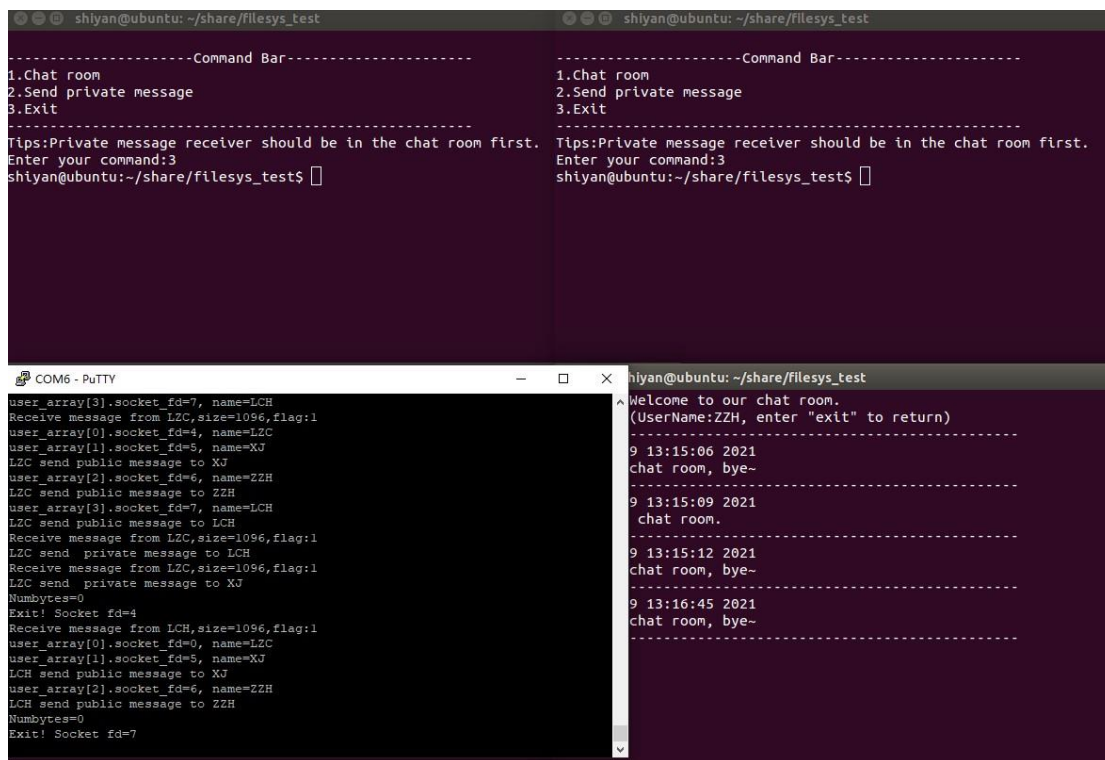


图 19.若客户端退出, 服务端显示已退出

六、设计总结

本次实验是我们第二次接触嵌入式实验，选择的是多人聊天室。最初我们实现的是在同一虚拟机下开启服务端终端和客户端终端，可以实现通信，但是交叉编译后挂载到板子上发现无法连接。查阅相关资料后发现是 `socket` 的 `sockaddr_in` 中的地址字段没有赋值，默认是 127.0.0.1 本地回环地址，同事都在本地时不会有影响，但放在两个终端进行操作时就需要将服务端所在的实验箱板子的 IP 地址赋给该字段才能正常连接。实验箱 IP: 192.168.1.53，虚拟机 IP: 192.168.1.3，修改后将服务端挂载到实验箱上，本地虚拟机连接实验箱，`putty` 打开串口调试连接实验箱打开聊天室服务端，然后在虚拟机上运行多个客户端终端。最开始代码写的比较乱，经过反复的修改和调试，有了比较清楚的人机交互逻辑以及各个界面的进入和退出，限制了连接服务端的上限为十个用户，超出将无法连接。发送的消息存储在结构体的 `mess` 字段中，最开始使用 `scanf()` 来读取键盘输入，后来发现这样将无法读取输入语句中包含的空格，会将空格识别成结束输入，导致一句分被分成好几次发送，后来改成了使用 `gets()` 来读取整段消息，但是提示高危操作，建议使用 `fgets()` 替代。或者此处也可以重新修改读取消息的逻辑，因为库函数内不含 `getline()`，可以另外写一个符合需求的自定义函数实现读取整段，这里为了方便直接用了 `gets()` 来实现，但同样也出现了因为 `scanf()` 和 `gets()` 机制的不同，会导致是否读取发送消息时的回车问题，需要实际调试后修改 `getchar()` 来吸收多余的回车。在实现发送消息给指定一个或多个人时候，多次出现数组越界和内存泄漏问题，最后采用了 `malloc` 动态定义二维字符数组，手动输入要送达消息的人数以及姓名，再用循环嵌套发送实现发送给多个人，为了防止内存泄漏需要及时 `free` 释放动态数组。

在最后一天验收的时候遇到了一个奇怪的问题，服务端可以正常开启，客户端时而连的上时而连不上，接线稳定无松动，也尝试更换了部分代码格式以及用 `--std=c99` 来指定版本也不行，尝试换了实验箱也不行，最后发现是其他组随意配置实验箱 IP 导致 IP 冲突。

总的来说，这次实验我们收获颇丰。特别是对于嵌入式系统交叉编译以及 `Linux` 相关指令有了更深刻的理解，这些都是经过实验得到的，将书本上抽象的知识得到了具体化。在查阅资料，和同学老师讨论的过程中我们也发现了许多自身存在的问题。今后在学习上我们会更注重实践以及不足之处的查漏补缺，继续努力。