Linux 操作系统与程序设计

课程设计报告书

报告人：

2022.12.28

注意不要雷同　　同学

https://github.com/dream4789/Computer-learning-resources.git

目录

1、 实验项目目的

《Linux》课程设计是在完成理论课程学习之后安排的综合实践训练，通过一周的综合课程设计，在学生掌握 Linux 的基本知识及常用的操作命令基础上，提高学生对所学知识的应用能力，特别是：煅炼学生搜集有关 Linux 资料，在资料文档的帮助下，运用 Linux 的基本知识及常用的操作命令来解决问题的能力。

2、实验项目的功能及模块划分

本实验是一个 Linux 下的网络即时通信工具其可以利用 linux 中的 socket 等实现一个可以让两个终端同时通讯的工具，而且还具有可以在服务器端查看所有终端的通讯情况的功能。

3、实验项目的人员组成及指责划分

张龙龙：组长，系统分析、整体设计和模块划分。

孙康：通信模块设计和编码。

张功旭：客户端界面设计和编码。

项前进：运行及测试。

4、设计与实现

本软件是一个 Linux 下基于 socket 的聊天室程序，能让局域网内的用户通过该软件进行简单的文字通信。在此基础上增加了 聊天室成员之间的发送私聊信息； 当新的成员加入后能自动收取最近一段时间内的聊天上下文； 用户能够查看历史聊天记录； 软件界面基于 Qt 实现，图形化界面方便用户操作。 主要模块划分： 服务端： 数据包发送和接受模块，聊天记录数据库读写模块，数据包处理模块，聊天记录查询模块 客户端： 数据包发送和接受模块，数据包处理模块，聊天记录查询模块，用户界面与展示模块

4．1 系统结构

本软件是一个 Linux 下基于 socket 的聊天室程序，能让局域网内的用户通过该软件进行简单的文字通信。在此基础上增加了 聊天室成员之间的发送私聊信息； 当新的成员加入后能自动收取最近一段时间内的聊天上下文； 用户能够查看历史聊天记录； 软件界面基于 Qt 实现，图形化界面方便用户操作。
主要模块划分：

服务端： 数据包发送和接受模块，聊天记录数据库读写模块，数据包处理模块，聊天记录查询模块
客户端： 数据包发送和接受模块，数据包处理模块，聊天记录查询模块，用户界面与展示模块

本系统采用 CS 架构，服务端采用固定的端口通信，每个客户端动态设置端口。客户端启动后向服务端告知自己所使用的端口号，以便可以双向通信，同时服务器负责为每个客户端分配一个唯一的 ID（服务器的 ID 为 1） 客户端和服务端以及客户端和客户端之间采用约定的数据格式进行通信，以便接收方可以正确的解析命令和数据。 数据包通用格式定义如下

```
#define MAX_UDP_SIZE 1000
struct udp_packet{
  int type;
int senderId;
  long size;
  char content[MAX_UDP_SIZE];
};
```

type： 表示该数据包的类型，直接决定 content 字段的含义
senderId：该数据包的发送者的 ID，
size:整个数据包的数据长度
content:数据包的内容，其数据格式由 type 决定。
服务器和客户端接受到数据包后，根据 type 字段的值来解析 content 字段的数据，从而作出正确的处理和响应。

4．2 客户端界面

4．2．1 注册

首先是服务端和客户端分别启动运行，客户端选择注册功能，向服务端发送 reg 指令。而此时的服务端已经开启了一个线程
pthread_create(&pid,NULL,*) ServerAceept, (void *)&soekfd)来接收客户端的连接。接收到客户端的连接请求后，服务端通过函
数 void ServerAccept(int *serverfd)再开辟一个子线程
pthread_create(&elientNnde. pid,NULL,(void*)handleclient,(void*)&cli

enk Node)来循环处理客户端发送过来的信息。服务端接收到封装在结构体 struet message 里的指令信息 reg,然后 ncNintregist(ruct message *recievemsg)函数。该函数主要有以下操作:

```
 int regist(struct message *recievemsg)
{
   Struct message cmpmsg;//用结构封装信息进行传输
   if(strcnmp(recievemsg->name,”all”,3)==0)
     Return-1;
  Fd=open(“user.txt”,O_RDWRO_CREATO_APPEND,0666))<0);
Do
{
  If((read-size=read(fd,&cmpmsg,sizeof(cmpmsg)))<0)
}while(read_size==sizeof(struct message));
   Write_size=write(fd,recivemsg,sizeof(struct message)
}
```

4．2．2 登录

同样服务端接受到客户端的 login 指令，然后进入 int login(struct message *recivvemsg)函数，该函数有以下操作：

```
Int login(struct message *recivemsg)
{
If(fd=open(“user.txt”,O_RDONLY))<0)
Do
{ if((strcmp(recivemsg->name,cmpmsg.name)==0)&&(strcmp(recievems
g->msg,cmpmsg.msg)==0))
{
Close(fd);
Return 0;
}
}while(read_size>0);
}
```

通过登录检测成功后，服务端用 insertendlist(clientlink,clientNode)将用户登录信息保存在列表中。

4．3 服务器的实现

4.3.1 私聊

客户端通过 who: message 的方式来发送私聊消息，服务端在 void handleclient(clientinf *client)函数中通过链表查找是否有 who 这个用户，如果有服务端则根据的 who 的套接字把 message 转发出去。

```
void  handleclient(clientinf*client)

while(1)

else  {

LinkList  L;

L=  clientlink;

L=L->next;

strcpy(mess.  name,c  lientNode.name);

while(L!=  NULL)//查找链表中的用户名是否和 who 相同

if  (strcmp(L->data.name,mess.flag)  ==  0)//查找成功{

/*向 who 用户转发信息*/

send(L->data.decr,&mess,sizeof(struct  message),0);

break;

L=  L->next;

}
Continue;
}
}
```

4.3.2 群聊

群聊是以 all: message 的方式发送信息的，和私聊的主要区别就是，这里要将链表的所有用户查找出来并且转发信息。

```
else  if  (strcmp(mess.flag,"all")=  0)

{if(strcmp(mess.msg.,")  !=  0)

{
LinkList  L;

L=  clientlink;
```

```
L=L->next;

strcpy(  mess.  name,clientNode.name);
while(L  !=  NULD)//向链表的所有用户发送信息.

send(L->data.decr,  &mess,s  izeo  f(struct  message),0);

L=  L->next;}
}
```

4．4 数据库访问或文件操作

聊天记录的保存是通过文件的操作实现的 具体代码

```
void ChatClient::saveRecord()
{ saveRecords("chatRecords.txt");
 }
 void ChatClient::saveRecords(const QString &fileName){
 QFile file(fileName);
if(!file.open(QFile::WriteOnly|QFile::Text))
{ QMessageBox::warning(this, tr("Application"),

tr("Cannotwritefile %1:\n%2.") .arg(fileName).arg(file.errorString())
);
return;
} QTextStream out(&file);
out ＜＜ chatBox-＞toPlainText();
}
```

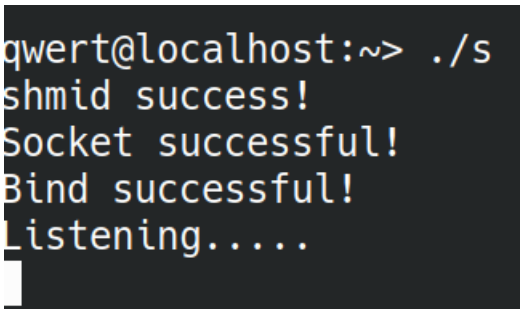
4．5 通信模块的实现

```
void udp_client(int sockfd, struct sockaddr_in *addr)
{
    int n;
    struct message sendmsg;
    struct message rcvmsg;



    printf("please enter user id(1 or 2):");
    scanf("%d", &sendmsg.id);
    strcpy(sendmsg.buf,"login");
    n = sendto(sockfd, (struct message *)&sendmsg, sizeof(struct message), 0, (struct
sockadrr *)addr, sizeof(struct sockaddr));
    n = recvfrom(sockfd, (struct message *)&rcvmsg, sizeof(struct message), 0,
NULL, NULL);
```

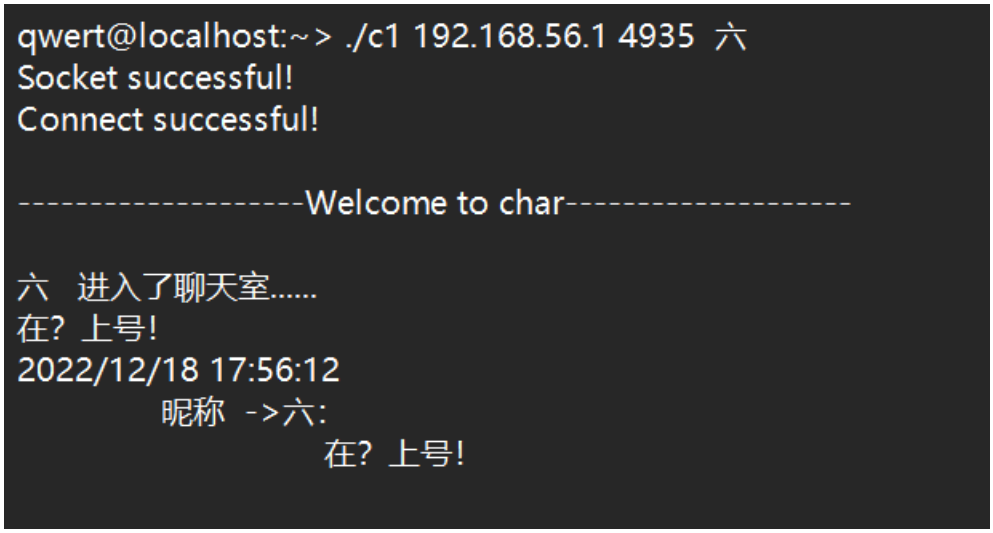
```
    printf("%s\n",rcvmsg.buf);
    for(;;){
        memset(sendmsg.buf, 0, MAX_BUF_SIZE);
        if(fgets(sendmsg.buf, MAX_BUF_SIZE, stdin) == NULL) break;
        n = sendto(sockfd, (struct message *)&sendmsg, sizeof(struct message), 0,
(struct sockadrr *)addr, sizeof(struct sockaddr));

        memset(rcvmsg.buf, 0, MAX_BUF_SIZE);
        n = recvfrom(sockfd, (struct message *)&rcvmsg, sizeof(struct message), 0,
NULL, NULL);
        printf("%d# user say:",rcvmsg.id);
        fputs(rcvmsg.buf, stdout);
    }
}

int  main(int  argc  ,char  *argv[])
{
int  sockfd;
struct  sockaddr_in  clientaddr;
sockfd  =  socket(AF_INET,  SOCK_DGRAM,  0);

if(sockfd  <  0)  {
perror("create  socket  error!\n");
exit(0);
}
bzero(&clientaddr,sizeof(clientaddr));
clientaddr.sin_family  =  AF_INET;
clientaddr.sin_port  =  htons(8003);
clientaddr.sin_addr.s_addr  =  inet_addr("127.0.0.1");
udp_client(sockfd,  &clientaddr);
close(sockfd);
}
```

5、测试与调试

5.1：服务器进行编译执行



5.2：客户端进行编译执行



5.3：服务器出现结果

```
qwert@localhost:~> ./s
shmid success!
Socket successful!
Bind successful!
Listening.....
Accept successful!

已连接了客户端1：192.168.56.1：34865
2022/12/18 17:56:12
        昵称 ->六:
                在？上号!

Accept successful!
已连接了客户端2： 192.168.56.1：34866
2022/12/18 17:57:25
        昵称 -> 七:
                冲冲冲!
```

5.4：另外打开一个终端执行对客户端代码编译执行

```
qwert@localhost:~>./c2 192.168.56.1 4935 七
Socket successful!
Connect successful!

--------------------Welcome to char--------------------

七 进入了聊天室.....
2022/12/18 17:56:12
        昵称 ->六:
                在？上号!

冲冲冲!
2022/12/18 17:57:25
        昵称 ->七:
                冲冲冲!
```

6、总结

这次课程设计由于我们所学知识有限所以开发出来的实时通讯软件并不复杂，但是我们感觉收获颇丰，不仅学习到了一些新知识，回顾了以前的一些关于 linux 的知识点，而且使自己的学习目标更加明确，学习方法更加完善，也体会到软件开发的趣味，更加清楚地认识到了自己在软件开发及学习上的一些不足之处。也提升了我们每一个成员的融入团队和团队协助的能力。

附录：程序代码

```
//s.c
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<netinet/in.h>
#include<sys/socket.h>
#include<string.h>
#include<unistd.h>
#include<signal.h>
#include<sys/ipc.h>
#include<errno.h>
#include<sys/shm.h>
#include<time.h>
#include<pthread.h>
#define PORT 4395
#define SIZE 1024
#define SIZE_SHMADD 2048
#define BACKLOG 3
int sockfd;
int fd[BACKLOG];
int i=0;
/*********套接字描述符*******/
int get_sockfd()
{        struct sockaddr_in server_addr; if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
{       fprintf(stderr,"Socket error:%s\n\a",strerror(errno));                exit(1); }else{
            printf("Socket successful!\n"); }         /*sockaddr 结构 */
      bzero(&server_addr,sizeof(struct sockaddr_in));
      server_addr.sin_family=AF_INET;
      server_addr.sin_addr.s_addr=htonl(INADDR_ANY);
      server_addr.sin_port=htons(PORT);
      /*绑定服务器的 ip 和服务器端口号*/
      if(bind(sockfd,(struct sockaddr *)(&server_addr),sizeof(struct sockaddr))==-1)
      {          fprintf(stderr,"Bind error:%s\n\a",strerror(errno));
            exit(1);
      } else{printf("Bind successful!\n");       }
       /* 设置允许连接的最大客户端数 */
       if(listen(sockfd,BACKLOG)==-1)
       {       fprintf(stderr,"Listen error:%s\n\a",strerror(errno)); exit(1);    } else{
            printf("Listening.....\n"); }
```

```
        return sockfd;
}

/*创建共享存储区*/
int shmid_create()
{          int shmid;          if((shmid = shmget(IPC_PRIVATE,SIZE_SHMADD,0777)) < 0)
{ perror("shmid error!"); exit(1); }
      else    printf("shmid success!\n");
      return shmid;
}
int main(int argc, char *argv[]) {       char shmadd_buffer[SIZE_SHMADD],buffer[SIZE];
struct sockaddr_in client_addr;
      int sin_size;
      pid_t ppid,pid;         int new_fd;
      int shmid;
      char *shmadd;
      /***********共享内存***************/
      shmid = shmid_create();
      //映射共享内存
      shmadd = shmat(shmid, 0, 0);
      /*****创建套接字描述符***********/
      int sockfd = get_sockfd();
      /*循环接收客户端*/
      while(1)
      {     /* 服务器阻塞,直到客户程序建立连接 */
         sin_size=sizeof(struct sockaddr_in);
         if((new_fd=accept(sockfd,(struct sockaddr *)(&client_addr),&sin_size))==-1)
         { fprintf(stderr,"Accept error:%s\n\a",strerror(errno));    exit(1);      }else{printf("Accept
successful!\n"); }
        fd[i++] = new_fd;
        printf("\n 已连接了客户端%d : %s:%d \n",i , inet_ntoa(client_addr.sin_addr),
ntohs(client_addr.sin_port));
          /*把界面发送给客户端*/
        memset(buffer,0,SIZE);
        strcpy(buffer,"\n————————————————————————————Welecom come char
————————————————————————————————————\n");
        send(new_fd,buffer,SIZE,0);
      //创建子进程客户端
      ppid = fork();    if(ppid == 0)
      {
           //将加入的新客户发送给所有在线的客户端/
         recv(new_fd,buffer,SIZE,0);
         strcat( buffer," 进入了聊天室....");
         for(i=0;i<BACKLOG;i++)
```

```
        {
          if(fd[i]!=-1)
          {
                send(fd[i],buffer,strlen(buffer),0);
          }
              }
          //创建子进程进行读写操作/
        pid = fork();
          while(1)
          {
            if(pid > 0)
            {
              //父进程用于接收信息/
              memset(buffer,0,SIZE);
              if((recv(new_fd,buffer,SIZE,0)) <= 0)
              {
                  close(new_fd);
                  exit(1); }
               memset(shmadd, 0, SIZE_SHMADD);
               strncpy(shmadd,  buffer,  SIZE_SHMADD);//将缓存区的客户端信息放入共享内
存里
               printf(" %s\n",buffer);
            }
            if(pid == 0)
            {
              //子进程用于发送信息/
              sleep(1);//先执行父进程
              if(strcmp(shmadd_buffer,shmadd) != 0)
              {
                  strcpy(shmadd_buffer,shmadd);
                  if(new_fd    > 0)
                  {
                      if(send(new_fd,shmadd,strlen(shmadd),0) == -1)
                      {
                         perror("send");
                      }
                      memset(shmadd, 0, SIZE_SHMADD);
                      strcpy(shmadd,shmadd_buffer);
                  }
              }
            }
         }
        }
    }       free(buffer);
```

```
    close(new_fd);
    close(sockfd);
    return 0;
}
```

//c.c

```
#include<stdio.h>
#include<netinet/in.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<string.h>
#include<stdlib.h>
#include<netdb.h>
#include<unistd.h>
#include<signal.h>
#include<errno.h>
#include<time.h>
#define SIZE 1024

int main(int argc, char *argv[])
{
    pid_t pid;
    int sockfd,confd;
    char buffer[SIZE],buf[SIZE];
    struct sockaddr_in server_addr;
```

```
struct sockaddr_in client_addr;
struct hostent *host;
short port;
char *name;
//四个参数
if(argc!=4)
{
    fprintf(stderr,"Usage:%s hostname \a\n",argv[0]);
    exit(1);
}
   //使用 hostname 查询 host 名字
if((host=gethostbyname(argv[1]))==NULL)
{
    fprintf(stderr,"Gethostname error\n");
    exit(1);
}
    port=atoi(argv[2]);
    name=argv[3];
   /*客户程序开始建立 sockfd 描述符 */
if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
{
    fprintf(stderr,"Socket
```

```
Error:%s\a\n",strerror(errno));
        exit(1);
    } else{
        printf("Socket successful!\n");
    }
        /*客户程序填充服务端的资料 */
    bzero(&server_addr,sizeof(server_addr)); // 初始化,置 0
    server_addr.sin_family=AF_INET;          // IPV4
    server_addr.sin_port=htons(port);   // (将本机器上的 short 数据转化为网络上的 short 数据)端口号
    server_addr.sin_addr=*((struct in_addr *)host->h_addr); // IP 地址
    /* 客户程序发起连接请求 */
    if(confd=connect(sockfd,(struct sockaddr *)(&server_addr),sizeof(struct sockaddr))==-1)
    {
        fprintf(stderr,"Connect Error:%s\a\n",strerror(errno));
        exit(1);
    }else{
        printf("Connect successful!\n");
    }
```

```
/*将客户端的名字发送到服务器端*/
send(sockfd,name,20,0);
 /*创建子进程，进行读写操作*/
pid = fork();//创建子进程
 while(1)
{
   /*父进程用于发送信息*/
        if(pid > 0)
        {
       /*时间函数*/
        struct tm *p;
        time_t timep;
        time(&timep);
        p = localtime(&timep);
        strftime(buffer,                    sizeof(buffer),
"%Y/%m/%d %H:%M:%S", p);
        /*输出时间和客户端的名字*/
        strcat(buffer," \n\t 昵称 ->");
        strcat(buffer,name);
        strcat(buffer,":\n\t\t  ");
        memset(buf,0,SIZE);
        fgets(buf,SIZE,stdin);
```

```
/*对客户端程序进行管理*/
    if(strncmp("e",buf,1)==0)
    {
      printf("该客户端下线...\n");
      strcat(buffer,"退出聊天室！");
      if((send(sockfd,buffer,SIZE,0)) <= 0)
      {
        perror("error send");
      }
      close(sockfd);
      sockfd = -1;
      exit(0);
    }else
  {
     strncat(buffer,buf,strlen(buf)-1);
     strcat(buffer,"\n");
   if((send(sockfd,buffer,SIZE,0)) <= 0)
     {
          perror("send");
     }
    }
  }
```

```
        else if(pid == 0)
        {
            /*子进程用于接收信息*/
              memset(buffer,0,SIZE);
              if(sockfd > 0)
              {
            if((recv(sockfd,buffer,SIZE,0)) <= 0)
              {
                 close(sockfd);
                 exit(1);
              }
              printf("%s\n",buffer);
           }
        }
    }   close(sockfd);
    return 0;
}
```