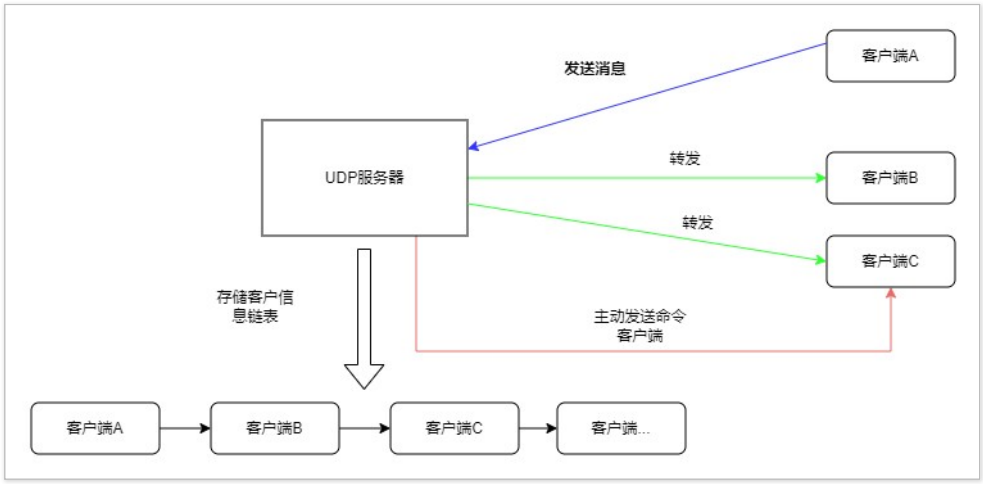
10.3 局域网聊天室之整体功能_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 10.3 局域网聊天室之整体功能涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

在上一个章节中，我们完成了群发服务器的功能。本节课我们在之前的基础上添加一些信息。



1. 添加一些伪代码思路
    1. 整体消息数据类型的设计
    2.
```
typedef struct
{
    char type;
    char name[15];
    char buf[1024];
}msg_t;
```
    3. 消息类型
    4.
```
#define CLIENT_LOGIN    10      加入服务器保存上线客户端链表
#define CLIENT_QUIT     20      从链表中删除（客户端输入"quit"）
#define CLIENT_TALK     30      遍历链表给其它客户端转发(客户端结果服务端转发)
#define SERVER_TALK     40      遍历链表给所有客户端转发(服务端主动发送)
#define SERVER_QUIT     50      遍历链表给所有客户端转发"quit"(广播信息),删除链表所有结点(
```
    5. 服务端保存链表结构体重新设计
        1. 注：因为服务器现在收和发都是独立的，故需要创建线程来实现。
    6.
```
typedef struct sockaddr_in datatype_t;


typedef struct  node
{
    datatype_t data;
    struct node *next;
}linknode_t;


typedef struct
{
    int sockfd;
    linknode_t*head;
}parm_t;
```

```
./client ip port 当前用户名
例如: ./client 127.0.0.1 8888 "jack"
```

1. 客户端代码设计思路
    1. 创建 socket
    2. 创建子线程，用于接收服务端发送过来的消息。
    3. ret = pthread_create(&tid,NULL,recv_message,&sockfd);
    4. 填充服务端的 ip 和 port，消息类型，及需要发送的字符。
        1. 填充 msg.type = CLIENT_LOGIN;
        2. strpcy(msg.buf,“I am login!\n”);
        3. 发送登录消息给服务端，只发送一次。
    5. 创建 while(1) 循环，发送数据。
        1. 从键盘输入数据存放到 msg.buf 中。
        2. 填充 msg.type = CLIENT_TALK. // 要与服务端进行交谈
        3. sendto 发送数据。
        4. 如果发送的 "quit" 消息，则更新 msg.type = CLIENT_QUIT;
        5. 把消息发送给服务端，然后循环结束。进程结束。
2. 服务器的设计
    1. 创建空的链表 head 来保存
    2. 创建 socket,bind() 绑定 ip 地址和端口
    3. 填充 parm.sockfd = sockfd;

- parm.head = head;
- 1. 创建子线程来发送消息给其他的客户端，传递 parm 参数
- ret = pthread_create(&tid,NULL,send_message,&parm);
- 注:
- 若是服务端主动发送聊天消息，则
- msg.type = SERVER_TALK;
- 若是服务端主动发送 "quit" 消息则。
- msg.type = SERVER_QUIT. // 所有客户端结束，链表清空
- 1. 主线程，while(1) 循环来接收所有的数据。
- n = recvfrom(sockfd,&msg,sizeof(msg),0,&peer_addr,&addrlen);

```
switch(msg.type)
{
case CLIENT_LOGIN:
    insert_head_linklist(head,peer_addr);
    boardcast_message(sockfd,head,&msg);
    break;

case CLIENT_TALK:
    boardcast_message(sockfd,head,&msg);
    break;

case CLIENT_QUIT:
    delete_linklist(head,peer_addr);
    boardcast_message(sockfd,head,&msg);
    break;
}
```

udp_client.h

```
#ifndef __HEAD_H__
#define __HEAD_H__
```

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

#define CLIENT_LOGIN    10
#define CLIENT_QUIT     20
#define CLIENT_TALK     30
#define SERVER_TALK     40
#define SERVER_QUIT     50

typedef struct
{
    char type;
    char name[15];
    char buf[1024];
}msg_t;

#endif
```

udp_client.c

```
#include "udp_client.h"

void *recv_message(void *arg)
{
    int sockfd = *(int *)arg;
    msg_t msg;
    int n = 0;
    struct sockaddr_in client_addr;
    int len = sizeof(client_addr);

    while(1)
    {
        memset(&msg,0,sizeof(msg));

        n = recvfrom(sockfd,&msg,sizeof(msg),0,(struct sockaddr *)&client_addr,&len);
        if(n < 0)
        {
            perror("Fail to recvfrom");
            exit(EXIT_FAILURE);
        }

        printf("%s say : %s\n",msg.name,msg.buf);

        if(msg.type == SERVER_QUIT)
            break;
    }

    exit(EXIT_SUCCESS);
}

void send_data(int sockfd,const char *name,struct sockaddr_in *addr,int len)
{
    int n = 0;
    msg_t msg;

    while(1)
    {
        putchar('>');
        memset(&msg,0,sizeof(msg));
        fgets(msg.buf,sizeof(msg.buf),stdin);
        msg.buf[strlen(msg.buf) - 1] = '\0';

        if(strncmp(msg.buf,"quit",4) == 0)
        {
            msg.type = CLIENT_QUIT;
        }else{
            msg.type = CLIENT_TALK;
        }

        strcpy(msg.name,name);

        n = sendto(sockfd,&msg,sizeof(msg),0,(struct sockaddr *)addr,len);
        if(n < 0)
        {
            perror("Fail to sendto");
            exit(EXIT_FAILURE);
```

```
        }

        if(msg.type == CLIENT_QUIT)
            break;
    }
    exit(EXIT_SUCCESS);
}


void send_login_data(int sockfd,const char *name,struct sockaddr_in *addr,int len)
{
    msg_t msg;
    int n = 0;

    msg.type = CLIENT_LOGIN;
    strcpy(msg.name,name);
    strcpy(msg.buf,"I am login!");

    n = sendto(sockfd,&msg,sizeof(msg),0,(struct sockaddr *)addr,len);
    if(n < 0)
    {
        perror("Fail to sendto");
        exit(EXIT_FAILURE);
    }

    return ;
}


int main(int argc, const char *argv[])
{
    int sockfd;
    struct sockaddr_in peer_addr;
    int len = sizeof(peer_addr);
    int ret = 0;
    pthread_t tid;
    if(argc != 4)
    {
        fprintf(stderr,"Usage : %s ip port username!\n",argv[0]);
        exit(EXIT_FAILURE);
    }


    sockfd = socket(AF_INET,SOCK_DGRAM,0);
    if(sockfd < 0)
    {
        perror("Fail to socket!");
        return -1;
    }


    memset(&peer_addr,0,sizeof(peer_addr));
    peer_addr.sin_family = AF_INET;
    peer_addr.sin_port = htons(atoi(argv[2]));
    peer_addr.sin_addr.s_addr = inet_addr(argv[1]);


    ret = pthread_create(&tid,NULL,recv_message,(void *)&sockfd);
    if(ret != 0)
    {
        fprintf(stderr,"Fail to pthread_create : %s\n",strerror(ret));
        exit(EXIT_FAILURE);
    }


    send_login_data(sockfd,argv[3],&peer_addr,len);


    send_data(sockfd,argv[3],&peer_addr,len);


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

Makefile

```
.PHONY : clean

CC := gcc
INCLUDE_DIR := -I .

OBJ := udp_client.o
```

```
LDFLAGS = -lpthread

TARGET := client

$(TARGET):$(OBJ)
    gcc $^ -o $@  $(LDFLAGS)

$(OBJ) : %.o : %.c
    $(CC) -c $< -o $@

clean:
    rm -rf *.o client
```

linklist.h

```
#ifndef __LINKLIST_H__
#define __LINKLIST_H__

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>

#define CLIENT_LOGIN    10
#define CLIENT_QUIT     20
#define CLIENT_TALK     30
#define SERVER_TALK     40
#define SERVER_QUIT     50

typedef struct
{
    char type;
    char name[15];
    char buf[1024];
}msg_t;

typedef struct sockaddr_in datatype_t;

typedef struct node
{
    datatype_t data;
    struct node *next;
}linknode_t;

typedef struct
{
    int sockfd;
    linknode_t *head;
}parm_t;

extern linknode_t *create_empty_linklist();
extern void insert_head_linklist(linknode_t *head,datatype_t data);
extern int find_linklist(linknode_t *head,datatype_t *data);
extern void broadcast_message(int sockfd,linknode_t *head,msg_t *msg,int msg_len);
extern int is_empty_linklist(linknode_t *head);
extern int delete_linklist(linknode_t *head,datatype_t data);
extern void clean_up(linknode_t *head);
#endif
```

linklist.c

```
#include "linklist.h"


linknode_t *create_empty_linklist()
{
    linknode_t *head = NULL;


    head = (linknode_t *)malloc(sizeof(linknode_t));

    if(NULL == head)
    {
        printf("malloc is fail!\n");
        return NULL;
    }

    memset(head,0,sizeof(linknode_t));
```

```
    return head;
}



void insert_head_linklist(linknode_t *head,datatype_t data)
{

    linknode_t *temp = (linknode_t *)malloc(sizeof(linknode_t));
    if(NULL == temp)
    {
        printf("malloc is fail!\n");
        return ;
    }


    temp->data = data;


    temp->next = head->next;
    head->next = temp;

    return ;
}


int find_linklist(linknode_t *head,datatype_t *data)
{
    linknode_t *p = head;

    while(p->next != NULL)
    {

        if(memcmp(&(p->data),data,sizeof(datatype_t)) == 0)
            return 1;

        p = p->next;
    }

    return 0;
}


void broadcast_message(int sockfd,linknode_t *head,msg_t *msg,int msg_len)
{
    linknode_t *p = head;

    while(p->next != NULL)
    {
        sendto(sockfd,msg,msg_len,0,(struct sockaddr *)(&(p->next->data)),sizeof(datatype_t));
        p = p->next;
    }
    return ;
}


void clean_up(linknode_t *head)
{
    linknode_t *p = head;
    linknode_t *q = NULL;

    while(p != NULL)
    {
        q = p->next;
        free(p);
        p = q;
    }
}

int is_empty_linklist(linknode_t *head)
{
    return head->next == NULL ? 1 : 0;
}


int delete_linklist(linknode_t *head,datatype_t data)
{
    linknode_t *p = head;
    linknode_t *q = NULL;
    int flag = 0;
    if(is_empty_linklist(head))
    {
        return -1;
    }
```

```
    while(p->next != NULL)
    {
        if(memcmp(&(p->next->data),&data,sizeof(datatype_t)) == 0)
        {

            q = p->next;
            p->next = q->next;
            free(q);
            q = NULL;

            flag = 1;

        }else{
            p = p->next;
        }

    }

    if(flag == 0)
        return -2;

    return 0;
}
```

udp_server.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include "linklist.h"

void recv_data(int sockfd,linknode_t *head)
{
    int n = 0;
    msg_t msg;
    struct sockaddr_in client_addr;
    int len = sizeof(client_addr);

    while(1)
    {
        memset(&msg,0,sizeof(msg));
        n = recvfrom(sockfd,&msg,sizeof(msg),0,(struct sockaddr *)&client_addr,&len);
        if(n < 0)
        {
            perror("Fail to recvfrom");
            exit(EXIT_FAILURE);
        }

        switch(msg.type)
        {
            case CLIENT_LOGIN:



                if(!find_linklist(head,&client_addr))
                {
                    insert_head_linklist(head,client_addr);
                }


                broadcast_message(sockfd,head,&msg,sizeof(msg));
                break;
            case CLIENT_TALK:
                broadcast_message(sockfd,head,&msg,sizeof(msg));
                break;
            case CLIENT_QUIT:
                delete_linklist(head,client_addr);
                broadcast_message(sockfd,head,&msg,sizeof(msg));
                break;
        }
    }
    return ;
}

int init_socket(const char *ip,const char *port)
{
    int sockfd;
    struct sockaddr_in my_addr;
    int len = sizeof(my_addr);
```

```
    sockfd = socket(AF_INET,SOCK_DGRAM,0);
    if(sockfd < 0)
    {
        perror("Fail to socket!");
        return -1;
    }


    memset(&my_addr,0,sizeof(my_addr));
    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(atoi(port));
    my_addr.sin_addr.s_addr = inet_addr(ip);


    if(bind(sockfd,(struct sockaddr *)&my_addr,len) < 0)
    {
        perror("Fail to bind");
        return -1;
    }

    printf("wait recv data!\n");

    return sockfd;
}

void *send_message(void *arg)
{
    int sockfd = ((parm_t *)arg)->sockfd;
    linknode_t *head = ((parm_t *)arg)->head;

    msg_t msg;

    while(1)
    {
        putchar('>');
        memset(&msg,0,sizeof(msg));
        fgets(msg.buf,sizeof(msg.buf),stdin);
        msg.buf[strlen(msg.buf) - 1] = '\0';

        strcpy(msg.name,"Server");
        if(strncmp(msg.buf,"quit",4) == 0)
        {

            msg.type = SERVER_QUIT;
            broadcast_message(sockfd,head,&msg,sizeof(msg));
            clean_up(head);
            break;
        }else{
            msg.type = SERVER_TALK;
            broadcast_message(sockfd,head,&msg,sizeof(msg));
        }
    }

    exit(EXIT_SUCCESS);
}


int main(int argc, const char *argv[])
{
    int sockfd;
    struct sockaddr_in my_addr;
    int len = sizeof(my_addr);
    linknode_t *head = NULL;
    parm_t parm;
    pthread_t tid;
    int ret;

    if(argc != 3)
    {
        fprintf(stderr,"Usage : %s ip port!\n",argv[0]);
        exit(EXIT_FAILURE);
    }


    sockfd = init_socket(argv[1],argv[2]);


    head = create_empty_linklist();

    parm.sockfd = sockfd;
    parm.head = head;


    ret = pthread_create(&tid,NULL,send_message,(void *)&parm);
    if(ret != 0)
```

```
    {
        fprintf(stderr,"Fail to pthread_create : %s\n",strerror(ret));
        exit(EXIT_FAILURE);
    }


    recv_data(sockfd,head);


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

Makefile

```
.PHONY : clean

CC := gcc
INCLUDE_DIR := -I .

LDFLAGS = -lpthread

OBJ := linklist.o udp_server.o

TARGET := server

$(TARGET):$(OBJ)
    gcc $^ -o $@ $(LDFLAGS)

$(OBJ) : %.o : %.c
    $(CC) -c $< -o $@

clean:
    rm -rf *.o server
```

全文完