大白老师 · 更新于 2022-12-21

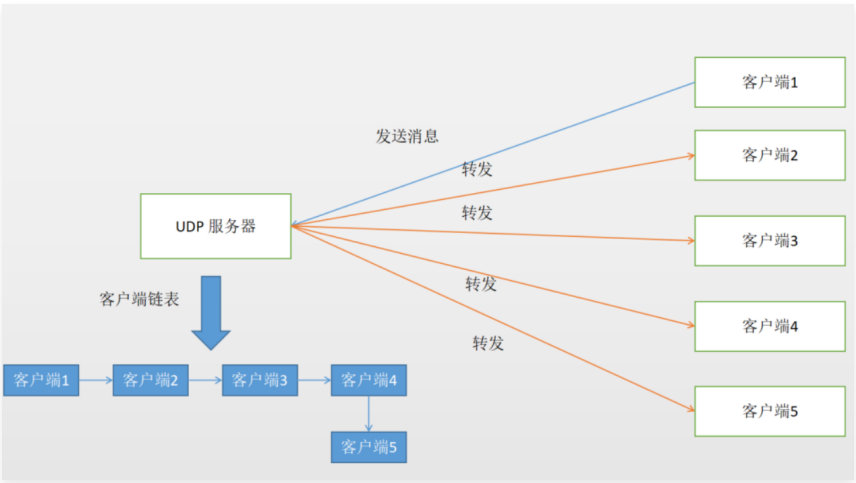
上一节 10.1 局域网聊天... 10.3 局域网聊天... 下一节

一.概述

- 说明

局域网聊天室整体代码量较多。客户端和服务器相关代码都比较多。这里我们根据需求先实现一个局域网群发服务器的功能，后期在对代码的架构进行扩展

- 项目需求
    - 需求
        - 服务端：实现一个服务器群发的功能，服务端设计一个链表，用来存储用户信息，当链表中某个用户发送数据的时候，链表中其他用户也可以收到。
        - 客户端：发送数据和接收数据应该是异步的。双方互不影响
        -



二.实现思路

1. 整体想法
    1. 客户端
        - 创建多线程，把客户端的收发分开。主线程发送数据，子线程接收数据
    2. 服务器
        - 创建1个链表
        - 链表中保存用户的ip + port。

2. 部门代码思路

1. 客户端多线程设计

2.

```
代码块
1  void *recv_message(void *arg)
2  {
3  ;
4  }
5
6  //创建子线程接收
7  pthread_create(&tid,NULL,recv_message,&sockfd);
8
9  //主线程发送
10 while(1)
11 {
12     //发送数据
13     sendto();
14 }
```

3. 服务器设计

- 链表节点设计

```
代码块
1    typedef struct sockaddr_in  datatype_t;
2
3    typedef struct node
4    {
5        datatype_t data;
6        struct node *next;
7    }linknode_t;
```

- 服务器遍历链表发送数据

```
代码块
1  //服务端收到消息
2  recvfrom(sockfd,buf,sizeof(buf),0,&peer_addr,&addrlen);
3
4  //判断该地址及端口是否在链表中已经存放了，不存在则把该地址
5  //填充到链表中去
6  insert_data_linklist(head,peer_addr);
7
8  //遍历链表，把消息发送给每一个客户端
9  int find_linklist(linknode_t*head,datatype_t data)
10 {
11     struct node *p;
12
13     for(p = head->next; p != NULL;  p = p->next)
14     {
15         //结构体类型数据不能直接比较
16         //if(p->data == data)
17         //        return 1;
18
19         if(memcmp(&(p->data),&data,sizeof(datatype_t)) == 0)
20                 return 1;
21
22     }
23
24     return 0;
```

- memcmp函数介绍
- memcmp简介，类似于strncmp。但是strncmp只能比较字符串。但是memcmp 函数能够按照内存字节来比较。

代码块

```
#include <string.h>

void *memcmp(const void *s1,const  void *s2,size_t n);
功能：memcmp是比较s1和s2所指向的内存前nbyte是否相同。（逐个字节的比较 ）
@s1    第一个比较字符数组的收地址
@s2    第二个比较字符数组的收地址
@n     比较前n个byte
返回值：
s1 > s2  ，返回1
s1 == s2,  返回 0
s1 < s2, 则返回 -1
```

三.代码总体思路

- 【bug】说明

由于此代码为项目的中间架构，此架构目前存在bug。所有登录的用户，需要先发送一个消息，用于接收数据，保持其IP地址和端口，之后才能收到数据。

- 代码设计
  - 服务端代码
  - linklist.h

代码块

```
#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>

typedef struct sockaddr_in datatype_t;

typedef struct node
{
    datatype_t data; //数据域保存有效数据
    struct node *next; //指针域保存下一个结点的地址
}linknode_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,char *msg,int ms
#endif
```

  - linklist.c

```c

#include "linklist.h"

//1.创建空的链表---为头结点在堆区分配空间
linknode_t *create_empty_linklist()
{
    linknode_t *head = NULL;

    //1.1 分配堆区空间
    head = (linknode_t *)malloc(sizeof(linknode_t));

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

    memset(head,0,sizeof(linknode_t));
    //head->next = NULL;
    //head->data = 0;
    return head;
}

//2.头插法：每次都在头结点后插入数据。
//特点：插入的顺序和输出的顺序是相反的。

void insert_head_linklist(linknode_t *head,datatype_t data)
{
    //2.1 为结点在堆区申请空间
    linknode_t *temp = (linknode_t *)malloc(sizeof(linknode_t));
    if(NULL == temp)
    {
        printf("malloc is fail!\n");
        return ;
    }

    //2.2 插入数据
    temp->data = data;

    //2.3 连接结点
    temp->next = head->next;
    head->next = temp;

    return ;
}

//在链表种查找该客户端是否已经存在用户信息，存放返回1，否则返回0
int find_linklist(linknode_t *head,datatype_t *data)
{
    linknode_t *p = head;

    while(p->next != NULL)
    {
        //说明链表中有该用户信息，老用户
        if(memcmp(&(p->data),data,sizeof(datatype_t)) == 0)
            return 1;

        p = p->next;
    }
    //若是循环结束，上面的memcmp都没有执行，说明是新用户，返回0
    return 0;
}

```

```
65  void broadcast_message(int sockfd,linknode_t *head,char *msg,int msg_len)
66  {
67      linknode_t *p = head;
68
69      while(p->next != NULL)
70      {
71          sendto(sockfd,msg,msg_len,0,(struct sockaddr *)(&(p->next->data)),
72          p = p->next;
73      }
74      return ;
    }
```

- udp_server.c

```
1
2   #include <stdio.h>
3   #include <sys/types.h>          /* See NOTES */
4   #include <sys/socket.h>
5   #include <netinet/in.h>
6   #include <arpa/inet.h>
7   #include <string.h>
8   #include <stdlib.h>
9   #include <unistd.h>
10  #include "linklist.h"
11
12  void recv_data(int sockfd,linknode_t *head)
13  {
14      int n = 0;
15      char buf[1024] = {0};
16      struct sockaddr_in client_addr;
17      int len = sizeof(client_addr);
18
19      while(1)
20      {
21          memset(buf,0,sizeof(buf));
22          n = recvfrom(sockfd,buf,sizeof(buf),0,(struct sockaddr *)&client_
23          if(n < 0)
24          {
25              perror("Fail to recvfrom");
26              exit(EXIT_FAILURE);
27          }
28
29          //若是新用户，ip + port可以到链表中
30          //若是链表中存在该用户信息，返回1
31          //否则，返回0
32          if(!find_linklist(head,&client_addr))
33          {
34              insert_head_linklist(head,client_addr);
35          }
36
37          //把接收到的消息，发送給所有的客户端
38          broadcast_message(sockfd,head,buf,n);
39      }
40      return ;
41  }
42
43  int init_socket(const char *ip,const char *port)
44  {
45      int sockfd;
46      struct sockaddr_in my_addr;
47      int len = sizeof(my_addr);
```

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

    //2.填充服务器自己的ip + port
    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);

    //3.把ip + port与socket绑定
    if(bind(sockfd,(struct sockaddr *)&my_addr,len) < 0)
    {
        perror("Fail to bind");
        return -1;
    }

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

    return sockfd;
}

//./a.out ip port
int main(int argc, const char *argv[])
{
    int sockfd;
    struct sockaddr_in my_addr;
    int len = sizeof(my_addr);
    linknode_t *head = NULL;
    if(argc != 3)
    {
        fprintf(stderr,"Usage : %s ip port!\n",argv[0]);
        exit(EXIT_FAILURE);
    }

    //1.初始化udp操作,获得套接字
    sockfd = init_socket(argv[1],argv[2]);

    //2.创建空的链表
    head = create_empty_linklist();

    //3.接收数据
    recv_data(sockfd,head);

    //4.关闭文件描述符
    close(sockfd);
    return 0;
}
```

- Makefile

代码块

```
.PHONY : clean

CC := gcc
INCLUDE_DIR := -I .

OBJ := linklist.o udp_server.o
```

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

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

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

- 客户端代码
- client.c

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

void *recv_message(void *arg)
{
    int sockfd = *(int *)arg;

    int n = 0;
    char buf[1024] = {0};
    struct sockaddr_in client_addr;
    int len = sizeof(client_addr);

    while(1)
    {
        memset(buf,0,sizeof(buf));
        //接收服务器转发的消息
        n = recvfrom(sockfd,buf,sizeof(buf),0,(struct sockaddr *)&client_a
        if(n < 0)
        {
            perror("Fail to recvfrom");
            exit(EXIT_FAILURE);
        }

        if(strncmp(buf,"quit",4) == 0)
            break;

        printf("recv %d bytes : %s\n",n,buf);
    }

    exit(EXIT_SUCCESS);
}

void send_data(int sockfd,struct sockaddr_in *addr,int len)
{
    int n = 0;
    char buf[1024] = {0};

    while(1)
    {
        putchar('>');
        memset(buf,0,sizeof(buf));
```

教程

底层核心技
驱动高级1

底层核心技
驱动高级2

人工智能必

居项目实战之

居项目实战之

居项目实战之

嵌入式项目

发阶段-三大

```c
        buf[strlen(buf) - 1] = '\0'; //'\n'--->'\0'

        if(strncmp(buf,"quit",4) == 0)
            break;

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

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

    //1.通过socket创建文件描述符
    sockfd = socket(AF_INET,SOCK_DGRAM,0);
    if(sockfd < 0)
    {
        perror("Fail to socket!");
        return -1;
    }

    //2.填充服务器的ip + port
    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]);

    //3.创建接收线程，接收数据
    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);
    }

    //3.发送数据
    send_data(sockfd,&peer_addr,len);

    //4.关闭文件描述符
    close(sockfd);
    return 0;
}
```

索引目录

一.概述

二.实现思路

三.代码总体思路

10.1 局域网聊天室项目概述 上一节 下一节 10.3 局域网聊天室之整体功能

意见反馈 收藏教程 标记书签