## Linux C — 聊天室

多线程聊天室

**服务器端：**

实现多用户群体聊天功能（人数上限可设置）；

每个用户所发送的消息，其他已连接服务器的用户均可以收到；

用户输入”bye”退出，服务器输入“quit”退出。

服务端总共有三个函数

主函数main：服务器端的初始化，接受连接；

消息处理函数rcv\_snd：接收某一用户的消息，简单处理后发送给其他所有用户；

退出函数quit：当输入quit时，服务器程序终止。

这三个函数分别属于三个线程（准确说是大于等于三个）。Main函数作为主线程，并创建一个退出函数所在的线程，以及一个每次连接新用户时创建一个对此连接的消息进行处理的线程。

**客户端：**

建立与服务器的连接，发送和接收消息

 客户端总共有两个函数

主函数main：初始化客户端套接字，与服务器建立连接，接收消息并在本地显示

发送函数snd：从控制台读入消息，发送到服务器

这两个函数分别在两个线程上运行，一个是主函数所在的线程，另一个是主函数创建的发送函数所在的线程。

附：

宏定义调试打印开关

#define DEBUG\_PRINT 1

#ifdef DEBUG\_PRINT

#define DEBUG(format, ...) printf(“FILE: “\_\_FILE\_\_”, LINE: %d: “format”\n”, \_\_LINE\_\_, ##\_\_VA\_ARGS\_\_)

#else

#define DEBUG(format, ...)

#endif

创建线程 gcc编译时需-lpthread

Phtread\_t thread;  Pthread\_creat(&thread, NULL, (void \*)quit, NULL);

Pthread\_creat(malloc(sizeof(pthread\_t)), NULL, (void \*)rcv\_snd, (void \*)i);

系统时间 #include <time.h>

Time\_t ticks = time(NULL); printf(“%s\n”, ctime(&ticks));

服务端

Struct sockaddr\_in serv\_addr;

Bzero(&serv\_addr, sizeof(serv\_addr));

Serv\_addr.sin\_family = AF\_INET;

Serv\_addr.sin\_port = htons(PORT);

Serv\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

Listenfd = socket(AF\_INET, SOCK\_STREAM, 0);

Bind(listenfd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr));

Listen(listenfd, 10);

Int len = sizeof(cli\_addr);

Connfd[i] = accept(listenfd, (struct sockaddr \*)&cli\_addr, &len);

客户端

Struct sockaddr\_in serv\_addr;

Bzero(&serv\_addr, sizeof(serv\_addr));

Serv\_addr.sin\_family = AF\_INET;

Serv\_addr.sin\_port = htons(PORT);

Inet\_pton(IP, &serv\_addr.sin\_addr);

Sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

Connect(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr));

**源代码 server.c**

1#include <stdio.h>

2 #include <stdlib.h> // exit

3 #include <string.h>

4 #include <unistd.h> // bind listen

5 #include <time.h> // time(NULL) ctime(&ticks)

6 #include <netinet/in.h>

7 #include <arpa/inet.h> // 必须包含，用于inet\_ntop

8

9 #define PORT 8000

10 #define MAXMEM 10

11 #define BUFFSIZE 128

12

13 //#define DEBUG\_PRINT 1 // 宏定义 调试开关

14 #ifdef DEBUG\_PRINT

15 #define DEBUG(format, ...) printf("FILE: "\_\_FILE\_\_", LINE: %d: "format"\n", \_\_LINE\_\_, ##\_\_VA\_ARGS\_\_)

16 #else

17 #define DEBUG(format, ...)

18 #endif

19

20 int listenfd, connfd[MAXMEM];

21

22 void quit();

23 void rcv\_snd(int p);

24

25 int main()

26 {

27 struct sockaddr\_in serv\_addr, cli\_addr;

28 // int len = sizeof(cli\_addr), i;

29 int i;

30 time\_t ticks;

31 pthread\_t thread;

32 char buff[BUFFSIZE];

33

34 printf("running...\n(Prompt: enter command ""quit"" to exit server)\n");

35 DEBUG("=== initialize..."); // 初始化填充服务端地址结构

36 bzero(&serv\_addr, sizeof(struct sockaddr\_in));

37 serv\_addr.sin\_family = AF\_INET;

38 serv\_addr.sin\_port = htons(PORT);

39 serv\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

40

41 DEBUG("=== socket..."); // socket 创建服务器端的监听套接字

42 listenfd = socket(AF\_INET, SOCK\_STREAM, 0);

43 if(listenfd < 0)

44 {

45 perror("fail to socket");

46 exit(-1);

47 }

48

49 DEBUG("=== bind..."); // bind 将套接字与填充好的地址结构进行绑定

50 if(bind(listenfd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0)

51 {

52 perror("fail to bind");

53 exit(-2);

54 }

55

56 DEBUG("=== listening..."); // listen 将主动连接套接字变为被动倾听套接字

57 listen(listenfd, MAXMEM);

58

59 /\* === 创建一个线程，对服务器程序进行管理，调用quit函数 === \*/

60 pthread\_create(&thread, NULL, (void \*)(quit), NULL);

61

62 // 将套接字描述符数组初始化为-1，表示空闲

63 for(i=0; i<MAXMEM; i++)

64 connfd[i] = -1;

65

66 while(1)

67 {

68 int len;// = sizeof(cli\_addr);

69 for(i=0; i<MAXMEM; i++)

70 {

71 if(connfd[i] == -1)

72 break;

73 }

74 // accept 从listen接受的连接队列中取得一个连接

75 connfd[i] = accept(listenfd, (struct sockaddr \*)&cli\_addr, &len);

76 if(connfd[i] < 0)

77 {

78 perror("fail to accept");

79 // continue; // 此句可以不用，accept会阻塞等待

80 }

81 ticks = time(NULL);

82 //sprintf(buff, "%.24s\r\n", ctime(&ticks));

83 printf("%.24s\n\tconnect from: %s, port %d\n",

84 ctime(&ticks), inet\_ntop(AF\_INET, &(cli\_addr.sin\_addr), buff, BUFFSIZE),

85 ntohs(cli\_addr.sin\_port)); // 注意 inet\_ntop的使用，#include <arpa/inet.h>

86

87 /\* === 针对当前套接字创建一个线程，对当前套接字的消息进行处理 === \*/

88 pthread\_create(malloc(sizeof(pthread\_t)), NULL, (void \*)(&rcv\_snd), (void \*)i);

89 }

90 return 0;

91 }

92

93 void quit()

94 {

95 char msg[10];

96 while(1)

97 {

98 scanf("%s", msg); // scanf 不同于fgets, 它不会读入最后输入的换行符

99 if(strcmp(msg, "quit") == 0)

100 {

101 printf("Byebye... \n");

102 close(listenfd);

103 exit(0);

104 }

105 }

106 }

107

108 void rcv\_snd(int n)

109 {

110 int len, i;

111 char name[32], mytime[32], buf[BUFFSIZE];

112 time\_t ticks;

113 int ret;

114

115 // 获取此线程对应的套接字用户的名字

116 write(connfd[n], "your name: ", strlen("your name: "));

117 len = read(connfd[n], name, 32);

118 if(len > 0)

119 name[len-1] = '\0'; // 去除换行符

120 strcpy(buf, name);

121 strcat(buf, "\tjoin in\n\0");

122 // 把当前用户的加入 告知所有用户

123 for(i=0; i<MAXMEM; i++)

124 {

125 if(connfd[i] != -1)

126 write(connfd[i], buf, strlen(buf));

127 }

128

129 while(1)

130 {

131 char temp[BUFFSIZE];

132 if((len=read(connfd[n], temp, BUFFSIZE)) > 0)

133 {

134 temp[len-1] = '\0';

135 // 当用户输入bye时，当前用户退出

136 if(strcmp(temp, "bye") == 0)

137 {

138 close(connfd[n]);

139 connfd[n] = -1;

140 pthread\_exit(&ret);

141 }

142 ticks = time(NULL);

143 sprintf(mytime, "%.24s\r\n", ctime(&ticks));

144 //write(connfd[n], mytime, strlen(mytime));

145 strcpy(buf, name);

146 strcat(buf, "\t");

147 strcat(buf, mytime);

148 strcat(buf, "\r\t");

149 strcat(buf, temp);

150 strcat(buf, "\n");

151

152 for(i=0; i<MAXMEM; i++)

153 {

154 if(connfd[i] != -1)

155 write(connfd[i], buf, strlen(buf));

156 }

157 }

158 }

159

160 }

**源代码 client.c**

1 /\*

2 \* FILE: client.c

3 \* DATE: 20180206

4 \* ==============

5 \*/

6

7 #include <stdio.h>

8 #include <stdlib.h>

9 #include <string.h>

10

11 #include <netinet/in.h>

12

13 #define BUFFSIZE 128

14 #define HOST\_IP "192.168.159.3"

15 #define PORT 8000

16

17 int sockfd;

18

19 void snd();

20

21 int main()

22 {

23 pthread\_t thread; // pthread\_t 线程，gcc编译时需加上-lpthread

24 struct sockaddr\_in serv\_addr; // struct sockaddr\_in

25 char buf[BUFFSIZE];

26 // 初始化服务端地址结构

27 bzero(&serv\_addr, sizeof(struct sockaddr\_in)); // bzero 清零

28 serv\_addr.sin\_family = AF\_INET; // sin\_family AF\_INET

29 serv\_addr.sin\_port = htons(PORT); // sin\_port htons(PORT)

30 inet\_pton(HOST\_IP, &serv\_addr.sin\_addr); // inet\_pton

31 // 创建客户端套接字

32 sockfd = socket(AF\_INET, SOCK\_STREAM, 0); // socket 创建套接字

33 if(sockfd < 0)

34 {

35 perror("fail to socket");

36 exit(-1);

37 }

38 // 与服务器建立连接

39 printf("connecting... \n");

40 if(connect(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) // connect

41 {

42 perror("fail to connect");

43 exit(-2);

44 }

45 /\* === 从此处开始 程序分做两个线程 === \*/

46 // 创建发送消息的线程，调用发送消息的函数snd

47 pthread\_create(&thread, NULL, (void \*)(&snd), NULL); // pthread\_create

48 // 接收消息的线程

49 while(1)

50 {

51 int len;

52 if((len=read(sockfd, buf, BUFFSIZE)) > 0) // read 读取通信套接字

53 {

54 buf[len] = '\0'; // 添加结束符，避免显示缓冲区中残留的内容

55 printf("\n%s", buf);

56 fflush(stdout); // fflush 冲洗标准输出，确保内容及时显示

57 }

58 }

59 return 0;

60 }

61

62 // 发送消息的函数

63 void snd()

64 {

65 char name[32], buf[BUFFSIZE];

66

67 fgets(name, 32, stdin); // fgets 会读取输入字符串后的换行符

68 write(sockfd, name, strlen(name)); // write 写入通信套接字

69 while(1)

70 {

71 fgets(buf, BUFFSIZE, stdin);

72 write(sockfd, buf, strlen(buf));

73 if(strcmp(buf, "bye\n") == 0) // 注意此处的\n

74 exit(0);

75 }

76 }

实验结果：





