Kejie Wang

Linux下的基于Pthread的多线程Socket编程

2016-05-27 | 0 comments | 166

本文主要基于Linux下的Pthread实现了一个服务器和客户端通过socket进行通信的系统。服务端的程序的设计主要是一个主线程首先调用Socket相关的函数socket，bind， listen在建立服务端的Socket之后，等待Accept上面，如果有新的客户端连接上来，则对于每一个客户端新建一个线程。在每一个客户端的线程中，其接收客户端发送的指令然后返回相关的信息，主线程Socket中默认的Accpet默认是阻塞的，这里采用了Linux中函数fctnl将其设置成非阻塞。除了客户端对应的线程和主线程之外，服务端还存在一个服务端的控制线程，其主要是接受服务端的命令，诸如退出，断开连接等等，对服务端的程序进行控制。客户端的程序为一个阻塞的循环，接受用户的输入然后解析命令进行相关的操作，诸如连接服务端，发送数据等等操作。

程序设计

服务端的程序的设计主要是一个主线程首先调用Socket相关的函数socket，bind， listen在建立服务端的Socket之后，等待Accept上面，如果有新的客户端连接上来，则对于每一个客户端新建一个线程。在每一个客户端的线程中，其接收客户端发送的指令然后返回相关的信息，主线程Socket中默认的Accpet默认是阻塞的，这里采用了Linux中函数fctnl将其设置成非阻塞。除了客户端对应的线程和主线程之外，服务端还存在一个服务端的控制线程，其主要是接受服务端的命令，诸如退出，断开连接等等，对服务端的程序进行控制。

客户端的程序为一个阻塞的循环，接受用户的输入然后解析命令进行相关的操作，诸如连接服务端，发送数据等等操作

服务端实现细节

服务端的程序主要包括主线程接受客户端请求响应建立客户端对应线程，客户端命令执行，服务端主要控制模块。

主线程

首先建立服务端Socket连接，其过程如下：

© 2017 ♥ Kejie Wang

- 首先调用socket函数建立一个socket，其中传入的参数AF_INET参数表示其采用TCP协议，并且调用Linux系统提供的fcntl函数设置这个socket为非阻塞。
- 然后初始化服务的地址，保护监听的IP（这里设置的监听服务器端所有的网卡），设置相应的端口以及协议类型（TCP），调用bind函数将其和前面的建立的socket绑定。
- 设置该socket开始监听，这主要对应TCP连接上面的第一次握手，将TCP第一次握手成功的放在队列（SYN队列）中，其中传入的参数BACKLOG为Socket中队列的长度。
- 然后调用Accept函数进行接收客户端的请求，其主要是从TCP上次握手成功的Accept队列中将客户端信息去除进行处理(此部分代码后面解释)。

最后服务端的主要是在一个循环中，其中最前面的serverExit主要是为了交互需要，后面会详细介绍。服务端调用Accept函数进行接收一个客户端的信息，如果存在连接上的客户端，则为其分配一个线程位置，然后调用pthread_create创建一个线程，并且将该客户端的信息参数传入线程。这里采用一个结构体的方式记录了客户端的相关信息，包括客户端IP，端口号，编号，是否连接等等。

这里为了防止服务器的无限的创建线程，设置了最大同时在线的客户端数目（MAXCONN），如果新连接上的客户端发现已经操作上线，那么该主线程就会睡眠，其等待在了一个客户端断开连接的信号量上面，如果有客户端断开连接，则主线程被唤醒，接受该客户端的连接。

需要注意的是这里涉及到很多的多线程的操作，很有可能发生数据冲突，需要很好使用pthread中的互斥锁防止发生数据冲突。

客户端服务线程

其主要是阻塞的接受客户端发送上来的命令，然后解析命令，对应每个命令进行不同的操作，将结果通过send函数发送给对应的客户端。这里主要实现了disconn断开连接，name返回客户端，list返回所有连接的客户端的信息（IP，端口，编号等等），send向别的客户端发送数据。
对于name和time命令只要将数据以字符串的形式发送给对应的客户端即可，而对于disconn命令，需要发送一个断开的信号，使得等待在这个信号量上的客户端能够连接上服务端，对于send命令需要根据发送的地址获取到目的的客户端信息，相对应的客户端发送相关的消息。服务端的程序另一个需要的是需要进行错误处理，诸如客户端的命令错误，格式错误，网络错误等等异常情况，其需要进行相应的报错，并且提供给客户端或者服务端相应的错误信息。
这里主要注意的是发送的数据需要指定数据的长度，而采用C的strlen指定的并不包括字符串最后的'\0'，因此其不会发送出去。这里有两种解决方案，一种是接收方在数据的最后加上'\0'或者发送的时候将'\0'一同发送出去。

服务端控制线程

服务端控制程序主要提供了一些命令给服务端可以对于服务端连接的客户端以及自身退出。这里主要实现了exit退出服务端，list列出所有连接的客户端信息，kill关闭某个客户端命令。其中exit命令为退出客户端，其主要是将一个退出退出标志位设置成1，然后主线程判断该标志位为1的时候会关闭所有的客户端，并且取消所有的客户端线程以及服务端程序线程，然后自身退出。对于list命令其和服务端发送的list指令实现相同，kill命令主要是关闭对于对应的客户端的socket，然后取消其对应的线程。

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

#define PORT  8888
#define BACKLOG 10
#define MAXCONN 100
#define BUFFSIZE 1024

typedef unsigned char BYTE;
typedef struct ClientInfo
{
    struct sockaddr_in addr;
    int clientfd;
    int isConn;
    int index;
} ClientInfo;

pthread_mutex_t activeConnMutex;
pthread_mutex_t clientsMutex[MAXCONN];
pthread_cond_t connDis;

pthread_t threadID[MAXCONN];
pthread_t serverManagerID;

ClientInfo clients[MAXCONN];

```

```
37  int serverExit = 0;
38
39  /*@brief Transform the all upper case
40  *
41  */
42  void tolowerString(char *s)
43  {
44      int i=0;
45      while(i < strlen(s))
46      {
47          s[i] = tolower(s[i]);
48          ++i;
49      }
50  }
51
52  void listAll(char *all)
53  {
54      int i=0, len = 0;
55      len += sprintf(all+len, "Index   \t\tIP Address   \t\tPort\n");
56      for(;i<MAXCONN;++i)
57      {
58          pthread_mutex_lock(&clientsMutex[i]);
59          if(clients[i].isConn)
60              len += sprintf(all+len, "%.8d\t\t%s\t\t%d\n",clients[i].index, inet_ntoa(clie
61          pthread_mutex_unlock(&clientsMutex[i]);
62      }
63  }
64
65  void clientManager(void* argv)
66  {
67      ClientInfo *client = (ClientInfo *)(argv);
68
69      BYTE buff[BUFFSIZE];
70      int recvbytes;
71
72      int i=0;
73      int clientfd = client->clientfd;
74      struct sockaddr_in addr = client->addr;
75      int isConn = client->isConn;
76      int clientIndex = client->index;
77
78      while((recvbytes = recv(clientfd, buff, BUFFSIZE, 0)) != -1)
79      {
80       //   buff[recvbytes] = '\0';
81          tolowerString(buff);    //case-insensitive
82
83          char cmd[100];
84          if((sscanf(buff, "%s", cmd)) == -1)    //command error
```

```
85          {
86             char err[100];
87             if(send(clientfd, err, strlen(err)+1, 0) == -1)
88             {
89                 strcpy(err, "Error command and please enter again!\n");
90                 fprintf(stdout, "%d sends an eroor command\n", clientfd);
91                 break;
92             }
93          }
94          else
95          {
96              char msg[BUFFSIZE]; //The message content
97              int dest = clientIndex; //message destination
98              int isMsg = 0;              //any message needed to send
99              if(strcmp(cmd, "disconn") == 0)
100             {
101                 pthread_cond_signal(&connDis);  //send a disconnetion signal and the wait
102                 break;
103             }
104             else if(strcmp(cmd, "time") == 0)
105             {
106                 time_t now;
107                 struct tm *timenow;
108                 time(&now);
109                 timenow = localtime(&now);
110                 strcpy(msg, asctime(timenow));
111                 isMsg = 1;
112             }
113             else if(strcmp(cmd, "name") == 0)
114             {
115                 strcpy(msg, "MACHINE NAME");
116                 isMsg = 1;
117             }
118             else if(strcmp(cmd, "list") == 0)
119             {
120                 listAll(msg);
121                 isMsg = 1;
122             }
123             else if(strcmp(cmd, "send") == 0)
124             {
125
126                 if(sscanf(buff+strlen(cmd)+1, "%d%s", &dest, msg)==-1 || dest >= MAXCONN)
127                 {
128                     char err[100];
129                     strcpy(err, "Destination ID error and please use list to check and en
130                     fprintf(stderr, "Close %d client eroor: %s(errno: %d)\n", clientfd, s
131                     break;
132                 }
```

```
133                 fprintf(stdout, "%d %s\n", dest, msg);
134                 isMsg = 1;
135             }
136             else
137             {
138                 char err[100];
139                 strcpy(err, "Unknown command and please enter again!\n");
140                 fprintf(stderr, "Send to %d message eroor: %s(errno: %d)\n", clientfd, st
141                 break;
142             }
143 
144 
145             if(isMsg)
146             {
147                 pthread_mutex_lock(&clientsMutex[dest]);
148                 if(clients[dest].isConn == 0)
149                 {
150                     sprintf(msg, "The destination is disconneted!");
151                     dest = clientIndex;
152                 }
153 
154                 if(send(clients[dest].clientfd, msg, strlen(msg)+1, 0) == -1)
155                 {
156                     fprintf(stderr, "Send to %d message eroor: %s(errno: %d)\n", clientfd
157                     pthread_mutex_unlock(&clientsMutex[dest]);
158                     break;
159                 }
160                 printf("send successfully!\n");
161                 pthread_mutex_unlock(&clientsMutex[dest]);
162             }
163         }  //end else
164     }   //end while
165 
166     pthread_mutex_lock(&clientsMutex[clientIndex]);
167     client->isConn = 0;
168     pthread_mutex_unlock(&clientsMutex[clientIndex]);
169 
170     if(close(clientfd) == -1)
171         fprintf(stderr, "Close %d client eroor: %s(errno: %d)\n", clientfd, strerror(erri
172     fprintf(stderr, "Client %d connetion is closed\n", clientfd);
173 
174     pthread_exit(NULL);
175 }
176 
177 void serverManager(void* argv)
178 {
179     while(1)
180     {
```

```
181             char cmd[100];
182             scanf("%s", cmd);
183             tolowerString(cmd);
184             if(strcmp(cmd, "exit") == 0)
185                 serverExit = 1;
186             else if(strcmp(cmd, "list") == 0)
187             {
188                 char buff[BUFFSIZE];
189                 listAll(buff);
190                 fprintf(stdout, "%s", buff);
191             }
192             else if(strcmp(cmd, "kill") == 0)
193             {
194                 int clientIndex;
195                 scanf("%d", &clientIndex);
196                 if(clientIndex >= MAXCONN)
197                 {
198                     fprintf(stderr, "Unkown client!\n");
199                     continue;
200                 }
201                 pthread_mutex_lock(&clientsMutex[clientIndex]);
202                 if(clients[clientIndex].isConn)
203                 {
204                     if(close(clients[clientIndex].clientfd) == -1)
205                         fprintf(stderr, "Close %d client eroor: %s(errno: %d)\n", clients[cl
206                 }
207                 else
208                 {
209                     fprintf(stderr, "Unknown client!\n");
210                 }
211                 pthread_mutex_unlock(&clientsMutex[clientIndex]);
212                 pthread_cancel(threadID[clientIndex]);
213 
214             }
215             else
216             {
217                 fprintf(stderr, "Unknown command!\n");
218             }
219         }
220     }
221 
222     int main()
223     {
224         int activeConn = 0;
225 
226         //initialize the mutex
227         pthread_mutex_init(&activeConnMutex, NULL);
228         pthread_cond_init(&connDis, NULL);
```

```
229     int i=0;
230     for(;i<MAXCONN;++i)
231         pthread_mutex_init(&clientsMutex[i], NULL);
232
233     for(i=0;i<MAXCONN;++i)
234         clients[i].isConn = 0;
235
236     //create the server manager thread
237     pthread_create(&serverManagerID, NULL, (void *)(serverManager), NULL);
238
239
240     int listenfd;
241     struct sockaddr_in  servaddr;
242
243     //create a socket
244     if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
245     {
246         fprintf(stderr, "Create socket error: %s(errno: %d)\n", strerror(errno), errno);
247         exit(0);
248     }
249     else
250         fprintf(stdout, "Create a socket successfully\n");
251
252     fcntl(listenfd, F_SETFL, O_NONBLOCK);       //set the socket non-block
253
254     //set the server address
255     memset(&servaddr, 0, sizeof(servaddr));  //initialize the server address
256     servaddr.sin_family = AF_INET;           //AF_INET means using TCP protocol
257     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);    //any in address(there may more than
258     servaddr.sin_port = htons(PORT);            //set the port
259
260     //bind the server address with the socket
261     if(bind(listenfd, (struct sockaddr*)(&servaddr), sizeof(servaddr)) == -1)
262     {
263          fprintf(stderr, "Bind socket error: %s(errno: %d)\n", strerror(errno), errno);
264          exit(0);
265     }
266     else
267         fprintf(stdout, "Bind socket successfully\n");
268
269     //listen
270     if(listen(listenfd, BACKLOG) == -1)
271     {
272         fprintf(stderr, "Listen socket error: %s(errno: %d)\n", strerror(errno), errno);
273         exit(0);
274     }
275     else
276         fprintf(stdout, "Listen socket successfully\n");
```

```
277
278
279     while(1)
280     {
281         if(serverExit)
282         {
283             for(i=0;i<MAXCONN;++i)
284             {
285                 if(clients[i].isConn)
286                 {
287                     if(close(clients[i].clientfd) == -1)         //close the client
288                         fprintf(stderr, "Close %d client eroor: %s(errno: %d)\n", clients|
289                     if(pthread_cancel(threadID[i]) != 0)         //cancel the correspondi
290                          fprintf(stderr, "Cancel %d thread eroor: %s(errno: %d)\n", (int)(
291                 }
292             }
293             return 0;    //main exit;
294         }
295
296         pthread_mutex_lock(&activeConnMutex);
297         if(activeConn >= MAXCONN)
298              pthread_cond_wait(&connDis, &activeConnMutex);
299         pthread_mutex_unlock(&activeConnMutex);
300
301         //find an empty postion for a new connnetion
302         int i=0;
303         while(i<MAXCONN)
304         {
305             pthread_mutex_lock(&clientsMutex[i]);
306             if(!clients[i].isConn)
307             {
308                 pthread_mutex_unlock(&clientsMutex[i]);
309                 break;
310             }
311             pthread_mutex_unlock(&clientsMutex[i]);
312             ++i;
313         }
314
315         //accept
316         struct sockaddr_in addr;
317         int clientfd;
318         int sin_size = sizeof(struct sockaddr_in);
319         if((clientfd = accept(listenfd, (struct sockaddr*)(&addr), &sin_size)) == -1)
320         {
321             sleep(1);
322             //fprintf(stderr, "Accept socket error: %s(errno: %d)\n", strerror(errno), er
323             continue;
324             //exit(0);
```

```
325         }
326         else
327             fprintf(stdout, "Accept socket successfully\n");
328 
329         pthread_mutex_lock(&clientsMutex[i]);
330         clients[i].clientfd = clientfd;
331         clients[i].addr = addr;
332         clients[i].isConn = 1;
333         clients[i].index = i;
334         pthread_mutex_unlock(&clientsMutex[i]);
335 
336         //create a thread for a client
337         pthread_create(&threadID[i], NULL, (void *)clientManager, &clients[i]);
338 
339     }    //end-while
340 }
```

客户端实现细节

客户端程序主要可以分成客户端主线程控制和服务端消息接受程序。

主线程

客户端的程序较服务端的程序则简单的多，主要是接受用户的指令按照指令执行相关的任务。这里主要实现了conn连接服务器，disconn断开服务器连接，list查询服务器上的所有连接，name查看服务器名字，time查看时间，send先其他的客户端发送消息这些命令。

客户端首先必须通过conn命令连接上对应的服务器，然后可以执行其他的指令。用户通过time，name查看时间和机器名，如果需要发送消息，可以通过list查看所有连接，然后向指定的连接发送消息，quit退出客户端。

服务器消息处理线程

由于主线程为一个阻塞的线程，因此这里创建了一个接受服务器端的消息的线程。当用户通过conn命令连接上服务器之后，主线程就会创建该接受线程，其不断的监听服务器发回的消息，然后将其输入的屏幕上。当用户通过disconn命令取消连接的时候，主线程则就会取消掉这个线程。下面是该线程相关的代码。

```
1  #include <sys/types.h>
2  #include <sys/socket.h>
3  #include <stdio.h>
4  #include <netinet/in.h>
5  #include <arpa/inet.h>
```

```
 6  #include <errno.h>
 7  #include <unistd.h>
 8  #include <string.h>
 9  #include <stdlib.h>
10  #include <pthread.h>
11
12  #define BUFFERSIZE 1024
13  typedef unsigned char BYTE;
14  pthread_t receiveID;
15
16  void tolowerString(char *s)
17  {
18      int i=0;
19      while(i < strlen(s))
20      {
21          s[i] = tolower(s[i]);
22          ++i;
23      }
24  }
25
26  void receive(void *argv)
27  {
28      int sockclient = *(int*)(argv);
29      BYTE recvbuff[BUFFERSIZE];
30      while(recv(sockclient, recvbuff, sizeof(recvbuff), 0)!=-1) //receive
31      {
32          fputs(recvbuff, stdout);
33          fputs("\n", stdout);
34      }
35      fprintf(stderr, "Receive eroor: %s(errno: %d)\n", strerror(errno), errno);
36  }
37
38  int main()
39  {
40      ///define sockfd
41      int sockclient = socket(AF_INET,SOCK_STREAM, 0);
42
43      ///definet sockaddr_in
44      struct sockaddr_in servaddr;
45      memset(&servaddr, 0, sizeof(servaddr));
46
47      int isConn = 0;
48
49      BYTE buff[BUFFERSIZE];
50
51
52      while (fgets(buff, sizeof(buff), stdin) != NULL)
53      {
```

```
54          tolowerString(buff);
55          char cmd[100], ip[100];
56          int port;
57          if(sscanf(buff, "%s", cmd) == -1)    //command error
58          {
59              fprintf(stderr, "Input eroor: %s(errno: %d) And please input again\n", strerr
60              continue;
61          }
62          if(strcmp(cmd, "conn") == 0)        //connecton command
63          {
64              char ip[100];
65              int port, ipLen=0;
66              if(sscanf(buff+strlen(cmd)+1, "%s", ip) == -1)    //command error
67              {
68                  fprintf(stderr, "Input eroor: %s(errno: %d) And please input again\n", st
69                  continue;
70              }
71              if((sscanf(buff+strlen(cmd)+strlen(ip)+2, "%d", &port)) == -1)    //command e
72              {
73                  fprintf(stderr, "Input eroor: %s(errno: %d) And please input again\n", st
74                  continue;
75              }
76             // fprintf(stdout, "%s %d\n",ip, port);
77              servaddr.sin_family = AF_INET;
78              servaddr.sin_port = htons(port);  ///server port
79              servaddr.sin_addr.s_addr = inet_addr(ip);  //server ip
80              if (connect(sockclient, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
81              {
82                  fprintf(stderr, "Connect eroor: %s(errno: %d)\n", strerror(errno), errno)
83                  continue;
84              }
85              fprintf(stdout, "Connect successfully\n");
86              isConn = 1;
87              pthread_create(&receiveID, NULL, (void *)(receive), (void *)(&sockclient));
88          }
89          else if(strcmp(cmd, "disconn") == 0)
90          {
91              if(isConn == 0)
92              {
93                  fprintf(stdout, "There is not a connection!\n");
94                  continue;
95              }
96              else
97              {
98                  pthread_cancel(receiveID);
99                  close(sockclient);
100             }
101             isConn = 0;
```

```
102           }
103           else if(strcmp(cmd, "quit") == 0)
104           {
105               if(isConn)
106               {
107                   pthread_cancel(receiveID);
108                   close(sockclient);
109               }
110               return 0;
111           }
112           else
113           {
114               if(send(sockclient, buff, strlen(buff)+1, 0) == -1) //send
115               {
116                   fprintf(stderr, "Send eroor: %s(errno: %d)\n", strerror(errno), errno);
117                   continue;
118               }
119
120               if(isConn == 0)
121               {
122                   fprintf(stdout, "Please use conn <ip> <port> command to build a connnect:
123                   continue;
124               }
125               memset(buff, 0, sizeof(buff));
126           }
127       }
128
129       close(sockclient);
130       return 0;
131   }
```

程序测试结果

实验测试采用了3个客户端同时连接一个服务端进行测试。同时运行这四个程序，首先3个客户端通过conn命令连接上客户端，我们发现3个客户端都同时连接上，说明conn命令正常工作，socket建立正确。然后在客户端1中我们对一些命令进行测试，输入name发现返回了机器的名字，输入time返回了时间，输入list，方向其显示了3个客户端的连接信息。这说明了name，time和list都正常工作。然后在客户端中通过send命令先客户端2发送一条消息，发现客户端2收到了消息，然后客户端2回复了一条消息，客户端1也收到了消息。先客户端3发送消息，其也正常的收到，其发送消息给客户端2，客户端2也能正常收到。着说明了程序中send命令运行正常，能够进行消息的收发。最后客户端1,2,3都通过quit命令退出了客户端，并且服务器端收到了该退出的消息，并且关闭了该连接和线程，最后服务器端退出了。

服务器

```
jack@jack-virtual-machine:~/Documents/course/ComputerNetwork/SocketProgramming$
./server
Create a socket successfully
Bind socket successfully
Listen socket successfully
Accept socket successfully
Accept socket successfully
Accept socket successfully
send successfully!
send successfully!
send successfully!
1 hello,client1!
send successfully!
0 hello!
send successfully!
2 hello,client2!
send successfully!
1 hello,client1!
send successfully!
list
Index                   IP Address              Port
00000000                127.0.0.1               55682
00000001                127.0.0.1               55938
00000002                127.0.0.1               56194
send successfully!
send successfully!
Send to 6 message eroor: Success(errno: 0)
Client 6 connetion is closed
Send to 4 message eroor: Success(errno: 0)
Client 4 connetion is closed
Send to 5 message eroor: Success(errno: 0)
Client 5 connetion is closed
exit
```

客户端1

```
jack@jack-virtual-machine:~/Documents/course/ComputerNetwork/SocketProgramming$
./client
conn 127.0.0.1 8888
Connect successfully
name
MACHINE NAME
list
Index                   IP Address              Port
00000000                127.0.0.1               55682
00000001                127.0.0.1               55938
00000002                127.0.0.1               56194

time
Fri May 27 02:17:20 2016

send 1 Hello,client1!
hello!
send 2 Hello,client2!
```

客户端2

```
jack@jack-virtual-machine:~/Documents/course/ComputerNetwork/SocketProgramming$
./client
conn 127.0.0.1 8888
Connect successfully
hello,client1!
send 0 Hello!
hello,client1!
time
Fri May 27 02:19:47 2016
```

客户端3

```
jack@jack-virtual-machine:~/Documents/course/ComputerNetwork/SocketProgramming$
./client
conn 127.0.0.1 8888
Connect successfully
hello,client2!
send 1 hello,client1!
time
Fri May 27 02:19:49 2016
```

linux, pthread, socket

‹ uC/OS-II在STM32F103上的移植 树莓派上的GPIO字符驱动程序 ›

0 Comments Sort by Oldest

Add a comment...

Facebook Comments Plugin