Manuel de référence

Généré par Doxygen 1.8.11

Table des matières

Chapitre 1

Index des structures de données

1.1 Structures de données

Liste des structures de données avec une brève description :

Chapitre 2

Index des fichiers

2.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

Chapitre 3

Documentation des structures de données

3.1 Référence de la structure charValue

Structure de données qui contient les données environnementales mesurées par le BME280.

Champs de données

- float Temp
- float Pressure
- float Humidity

3.1.1 Description détaillée

Structure de données qui contient les données environnementales mesurées par le BME280.

Définition à la ligne 23 du fichier BME280.c.

3.1.2 Documentation des champs

3.1.2.1 float charValue : :Humidity

Humidité mesurée.

Définition à la ligne 27 du fichier BME280.c.

3.1.2.2 float charValue : :Pressure

Pression locale mesurée.

Définition à la ligne 26 du fichier BME280.c.

3.1.2.3 float charValue : :Temp

Température ambiante mesurée.

Définition à la ligne 25 du fichier BME280.c.

La documentation de cette structure a été générée à partir du fichier suivant :

- src/BME280.c

3.2 Référence de la structure socketInt

Structure de données pour la configuration du socket.

Champs de données

— int Client
— int Server

3.2.1 Description détaillée

Structure de données pour la configuration du socket.

Définition à la ligne 22 du fichier SocketImplementation.c.

3.2.2 Documentation des champs

3.2.2.1 int socketInt : :Client

Entier correspondant au socket client.

Définition à la ligne 24 du fichier SocketImplementation.c.

3.2.2.2 int socketInt : :Server

Entier correspondant au socket serveur.

Définition à la ligne 25 du fichier SocketImplementation.c.

La documentation de cette structure a été générée à partir du fichier suivant :
— src/SocketImplementation.c

Chapitre 4

Documentation des fichiers

4.1 Référence du fichier src/BME280.c

Fichier modifié qui contient l'implémentation des fonctions pour le senseur BME280.

```
#include <stdio.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <fcntl.h>
```

Structures de données

— struct charValue
Structure de données qui contient les données environnementales mesurées par le BME280.

Fonctions

— struct charValue dataReader ()
Fonction de lecture des données environnementales à partir du BME280.

4.1.1 Description détaillée

Fichier modifié qui contient l'implémentation des fonctions pour le senseur BME280.

Auteur

Jean Jacques Akoffodji

Date

03 Mars 2018

Distributed with a free-will license. Use it any way you want, profit or free, provided it fits in the licenses of its associated works. This code is designed to work with the BME280_I2CS I2C Mini Module available from Control↩ Everything.com.

4.1.2 Documentation des fonctions

4.1.2.1 struct charValue dataReader ( )

Fonction de lecture des données environnementales à partir du BME280.

Renvoie

oneData structure de données les mesures du BME280

Définition à la ligne 36 du fichier BME280.c.

```
37 {
38     struct charValue oneData;
39     // Create I2C bus
40     int file;
41     char *bus = "/dev/i2c-1";
42     if((file = open(bus, O_RDWR)) < 0)
43     {
44         printf("Failed to open the bus. \n");
45         exit(1);
46     }
47     // Get I2C device, BME280 I2C address is 0x76(136)
48     ioctl(file, I2C_SLAVE, 0x77);
49
50     // Read 24 bytes of data from register(0x88)
51     char reg[1] = {0x88};
52     write(file, reg, 1);
53     char b1[24] = {0};
54     if(read(file, b1, 24) != 24)
55     {
56         printf("Error : Input/Output error \n");
57         exit(1);
58     }
59
60     // Convert the data
61     // temp coefficents
62     int dig_T1 = (b1[0] + b1[1] * 256);
63     int dig_T2 = (b1[2] + b1[3] * 256);
64     if(dig_T2 > 32767)
65     {
66         dig_T2 -= 65536;
67     }
68     int dig_T3 = (b1[4] + b1[5] * 256);
69     if(dig_T3 > 32767)
70     {
71         dig_T3 -= 65536;
72     }
73
74     // pressure coefficents
75     int dig_P1 = (b1[6] + b1[7] * 256);
76     int dig_P2 = (b1[8] + b1[9] * 256);
77     if(dig_P2 > 32767)
78     {
79         dig_P2 -= 65536;
80     }
81     int dig_P3 = (b1[10] + b1[11] * 256);
82     if(dig_P3 > 32767)
83     {
84         dig_P3 -= 65536;
85     }
86     int dig_P4 = (b1[12] + b1[13] * 256);
87     if(dig_P4 > 32767)
88     {
89         dig_P4 -= 65536;
90     }
91     int dig_P5 = (b1[14] + b1[15] * 256);
92     if(dig_P5 > 32767)
93     {
94         dig_P5 -= 65536;
95     }
96     int dig_P6 = (b1[16] + b1[17] * 256);
97     if(dig_P6 > 32767)
98     {
99         dig_P6 -= 65536;
100     }
101     int dig_P7 = (b1[18] + b1[19] * 256);
102     if(dig_P7 > 32767)
103     {
104         dig_P7 -= 65536;
105     }
106     int dig_P8 = (b1[20] + b1[21] * 256);
107     if(dig_P8 > 32767)
108     {
109         dig_P8 -= 65536;
110     }
```

```
111     int dig_P9 = (b1[22] + b1[23] * 256);
112     if(dig_P9 > 32767)
113     {
114         dig_P9 -= 65536;
115     }
116 
117     // Read 1 byte of data from register(0xA1)
118     reg[0] = 0xA1;
119     write(file, reg, 1);
120     char data[8] = {0};
121     read(file, data, 1);
122     int dig_H1 = data[0];
123 
124     // Read 7 bytes of data from register(0xE1)
125     reg[0] = 0xE1;
126     write(file, reg, 1);
127     read(file, b1, 7);
128 
129     // Convert the data
130     // humidity coefficents
131     int dig_H2 = (b1[0] + b1[1] * 256);
132     if(dig_H2 > 32767)
133     {
134         dig_H2 -= 65536;
135     }
136     int dig_H3 = b1[2] & 0xFF ;
137     int dig_H4 = (b1[3] * 16 + (b1[4] & 0xF));
138     if(dig_H4 > 32767)
139     {
140         dig_H4 -= 65536;
141     }
142     int dig_H5 = (b1[4] / 16) + (b1[5] * 16);
143     if(dig_H5 > 32767)
144     {
145         dig_H5 -= 65536;
146     }
147     int dig_H6 = b1[6];
148     if(dig_H6 > 127)
149     {
150         dig_H6 -= 256;
151     }
152 
153     // Select control humidity register(0xF2)
154     // Humidity over sampling rate = 1(0x01)
155     char config[2] = {0};
156     config[0] = 0xF2;
157     config[1] = 0x01;
158     write(file, config, 2);
159     // Select control measurement register(0xF4)
160     // Normal mode, temp and pressure over sampling rate = 1(0x27)
161     config[0] = 0xF4;
162     config[1] = 0x27;
163     write(file, config, 2);
164     // Select config register(0xF5)
165     // Stand_by time = 1000 ms(0xA0)
166     config[0] = 0xF5;
167     config[1] = 0xA0;
168     write(file, config, 2);
169 
170     // Read 8 bytes of data from register(0xF7)
171     // pressure msb1, pressure msb, pressure lsb, temp msb1, temp msb, temp lsb, humidity lsb, humidity msb
172     reg[0] = 0xF7;
173     write(file, reg, 1);
174     read(file, data, 8);
175 
176     // Convert pressure and temperature data to 19-bits
177     long adc_p = ((long)(data[0] * 65536 + ((long)(data[1] * 256) + (long)(data[2] & 0xF0)))) / 16;
178     long adc_t = ((long)(data[3] * 65536 + ((long)(data[4] * 256) + (long)(data[5] & 0xF0)))) / 16;
179     // Convert the humidity data
180     long adc_h = (data[6] * 256 + data[7]);
181 
182     // Temperature offset calculations
183     float var1 = (((float)adc_t) / 16384.0 - ((float)dig_T1) / 1024.0) * ((float)dig_T2);
184     float var2 = ((((float)adc_t) / 131072.0 - ((float)dig_T1) / 8192.0) *
185                   (((float)adc_t)/131072.0 - ((float)dig_T1)/8192.0)) * ((float)dig_T3);
186     float t_fine = (long)(var1 + var2);
187     float cTemp = (var1 + var2) / 5120.0;
188 
189     // Pressure offset calculations
190     var1 = ((float)t_fine / 2.0) - 64000.0;
191     var2 = var1 * var1 * ((float)dig_P6) / 32768.0;
192     var2 = var2 + var1 * ((float)dig_P5) * 2.0;
193     var2 = (var2 / 4.0) + (((float)dig_P4) * 65536.0);
194     var1 = (((float) dig_P3) * var1 * var1 / 524288.0 + ((float) dig_P2) * var1) / 524288.0;
195     var1 = (1.0 + var1 / 32768.0) * ((float)dig_P1);
196     float p = 1048576.0 - (float)adc_p;
197     p = (p - (var2 / 4096.0)) * 6250.0 / var1;
```

```
198     var1 = ((float) dig_P9) * p * p / 2147483648.0;
199     var2 = p * ((float) dig_P8) / 32768.0;
200     float pressure = (p + (var1 + var2 + ((float)dig_P7)) / 16.0);
201
202     // Humidity offset calculations
203     float var_H = (((float)t_fine) - 76800.0);
204     var_H = (adc_h - (dig_H4 * 64.0 + dig_H5 / 16384.0 * var_H)) * (dig_H2 / 65536.0 * (1.0 + dig_H6 /
      67108864.0 * var_H * (1.0 + dig_H3 / 67108864.0 * var_H)));
205     float humidity = var_H * (1.0 -  dig_H1 * var_H / 524288.0);
206     if(humidity > 100.0)
207     {
208         humidity = 100.0;
209     }else
210         if(humidity < 0.0)
211         {
212             humidity = 0.0;
213         }
214
215
216     oneData.Temp = cTemp;
217     oneData.Pressure = pressure;
218     oneData.Humidity = humidity;
219     return(oneData);
220
221 }
```

4.2 Référence du fichier src/DesiredTemp.c

Fichier qui contient la fonction main du programme dédié à lecture de la température désirée entrée par l'utilisateur.

```
#include "SocketImplementation.c"
```

Fonctions

— void main ()

4.2.1 Description détaillée

Fichier qui contient la fonction main du programme dédié à lecture de la température désirée entrée par l'utilisateur.

Auteur

Jean Jacques Akoffodji

Date

03 Mars 2018

4.2.2 Documentation des fonctions

4.2.2.1 void main ( )

Définition à la ligne 10 du fichier DesiredTemp.c.

```
11 {
12         int socket_temp;
13
14     socket_temp = socketConfiguration(4599);
15
16     while(1)
17     {
18         double d;
19                 char temp[20];
20                 char input[20];
21
22                 printf("\nEnter desired temp [5.00, 30.00]\n");
23                 fgets(input, 20, stdin);
```

```
24
25                  d = atof(input);
26
27                  if(d < 5 || d > 30) {
28                          printf("\nTemp outside valid interval\n");
29                  } else {
30                          sprintf(temp, "%.2f", d);
31                          printf("\nSending temp: %s\n", temp);
32
33              send(socket_temp, &d, sizeof(d),0);
34
35                  }
36          }
37
38          close(socket_temp);
39  }
```

4.3 Référence du fichier src/Interface.c

Fichier qui contient la fonction main du programme principale et les fonctions utilisées pour les threads.

```
#include <semaphore.h>
#include <pthread.h>
#include <sched.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include "BME280.c"
#include "SocketImplementation.c"
```

Fonctions

- void sendTempD (int socket_desc)
  Fonction d'envoi de la température désirée et de la puissance calculée au programme d'affichage.
- void * socketBME280 (void *x)
  Fonction associée au thread d'acquisition des données environnementales.
- void * affichageBME280 (void *x)
  Fonction associée au thread d'envoi périodique des données environnementales au programme d'affichage.
- void * affichageTempDesired (void *x)
  Fonction associée au thread d'envoi périodique de la température désirée au programme d'affichage.
- void powerCalculation ()
  Fonction de calcul de la puissance nécessaire.
- void * socketTempDesired (void *x)
  Fonction associée au thread de reception de la température désirée entrée par l'utilisateur.
- void main ()

Variables

- struct charValue dataBME280
  Variable globale contenant les données environnementales mesurées.
- double globalPower
  Variable globale pour la puissance calculée.
- double temperatureDesired
  Variable globale pour la température entrée par l'utilisateur.
- sem_t mutex
  Variable globale pour la protection des ressources partagées par les threads.
- int socketAffichage
  Variable globale contenant le socket connecté au programme d'affichage.

4.3.1 Description détaillée

Fichier qui contient la fonction main du programme principale et les fonctions utilisées pour les threads.

Auteur

Jean Jacques Akoffodji

Date

03 Mars 2018

4.3.2 Documentation des fonctions

4.3.2.1 void * affichageBME280 (void * *x*)

Fonction associée au thread d'envoi périodique des données environnementales au programme d'affichage.

Définition à la ligne 121 du fichier Interface.c.

```
122 {
123     char message[15];
124 
125     struct sched_param param;
126 
127     param.sched_priority = 2;
128     sched_setscheduler(0, SCHED_FIFO, &param);
129 
130     while (1)
131     {
132 
133         sleep(3);
134         sem_wait(&mutex);
135         sprintf(message, "TP%.2f\n",dataBME280.Temp);
136         puts(message);
137         if( send(socketAffichage , &message , strlen(message) , 0) < 0)
138         {
139             puts("Temperature ambiante Send failed");
140         }
141 
142         sprintf(message, "PR%.2f\n",dataBME280.Pressure);
143         if( send(socketAffichage , &message , strlen(message) , 0) < 0)
144         {
145             puts("Pressure Send failed");
146         }
147 
148         sprintf(message, "HU%.2f\n",dataBME280.Humidity);
149         if( send(socketAffichage , &message , strlen(message) , 0) < 0)
150         {
151             puts("Humidity Send failed");
152         }
153         sem_post(&mutex);
154 
155     }
156 }
```

4.3.2.2 void * affichageTempDesired (void * *x*)

Fonction associée au thread d'envoi périodique de la température désirée au programme d'affichage.

Définition à la ligne 163 du fichier Interface.c.

```
164 {
165     char message[15];
166 
167     struct sched_param param;
168 
169     param.sched_priority = 1;
170     sched_setscheduler(0, SCHED_FIFO, &param);
171 
172     while (1)
173     {
174 
175         sleep(3);
176         sem_wait(&mutex);
177 
178         sprintf(message, "TD%.2f\n",temperatureDesired);
179         puts(message);
180         if( send(socketAffichage , &message , strlen(message) , 0) < 0)
```

```
181         {
182             puts("Temperature ambiante Send failed");
183         }
184 
185         sem_post(&mutex);
186 
187     }
188 }
```

4.3.2.3 void main ( )

Définition à la ligne 251 du fichier Interface.c.

```
252 {
253 
254     struct charValue oneData;
255     int Server, Client;
256     struct socketInt socketTemp, socketBME;
257     double temp_desired;
258     double powerNeed;
259     char message[50];
260     struct charValue val;
261 
262 
263 
264     pthread_t threadBME;
265     pthread_t threadTempD;
266     pthread_t threadAffichage;
267     pthread_t threadAffichageTempD;
268 
269 
270     sem_init(&mutex,0,1);
271 
272 
273     socketAffichage = socketConfiguration(1234);
274 
275 
276     pthread_create(&threadBME,NULL,socketBME280,NULL);
277 
278     pthread_create(&threadTempD,NULL,socketTempDesired,NULL);
279 
280     pthread_create(&threadAffichage,NULL,affichageBME280,NULL);
281 
282     pthread_create(&threadAffichageTempD,NULL,affichageTempDesired,NULL);
283 
284 
285     pthread_join(threadBME,NULL);
286     pthread_join(threadTempD,NULL);
287     pthread_join(threadAffichage,NULL);
288     pthread_join(threadAffichageTempD,NULL);
289 
290 }
```

4.3.2.4 void powerCalculation ( )

Fonction de calcul de la puissance nécessaire.

Définition à la ligne 196 du fichier Interface.c.

```
197 {
198     globalPower = ((temperatureDesired-dataBME280.
      Temp)/6)*100;
199 
200     if (globalPower < 0 )
201     {
202         globalPower = 0;
203     }
204 
205     if (globalPower > 100 )
206     {
207         globalPower = 100;
208     }
209 }
```

4.3.2.5 void sendTempD (int *socket_desc*)

Fonction d'envoi de la température désirée et de la puissance calculée au programme d'affichage.

Paramètres

socket_desc	le socket client connecté au programme d'affichage

Définition à la ligne 59 du fichier Interface.c.

```
63 {
64     char message[12];
65 
66     sprintf(message, "TD%.2f\n",temperatureDesired);
67     puts(message);
68     if( send(socket_desc , &message , strlen(message) , 0) < 0)
69     {
70         puts("Temperature Desired Send failed");
71     }
72 
73     sprintf(message, "PW%.2f\n",globalPower);
74     puts(message);
75     if( send(socket_desc , &message , strlen(message) , 0) < 0)
76     {
77         puts("Power Send failed");
78     }
79 
80 }
```

4.3.2.6 void * socketBME280 (void * *x*)

Fonction associée au thread d'acquisition des données environnementales.

Définition à la ligne 87 du fichier Interface.c.

```
88 {
89     char message[30];
90     struct socketInt socketBME;
91     socketBME = configuration_server(4598);
92 
93     struct sched_param param;
94 
95     param.sched_priority = 3;
96     sched_setscheduler(0, SCHED_FIFO, &param);
97 
98     while (1)
99     {
100 
101         sleep(3);
102         sem_wait(&mutex);
103         if (recv(socketBME.Client, &dataBME280, sizeof(dataBME280), 0) < 0)
104         {
105             printf("erreur au recv() data BME280\n");
106         }
107         sprintf(message, "Temperature mesured %.2f\n",dataBME280.Temp);
108         printf(message);
109         sprintf(message, "Humidity mesured %.2f\n",dataBME280.Humidity);
110         printf(message);
111         sprintf(message, "Pressure mesured %.2f\n",dataBME280.Pressure);
112         printf(message);
113         sem_post(&mutex);
114     }
115 }
```

4.3.2.7 void * socketTempDesired (void * *x*)

Fonction associée au thread de reception de la température désirée entrée par l'utilisateur.

Définition à la ligne 216 du fichier Interface.c.

```
217 {
218     char message[30];
219     struct socketInt socketTemp;
220     double oneTemp;
221     socketTemp = configuration_server(4599);
222     struct sched_param param;
223 
224     param.sched_priority = 4;
225     sched_setscheduler(0, SCHED_FIFO, &param);
226 
227     sleep(1);
228 
229     while (1)
230     {
231 
232         if (recv(socketTemp.Client, &oneTemp, sizeof(oneTemp), 0) < 0)
233         {
234             printf("erreur au recv() data Temp desired\n");
235         }
236         sprintf(message, "Temperature desired %.2f\n",oneTemp);
237 
238         printf(message);
239 
240         sem_wait(&mutex);
241 
242         temperatureDesired =oneTemp;
243         powerCalculation();
244         sendTempD(socketAffichage);
245 
246         sem_post(&mutex);
247     }
248 }
```

4.3.3 Documentation des variables

4.3.3.1 struct charValue dataBME280

Variable globale contenant les données environnementales mesurées.

Définition à la ligne 23 du fichier Interface.c.

4.3.3.2 double globalPower

Variable globale pour la puissance calculée.

Définition à la ligne 30 du fichier Interface.c.

4.3.3.3 sem_t mutex

Variable globale pour la protection des ressources partagées par les threads.

Définition à la ligne 44 du fichier Interface.c.

4.3.3.4 int socketAffichage

Variable globale contenant le socket connecté au programme d'affichage.

Définition à la ligne 51 du fichier Interface.c.

4.3.3.5 double temperatureDesired

Variable globale pour la température entrée par l'utilisateur.

Définition à la ligne 37 du fichier Interface.c.

4.4 Référence du fichier src/SensorData.c

Fichier qui contient la fonction main du programme dédié à l'aquisition de données du BME280.

```
#include "BME280.c"
#include "SocketImplementation.c"
```

Fonctions

— void main ()

4.4.1 Description détaillée

Fichier qui contient la fonction main du programme dédié à l'aquisition de données du BME280.

Auteur

Jean Jacques Akoffodji

Date

03 Mars 2018

4.4.2 Documentation des fonctions

4.4.2.1 void main ( )

Définition à la ligne 13 du fichier SensorData.c.

```
14 {
15         int socketBME;
16     struct charValue val;
17
18     socketBME = socketConfiguration(4598);
19
20     while(1)
21     {
22
23         sleep(1);
24         val = dataReader();
25         send(socketBME, &val, sizeof(val),0);
26
27         }
28
29         close(socketBME);
30 }
```

4.5 Référence du fichier src/SocketImplementation.c

Fichier qui contient l'implémentation des fonctions de configuration pour le socket.

```
#include <sys/time.h>
#include <sys/resource.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
```

Structures de données

— struct socketInt
Structure de données pour la configuration du socket.

Macros

— #define LISTEN_BACKLOG 50

Fonctions

— int socketConfiguration (int PORT)
Fonction de configuration d'un socket client en localhost.
— struct socketInt configuration_server (int PORT)
Fonction de configuration d'un socket serveur en localhost.

4.5.1 Description détaillée

Fichier qui contient l'implémentation des fonctions de configuration pour le socket.

Auteur

Jean Jacques Akoffodji

Date

03 Mars 2018

4.5.2 Documentation des macros

4.5.2.1 #define LISTEN_BACKLOG 50

Définition à la ligne 14 du fichier SocketImplementation.c.

4.5.3 Documentation des fonctions

4.5.3.1 struct socketInt configuration_server (int *PORT*)

Fonction de configuration d'un socket serveur en localhost.

Paramètres

PORT	le port sur lequel on se connecte /return unSocket la structure de données contenant les socket serveur et client

Définition à la ligne 76 du fichier SocketImplementation.c.

```
81 {
82     int socket_error, Client, Server;
83     struct sockaddr_in server_ad, client_ad;
84     socklen_t server_size, client_size;
85     struct socketInt unSocket;
86 
87     Server = socket(AF_INET, SOCK_STREAM, 0);
88     if (Server < 0)
89     {
90         printf("erreur de creation du socket\n");
91         exit(1);
92     }
93 
```

```
94      server_size = sizeof(server_ad);
95      client_size = sizeof(client_ad);
96
97      bzero((char *)&server_ad,sizeof(server_ad));
98      server_ad.sin_family      = AF_INET;
99      server_ad.sin_addr.s_addr = INADDR_ANY;
100      server_ad.sin_port = htons(PORT);
101
102
103      bzero((char *)&client_ad,sizeof(client_ad));
104      client_ad.sin_family      = AF_INET;
105      client_ad.sin_addr.s_addr = INADDR_ANY;
106      client_ad.sin_port = htons(PORT);
107
108
109      if (bind(Server, (struct sockaddr *)&server_ad, sizeof(server_ad)) < 0)
110      {
111          printf("erreur au bind()\n");
112          exit(1);
113      }
114
115
116      if (listen(Server, LISTEN_BACKLOG) < 0)
117      {
118          printf("erreur au listen()\n");
119          exit(1);
120      }
121
122      printf("Attente d'une connexion du client\n");
123      Client = accept(Server, (struct sockaddr *)&client_ad, &client_size);
124      printf("Un client se connecte avec la socket %d de %s:%d\n \n", Client, inet_ntoa(client_ad.sin_addr),
     htons(client_ad.sin_port));
125
126      unSocket.Client = Client;
127      unSocket.Server = Server;
128
129      return unSocket;
130 }
```

4.5.3.2 int socketConfiguration (int *PORT*)

Fonction de configuration d'un socket client en localhost.

Paramètres

PORT	le port sur lequel on se connecte /return socket_desc le socket client

Définition à la ligne 34 du fichier SocketImplementation.c.

```
39 {
40      int socket_desc;
41      struct sockaddr_in server;
42
43
44
45      //Create socket
46      socket_desc = socket(AF_INET , SOCK_STREAM , 0);
47      if (socket_desc == -1)
48      {
49          printf("Could not create socket");
50      }
51      else
52      {
53          puts("Socket created");
54      }
55
56      server.sin_addr.s_addr = inet_addr("127.0.0.1");
57      server.sin_family = AF_INET;
58      server.sin_port = htons(PORT);
59
60      //Connect to remote server
61      if (connect(socket_desc , (struct sockaddr *)&server , sizeof(server)) < 0)
62      {
63          puts("connect error");
64      }
65
66      puts("Connected");
67
```

```
68     return socket_desc;
69 }
```

Index