```
1  /*
2   * File:   ServerUDPAll.h
3   * Author: user
4   *
5   * Created on 12 dicembre 2008, 11.09
6   */
7
8  #ifndef _SERVERUDPALL_H
9  #define _SERVERUDPALL_H
10 #define MAX_ETH 1500 //valore massimo di caratteri che un frame ethernet puo' contenere
11 //#include "addLib.h"
12 #include "Lista_TCP.h"
13 #include <sys/time.h>
14 #define SEC 0
15 #define USEC 500000
16
17 class Socket
18 {
19 private:
20 public:
21     Socket();//costruttore della classe socket
22 protected:
23     int sock_id; //sock_id diventera' l'identificatore del socket
24 };
25
26 Socket::Socket()
27 {}
28
29 class SocketUDP :public Socket //eredita la classe Socket in SocketUDP
30 {
31 private:
32 public:
33     SocketUDP(); //costruttore
34     void invia(char*,Address*); //metodo che invia una stringa
35     char* ricevi(Address *); //metodo che riceve una stringa inviata
36 protected:
37 };
38
39 SocketUDP::SocketUDP():Socket()
40 {
41     sock_id=socket(AF_INET,SOCK_DGRAM,0);//creo un socket TCP/IP che utilizza il protocollo UDP. Ritornera'un identificatore per essere localizzato
42 }
43
44 class SocketTCP :public Socket //eredita la classe Socket in SocketTCP
45 {
46 private:
47     int flag;
48 public:
49     SocketTCP(int); //costruttore
50 protected:
51 };
52
53 SocketTCP::SocketTCP(int band):Socket()
54 {
```

```
55      struct timeval tempo;
56      int err;
57      flag=band;
58      sock_id=socket(AF_INET,SOCK_STREAM,0);//creo un socket TCP/IP che utilizza
           il protocollo TCP. Ritornera'un identificatore per essere localizzato
59      if(band==1)
60      {
61       tempo.tv_sec=SEC;
62       tempo.tv_usec=USEC;
63       err=setsockopt(sock_id, SOL_SOCKET, SO_RCVTIMEO,&tempo,(socklen_t)sizeof
           (tempo));
64        if(err<0)
65        {
66         errore("SockOpt:",err);
67        }
68      }
69  }
70
71  class ServerTCP:public SocketTCP
72  {
73  private:
74      List* myList;
75  public:
76      ServerTCP(Address*,int);//costruttore che effettua il legame tra socket e
          porta del server
77      ServerTCP(Address*,List*,int);
78      int accetta();
79      Iterator* createIterator();
80      ~ServerTCP();
81      bool close(Connection*);
82  protected:
83  };
84
85  ServerTCP::~ServerTCP()
86  {
87      shutdown(sock_id,SHUT_RDWR);
88      delete(myList);
89  }
90
91  ServerTCP::ServerTCP(Address* indirServer,int band)
92  :SocketTCP(band)
93  {
94      int ret_code;
95      myList=new List();
96      ret_code=bind(sock_id,(struct sockaddr*)&(indirServer->get_binary
          ()),sizeof(Binary));//lega socket e porta del server
97      listen(sock_id,18); //numero connessioni accettabili
98      if(ret_code<0)
99          errore("bind()=",ret_code);
100 }
101
102 ServerTCP::ServerTCP(Address* indirServer,List* _newL,int band)
103 :SocketTCP(band)
104 {
105     int ret_code;
106     myList=_newL;
```

```
107     ret_code=bind(sock_id,(struct sockaddr*)&(indirServer->get_binary
          ()),sizeof(Binary));//lega socket e porta del server
108     if(ret_code<0)
109         errore("bind()=",ret_code);
110 }
111
112 Iterator* ServerTCP::createIterator()
113 {
114     return(myList->createIterator());
115 }
116
117 int ServerTCP::accetta()
118 {
119     int conn_id,len_mittente;
120     Connection* conny;
121     Binary sock;
122     Address* myself;
123     len_mittente=sizeof(Binary);
124
125     conn_id=accept(sock_id,(struct sockaddr*)&(sock),(socklen_t*)
          &len_mittente);
126     if(conn_id > 0)
127     {
128           myself=new Address(sock);
129           conny=new Connection(myself,conn_id);
130           myList->addOnTail(conny);
131     }
132     return (conn_id);
133 }
134
135 bool ServerTCP::close(Connection* conny)
136 {
137
138     return (myList->remove((Node*)conny)) ;
139 }
140
141 class ClientTCP :public SocketTCP
142 {
143 private:
144     Connection* myServer;
145 public:
146     ClientTCP(int);
147     Connection* Connetti(Address*);
148     char* ricevi();
149     void invia(char*);
150     ~ClientTCP();
151 protected:
152 };
153
154 ClientTCP::~ClientTCP()
155 {
156     delete(myServer);
157     shutdown(sock_id,SHUT_RDWR);
158 }
159
160 ClientTCP::ClientTCP(int band)
```

```
161 :SocketTCP(band)
162 {
163 }
164
165 Connection* ClientTCP::Connetti(Address* _Add)
166 {
167    int conn_id,len_addr;
168    len_addr=sizeof(Binary);
169    conn_id=connect(sock_id,(struct sockaddr*)&(_Add->get_binary()),(socklen_t)
        len_addr);
170     if(conn_id < 0)
171     {
172      errore("Connect err:",conn_id);
173     }
174    myServer= new Connection(_Add,sock_id);
175    return (myServer);
176 }
177
178 char* ClientTCP::ricevi()
179 {
180     return(myServer->ricevi());
181 }
182
183 void ClientTCP::invia(char* msg)
184 {
185     myServer->invia(msg);
186 }
187
188 void SocketUDP::invia(char* msg,Address *indirMitt)
189 {
190  int len_tx;
191  int binLen;
192  int lenMsg;
193  binLen=sizeof(Binary); //inserisci in binLen la grandezza di un Binary
194  lenMsg=lenStr(msg); //inserisci in lenMsg la lunghezza del messaggio
195  len_tx=sendto(sock_id,msg,lenMsg,0,(struct sockaddr*)&(indirMitt->get_binary
       ()),(socklen_t)binLen);//funzione che invia una stringa
196  if(len_tx!=lenStr(msg)) //se la lunghezza del messaggio inviato
       effettivamente e' minore della lunghezza del messaggio originale
197  {
198      errore("sendto()=",len_tx);//gestione dell'errore in invio
199  }
200 };
201
202 char* SocketUDP::ricevi(Address *indirMitt)
203 {
204     int rx_len;
205     int len_mittente;
206     char buffer[MAX_ETH];//variabile che conterra' il messaggio ricevuto
207     Binary mittente;
208     len_mittente=sizeof(Binary);
209     rx_len=recvfrom(sock_id,buffer,MAX_ETH,0,(struct sockaddr*)&mittente,
         (socklen_t*)&len_mittente); //riceve un messaggio inviato dall'esterno e
          ritorna la sua lunghezza
210     if(rx_len>0)//se il messaggio ricevuto ha almeno un carattere al'interno
211     {
```

```
212         buffer[rx_len]='\0';//inserisci il valore di fine-stringa ad essa
213         //printf("Ho ricevuto: %s",buffer);
214     }
215     else //altrimenti
216     {
217         errore("recvfrom()=",rx_len);//ritorna il genere di errore riscontrato
218     }
219     indirMitt->set_binary(mittente);//inserisci la binary compilata dalla
          funzione recvfrom in quella che e' stata inviata dal programma chiamante
           questa funzione
220     return (cpyStr(buffer)); //ritorna la copia del messaggio ricevuto
221 }
222
223 class ServerUDP:public SocketUDP
224 {
225 private:
226 public:
227     ServerUDP(Address*);//costruttore che effettua il legame tra socket e
          porta del server
228     //void errore(char*,int);
229 protected:
230 };
231
232 ServerUDP::ServerUDP(Address* indirServer):SocketUDP()
233 {
234     int ret_code;
235     ret_code=bind(sock_id,(struct sockaddr*)&(indirServer->get_binary
          ()),sizeof(Binary));//lega socket e porta del server
236     if(ret_code<0)
237         errore("bind()=",ret_code);
238 }
239
240 class ClientUDP :public SocketUDP
241 {
242 private:
243 public:
244     ClientUDP();
245 protected:
246 };
247
248 ClientUDP::ClientUDP()
249 {
250 }
251
252 #endif  /* _SERVERUDPALL_H */
253
254
```