```
// UDPclient.c

/* Testo del problema
   Il processo client invia la stringa "INVIAMI UN NUMERO" diverse volte in un intervallo
   di tempo di 1 secondo  ad un processo server in esecuzione su un host che ha indirizzo IP
   e numero di porta (1050) digitati da tastiera.

   Il server, se riconosce la stringa ricevuta "INVIAMI UN NUMERO", risponde inviando ogni volta
   il numero progressivo successivo  al numero attualmente memorizzato nel file number.txt

   Il client dovrà contare e visualizzare quanti valori è riuscito a ricevere  dal sever
   in un secondo e scrivere (append)  questo numero nel file NumRicevuti.txt

   E' stata implementata la funzione registra(...) salvata nella libreria registra.h

   Prova effettuata: UDPclient 127.0.0.1   (127.0.0.1 è l'indirizzo IP dell'host
                                            su cui è attivo il processo server;
                                            in questo caso server e  client   sono sullo stesso host)
*/

#include <stdio.h>
#include <time.h>
#include <string.h>

#include "UDP.H"        // libreria delle funzioni UDP_...   richiamate
#include "registra.h"

#define TIMEOUT 1*CLOCKS_PER_SEC // 1 secondo

int main(int argc, char* argv[])
{
 unsigned char buffer[1024] = "INVIAMI UN NUMERO"; // stringa di richiesta
 unsigned long ip_address;
 unsigned short port_number;
 unsigned long start, now;
 unsigned int *num = (unsigned int*)buffer;

 char ip_dest[16];               // contiene l'indirizzo  IP del destinatario in forma puntata
unsigned int  numRicevuti=0;

 /*if (argc < 2)
  {
        printf("Deve essere fornito l'indirizzo IP del server!\r\n");
          system("pause"); //-->Aggiunto
        return -1;
  }
*/

 // inizializzazione socket con numero di porta UDP arbitrario
 if (UDP_init(0) < 0)
  {
        printf("Errore inizializzazione socket!\r\n");
        system("pause"); //-->Aggiunto
        return -1;
  }
```

```
// Richiesta da tastiera del IP del destinatario in forma puntata
 printf("Inserisci l'indirizzo IP del destinatario nella forma puntata: ");
 scanf("%s",&ip_dest);

 // Richiesta da tastiera del numero di porta del destinatario
 printf("Inserisci l'indirizzo numero di porta del destinatario (1050): ");
 scanf("%u",&port_number);

 // Inizializzazione indirizzo IP e n°porta del destinatario
 ip_address = IP_to_bin(ip_dest);                    // IP_to_bin(argv[1]);
                                                     //port_number = 23365;
 // Invia richiesta al server
 UDP_send( ip_address, port_number, buffer, strlen((char*)buffer));

 start = clock(); // tempo iniziale attesa
 now = clock(); // tempo attuale

 // ciclo di attesa risposta per al massimo un tempo pari a TIMEOUT
 while ((now - start) < TIMEOUT)
    {
          if (UDP_receive( &ip_address, &port_number, buffer, sizeof(buffer)) == sizeof(unsigned int))
                {
                  // ricevuto un datagram della dimensione corretta
                  // visualizzazione numero
                  printf("Ricevuto numero %u.\r\n", *num);
                  numRicevuti++;

                  //UDP_close();
                  // system("pause"); //-->Aggiunto
                  //return 0;
                 }

          // Invia di nuovo una richiesta al server
          UDP_send( ip_address, port_number, buffer, strlen((char*)buffer));
          now = clock(); // aggiornamento tempo attuale
    }

// trascorso un tempo pari a TIMEOUT senza ricevere risposta
 if (!numRicevuti)
   printf("Nessuna risposta ricevuta!\r\n");
 else
   printf("Ricevuti-->  %u numeri\r\n",numRicevuti );

 // Scrive nel file NumRicevuti.txt  il numero di valori ricevuti in un secondo
if (registra(numRicevuti)==-1)
   printf("La registrazione del numero dei valori ricevuti NON e' andata a buon fine \n");

 UDP_close();
 system("pause"); //-->Aggiunto
 return 0;
}
```

// UDPserver.c

```
// Quando riceve da un client la stringa "INVIAMI UN NUMERO" sulla porta 1050
// genera il numero progressivo successivo al numero attualmente
// memorizzato nel file number.txt e lo invia al client

#include <stdio.h>
#include <string.h>

#include "UDP.H"        // libreria delle funzioni UDP_... richiamate
#include "sequence.h"


int main(void)
{
 unsigned char buffer[1024]; // buffer per ricezione
 unsigned long ip_address;
 unsigned short port_number;
 unsigned int num; // variabile per numero da generare
 int n;

 // inizializzazione socket con porta UDP numero 1050    //23365
 if (UDP_init(1050) < 0)
  {
   printf("Errore inizializzazione socket!\r\n");
   return -1;
  }
 printf("Servizio attivo... \r\n");
 while (1)
    {
     if ((n = UDP_receive( &ip_address, &port_number, buffer, sizeof(buffer))) > 0)
       {
        // ricezione di un datagram e verifica del messaggio
        buffer[n] = '\0'; // terminatore di stringa
        if (strcmp((char*)buffer, "INVIAMI UN NUMERO") == 0)
          {
           // richiesta di generazione di un nuovo numero
           num = sequence();
           UDP_send(ip_address, port_number, (void*)&num, sizeof(unsigned int));
           printf("...inviato numero %u.\r\n", num);
          }
       }
    }
 UDP_close();
 return 0;
}
```

UDP.H

```
// UDP.H
// Prototipi e definizione delle funzioni

#include <WinSock2.h>

// --- Prototipi
// converte un indirizzo IPv4 nella notazione ASCII decimale usuale
// (esempio: “192.168.1.1”) in un valore numerico a 32 bit
unsigned long IP_to_bin(char ip_address[]);

// inizializza il socket impostando il numero di porta locale
// restituisce -1 in caso di errore, 0 in caso di successo
int UDP_init(unsigned short port_number);

// trasmette al socket identificato dall’indirizzo IP e dal numero
// di porta UDP un numero byte di dati contenuti nel buffer data
// restituisce -1 in caso di errore, altrimenti il numero di byte
// trasmessi
int UDP_send( unsigned long ip_address, unsigned short port_number, unsigned char data[], int byte);

// riceve un datagram nel buffer data di dimensione size
// restituisce -1 in caso di errore o di dati non disponibili,
// il numero di byte ricevuti in caso di successo; indirizzo IP e
// numero di porta UDP del mittente sono restituiti come parametri
int UDP_receive( unsigned long *ip_address, unsigned short *port_number, unsigned char data[], int size);

// chiude il socket
void UDP_close();
```

```
// Variabile globale
SOCKET socket_id; // identificatore del socket
```

```
// --- Definizione delle funzioni
unsigned long IP_to_bin(char ip_add[])
{
 unsigned long add;
 unsigned char byte;
 char *token;

 if ((token = strtok(ip_add,".")) == NULL)
   return 0x00000000;
 byte = (unsigned char)atoi(token);
 add = (unsigned long)byte * 16777216;
 if ((token = strtok(NULL,".")) == NULL)
   return 0x00000000;
  byte = (unsigned char)atoi(token);
  add += (unsigned long)byte * 65536;
  if ((token = strtok(NULL,".")) == NULL)
    return 0x00000000;
  byte = (unsigned char)atoi(token);
  add += (unsigned long)byte * 256;
  if ((token = strtok(NULL,".")) == NULL)
    return 0x00000000;
  byte = (unsigned char)atoi(token);
  add += (unsigned long)byte * 1;
  return add;
```

```
}

int UDP_init(unsigned short port_number)
{
 WSADATA wsaData;
 struct sockaddr_in add; // struttura per indirizzo
 unsigned long arg = 1;

 // inizializzazione WinSock (versione 2.2)
 if (WSAStartup(0x0202, &wsaData) != 0)
  return -1;
 // apertura socket UDP
 if ((socket_id = socket( AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == INVALID_SOCKET)
  {
   WSACleanup();
   return -1;
  }
 // costruzione struttura indirizzo
 memset(&add, 0, sizeof(add)); // azzeramento struttura
 add.sin_family = AF_INET; // dominio indirizzi IP
 add.sin_addr.s_addr = 0;  // indirizzo IP locale
 add.sin_port = htons(port_number);// numero di porta assegnato
                        // htons (host to network-short) converte il numero
                        // di porta in un short integer (16 bit)secondo
                        // il formato Big Endian (formato neutro della rete).
                        // Questo viene fatto per ragioni di portabilità

 if (bind( socket_id, (struct sockaddr *)&add, sizeof(add))== SOCKET_ERROR)
  {
   closesocket(socket_id);
   WSACleanup();
   return -1;
  }
 // impostazione del socket come non bloccante
 if (ioctlsocket(socket_id, FIONBIO, &arg) == SOCKET_ERROR)
  {
   closesocket(socket_id);
   WSACleanup();
   return -1;
  }
 return 0;
}
```

```
int UDP_send( unsigned long ip_address, unsigned short port_number, unsigned char data[], int byte)
{
 struct sockaddr_in add; // struttura per indirizzo di destinazione
 int n;

 // costruzione struttura indirizzo
 memset(&add, 0, sizeof(add)); // azzeramento struttura
 add.sin_family = AF_INET; // dominio indirizzi IP
 add.sin_port = htons(port_number); // numero di porta UDP
 add.sin_addr.s_addr = htonl(ip_address); // indirizzo IP
                        // htonl (host to network-long) converte il numero
                        // di porta in un long integer (32 bit)secondo
```

```
                                    // il formato Big Endian (formato neutro della rete).

// trasmissione datagram
 if ((n = sendto (socket_id, (void *)data, byte, 0, (struct sockaddr*)&add, sizeof(add))) < 0)
   return -1;
 return n;
}
```

```
int UDP_receive( unsigned long *ip_address, unsigned short *port_number, unsigned char data[], int size)
{
 struct sockaddr_in add; // struttura per indirizzo mittente
 size_t dim = sizeof(add);
 int n;

 // ricezione dati (non bloccante)
 if ((n = recvfrom (socket_id, (void *)data, size, 0, (struct sockaddr*)&add, &dim)) <= 0)
   return -1;
 // estrazione indirizzo IP e numero di porta UDP
 *ip_address = (unsigned long)(ntohl(add.sin_addr.s_addr));
 *port_number = (unsigned short)(ntohs(add.sin_port));
 return n;
}
```

```
void UDP_close()
{
 // chiusura socket
 closesocket(socket_id);
 // terminazione WinSock
 WSACleanup();
 return;
}
```

sequence.h

```
#include <stdio.h>
// restituisce il nuovo numero sequenziale,
// il valore 0 in caso di errore

unsigned int sequence(void)
{
 FILE* file;
 unsigned int n;

 // apertura del file
 file = fopen("number.txt", "r+");

 if (file == NULL)
   return 0;

 // lettura da file dell'ultimo valore utilizzato
 if (fscanf(file, "%u", &n) != 1)
   {
    fclose(file);
    return 0;
   }

 // generazione nuovo valore da utilizzare
 n++;

 // salvataggio su file del nuovo valore
 fseek(file, 0, SEEK_SET);
 fprintf(file, "%u", n);
 fclose(file);

 return n;
}
```

---

registra.h

```
#include <stdio.h>

// Restituisce il valore -1 in caso di errore
// altrimenti appende a fine file il valore ricevuto come parametro
int registra(unsigned int n)
{
 FILE* fileout;

 // apertura del file in append
 fileout = fopen("NumRicevuti.txt", "a");

 if (fileout == NULL)  // in caso di errore
    return -1;
 else
    // scrive nel file il numero ricevuto
    fprintf(fileout, "%u\n", n);

 // chiusura file
 fclose(fileout);
 return 0;
 }
```