Laboratorio di Programmazione in Rete

a.a. 2005/2006

http://www.di.uniba.it/~lisi/courses/prog-rete/prog-rete0506.htm

dott.ssa Francesca A. Lisi
lisi@di.uniba.it

Orario di ricevimento: mercoledì ore 10-12

N.B. Il presente materiale didattico è stato
prodotto da: dott.ssa V. **Carofiglio**
rielaborato da: dott.ssa F.A. **Lisi**

dib

Sommario della lezione di oggi

- Mapping UNIX-Windows
- Costruzione di messaggi (cap.3)
- Interazione client-server in C basata su socket UDP per Windows (cap.4)

Mapping UNIX-Windows: Inclusione di file header

windows

```
#include <stdio.h>
#include <stdlib.h>

#include <winsock.h>
```

Include tutte le definizioni ed i prototipi

unix

```
#include <stdio.h>
#include <stdlib.h>

#include < sys/socket.h >

#include <arpa/inet.h>
```

Per socket(), connect(), send(), recev()

Per sockaddrin

Mapping UNIX-Windows: Setup dell'applicazione

Il codice è identico a meno del codice per inizializzare l'applicazione

windows

```
WSADATA wsaData;
Int iResult = WSAStartup(MAKEWORD(2 ,2), &wsaData);
If (iResult != NO_ERROR)
   printf("error at WSASturtup\n");
```

Codice di inizializzazione della libreria winsock

comunicazione

Il codice è' identico

Mapping UNIX-Windows: Chiusura dell'applicazione

windows

```
Closesocket(m_socket);
WSAcleanup();

Exit(0)
```

De-allocazione delle risorse usate da winsock

unix

```
Close(m_socket);


Exit(0)
```

Mapping UNIX-Windows: Gestione dell'errore

windows

```
#include<stdio.h>
#include<stdlib.h>
#include<winsock.h>

void ErrorManagement(char* errorMessage) {
   printf( "Error at socket(): %ld\n", WSAGetLastError() );
    WSACleanup();
    exit(1);
}
```

unix

```
#include<stdio.h>
#include<stdlib.h>

void ErrorManagement(char* errorMessage) {
   perror(errorMessage);
   exit(1);
}
```

Costruzione di messaggi:
Codifica dell'informazione

Client e Server devono accordarsi su come l'informazione debba essere codificata

possibilità 1:

Stringhe di cifre decimali: *sequenze di byte i cui valori sono determinati in accordo con una qualche codifica (es: ASCII)*

Per rappresentare 17.998.720 e 47.034.615

49	55	57	56	50	55	50	48	32	52	55	48	51	52	54	49	53	32
'1'	'7'	'9'	'9'	'8'	'7'	'2'	'0'	' '	'4'	'7'	'0'	'3'	'4'	'6'	'1'	'5'	' '

A

B

```
sprintf(sendbuf,"%d%d ". A, B);
send( m_socket, sendbuf, strlen(sendbuf), 0 );
```

Buffer contenente i dati da trasmettere

Costruzione di messaggi:
Codifica dell'informazione (cont.)

Possibilità 1: Stringhe di cifre decimali

```
sprintf(sendbuf,"%d%d ", A, B);
send( m_socket, sendbuf, strlen(sendbuf), 0 );
```

- ***sendbuf*** *deve essere abbastanza grande (numeri più grandi, segno .....)*
- *un comune errore:*

```
#define BUFSIZE 132
.
.
.
Char sendbuf[BUFSIZE]
.
.
sprintf(sendbuf,"%d%d ". A, B)
send( m_socket, sendbuf, BUFSIZE, 0 );
```

- ***Il ricevente riceve byte extra "spazzatura"***

Costruzione di messaggi:
Codifica dell'informazione (cont.)

Possibilità 1: Stringhe di cifre decimali

```
sprintf(sendbuf,"%d%d ",      B);
send( m_socket, sendbuf      f), 0 )
```

- ***sendbuf** deve ...o .....)*
- *un comune er... ...ne extra "spazzatura"*

```
#define BUFSIZE 12
.
.
.
Char sendbuf[BUF
.
.
sprintf(sendbuf,"%d%d      B)
send( m_socket, sendbu     BUFSIZE, 0 );
```

SCONVENIENTE!!

Costruzione di messaggi:
Codifica dell'informazione (cont.)

Client e Server devono accordarsi su come l'informazione debba essere codificata

possibilità 2:

Usare una Struct

Per rappresentare 17.998.720 e 47.034.615

49	55	57	56	50	55	50	48	32	52	55	48	51	52	54	49	53	32
'1'	'7'	'9'	'9'	'8'	'7'	'2'	'0'	' '	'4'	'7'	'0'	'3'	'4'	'6'	'1'	'5'	' '

A B

```
Struct  {
            Int a
            Int b;
} msgStruct;
..
struct msgStruct msg;
```

```
msg.a=A;
msg.b=B;
send( m_socket, &msg, sizeof(msgStruct), 0 );
```

Costruzione di messaggi:
Codifica dell'informazione (cont.)

Client e Server devono accordarsi su come l'informazione debba essere codificata

possibilità 2:

Usare una Struct

Per rappresentare **17.998.720** ***e*** **47.034.615**

49	55	57	56	50	55	50	48	32	52	55	48	51	52	54	49	53	32
'1'	'7'	'9'	'9'	'8'	'7'	'2'	'0'	' '	'4'	'7'	'0'	'3'	'4'	'6'	'1'	'5'	' '

A B

```
Struct  {
            Int a;
            Int b;
} msgStruct;
..
```

```
send( m_socket, &A, sizeof(A), 0 );
send( m_socket, &B, sizeof(A), 0 );
```

Se usiamo TCP non è necessario copiare A e B nella struct

Costruzione di messaggi:
Codifica dell'informazione (cont.)

Client e Server devono accordarsi su come l'informazione debba essere codificata

po ilità 2

Usa

49	55	57	5		4	49	53	32	
'1'					'6'	'1'	'5'	' '	

B

La sequenza di byte inviati come risultato della codifica dipende dall'architettura della macchina su cui il programma viene eseguito: Ovvero….

```
Struct  {
        Int Int1;
        Int Int2;
} msgStruct;
..
```

```
sizeof(A), 0 );
( ket, &B, sizeof(A), 0 );
```

Se usiamo TCP non è necessario copiare A e B nella struct

Costruzione di messaggi:
Big-Endian vs. Little-Endian

In base a come assumono che siano accodati i byte delle rappresentazioni multi-byte.

1	18	163	128	2	205	176	247

Big-Endian: Il byte più significativo ha lo stesso indirizzo dell'intera parola

128	163	18	1	247	176	205	2

Little-Endian: Il byte all'interno dei due interi occorre in ordine inverso rispetto al Big-Endian

Il punto:

Quando viene trasferita (inviata/ricevuta) una struttura tramite socket i byte all'interno della struttura vengono inviati (ricevuti e memorizzati) in ordine ascendente (dal meno significativo al più significativo)

Costruzione di messaggi:
Big-Endian vs. Little-Endian (cont.)

Che succede se due programmi che comunicano hanno un ordinamento di byte opposto?

```
Struct  {
            Int a;
            Int b;
} msgStruct;
..
```

1	18	163	128	2	205	176	247

-2.136.796.671

128	163	18	1	247	176	205	2

-139.408.126

Big-Endian: Ordinamento Standard (per convenzione)

Costruzione di messaggi:

Alcune funzioni utili

- **htons()**
  - converte uno short integer (2 byte) dalla rappresentazione della piattaforma a quella della rete
- **ntohs()**
  - esegue la conversione inversa
- **htoni()**
  - per interi, da piattaforma a rete
- **ntohi()**
  - esegue la conversione inversa
- **htonl()**
  - converte long integer (4 byte) dalla piattaforma alla rete
- **ntohl()**
  - esegue la conversione inversa

Costruzione di messaggi:
Codifica dell'informazione

Client e Server devono accordarsi su come l'informazione debba essere codificata

possibilità 2:

Usare una Struct

Per rappresentare 17.998.720 e 47.034.615

49	55	57	56	50	55	50	48	32	52	55	48	51	52	54	49	53	32
'1'	'7'	'9'	'9'	'8'	'7'	'2'	'0'	'	'	'4'	'7'	'0'	'3'	'4'	'6'	'1'	'5'	'	'

A B

```
Struct  {
            Int a;
            Int b;
} msgStruct;
..
```

```
struct msgStruct msg;
....
msg.a=htonl(A)
msg.b=htonl(A)
send( m_socket, &msg, sizeof(msgStruct), 0 );
```

Costruzione di messaggi:
Framing & Parsing

Client e Server devono accordarsi su come l'informazione debba essere codificata

Formattare il messaggio in modo che il ricevente possa "parserizzarlo"

(Es: individuare inizio e fine del messaggio, limiti tra i campi del messaggio, ecc. …..)

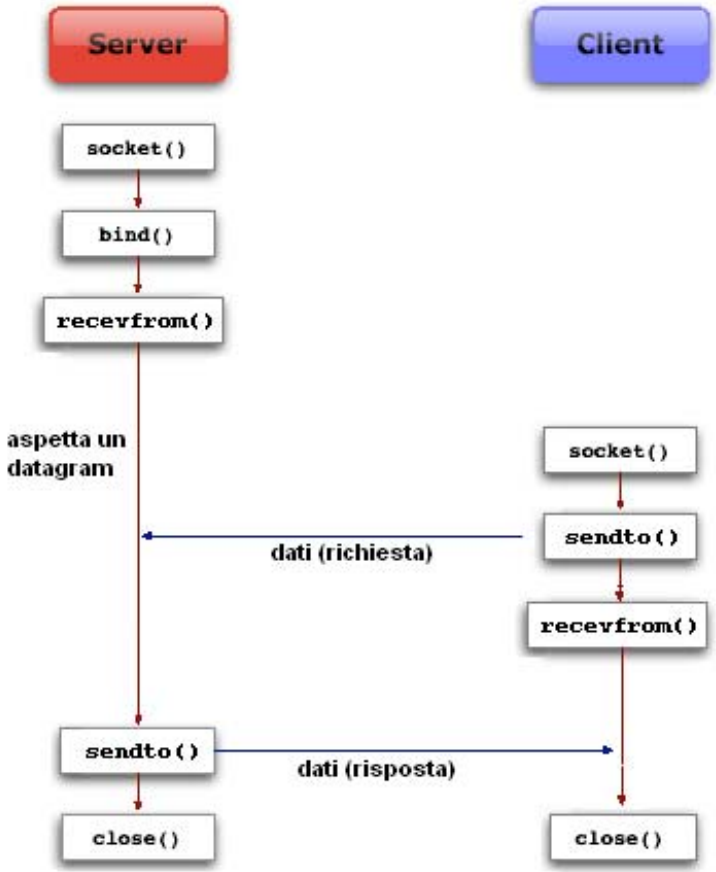
Caso1: il messaggio ha dimensione fissata, nota a priori

Viene ricevuto il numero di byte atteso in un buffer

Caso2: il messaggio contiene delimitatori

Implementare una procedura di "parsing"

Interazione UDP Client/Server



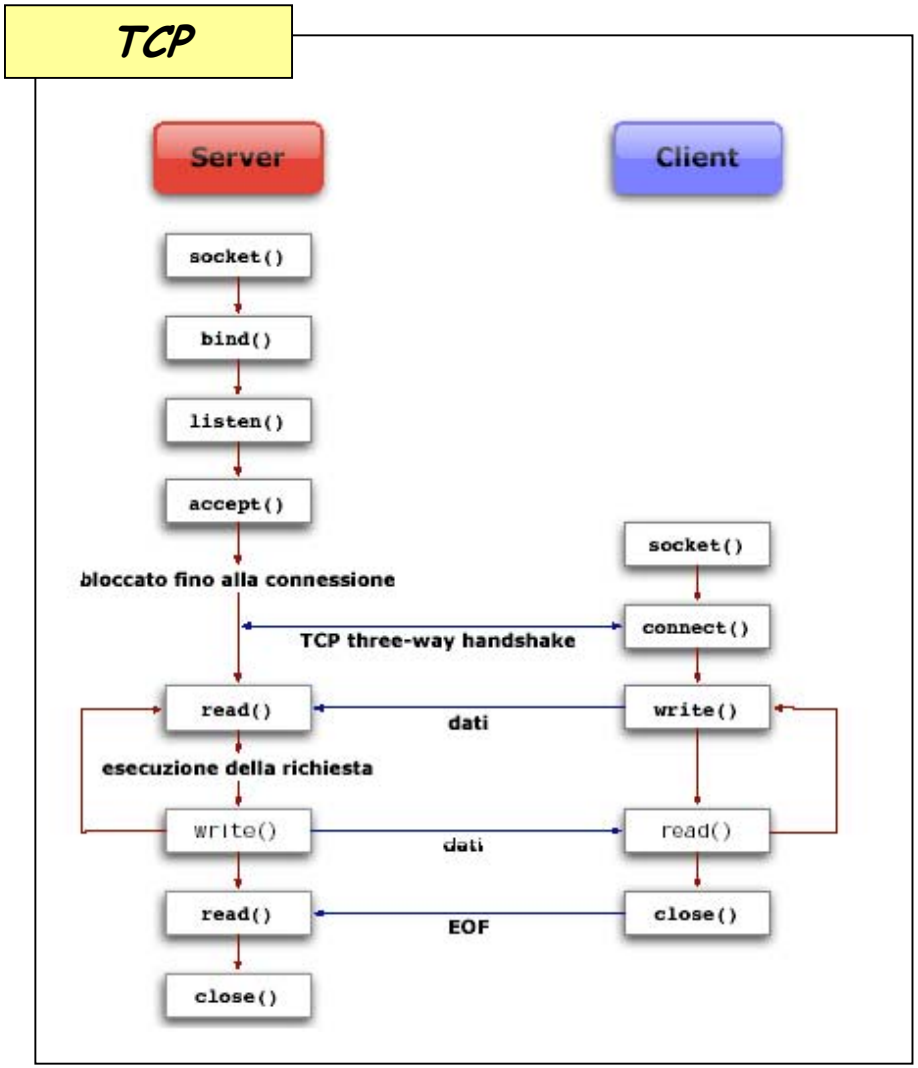
Server

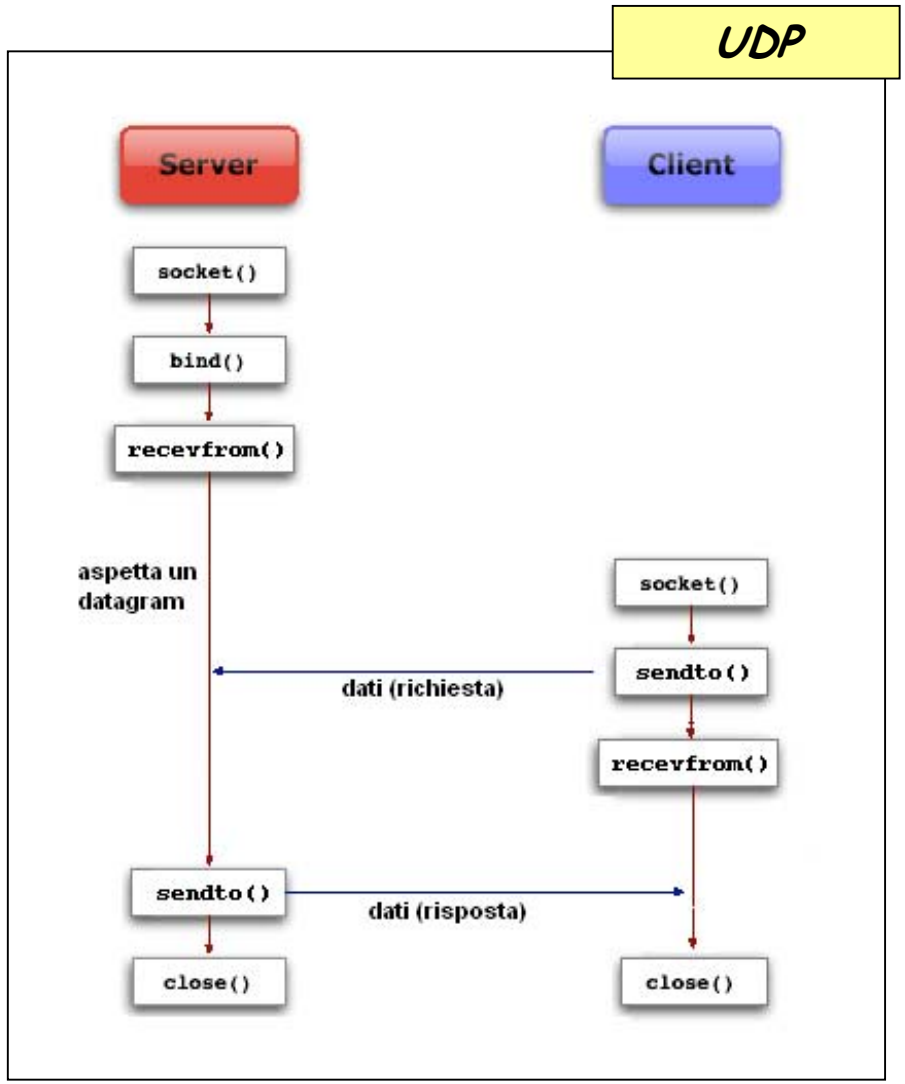
1. (Inizializzare una WSA)
2. Creare una socket
3. Assegnare un local address alla socket
4. Iterativamente:
   a. Inviare e ricevere dati
   b. Chiudere la connessione

Client

1. (Inizializzare una WSA)
2. Creare una Socket
3. Inviare e ricevere dati
4. Chiudere la connessione

TCP vs. UDP


TCP
Server
Client
socket()
bind()
listen()
accept()
bloccato fino alla connessione
socket()
connect()
TCP three-way handshake
read()
dati
write()
esecuzione della richiesta
write()
dati
read()
read()
EOF
close()
close()


UDP
Server
Client
socket()
bind()
recevfrom()
aspetta un datagram
socket()
sendto()
dati (richiesta)
recevfrom()
sendto()
dati (risposta)
close()
close()

Funzione recevfrom()

Riceve un datagram e memorizza l'indirizzo da cui i dati sono stati inviati

```
int recvfrom( SOCKET s, char* buf, int len, int flags, struct sockaddr* from, int* fromlen);
```

- Descrittore di una socket (eventualmente) connessa
- Buffer contenente i dati in ingresso
- Lunghezza dei dati in buf, in byte
- Indicatore che specifica il modo in cui la chiamata è fatta
- Puntatore opzionale ad una struttura che contiene l'indirizzo della socket target
- Lunghezza dei dati in to, in byte

Il flag può essere usato per influenzare il comportamento della funzione

La funzione è normalmente usata per socket non orientate alla connessione.
L'indirizzo locale della socket deve essere noto
Per applicazioni Server, questo è fatto esplicitamente con la funzione bind()
Il binding esplicito è scoraggiato per applicazioni client (in tal caso la funxzione effettua un binding implicito

Funzione recevfrom(): esempio d'uso

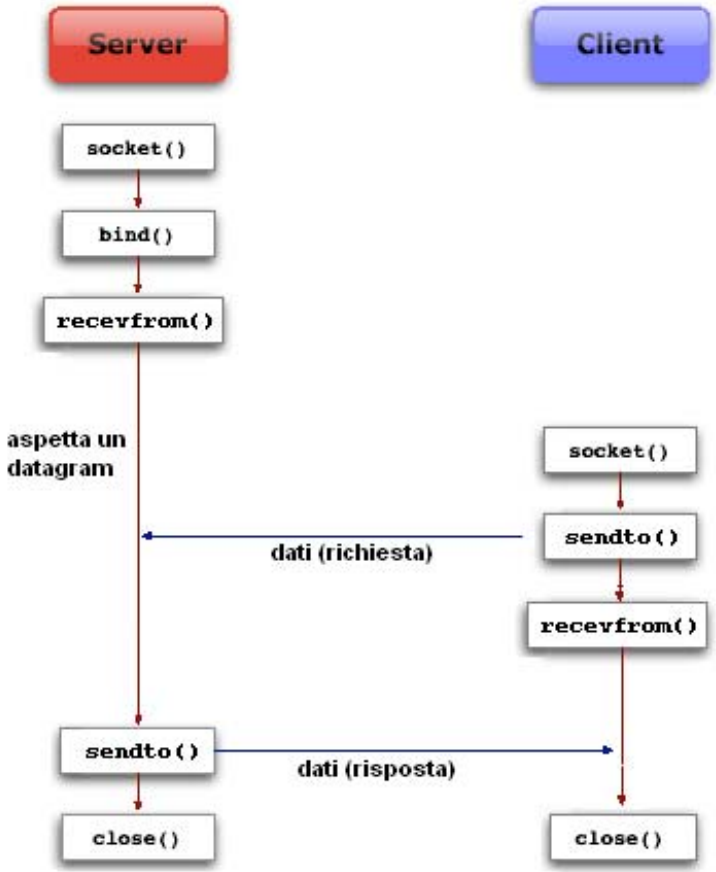
```
#include <stdio.h>
#include "winsock2.h"

void main() {
  WSADATA wsaData;
  SOCKET RecvSocket;
  sockaddr_in RecvAddr;
  int Port = 27015;
  char RecvBuf[1024];
  int  BufLen = 1024;
  sockaddr_in SenderAddr;
  int SenderAddrSize = sizeof(SenderAddr);
  //-----------------------------------------------
  // Initialize Winsock
  WSAStartup(MAKEWORD(2,2), &wsaData);
  //-----------------------------------------------
  // Create a receiver socket to receive datagrams
  RecvSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
  //-----------------------------------------------
```

Funzione recevfrom( ): esempio d'uso(cont.)

```
 // Bind the socket to any address and the specified port.
  RecvAddr.sin_family = AF_INET;
  RecvAddr.sin_port = htons(Port);
  RecvAddr.sin_addr.s_addr = htonl(INADDR_ANY);
  bind(RecvSocket, (SOCKADDR *) &RecvAddr, sizeof(RecvAddr));
  //---------------------------------------------
  // Call the recvfrom function to receive datagrams
  // on the bound socket.
  printf("Receiving datagrams...\n");
  recvfrom(RecvSocket,  RecvBuf, BufLen,  0, (SOCKADDR *)&SenderAddr,   &SenderAddrSize);
  //---------------------------------------------
  // Close the socket when finished receiving datagrams
  printf("Finished receiving. Closing socket.\n");
  closesocket(RecvSocket);
  //---------------------------------------------
  // Clean up and exit.
  printf("Exiting.\n");
  WSACleanup();
  return;
}
```

Interazione UDP Client/Server



Server

1. (Inizializzare una WSA)
2. Creare una socket
3. Assegnare un local address alla socket
4. Iterativamente:
   a. Inviare e ricevere dati
   b. Chiudere la connessione

Client

1. (Inizializzare una WSA)
2. Creare una Socket
3. Inviare e ricevere dati
4. Chiudere la connessione

Funzione sendto()

Invia dati ad una specifica destinazione

int sendto(SOCKET *s*, const char* *buf*, int *len*, int *flags*, const struct sockaddr* *to*, int *tolen*);

- *s*: Descrittore di una socket (eventualmente) connessa
- *buf*: Buffer contenente I dati da trasmettere
- *len*: Lunghezza dei dati in buf, in byte
- *flags*: Indicatore che specifica il modo in cui la chiamata è fatta
- *to*: Puntatore opzionale ad una struttura che contiene l'indirizzo della socket target
- *tolen*: Lunghezza dei dati in to, in byte

Il flag può essere usato per influenzare il comportamento della funzione

La funzione è normalmente usata per socket non orientate alla connessione per inviare datagram ad una specifica socket identificata dai parametri. I parametri to e tolen vengono ignorati in caso di socket orientate alla connessione e la funzione diventa equivalente ad una send()

Funzione sendto(): valori di ritorno

Invia dati ad una specifica destinazione

int sendto(SOCKET *s*, const char* *buf*, int *len*, int *flags*, const struct sockaddr* *to*, int *tolen*);

- *s*: Descrittore di una socket (eventualmente) connessa
- *buf*: Buffer contenente I dati da trasmettere
- *len*: Lunghezza dei dati in buf, in byte
- *flags*: Indicatore che specifica il modo in cui la chiamata è fatta
- *to*: Puntatore opzionale ad una struttura che contiene l'indirizzo della socket target
- *tolen*: Lunghezza dei dati in to, in byte

La funzione restituisce il numero di byte trasmessi in caso di successo.

Un codice di errore, altrimenti

Funzione sendto(): un esempio d'uso

```
#include <stdio.h>
#include "winsock2.h"
void main() {

  WSADATA wsaData;
  SOCKET SendSocket;
  sockaddr_in RecvAddr;
  int Port = 27015;
  char SendBuf[1024];
  int BufLen = 1024;
  //-----------------------------------------
  // Initialize Winsock
  WSAStartup(MAKEWORD(2,2), &wsaData);
  //-----------------------------------------
  // Create a socket for sending data
  SendSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

Funzione sendto(): un esempio d'uso (cont.)

```
//-------------------------------------------
  // Set up the RecvAddr structure with the IP address of
  // the receiver (in this example case "123.456.789.1")
  // and the specified port number.
  RecvAddr.sin_family = AF_INET;
  RecvAddr.sin_port = htons(Port);
  RecvAddr.sin_addr.s_addr = inet_addr("123.456.789.1");
  //-------------------------------------------
  // Send a datagram to the receiver
  printf("Sending a datagram to the receiver...\n");
  sendto(SendSocket,  SendBuf,   BufLen,   0,   (SOCKADDR *) &RecvAddr,    sizeof(RecvAddr));
  //-------------------------------------------
  // When the application is finished sending, close the socket.
  printf("Finished sending. Closing socket.\n");
  closesocket(SendSocket);
  //-------------------------------------------
  // Clean up and quit.
  printf("Exiting.\n");
  WSACleanup();
  return;
}
```

Pratica in laboratorio

- Ricomporre l'esempio di interazione UDP riportato in queste slide
- Fare il porting su Windows degli esempi riportati nel cap. 4 del testo di Donahoo & Calvert
- Svolgere esercizi a pag. 42 del testo