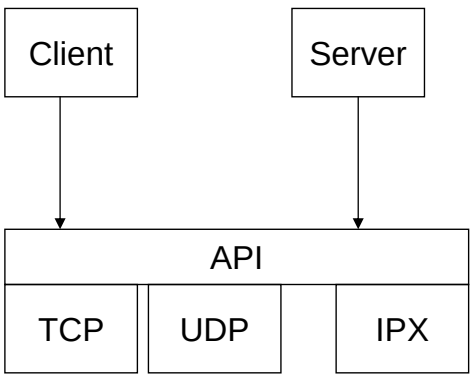
L’interfaccia socket

Application Programming Interface: API
Socket API
Procedure base
Altre procedure
Ordinamento dei byte
Interazione client-server orientata alla connessione
Interazione client-server senza connessione
Testing di applicazioni client-server

Application Programming Interface: API

- API: servizi invocabili dalle applicazioni, interfacce, astrazioni fornite dal SO all’applicazione
- Un API fornisce:
  - un insieme di procedure che l’applicazione puo’ invocare e gli argomenti che ogni procedura si aspetta
- API di comunicazione
  - sockets (de facto standard)
    - BSD sockets (Unix)
    - winsock (wintel)
    - java network
  - transport layer interface (TLI)
  - Novell Netware API



Socket API

- Sviluppati come parte del BSD Unix ma disponibili su altri SO come socket library
- Il socket e' visto come un dispositivo creato e posseduto dall'applicazione e controllato dal SO dell'host locale,
- Un processo applicativo puo' sia spedire che ricevere messaggi a/da un altro processo applicativo mediante il proprio socket
- La comunicazione via socket usa il modello di I/O di Unix
  - open-read-write-close
  - file descriptor = (pathname, flag)
- Quando l'applicazione crea un socket, all'applicazione e' assegnato un descrittore (int) per riferirsi al socket
- Due tipi di servizio di trasporto via socket API:
  - senza connessione: datagram, inaffidabile (UDP)
  - orientato alla connessione: stream, affidabile (TCP)

Creazione di un socket

descriptor = socket(family, type, protocol)

- Restituisce un descrittore di socket che e' usato in tutte le chiamate successive

```
int socket ( int family, int service, int protocol)
```

- family specifica la famiglia del protocollo di trasporto
  - TCP/IP Internet: AF_INET
  - Xerox NS: AF_NS
  - intra-host UNIX: AF_UNIX
  - DEC DNA: AF_DECNET
- type/service specifica lo specifico servizio di trasporto
  - datagram: SOCK_DGRAM (UDP protocol in AF_INET)
  - reliable, in-order: SOCK_STREAM (TCP protocol in AF_INET)
  - raw socket: SOCK_RAW (direct access to network layer)
- protocol specifica ulteriormente il raw socket altrimenti e' 0

```
#include <sys/types.h>
#include<sys/ socket.h>
 int sockfd;
 if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)   /* handle error */
```

Assegnazione di un indirizzo di rete al socket

- Ogni socket deve essere associato a un numero di porta a 16-bit, locale e unico rispetto all'host
- Necessita' di associare il socket con un indirizzo di rete globalmente unico (host address e port)
  - Il sistema operativo sa che i messaggi in arrivo a questo indirizzo devono essere consegnati al socket
  - un indirizzo di ritorno per i messaggi in partenza

Numeri di porta

- 1 - 255: riservati a servizi standard
  - 21: ftp
  - 23: telnet
  - 25: SMTP
  - 80:http daemon
- 1 - 1023: disponibili solo a utenti privilegiati
- 1024 - 4999: per processi automaticamente assegnati
- 5000 - : solo per processi utente

Formato dell'indirizzo di rete

- Formato generico

```
struct sockaddr {
        u_char    sa_len;        /* total length of the address */
        u_char    sa_family;     /* family of the address */
        char      sa_data[14];   /* the address itself */
};
```

- Formato della famiglia TCP/IP

```
struct sockaddr_in {                        /* INET socket addr info */
        u_char    sin_len;                  /* total length of the address */
        u_char    sin_family;               /* family of the address; set me to AF_INET */
        u_short   sin_port;                 /* protocol port number; 16 bit number, nbo */
        struct    in_addr sin_addr;         /* IP address of host */
        char      sin_zero[8];              /* not used (set to zero) */
};
struct in_addr {
        u_long    s_addr;                   /* 32 bit host address, nbo */
};
```

Assegnazione di un indirizzo di rete al socket: bind()

bind (socket, localaddr, addrlen)

- usato da un server per fornire un indirizzo IP e un numero di porta dove attendere i contatti
- costante simbolica INADDR_ANY per specificare solo un numero di porta

```
int bind (int sockfd, struct sockaddr *myaddr, int addresslen)
```

- sockfd e' la variabile a cui e' assegnato il valore di ritorno di socket()
- *myaddr: indirizzo della struttura sockaddr_in che contiene le informazioni sull'indirizzo locale
- addresslen e' la dimensione della struttura dell'indirizzo

Disponibilita' ad accettare connessioni: listen()

listen (socket, queuesize)

- usato dai server in interazioni orientate alla connessione
- Fa sapere al sistema operativo che il server e' in attesa di richieste di connessione dai client (modo passivo)
- Una coda separata per ogni socket

```
int listen ( int sockfd, int maxwaiting)
```

- sockfd e' la variabile a cui e' assegnato il valore di ritorno di socket().
  Il processo accetta le connessioni in arrivo su questo socket id
- maxwaiting:numero massimo di richieste di connessione che possono essere accodate in attesa che il server faccia accept(). Valore tipico 5

Connessione del client al server: connect()

connect (socket, destaddr, addrlen)

- In un servizio orientato alla connessione, il client usa connect() per richiedere la connessione al server e la comunicazione via socket
- Il protocollo sottostante di trasporto (es. TCP) realizza la connessione
- connect() restituisce il controllo quando il server esplicitamente accetta la connessione, mediante accept(), o per timeout (nessuna risposta dal server)

```
int connect (int sockfd, struct sockaddr *toaddrptr,
   int addresslen)
```

- sockfd e' la variabile a cui e' assegnato il valore di ritorno di socket(). Il processo accetta le connessioni in arrivo su questo socket id
- *toaddrptr e' l'indirizzo della struttura sockaddr_in che contiene le informazioni sull'indirizzo del server
- addresslen e' la dimensione della struttura dell'indirizzo

Connessione del server al client: accept()

newsock = accept(socket, client_address, client_addresslen)

- Eseguito dal server, dopo listen() in un servizio orientato alla connessione
- Il server fara' accept() delle richieste di connessione attraverso il socket specificato, e restituira' un nuovo socket creato per comunicare con il client
- Se una richiesta di connessione e' gia' in coda, accept() restituisce il controllo immediatamente altrimenti blocca il server fino a che un client effettua una richiesta di connessione

```
int accept (int sockfd, struct sockaddr *fromaddrptr,
   int *addresslen)
```

- sockfd e' la variabile a cui e' assegnato il valore di ritorno di socket()
- *fromaddrptr e' l'indirizzo della struttura sockaddr_in che contiene le informazioni sull'indirizzo del socket che ha spedito questi dati. E' un valore di ritorno
- addresslen e' la dimensione della struttura dell'indirizzo. E' un valore di ritorno

Chiusura di un socket

close (socket)

- Informa il SO di terminare l'uso del socket
- Il descrittore e' rilasciato
- Se il socket e' usato in un servizio orientato alla connessione, close() chiude la connessione prima di chiudere il socket

Spedizione e ricezione di dati via socket

Servizi orientati alla connessione

- Spedizione
  - send (socket, data, length, flags)
  - write (socket_descriptor, buffer, length)
- Ricezione
  - recv (socket, data, length, flags)
  - read (socket_descriptor, buffer, length)

Servizi non orientati alla connessione

- Spedizione
  - sendto (socket, data, length, flags, destaddr, addrlen)
- Ricezione
  - recvfrom (socket, data, length, flags, fromaddr, addrlen)

Spedizione via socket: sendto()

int sendto (int sockfd, char *buff, int bufflen, int flags, struct sockaddr *toaddrptr, int addresslen)

- sockfd e' la variabile a cui e' assegnato il valore di ritorno di socket()
- *buff e' un insieme di locazioni di memoria consecutive che contengono il messaggio da spedire
- bufflen e' il numero di bytes da spedire
- flags puo' essere usato per specificare opzioni. Spesso e' 0
- *toaddrptr e' l'indirizzo della struttura sockaddr_in che contiene le informazioni sull'indirizzo del destinatario
- addresslen e' la dimensione della struttura dell'indirizzo
- Codice di ritorno OK non implica che i dati sono correttamente ricevuti ma solo che sono correttamente spediti

Ricezione via socket: recvfrom()

int recvfrom (int sockfd, char *buff, int bufflen, int flags, struct sockaddr *fromaddrptr, int *addresslen)

- sockfd e' la variabile a cui e' assegnato il valore di ritorno di socket()
- *buff e' un insieme di locazioni di memoria consecutive in cui I dati ricevuti possono essere scritti
- bufflen e' il numero di bytes da leggere
- flags puo' essere usato per specificare opzioni. Spesso e' 0
- *fromaddrptr e' l'indirizzo della struttura sockaddr_in che contiene le informazioni sull'indirizzo del socket che ha spedito i dati. E' un valore restituito
- addresslen e' la dimensione della struttura dell'indirizzo. E' un valore di ritorno
- recvfrom() restituisce il numero di bytes realmente ricevuti

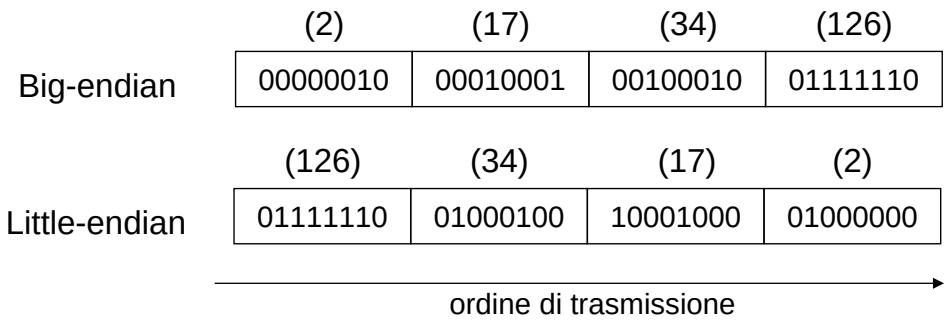
Altre procedure

- getpeername()
  - restituisce l'indirizzo completo del client che ha iniziato una connessione
- gethostname()
  - restituisce informazioni sull'host locale
- getprotobyname()
  - converte il nome di un protocollo nella forma binaria usata da socket()
- gethostbyname()
  - converte il nome DNS in un indirizzo IP (notazione decimale a punti)
- inet_addr()
  - converte una stringa in notazione decimale a punti in un indirizzo a 32 bit
- gethostbyaddr()
  - converte un indirizzo IP in un nome DNS

Ordinamento dei byte

- Problema della rappresentazione dei dati:
  - non tutti gli host memorizzano gli interi nello stesso modo
- Network order o rappresentazione big endian (Motorola 680x0)
  - il bit di ordine piu' alto e' trasmesso per primo
- Host order o rappresentazione little endian (Intel 80x86)
  - il bit di ordine piu' alto e' trasmesso per ultimo

Esempio: numero intero 34 677 374 = $2*2^{24}+17*2^{16}+34*2^{8}+126*2^{0}$

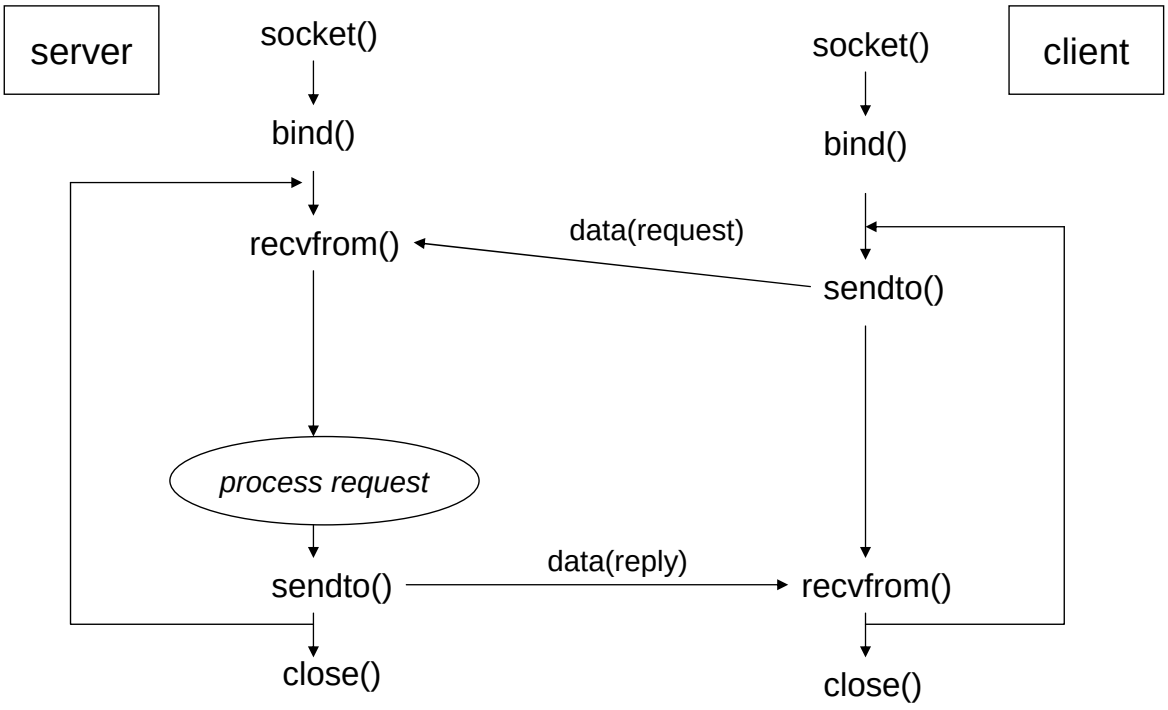


Procedure di ordinamento dei byte

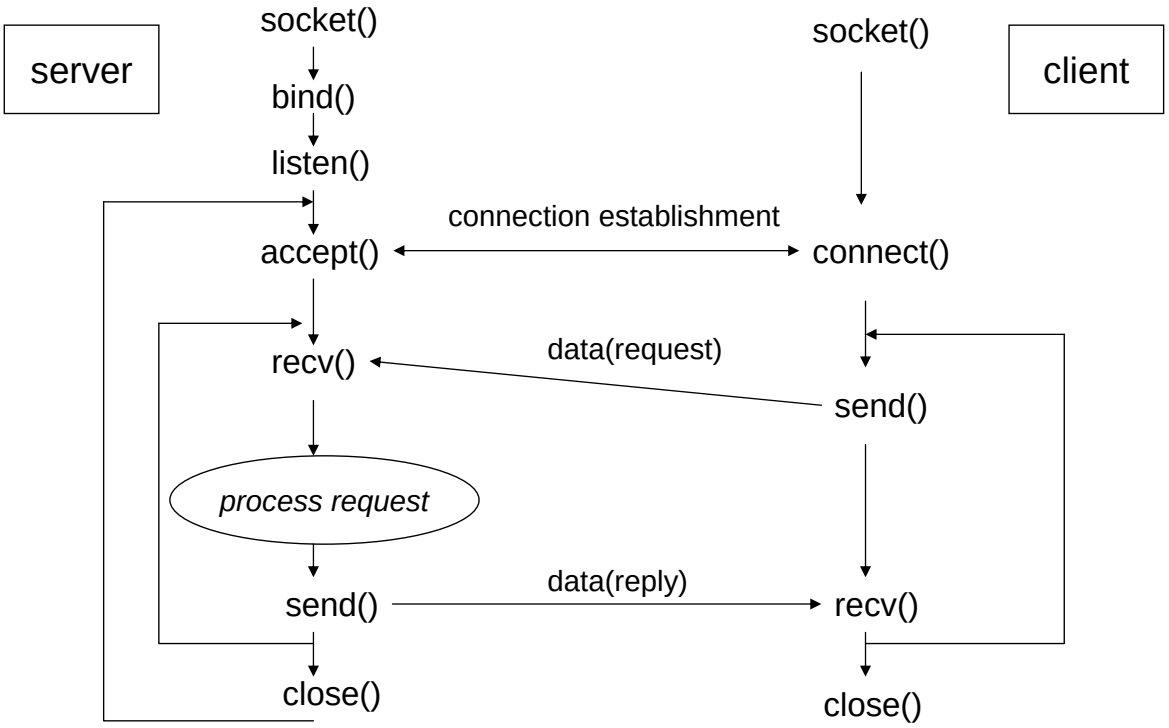
- long integer (32 bit): per gli indirizzi IP
  - u_long htonl (u_long hostlong); /* converte da hbo a nbo */
  - u_long ntohl (u_long netlong); /* converte da nbo a hbo */
- short integer (16 bit): per i numeri di porta
  - u_short htons (u_short hostshort); /* converte da hbo a nbo*/
  - u_short ntohs (u_short netshort); /* converte da nbo a hbo */

Indirizzi di rete restituiti da procedure (es. gethostbyname) sono gia' in formato network order e non richiedono conversione prima di essere spediti

Interazione client/server senza connessione



Interazione client/server orientata alla connessione



Testing di applicazioni che usano sockets

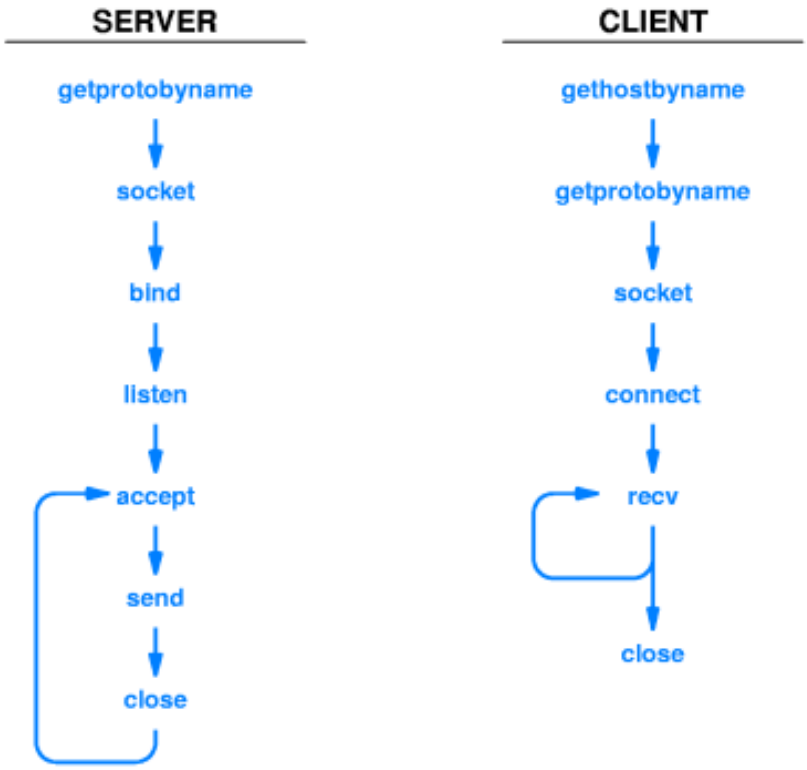
- Testing del server
  - uso di telnet per contattare il server
    - telnet server_address port_number
  - casi di test da standard input
  - comportamento osservato su standard output
- Testing del client con server preesistente
  - installazione del server in locale
  - uso di *localhost* come indirizzo di rete

Esempio di applicazione client/server

- Servizio orientato alla connessione
- Server sequenziale (un solo flusso di controllo: i client devono aspettare il soddisfacimento delle richieste non concluse)
- Il server restituisce come messaggio il numero di richieste che hanno avuto accesso al servizio:

  `This server has been contacted … times`

Esempio di applicazione client/server



Codice client (1)

```
/* client.c - code for example client program that uses TCP */

#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#endif

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

#define PROTOPORT       5193            /* default protocol port number */

extern  int             errno;
char    localhost[] =   "localhost";    /* default host name            */
/*----------------------------------------------------------------------
 * Program:   client
 *
 * Purpose:   allocate a socket, connect to a server, and print all output
```

Codice client (2)

```
 * Syntax:    client [ host [port] ]
 *
 *               host  - name of a computer on which server is executing
 *               port  - protocol port number server is using
 *
 * Note:      Both arguments are optional.  If no host name is specified,
 *            the client uses "localhost"; if no protocol port is
 *            specified, the client uses the default given by PROTOPORT.
 *
 *----------------------------------------------------------------------
 */
main(argc, argv)
int     argc;
char    *argv[];
{
        struct  hostent  *ptrh;  /* pointer to a host table entry       */
        struct  protoent *ptrp;  /* pointer to a protocol table entry   */
        struct  sockaddr_in sad; /* structure to hold an IP address     */
        int     sd;              /* socket descriptor                   */
        int     port;            /* protocol port number                */
        char    *host;           /* pointer to host name                */
        int     n;               /* number of characters read           */
        char    buf[1000];       /* buffer for data from the server     */
#ifdef WIN32
        WSADATA wsaData;
        WSAStartup(0x0101, &wsaData);
#endif
        memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
        sad.sin_family = AF_INET;         /* set family to Internet     */
```

Codice client (3)

```
/* Check command-line argument for protocol port and extract    */
/* port number if one is specified.  Otherwise, use the default */
/* port value given by constant PROTOPORT                       */

if (argc > 2) {                 /* if protocol port specified   */
        port = atoi(argv[2]);   /* convert to binary            */
} else {
        port = PROTOPORT;       /* use default port number      */
}
if (port > 0)                   /* test for legal value         */
        sad.sin_port = htons((u_short)port);
else {                          /* print error message and exit */
        fprintf(stderr,"bad port number %s\n",argv[2]);
        exit(1);
}

/* Check host argument and assign host name. */

if (argc > 1) {
        host = argv[1];         /* if host argument specified   */
} else {
        host = localhost;
}
```

Codice client (4)

```
/* Convert host name to equivalent IP address and copy to sad. */

ptrh = gethostbyname(host);
if ( ((char *)ptrh) == NULL ) {
        fprintf(stderr,"invalid host: %s\n", host);
        exit(1);
}
memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);

/* Map TCP transport protocol name to protocol number. */

if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
}

/* Create a socket. */

sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0) {
        fprintf(stderr, "socket creation failed\n");
        exit(1);
}

/* Connect the socket to the specified server. */

if (connect(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
        fprintf(stderr,"connect failed\n");
        exit(1);
}
```

Codice client (5)

```
        /* Repeatedly read data from socket and write to user's screen. */

        n = recv(sd, buf, sizeof(buf), 0);
        while (n > 0) {
                write(1,buf,n);
                n = recv(sd, buf, sizeof(buf), 0);
        }

        /* Close the socket. */

        closesocket(sd);

        /* Terminate the client program gracefully. */

        exit(0);
}
```


Codice server (1)

```
/* server.c - code for example server program that uses TCP */
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

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

#define PROTOPORT       5193            /* default protocol port number */
#define QLEN            6               /* size of request queue        */

int     visits      =   0;              /* counts client connections    */
/*------------------------------------------------------------------------
 * Program:   server
 *
 * Purpose:   allocate a socket and then repeatedly execute the following:
 *              (1) wait for the next connection from a client
 *              (2) send a short message to the client
 *              (3) close the connection
 *              (4) go back to step (1)
```

Codice server (2)

```
 * Syntax:    server [ port ]
 *
 *              port  - protocol port number to use
 *
 * Note:      The port argument is optional.  If no port is specified,
 *            the server uses the default given by PROTOPORT.
 *
 *------------------------------------------------------------------------
 */
main(argc, argv)
int     argc;
char    *argv[];
{
        struct  hostent  *ptrh;  /* pointer to a host table entry       */
        struct  protoent *ptrp;  /* pointer to a protocol table entry   */
        struct  sockaddr_in sad; /* structure to hold server's address  */
        struct  sockaddr_in cad; /* structure to hold client's address  */
        int     sd, sd2;         /* socket descriptors                  */
        int     port;            /* protocol port number                */
        int     alen;            /* length of address                   */
        char    buf[1000];       /* buffer for string the server sends  */

#ifdef WIN32
        WSADATA wsaData;
        WSAStartup(0x0101, &wsaData);
#endif
        memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
        sad.sin_family = AF_INET;         /* set family to Internet     */
        sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP address   */
```

Codice server (3)

```
    /* Check command-line argument for protocol port and extract    */
    /* port number if one is specified.  Otherwise, use the default */
    /* port value given by constant PROTOPORT                       */

    if (argc > 1) {                 /* if argument specified        */
            port = atoi(argv[1]);   /* convert argument to binary   */
    } else {
            port = PROTOPORT;       /* use default port number      */
    }
    if (port > 0)                   /* test for illegal value       */
            sad.sin_port = htons((u_short)port);
    else {                          /* print error message and exit */
            fprintf(stderr,"bad port number %s\n",argv[1]);
            exit(1);
    }

    /* Map TCP transport protocol name to protocol number */

    if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
            fprintf(stderr, "cannot map \"tcp\" to protocol number");
            exit(1);
    }

    /* Create a socket */

    sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
    if (sd < 0) {
            fprintf(stderr, "socket creation failed\n");
            exit(1);
    }
```

Codice server (4)

```
/* Bind a local address to the socket */

if (bind(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
        fprintf(stderr,"bind failed\n");
        exit(1);
}

/* Specify size of request queue */

if (listen(sd, QLEN) < 0) {
        fprintf(stderr,"listen failed\n");
        exit(1);
}

/* Main server loop - accept and handle requests */

while (1) {
        alen = sizeof(cad);
        if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0) {
                fprintf(stderr, "accept failed\n");
                exit(1);
        }
        visits++;
        sprintf(buf,"This server has been contacted %d time%s\n",
                visits,visits==1?".":"s.");
        send(sd2,buf,strlen(buf),0);
        closesocket(sd2);
}
```