**La Programmazione Client/Server in C**

Socket

Una socket è un canale di comunicazione tra due processi che possono risiedere anche su host diversi.

I due processi comunicheranno fra loro leggendo e/o scrivendo i dati nella socket.

Vi sono tre tipi fondamentali di socket

\* **Datagram Socket**. Supporta la comunicazione bidirezionale e senza connessione (un host invia dei dati ad un altro host senza stabilire una connessione). Non è affidabile, non garantisce la ricezione dei pacchetti, ne' che i pacchetti saranno ricevuti nello stesso ordine con cui sono stati inviati. Nelle comunicazioni internet è utilizzata dal protocollo UDP.

\* **Stream Socket.** Supporta la comunicazione bidirezionale e con connessione (un host per inviare dei dati ad un altro host stabilisce una connessione con l'altro host). È affidabile, garantisce la ricezione dei pacchetti e che questi saranno ricevuti nello stesso ordine con cui sono stati inviati. Nelle comunicazioni internet è utilizzata dal protocollo TCP.

\* **Raw Socket**. Socket di basso livello tramite le quali è possibile accedere al livello network.

**Interazione Client/Server tramite il protocollo TCP**

Affinché vi possa essere una comunicazione tra client e server mediante il protocollo TCP è necessario innanzitutto stabilire una connessione tra il client ed il server.

Una volta stabilita la connessione vi potrà essere lo scambio di dati, terminato il quale si chiude la connessione.

**Creazione della Socket del Server**

Il primo passo consiste nella creazione del canale di comunicazione tra il server ed un eventuale client, tramite questo canale il client potrà comunicare con il server. Questo canale di comunicazione sarà costituito da una socket.

In ambiente windows prima della creazione della socket del server è necessario inizializzare la libreria winsock tramite la funzione WSAStartup

***/\* Inizializzazione della libreria Socket \*/***

***WORD wVersionRequested = MAKEWORD(2,2);***

***WSADATA wsaData;***

***wsastartup = WSAStartup(wVersionRequested, &wsaData);***

***if (wsastartup != NO\_ERROR) printf("Errore WSAStartup()\n");***

Una volta inizializzata, solo in ambiente windows, la libreria winsock, si creerà la socket.

La socket sarà creata con la funzione socket come indicato di seguito

***/\* Creazione della Socket che si porrà in ascolto di richieste del Client\*/***

***listenSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);***

***if (listenSocket < 0)***

***printf("Server: errore nella creazione della socket.\n");***

***else printf("La Listening Socket è partita\n");***

Si è costruita una socket che come indirizzamento utilizza IPv4, utilizza una connessione TCP (SOCK\_STREAM), e la comunicazione avviene secondo il protocollo TCP (IPPROTO\_TCP).

Questa parte di codice è comune al server e al client.

La funzione WSAStartup inizializza l'uso della winsock da parte del processo

***int WSAStartup(WORD wVersionRequested,LPWSADATA lpWSAData);***

consente all'applicazione di specificare la versione delle windows sockets richiesta.

La funzione ha due parametri, wVersionRequested che indica la più recente versione delle specifiche windows sockets che l'applicazione può usare. Quando viene chiamata la funzione WSAStartup la winsock esamina la versione delle windows sockets richiesta dall'applicazione, passata con il parametro wVersionRequested. Se la versione richiesta dall'applicazione è maggiore o uguale della minore delle versioni supportata dalla winsock la chiamata ha successo e la winsock restituisce le informazioni dettagliate nella struttura WSADATA, struttura puntata dal parametro lpWSAData.

***typedef struct WSAData {***

***WORD wVersion;***

***WORD wHighVersion;***

***char szDescription[WSADESCRIPTION\_LEN+1];***

***char szSystemStatus[WSASYS\_STATUS\_LEN+1];***

***unsigned short iMaxSockets;***

***unsigned short iMaxUdpDg;***

***char FAR\* lpVendorInfo;***

***} WSADATA,***

***\*LPWSADATA;***

|  |  |
| --- | --- |
| Elemento | Descrizione |
| wVersion | Indica la versione delle specifiche windows sockets che la Ws2\_32.dll si aspetta che il chiamante usi |
| wHighVersion | Indica la più recente versione delle specifiche windows sockets che questa .dll può supportare. Di solito è la stessa indicata dal parametro wVersion. |
| szDescription | Stringa (Null-terminated) in cui la Ws2\_32.dll inserirà la descrizione dell'implementazione delle Windows sockets. |
| szSystemStatus | Stringa (Null-terminated) in cui la Ws2\_32.dll inserirà informazioni rilevanti sullo stato o sulla configurazione. |
| iMaxSockets | Può essere ignorato per la versione 2 delle Windows Sockets e per le versioni successive |
| iMaxUdpDg | Ignorato per la versione 2 delle Windows Sockets e successive |
| lpVendorInfo | Ignorato per la versione 2 delle Windows Sockets e successive |

La funzione socket è definita sia in ambiente windows che in ambiente linux ed il suo prototipo è il seguente

***SOCKET socket(int af, int type, int protocol)***

Con af si indica la famiglia di indirizzi da utilizzare. I possibili valori assumibili da questo parametro sono i seguenti

|  |  |
| --- | --- |
| Famiglia | Descrizione |
| **AF\_INET** | IPv4 |
| **AF\_INET6** | IPv6 |
| **AF\_LOCAL** | Unix Domain Protocol |
| **AF\_ROUTE** | Routing Sockets |
| **AF\_KEY** | Key Socket |

Con type si indica il tipo di socket da creare. I possibili valori assumibili da questo parametro sono i seguenti

|  |  |
| --- | --- |
| Type | Descrizione |
| **SOCKET\_STREAM** | Stream Socket (TCP). Fornisce una connessione sicura, ordinata e bidirezionale. Consente la comunicazione tramite un flusso di byte di lunghezza arbitraria. Utilizzano un protocollo con connessione. |
| **SOCK\_DGRAM** | Datagram Socket (UDP). Non fornisce una connessione, la comunicazione avviene attraverso datagram, buffer di una dimensione massima prefissata (tipicamente piccola). Utilizzano un protocollo senza connessione. |
| **SOCK\_RAW** | Unix Domain protocols. Supporta le socket raw. Le socket raw consentono ad una applicazione di inviare e ricevere pacchetti con header customized |

con protocol si indica il protocollo da utilizzare, i possibili valori sono IPPROTO\_IP, IPPROTO\_IPV6, IPPROTO\_TCP, IPPROTO\_UDP, SOL\_SOCKET.

La funzione socket restituisce un socket descriptor (>0) in caso di successo o il valore -1 in caso di errore. Ogni socket descriptor identifica univocamente un canale di comunicazione che il server mette a disposizione.

Una lista di possibili codici di errore è la seguente

|  |  |
| --- | --- |
| Errore | Descrizione |
| EHOSTDOWN | The networking subsystem has not been started |
| EAFNOSUPPORT | The specified address family is not supported on this version of the system |
| ESOCKTNOSUPPORT | The specified socket type is not supported in this address family |
| EPROTONOSUPPORT | The specified protocol is not supported |
| EMFILE | The per-process descriptor table is full |
| ENOBUFS | No buffer space is available. The socket cannot be created |
| ENFILE | The system's table of open files is temporarily full, and no more socket calls can be accepted |
| EPROTOTYPE | The type of socket and protocol do not match |
| ETIMEDOUT | The connection timed out |

**Assegnazione di un Indirizzo**

Una volta creato il canale di comunicazione il client deve essere messo in condizione di accedervi. Per potervi accedere deve esistere un indirizzo per accedervi e il client deve conoscere l’indirizzo di questo canale.

A tale scopo si associa al server un indirizzo e alla socket una porta che la identifica univocamente.

Per associare indirizzo e porta alla socket si usa la funzione **bind** che assegna la gestione di un indirizzo e di una porta locali ad un socket

La funzione bind è definita sia in ambiente windows che in ambiente linux ed il suo prototipo è il seguente

***int bind(int sockfd, const struct sockaddr \*myaddr, socklen\_t addrlen)***

dove con sockfd si indica l’identificativo della socket (il socket descriptor), con myaddr si indica l’indirizzo (indirizzo:porta) gestito dalla socket, con addrlen si indica la lunghezza dell’indirizzo (in byte). La funzione restituisce 0 in caso di successo o -1 in caso di errore.

La struttura sockaddr varia a seconda del protocollo selezionato. Nel caso si usi IPv4 la struttura di tipo sockaddr ha la forma seguente

***struct sockaddr\_in{***

***short int sin\_family; /\* AF\_INET \*/***

***unsigned short int sin\_port; /\* numero di porta \*/***

***struct in\_addr sin\_addr;***

***char sin\_zero[8] /\* non usato \*/***

***}***

possono essere specificati un indirizzo IP (sin\_addr), o una porta (TCP o UDP, sin\_port), o entrambi, o nessuno dei due. Quando un valore non viene specificato esso viene scelto automaticamente dal sistema operativo.

Normalmente la funzione bind() viene usata solo quando si intende realizzare un server in quanto è necessario specificare su quale indirizzo IP e su quale porta il server rimarrà in attesa di connessioni da parte dei client.

Il numero di porta scelto può variare tra 0 e 65535 e non deve essere già in uso da un'altra socket.

L'assegnazione dell'indirizzo alla socket prevede l'esecuzione dei seguenti passi

***/\* Effettua la bind sull’indirizzo e porta ora specificati \*/***

***port = 4000;***

***Server\_addr.sin\_family = AF\_INET;***

***Server\_addr.sin\_addr.s\_addr = "127.0.0.1";***

***Server\_addr.sin\_port = htons(port);***

***if (bind(listenSocket,(LPSOCKADDR)&Server\_addr,sizeof(struct sockaddr)) < 0)***

***printf("Server: errore durante la bind.\n");***

Si specifica l'indirizzo (per specificare un indirizzo generico, con IPv4 si usa il valore **INADDR\_ANY**) e la porta della socket, quindi si effettua la bind.

**Server in Ascolto**

Una volta che il canale di comunicazione è stato creato e sono stati dati i riferimenti ad esso il server si deve mettere in ascolto in attesa di una richiesta di connessione da parte di un client. Per mettersi in ascolto il server usa la funzione **listen**

Nell'esempio proposto il server si pone in ascolto sulla socket identificata da listenSocket e accetta fino ad un massimo di SOMAXCONN tentativi di connessione

***/\* La socket si pone in "ascolto" tramite la listen() \*/***

***ls\_result = listen(listenSocket, SOMAXCONN);***

***if (ls\_result < 0) printf("Server: errore durante la listen.\n");***

***else printf("La Socket è in Ascolto\n");***

La funzione listen è definita sia in ambiente windows che in ambiente linux ed il suo prototipo è il seguente

***int listen(int sockfd, int backlog)***

dove con sockfd si indica la socket che si pone in ascolto, e con backlog si indica il numero massimo di connessioni in attesa di completamento contemporaneamente gestibili dalla socket. Queste connessioni sono quelle connessioni pendenti, connessioni richieste dai client ma non ancora accettate dal server tramite la funzione **accept.**

Questa funzione deve essere usata solo all’interno di un server. La funzione restituisce 0 in caso di successo o -1 in caso di errore.

**Accettazione di una Richiesta di Connessione del Client da parte del Server**

Nel momento in cui il server si è posto in ascolto nel canale di comunicazione rappresentato da una socket e giunge una richiesta di connessione il server l’accetta ed instaura una connessione con il client richiedente.

Nel caso in cui ci siano più richieste di connessione a formare una coda il server accetterà ed esaudirà le richieste nell’ordine in cui queste sono arrivate.

Per far accettare la richiesta di connessione al server si usa la funzione **accept**.

***/\* La socket accetta la richiesta di connessione del Client \*/***

***sin\_size = sizeof(struct sockaddr\_in);***

***remoteSocket = accept(listenSocket, (struct sockaddr \*)&Client\_addr,***

***&sin\_size);***

***printf("Accettata Connessione con Client: %s\n",***

***inet\_ntoa(Client\_addr.sin\_addr));***

La funzione accept è definita sia in ambiente windows che in ambiente linux ed il suo prototipo è il seguente

***int accept(int sockfd, struct sockaddr \*clientaddr, socklen\_t \*addrlen)***

dove con sockfd si indica la socket su cui il server ha accettato la connessione, con clientaddr si indica l’indirizzo del client che ha effettuato la richiesta di connessione che è stata accettata, con addrlen si indica la lunghezza di questo indirizzo.

La funzione restituisce un socket descriptor relativo ad una nuova socket creata in seguito all’accettazione della connessione. Sarà su questa nuova socket che avverrà la comunicazione tra il server e il client che ha fatto la richiesta di connessione. Sulla socket originale il server continua a restare in ascolto per nuove richieste di connessione dei client.

Se al momento dell’esecuzione della istruzione, nessun client si è collegato, la funzione mette in attesa il processo. In caso di errore viene restituito il valore -1. Ritorna un numero di socket positivo in caso di successo oppure -1 se c’è errore; in tal caso errno può assumere gli stessi valori visti nel caso di listen().

**Richiesta di Connessione del Client**

Il client per potersi connettere con il server deve creare anch’esso una socket. Una volta creata la socket il client può avanzare una richiesta di connessione al server.

La richiesta di connessione viene effettuata dal client tramite la funzione connect.

***addr.sin\_family = AF\_INET;***

***addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");***

***addr.sin\_port = htons(port);***

***clientsocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);***

***connect(clientsocket, (LPSOCKADDR)&addr, sizeof(addr));***

Non è necessario eseguire una chiama a bind prima di connect, il sistema operativo sceglierà automaticamente quale indirizzo IP e quale porta locali utilizzare per il collegamento. In caso di esito positivo della funzione connect, la connessione è completata e i processi possono comunicare.

La funzione connect è definita sia in ambiente windows che in ambiente linux ed il suo prototipo è il seguente

***int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen)***

dove con sockfd si indica l’identificativo del socket, con addr si indica l’indirizzo del server con cui ci si vuole connettere, e con addrlen si indica la lunghezza di questo indirizzo.

Ritorna 0 in caso di successo e -1 se c’è un errore, nel qual caso errno assume i valori del codice che indica l’errore.

La struttura sockaddr deve necessariamente contenere l’indirizzo IP e la porta del server a cui ci si vuole connettere.

**Comunicazioni Client Server**

Una volta stabilita la connessione tra client e server questi potranno comunicare tra loro, uno invierà dei dati che l’altro riceverà e viceversa.

Per inviare e ricevere dati si usano le due funzioni send e recv. Il client invierà un messaggio al server, tramite la funzione send, indicando la socket tramite la quale si svolge la comunicazione con quel server, il messaggio da inviare, la dimensione del messaggio, in byte, ed il valore del flag, di solito posto a zero.

***send(clientsocket, messaggio, sizeof(messaggio), 0);***

Il server riceverà il messaggio, tramite la funzione recv, indicando la socket tramite la quale si svolge la comunicazione con quel client, la variabile in cui porre il messaggio in arrivo, la dimensione, in byte, del messaggio da leggere, ed il valore del flag, di solito posto a zero.

***recv(remoteSocket, buffer, sizeof(buffer), 0);***

La funzione send è definita sia in ambiente windows che in ambiente linux ed il suo prototipo è il seguente

***int send (int sock, char \*buffer, int len, int flags)***

dove con sock si indica la socket attraverso cui instradare il messaggio che si trova nella stringa buffer e la cui dimensione in byte è len.

La funzione recv è definita sia in ambiente windows che in ambiente linux ed il suo prototipo è il seguente

***int recv (int sock, char \*buffer, int len, int flags)***

dove con sock si indica la socket da cui si leggerà il messaggio in arrivo che sarà posto nella stringa buffer e la cui dimensione in byte è len.

**Chiusura della Connessione**

Una volta conclusa la comunicazione viene chiuso il canale di connessione costituito dalla socket.

Per farlo si utilizza, sia nel client che nel server, la funzione close

***close(ssocket);***

***WSACleanup();***

La funzione close è definita sia in ambiente windows che in ambiente linux ed il suo prototipo è il seguente

***int close(int sock);***

dove con sock si indica la socket che deve essere chiusa. Restituisce 0 in caso di successo o -1 in caso di errore.

I dati ancora presenti in coda vengono comunque inviati prima che della chiusura della socket.

**//Server TCP**

#include <iostream>

#include <WinSock2.h>

using namespace std;

int main()

{ WSADATA wsaData; // La struttura WSADATA contiene informazioni sull’implementazione delle socket windows

SOCKET requestSocketId; // Creare un elemento di tipo socket chiamato requestSocketI (Server):

SOCKET communicationSocketId; // Creare un elemento di tipo socket chiamato communicationSocketId (Client):

struct sockaddr\_in server; // Definisce la variabile strutturata server del Tipo sockaddr\_in

struct sockaddr\_in client;

server.sin\_family = AF\_INET; // Valorizza gli attributi della variabile strutturata server (Famiglia Internet)

//server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server.sin\_addr.s\_addr = htonl(INADDR\_ANY); // Il Server accetta richieste da ogni interfaccia di rete

server.sin\_port = htons(5000); // Porta 5000

char buffer[1024];

int errorCode;

int clientSize;

int n;

errorCode = WSAStartup(0x0202, &wsaData); // Inizializza la versione di socket windows richiesta

if(errorCode != 0)

{ cout<<"Errore inizializzazione libreria WinSock2.h!\n";

}

else

{ requestSocketId = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP); // Cres una Socket della Famiglia Internet / TCP (stream) /

if(requestSocketId == INVALID\_SOCKET)

{ cout<<"Errore apertura socket!\n";

}

else

{ errorCode = bind(requestSocketId, (struct sockaddr \*) &server, sizeof(server)); // Associa un Nome (IndirizzoIP:Porta) alla Socket creata

if(errorCode == SOCKET\_ERROR)

{ cout<<"Errore di binding con la porta 5000!\n";

}

else

{ do

{ errorCode = listen(requestSocketId, 8); // Si mette in stato di attesa di richiesta di connessione

if(errorCode == SOCKET\_ERROR)

{ cout<<"Errore impostazione socket!\n";

}

else

{ cout<<"Server in ascolto...\n";

clientSize = sizeof(client);

communicationSocketId = accept(requestSocketId, (struct sockaddr \*) &client, &clientSize); // Accetta una connessione in entrata su questa Socket

if(communicationSocketId != INVALID\_SOCKET)

{ cout<<"Client connected!

("<<inet\_ntoa(client.sin\_addr)<<":"<<ntohs(client.sin\_port)<<")\n";

do

{ //memset(buffer, 0, 1024);

n = recv(communicationSocketId, buffer, sizeof(buffer), 0); // Riceve dati da una Socket connessa

if(n > 0)

{ cout<<"Dati ricevuti da

("<<inet\_ntoa(client.sin\_addr)<<":"<<ntohs(client.sin\_port)<<"): "<<buffer<<endl;

cout<<"Inserisci la risposta: ";

fflush(stdin);

scanf("%[^\n]s",&buffer);

//cin.getline(buffer,1024);

send(communicationSocketId, buffer, sizeof(buffer), 0); // Invia dati ad una Socket connessa

}

} while(n > 0);

cout<<"Client disconnected! ("<<inet\_ntoa(client.sin\_addr)<<":"<<ntohs(client.sin\_port)<<")\n";

}

}

} while(strcmp(buffer,"fine")!=0);

}

}

}

closesocket(requestSocketId);

WSACleanup();

}

**//Client TCP**

#include <iostream>

#include <WinSock2.h>

using namespace std;

int main()

{ WSADATA wsaData;

SOCKET socketId;

struct sockaddr\_in server;

struct sockaddr\_in client;

char buffer[1024];

client.sin\_family = AF\_INET;

client.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

client.sin\_port = htons(5001);

server.sin\_family = AF\_INET;

server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server.sin\_port = htons(5000);

int errorCode;

int clientSize;

int serverSize;

int n;

errorCode = WSAStartup(0x0202, &wsaData);

if(errorCode != 0)

{ cout<<"Errore inizializzazione libreria WinSock2.h!\n";

}

else

{ socketId = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if(socketId == INVALID\_SOCKET)

{ cout<<"Errore apertura socket!\n";

}

else

{ errorCode = bind(socketId, (struct sockaddr \*) &client, sizeof(client));

if(errorCode == SOCKET\_ERROR)

{ cout<<"Errore di binding con la porta 5001!\n";

}

else

{ errorCode = connect(socketId,(struct sockaddr\*)&server, sizeof(server));

if(errorCode == SOCKET\_ERROR)

{ cout<<"Errore impostazione socket!\n";

}

else

{ cout<<"Connesso al server!\n";

do

{ cout<<"Inserisci messaggio: ";

fflush(stdin);

scanf("%[^\n]s",&buffer);

//cin.getline(buffer,1024);

n = send(socketId, buffer, sizeof(buffer), 0);

if(n > 0)

{ n = recv(socketId, buffer, sizeof(buffer), 0);

cout<<"Risposta del server: "<<buffer<<endl;

}

} while(strcmp(buffer,"fine")!=0);

}

}

}

}

closesocket(socketId);

WSACleanup();

}