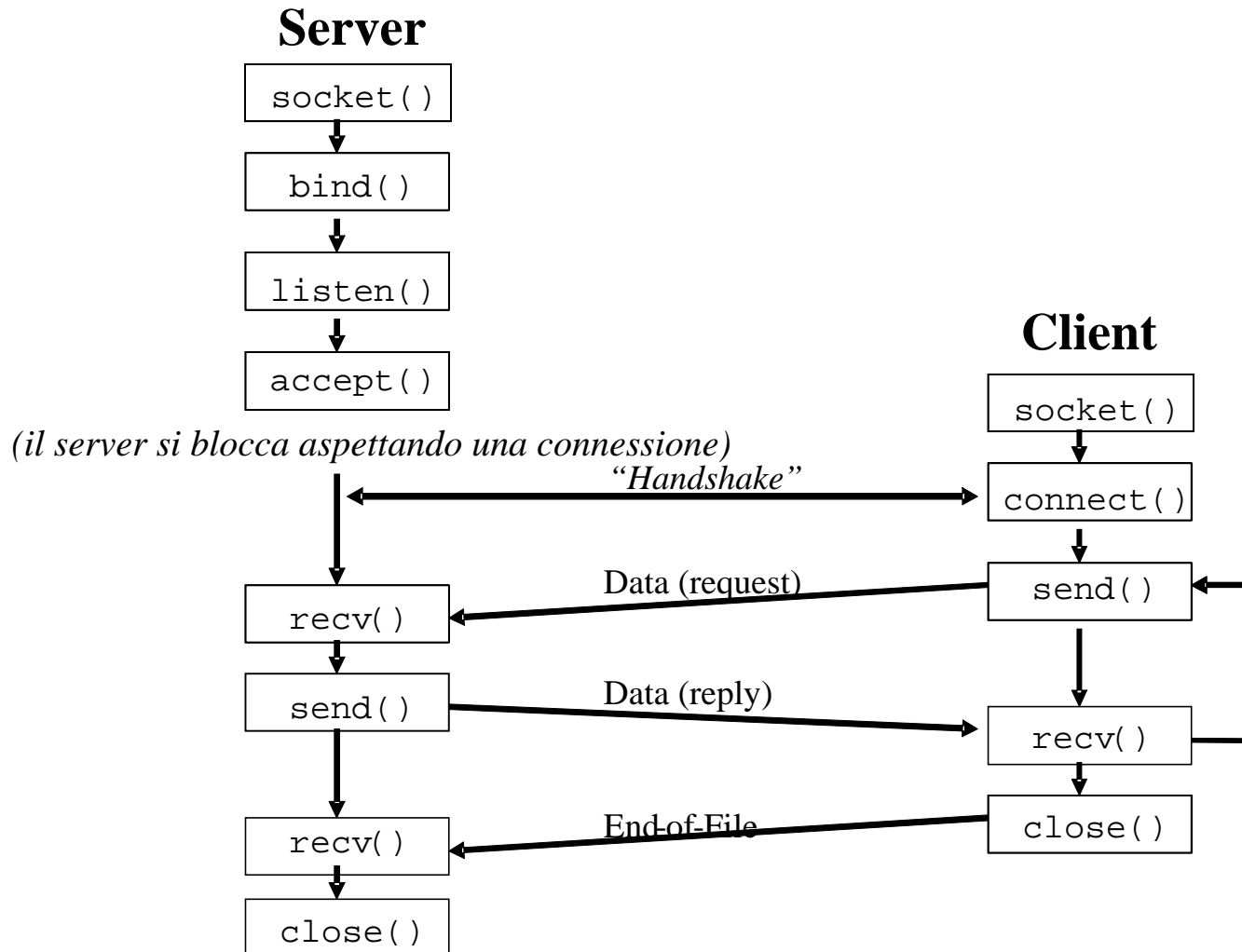


Esercitazione sulle Socket

Sommario

- Echo Server
 - Specifica
 - Descrizione programma (TCP Socket)
 - Client UNIX/WIN
 - Server UNIX/WIN
 - Server *multithread* UNIX/WIN
 - Descrizione programma (UDP Socket)

Interazione client/server (TCP Socket)



Descrizione client (UNIX) (main)

```
/*
   TCPCLIENT.C
   =====
*/

int main(int argc, char *argv[])
{
    /* dichiarazione variabili */
    /* connessione al server */
    /* lettura della stringa da tastiera */

    printf("Inserire la stringa da spedire: ");
    fgets(buffer, MAX_LINE, stdin);

    /* trasmissione stringa al server e ricezione della risposta */

    Writeline(conn_s, buffer, strlen(buffer));
    memset(buffer, 0, sizeof(char)*(strlen(buffer)+1));
    Readline(conn_s, buffer, MAX_LINE-1);

    /* output della stringa ricevuta */

    printf("Risposta del server: %s\n", buffer);
}
```

Descrizione client (WIN) (main)

- La struttura del programma è la stessa
- Qualche differenza legata alla piattaforma
 - Inizializzazione delle socket

```
int main(int argc, char *argv[])
{
    /* dichiarazione variabili */

    if (WSAStartup(MAKEWORD(1,1), &wsaData) != 0)
    {
        printf("errore in WSAStartup()\n");
        exit(EXIT_FAILURE);
    }
    /* connessione al server */
    /* lettura della stringa da tastiera */
    /* trasmissione stringa al server e ricezione della risposta */
    /* output della stringa ricevuta */

    WSACleanup();
}
```

Descrizione client (WIN) (include e variabili)

```
#include <stdio.h>
#include <winsock.h>
#include <stdlib.h>
#include "helper.h"      /* Our own helper functions */

int main(int argc, char *argv[])
{
    SOCKET      conn_s;      /* connection socket */

    u_long      nRemoteAddr;
    WSADATA     wsaData;

}
```

Descrizione client (WIN) (IP remoto)

```
/* setta l'IP del server */

nRemoteAddr = inet_addr(szAddress);

if (nRemoteAddr == INADDR_NONE)
{
    printf("client: indirizzo IP non valido.\nclient: risoluzione
           nome...");

    if ( (he=gethostbyname(szAddress)) == 0 )
    {
        printf("fallita.\n");
        exit(EXIT_FAILURE);
    }

    printf("riuscita.\n\n");

    nRemoteAddr = *((u_long *)he->h_addr_list[0]);
}

servaddr.sin_addr.s_addr = nRemoteAddr;
```

Descrizione client (WIN) (helper.h e helper.c)

- Nella `write_line` l'istruzione

```
if ( (nwritten = write(sockd, buffer, nleft)) <= 0 )
```

è sostituita con

```
if ( (nwritten = send(sockd, buffer, nleft, 0)) <= 0 )
```

- Nella `read_line` l'istruzione

```
rc = read(sockd, &c, 1);
```

viene sostituita con

```
rc = recv(sockd, &c, 1, 0);
```

Descrizione server (UNIX) (main)

```
/*
  TCPSERVER.C
  =====
*/

#include <signal.h>
int main(int argc, char *argv[]) {
    int      list_s;          /* listening socket      */
    int      conn_s;         /* connection socket    */

    ParseCmdLine(argc, argv, &endptr);

    /* creazione della socket */
    while ( 1 )
    {
        /* attesa di connessione */
        Readline(conn_s, buffer, MAX_LINE-1);
        printf("server: il client ha scritto\n\t%s\n",buffer);
        Writeline(conn_s, buffer, strlen(buffer));

        /* chiusura socket del client */
    }
    /* chiusa socket del server */
}
```

Descrizione server (UNIX) (creazione socket)

```
if ( (list_s = socket(AF_INET, SOCK_STREAM, 0)) < 0 )
{
    fprintf(stderr, "server: errore nella creazione della socket.\n");
    exit(EXIT_FAILURE);
}

memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family      = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port        = htons(port);

if ( bind(list_s, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0 )
{
    fprintf(stderr, "server: errore durante la bind.\n");
    exit(EXIT_FAILURE);
}

if ( listen(list_s, LISTENQ) < 0 )
{
    fprintf(stderr, "server: errore durante la listen.\n");
    exit(EXIT_FAILURE);
}
```

Descrizione server (UNIX) (attesa connessione client)

```
/* Wait for a connection, then accept() it */
sin_size = sizeof(struct sockaddr_in);
if ( (conn_s = accept(list_s, (struct sockaddr *)&their_addr, &sin_size) ) < 0 )
{
    fprintf(stderr, "server: errore nella accept.\n");
    exit(EXIT_FAILURE);
}

printf("server: connessione da %s\n", inet_ntoa(their_addr.sin_addr));

/* Close the connected socket */
if ( close(conn_s) < 0 )
{
    fprintf(stderr, "server: errore durante la close.\n");
    exit(EXIT_FAILURE);
}
```

Descrizione server (WIN) (main)

- La struttura del programma è la stessa
- Qualche differenza legata alla piattaforma
 - Inizializzazione delle socket (le istruzioni sono QUASI IDENTICHE a quelle mostrate per il client)
 - Errori (INVALID_SOCKET, SOCKET_ERROR)

Per Windows al posto di

```
close(conn_s);
```

si usa

```
closesocket(conn_s);
```

```
/* Close the connected socket */  
if ( closesocket(conn_s) == SOCKET_ERROR )  
{  
    fprintf(stderr, "server: errore durante la close.\n");  
    exit(EXIT_FAILURE);  
}
```

Descrizione server multithread (main)

```
/*
  TCPSERVER.C
  =====
*/

#include <signal.h>
int main(int argc, char *argv[]) {
    int      list_s;          /* listening socket      */
    int      conn_s;         /* connection socket     */

    ParseCmdLine(argc, argv, &endptr);

    /* creazione della socket */
    while ( 1 )
    {
        /* attesa di connessione */
        /* creazione thread di gestione */
        /* Readline(conn_s, buffer, MAX_LINE-1);
         * printf("server: il client ha scritto\n\t%s\n",buffer);
         * Writeline(conn_s, buffer, strlen(buffer)); */
        /* chiusura socket del client */
    }
    /* chiusa socket del server */
}
```

Server multithread (UNIX) (thread di gestione)

```
void * receiveThread(void * param);

/* Creazione thread di gestione */
while ( 1 ) {
    proc_id=pthread_create(&tid, NULL, receiveThread, (void*)&conn_s);
}
pthread_join(tid, &status);

void * receiveThread(void * param) {
    char buf[MAX_LINE];
    int my_sock = *((int*)param);

    Readline(my_sock, buf, MAX_LINE-1);
    printf("server: il client ha scritto\n\t%s\n", buf);
    Writeline(my_sock, buf, strlen(buf));

    close(my_sock);

    pthread_exit((void *)&status);
}
```

Server multithread (WIN) (thread di gestione)

```
void receiveThread(void* sd);

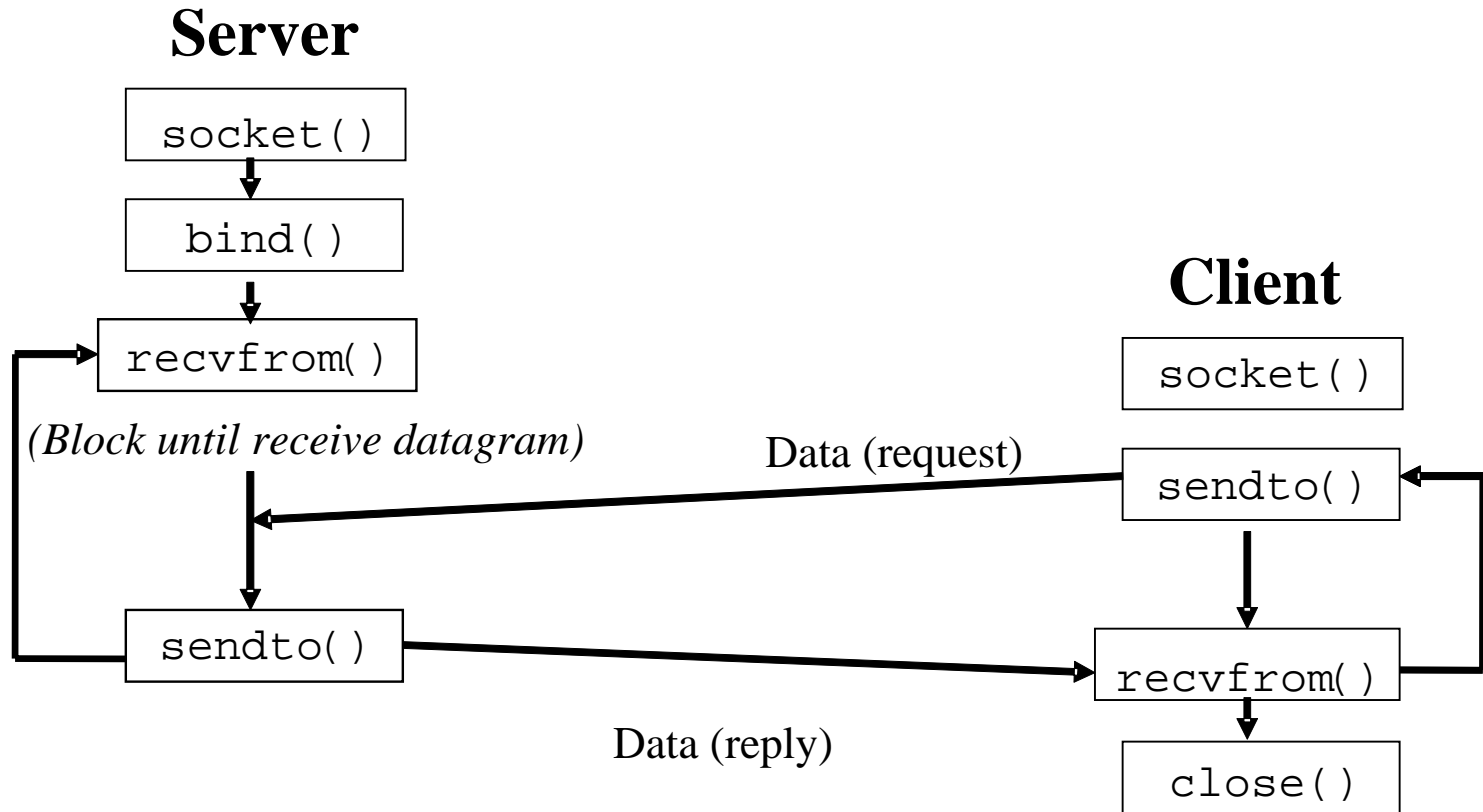
/* Creazione thread di gestione */
while ( 1 ) {
    CreateThread( NULL, 0,
        (LPTHREAD_START_ROUTINE)receiveThread, /* is equivalent to declare
                                                DWORD WINAPI receiveThread(void*)*/
        (LPVOID)conn_s, /* is equivalent to (void*)conn_s*/
        NORMAL_PRIORITY_CLASS,
        &nThread);
}

void receiveThread(void* sd) {
    char buf[MAX_LINE];
    SOCKET conn = (SOCKET)sd;

    Readline(conn, buf, MAX_LINE-1);
    printf("server: il client ha scritto\n\t%s\n",buf);
    Writeline(conn, buf, strlen(buf));

    /* Close the connected socket */
    closesocket(conn);
}
```

Interazione client/server (UDP Socket)



Descrizione client *incompleto* (UNIX) (main)

```
/*
    UDPCLIENT.C
    =====
*/

int main(int argc, char *argv[]) {
    /* dichiarazione variabili */
    /* creazione della socket e relativa bind al SO */
    /* trasmissione dati */
        rc = sendto(sd, argv[i], strlen(argv[i])+1, 0,
                    (struct sockaddr *) &remoteServAddr,
                    sizeof(remoteServAddr));

        if(rc<0) {
            printf("impossibile trasmettere dati");
            close(sd);
            exit(1);
        }
    }
    /* ricezione dati (tramite recvfrom()) */
    return 1;
}
```

Descrizione client *incompleto* (UNIX) (include e variabili)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
```

```
#define REMOTE_SERVER_PORT 7000
#define MAX_MSG 100
```

```
int sd, rc, i; /* socket, ..... */
struct sockaddr_in cliAddr, remoteServAddr; /* indirizzi del client e
del server */
struct hostent *h; /* info sull'host */
```

Descrizione client *incompleto* (UNIX) (conn.)

```
/* socket creation */
sd = socket(AF_INET,SOCK_DGRAM,0);
if (sd<0)
{
    printf("%s: impossibile aprire la socket \n",argv[0]);
    exit(1);
}

/* bind any port */
cliAddr.sin_family = AF_INET;
cliAddr.sin_addr.s_addr = htonl(INADDR_ANY);
cliAddr.sin_port = htons(0);

rc = bind(sd, (struct sockaddr *) &cliAddr, sizeof(cliAddr));
if(rc<0)
{
    printf("%s: impossibile aprire la porta\n", argv[0]);
    exit(1);
}
```

NON C'È LA CONNECT !!!!!

Descrizione server *incompleto* (UNIX) (main)

```
/*
  UDPSERVER.C
  =====
*/
int main(int argc, char *argv[]) {
  /* dichiarazione variabili */
  /* creazione della socket e relativa bind al SO*/

  while(1) {
    /* ricezione messaggio */
    cliLen = sizeof(cliAddr);
    n = recvfrom(sd, msg, MAX_MSG, 0,
                (struct sockaddr *) &cliAddr, &cliLen);
    if (n<0) {
      printf("%s: impossibile ricevere dati \n",argv[0]);
      continue;
    }
    /* trasmissione dati (tramite sendto()) */
  }
  return 0;
}
```

Descrizione server *incompleto* (UNIX) (include e variabili)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close() */
#include <string.h> /* memset() */

#define LOCAL_SERVER_PORT 7000
#define MAX_MSG 100

int sd, rc, n, cliLen; /* socket, ..... */
struct sockaddr_in cliAddr, servAddr; /* address del server
                                     e del client */
char msg[MAX_MSG]; /* buffer di ricezione */
```

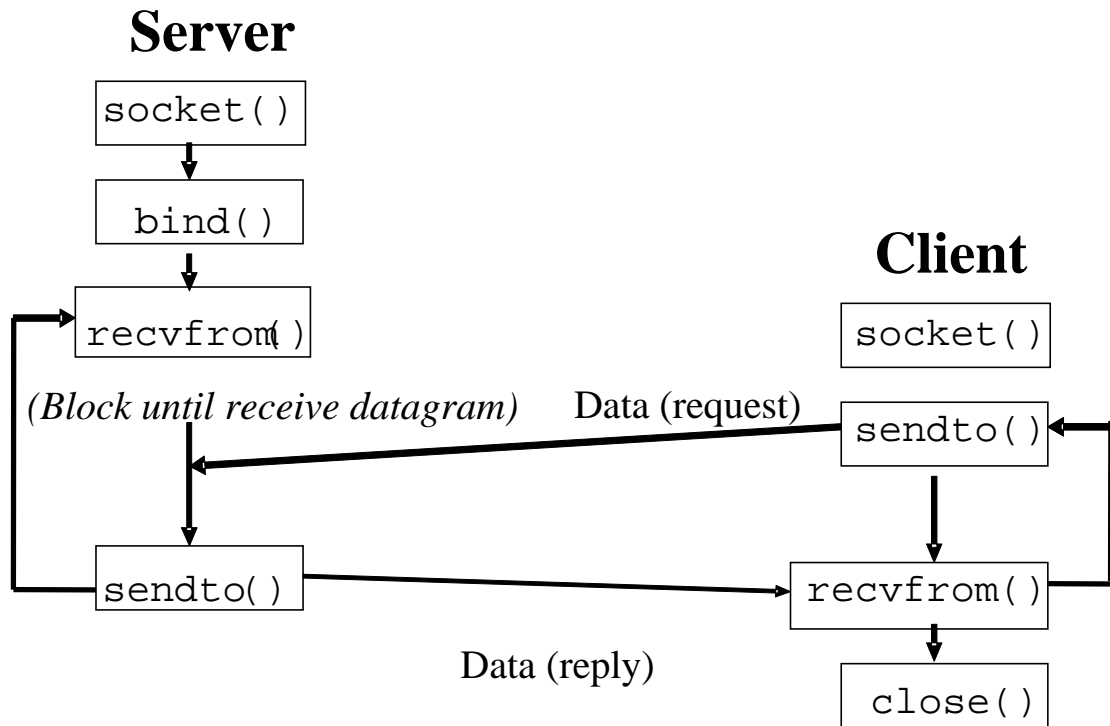
Descrizione server *incompleto* (UNIX) (conn.)

```
/* socket creation */
sd = socket(AF_INET, SOCK_DGRAM, 0);
if(sd<0)
{
    printf("%s: errore nell'apertura dell socket.\n",argv[0]);
    exit(1);
}

/* bind local server port */
servAddr.sin_family = AF_INET;
servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
servAddr.sin_port = htons(LOCAL_SERVER_PORT);
rc = bind (sd, (struct sockaddr *) &servAddr,sizeof(servAddr));
if(rc<0)
{
    printf("%s: errore nella bind %d \n",
           argv[0], LOCAL_SERVER_PORT);
    exit(1);
}
```

NON CI SONO LISTEN E ACCEPT !!!

UDP Client/Server (NOTE)



- UDP è non-reliable
- Se il pacchetto del client non arriva al server il client rimane bloccato nella `recvfrom()`
- Lo stesso se è la risposta del server a perdersi

Conclusioni

- Generalità Socket
- Echo server (UNIX/WINDOWS, TCP/UDP)
- Sul sito saranno disponibili i codici del client e del server per entrambe le piattaforme