Programmazione e Sicurezza delle Reti

Programmazione di rete client/server mediante l'interfaccia socket

Prof. Davide Quaglia

UNIVERSITÀ di VERONA
Dipartimento di INFORMATICA

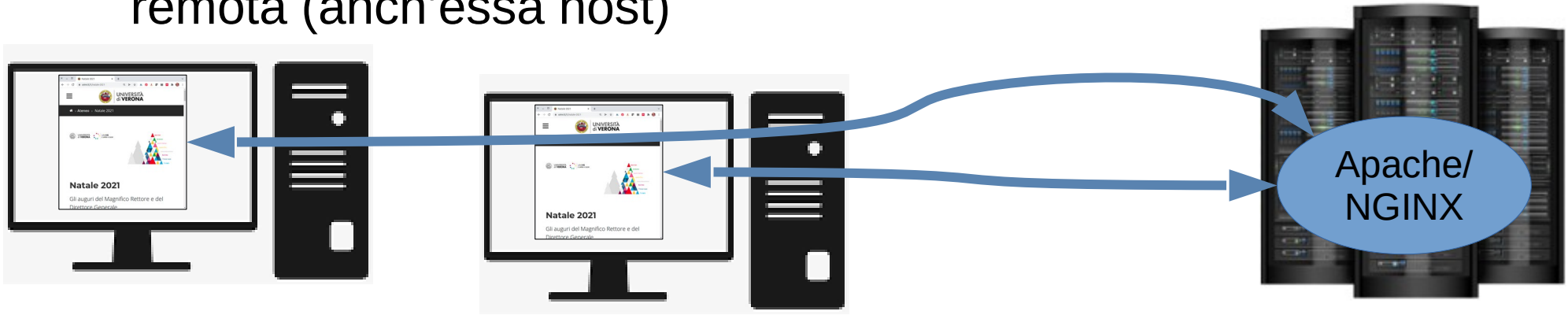
NeST
Networked Systems and Technologies

Host, processo e applicazione

- L'host (in Inglese "colui che ospita") è una **macchina** (PC, supercomputer, tablet, smartphone, smart watch, semaforo intelligente, ecc…)
  - Sempre identificata da un indirizzo IP a cui, opzionalmente, può essere associato un nome Internet
- Processo: **programma in esecuzione** sull'host che trasmette/riceve pacchetti verso/da altri processi su altri host attraverso la rete
  - Identificato da un numero di porta nell'intervallo 0..65535
- Applicazione: collaborazione tra un **insieme di processi** sparsi sulla rete per fare qualcosa di utile per l'utente (es. Web, chat, e-mail, telemedicina)

Esempi di applicazioni che usano la rete

- Web è l'applicazione
- Chrome/Firefox/Edge/Safari è il processo in esecuzione sul mio PC/tablet/smartphone che funge da host.
- Apache/NGINX è il processo in esecuzione sulla macchina remota (anch'essa host)

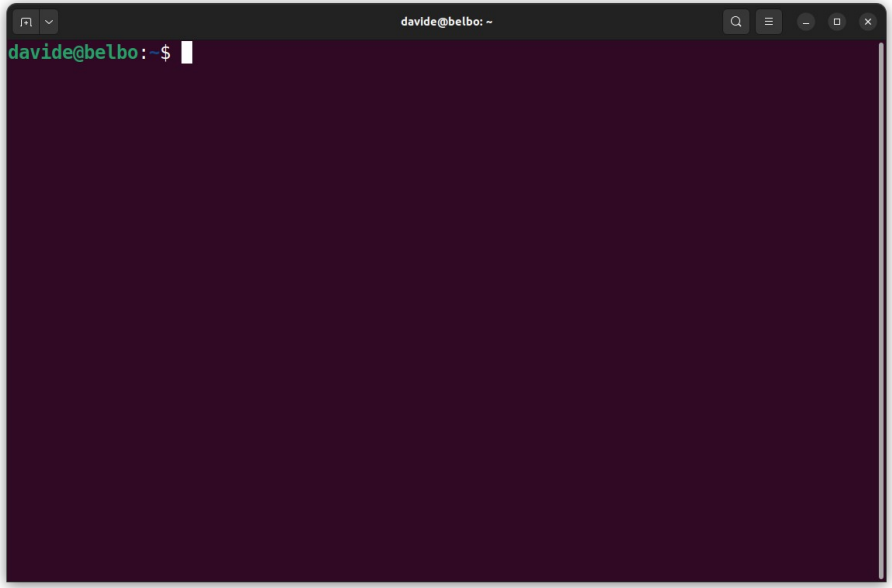


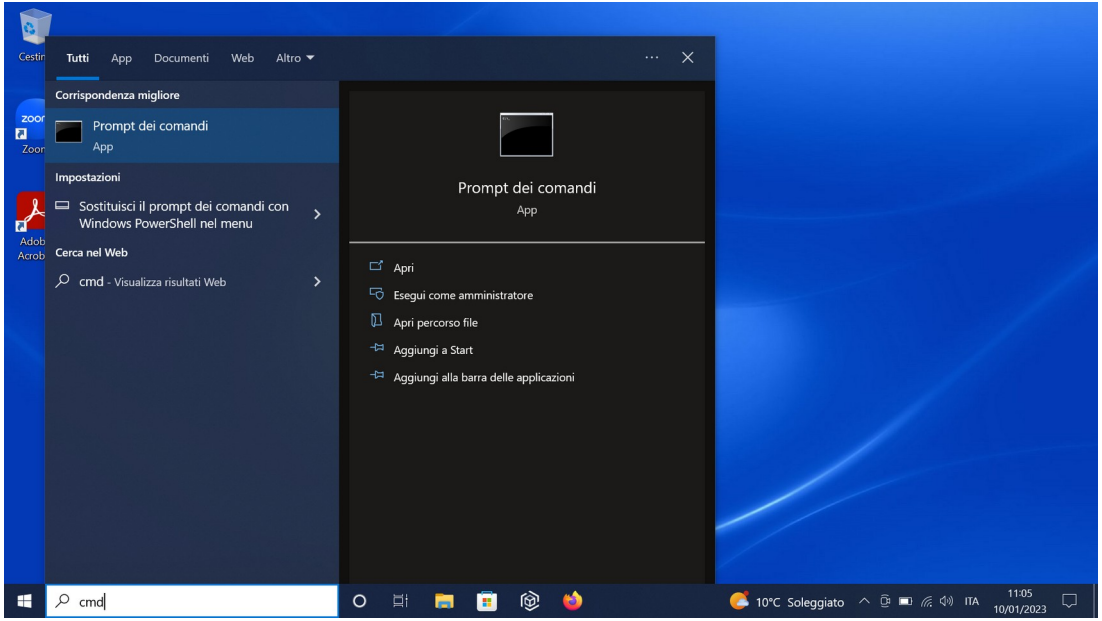
Esempi di applicazioni che usano la rete (cont.)

- Telegram è l'applicazione
- APP Telegram è il processo che gira sul mio smartphone che funge da host
- Telegram Server è il processo in esecuzione sulla macchina remota (anch'essa host)

La finestra dei comandi del PC (detta “terminale” o “shell” o “console”)

- In Windows cercare l’APP “CMD” oppure installate “PowerShell”

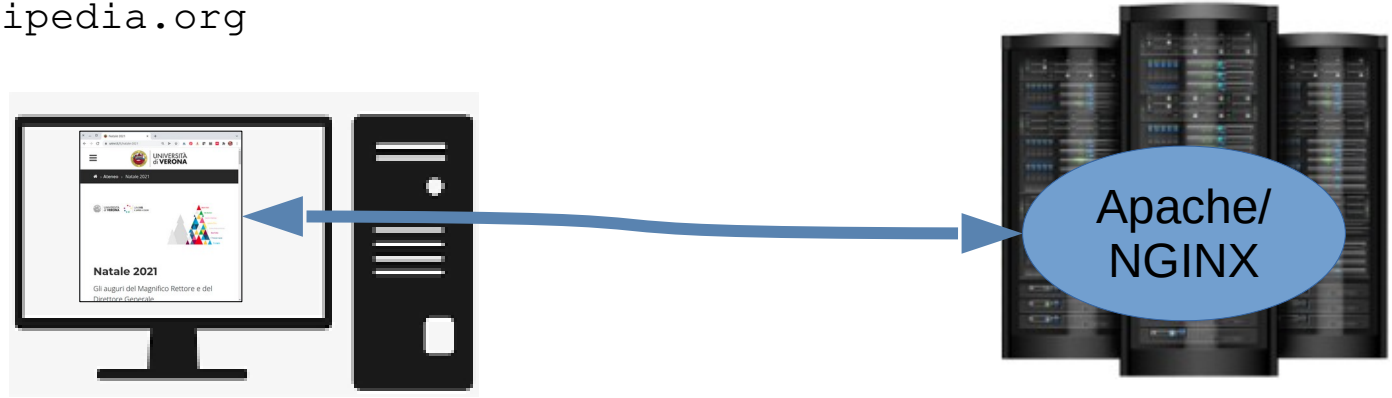




- Il Linux e MAC OS cercare l’applicazione “terminale” o “console”

Scoprire l'indirizzo IP degli host

- Prendiamo l'esempio del Web e consideriamo il sito www.wikipedia.org
- L'indirizzo IP del mio host si trova digitando un comando da terminale (finestra CMD in Windows)
  - In Linux e MAC OS: `ifconfig –a`
  - In Windows: `ipconfig /all`
- L'host del server di Wikipedia ha nome Internet www.wikipedia.org e indirizzo IP che si trova digitando il seguente comando da terminale (finestra CMD in Windows)
  - `nslookup www.wikipedia.org`

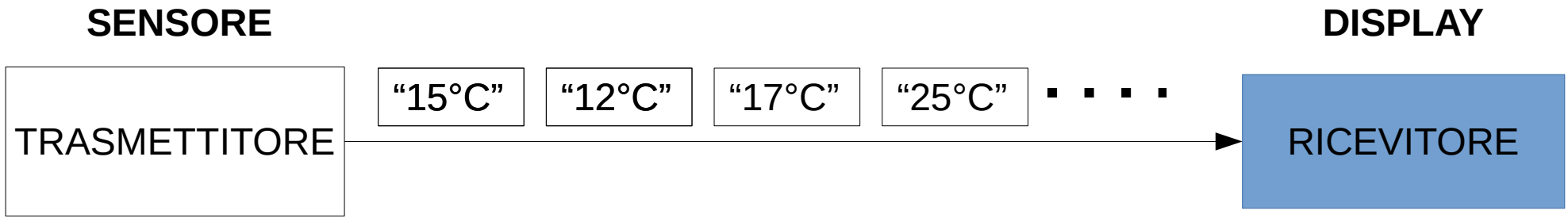


Modalità di trasmissione in Internet

- Trasmettere un bit o un byte per volta non è efficiente
- Sequenza di byte chiamata utilizzando vari sinonimi:
  - Pacchetto
  - Protocol Data Unit (PDU)
  - Datagram

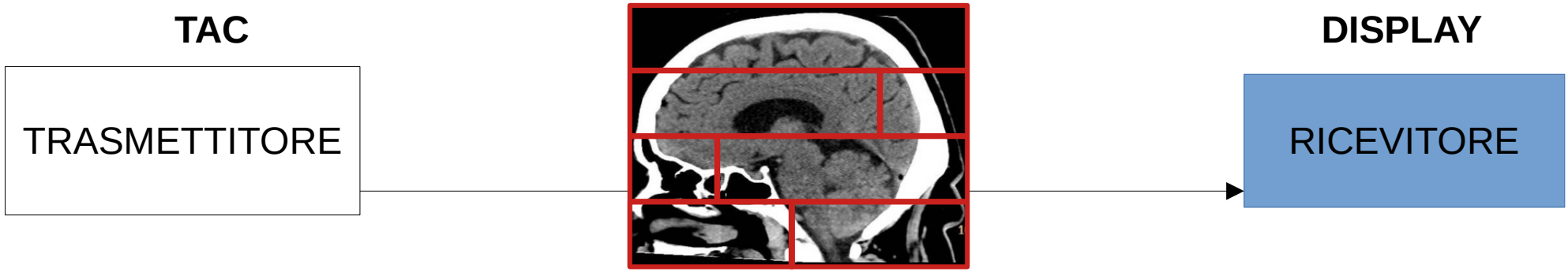
Applicazioni orientate al datagram

- Ogni pacchetto scambiato tra gli host è indipendente dai precedenti e successivi
- Le perdite di pacchetti non vengono tenute in considerazione
- Es. trasmissione di temperature



Applicazioni orientate alla connessione

- Tra i pacchetti trasmessi c'è una relazione: sono parte di un messaggio più grande (ad es. un'immagine)



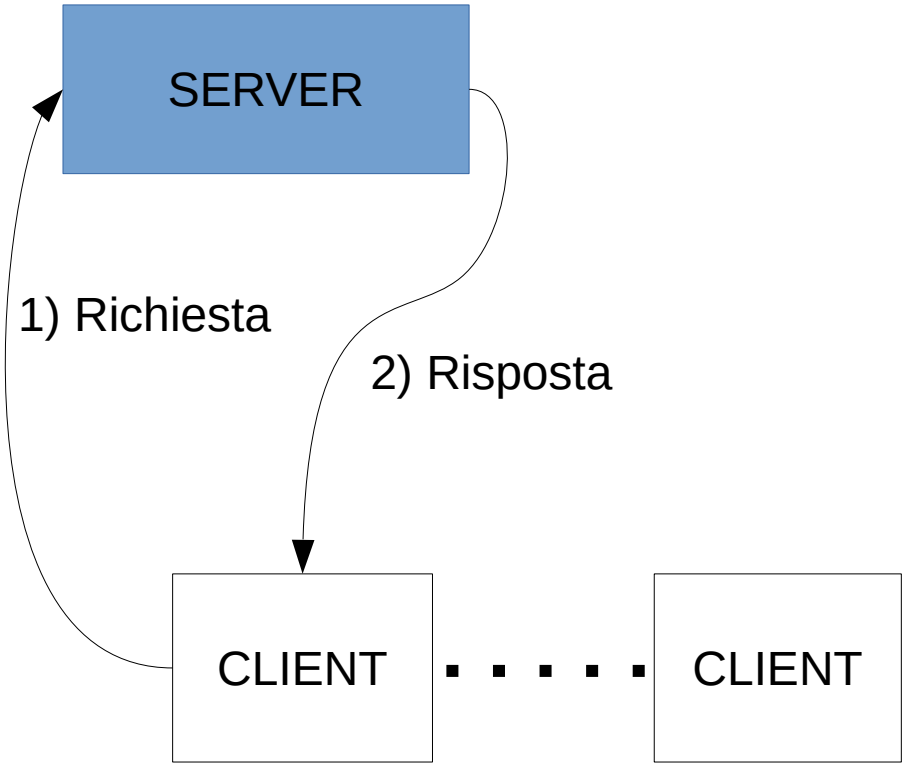
- Il sistema operativo deve introdurre nei pacchetti un numero di sequenza e rilevare eventuali pacchetti persi per poterli ritrasmettere
- Vantaggi
  - L'utente scrive/legge su un archivio remoto con la stessa naturalezza di quando scrive/legge su un archivio locale come se la rete in mezzo non ci fosse
- Svantaggi
  - Gli host trasmettitore e ricevitore devono "lavorare" di più dentro il sistema operativo
  - Può nascere un maggior ritardo di trasmissione in caso di ritrasmissione di pacchetti persi

Schemi di applicazioni che usano la rete

- Le applicazioni di rete sono insiemi di processi su host diversi che si scambiano messaggi attraverso la rete
- Esistono degli schemi base che regolano lo scambio di messaggi:
  - Client/server
  - Publisher/Subscriber (Pub/Sub)

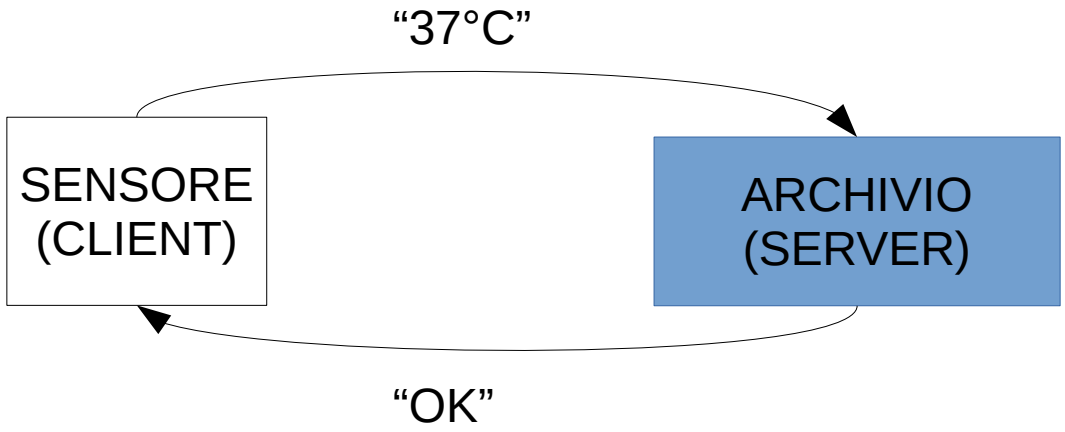
Modello client/server

- E’ la situazione più frequente
- Il client fa sempre il **primo passo** con una **richiesta**
- Il server fa il **secondo passo** e manda la **risposta** e poi si rimette in attesa di altre richieste
- NOTA: La richiesta del client può essere una richiesta di un dato oppure la trasmissione di un dato (cioè la richiesta di prendere in consegna un certo dato); quello che determina il ruolo di client e server è l’ordine dei messaggi e non il contenuto



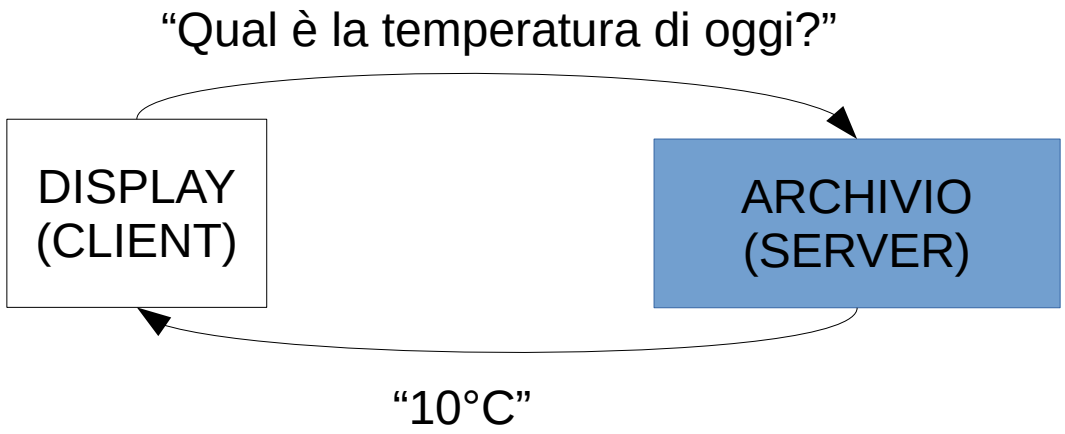
Esempio di applicazione client/server

- Il sensore di temperatura corporea funge da client e **manda** al server una temperatura
- Il server risponde con un “OK”



Altro esempio di applicazione client/server

- Il display funge da client e **chiede** al server una temperatura
- Il server risponde con il dato



Attenzione ai nomi!

- Client e server sono processi e non host
- L'insieme di almeno un client e un server costituisce l'applicazione di rete
  - In particolare si può dire che un certo processo "gioca" il ruolo di client o server all'interno dell'applicazione
- In un'applicazione client/server **il client è colui che fa il primo passo** indipendentemente dal fatto che chieda o trasmetta un dato

Preparazione alla parte pratica

- Imparare ad aprire ed usare il terminale o shell del PC
- Installare ed imparare ad usare un editor di file di testo per programmatori
- Installare ed imparare ad usare GCC per compilare programmi scritti in C
  - In Windows si può usare Cygwin

Consultare documento di istruzioni pubblicato a parte!

Primo esempio di client

```
#include "network.h"

int main(void) {
    socketif_t socket;
    char request[]="Ciao sono il client!\n";
    char response[MTU];
    char hostAddress[MAXADDRESSLEN];
    int port;

    socket = createUDPInterface(20000);
    UDPSend(socket, request, strlen(request), "127.0.0.1", 35000);
    UDPReceive(socket, response, MTU, hostAddress, &port);
    printf("[CLIENT] Ho ricevuto un messaggio da host/porta %s/%d\n", hostAddress, port);
    printf("[CLIENT] Contenuto: %s\n", response);
}
```

Primo esempio di client

```
#include "network.h"

int main(void) {
    socketif_t socket;
    char request[]="Ciao sono il client!\n";
    char response[MTU];
    char hostAddress[MAXADDRESSLEN];
    int port;

    socket = createUDPInterface(20000);
    UDPSend(socket, request, strlen(request), "127.0.0.1", 35000);
    UDPReceive(socket, response, MTU, hostAddress, &port);
    printf("[CLIENT] Ho ricevuto un messaggio da host/porta %s/%d\n", hostAddress, port);
    printf("[CLIENT] Contenuto: %s\n", response);
}
```

Istanziazione dell'interfaccia socket

Invio/ricezione di dati

Pseudocodice client

1. INIZIO
2. Dichiarazione delle variabili tra cui la variabile socket
3. Inizializzazione dell'interfaccia socket sulla porta 20000
4. Invia tramite UDP il messaggio "Ciao sono il client!" al destinatario avente indirizzo IP "127.0.0.1" e porta 35000
5. Mettiti in attesa dell'arrivo di un messaggio UDP
6. Scrivi "[CLIENT] Ho ricevuto un messaggio da"
7. Scrivi host/porta del mittente e contenuto ricevuto
8. FINE

Primo esempio di server

```
#include "network.h"

int main(void) {
    socketif_t socket;
    char response[]="Sono il server: ho ricevuto correttamente il tuo messaggio!\n";
    char request[MTU];
    char hostAddress[MAXADDRESSLEN];
    int port;

    socket=createUDPInterface(35000);
    printf("[SERVER] Sono in attesa di richieste da qualche client\n");
    UDPReceive(socket, request, MTU, hostAddress, &port);
    printf("[SERVER] Ho ricevuto un messaggio da host/porta %s/%d\n", hostAddress, port);
    printf("[SERVER] Contenuto: %s\n", request);
    UDPSend(socket, response, strlen(response), hostAddress, port);
}
```

Primo esempio di server

```
#include "network.h"

int main(void) {
    socketif_t socket;
    char response[]="Sono il server: ho ricevuto correttamente il tuo messaggio!\n";
    char request[MTU];
    char hostAddress[MAXADDRESSLEN];
    int port;

    socket=createUDPInterface(35000);
    printf("[SERVER] Sono in attesa di richieste da qualche
    UDPReceive(socket, request, MTU, hostAddress, &port);
    printf("[SERVER] Ho ricevuto un messaggio da host/porta %s/%d      ostAddress, port);
    printf("[SERVER] Contenuto: %s\n", request);
    UDPSend(socket, response, strlen(response), hostAddress, port);
}
```

Istanziazione dell'interfaccia socket

Invio/ricezione di dati

Pseudocodice server

1. INIZIO
2. Dichiarazione delle variabili tra cui la variabile socket
3. Inizializzazione dell'interfaccia socket sulla porta 35000
4. Scrivi "[SERVER] Sono in attesa di richieste da qualche client"
5. Mettiti in attesa dell'arrivo di un messaggio UDP
6. Scrivi "[SERVER] Ho ricevuto un messaggio da"
7. Scrivi host/porta del mittente e contenuto ricevuto
8. Invia a tale mittente tramite UDP il messaggio "Sono il server: ho ricevuto correttamente il tuo messaggio!"
9. FINE

Istruzioni per eseguire l'esempio

- Creare una cartella e mettere dentro: network.h, network.c, serverUDP.c, clientUDP.c
  - ATTENZIONE: chi usa l'ambiente Cygwin sotto Windows deve copiare i file nella propria "home" di Cygwin come mostrato nel documento di istruzioni
- Aprire due finestre di terminale e posizionarsi nella cartella creata
- In una delle due finestre compilare il server
  - `gcc network.c serverUDP.c -o serverUDP -lpthread`
- Nell'altra finestra compilare il client
  - `gcc network.c clientUDP.c -o clientUDP -lpthread`
- In una finestra eseguire il server
  - `./serverUDP`
  - ATTENZIONE che in Windows tutti gli eseguibili hanno il nome del file che termina con ".exe"
- Nell'altra finestra eseguire il client
  - `./clientUDP`

Come scrivere le applicazioni di rete

- Tutti i sistemi operativi forniscono **un'interfaccia software** per costruire questi programmi con un qualsiasi linguaggio di programmazione, ad es:
  - C
  - Java
  - Python
- Questa interfaccia si chiama **socket**

Fase 1: creare l'interfaccia Socket

- Il programma prima di usare la rete deve creare un oggetto di tipo socket

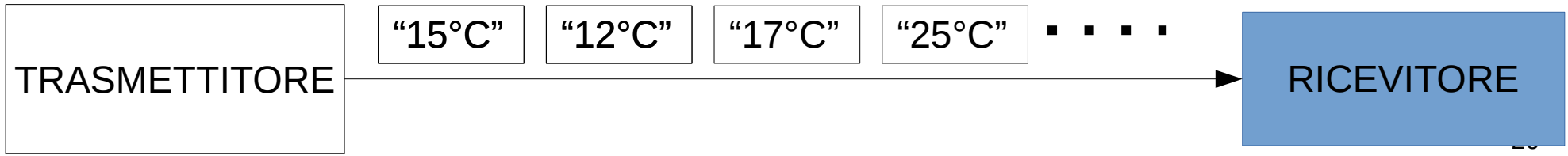
- Il socket è identificato da 3 parametri
  - Indirizzo IP locale
  - Porta locale
  - Modalità di trasmissione: UDP oppure TCP

Decidere il numero di porta locale

- Intero senza segno su 16 bit
  - Da 0 a 65535
- Il programma **server** deve decidere esplicitamente il numero di porta locale affinché i client possano saperlo
  - I numeri da 0 a 1023 sono riservati a protocolli applicativi noti
    - Ad es. web → 80 https → 443
    - Il programma che crea un oggetto socket su queste porte deve avere i privilegi di root
- Il programma **client** può deciderlo esplicitamente oppure lasciarlo decidere al proprio sistema operativo

UDP: applicazioni orientate al datagram

- Ogni pacchetto scambiato tra gli host è **logicamente indipendente** dai precedenti e successivi
- Le perdite di pacchetti non vengono compensate in automatico nè dalla rete nè dal sistema operativo.
- Per questo motivo l'UDP è più leggero del TCP come uso di risorse di calcolo e di rete ma il tipo di applicazione deve essere compatibile con questo tipo di comportamento
  - Es. trasmissione di temperature



Fase 2: la trasmissione/ricezione

- Serve conoscere 2 ulteriori parametri
  - IP del proprio interlocutore
  - Porta del proprio interlocutore
- Se siamo nel caso UDP
  - Chiamata a funzione per trasmettere
  - Chiamata a funzione per ricevere

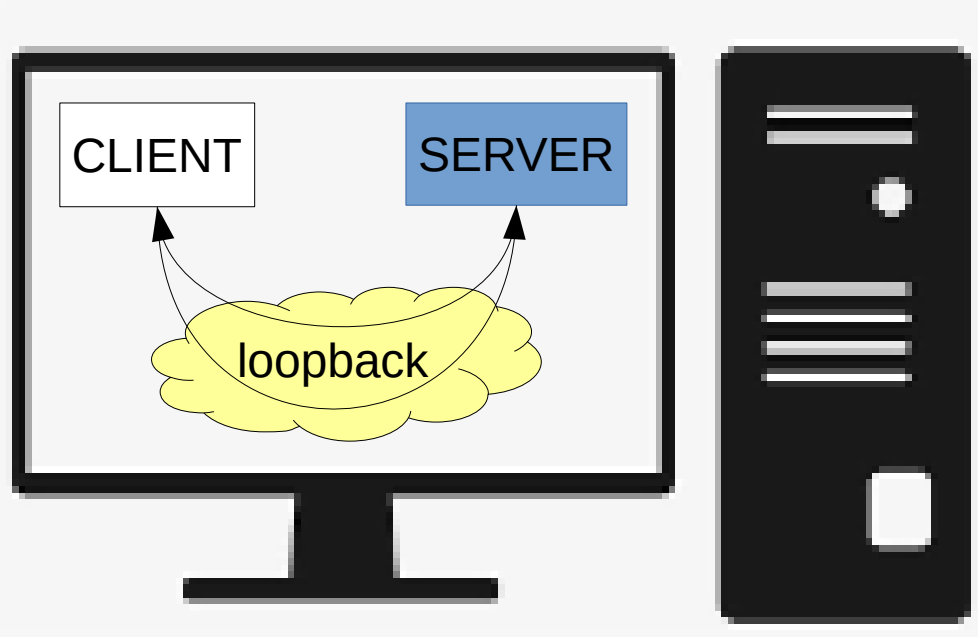
Scoprire l'indirizzo IP del proprio interlocutore

- Lato client all'utente basta sapere il **nome Internet** del server perché il sistema operativo automaticamente ricava il suo indirizzo IP (risoluzione del nome) mediante DNS
  - Se il server non ha nome occorre chiedere ad un utente sul server di scoprire e comunicarci il suo indirizzo IP mediante un comando da terminale
    - In Linux e MAC OS: ifconfig -a
    - In Windows: ipconfig /all

- Lato server, siccome una risposta parte sempre dopo la relativa richiesta, l'indirizzo IP del client è contenuto nelle informazioni di mittente della richiesta

Il nome “localhost” oppure 127.0.0.1

- Per scopi particolari (ad es. didattici) client e server potrebbero girare sulla stessa macchina
- In questo caso non serve una rete fisica vera ma viene usata una rete fittizia chiamata “loopback”
- In tal caso l’interfaccia socket sia del client sia del server hanno indirizzo IP = 127.0.0.1 e nome “localhost”
- Posso testare client e server sul mio PC prima di distribuirli veramente sulla rete



Scoprire la porta del proprio interlocutore

- Lato client, la porta del server deve essere resa nota
  - Colui che ha programmato il server la rende pubblicamente nota
  - Si può ricavare dal tipo di protocollo/programma, ad es:
    - Web → porta 80
    - HTTPS → porta 443
- Lato server, siccome una risposta parte sempre dopo la relativa richiesta, la porta del client è contenuta nelle informazioni del mittente della richiesta

Esercizi basati su clientUDP e serverUDP

1) Lanciare prima il server e poi il client. Cosa si osserva? Invertire la sequenza di lancio. Cosa si osserva?
2) Modificare i sorgenti per mettere il server che **riceve** sulla porta 10000 e il client che **trasmette** dalla propria porta 30000 (ogni modifica dei sorgenti richiede una loro ricompilazione)
3) Mettere il server in ascolto sulla porta 100 e osservare cosa succede
   - Bisogna modificare anche il client? Dove?
   - Per chi usa il proprio PC con Linux o una virtual machine Linux, lanciare il server con il comando "`sudo ./serverUDP`" e osservare cosa cambia
4) Sostituire "127.0.0.1" prima con "localhost" e poi con "pippo" e osservare cosa succede
5) [Da fare solo se in Lab Delta] Accordarsi per lavorare su coppie di macchine in modo che server e client siano su macchine diverse. Come bisogna modificare i sorgenti?
6) Lanciare due volte il server usando due terminali. Cosa si osserva? Funzionano entrambi?
7) Modificare il server in maniera che soddisfi 5 richieste prima di terminare
   - E se volessi che non terminasse mai?

Altro esempio UDP: clientUDP_inc

```
#include "network.h"

int main(void) {
    int request;
    int response;
    socketif_t socket;
    char hostAddress[MAXADDRESSLEN];
    int port;

    socket = createUDPInterface(20000);
    printf("Inserisci un numero intero:\n");
    scanf("%d", &request);
    UDPSend(socket, &request, sizeof(request), "127.0.0.1", 35000);
    UDPReceive(socket, &response, sizeof(response), hostAddress, &port);
    printf("[CLIENT] Ho ricevuto un messaggio da host/porta %s/%d\n", hostAddress, port);
    printf("[CLIENT] Contenuto: %d\n", response);
}
```

Pseudocodice clientUDP_inc

1. INIZIO
2. Dichiarazione delle variabili tra cui la variabile socket e due interi
3. Inizializzazione dell'interfaccia socket sulla porta 20000
4. Scrivi "Inserisci un numero intero:"
5. Leggi request da tastiera
6. Invia request tramite UDP al destinatario avente indirizzo IP "127.0.0.1" e porta 35000
7. Mettiti in attesa dell'arrivo di un messaggio UDP e metti il valore ricevuto in response
8. Scrivi "[CLIENT] Ho ricevuto un messaggio da"
9. Scrivi host/porta del mittente
10. Scrivi response
11. FINE

Altro esempio UDP: serverUDP_inc

```
#include "network.h"

int main(void) {
    int request;
    int response;
    socketif_t socket;
    char hostAddress[MAXADDRESSLEN];
    int port;

    socket=createUDPInterface(35000);
    printf("[SERVER] Sono in attesa di richieste da qualche client\n");
    UDPReceive(socket, &request, sizeof(request), hostAddress, &port);
    printf("[SERVER] Ho ricevuto un messaggio da host/porta %s/%d\n", hostAddress, port);
    printf("[SERVER] Contenuto: %d\n", request);
    response = request + 1;
    UDPSend(socket, &response, sizeof(response), hostAddress, port);
}
```

Pseudocodice serverUDP_inc

1. INIZIO
2. Dichiarazione delle variabili tra cui la variabile socket e due interi
3. Inizializzazione dell'interfaccia socket sulla porta 35000
4. Scrivi "[SERVER] Sono in attesa di richieste da qualche client"
5. Mettiti in attesa dell'arrivo di un messaggio UDP e metti il valore ricevuto in request
6. Scrivi "[SERVER] Ho ricevuto un messaggio da"
7. Scrivi host/porta del mittente
8. Scrivi request
9. response=request+1
10. Invia response a tale mittente tramite UDP
11. FINE

Esercizi su UDP

7) Compilare ed eseguire il secondo esempio
8) Modificarlo per costruire una semplice sommatrice
   - Il client acquisisce ripetutamente da tastiera un numero intero e lo manda al server finché l'utente digita zero
   - Il server accumula in una variabile "somma" i valori mandati dal client finché il client manda zero
   - Quando il client manda zero il server risponde al client con la somma ottenuta

Esercizio 9: sommatrice UDP e perdita di pacchetti [Da fare solo con i PC del Lab Delta]

- Usare la sommatrice su due macchine distinte provando, sulla macchina del client, a staccare il cavo di rete prima di un invio di un dato, ad es.
  - Digitare “2345” + INVIO
  - Digitare “5187” + INVIO
  - **Staccare il cavo**
  - Digitare “2” + INVIO
  - **Riattaccare il cavo e aspettare 30 sec che il sistema operativo si riassesti**
  - Digitare “1” + INVIO
  - “0”
- Che somma leggo? E’ corretta?

Esercizio 10: sommatrice UDP e influenze reciproche

- Invocare il server della sommatrice con due client diversi (tutti e tre possono anche essere sulla stessa macchina ovviamente su finestre terminali diverse), ad es.

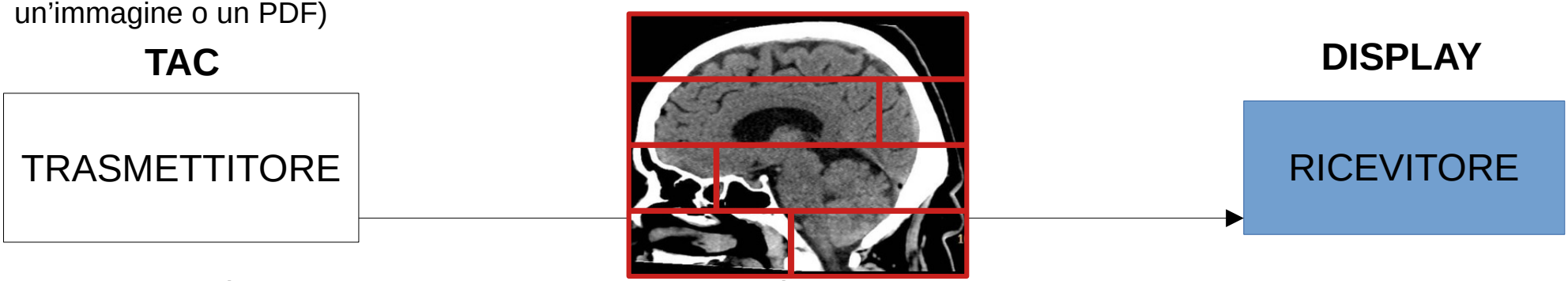
Client A:
- Digitare “2345” + INVIO
- Digitare “5187” + INVIO
- Digitare “2” + INVIO
- Digitare “1” + INVIO
- “0”

Client B:
- Digitare “2” + INVIO
- Digitare “8” + INVIO
- “0”

- Che somma leggo da ciascun client? E’ la somma che ciascun client **da solo** si aspetterebbe?

TCP: applicazioni orientate alla connessione

- E' utilizzata quando tra i pacchetti trasmessi c'è una relazione: sono parte di un **"messaggio più grande"** (ad es. un'immagine o un PDF)



- Il "messaggio più grande" deve essere incluso in una entità logica chiamata **connessione**
- L'oggetto socket si preoccupa di numerare i pacchetti appartenenti alla stessa connessione per rilevare eventuali pacchetti persi e poterli ritrasmettere
- Vantaggi
  - L'utente scrive/legge su un archivio remoto con la stessa naturalezza di quando scrive/legge su un archivio locale come se la rete in mezzo non ci fosse
- Svantaggi
  - Gli host trasmettitore e ricevitore devono "lavorare" di più dentro il sistema operativo
  - I pacchetti persi e ritrasmessi arrivano in ritardo e questo può essere compatibile oppure no con il tipo di applicazione

Fase 2 (trasmissione/ricezione) in caso TCP

- Se usiamo TCP
  - Chiamata a funzione per aprire la connessione
  - Chiamata a funzione per trasmettere
  - Chiamata a funzione per ricevere
  - Chiamata a funzione per chiudere la connessione

Esempio TCP: client

```
#include "network.h"

int main(void) {
    connection_t connection;
    int request, response;

    printf("[CLIENT] Creo una connessione logica col server\n");
    connection = createTCPConnection("localhost", 35000);
    if (connection < 0) {
        printf("[CLIENT] Errore nella connessione al server: %i\n", connection);
    }
    else
    {
        printf("[CLIENT] Inserisci un numero intero:\n");
        scanf("%d", &request);
        printf("[CLIENT] Invio richiesta con numero al server\n");
        TCPSend(connection, &request, sizeof(request));
        TCPReceive(connection, &response, sizeof(response));
        printf("[CLIENT] Ho ricevuto la seguente risposta dal server: %d\n", response);
        closeConnection(connection);
    }
}
```

Esempio TCP: client

```
#include "network.h"

int main(void) {
    connection_t connection;
    int request, response;

    printf("[CLIENT] Creo una connessione logica col server\n");
    connection = createTCPConnection("localhost", 35000);
    if (connection < 0) {
        printf("[CLIENT] Errore nella connessione al server: %i\n", connection);
    }
    else
    {
        printf("[CLIENT] Inserisci un numero intero:\n");
        scanf("%d", &request);
        printf("[CLIENT] Invio richiesta con numero al server\n");
        TCPSend(connection, &request, sizeof(request));
        TCPReceive(connection, &response, sizeof(response));
        printf("[CLIENT] Ho ricevuto la seguente risposta dal server: %d\n", response);
        closeConnection(connection);
    }
}
```

Istanziazione dell'interfaccia socket e apertura della connessione TCP (three-way handshake)

Invio/ricezione di dati

Chiusura della connessione TCP

Pseudocodice client TCP

1. INIZIO
2. Dichiarazione delle variabili tra cui la variabile connessione e due interi
3. Instaurazione della connessione verso il server destinatario avente nome "localhost" e porta 35000
4. SE c'è stato errore ALLORA
   1. Scrivi "Errore nella connessione al server"
5. ALTRIMENTI
   1. Scrivi "Inserisci un numero intero:"
   2. Leggi request da tastiera
   3. Invia request tramite la connessione TCP
   4. Mettiti in attesa dell'arrivo di un messaggio dalla connessione TCP e metti il valore ricevuto in response
   5. Scrivi "[CLIENT] Ho ricevuto un messaggio dal server"
   6. Scrivi response
   7. Chiudi la connessione
6. FINE SE
7. FINE

Esempio TCP: server

```
#include "network.h"

int main(void) {
    int request, response;
    socketif_t socket;
    connection_t connection;

    socket = createTCPServer(35000);
    if (socket < 0){
        printf("[SERVER] Errore di creazione del socket: %i\n", socket);
    }
    else
    {
        printf("[SERVER] Sono in attesa di richieste di connessione da qualche client\n");
        connection = acceptConnection(socket);
        printf("[SERVER] Connessione instaurata\n");
        TCPReceive(connection, &request, sizeof(request));
        printf("[SERVER] Ho ricevuto il seguente messaggio dal client: %d\n", request);
        response = request + 1;
        printf("[SERVER] Invio la risposta al client\n");
        TCPSend(connection, &response, sizeof(response));
        closeConnection(connection);
    }
}
```

Esempio TCP: server

```
#include "network.h"

int main(void) {
    int request, response;
    socketif_t socket;
    connection_t connection;

    socket = createTCPServer(35000);
    if (socket < 0){
        printf("[SERVER] Errore di creazione del sock
    }
    else
    {
        printf("[SERVER] Sono in attesa di richieste di conn                da qualche client\n");
        connection = acceptConnection(socket);
        printf("[SERVER] Connessione instaurata\n");
        TCPReceive(connection, &request, sizeof(request));
        printf("[SERVER] Ho ricevuto il seguente messaggio dal client: %d\n", request);
        response = request + 1;
        printf("[SERVER] Invio la risposta al client\n");
        TCPSend(connection, &response, sizeof(response));
        closeConnection(connection);
    }
}
```

Istanziazione dell'interfaccia socket

Accettazione all'apertura della connessione TCP (three-way handshake)

Invio/ricezione di dati

Chiusura della connessione TCP

Pseudocodice server TCP

1. INIZIO
2. Dichiarazione delle variabili tra cui la variabile socket, la variabile connessione e due interi
3. Inizializzazione dell'interfaccia socket sulla porta 35000
4. SE c'è stato errore ALLORA
   1. Scrivi "Errore nella creazione del socket"
5. ALTRIMENTI
   1. Scrivi "[SERVER] Sono in attesa di richieste da qualche client"
   2. Accetta la richiesta di connessione
   3. Scrivi "[SERVER] Connessione instaurata"
   4. Mettiti in attesa dell'arrivo di un messaggio TCP e metti il valore ricevuto in request
   5. Scrivi "[SERVER] Ho ricevuto il seguente messaggio dal client"
   6. Scrivi request
   7. response=request+1
   8. Invia response al client
   9. Chiudi la connessione
6. FINE SE
7. FINE

Esercizi su TCP

11) Lanciare due volte il server usando due terminali. Cosa si osserva? Funzionano entrambi?

12) Scrivere la sommatrice usando TCP, compilare ed eseguire

13) Provare a rifare il caso dell'Esercizio 10 ma con questa nuova versione della sommatrice.

    Cosa si può osservare? Che soluzione si può trovare? C'è influenza reciproca tra i due client?

14) [Da fare solo con i PC del Lab Delta] Usare la sommatrice TCP su due macchine distinte provando, sulla macchina del client, a staccare il cavo di rete prima di un invio di un dato, ad es.
    - Digitare "2345" + INVIO
    - Digitare "5187" + INVIO
    - **Staccare il cavo**
    - Digitare "2" + INVIO
    - **Riattaccare il cavo e aspettare 30 sec che il sistema operativo si riassesti**
    - Digitare "1" + INVIO
    - "0"

    Che somma leggo? E' corretta?

I/O evoluto con TCP

```
#include "network.h"

int main(){
    char receive;
    socketif_t server;
    FILE *fd;

    server = createTCPServer(35000);
    if (server < 0){
        printf("Error: %i\n", server);
        return -1;
    }

    while(true){
        fd = acceptConnectionFD(server);

        receive=fgetc(fd);
        fflush(fd);
        printf("[server] %c\n", receive);

        fputc('B', fd);
        fflush(fd);

        fclose(fd);
    }

    closeConnection(server);
    return 0;
}
```

```
#include "network.h"

int main(){
    char receive;
    FILE *fd;

    fd = createTCPConnectionFD("localhost", 35000);

    fputc('A', fd);
    fflush(fd);

    receive=fgetc(fd);
    fflush(fd);
    printf("[client] %c\n", receive);

    fclose(fd);
}
```

I/O evoluto con TCP (2)

```
#include "network.h"

int main(){
    char receive[MTU];
    socketif_t server;
    FILE *fd;

    server = createTCPServer(35000);
    if (server < 0){
        printf("Error: %i\n", server);
        return -1;
    }

    while(1){
        fd = acceptConnectionFD(server);

        fgets(receive, sizeof(receive), fd);
        fflush(fd);
        printf("%s\n", receive);

        fputs("Hi from the server\n", fd);
        fflush(fd);

        fclose(fd);
    }

    return 0;
}
```

```
#include "network.h"

int main(){
    char receive[MTU];
    FILE *fd;

    fd = createTCPConnectionFD("localhost", 35000);

    fputs("Hi from the client\n", fd);
    fflush(fd);

    fgets(receive, sizeof(receive), fd);
    fflush(fd);
    printf("%s\n", receive);

    fclose(fd);
}
```

Trasferimento di un file

- Partiamo dal codice per la copia di un file in locale…
- ...lo modifichiamo per copiarlo da un host all'altro

Codice per la copia di un file

```
#include <stdio.h>
#include <stdlib.h> // For exit()

int main()
{
    FILE *fptr1, *fptr2;
    char filename[100], c;

    printf("Nome del file sorgente:\n");
    scanf("%s", filename);

    // Apro il file sorgente in lettura
    fptr1 = fopen(filename, "r");
    if (fptr1 == NULL)
    {
        printf("Errore apertura file %s \n", filename
        exit(0);
    }

    printf("Nome del file destinazione:\n");
    scanf("%s", filename);
```

```
    // Apro il file destinazione in scrittura
    fptr2 = fopen(filename, "w");
    if (fptr2 == NULL)
    {
        printf("Errore apertura file %s\n", filename);
        exit(0);
    }

    // Leggo il primo byte dalla sorgente
    c = fgetc(fptr1);
    while (c != EOF)
    {
        // Scrivo il byte nella destinazione
        fputc(c, fptr2);
        c = fgetc(fptr1);
    }

    printf("\nContenuto copiato su %s\n", filename);

    fclose(fptr1);
    fclose(fptr2);
    return 0;
}
```

Esercizio: applicazione di trasferimento file

- Il client chiede al server un file specificandone il nome
- Il server lo trasmette un byte alla volta
- Il client lo salva in locale con lo stesso nome

- Quale protocollo usiamo?
- [Da fare solo con i PC del Lab Delta] Lanciare client e server su due macchine diverse, trasferire un file di grosse dimensioni in modo da avere il tempo di staccare e riattaccare il cavo di rete. Cosa succede al file trasferito?

Temi d'esame (per ognuno ragionare sulla scelta del protocollo di livello trasporto)

- Implementare un servizio di calcolo remoto: il client spedisce al server in un unico messaggio due numeri interi e il tipo di operatore (somma, sottrazione, moltiplicazione, divisione) e il server, dopo aver svolto il calcolo, restituisce il risultato al client che così termina la sua esecuzione.
- Implementare un servizio di calcolo remoto: il client spedisce al server in 3 distinti messaggi due numeri interi e il tipo di operatore (somma, sottrazione, moltiplicazione, divisione) e il server, dopo aver svolto il calcolo, restituisce il risultato al client che così termina la sua esecuzione.
- Implementare un servizio di telepass: quando l'auto si avvicina al casello il terminale in esso presente spedisce al server il numero di targa; il server tiene traccia in una struttura dati del numero di passaggi per ogni targa e restituisce al client tale numero.
- Implementare una semplice versione di terminale remoto. Il client si connette ad un server di cui si conosce IP e porta e chiede all'utente di scrivere un comando come in un normale terminale a caratteri (detto anche shell o console) di Linux. Il comando viene eseguito dal server e il relativo output viene stampato a video dal client.