Sistemi Operativi e Reti di Calcolatori (SOReCa)

Corso di Laurea in *Ingegneria Informatica e Automatica (BIAR)*

Terzo Anno | Primo Semestre

A.A. 2024/2025

Esercitazione [07] Client/Server con Socket

Riccardo Lazzeretti lazzeretti@diag.uniroma1.it
Paolo Ottolino paolo.ottolino@uniroma1.it
Edoardo Liberati e.liberati@diag.uniroma1.it

DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA UNIVERSITÀ DI ROMA

Sommario

- Soluzioni precedente esercitazione
- TCP:
  - Byte Order: network vs host
  - Client/Server
  - Protocollo: messaggi testuali
- Esercizio
  - EchoServer
- UDP
  - EchoServer

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

TCP: Funzioni Primitive

Byte Order: Network, Host
Client/Server: Socket
Protocollo: messaggi

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Obiettivi

1. Capire la differenza tra network byte order e host byte order

2. Imparare ad impostare un'applicazione client/server che preveda:
   - Server single-thread
     - Come mettersi in ascolto su una porta nota?
     - Come accettare una connessione da client?
   - Client
     - Come connettersi ad un server in ascolto?

3. Semplice protocollo basato su messaggi testuali

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Funzioni Primitive

Byte Order: network e host

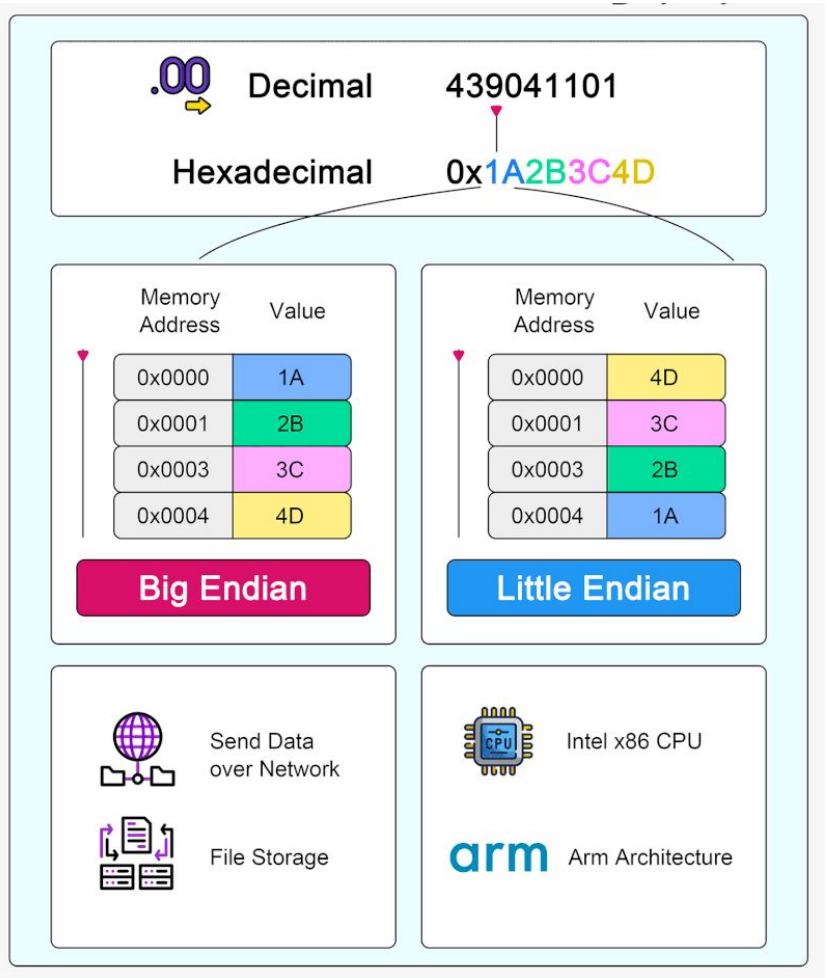
Nello scambio di dati numerici tra macchine con architetture (potenzialmente) differenti, occorre verificare il **byte order**

- Un dato numerico è rappresentato come una sequenza di byte
- *Il primo byte di tale sequenza è il più significativo* (**Big Endian**) *o il meno significativo* (**Little Endian**)*?*

- Dati numerici scambiati tra macchine che usano byte order diversi (**host byte order**) vengono interpretati in maniera diversa

- La maggior parte dei protocolli di rete (inclusi IPv4 e TCP) usano Big Endian come **network byte order** nell'header

- La maggior parte delle architetture CPU (Intel, AMD, ARM) sono Little Endian come **host byte order**



Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

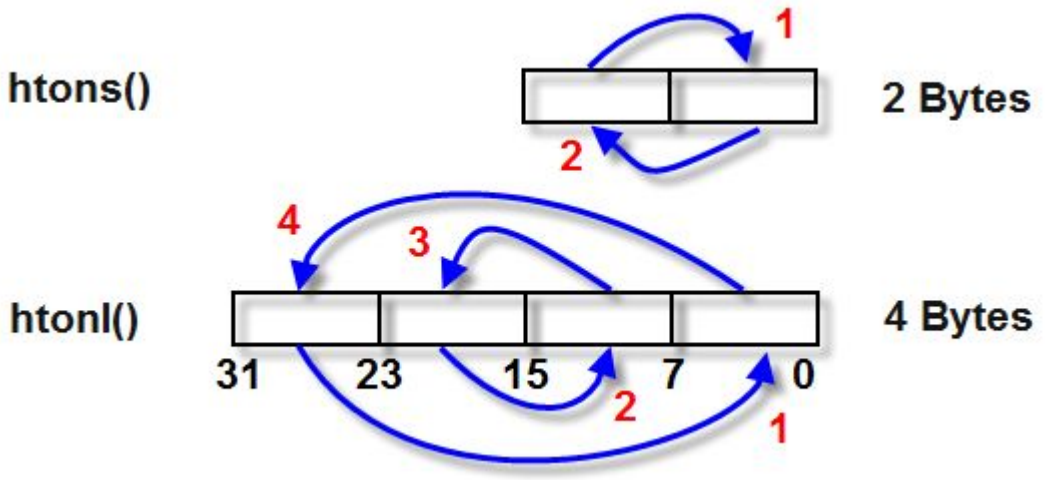
Sapienza Università di Roma

Funzioni Primitive

Byte Order: Funzioni di conversion per la porta

Il numero di porta di una socket TCP può variare tra 0 e 65535

- la definizione del tipo generico `uint16_t` può essere inclusa tramite **`<arpa/inet.h>`** o più in generale `<stdint.h>`
- su Linux IA32 e x86_64 equivale ad un `unsigned short`
- le porte nel range 0-1023 richiedono privilegi di root



- Funzione `htons()` (host-to-network-ushort)
  - `uint16_t htons(uint16_t hostshort);`
  - Converte un ushort da host byte order a network byte order

- Funzione `ntohs()` (network-to-host-ushort)
  - `uint16_t ntohs(uint16_t netshort);`
  - Converte un ushort da network byte order a host byte order

Funzioni Primitive

Byte Order: Funzioni di conversion per l'Indirizzo

Un indirizzo IPv4 è rappresentato con `struct in_addr`
Il campo `sin_addr` di `struct sockaddr_in` è infatti di tipo `struct in_addr`

- *Reminder*: variabili di tipo `struct sockaddr_in` vengono usate nella `bind()` e nella `accept()` lato server e nella `connect()` lato client

`struct in_addr` contiene il campo `s_addr` di tipo `in_addr_t` che rappresenta l'indirizzo in network byte order
Entrambe le strutture sono definite in `<netinet/in.h>`

```
in_addr_t inet_addr(const char *cp)
```

- Converte un indirizzo IPv4 dalla forma dotted string (x.y.z.w) al network byte order
- Il valore di ritorno viene di solito assegnato al campo `s_addr` di `struct in_addr`

```
const char *inet_ntop(int af, const void *src, char *dst, socklen_t n);
```

- Converte l'indirizzo di rete `src` della address family `af` in una stringa di lunghezza `n` e la copia in `dst`
- Ritorna un puntatore a `dst`, oppure `NULL` in caso di errore
- Le macro `AF_INET` e `INET_ADDRSTRLEN` possono essere usate rispettivamente per il primo e l'ultimo argomento
  - Quanto vale `INET_ADDRSTRLEN`?

Funzioni Primitive

Socket (Server, Client)

Come mettersi in ascolto su una porta nota?

- Creazione socket - funzione `socket()`
- Binding della socket su un indirizzo locale - funzione `bind()`
- Infine, mettersi in ascolto - funzione `listen()`

Come accettare una connessione da client?

- Attesa di una connessione - funzione `accept()`
- Una volta accettata una connessione, si ha a disposizione un descrittore di socket da usare per scambiare messaggi (tramite `send()`/`recv()`)
- Una volta terminato lo scambio di messaggi, la connessione col client va chiusa - funzione `close()`

Funzioni Primitive

Socket (Server, Client): Strutture Dati

`struct in_addr`: rappresenta un indirizzo IP a 32 bit

`struct sockaddr_in`: descrizione di una socket; al suo interno le informazioni principali sono:

- Famiglia dell'indirizzo (`sin_family`)
  - Per i nostri scopi, `AF_INET`: protocollo IPv4
  - Ne esistono altre, es: `AF_UNIX`, `AF_BLUETOOTH`
- Indirizzo IP (`sin_addr.s_addr`), per i nostri scopi:
  - Lato server, `INADDR_ANY`: in ascolto su tutte le interfacce
  - Lato client, specifica l'indirizzo IP del server
- Numero porta (`sin_port`)
  - Bisogna rispettare l'ordine di trasmissione dei byte per la rete
  - `sin_port = htons(port)` per invertire l'ordine dei bytes

Funzioni Primitive

Socket (Server, Client): funzione `socket()`

```
int socket(int family, int type, int protocol);
```

Crea una socket, ossia un endpoint di comunicazione

Argomenti

- `family`: per i nostri scopi, `AF_INET`
  (vedi struttura dati `struct sockaddr_in`)
- `type`: per i nostri scopi, `SOCK_STREAM` (protocollo TCP)
  - Ne esistono altre, es: `SOCK_DGRAM` (protocollo UDP)
- `protocol`: per i nostri scopi, `0`

Valore di ritorno

- In caso di successo, il descrittore della socket
- In caso di errore, `-1`, `errno` è settato

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Funzioni Primitive

Socket (Server): funzione `bind()`

```
int bind(int fd, const struct sockaddr *addr, socklen_t len);
```

Assegna un indirizzo ad una socket

Argomenti

- `fd`: descrittore della socket (restituito da `socket()`)
- `addr`: puntatore ad una struttura dati che specifica l'indirizzo
  - Per i nostri scopi: per usare i valori della struttura `struct sockaddr_in` va effettuato il cast a `struct sockaddr`
- `len`: dimensione della struttura dati puntata da `addr`

Valore di ritorno

- In caso di successo, `0`
- In caso di errore, `-1` , `errno` è settato

Funzioni Primitive

Socket (Server): funzione `listen()`

```
int listen(int sockfd, int backlog);
```

Marca la socket come passiva, i.e., specifica che può essere usata per accettare connessioni tramite la funzione `accept()`

Argomenti

- `sockfd`: descrittore della socket (restituito da `socket()`)
- `backlog`: lunghezza massima della coda per le connessioni
  - Se una connessione arriva quando la coda è piena, la connessione viene rifiutata

Valore di ritorno

- In caso di successo, `0`
- In caso di errore, `-1` , `errno` è settato

Funzioni Primitive

Socket (Server): funzione `accept()`

```
int accept(int fd, struct sockaddr *addr, socklen_t *len);
```

Accetta una connessione su una socket in ascolto

- È una chiamata bloccante: rimane in attesa di connessioni

Argomenti

- `fd`: descrittore della socket (restituito da `socket()`)
- `addr`: puntatore ad una struttura dati `struct sockaddr` che verrà riempita con le info della socket del client
- `len`: puntatore ad un intero che verrà settato con la dimensione della struttura dati `addr`

Valore di ritorno

- In caso di successo, un descrittore per comunicare col client
- In caso di errore, `-1`, `errno` è settato

Funzioni Primitive

Socket (Server, Client): funzione `close()`

```
int close(int fd);
```

Nel caso `fd` sia un descrittore di socket, chiude la socket stessa

- `read()` *successive dall'altro endpoint restituiranno* `0` *!!!*

Argomenti

- `fd`: descrittore della socket (ritornato da `socket()`)

Valore di ritorno

- In caso di successo, `0`
- In caso di errore, `-1` , `errno` è settato

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Funzioni Primitive

Socket (Client)

Come connettersi ad un server in ascolto?

- Creazione socket - funzione `socket()`
- Connessione al server - funzione `connect()`
- Una volta terminato lo scambio di messaggi, la connessione col client va chiusa - funzione `close()`

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Funzioni Primitive

Socket (Client): funzione `connect()`

```
int connect(int fd, const struct sockaddr *addr, socklen_t l);
```

Tenta una connessione su una socket in ascolto

Argomenti

- `fd`: descrittore della socket (ritornato da `socket()`)
- `addr`: puntatore ad una struttura dati `struct sockaddr` che descrive la socket alla quale connettersi (quella del server)
- `l`: dimensione della struttura dati puntata da `addr`

Valore di ritorno

- In caso di successo, `0`
- In caso di errore, `-1` , `errno` è settato

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Funzioni Primitive

Protocollo con messaggi testuali

Implementazione un protocollo client-server basato su messaggi di testo

- Il server è in ascolto su una porta nota
- Il client si connette al server
- Inizia uno scambio di messaggi di testo secondo uno schema predefinito («protocollo»)
- Protocollo di base
  - Il client invia una richiesta al server
  - Il server riceve la richiesta, la elabora, produce una risposta
  - Il server invia la risposta al client

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

EchoServer

Lab07, es1

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Invio e Ricezione messaggi su Socket

Lab07 es1: EchoServer

Server single-thread in ascolto su una porta nota

Il client si connette al server:

1. Il server invia un welcome message
2. L'utente inserisce da terminale un messaggio
3. Il client invia il messaggio inserito al server
4. Se il messaggio inviato dal client è «QUIT», entrambi terminano la connessione.
5. In caso contrario, il server risponde con lo stesso messaggio ricevuto. Entrambi ripartono dal punto 2.

Sorgenti: `client.c` e `server.c`

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Invio e Ricezione messaggi su Socket

Lab07 es1: EchoServer (Client)

Esercizio (lato client)

- Completare le parti mancanti, relative a:
  - Creazione e distruzione socket
  - Instaurare una connessione con il server
  - Invio/ricezione di messaggi via socket (gestire letture/scritture parziali)
    - Attenzione: non conosciamo la dimensione del messaggio
- Per l'esecuzione, lanciare `prof_server` e `client` su terminali diversi

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Invio e Ricezione messaggi su Socket

Lab07 es1: EchoServer (Server)

Esercizio (lato server)

- Completare le parti mancanti, relative a:
  - Creazione, apertura e distruzione socket
  - Accettare una connessione in ingresso
  - Invio/ricezione di messaggi via socket (gestire letture/scritture parziali)
    - Attenzione: non conosciamo la dimensione del messaggio

- Per l'esecuzione, lanciare `server` e `prof_client` su terminali diversi

- Se tutto funziona, lanciare `server` e `client` su terminali diversi

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

UDP: Funzioni Primitive

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

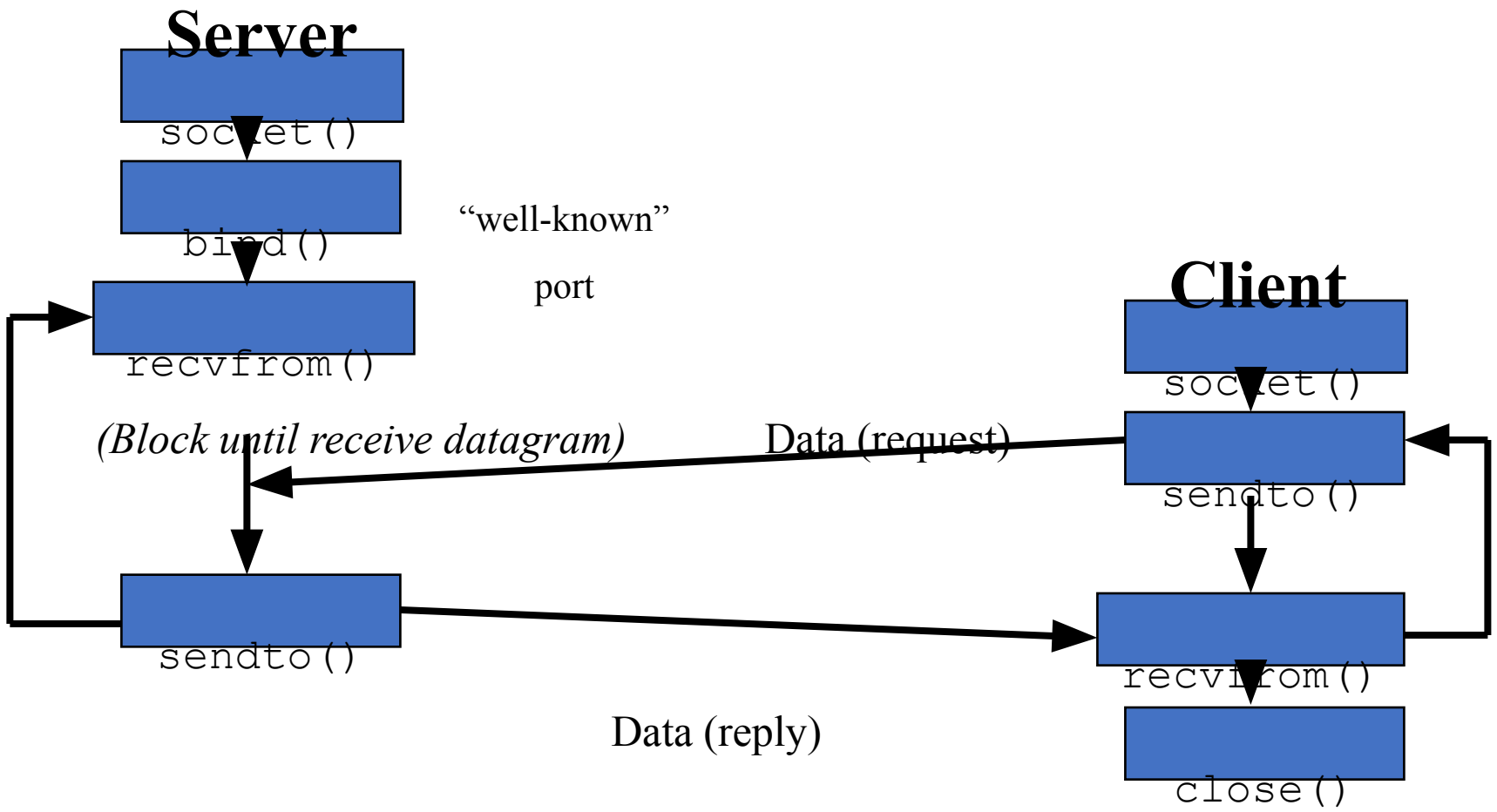
Funzioni Primitive

UDP: Client / Server

No “handshake”

- No simultaneous close

- No `fork()` for concurrent servers!



Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Funzioni Primitive

Socket (Server, Client): funzione `socket()`

```
int socket(int family, int type, int protocol);
```

Crea una socket, ossia un endpoint di comunicazione

Argomenti

- `family`: per i nostri scopi, `AF_INET`
  (vedi struttura dati `struct sockaddr_in`)
- `type`: per i nostri scopi, `SOCK_STREAM` (protocollo TCP)
  - Ne esistono altre, es: `SOCK_DGRAM` (protocollo UDP)
- `protocol`: per i nostri scopi, `0`

Valore di ritorno

- In caso di successo, il descrittore della socket
- In caso di errore, `-1`, `errno` è settato

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Funzioni Primitive

Socket (invio messaggi): funzione `sendto()`

La funzione `sendto()` è definita in `sys/socket.h`

```
ssize_t sendto(int fd, const void *buf, size_t n, int flags,        const struct
sockaddr *dest_addr, socklen_t l );
```

- `fd, buf, n, flags`: come in `send()`
- `dest_addr, l`: come in `connect()`

Ritorna il numero di byte realmente scritti, o `-1` in caso di errore

- Default: semantica **bloccante**
  - Se buffer di invio nel kernel non contiene spazio sufficiente per il messaggio da inviare, rimane bloccata in attesa…

Funzioni Primitive

Socket (ricezione messaggi): funzione `recvfrom()`

La funzione `recvfrom()` è definita in `sys/socket.h`

```
ssize_t recv(int fd, void *buf, size_t n, int flags,
struct sockaddr *src_addr, socklen_t *addrlen );
```

- `fd, buf, n, flags`: come in recv()
- Se `src_addr` è una variabile la funzione inserisce le informazione del mittente nella variabile e inserisce in `addrlen` la lunghezza della struttura
  - Utile per rispondere o distinguere tra più possibili mittenti
- Se `src_addr` è `NULL`, non salva il mittente, `addrlen` non viene modificato e può essere `NULL`
  - Utile quando non ci interessa sapere chi ha inviato il messaggio

Ritorna il numero di byte realmente letti, o `-1` in caso di errore

- Ritorna `0` in caso di connessione chiusa
- Default: semantica **bloccante**
  - Se l'altro endpoint non invia nulla, rimane bloccata in attesa
  - Nel client andrebbe modificata la socket con un timeout, ma tralasciamo questo aspetto
  - Trasferisce i dati disponibili fino a quel momento nel buffer del kernel, entro il limite di `n` bytes, piuttosto che restare in attesa di ricevere l'intera quantità specificata…

EchoServer UDP

Lab07, es2

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma

Invio e Ricezione messaggi su Socket UDP

Lab07 es2: EchoServer

Server single-thread in ascolto su una porta nota

Modificare il codice dell'esercizio precedente per supportare una connessione UDP

Sorgenti: `client.c` e `server.c`

Dipartimento di Ingegneria informatica
automatica e gestionale Antonio Ruberti

Sapienza
Università di Roma