

## Application Programming Interface

- Standardizza l'interazione con il sistema operativo (SO)
- Consente ai processi applicativi di utilizzare i protocolli di rete, definendo:
  - le funzioni consentite
  - gli argomenti per ciascuna funzione
- **Socket:** Internet API
  - Definito inizialmente per sistemi Unix BSD per utilizzare i protocolli TCP/IP
    - Utilizza un'estensione del **paradigma di I/O** di questi sistemi
  - Divenuto uno standard di riferimento per tutta la programmazione su reti, disponibile su vari sistemi operativi (ad es. WinSock)
  - Due tipi di servizio di trasporto
    - Datagram non affidabile (UDP)
    - Flusso di byte orientato alla connessione ed affidabile (TCP)

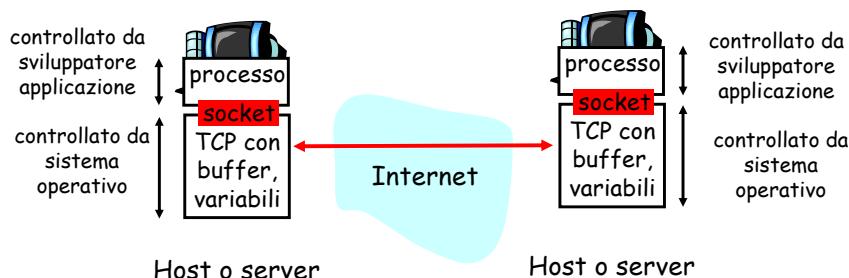
From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

1

## Programmazione di applicazioni di rete con socket - parte 1

### Socket

- E' un'interfaccia (o "porta") tra il processo applicativo e il protocollo di trasporto end-to-end (UCP o TCP)
- E' un'interfaccia **locale** all'host, controllata dal sistema operativo, creata/posseduta dall'applicazione tramite la quale il processo applicativo può inviare/ricevere messaggi a/di un altro processo applicativo (remoto o locale)



From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

### Socket (2)

- I socket sono delle API che consentono ai programmatore di gestire le comunicazioni tra processi
  - E' un meccanismo di **InterProcess Communication** (IPC)
- A differenza degli altri meccanismi di IPC (pipe, code di messaggi, FIFO e memoria condivisa), i socket consentono la comunicazione tra processi che possono risiedere su macchine **diverse** e sono collegati tramite una rete
  - Costituiscono lo strumento di base per realizzare servizi di rete
- **Astrazione dei processi sottostanti**
  - In pratica, i socket consentono ad un programmatore di effettuare trasmissioni TCP e UDP senza curarsi dei dettagli "di più basso livello" che sono uguali per ogni comunicazione (*three-way handshaking, finestre, ...*)

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

2

3

## Socket (3)

- **Socket**
  - astrazione del SO
  - creato dinamicamente dal SO su richiesta
  - persiste soltanto durante l'esecuzione dell'applicazione
  - il suo ciclo di vita è simile a quello di un file (apertura, collegamento ad un endpoint, lettura/scrittura, chiusura)
- **Descrittore del socket**
  - un intero
  - uno per ogni socket attivo
  - significativo soltanto per l'applicazione che possiede il socket
- **Endpoint del socket**
  - ogni associazione socket è una quintupla di valori  
**{protocollo, indirizzo locale, porta locale, indirizzo remoto, porta remota}**
  - l'associazione deve essere completamente specificata affinché la comunicazione abbia luogo

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

4

## Parametri della funzione socket()

domain	<b>Dominio del socket</b> (famiglia di protocolli); alcuni valori: <ul style="list-style-type: none"><li>- valore usuale: AF_INET (IPv4), ossia InterNET Protocol Family (comunicazione tra processi in Internet);</li><li>- AF_INET6 per IPv6</li><li>- AF_UNIX per comunicazioni tra processi sullo stesso host Unix</li></ul>
type	<b>Tipo di comunicazione</b> <ul style="list-style-type: none"><li>- SOCK_STREAM: orientato alla connessione (flusso continuo)</li><li>- SOCK_DGRAM: senza connessione (pacchetti)</li><li>- SOCK_RAW: per applicazioni dirette su IP (riservato all'uso di sistema)</li></ul>
protocol	<b>Protocollo specifico:</b> solitamente, si pone a 0 per selezionare il protocollo di default indotto dalla coppia domain e type (tranne che per SOCK_RAW): <ul style="list-style-type: none"><li>- AF_INET + SOCK_STREAM determinano una trasmissione TCP (protocol = IPPROTO_TCP)</li><li>- AF_INET + SOCK_DGRAM determinano una trasmissione UDP (protocol = IPPROTO_UDP)</li></ul>

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

## Dichiarazione al sistema operativo

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

- E' la prima funzione eseguita sia dal client sia dal server
- Definisce un socket: crea le strutture e riserva le risorse necessarie per la gestione di connessioni
- Restituisce un intero che va interpretato come un descrittore di file: un identificatore dell'entità appena creata
- Il client utilizzerà il socket direttamente, specificando il descrittore in tutte le funzioni che chiamerà
- Il server utilizzerà il socket indirettamente, come se fosse un modello o un prototipo per creare altri socket che saranno quelli effettivamente usati
- Se la creazione del socket fallisce, viene restituito il valore -1

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

5

## Gestione degli errori

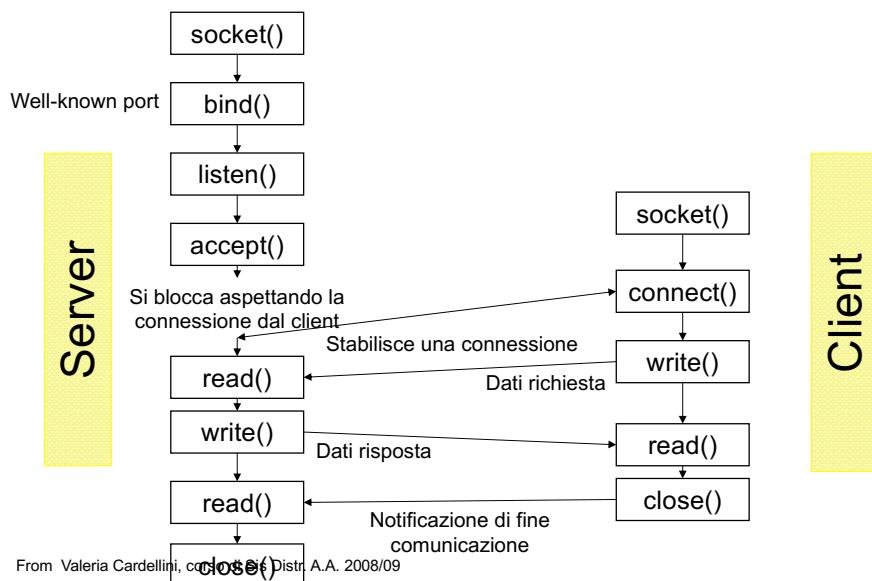
- In caso di successo le funzioni dell'API socket
  - restituiscono un valore positivo
- In caso di fallimento le funzioni dell'API socket
  - restituiscono un valore negativo (-1)
  - assegnano alla variabile globale **errno** un valore positivo
    - Ogni valore identifica un tipo di errore ed il significato dei nomi simbolici che identificano i vari errori è specificato in sys/errno.h
    - Le funzioni strerror() e perror() permettono di riportare in opportuni messaggi la condizione di errore verificatasi
- Il valore contenuto in errno si riferisce all'ultima chiamata di sistema effettuata
- Dopo aver invocato una funzione dell'API si deve
  - verificare se il codice di ritorno è negativo (errore)
  - in caso di errore, leggere immediatamente il valore di errno

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

7

6

## Comunicazione orientata alla connessione



8

## Per inizializzare indirizzo locale e processo locale (server)

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

- Serve a far sapere al SO a quale processo vanno inviati i dati ricevuti dalla rete
- sockfd:** descrittore del socket  
**addr:** puntatore ad una struct contenente l'indirizzo locale  
**len:** dimensione *in byte* della struct contenente l'indirizzo locale

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

9

## Struttura degli indirizzi

- Struttura generica
 

```
struct sockaddr {
    sa_family_t sa_family;
    char sa_data[14];}
```
- Struttura degli indirizzi IPv4
 

```
struct sockaddr_in {
    sa_family_t syn_family; /* AF_INET */
    in_port_t syn_port; /* numero di porta (2 byte) */
    struct in_addr sin_addr; /* struttura indirizzo IP (4 byte) */
    char sin_zero[8]; /* non usato */}
```

Equivale a `sa_data[14]` in `struct sockaddr`
- Struttura dati `in_addr`

```
struct in_addr {
    in_addr_t s_addr;}
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

10

## Inizializzare indirizzo IP e numero di porta

- L'indirizzo IP ed il numero di porta in `sockaddr_in` devono essere nel **network byte order**
  - Prima il byte più significativo, ovvero ordinamento **big endian**
- Funzioni di conversione da rappresentazione testuale/binaria dell'indirizzo/numero di porta a valore binario da inserire nella struttura `sockaddr_in`
- Per inizializzare i campi di `sockaddr_in`

```
unsigned short htons(int hostshort); /* numero di porta */
unsigned long htonl(int hostlong); /* indirizzo IP */
```

  - `htons:` host-to-network byte order short (16 bit)
  - `htonl:` host-to-network byte order long (32 bit)
- Per il riordinamento (da rete a macchina locale):
  - `ntohs(), ntohs()`

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

11

## Inizializzare indirizzo IP e numero di porta (2)

- Per convertire gli indirizzi IP dalla notazione puntata (*dotted decimal*) in formato ASCII al network byte order in formato binario
  - Es: `inet_aton("160.80.85.38", &(sad.sin_addr));`
- Per la conversione inversa
  - `char *inet_ntoa(struct in_addr addrptr);`
- Esistono anche le funzioni `inet_pton()` e `inet_ntop()`
  - `int inet_pton(int af, const char *src, void *addr_ptr);`
  - `char *inet_ntop(int af, const void *addr_ptr, char *dest, size_t len);`
  - A differenza di `inet_aton()` e `inet_ntoa()`, possono convertire anche gli indirizzi IPv6

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

12

## Indirizzo IP in bind()

- Si può assegnare un indirizzo IP specifico
  - L'indirizzo deve appartenere ad un'interfaccia di rete della macchina
- Per indicare un indirizzo IP generico (0.0.0.0)
  - Si usa la costante INADDR\_ANY
- Per indicare l'interfaccia di loopback (127.0.0.1)
  - Si usa la costante INADDR\_LOOPBACK

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

13

## Header file

- `sys/socket.h` definisce i simboli che iniziano con `PF_` e `AF_` ed i prototipi delle funzioni (ad es., `bind()`)
- `netinet/in.h` definisce i tipi di dato per rappresentare gli indirizzi Internet (ad es., la definizione di struct `sockaddr_in`)
- `arpa/inet.h` definisce i prototipi delle funzioni per manipolare gli indirizzi IP (ad es., `inet_aton()`)
- In più, `sys/types.h`, `netdb.h` e `unistd.h` (per `close()`)

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

14

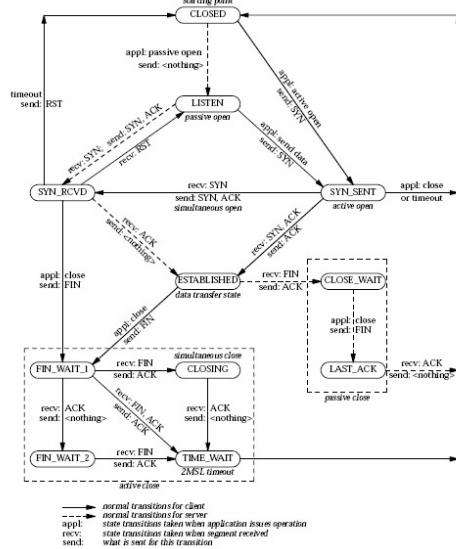
## Esempio funzione bind()

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
...
struct sockaddr_in sad;
int sd;
...
if ((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("errore in socket");
    exit(-1);
}
memset((void *)&sad, 0, sizeof(sad));
sad.sin_family = AF_INET;
sad.sin_port = htons(1234);
sad.sin_addr.s_addr = htonl(INADDR_ANY); /* il server accetta
richieste su ogni interfaccia di rete */
if (bind(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
    ...
}
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

15

## Diagramma di stato di TCP

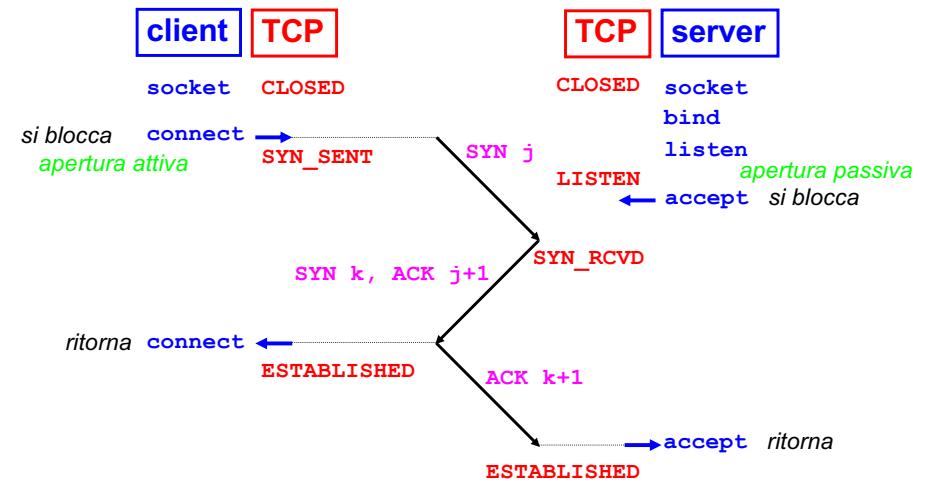


From Valeria Cardellini, corso di Sis Distr. A.A

TCP state transition diagram.

16

## Three way handshake del TCP



From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

17

## Funzione connect() (client)

```
int connect(int sockfd, const struct sockaddr *servaddr,
           socklen_t addrlen);
```

- Permette al client TCP di aprire la connessione con un server TCP (invio segmento SYN)
  - Bloccante: termina solo quando la connessione è stata creata
  - Restituisce 0 in caso di successo, -1 in caso di errore
- Parametri della funzione
  - sockfd**: descrittore del socket
  - servaddr**: indirizzo (indirizzo IP + numero di porta) del server cui ci si vuole connettere
  - addrlen**: dimensione in byte della struttura con l'indirizzo remoto del server
- In caso di errore, la variabile **errno** può assumere i valori:
  - ETIMEDOUT: scaduto il timeout del SYN
  - ECONNREFUSED: nessuno in ascolto (RST in risposta a SYN)
  - ENETUNREACH: errore di indirizzamento (messaggio ICMP di destinazione non raggiungibile)

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

18

## Gestione di errori transitori in connect()

### Algoritmo di backoff esponenziale

- Se connect() fallisce, il processo attende per un breve periodo e poi tenta di nuovo la connessione, incrementando progressivamente il ritardo, fino ad un massimo di circa 2 minuti

```
#include <sys/socket.h>
```

```
# define MAXSLEEP 128
```

```
int connect_retry(int sockfd, const struct sockaddr *addr, socklen_t alen)
{
    int nsec;

    for (nsec=1; nsec <= MAXSLEEP; nsec <<=1) {
        if (connect(sockfd, addr, alen) == 0)
            return(0); /*connessione accettata */
        /* Ritardo prima di un nuovo tentativo di connessione */
        if (nsec <= MAXSLEEP/2)
            sleep(nsec);
    }
    return(-1);
}
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

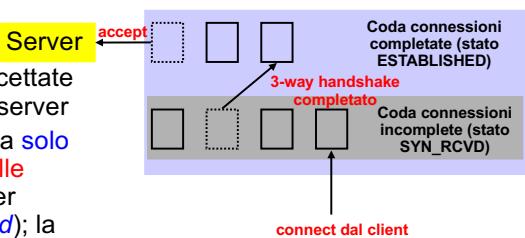
19

## Funzione listen() (server)

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

- Mette il socket in ascolto di eventuali connessioni
  - dallo stato CLOSED allo stato LISTEN
- Specifica quante connessioni possono essere accettate dal server e messe in attesa di essere servite nella coda di richieste di connessione (**backlog**)
  - Le connessioni possono essere accettate o rifiutate dal SO senza interrogare il server
- Storicamente, nel backlog c'erano sia le richieste di connessione in corso di accettazione, sia quelle accettate ma non ancora passate al server
- In Linux, il backlog identifica **solo** la lunghezza della **coda delle connessioni complete** (per prevenire l'attacco *syn flood*); la **lunghezza massima è pari a SOMAXCONN**

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09



20

## Funzione accept() (server)

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

- Permette ad un server di prendere dal backlog la prima connessione completata sul socket specificato
  - Se il backlog è vuoto, il server rimane bloccato sulla chiamata finché non viene accettata una connessione
- Restituisce
  - 1 in caso di errore
  - Un nuovo descrittore di socket creato e assegnato automaticamente dal SO e l'indirizzo del client connesso
- Parametri della funzione
  - sockfd**: descrittore del socket originale
  - addr**: viene riempito con l'indirizzo del client (indirizzo IP + porta)
  - addrlen**: viene riempito con la dimensione dell'indirizzo del client; prima della chiamata inizializzato con la lunghezza della struttura il cui indirizzo è passato in **addr**

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

21

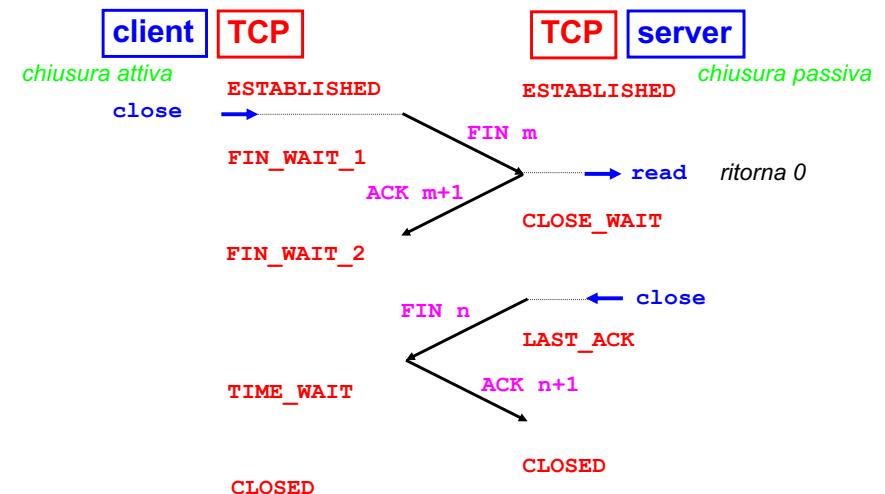
## Socket d'ascolto e socket connesso

- Il server utilizza due socket diversi per ogni connessione con un client
- Socket d'ascolto** (listening socket): creato da `socket()`
  - Utilizzato per tutta la vita del processo server
  - In genere usato solo per accettare richieste di connessione
  - Resta per tutto il tempo nello stato LISTEN
- Socket connesso** (connected socket): creato da `accept()`
  - Usato solo per la connessione con un dato client
  - Usato per lo scambio di dati con il client
  - Si trova automaticamente nello stato ESTABLISHED
- I due socket identificano due connessioni distinte

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

22

## Four way handshake del TCP



From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

23

## La chiusura di una connessione TCP

```
int close(int sockfd);
```

- Il processo che invoca per primo close() avvia la **chiusura attiva** della connessione
- Il descrittore del socket è marcato come chiuso
  - Il processo non può più utilizzare il descrittore ma la connessione non viene chiusa subito, perché il TCP continua ad utilizzare il socket trasmettendo i dati che sono eventualmente nel buffer
- Restituisce 0 in caso di successo, -1 in caso di errore
- All'altro capo della connessione, dopo che è stato ricevuto ogni dato eventualmente rimasto in coda, la ricezione del FIN viene segnalata al processo come un EOF in lettura
  - Rilevato l'EOF, il secondo processo invoca close() sul proprio socket, causando l'emissione del secondo FIN

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

24

## Per leggere/scrivere dati

- Per leggere e scrivere su un socket si usano le funzioni read() e write()

```
int read(int sockfd, void *buf, size_t count);
int write(int sockfd, const void *buf, size_t count);
```
- Attenzione:** non si può assumere che write() o read() terminino avendo sempre letto o scritto il numero di caratteri richiesto
  - read() o write() restituiscono il numero di caratteri **effettivamente** letti o scritti, oppure -1 in caso di errore
- Come determinare la condizione EOF?
  - read() restituisce 0 (per indicare che la parte remota dell'associazione è stata chiusa)
- Cosa succede se si prova a scrivere su un socket la cui parte remota è stata chiusa?
  - Il SO invia il segnale SIGPIPE e write() restituisce -1, impostando errore per SIGPIPE

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

26

## La chiusura di una connessione TCP (2)

- Per chiudere una connessione TCP, si può usare anche la funzione shutdown()

```
int shutdown(int sockfd, int how);
```

- shutdown() permette la chiusura asimmetrica
- Il parametro how può essere uguale a:
  - SHUT\_RD** (0): ulteriori receive sul socket sono disabilitate (viene chiuso il lato in lettura del socket)
  - SHUT\_WR** (1): ulteriori send sul socket sono disabilitate (viene chiuso il lato in scrittura del socket)
  - SHUT\_RDWR** (2): ulteriori send e receive sulla socket sono disabilitate
    - Non è uguale a close(): la sequenza di chiusura del TCP viene effettuata immediatamente, indipendentemente dalla presenza di altri riferimenti al socket (ad es. descrittore ereditato dai processi figli oppure duplicato tramite dup())
    - Con close(), la sequenza di chiusura del TCP viene innescata solo quando il numero di riferimenti del socket si annulla

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

25

## Per leggere/scrivere dati (2)

- In alternativa a write() e read(), si possono usare

```
int send (int sockfd, const void* buf, size_t len, int flags);
int recv(int sockfd, void *buf, size_t len, int flags);
```
- Se flags=0 allora send() e recv() equivalgono a write() e read()

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

27

## Funzione readn

```
#include <errno.h>
#include <unistd.h>
int readn(int fd, void *buf, size_t n)
{
    size_t nleft;
    ssize_t nread;
    char *ptr;

    ptr = buf;
    nleft = n;
    while (nleft > 0) {
        if ((nread = read(fd, ptr, nleft)) < 0) {
            if (errno == EINTR) /* funzione interrotta da un segnale prima di aver
                                potuto leggere qualsiasi dato.*/
                nread = 0;
            else
                return(-1); /*errore */
        }
        nleft -= nread;
        ptr += nread;
    }
}
```

Legge  $n$  byte da un socket,  
eseguendo un ciclo di lettura fino a  
leggere  $n$  byte o incontrare EOF o  
riscontrare un errore

Restituisce 0 in caso di successo,  
-1 in caso di errore, il numero di  
byte non letti in caso di EOF

## Funzione readn (2)

```
else if (nread == 0) break; /* EOF: si interrompe il ciclo */
nleft -= nread;
ptr += nread;
} /* end while */
return(nleft); /* return >= 0 */
}
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

28

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

29

## Funzione writen

```
#include <errno.h>
#include <unistd.h>

ssize_t writen(int fd, const void *buf, size_t n)
{
    size_t nleft;
    ssize_t nwritten;
    const char *ptr;

    ptr = buf;
    nleft = n;
    while (nleft > 0) {
        if ((nwritten = write(fd, ptr, nleft)) <= 0) {
            if ((nwritten < 0) && (errno == EINTR)) nwritten = 0;
            else return(-1); /* errore */
        }
        nleft -= nwritten;
        ptr += nwritten;
    } /* end while */
    return(nleft);
}
```

Scrive  $n$  byte in un socket,  
eseguendo un ciclo di scritture fino  
a scrivere  $n$  byte o riscontrare un  
errore

Restituisce 0 in caso di successo,  
-1 in caso di errore

## Progettazione di un client TCP

Passi base per progettare un client TCP

1. Creazione di un endpoint
  - Richiesta al sistema operativo
2. Creazione della connessione
  - Implementazione del 3-way handshake per l'apertura della connessione TCP
3. Lettura e scrittura sulla connessione
  - Analogico a operazione su file in Unix
4. Chiusura della connessione
  - Implementazione del 4-way handshake per la chiusura della connessione TCP

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

30

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

31

## Progettazione di un server TCP

Passi base per progettare un server TCP

1. Creazione di un endpoint
  - Richiesta al sistema operativo
2. Collegamento dell'endpoint ad una porta
3. Ascolto sulla porta
  - Processo sospeso in attesa
4. Accettazione della richiesta di un client
5. Letture e scritture sulla connessione
6. Chiusura della connessione

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

32

## Client TCP daytime

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

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

#define SERV_PORT      5193
#define MAXLINE        1024

int main(int argc, char *argv[ ])
{
    int      sockfd, n;
    char    recvline[MAXLINE + 1];
    struct  sockaddr_in servaddr;
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

34

## Esempio: daytime TCP

- Il client interroga il server per conoscere la data e l'ora
- Il server ottiene l'informazione dal SO e la invia al client
- Il client stampa l'informazione su stdout
- Assunzioni
  - Il server invia la risposta in un'unica stringa alfanumerica
  - Il client legge la stringa, la visualizza a schermo e termina
  - Client e server utilizzano una connessione TCP
- Per eseguire il client  
daytime\_clientTCP <indirizzo IP server>
- Per eseguire il server  
daytime\_serverTCP

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

33

## Client TCP daytime (2)

```
if (argc != 2) { /* controlla numero degli argomenti */
    fprintf(stderr, "utilizzo: daytime_clientTCP <indirizzo IP server>\n");
    exit(-1);
}

if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) { /* crea il socket */
    perror("errore in socket");
    exit(-1);
}

memset((void *)&servaddr, 0, sizeof(servaddr));           /* azzerza servaddr */
servaddr.sin_family = AF_INET;   /* assegna il tipo di indirizzo */
servaddr.sin_port = htons(SERV_PORT); /* assegna la porta del server */
/* assegna l'indirizzo del server prendendolo dalla riga di comando; l'indirizzo è
una stringa da convertire in intero secondo network byte order. */
if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0) {
    /* inet_pton (p=presentation) anche per indirizzi IPv6 */
    perror("errore in inet_pton");
    exit(-1);
}
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

35

## Client TCP daytime (3)

```
/* stabilisce la connessione con il server */
if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0) {
    perror("errore in connect");
    exit(-1);
}
/* legge dal socket fino a quando non trova EOF */
while ((n = read(sockfd, recvline, MAXLINE)) > 0) {
    recvline[n] = 0; /* aggiunge il carattere di terminazione */
    if (fputs(recvline, stdout) == EOF) { /* stampa recvline sullo stdout */
        perror("errore in fputs");
        exit(-1);
    }
}
if (n < 0) {
    perror("errore in read");
    exit(-1);
}
exit(0);
}
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

36

## Server TCP daytime

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define SERV_PORT      5193
#define BACKLOG         10
#define MAXLINE        1024

int main(int argc, char *argv[ ])
{
    int      listenfd, connfd;
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

37

## Server TCP daytime (2)

```
struct sockaddr_in      servaddr;
char                  buff[MAXLINE];
time_t                ticks;

if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) { /* crea il socket */
    perror("errore in socket");
    exit(-1);
}

memset((void *)&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY); /* il server accetta
connessioni su una qualunque delle sue interfacce di rete */
servaddr.sin_port = htons(SERV_PORT); /* numero di porta del server */

/* assegna l'indirizzo al socket */
if (bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0) {
    perror("errore in bind");
    exit(-1);
}
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

38

## Server TCP daytime (3)

```
if (listen(listenfd, BACKLOG) < 0 ) {
    perror("errore in listen");
    exit(-1);
}

while (1) {
    if ( (connfd = accept(listenfd, (struct sockaddr *) NULL, NULL)) < 0) {
        perror("errore in accept");
        exit(-1);
    }

    /* accetta una connessione con un client */
    ticks = time(NULL); /* legge l'orario usando la chiamata di sistema time */
    /* scrive in buff l'orario nel formato ottenuto da ctime
       snprintf impedisce l'overflow del buffer */
    snprintf(buff, sizeof(buff), "%-.24s\r\n", ctime(&ticks));
    /* ctime trasforma la data e l'ora da binario in ASCII
       \r\n per carriage return e line feed */
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

39

## Server TCP daytime (4)

```
/* scrive sul socket di connessione il contenuto di buff */
if (write(connsd, buff, strlen(buff)) != strlen(buff)) {
    perror("errore in write");
    exit(-1);
}

if (close(connsd) == -1) { /* chiude la connessione */
    perror("errore in close");
    exit(-1);
}
}
exit(0);
}
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

40

## Esercizio

Sviluppare in C un'applicazione client/server che offre il **servizio di trasferimento di un file** da un client ad un server.

Il **client** richiede all'utente, in modo interattivo, il nome del file da trasferire e, se il file esiste, apre una connessione con il server. Quando la connessione viene accettata, il client invia al server il nome del file, seguito dal suo contenuto. Infine, il client attende l'esito dell'operazione dal server, stampa a schermo l'esito dell'operazione e quindi torna in attesa di ricevere dall'utente una nuova richiesta di trasferimento.

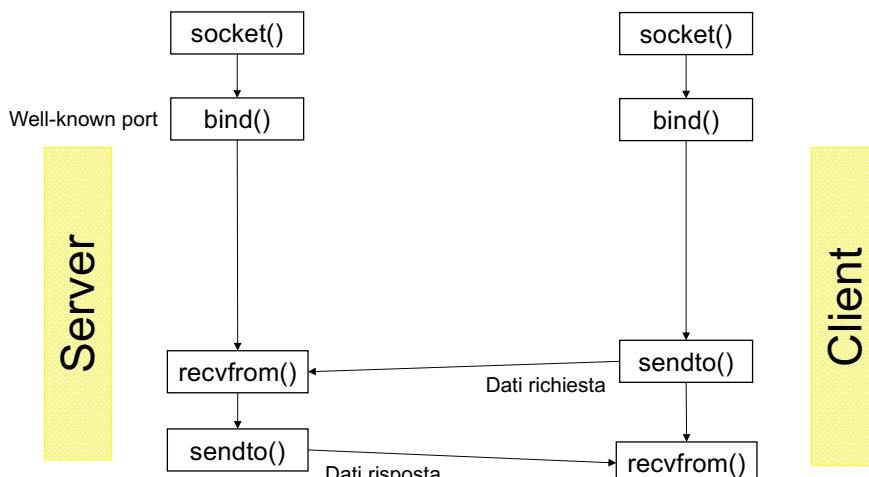
Il **server** riceve una richiesta di connessione e salva il file richiesto in una directory locale. Alla fine della ricezione del file, il server invia al client l'esito della operazione, che può essere di due tipi:

- update: il file esisteva già ed è stato sovrascritto
- new: è stato creato un nuovo file

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

41

## Comunicazione senza connessione



From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

42

## Comunicazione senza connessione (2)

- Per creare un socket
  - socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);
  - SOCK\_DGRAM è il tipo di protocollo per datagram
  - IPPROTO\_UDP è lo specifico protocollo richiesto (anche 0)
- Il server invoca bind() per associare al socket l'indirizzo locale e la porta locale
  - Il numero della porta è prestabilito per l'applicazione
- Il client invoca bind() per associare al socket l'indirizzo locale e la porta locale
  - Il numero della porta è in qualche modo arbitrario
  - Il numero di porta 0 lascia al SO la scelta della porta
- Il server ed il client hanno così impostato la semi-associazione locale del socket; l'impostazione delle altre due componenti dell'associazione può essere fatta per ogni pacchetto

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

43

## Funzione sendto()

```
int sendto(int sockfd, const void *buf, size_t len, int flags,  
          const struct sockaddr_in *to, socklen_t tolen);
```

- Restituisce il numero di byte inviati (oppure -1 in caso di errore)
- Trasmissione non affidabile: sendto() non restituisce errore se il pacchetto non raggiunge l'host remoto
  - soltanto errori locali: ad es., dimensione del pacchetto maggiore della dimensione massima del datagram usato (errore EMSGSIZE)
- Parametri della funzione
  - sockfd**: descrittore del socket a cui si fa riferimento
  - buf**: puntatore al buffer contenente il pacchetto da inviare
  - len**: dimensione (in byte) del pacchetto da inviare
  - flags**: intero usato come maschera binaria per impostare una serie di modalità di funzionamento della comunicazione; per ora 0
  - to**: l'indirizzo IP e la porta della macchina remota che riceverà il pacchetto inviato
  - tolen**: dimensione (in byte) della struttura sockaddr

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

44

## Funzione recvfrom()

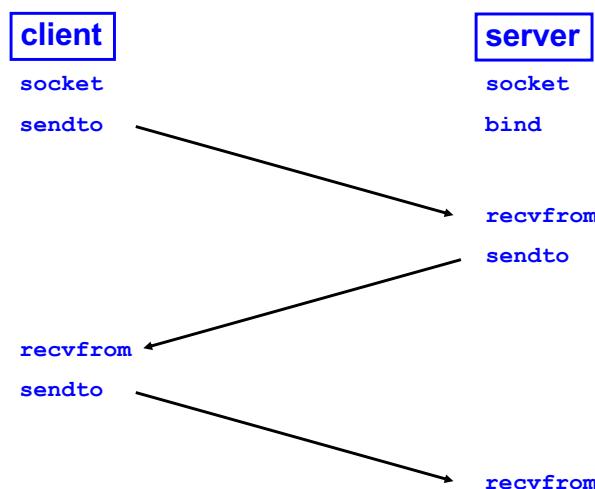
```
int recvfrom(int sockfd, const void *buf, size_t len, int flags,  
            struct sockaddr *from, socklen_t *fromlen);
```

- Restituisce il numero di byte ricevuti (-1 in caso di errore)
- Parametri della funzione
  - sockfd**: descrittore del socket a cui si fa riferimento
  - buf**: puntatore al buffer contenente il pacchetto da ricevere
  - len**: dimensione del buffer (numero massimo di byte da leggere)
    - Se il pacchetto ricevuto ha dimensione maggiore di len, si ottengono i primi len byte ed il resto del pacchetto è perso
  - from**: puntatore alla struttura contenente l'indirizzo IP e la porta della macchina remota che ha spedito il pacchetto; usato per ottenere l'indirizzo del mittente del pacchetto
  - fromlen**: puntatore alla dimensione (in byte) della struttura sockaddr
- Riceve i pacchetti da qualunque macchina remota: non è possibile rifiutare un pacchetto
  - Per verificare l'indirizzo del client, si usa il flag MSG\_PEEK che non toglie il pacchetto dalla coda del SO

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

45

## Scambio di pacchetti in comunicazione UDP



From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

46

## Uso di connect() con i socket UDP

- connect() può essere usata anche in applicazioni client di tipo senza connessione per impostare una volta per tutte le due componenti remote dell'associazione del client e gestire la presenza di errori asincroni
  - ad es. indirizzo inesistente o su cui non è in ascolto un server
- Usando connect() il client può lasciare nulli gli ultimi campi in sendto() oppure usare la write() o la send()

```
int sendto(sockfd, buf, len, flags, NULL, 0);  
int write(sockfd, buf, len);
```
- connect() impedisce la ricezione sul socket di pacchetti non provenienti dalla macchina remota registrata

```
int recvfrom(sockfd, buf, len, flags, NULL, NULL);  
int read(sockfd, buf, len);
```
- Non ha senso usare connect() sul server!

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

47

## Esempio: daytime UDP

- Il client interroga il server per ottenere la data e l'ora
  - Il server ottiene l'informazione dal SO e la invia al client
  - Il client stampa l'informazione su stdout
- 
- Assunzioni
    - Il server invia la risposta in un'unica stringa alfanumerica
    - Il client legge la stringa, la visualizza sullo schermo e termina
    - Client e server utilizzano una comunicazione UDP
- 
- Per eseguire il client  
daytime\_clientUDP <indirizzo IP server>
  - Per eseguire il server  
daytime\_serverUDP

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

48

## Client UDP daytime

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>

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

#define SERV_PORT      5193
#define MAXLINE        1024

int main(int argc, char *argv[ ])
{
    int      sockfd, n;
    char    recvline[MAXLINE + 1];
    struct  sockaddr_in servaddr;
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

49

## Client UDP daytime (2)

```
if (argc != 2) { /* controlla numero degli argomenti */
    fprintf(stderr, "utilizzo: daytime_clientUDP <indirizzo IP server>\n");
    exit(1);
}

if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) { /* crea il socket */
    perror("errore in socket");
    exit(-1);
}

memset((void *)&servaddr, 0, sizeof(servaddr));           /* azzerza servaddr */
servaddr.sin_family = AF_INET;   /* assegna il tipo di indirizzo */
servaddr.sin_port = htons(SERV_PORT); /* assegna la porta del server */
/* assegna l'indirizzo del server prendendolo dalla riga di comando. L'indirizzo è
una stringa da convertire in intero secondo network byte order. */
if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0) {
    /* inet_pton (p=presentation) vale anche per indirizzi IPv6 */
    perror("errore in inet_pton");
    exit(-1);}
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

50

## Client UDP daytime (3)

```
/* Invia al server il pacchetto di richiesta*/
if (sendto(sockfd, NULL, 0, 0, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0) {
    perror("errore in sendto");
    exit(-1);
}
/* legge dal socket il pacchetto di risposta */
n = recvfrom(sockfd, recvline, MAXLINE, 0 , NULL, NULL);
if (n < 0) {
    perror("errore in recvfrom");
    exit(-1);
}
if (n > 0) {
    recvline[n] = 0;    /* aggiunge il carattere di terminazione */
    if (fputs(recvline, stdout) == EOF) { /* stampa recvline sullo stdout */
        perror("errore in fputs");
        exit(-1);
    }
}
exit(0);
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

51

## Server UDP daytime

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define SERV_PORT      5193
#define MAXLINE        1024

int main(int argc, char *argv[ ])
{
    int      sockfd, len;
    struct  sockaddr_in     addr;
    char    buff[MAXLINE];
    time_t   ticks;
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

52

## Server UDP daytime (2)

```
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) { /* crea il socket */
    perror("errore in socket");
    exit(-1);
}

memset((void *)&addr, 0, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY); /* il server accetta pacchetti su
una qualunque delle sue interfacce di rete */
addr.sin_port = htons(SERV_PORT); /* numero di porta del server */

/* assegna l'indirizzo al socket */
if (bind(sockfd, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
    perror("errore in bind");
    exit(-1);
}
```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

53

## Server UDP daytime (3)

```
while (1) {
    len = sizeof(addr);
    if ( (recvfrom(sockfd, buff, MAXLINE, 0, (struct sockaddr *)&addr, &len)) < 0) {
        perror("errore in recvfrom");
        exit(-1);
    }

    ticks = time(NULL); /* legge l'orario usando la chiamata di sistema time */
    /* scrive in buff l'orario nel formato ottenuto da ctime
       snprintf impedisce l'overflow del buffer */
    snprintf(buff, sizeof(buff), "%-.24s\r\n", ctime(&ticks));
    /* ctime trasforma la data e l'ora da binario in ASCII
       \r\n per carriage return e line feed*/
    /* scrive sul socket il contenuto di buff */
    if (sendto(sockfd, buff, strlen(buff), 0, (struct sockaddr *)&addr, sizeof(addr)) < 0)
    {
        perror("errore in sendto");
        exit(-1);
    }
}
exit(0);

```

From Valeria Cardellini, corso di Sis Distr. A.A. 2008/09

54