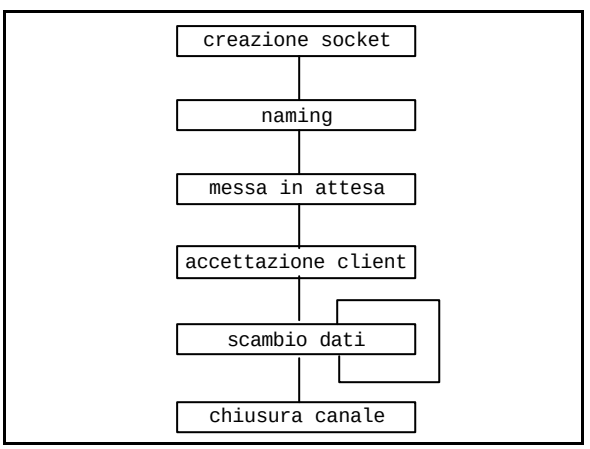
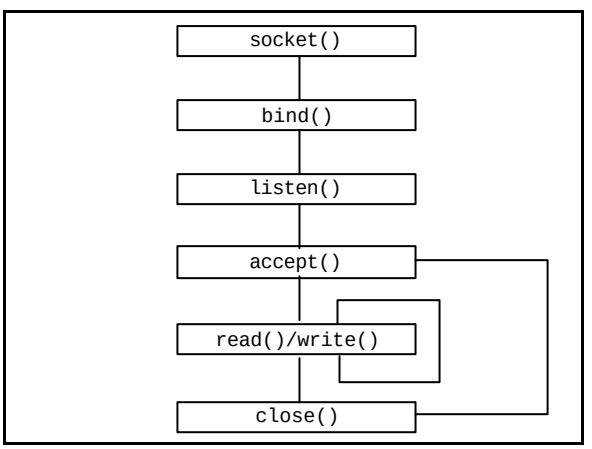
Implementazione di un server

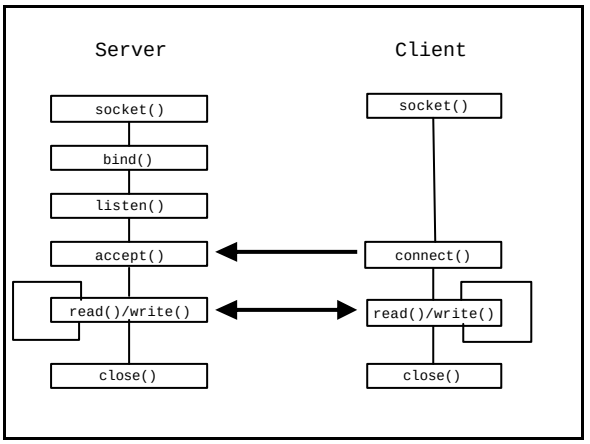
Dobbiamo scrivere un programma in C che offre un servizio

Server iterativo

- Offro servizio ad un client alla volta
- Il sistema operativo tiene gli altri client "in coda"
- È la variante più semplice







bind()

- Associa ad una socket l'indirizzo trasnport sul quale dovrà attendere i client
- Viene anche chiamato "naming" della socket
- Anche in questo caso dobbiamo "riempire" una struttura che contiene un indirizzo transport

bind()

struct sockaddr_in
struct sockaddr_un

sizeof(localaddr)

```
int error;
error = bind(socket, localaddr, addrlen);
```

socket()

Local address

- L'indirizzo IP e la porta prendono significati diversi rispetto al client
- L'indirizzo IP
  - È lindirizzo LOCALE tramite il quale sarà possibile accettare connessioni
    (ha senso solo nel caso ci siano più schede di rete)

```
local_addr.sin_addr.s_addr = INADDR_ANY;
```

- La porta
  - È la porta sulla quale il server sarà in attesa.
  - Questa è un informazione che interessa al client

Le porte disponibili

- Non tutte le porte sono uguali
  - Da 1 a 1024 sono riservate al sistema
    - Solo il superuser può fare una bind a quella porta
    - Sono quasi tutte standardizzate
  - Da 1025 a 65532 sono allocabili dagli utenti
    - Chiunque può fare bind
    - Se specifichiamo “porta 0” alla bind li sistema alloca il server sulla prima libera maggiore di 1024
    - Alcune sono standardizzate

```
int s;
struct sockaddr_in local_addr;

s = socket(AF_INET, SOCK_STREAM, 0);
if ( s < 0 ) {
       perror("socket() ");
       exit(1);
}

local_addr.sin_family = AF_INET;
local_addr.sin_port = htons(0);
local_addr.sin_addr.s_addr = INADDR_ANY;

If (bind(s, &local_addr, sizeof(local_addr)) == -1) {
       perror("bind() ");
       exit(1);
}
```

E adesso ?

- Ho lasciato scegliere al sistema la porta, e adesso ... come faccio a sapere qual’è ?

```
int error;
error = getsockbyname(socket, sockaddr, addrlen);
```

indirizzo da riempire

```
struct sockaddr_in query_addr;
int query_addrlen;

...

if (bind(s, &local_addr, sizeof(local_addr)) == -1) {
       perror("bind() ");
       exit(1);
}

query_addrlen = sizeof(query_addr);
if (getsockname(s, &query_addr, &query_addrlen) == -1){
       perror("getsockname() ");
       exit(2);
}

printf(“binded on port %d\n”,
                    ntohs(query_addr -> sin_port));
```

listen()

- Abilita la socket a ricevere connessioni dai client
- Stabilisce la lunghezza massima della coda d'attesa
- Quando un client si presenta e tutta la coda è già piena, allora a questo viene rifiutato il servizio
- Per motivi non molto chiari la lunghezza massima della coda è 5
- Dopo che una socket viene abilitata all'attesa, non potrà più essere usata per trasferire dati, ma solo per attendere richieste di connessione

listen()

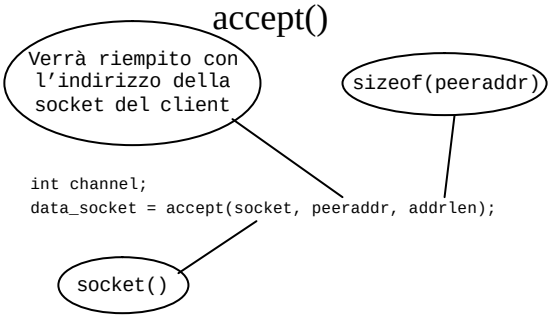
```
int error;
error = listen(socket, queuelen);
```

```
if (listen(s, 5) == -1) {
      perror("listen() ");
      exit(1);
}
```

accept()

- Serve ad "accettare" richieste di connessione da parte di un client
- Il processo rimane bloccato fino a che un client non esegue una `connect` all'indirizzo transport su cui siamo in attesa

accept()



```
struct sockaddr_in accept_addr;
int accept_addrlen;
char client_host[255];
int client_port;

...

accept_addrlen = sizeof(struct sockaddr_in);

data_sock = accept(s, &accept_addr, &accept_addrlen);
if (data_sock < 0) {
       perror("accept() ");
       exit(1);
}

strcpy(client_host, inet_ntoa(accept_addr.sin_addr));
client_port = ntohs(accept_addr.sin_port);

printf(“client on host %s, port %d\n",
                    caller_host, client_port);
```