Laboratorio di sistemi operativi
A.A. 2010/2011
Gruppo 2
Gennaro Oliva
20
Socket

Socket

- I socket sono un tipo di IPC che può essere utilizzato per la comunicazione tra due processi in esecuzione sullo stesso calcolatore oppure su calcolatori diversi interconnessi mediante una rete telematica
- Furono introdotti nella versione 4.2 di BSD (1983)
- Uno degli obiettivi della progettazione dei socket è stato quello di consentire l'utilizzo della stessa interfaccia per consentire entrambi i tipi di comunicazione e di l'uso di diversi protocolli di comunicazione tra cui il TCP/IP
- Con l'espressione TCP/IP si identifica una suite di protocolli chiamata “Internet Protocol Suite” utilizzata per le comunicazione sulla rete Internet

Applicazioni di rete

- I socket vengono utilizzati nell'implementazione di applicazioni di rete
- Applicazioni di rete: programmi in esecuzione su macchine differenti che operano in modo indipendente e che si scambiano informazioni utilizzando un sottosistema di comunicazione
- Principali modelli di programmazione per applicazioni di rete
  - client–server
  - peer–to-peer
  - three-tier

Modello client-server

- Nel modello client-server si distinguono due categorie di programmi
  - quelli che forniscono un servizio chiamati server
  - quelli che utilizzano il servizio chiamati client
- I servizi fondamentali di internet utilizzano questo modello: http, ftp, telnet, ssh, etc.
- Un server può (di norma deve) essere in grado di "servire" più client "contemporaneamente"
- Distinguiamo due classi di server:
  - concorrenti (servono più richieste in contemporanea)
  - iterativi (servono una richiesta per volta)

Identificazione del server

- Un client identifica un server utilizzando due livelli di indirizzamento:
- Il primo livello individua il calcolatore su cui è in esecuzione il processo server
- In una rete, ogni calcolatore è detto host
- Ad ogni host di una rete IP viene associato un indirizzo IP che lo identifica univocamente
- Gli indirizzi IPv4 sono costituiti da 32 bit (4 byte) che vengono solitamente descritti mediante la notazione dotted ovvero con 4 numeri decimali rappresentati su 1 byte (quindi ogni numero varia tra 0 e 255) separati da un punto

Indirizzi IPv4

- Un esempio di indirizzo IPv4 è il seguente: 192.132.34.22
- Questo tipo di indirizzamento limita il numero di indirizzi possibili a 2^{32} (4,294,967,296)
- La rete Internet esclude 18.000.000 indirizzi utilizzati per le reti private
- Per ovviare al problema della mancanza di indirizzi IP dovuta alla costante crescita di Internet è stato introdotto l'IPv6 in cui gli host vengono identificati con indirizzi a 6 byte

Porte

- Il secondo livello di indirizzamento utilizzato da un client per identificare un server, consente di specificare uno specifico processo server in esecuzione sull'host
- In un ambiente multitasking più processi in esecuzione su uno stesso host devono poter comunicare mediante lo stesso sottosistema di rete ed è necessaro distinguere le diverse comunicazioni
- A tal fine ad ogni applicazione server viene associato un numero di porta ovvero un intero a 16 bit

Porte

- I numeri di porta vanno da 0 a 65535, alcuni sotto intervalli di questo range identificano delle porte particolari:
- 0-1023 porte ben note utilizzabili solo da processi root
- 1024-49151 porte registrate utilizzate per servizi registrati presso un autorità di controllo chiamata IANA (Internet Assigned Numbers Authority)
- 49152-65535 porte private o dinamiche utilizzate solitamente dai client come porte effimere

Esempi di porte riservate

- Esempi di porte riservate
- 21 ftp (trasferimento file)
- 22 ssh (login remoto sicuro)
- 25 smtp (invio email)
- 80 http (web)
- 143 imap (lettura email)
- Lista ufficiale su: http://www.iana.org/
- La corrispondenza tra nomi simbolici e numeri si trova nel file /etc/services

endpoint e socket

- Il client quindi identifica il processo server con cui dialogare mediante il suo indirizzo ip e la porta effettua e indirizza tutte le richieste di servizio utilizzando queste informazioni
- La stessa cosa fa il server che comunica con il client utilizzando il suo indirizzo IP e la sua porta
- La coppia (indirizzo ip, porta) viene detta endpoint
- Ogni connessione avviene tra due endpoint

Il server

- Un server attendere le connessioni dei client attraverso la porta che lo identifica
- Un server effettua tipicamente la seguente sequenza di operazioni

1) Crea il socket usando socket
2) Gli associa un endpoint usando bind
3) Si pone in attesa di nuove connessioni usando listen
4) Stabilisce una nuova connessione usando accept
5) Interagisce con il client usando read e write
6) Chiude il socket usando close

socket

- Un socket viene generato utilizzando:

  int socket(int fam, int tipo, int proto);

- fam determina la natura della comunicazione individuata da una serie di costanti che iniziano con AF_ (address family) elencate nella pagina di manuale di socket
- Alcune costanti per il parametro fam:

Famiglia	Scopo	man
AF_UNIX AF_LOCAL	Comunicazioni locali	unix(7)
AF_INET	IPv4	ip(7)
AF_INET6	IPv6	ipv6(7)
AF_PACKET	Raw	packet(7)
…	…	…

Tipo di socket

- Il parametro tipo caratterizza ulteriormente il socket

  int socket(int fam, int tipo, int proto);
- SOCK_STREAM canale bidirezionale, sequenziale, affidabile che opera su connessione su cui i dati vengono ricevuti e trasmessi come un flusso continuo (TCP in AF_INET, o socket UNIX in AF_LOCAL)
- SOCK_DGRAM usato per trasmettere pacchetti di dati di lunghezza massima prefissata (datagram), indirizzati singolarmente senza connessione in maniera non affidabile (UDP in AF_INET)
- SOCK_RAW canale di basso livello per accedere ai protocolli di rete e alle varie interfacce. Solitamente non utilizzato dalle applicazioni
- ...

Tipo di socket

- Il parametro proto è solitamente pari a `0`, valore che seleziona il protocollo di default per una coppia data famiglia,tipo
- int socket(int fam, int tipo, int proto);
- Se una coppia famiglia, tipo supporta diversi protocolli è possibile selezionarne uno specifico utilizzando diversi valori per proto

socket file descriptor

- La system call socket, come la open restituisce un file descriptor (o socket descriptor) da utilizzare per le operazioni di I/O, read, write, close, dup, dup2 ma non lseek

bind

- L'associazione di un endpoint ad un socket viene effettuata utilizzando:

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

- La bind associa l'endpoint epoint al socket fd
- Il tipo effettivo del secondo argomento dipende dalla famiglia del socket
  - per socket locali è una struct di tipo sockaddr_un che memorizza un pathname
  - per i socket TCP/UDP è una struct di tipo sockaddr_in che memorizza indirizzo IP e numero di porta del servizio
- Il terzo argomento è pari alla dimensione in byte del secondo argomento e viene solitamente calcolata utilizzando la funzione sizeof()
- bind restituisce `0` se l'operazione è avvenuta con successo oppure -1 in caso di errore ovvero se la porta è già in uso

Codifica dati

- Come viene memorizzato un dato superiore al byte?
- Si consideri il caso di un intero costituito da 4 byte prendendo ad esempio il numero 32769, cioe' $2^{15} + 1$
- Utilizzando la codifica "Big Endian" che memorizza prima il byte più significativo:

  00000000 00000000 10000000 00000001
- Mentre con la codifica "Little Endian" che memorizza prima il byte meno significativo:

  00000001 10000000 00000000 00000000
- Inviando un intero da una architettura big-endian ad una architettura little-endian (o viceversa) senza convertirlo causerebbe in un errore di interpretazione del dato

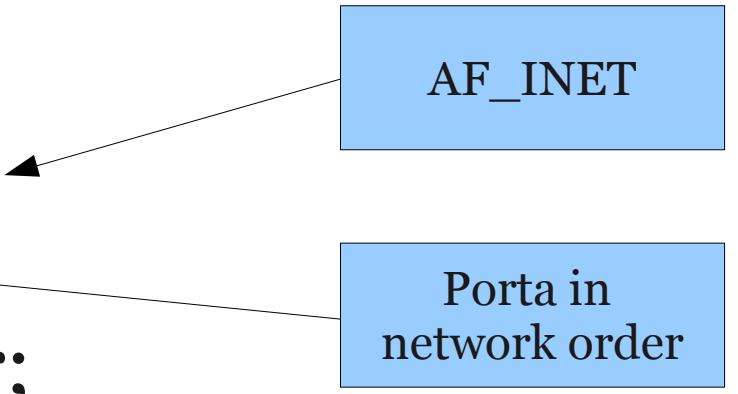
Funzioni di conversione

- La suite di protocolli internet utilizza big endian pertanto sono necessarie funzioni di conversione dalla rappresentazione host a quella network e viceversa da utilizzare prima di inviare un qualsiasi intero su rete e dopo aver ricevuto un intero dalla rete
- #include <netinet/in.h>
  uint32_t htonl(uint32_t x)
  uint16_t htons(uint16_t x)
  uint32_t ntohl(uint32_t x)
  uint16_t ntohs(uint16_t x)

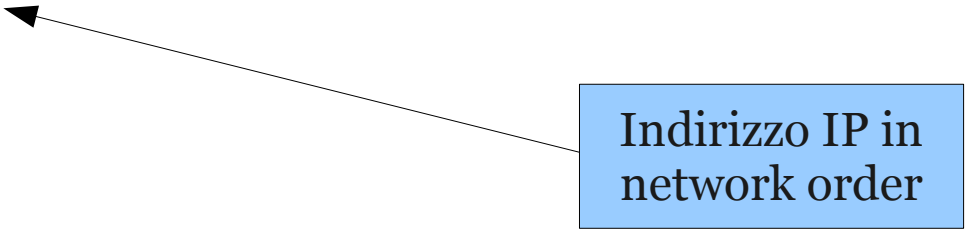
  h = Host
  n = Network (big endian)
  l = Long (4 bytes)
  s = Short (2 bytes)
- Funzionano a prescindere dalla codifica dell host!!!

Strutture dati per endpoint

- La struct che memorizza un endpoint è così composta:
- struct sockaddr_in {
  sa_family_t sin_family;
  u_int16_t sin_port;
  struct in_addr sin_addr;
  };



- struct in_addr {
  u_int32_t s_addr;
  };



Esempio

- Per memorizzare la porta 13 nel campo corrispondente della struct sockaddr_in epoint si può utilizzare:

  epoint.sin_port=htons(13);

- Per memorizzare l'indirizzo 192.132.34.22 all'interno della struct in_addr addr si può utilizzare:

  inet_aton("192.132.34.22",&addr.sin_addr);

- La funzione converte la stringa passata come primo argomento in un indirizzo di rete scritto in network order e lo memorizza nella locazione di memoria puntata dal secondo argomento
- Restituisce 0 se l'indirizzo non è valido o un valore diverso altrimenti

INADDR_ANY

- Ogni host può avere uno o più indirizzi IP collegati ad una o più interfacce di rete
- In tal caso le richieste di connessione da parte di un client possono arrivare su vari indirizzi
- Su macchine con più indirizzi è possibile:
  - consentire che il servizio sia raggiungibile su tutti gli indirizzi, specificando INADDR_ANY nel campo sin_addr della struttura sockaddr_in
  - restringere l'accesso al servizio alle connessioni proveniente da un indirizzo specifico, indicando l'indirizzo IP corrispondente nel campo sin_addr della struttura sockaddr_in
- Il primo metodo consente inoltre di implementare il server senza dover ogni volta ricavare l'indirizzo IP dell'host su cui viene eseguito

listen

- Per mettersi in ascolto di nuove connessioni si utilizza
- int listen(int fd, int coda);
- fd specifica il socket, precedentemente associato all'endpoint, da cui ricevere le connessioni mentre coda specifica il numero di connessioni da mantenere in attesa
- Quando il numero di connessioni in attesa è minore o uguale a coda, il processo client resta in attesa di connessione fino al raggiungimento di un time-out
- Quando il numero di connessioni in attesa supera coda, il client che prova a connettersi riceve immediatamente "connection refused"
- La listen può essere utilizzata soltanto per socket SOCK_STREAM e SOCK_SEQPACKET.

accept

- L'accettazione di una connessione da parte di un server viene effettuata utilizzando:
- int accept(int fd, struct sockaddr *client, socklen_t *dimensione_indirizzo);
- fd è il socket in ascolto, client è il puntatore ad una struct sockaddr allocata dal server che viene assegnata dall'accept che vi memorizza le informazioni sul client (indirizzo IP, numero di porta da cui il client si connette)
- È possibile ignorare queste informazioni passando NULL

accept

- La accept restituisce un nuovo descrittore oppure -1 in caso di errore
- Questo nuovo socket costituisce il canale di comunicazione con il client connesso, mentre il vecchio socket continua a restare in ascolto di nuove connessioni
- L'accept di default blocca il processo che la invoca se non vi sono connessioni in attesa, ma è possible specificare al socket di funzionare in modalità non-blocking e verificare la presenza di connessioni dal valore restituito

Esempio di server

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <time.h>
int main(int argc, char **argv) {
  int    lfd, cfd;
  struct sockaddr_in  sad;
  char   buff[4096];
  time_t ticks;
  if ((lfd=socket(AF_INET,SOCK_STREAM,0))<0)
    perror("socket"), exit(1);
  servaddr.sin_family=AF_INET;
  servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
  servaddr.sin_port=htons(1313);
  if (bind(lfd,(struct sockaddr *) &sad,sizeof(sad))<0)
    perror("bind"), exit(1);
```

Indirizzo IP
Porta

Esempio di server

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <time.h>
int main(int argc, char **argv) {
  int    lfd, cfd;
  struct sockaddr_in  sad;
  char   buff[4096];
  time_t ticks;
  if ((lfd=socket(AF_INET,SOCK_STREAM,0))<0)
    perror("socket"), exit(1);
  servaddr.sin_family=AF_INET;
  servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
  servaddr.sin_port=htons(1313);
  if (bind(lfd,(struct sockaddr *) &sad,sizeof(sad))<0)
    perror("bind"), exit(1);
```

Associazione socket-endpoint

Esempio di server

```
  if ( listen(lfd, 1024) < 0 )
    perror("listen"), exit(1);
  while (1) {
    if ((cfd=accept(lfd,(struct sockaddr *)NULL,NULL))<0)
      perror("accept"), exit(1);
    ticks = time(NULL);
    snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
    if (write(cfd,buff,strlen(buff))!=strlen(buff))
      perror("write"), exit(1);
    close(cfd);
  }
}
```

Ascolto

Esempio di server

```
  if ( listen(lfd, 1024) < 0 )
    perror("listen"), exit(1);
  while (1) {
    if ((cfd=accept(lfd,(struct sockaddr *)NULL,NULL))<0)
      perror("accept"), exit(1);
    ticks = time(NULL);
    snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
    if (write(cfd,buff,strlen(buff))!=strlen(buff))
      perror("write"), exit(1);
    close(cfd);
  }
}
```

Nuova Connessione

Esempio di server

```
  if ( listen(lfd, 1024) < 0 )
    perror("listen"), exit(1);
  while (1) {
    if ((cfd=accept(lfd,(struct sockaddr *)NULL,NULL))<0)
      perror("accept"), exit(1);
    ticks = time(NULL);
    snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
    if (write(cfd,buff,strlen(buff))!=strlen(buff))
      perror("write"), exit(1);
    close(cfd);
  }
}
```

Fornitura del servizio

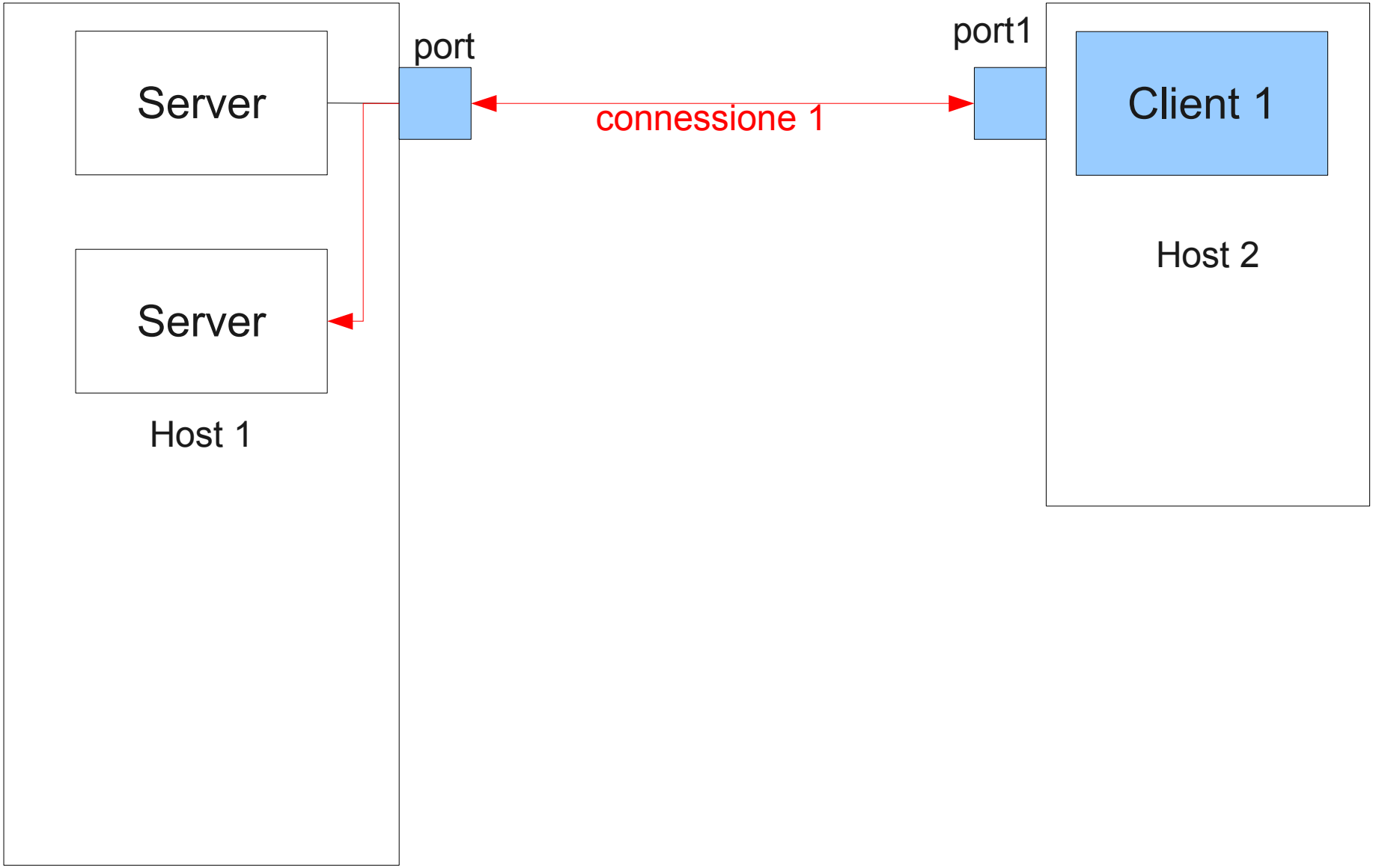
Esempio di server

```
  if ( listen(lfd, 1024) < 0 )
    perror("listen"), exit(1);
  while (1) {
    if ((cfd=accept(lfd,(struct sockaddr *)NULL,NULL))<0)
      perror("accept"), exit(1);
    ticks = time(NULL);
    snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
    if (write(cfd,buff,strlen(buff))!=strlen(buff))
      perror("write"), exit(1);
    close(cfd);
  }
}
```

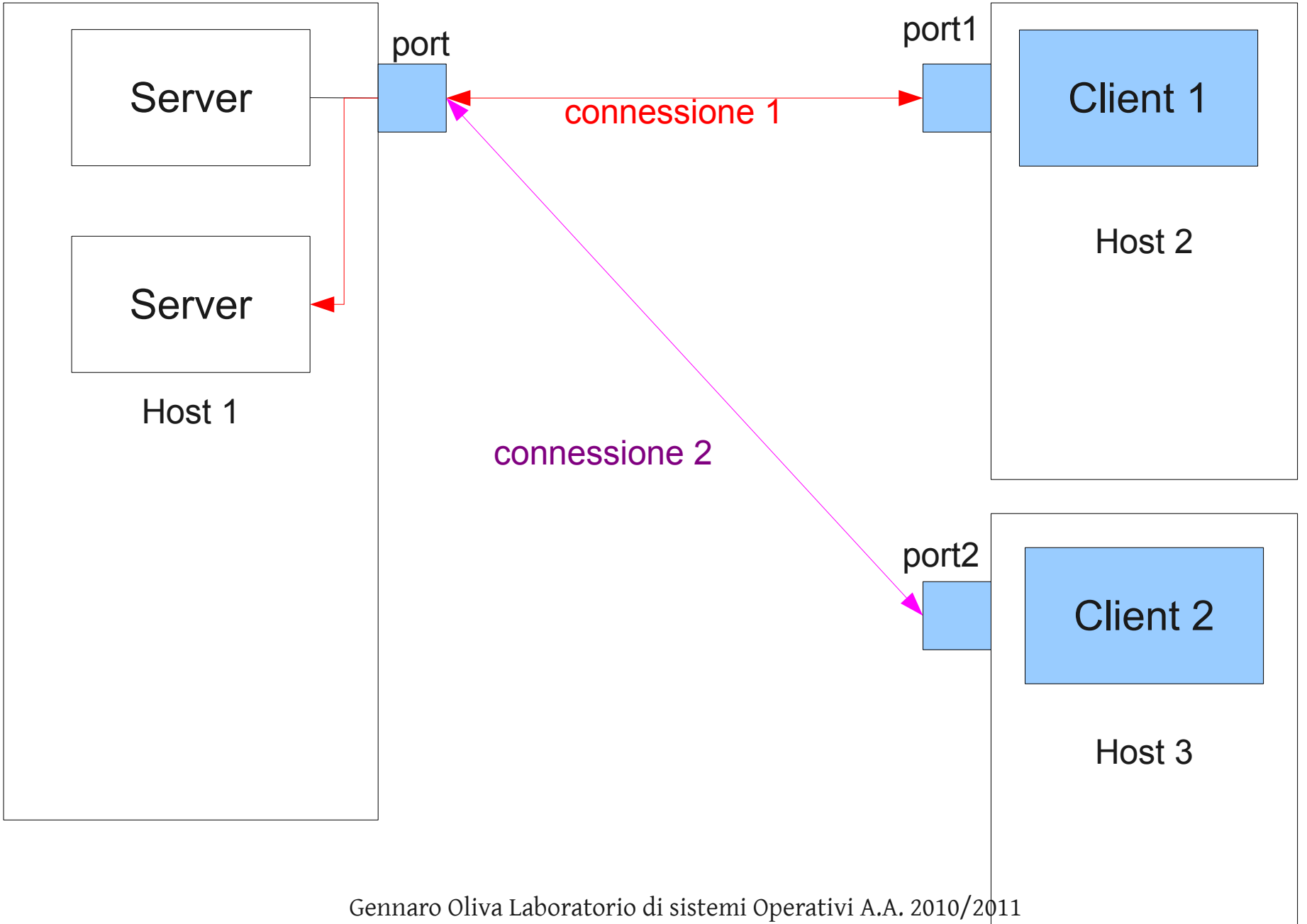
Chiusura connessione

Connessione client server

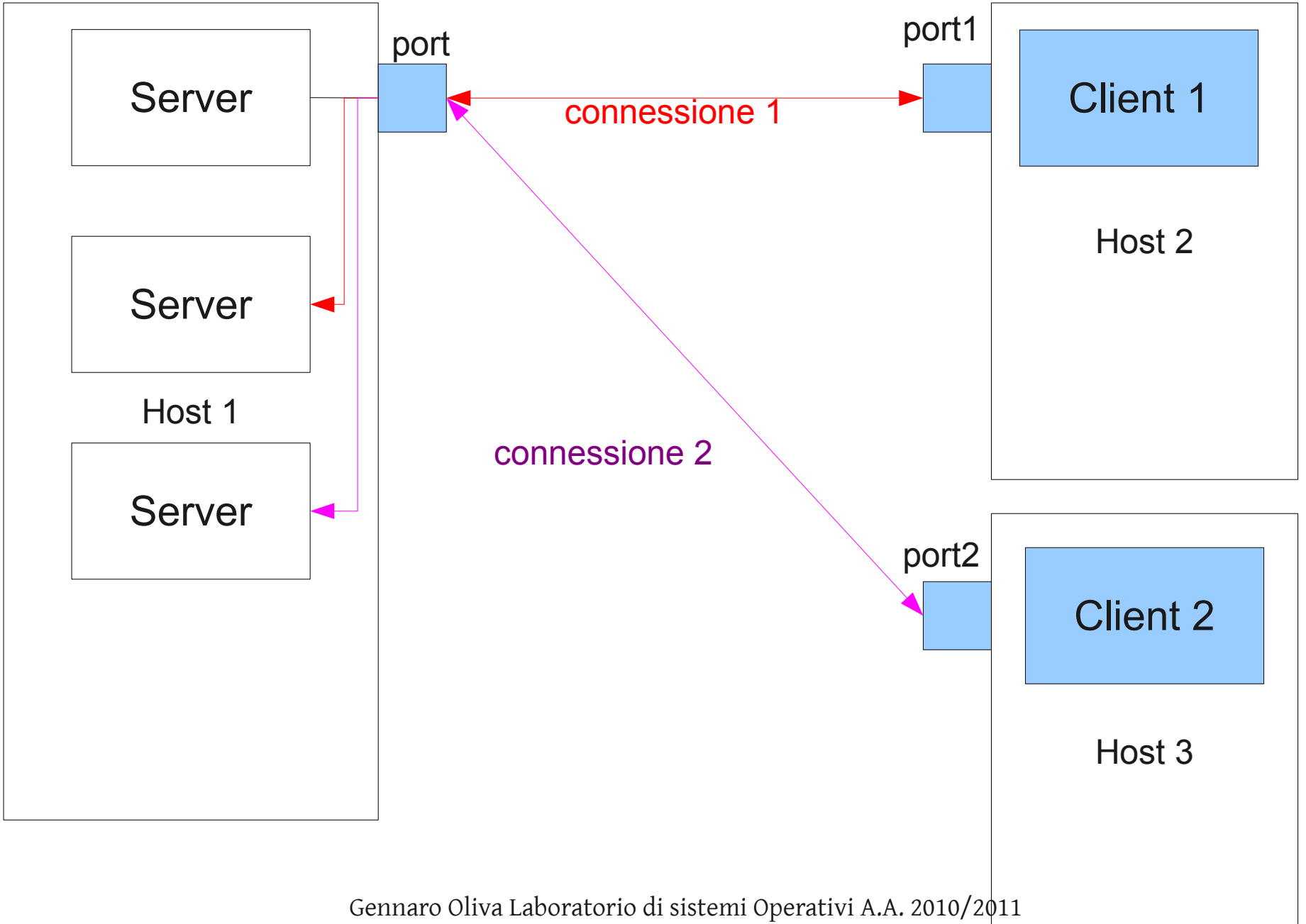
Connessione client server



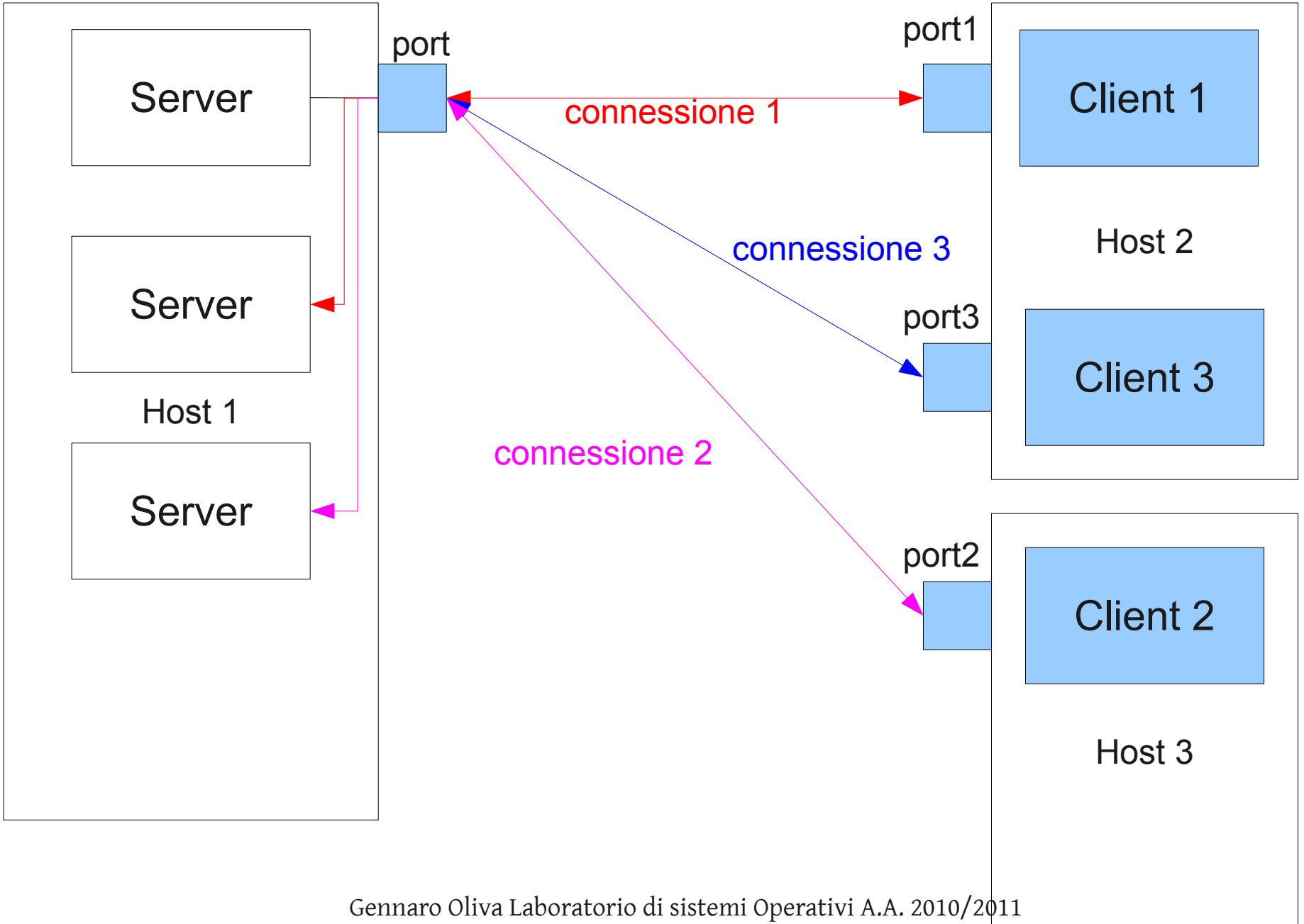
Connessione client server



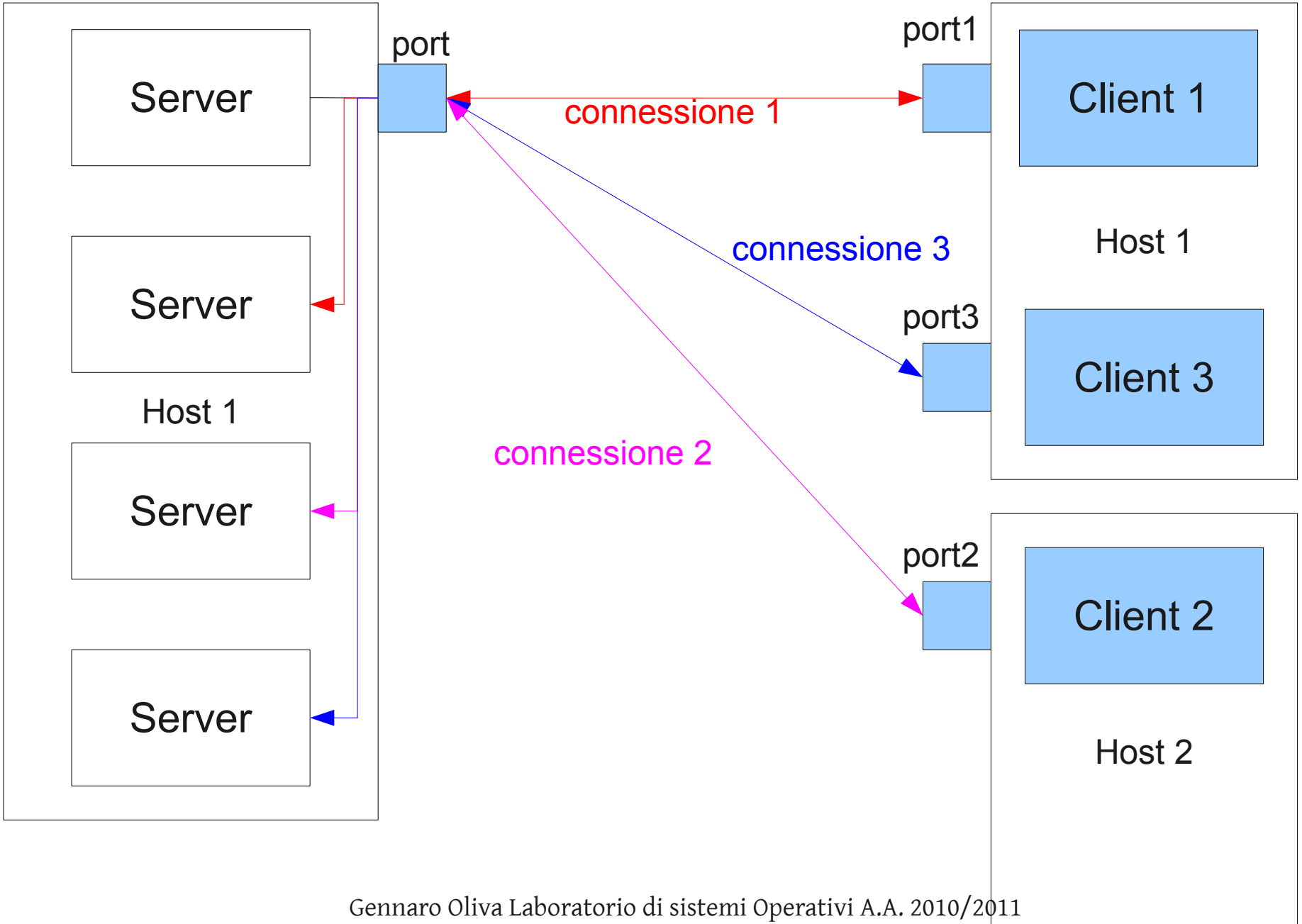
Connessione client server



Connessione client server



Connessione client server



Il client

- Un client effettua una richiesta di connessione al server utilizzando l'endpoint che lo identifica
- Un client effettua tipicamente la seguente sequenza di operazioni
  - Crea il socket usando socket
  - Stabilisce richiede una connessione usando connect
  - Interagisce con il server usando read e write
  - Chiude il socket usando close

connect

- La connessione al server avviene utilizzando:

  int connect(int fd, const struct sockaddr *saddr, socklen_t len);

- fd e un socket descriptor restituito da una precedente chiamata a socket
- saddr ad una struttura sockaddr contenente l'endpoint che identifica il server
- len contiene la dimensione in byte del secondo parametro calcolata utilizzando la funzione sizeof
- La system call connect restituisce `0`, in caso di successo, oppure -1 in caso contrario assegnando opportunamente la variabile errno

connect

int connect(int fd, const struct sockaddr *saddr, socklen_t len);

- A differenza di quando accade sul server con l'accept non viene generato nuovo file descriptor per identificare il canale di comunicazione, ma il canale di comunicazione è identificato da fd
- Il numero di porta in uscita utilizzato dal client (e quindi l'endpoint del client) è deciso dal kernel è viene prelevato dall'intervallo di porte effimere utilizzando una porta non impegnata

Esempio di client

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
int main(int argc, char **argv) {
  int    sockfd, n;  char   recvline[1025] ;
  struct sockaddr_in servaddr;
  if (argc != 2)
    fprintf(stderr,"usage: %s <Ipaddress>\n",argv[0]), exit (1);
  if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    perror("socket"), exit (1);
  servaddr.sin_family = AF_INET;
  servaddr.sin_port   = htons(1313);
  if (inet_aton(argv[1], &servaddr.sin_addr) == 0) {
      fprintf(stderr,"inet_aton error for %s\n", argv[1]);
      exit (1);
    }
  if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0)
    perror("connect"), exit(1);
  while ( (n = read(sockfd, recvline, 1024)) > 0) {
    if ( write(STDOUT_FILENO,recvline,n) < 0 )
      perror("write"),exit(1);
  }
  if (n < 0)
    perror("read"),exit(1);
  close(sockfd);
  exit(0);
}
```

Indirizzo

Porta

Esempio di client

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
int main(int argc, char **argv) {
  int    sockfd, n;  char   recvline[1025] ;
  struct sockaddr_in servaddr;
  if (argc != 2)
    fprintf(stderr,"usage: %s <Ipaddress>\n",argv[0]), exit (1);
  if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    perror("socket"), exit (1);
  servaddr.sin_family = AF_INET;
  servaddr.sin_port   = htons(1313);
  if (inet_aton(argv[1], &servaddr.sin_addr) == 0) {
      fprintf(stderr,"inet_aton error for %s\n", argv[1]);
      exit (1);
    }
  if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0)
    perror("connect"), exit(1);
  while ( (n = read(sockfd, recvline, 1024)) > 0) {
    if ( write(STDOUT_FILENO,recvline,n) < 0 )
      perror("write"),exit(1);
  }
  if (n < 0)
    perror("read"),exit(1);
  close(sockfd);
  exit(0);
}
```

Connessione

Esempio di client

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
int main(int argc, char **argv) {
  int    sockfd, n;  char   recvline[1025] ;
  struct sockaddr_in servaddr;
  if (argc != 2)
    fprintf(stderr,"usage: %s <Ipaddress>\n",argv[0]), exit (1);
  if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    perror("socket"), exit (1);
  servaddr.sin_family = AF_INET;
  servaddr.sin_port   = htons(1313);
  if (inet_aton(argv[1], &servaddr.sin_addr) == 0) {
      fprintf(stderr,"inet_aton error for %s\n", argv[1]);
      exit (1);
    }
  if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0)
    perror("connect"), exit(1);
  while ( (n = read(sockfd, recvline, 1024)) > 0) {
    if ( write(STDOUT_FILENO,recvline,n) < 0 )
      perror("write"),exit(1);
  }
  if (n < 0)
    perror("read"),exit(1);
  close(sockfd);
  exit(0);
}
```

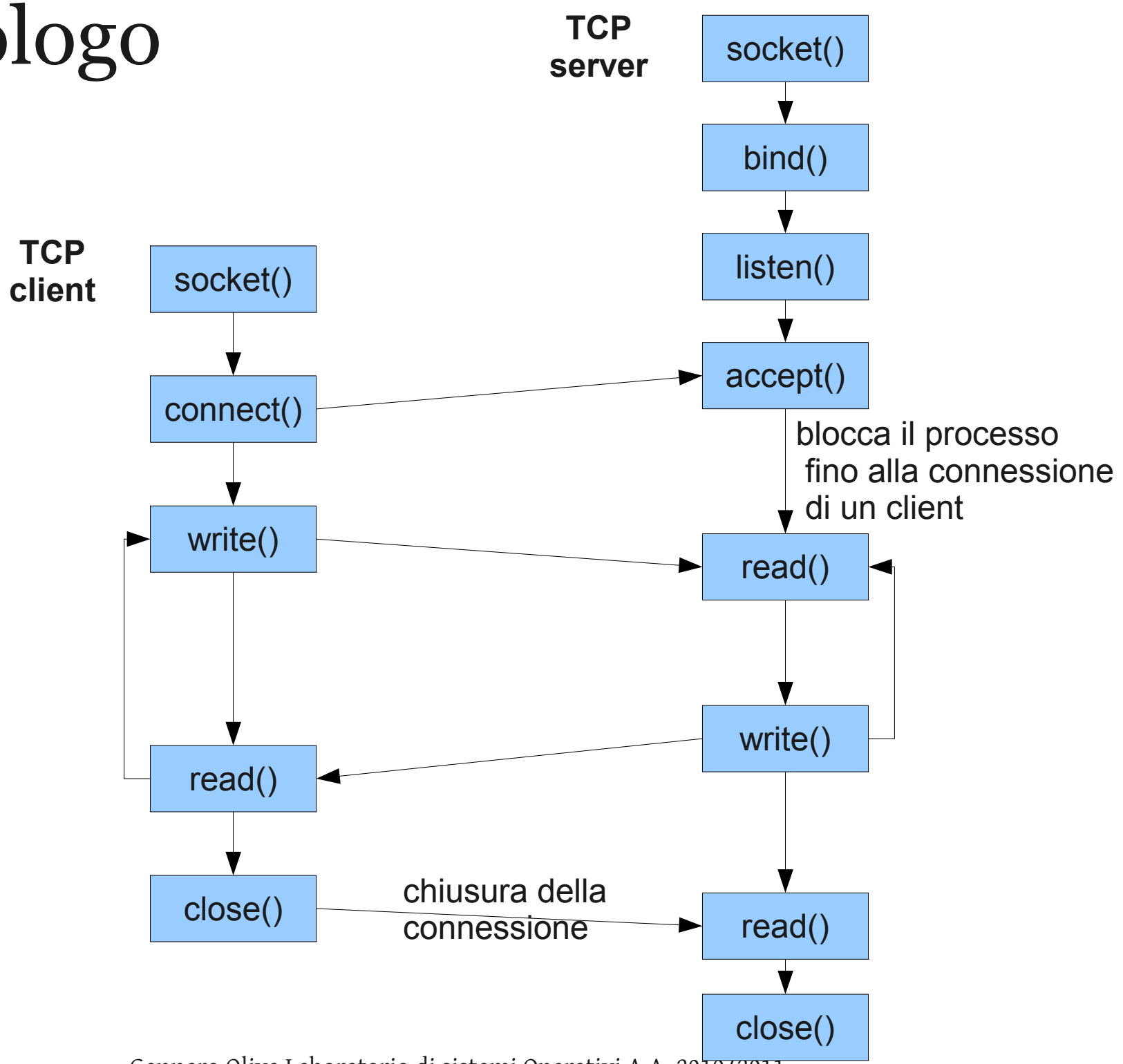
Lettura risposta

Esempio di client

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
int main(int argc, char **argv) {
  int    sockfd, n;  char   recvline[1025] ;
  struct sockaddr_in servaddr;
  if (argc != 2)
    fprintf(stderr,"usage: %s <Ipaddress>\n",argv[0]), exit (1);
  if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    perror("socket"), exit (1);
  servaddr.sin_family = AF_INET;
  servaddr.sin_port   = htons(1313);
  if (inet_aton(argv[1], &servaddr.sin_addr) == 0) {
      fprintf(stderr,"inet_aton error for %s\n", argv[1]);
      exit (1);
    }
  if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0)
    perror("connect"), exit(1);
  while ( (n = read(sockfd, recvline, 1024)) > 0) {
    if ( write(STDOUT_FILENO,recvline,n) < 0 )
      perror("write"),exit(1);
  }
  if (n < 0)
    perror("read"),exit(1);
  close(sockfd);
  exit(0);
}
```

Chiusura connessione

Riepologo


TCP server
socket()
bind()
listen()
accept()
blocca il processo fino alla connessione di un client
read()
write()
read()
close()
TCP client
socket()
connect()
write()
read()
close()
chiusura della connessione

I/O su socket

- Nella realizzazione di applicazioni client-server è necessario stabilire un protocollo applicativo in modo che ogni sequenza di byte scritta sul socket venga prelevata per intero ed interpretata coerentemente dalla controparte
- Comportamento di default per la lettura su socket:
  - la lettura su socket è bloccante se non vi sono dati in attesa.
  - la read ritorna zero quando non vi sono dati in attesa di essere letti ed il socket è stato chiuso dalla controparte
  - essendo la lettura da socket una operazione "lenta", è possible che la read venga interrotta e ritorni un numero di byte inferiore a quello attesi nel qual caso è necessario invocare nuovamente la socket per leggere i byte restanti
- Comporamento di default per la scrittura su socket:
  - la write può essere interrotta durante la scrittura ed in tal caso è deve essere invocata nuovamente per scrivere i byte restanti
  - la scrittura su un socket chiuso dalla controparte causa la generazione del segnale SIGPIPE, la cui azione di default è la terminazione del processo
  - se il segnale SIGPIPE viene ignorato esplicitamente la write restituisce -1 e imposta errno a EPIPE

Server iterativi e concorrenti

- L'implementazione di server iterativi ha due limiti fondamentali:
- Il server può essere bloccato dal client in attesa di input;
- I client connessi non possono essere serviti finchè il server non termina la sessione con il client corrente
- Per migliorare l'efficienza del server, è opportuno utilizzare un architettura di tipo concorrente

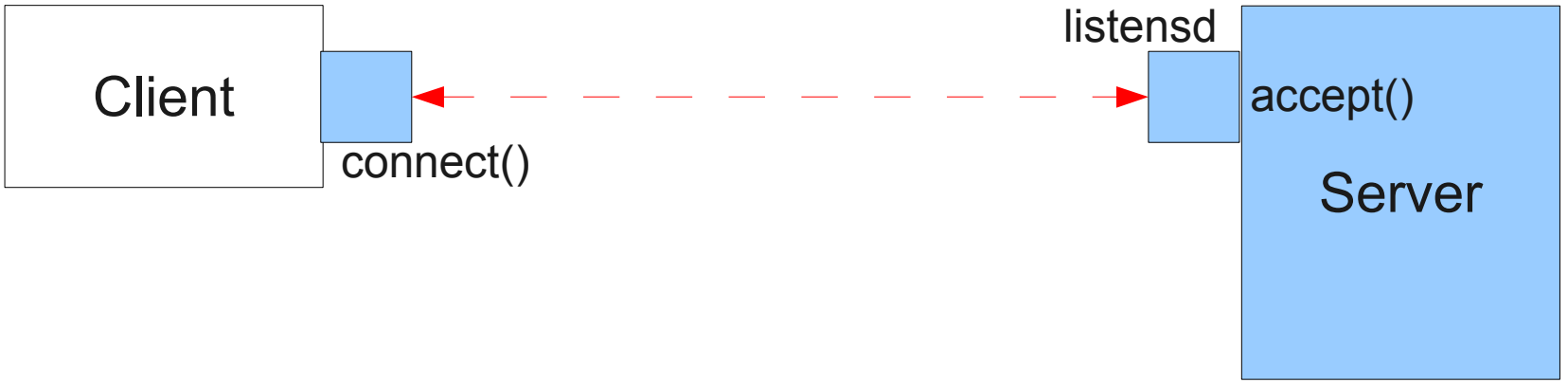
Server concorrenti

- In un server concorrente per ogni nuova connessione da client il server genera un nuovo figlio deputato a gestirla
- Il server esegue una fork dopo ogni accept
  - il padre torna subito ad eseguire una nuova accept
  - il figlio gestisce la connessione con un solo client e poi termina
- In alternativa alla programmazione multi-processo è possibile realizzare la stessa infrastruttura utilizzando una programmazione multi-thread

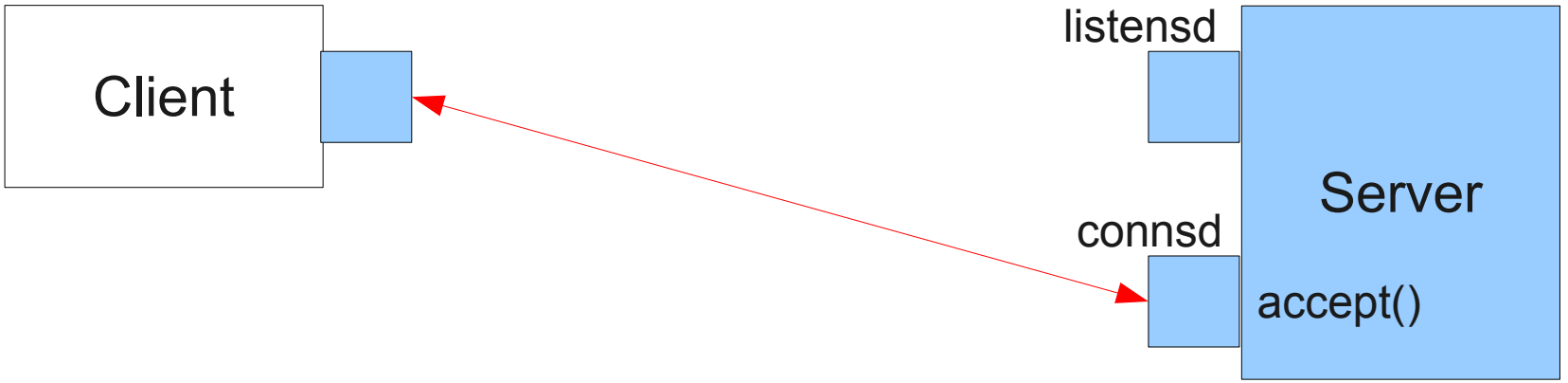
Server concorrente

- ```
  socket(...); bind(...); listen(...);
  while (1) {
  connsd=accept(listensd, NULL, NULL);
    if ( (pid = fork() ) == 0 ) {
      close(listensd);
      …
      exit(0);
      }
  close(connsd);
  }
  ```
  ```

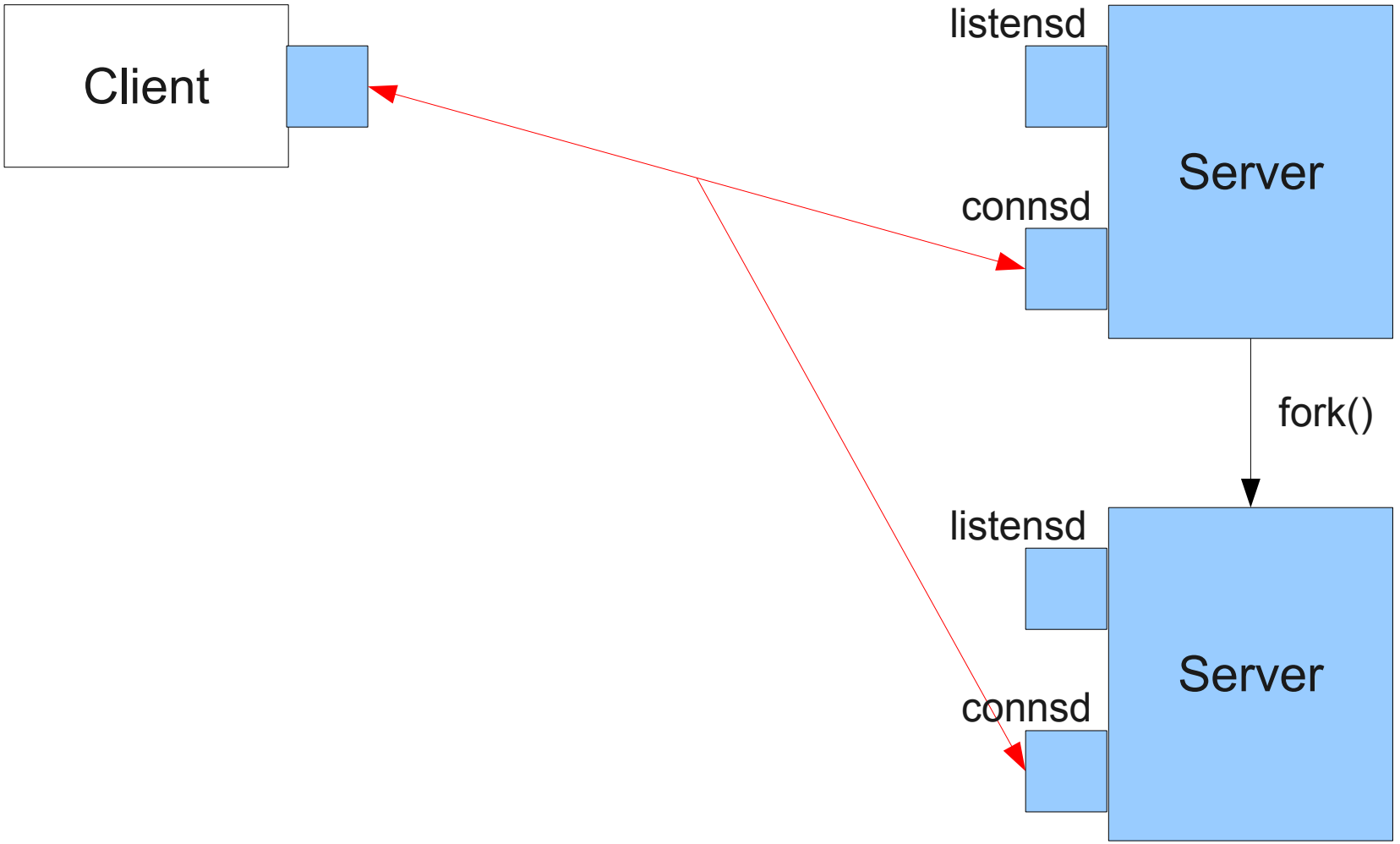
Server concorrente



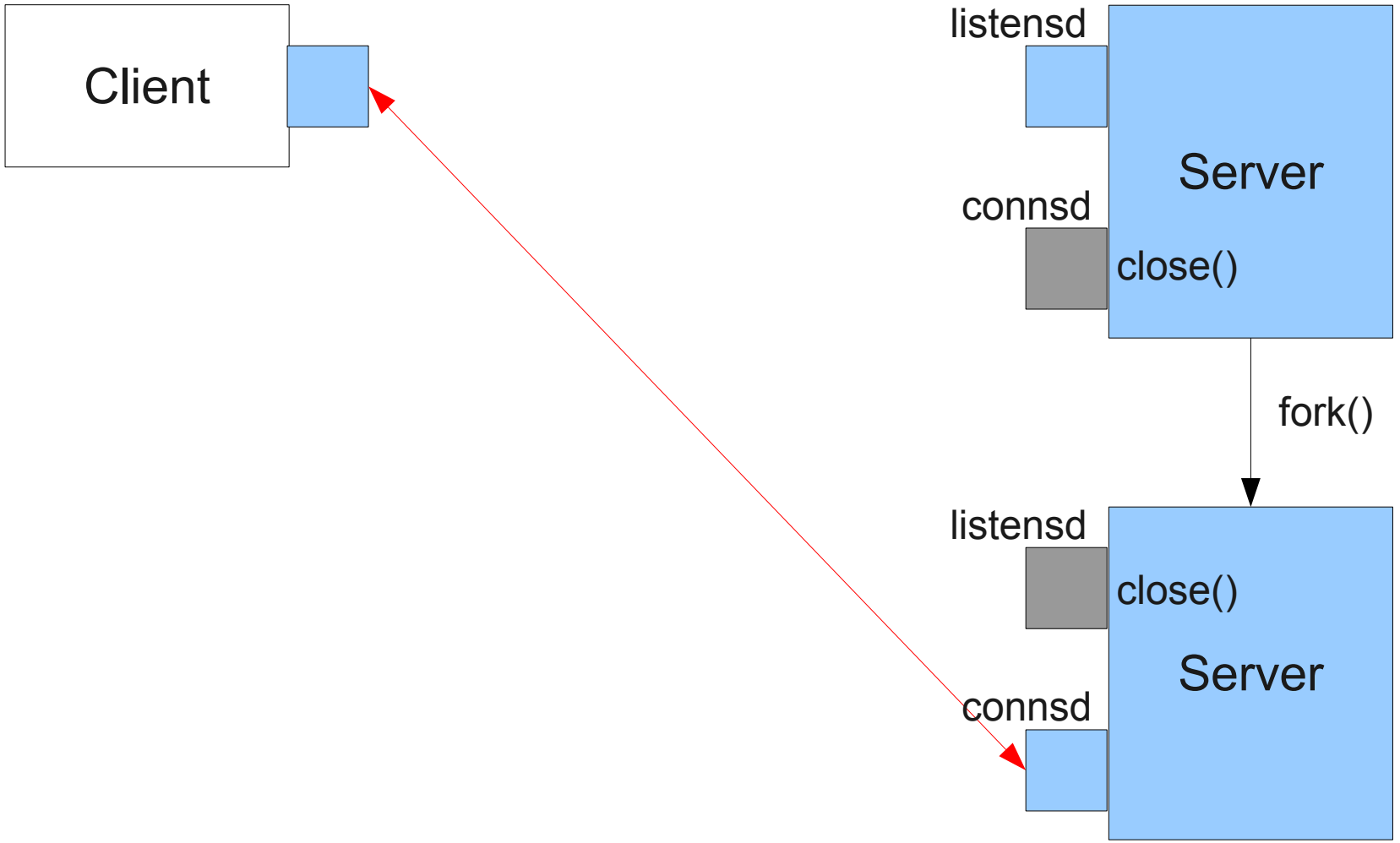
Server concorrente



Server concorrente



Server concorrente



Nomi simbolici

- Memorizzare indirizzi IP risulta complicato da memorizzare per le persone
- Con IPv6 gli indirizzi diventeranno particolarmente lunghi utilizzando una notazione di otto numeri a quattro cifre esadecimali sparate da ':' (ad esempio 2001:0db8:85a3:0000:0000:8a2e:0370:7334)
- È possibile attribuire ad un indirizzo IP un nomi testuale simbolico più semplici da ricordare
- Esempio: shell.studenti.unina.it

Nomi simbolici

- Sulla rete Internet gli host vengono identificati mediante un nome simbolico detto fully qualified domain name (FQDN)
- Il FQDN è composto dal nome locale dell'host e dal dominio di appartenenza
- shell.studenti.unina.it (192.132.34.22)
- L'associazione tra nomi simbolici degli host ed i corrispondenti indirizzi viene effettuata dal DNS (Domain Name System)

Domain Name System

- Il DNS è un database distribuito che memorizza le altre le seguenti tipologie di record:
  - A che associano FQDN ad indirizzi IPv4
  - AAAA che associano FQDN ad indirizzi IPv6
  - PTR che associano un indirizzo IP ad un hostname
  - MX che specificano chi agisce da mail exchanger per un determinato dominio
  - CNAME che forniscono alias per un host

Risoluzione

- L'operazione di trovare l'indirizzo IP associato ad un hostname viene detta risoluzione diretta
- L'operazione contraria e' detta risoluzione inversa
- La risoluzione sulla rete Internet viene effettuata solitamente effettuando query ad un server DNS
- È possibile interrogare il DNS da shell per risoluzioni dirette ed inverse utilizzando il comando host
- Esempio:

  $ host shell.studenti.unina.it
  shell.studenti.unina.it has address 192.132.34.22
  $ host 192.132.34.22
  22.34.132.192.in-addr.arpa domain name pointer shell.studenti.unina.it

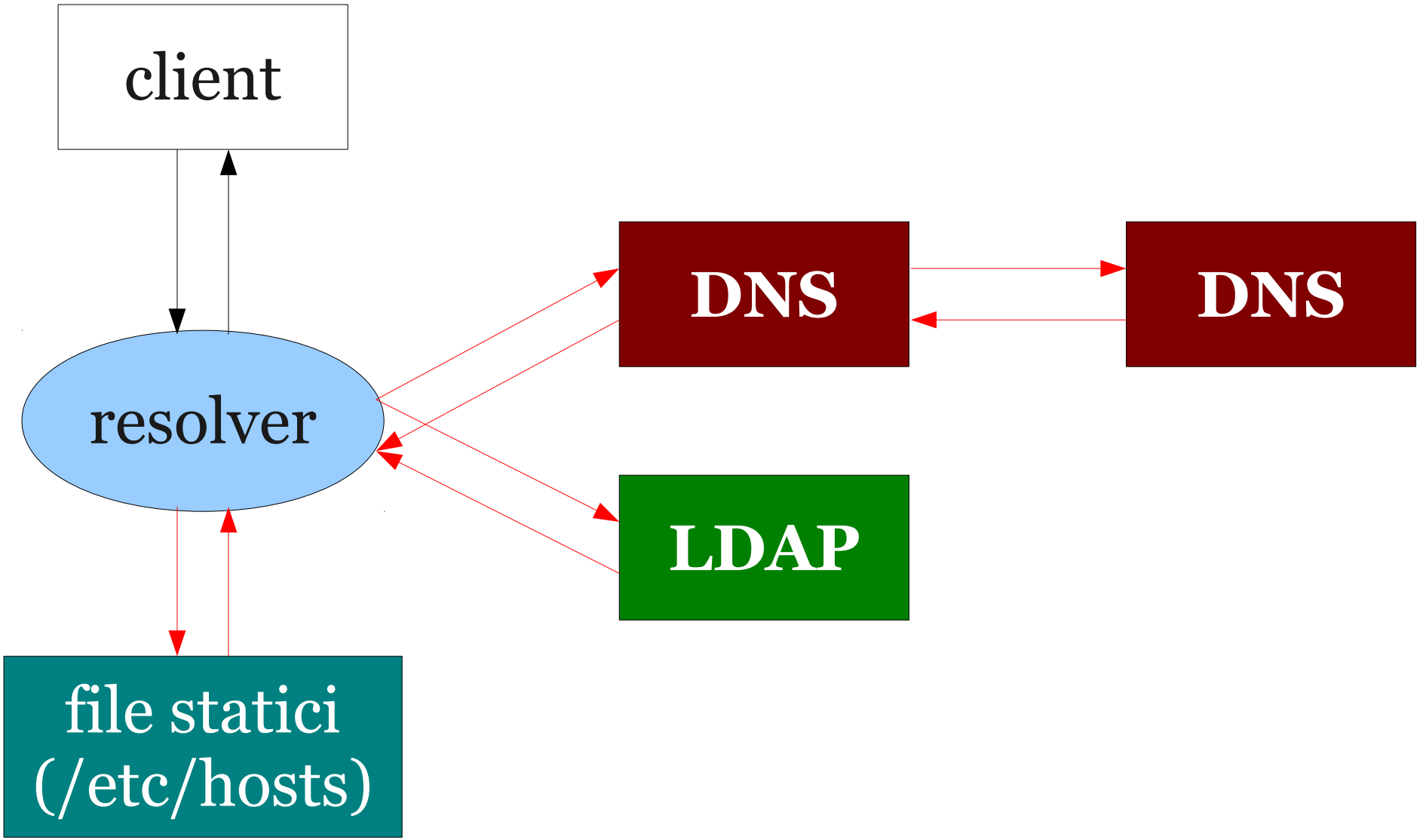
resolver

- Il linguaggio C fornisce un insieme di routine detto resolver che consentono di effettuare query al DNS o più in generale di effettuare una risoluzione
- Il DNS non e' l'unico servizio a fornire associazione tra indirizzi e host
- Alternative al DNS sono il file locale /etc/hosts o servizi come NIS o LDAP

/etc/hosts

- Il file /etc/hosts è un file di testo che associa indirizzi IP e hostname le cui linee hanno formato:
- IP hostname [alias...]
- Di regola il file /etc/hosts viene consultato prima di interrogare il DNS per cui si è solito inserirvi gli host acceduti più frequentemente
- L'ordine con cui il resolver utilizza le alternative possibili e' stabilito dal system administrator
- Nei sistemi Linux questa informazione e' conservata nel file /etc/nsswitch.conf
- Il nome del server DNS e' contenuto nel file /etc/resolv.conf

Funzionamento del resolver



gethostbyname

- In un programma C, la risoluzione mediante resolver può essere effettuato utilizzando:

  struct hostent *gethostbyname (const char *nome);

- nome è una stringa contenente un nome simbolico oppure un indirizzo IP da risolvere
- La funzione restituisce l'indirizzo ad una struct hostent contenente varie informazioni sull'host passato come argomento oppure NULL in caso di errore
- La struct restituita non è allocata nel programma chiamante, ma all'interno degli header della libreria netdb.h e può essere riutilizzata nelle invocazioni successive della stessa system call (non è thread-safe).

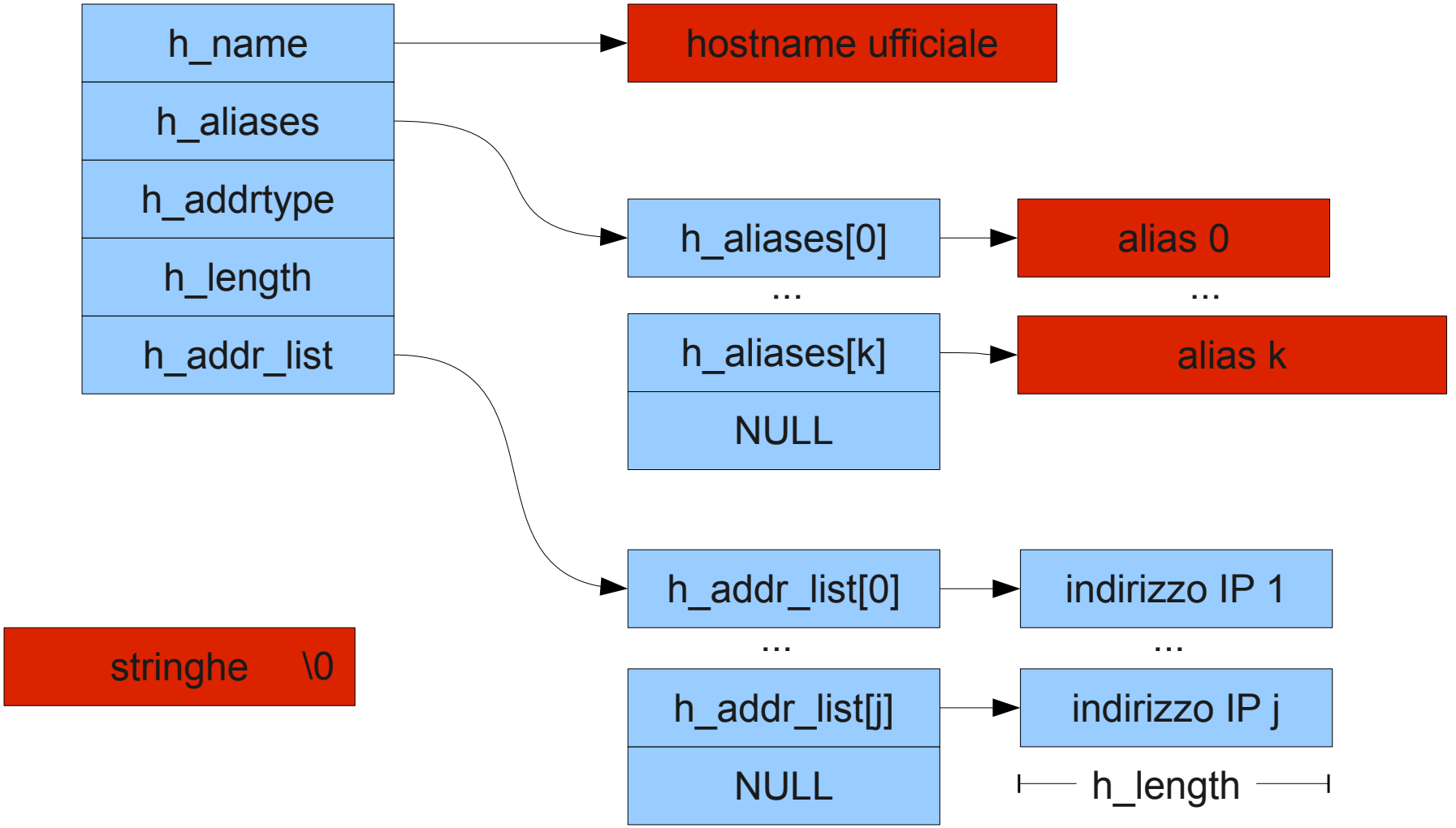
La struttura restituita

```
struct hostent {
    stringa zero-terminata
    char *h_name; /* nome canonico dell'host */
    array di string il cui ultimo elemento e null
    char **h_aliases; /* lista di alias */
    famiglia degli indirizzi
    int h_addrtype; /* famiglia dell'indirizzo */
    lunghezza degli indirizzi
    int h_length; /* lunghezza dell'indirizzo */
    lista di indirizzi IP dell'host
    char **h_addr_list; /* lista di indirizzi */

}
```

hostent

- Ad ogni nome simbolico possono essere associati più IP e viceversa



inet_ntop

- Per convertire un indirizzo IP da network order (intero a 32 bit o uint32_t) a notazione dotted è possibile utilizzare:

  const char *inet_ntop(int f, const void *src,
  char *dst, socklen_t cnt);

- Converte l'indirizzo memorizzato in formato network in src in notazione dotted (192.167.11.34) memorizzandolo in dst
- f e' la famiglia dell'indirizzo (nel caso del TCP AF_INET)
- cnt è la lunghezza di dst

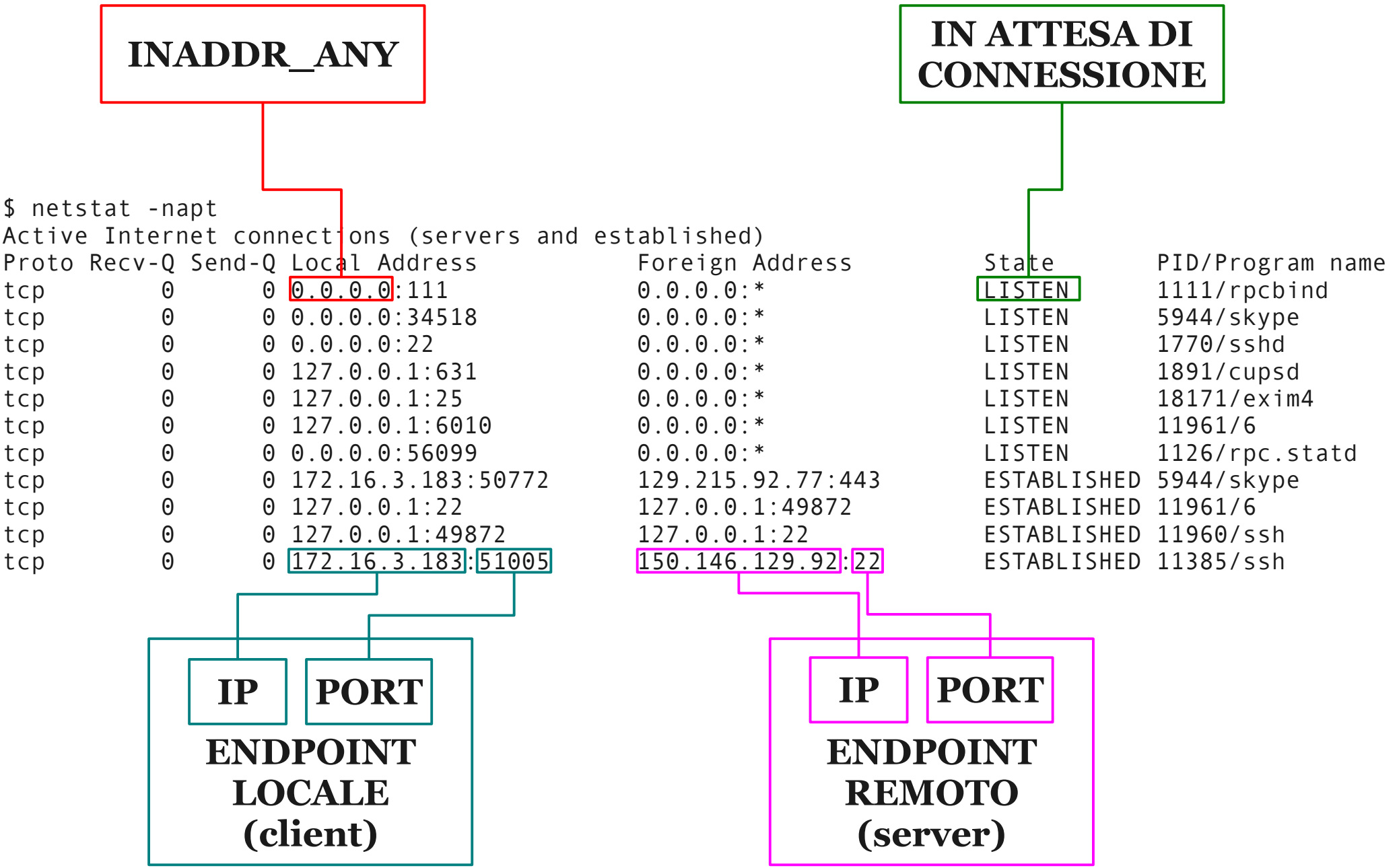
Esempio d'uso di gethostbyname

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
int main(int argc, char *argv[]) {
  struct hostent *p;
  char name[16];
  if (argc != 2)
    exit(1);
  p = gethostbyname(argv[1]);
  if ( p == NULL )
    herror("gethostbyname"), exit(1);
  if (!inet_ntop(AF_INET,p->h_addr_list[0],name,16))
    perror("inet_ntop"), exit(1);
  printf("%s\n",name);
  exit(0);
}
```

netstat

- netstat [-t] [-all] [-p] [-n]
- elenca i socket attivi sul sistema
- “-t” mostra solo i socket TCP (ovvero AF_INET, SOCK_STREAM)
- “-a” mostra tutti i socket anche quelli in ascolto
- “-p” specifica il pid del processo che utilizza il socket
- “-n” mostra gli indirizzi IP e le porte in formato numerico (invece di simbolico) più veloce perchè non effettua la risoluzione

Esempio di output di netstat



Telnet e ifconfig

- Il comando telnet
  - crea un socket e lo collega all'endpoint specificato sulla linea di comando (omettendo la porta lo collega alla 23)
  - Esempio:

    $ telnet pop.gmail.com 110
  - si mette in comunicazione con il server pop3 di google mail
  - ciò che si scrive sullo standard input viene inviato sul socket
  - ciò che proviene dal socket viene stampato su standard output
- Il comando telnet viene spesso utilizzato come client generico nel debug del protocollo applicativo di un server
- Il comando /sbin/ifconfig mostra gli indirizzi IP del sistema