Laboratorio di Reti di Calcolatori

Lezione 3

Berkeley Sockets

- I socket di Berkeley sono un'API che definisce una libreria C per comunicazioni inter-processo anche su rete
- Introdotti con la versione 4.2 di BSD Unix (nel 1983)
- Sono lo standard de facto per la realizzazione di applicazioni di rete

Socket

- I **socket** sono uno dei principali **meccanismi di comunicazione** utilizzato in ambito Unix
  - meccanismi di comunicazione **interprocesso**
- Un **socket** costituisce un **canale** di comunicazione **fra due processi** su cui si possono **leggere** e **scrivere** dati
- I socket consentono la comunicazione fra processi
  - sulla stessa macchina
  - su macchine connesse attraverso una rete
- Un **socket** in ambito Unix è **rappresentato** da un **descrittore** di file

Struttura di un client

- Crea il socket
- Si connette ad un server
- ...
- Chiude il socket

- socket(...)
- connect(...)
- ...
- close(...)

```
#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) {
    fprintf(stderr,"socket error\n");
    exit (1);
  }
    servaddr.sin_family = AF_INET;
    servaddr.sin_port   = htons(65000);
    if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0) {
      fprintf(stderr,"inet_pton error for %s\n", argv[1]);
      exit (1);
    }
```

```
if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0) {
    fprintf(stderr,"connect error\n");
    exit(1);
  }
  while ( (n = read(sockfd, recvline, 1024)) > 0) {
    recvline[n] = 0;
    if (fputs(recvline, stdout) == EOF) {
      fprintf(stderr,"fputs error\n");
      exit(1);
    }
  }
  if (n < 0) {
    fprintf(stderr,"read error\n");
    exit(1);
  }
  exit(0);
}
```

Socket

- int socket(int famiglia, int tipo, int protocollo);

Famiglia	Scopo	man
AF_UNIX,PF_LOCAL	Local communication	unix(7)
AF_INET	IPv4 Internet protocols	ip(7)
AF_INET6	IPv6 Internet protocols	
AF_IPX	IPX - Novell protocols	
AF_NETLINK	Kernel user interface device	netlink(7)
AF_X25	ITU-T X.25 / ISO-8208 protocol	x25(7)
AF_AX25	Amateur radio AX.25 protocol	
AF_ATMPVC	Access to raw ATM PVCs	
AF_APPLETALK	Appletalk	ddp(7)
AF_PACKET	Low level packet interface	packet(7)

Definiti in <sys/socket.h>

Socket

- int socket(int famiglia, int tipo, int protocollo);

Famiglia	Scopo	man
AF_UNIX,PF_LOCAL	Local communication	unix(7)
AF_INET	IPv4 Internet protocols	ip(7)
…		

- N.B. E' possible trovare la definizione degli stessi protocolli con suffisso **PF** invece di **AF**
  - AF: Address Family
  - PF: Protocol Family
- Rappresentano lo stesso **identico** protocollo

Definiti in <sys/socket.h>

Socket - tipo

- int socket(int famiglia, int tipo, int protocollo);
  - SOCK_STREAM canale bidirezionale, sequenziale affidabile che opera su connessione. I dati vengono ricevuti e trasmessi come un flusso continuo
  - SOCK_DGRAM usato per trasmettere pacchetti di dati di lunghezza massima prefissata (datagram), indirizzati singolarmente senza connessione in maniera non affidabile.
  - SOCK_SEQPACKET canale bidirezionale, sequenziale e affidabile che opera su connessione. I dati vengono trasmessi per pacchetti di dimensione massima fissata e vanno letti integralmente
  - SOCK_RAW canale di basso livello per accedere ai protocolli di rete e alle varie interfacce. Solitamente non utilizzato dalle applicazioni
  - SOCK_RDM canale di trasmissione di dati affidabile in cui non è garantito l'ordine di arrivo dei pacchetti.
  - SOCK_PACKET Obsoleto

Socket - tipo

- int socket(int famiglia, int tipo, int protocollo);
  - 0: Tale valore definisce il protocollo standard per ogni coppia *(famiglia, protocollo)*. Non tutti i protocolli sono implementati in ogni architettura di rete…usiamo sempre 0

Indirizzo TCP/IP del server

- `struct sockaddr_in {`

```
    sa_family_t sin_family;
    u_int16_t sin_port;
    struct in_addr sin_addr;
};
```

AF_INET

Porta in network order

- `struct in_addr {`

```
    u_int32_t s_addr;
};
```

indirizzo IP in network order

Endianess

- Come viene memorizzato un dato superiore al byte?
- Si consideri un intero costituito da 4 byte
- La codifica “Big Endian” (prima il byte piu' significativo):

  00000000 00000000 10000000 00000001

- La codifica “Little Endian” (prima il byte meno significativo):

  00000001 10000000 00000000 00000000

Codifica dati

- L'utilizzo di una codifica è stabilito dall'architettura del processore
  - Intel - Apple Silicon (default little endian)
  - Motorola (big endian)
- La suite di protocolli internet utilizza **big endian** pertanto sono necessarie funzioni di conversione
- conversioni tra unsigned:

- #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)

Porta associata al servizio

- servaddr.sin_port = htons(13)

- L'istruzione memorizza nel campo sin_port della struct serveaddr l'intero 13 scritto in network order

- 13 e' la porta su cui risponde il server che stiamo contattando

Conversione dell'IP del server

- **inet_pton**(PF_INET, argv[1], &servaddr.sin_addr)
  - IPV4 e IPV6
- Questa funzione converte la stringa passata come secondo argomento in un indirizzo di rete scritto in network order e lo memorizza nella locazione di memoria puntata dal terzo argomento

- Il nostro programma quindi dovra' specificare come argomento l'indirizzo IP del server a cui fare la richiesta

- La funzione restituisce un numero negativo o zero in caso di errore ed un numero positivo in caso di successo

Conversione dell'IP del server

- Solo per IPV4
- #include <arpa/inet.h>
- in_addr_t **inet_addr**(const char *strptr)
  - Converte la stringa dell'indirizzo dotted decimal in nel numero IP in network order.
- int **inet_aton**(const char *src, struct in_addr *dest)
  - Converte la stringa dell'indirizzo dotted decimal in un indirizzo IP.
- char ***inet_ntoa**(struct in_addr addrptr)
  - Converte un indirizzo IP in una stringa dotted decimal.

Connect

- connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr))
  - Connette il socket *sockfd* all'indirizzo serv_addr
  - Il terzo argomento e' la dimensione in byte della struttura
  - Il cast e' necessario in quanto la funzione puo' essere utilizzata con diversi tipi di socket e quindi con diversi tipi di strutture
  - Restituisce 0 (successo) oppure -1 (errore)

Stampa del messaggio

```
while ( (n = read(sockfd, recvline, 1024)) > 0)
{
  recvline[n] = 0;
  if (fputs(recvline, stdout) == EOF) {
    fprintf(stderr,"fputs error\n");
    exit(1);
}
```

- Prima di terminare il programma stampa a video il messaggio ricevuto dal server
- Per leggere e scrivere su socket si utilizzano le system call **read** e **write**

Struttura di un'applicazione server elementare

- Crea il socket
- Gli assegna un indirizzo
- Si mette in ascolto
- Accetta una nuova connessione
- ...
- Chiude il socket

- socket(...)
- bind(...)
- listen(...)
- accept(...)
- ...
- close(...)

```
#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          listenfd, connfd;
  struct sockaddr_in  servaddr;
  char         buff[4096];
  time_t       ticks;
  if ( ( listenfd = 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(13);
  if ( bind(listenfd, (struct sockaddr *) &servaddr,
                                          sizeof(servaddr)) < 0 )
{
    perror("bind");
    exit(1);
  }
```

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

```
#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>
```

Direttive al preprocessore

Direttive al preprocessore

- read,write,close
- socket, bind, listen, connect
- struct sockaddr_in
- exit
- time
- strlen

- <unistd.h>
- <sys/types.h> <sys/socket.h>
- <arpa/inet.h>
- <stdlib.h>
- <time.h>
- <string.h>

```
int main(int argc, char **argv)
{
  int          listenfd, connfd;
  struct sockaddr_in  servaddr;
  char         buff[4096];
  time_t       ticks;
  if ( ( listenfd = 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(65000);
  if ( bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0 ) {
    perror("bind");
    exit(1);
  }
```

Creazione della socket

Perror

- void perror(const char *s);
- La funzione perror produce un messaggio sullo standard error che descrive l'ultimo errore avvenuto durante una system call o una funzione di libreria
- Se l'argomento passato non è NULL, viene stampato prima del messaggio d'errore seguito da ':'
- Di solito si passa il nome della routine invocata

```
int main(int argc, char **argv)
{
  int          listenfd, connfd;
  struct sockaddr_in  servaddr;
  char         buff[4096];
  time_t       ticks;
  if ( ( listenfd = 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(65000);
  if ( bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0 ) {
    perror("bind");
    exit(1);
  }
```

Indirizzo Server

INADDR_ANY

- servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
  - INADDR_ANY viene utilizzato come indirizzo del server
  - L'applicazione accetterà connessioni da **qualsiasi** indirizzo associato al server
  - Se avessimo utilizzato 127.0.0.1 avremmo potuto eseguire **soltanto connessioni** dalla macchina su cui gira il server

```
int main(int argc, char **argv)
{
  int          listenfd, connfd;
  struct sockaddr_in  servaddr;
  char         buff[4096];
  time_t       ticks;
  if ( ( listenfd = 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(65000);
  if ( bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0 ) {
    perror("bind");
    exit(1);
  }
```

Assegnazione Indirizzo

Bind

- int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
  - Assegna l'indirizzo addr al socket sockfd
  - addr è un sockaddr di tipo generico
  - Nei socket TCP **fallisce** se la porta è in uso
  - addrlen è sizeof del secondo argomento
  - Restituisce 0 oppure -1

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

Messa in ascolto

Listen

- int listen(int sockfd, int lunghezza_coda);
- Mette il socket in modalità di ascolto in attesa di nuove connessioni
- Il secondo argomento specifica quante connessioni possono essere in attesa di essere accettate
- Restituisce 0 oppure -1

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

Accettazione nuova connessione

Accept

- int accept(int sockfd, struct sockaddr *clientaddr, socklen_t *addr_dim);
  - Il secondo e terzo argomento servono ad identificare il client possono essere **NULL**
  - Restituisce un nuovo descrittore o -1
  - Il nuovo socket e' associato alla nuova connessione
  - Il vecchio socket resta in ascolto

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

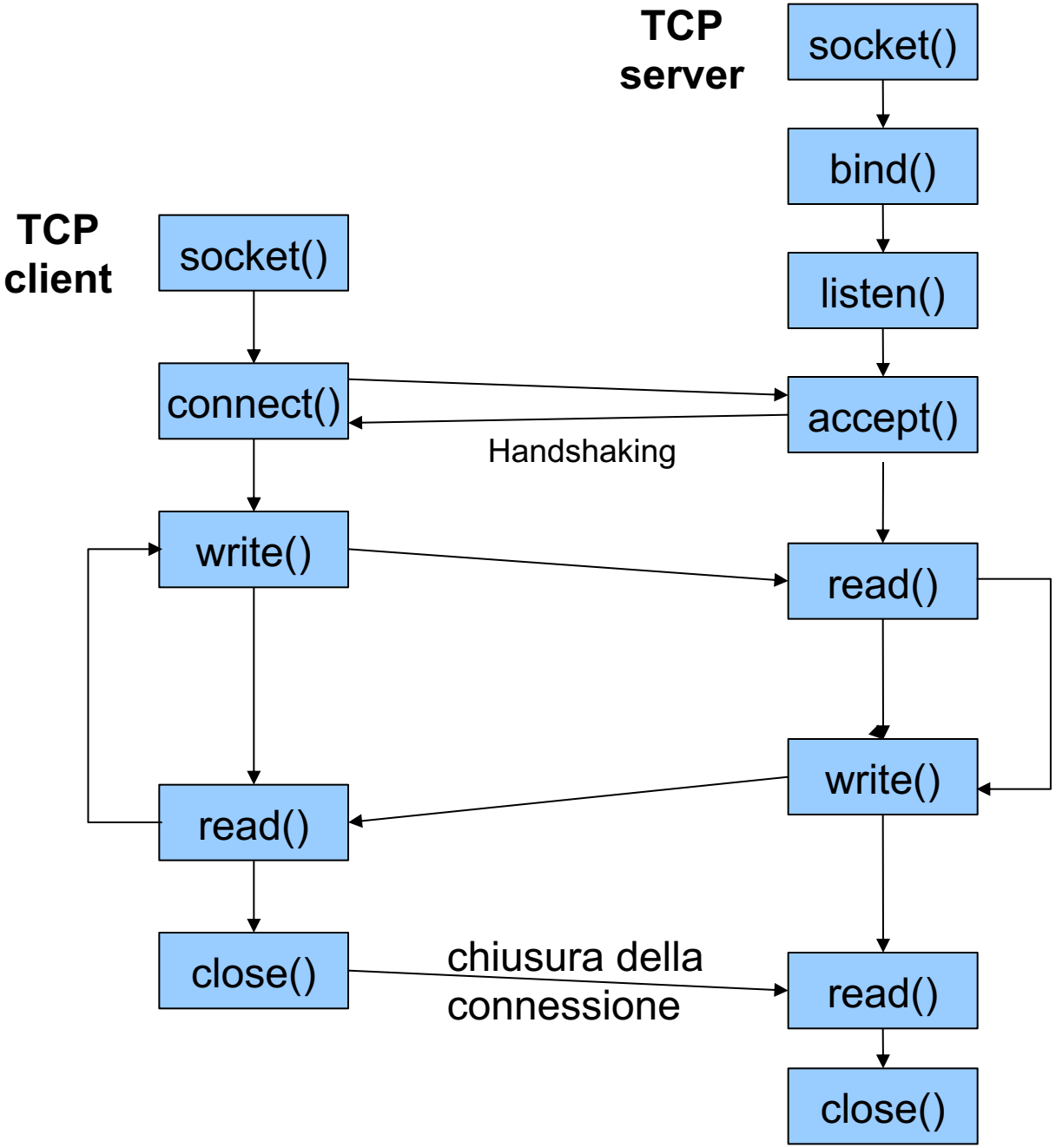
gestione richiesta

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

chiusura connessione

Close

- Per terminare una connessione TCP si utilizza la system call close
- Una volta invocato il descrittore del socket non e' piu' utilizzabile dal processo per operazioni di lettura o scrittura
- Il sottosistema di rete invia i dati in coda e successivamente chiude la connessione


TCP server
socket()
bind()
listen()
accept()
TCP client
socket()
connect()
Handshaking
write()
read()
write()
read()
close()
chiusura della connessione
read()
close()

Funzioni Wrapper

- Nei programmi reali e' necessario verificare la condizione di uscita di ogni chiamata a funzione
- Spesso gli errori determinano la necessità di terminare l'esecuzione
- Per migliorare la leggibilità del codice si possono definire delle funzioni wrapper che chiamano la funzione, ne verificano l'uscita e terminano l'esecuzione in caso di errore

Esempio di Funzione Wrapper per socket

La procedura Socket è un wrapper per socket

```
int Socket(int family, int type, int protocol)
{
     int  n;
     if ( (n = socket(family, type, protocol)) < 0) {
          perror("socket");
   exit(1);
}
     return(n);
}
```

Esercizi

- **Esercizio1:** Realizzare una libreria di wrapper per le chiamate alle procedure per le socket:
  - socket, connect, bind, listen, accept
- **Esercizio2:** Utilizzando le funzioni wrapper
  - riscrivere il client daytime
  - riscrivere il server daytime
  - test