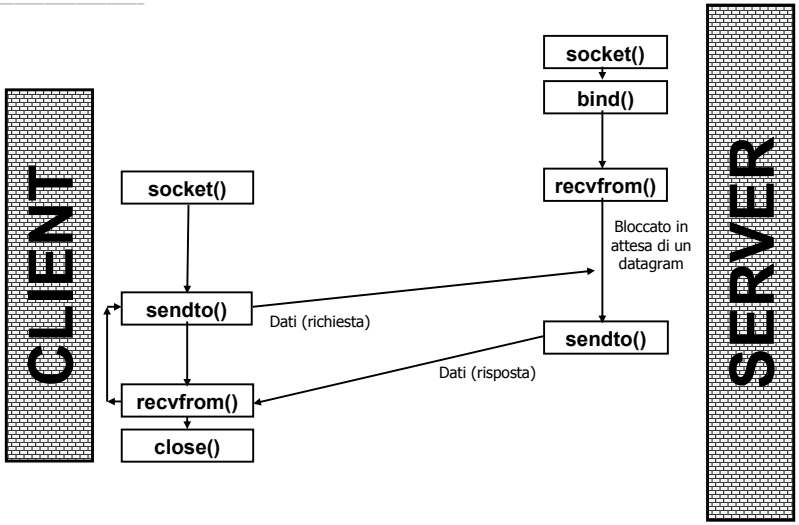
Reti Programmazione a. a. 2003/04

Socket UDP

Prof. Vincenzo Auletta
auletta@dia.unisa.it
http://www.dia.unisa.it/professori/auletta/

Università degli studi di Salerno
Laurea in Informatica

Struttura di un'Applicazione UDP



Funzioni di Input/Output


```
#include <sys/socket.h>
int recvfrom(int sd, void* buf, int n, int flags,
             struct sockaddr* from, socklen_t *len);
int sendto(int sd, const void* buf, int n, int flags,
             const struct sockaddr* from, socklen_t len);
```

- Per leggere o scrivere su un socket UDP si utilizzano funzioni di sistema differenti da TCP
  - devono specificare l'indirizzo del server e la sua lunghezza
- Restituiscono
  - numero di byte letti/scritti se OK (>= 0)
  - -1 se c'è un errore
- La sendto esce dopo aver completato la spedizione
  - non garantisce che i dati siano arrivati a destinazione

Parametri di sendto

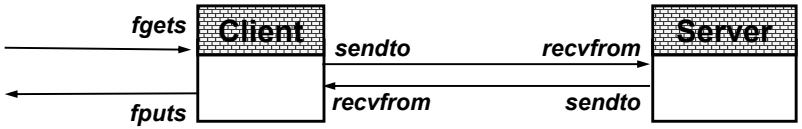
- int sd
  - Descrittore del socket
- void* buf
  - Buffer da cui leggere i byte da inviare
- size_t n
  - Numero di byte da inviare
- int flags
  - Vedremo in seguito
- struct sockaddr* to
  - Indirizzo del socket a cui inviare il datagram
- socklen_t len
  - Lunghezza dell'indirizzo puntato da to

Parametri di recvfrom

- int sd
  - Descrittore del socket
- void* buf
  - Buffer in cui scrivere i byte ricevuti
- size_t n
  - Massimo numero di byte che si possono ricevere
- int flags
  - Vedremo in seguito
- struct sockaddr* from
  - Indirizzo del socket da cui ha ricevuto il datagram
  - Può essere NULL
- socklen_t *len
  - Lunghezza dell'indirizzo puntato da from

Applicazione echo UDP

- il server replica tutti i messaggi inviatigli dal client
  - il client legge linee di testo dallo standard input e le invia al server
  - il server legge linee di testo dal socket e li rimanda al client
  - il client legge righe di testo dal socket e li invia allo standard output



Controllo sul Mittente

- Un'applicazione in ascolto su una porta UDP accetta tutti i datagram ricevuti
  - Il client deve controllare se il datagram ricevuto è la risposta del server o proviene da un'altra applicazione
  - Tutti i datagram ricevuti che non provengono dal server devono essere scartati
- Possibile soluzione
  - Il client confronta l'indirizzo del socket da cui ha ricevuto il datagram con quello del socket a cui ha inviato la richiesta
  - Se il server ha più indirizzi IP potremmo scartare datagram legittimi

Client echo

```
int main(int argc, char **argv)  {
  int                           sockd, n, porta;
  struct sockaddr_in            servaddr;

  if (argc != 3)
     err_quit("utilizzo: echo_client <IPaddress> <porta>"); (1)
  if ( (sockd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
     err_sys("errore in socket");
  bzero(&servaddr, sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  porta = atoi(argv[2]);
  servaddr.sin_port   = htons(porta);
  if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
     err_quit("errore in inet_pton per %s", argv[1]);
  dg_cli_echo(stdin, sockd, (SA *) &servaddr,
                  sizeof(servaddr));                          (2)
  exit(0);
}
```

1. Legge da linea di comando il numero di porta del server
2. chiama la funzione dg_cli_echo
   - che svolge tutto il lavoro del client
   - la funzione prende in input l'indirizzo del server a cui spedire i datagram

Funzione dg_cli_echo – 1

```
void dg_cli_echo(FILE *fp, int sockd, const SA * p_servaddr, socklen_t
servlen) {
   int                n;
   char   sendline[MAXLINE], recvline[MAXLINE + 1];
   char   buff[MAXLINE], str[MAXLINE], port[MAXLINE];
   socklen_t       len;
   void   *ptr;
   struct sockaddr *p_replyaddr;
   struct sockaddr_in *sa;

   if( (ptr = malloc(servlen)) == NULL )                          (1)
      err_sys("errore nella malloc");
   p_replyaddr = ptr;
   while (fgets(sendline, MAXLINE, fp) != NULL) {                 (2)
```

1. alloca la memoria per contenere l'indirizzo del mittente
2. Legge fino all'EOF

Funzione dg_cli_echo – 2

```
   n = strlen(sendline);
   if( sendto(sockd, sendline, n, 0, p_servaddr, servlen) != n )      (3)
      err_sys("errore nella sendto");
   len = servlen;
   if( (n = recvfrom(sockd, recvline, MAXLINE, 0, p_replyaddr, &len))
         < 0 )                                                       (4)
      err_sys("errore nella recvfrom");
```

3. spedisce una riga di testo all'indirizzo contenuto in *p_servaddr
4. salva nel buffer il contenuto del datagram ricevuto
   - L'indirizzo del mittente è salvato in *p_replyaddr

Funzione dg_cli_echo – 3

```
   if( (len != servlen) || memcmp(p_servaddr, p_replyaddr, len) != 0 ) {      (5)
        sa = (struct sockaddr_in *) p_replyaddr;
        if( inet_ntop(AF_INET, &sa->sin_addr, buff, sizeof(buff)) == NULL )
            err_sys("errore nella inet_ntop");
        if( ntohs(sa->sin_port) != 0 ) {
          snprintf(port, sizeof(port), ".%d", ntohs(sa->sin_port));
          strcat(str, port);
        }
        printf(" messaggio da %s ignorato\n", str);
        continue;
    }
    recvline[n] = 0;
    if( fputs(recvline, stdout) == EOF )
       err_sys("errore nella fputs");
  }
}
```

5. Controlla l'indirizzo sorgente del datagram ricevuto
   - se è diverso da quello in *p_servaddr lo scarta
   - altrimenti stampa su stdout

Server echo

```
int main(int argc, char **argv)  {
   int                    sockd, porta;
   struct sockaddr_in     servaddr, cliaddr;

   if (argc != 2)
      err_quit("utilizzo: echo_server <porta>");                  (1)
   if( (sockd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 )
      err_sys("socket error");
      bzero(&servaddr, sizeof(servaddr));
      servaddr.sin_family      = AF_INET;
      servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
      porta = atoi(argv[1]);
      servaddr.sin_port        = htons(porta);
      if( bind(sockd, (SA *) &servaddr, sizeof(servaddr)) < 0 )
            err_sys("bind error");                                (2)
      dg_srv_echo(sockd, (SA *) &cliaddr, sizeof(cliaddr)); (3)
}
```

1. Legge il numero di porta da linea di comando
2. bind sulla porta
3. chiama la funzione dg_srv_echo()

Funzione dg_src_echo

```
void dg_srv_echo(int sockfd, SA *p_cliaddr, socklen_t clilen)  {
  int               n;
  socklen_t         len;
  char              mesg[MAXLINE];

  for ( ; ; ) {
     len = clilen;
     if( (n = recvfrom(sockfd, mesg, MAXLINE, 0, p_cliaddr, &len)) < 0)  (1)
          err_sys("errore nella recvfrom");
     if( sendto(sockfd, mesg, n, 0, p_cliaddr, len) != n )                (2)
          err_sys("errore nella sendto");
  }
}
```

1. memorizza in *p_cliaddr l'indirizzo del client che ha spedito il datagram
2. L'indirizzo è passato alla sendto per spedire la risposta

Indirizzi di Socket

- Lato client
  - Indirizzo IP e numero di porta sorgente definiti dal kernel
    - L'applicazione può fare la bind su una porta
  - Indirizzo IP e numero di porta destinazione passate alla sendto
- Lato server
  - Indirizzo IP e numero di porta sorgente recuperate dalla recvfrom
  - Numero di porta destinazione recuperato con getsockname()
  - Indirizzo IP destinazione recuperabile settando l'opzione IP_RECVSTADDR e usando la recvmsg

Datagrammi Perduti

- Se un datagram si perde (es. un router lo butta via) l'applicazione che lo sta attendendo può rimanere bloccata in una recvfrom()
- Soluzione
  - In alcuni è possibile porre un timeout sulla recvfrom()
    - Comunque non è possibile scoprire se il messaggio del client non è mai arrivato al server oppure se la risposta del server non è arrivata al client
  - In alcuni casi (es. transazioni bancarie) fa molta differenza
    - Utilizzare TCP

Errori Asincroni

- Una funzione provoca un errore asincrono se il segnale di errore arriva dopo il completamento della funzione
  - Es. la sendto restituisce OK perché il datagram è stato spedito. Successivamente ICMP restituisce un messaggio di errore di "host unreachable"
- Per default gli errori asincroni non sono passati ad un socket UDP
  - Lo stesso socket può essere utilizzato per inviare datagram a più destinazioni
  - ICMP restituisce l'header del datagram che ha provocato l'errore
  - Il kernel non ha modo di passare queste operazioni all'applicazione

Client UDP Connessi

- E' possibile creare un socket UPD connesso utilizzando la funzione connect()
  - Significato differente dalla connect() su socket TCP
- La connect() su un socket UDP implica che il kernel memorizza l'indirizzo IP e la porta con cui si vuole comunicare
  - Il client potrà inviare datagram solo all'indirizzo specificato dalla connect()
- Si deve distinguere tra
  - socket UDP non connessi (default)
  - socket UDP connessi

Caratteristiche di un Socket UDP Connesso

- Può inviare datagram soltanto all'indirizzo specificato nella chiamata alla connect
  - non si usa sendto ma write o send
  - i datagram verranno automaticamente spediti all'indirizzo specificato nella chiamata a connect
- Può ricevere solo datagram inviati dall'indirizzo specificato nella chiamata alla connect
  - non si usa recvfrom, ma read o readv
  - un server UDP può comunicare con un solo client per volta
- Errori asincroni possono essere controllati
  - Il kernel segnala l'errore in errno

Riutilizzo della connect

- È possibile richiamare più volte la connect() all'interno della stessa applicazione
- Può essere usato per
  - per modificare l'indirizzo del server con cui si vuole comunicare
  - Disconnettere il socket
    - specificando AF_UNSPEC nel campo sin_family
    - Potrebbe ritornare l'errore EAFNOSUPPORT, ma non è un problema

Funzione dg_cliconn_echo – 1

```
void dg_cliconn_echo(FILE *fp, int sockd, const SA * p_servaddr,
socklen_t servlen)  {
   int              n;
   char   sendline[MAXLINE], recvline[MAXLINE + 1];
   char   buff[MAXLINE], str[MAXLINE], port[MAXLINE];
   socklen_t       len;
   void   *ptr;

   if (connect(sockd, (struct sockaddr *) p_servaddr, servlen) < 0)     (1)
      err_sys("errore nella connect");
```

1. Connette il client all'indirizzo contenuto in *p_servaddr
   - Il client non può inviare datagram ad altri indirizzi

Funzione dg_cliconn_echo – 2

```
while (fgets(sendline, MAXLINE, fp) != NULL) {
      n = strlen(sendline);
      if( write(sockd, sendline, n) != n )                    (3)
          err_sys("errore nella write");
      if( (n = read(sockd, recvline, MAXLINE)) < 0 )          (4)
              err_sys("errore nella read");
      recvline[n] = 0;
      if( fputs(recvline, stdout) == EOF )
              err_sys("errore nella fputs");
    }
}
```

3. Usa la write per scrivere sul socket
4. Usa la read per leggere dal socket

Inaffidabilità di UDP

- UDP non dà alcuna garanzia sulla consegna dei datagram
- Consideriamo la seguente applicazione client/server UDP
  - Il client spedisce un serie di pacchetti, senza aspettare alcuna risposta
  - Il server cicla all'infinito e conta i datagram ricevuti
    - Termina quando riceve un segnale SIGINT (CTRL-C) da stdin
    - c'e' un gestore di segnale che gestisce SIGINT e stampa il numero di datagram ricevuti

Funzione count_client

```
int main(int argc, char **argv) {
  int                   sockd;
  struct sockaddr_in    servaddr;

  if (argc != 2)
     err_quit("utilizzo: count_client
<IPaddress>");
  /* riempie la struttura servaddr */
  if( (sockd = socket(AF_INET,
              SOCK_DGRAM, 0)) < 0 )
     err_sys("errore nella socket");
  dg_cli_count(stdin, sockd, (SA *)
              &servaddr, sizeof(servaddr));
  exit(0);
}
```

```
#define     NDG         2000
/* #datagram da inviare */
#define     DGLEN       1400
/* lunghezza di ogni datagram */

void dg_cli_count(FILE *fp, int sockd, const SA
*pservaddr, socklen_t servlen)  {
  int         i;
  char        sendline[MAXLINE];

  for (i = 0; i < NDG; i++)
       if( sendto(sockd, sendline, DGLEN, 0,
              pservaddr, servlen) != DGLEN )
          err_sys("errore nella sendto");
   printf("inviati %d datagram al server\n", NDG);
}
```


Funzione count_server

```
int main(int argc, char **argv)  {
  int           sockd;
  struct sockaddr_in    servaddr, cliaddr;

  if( (sockd = socket(AF_INET,
              SOCK_DGRAM, 0)) < 0 )
     err_sys("socket error");
  /* riempie struttura servaddr */
  if( bind(sockd, (SA *) &servaddr,
              sizeof(servaddr)) < 0 )
    err_sys("bind error");
  dg_srv_count(sockd, (SA *) &cliaddr,
              sizeof(cliaddr));

}
```

```
static void recvfrom_int(int signo)  {
  printf("\ndatagrams ricevuti: %d\n", count);
  exit(0);
}
```

```
static void  recvfrom_int(int);
static int     count;

void dg_srv_count(int sockd, struct sockaddr
              *p_cliaddr, socklen_t clilen) {
  int                   n;
  socklen_tlen;
  char                  mesg[MAXLINE];

  if( signal(SIGINT, recvfrom_int) == SIG_ERR )
     err_sys("errore nella signal");
  for ( ; ; ) {
     len = clilen;
     if( (n = recvfrom(sockd, mesg, MAXLINE, 0,
              p_cliaddr, &len)) < 0 )
        err_sys("errore nella recfrom");
     count++;
  }
}
```

Esempio

Output di netstat prima di lanciare count_client

```
home/auletta/UDP> netstat –s | less
Udp:
    294 packets received
    0 packets to unknown port received
    5706 packet receive errors
    6000 packets sent
```

```
home/auletta/UDP_COUNT> count_client
Ho inviato 2000 datagram
home/auletta/UDP_COUNT> count_server
^C
Ricevuti 120 datagram
```

Output di netstat dopo aver lanciato count_client

```
home/auletta/UDP> netstat –s | less
Udp:
    414 packets received
    0 packets to unknown port received.
    7586 packet receive errors
    8000 packets sent
```


Server UDP e TCP

- Utilizzando il modello di I/O multiplexing è possibile progettare server che operano contemporaneamente su porte TCP e UDP
  - struttura simile al server TCP iterativo che usava la select
- Utile per applicazioni che possono operare sia su TCP che UDP (es. DNS)

Server echo UDP e TCP – 1

```
void sig_chld(int);
int main(int argc, char **argv)  {
  /* dichiarazioni variabili */

  if( (listensd = socket(AF_INET, SOCK_STREAM, 0)) < 0)        (1)
      err_sys("errore in socket");
  /* riempie la struttura servaddr con l'indirizzo del socket TCP */
  setsockopt(listensd, SOL_SOCKET, SO_REUSEADDR, &on,
sizeof(on));                                                  (2)
  /* esegue la bind e la listen su listensd */
  if( (udpsd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)           (3)
    err_sys("errore in socket");
  /* riempie la struttura servaddr con l'indirizzo del socket UDP */
  if( (bind(udpsd, (SA *) &servaddr, sizeof(servaddr))) < 0)  (4)
     err_sys("errore nella bind");
  if( signal(SIGCHLD, gestisci_zombie) < 0 )                  (5)
     err_sys("errore nella signal");
```

1. Crea il socket TCP
2. Setta il socket in modo che possa utilizzare una porta occupata
3. Crea il socket UDP
4. Assegna un indirizzo al socket UDP
5. Registra il gestore di SIGCHLD

Server echo UDP e TCP – 2

```
FD_ZERO(&rset);
maxsd = MAX(listensd, udpsd) + 1;
for ( ; ; ) {
   FD_SET(listensd, &rset);                                   (6)
   FD_SET(udpsd, &rset);
   if( (ready = select(maxsd+1, &rset, NULL, NULL, NULL)) < 0 ) {   (7)
      if( errno == EINTR )
          continue;
      else
          err_sys("errore nella select");
   }
```

6. Inizializza l'insieme rset con i descrittori listensd e udpsd
7. Chiama la select

Server echo UDP e TCP – 3

```
if( FD_ISSET(listensd, &rset) ) {                                    (8)
    len = sizeof(cliaddr);
    if( (connsd = accept(listensd, (struct sockaddr *) &cliaddr, &len)) < 0)
        err_sys("errore nella accept");
    if( (childpid = fork()) == 0) {
        if( close(listensd) == -1 )
          err_sys("errore in close");
        str_srv_echo(connsd);
        if( close(connsd) == -1 )
          err_sys("errore in close");
        exit(0);
     }
     close(connsd);
  }
```

8. Se è arrivata una richiesta di connessione al socket di ascolto
   - Esegue la accept
   - forka

Server echo UDP e TCP – 4

```
    if (FD_ISSET(udpsd, &rset)) {                                    (9)
      len = sizeof(cliaddr);
      if( (n = recvfrom(udpsd, buff, MAXLINE, 0, (struct sockaddr *) &cliaddr, &len)) < 0 )
          err_sys("errore nella recvfrom");
      if( sendto(udpsd, buff, n, 0, (struct sockaddr *) &cliaddr, len) != n )
          err_sys("errore nella sendto");
    }
  }
}
```

9. Se è arrivato un datagram al socket UDP
   - Fa l'echo