UNIVERSITÀ DEGLI STUDI DI PARMA
Dipartimento di Ingegneria e Architettura
Corso di Laurea in Ingegneria informatica, elettronica e delle telecomunicazioni
a.a. 2020-2021

SISTEMI OPERATIVI

Esercitazione 6

1 Socket

Una socket fornisce una interfaccia di comunicazione tra processi che possono essere locali oppure trovarsi su nodi distinti di una rete. In generale ogni socket è identificata da un indirizzo, che nel caso specifico di socket create nel dominio di comunicazione `AF_INET` è una coppia (indirizzo IP del nodo, numero di porta).

Si suggerisce di consultare la pagina di manuale generale dedicata alle socket UNIX eseguendo il comando `man 7 socket`. Le principali primitive che consentono la creazione e la gestione delle socket sono le seguenti:

PRIMITIVE	Descrizione
socket	crea una socket di un dato dominio, tipo e protocollo
bind	assegna un nome (indirizzo) alla socket
listen	rende la socket utilizzabile per accettare richieste di connessioni entranti
accept	accetta da una socket server una richiesta di connessione da una socket client
connect	invia tramite una socket client una richiesta di connessione ad una socket server

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol)
```

L'argomento *domain* specifica il dominio di comunicazione e quindi la famiglia di protocolli (e di indirizzi) da utilizzare per la comunicazione mediante la socket (tipicamente `AF_INET`), *type* specifica il tipo della socket (ad esempio `SOCK_STREAM` per ottenere una comunicazione orientata alla connessione e affidabile). L'argomento *protocol* vale spesso zero, quando è disponibile un solo protocollo alla interno della famiglia di protocolli che mette a disposizione quel tipo di socket per il dominio prescelto. Il valore di ritorno della funzione è il *file descriptor* che identifica la socket creata. Va utilizzato sia per inviare che per ricevere messaggi attraverso la socket.

```
int bind(int sockd,struct sockaddr* my_addr_p,int addrlen)
```

L'intero *sockfd* è il descrittore restituito dalla funzione socket. L'argomento *my_addr_p* punta ad una struttura che contiene il nome (indirizzo) da assegnare alla socket. L'argomento *addrlen* specifica la dimensione di tale struttura.

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

L'argomento *backlog* specifica il massimo numero di connessioni che possono essere pendenti, ovvero ancora da accettare, nella coda di una socket.

```
int accept(int sockfd,struct sockaddr* addr_p,int* len_p)
```

L'argomento *sockfd* specifica il descrittore della socket del processo server, se il valore di *addr_p* non è `NULL`, a quell'indirizzo verrà memorizzata una struttura che contiene il nome/indirizzo della socket client che effettua la connessione. In caso di successo, il valore di uscita rappresenta il file descriptor da utilizzare nel server (o nei suoi figli) per comunicare con il cliente che si è connesso.

```
int connect(int sockfd,struct sockaddr* addr_p,int len)
```

L'argomento *sockfd* specifica il descrittore della socket del processo client, *addr_p* punta ad una struttura che contiene il nome/indirizzo della socket server alla quale il processo client intende connettersi.

Chiusura di una socket e successiva bind per lo stesso indirizzo

Dopo la chiusura di una socket di tipo `SOCK_STREAM`

```
close(sock);
```

il protocollo TCP attende un tempo tra 1 e 4 minuti (nello stato `TIME_WAIT` - cfr. l'output del comando *netstat -a*) prima di rimuoverla effettivamente, al fine di assicurarsi che eventuali pacchetti duplicati vaganti siano consegnati al destinatario. In alcuni casi questo ritardo può essere evitabile senza conseguenze, come nel caso del debug di un server che deve essere frequentemente terminato e rieseguito. A questo scopo si utilizza la `setsockopt` per forzare il riuso dell'indirizzo nel bind che quindi non fallirà anche in presenza di una socket in fase di chiusura e avente lo stesso indirizzo:

```
int on = 1;
ret = setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
. . .
bind(. . .);
```

2 Esercizi

> Tutti i file con il codice sorgente degli esercizi proposti (es∗.c) si trovano nel direttorio eserc6 della cartella con i file delle esercitazioni (ad es. ∼/so-esercitazioni).

Esercizio 1:

Esempio di scambio di messaggi tra client e server **non concorrente** mediante socket. Il client invia un messaggio al server. Il server, dopo aver ricevuto il messaggio comunica al client l'avvenuta ricezione.

Utilizzo (eseguire i due programmi ciascuno in un proprio terminale):

./server1 <numero porta>

./client1 localhost <numero porta>

Utilizzare come <numero porta> 10000+ le ultime quattro cifre del proprio numero di matricola.

Server1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>



int main(int argc, char *argv[])
{
     int sockfd, newsockfd, portno, clilen;
     char buffer[256]="";
     struct sockaddr_in serv_addr, cli_addr;
     int n,on,ret;

     if (argc < 2) {
         fprintf(stderr,"Uso %s numeroporta\n",argv[0]);
         exit(1);
     }
     sockfd = socket(AF_INET, SOCK_STREAM, 0);
     if (sockfd < 0) {
        perror("Errore socket");
exit(2);
     }


     // Per riutilizzare l'indirizzo di una socket in fase di chiusura
     on=1;
     ret = setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));


     portno = atoi(argv[1]);
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = INADDR_ANY;
     serv_addr.sin_port = htons(portno);
     if (bind(sockfd, (struct sockaddr *) &serv_addr,
              sizeof(serv_addr)) < 0) {
```

```
              perror("Errore bind");
       exit(3);
       }
     listen(sockfd,5);
     clilen = sizeof(cli_addr);
     newsockfd = accept(sockfd,
                 (struct sockaddr *) &cli_addr,
                 (socklen_t *)&clilen);
     if (newsockfd < 0) {
          perror("Errore accept");
  exit(4);
}

     n = read(newsockfd,buffer,255);
     if (n < 0) {
perror("Errore read");
exit(5);
}
     printf("(SERVER) Ecco il messaggio ricevuto dal client: %s\n",buffer);
     n = write(newsockfd,"MESSAGGIO RICEVUTO, FINE COMUNICAZIONE",38);
     if (n < 0) {
perror("Errore write");
exit(6);
}

     close(sockfd);
     close(newsockfd);

     return 0;
}
```

Client1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>

void error(char *msg)
{
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[])
```

```
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char send_buffer[256];
    char rec_buffer[256]="";

    if (argc < 3) {
       fprintf(stderr,"uso %s nomehost porta\n", argv[0]);
       exit(1);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Errore socket");
exit(2);
}
    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"ERRORE, l'host non esiste\n");
        exit(3);
    }

    serv_addr.sin_family = AF_INET;
    memcpy((char *)&serv_addr.sin_addr,(char *)server->h_addr,server->h_length);

    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0) {
        perror("Errore connect");
exit(4);
}
    printf("(CLIENT) Scrivere un messaggio: ");

    fgets(send_buffer,255,stdin);
    n = write(sockfd,send_buffer,strlen(send_buffer));
    if (n < 0) {
        perror("Errore write");
exit(5);
     }
    n = read(sockfd,rec_buffer,255);
    if (n < 0) {
        perror("Errore read");
exit(6);
}
    printf("(CLIENT) Ecco il messaggio ricevuto dal server: %s\n",rec_buffer);

    close(sockfd);
```

```
    return 0;
}
```

Esercizio 2:

Il client invia un file a un server **non concorrente**. Il server salva il file ricevuto in un file locale.
Utilizzo (invocare i due programmi in due terminali distinti):
./server1 <numero porta> <nome nuovo file>
./client1 <nome file da trasferire> localhost <numero porta>
Utilizzare come <numero porta> 10000+ le ultime quattro cifre del proprio numero di matricola.

Server2.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>


#define MAXBUF     8192

int main(int argc, char * argv[])
{
  int server_socket,connect_socket,portno;
  unsigned int client_addr_len;
  int retcode,fd;
  struct sockaddr_in client_addr, server_addr;
  char line[MAXBUF];

  if(argc!=3)
  {
    printf("Uso:\n%s port nomefilelocale\n",argv[0]);
    return(1);
  }
  printf("Server: fase di inizializzazione\n");
  server_socket = socket(AF_INET,SOCK_STREAM,0);
  if(server_socket == -1)
  {
    perror("Errore socket");
    return(2);
  }
```

```
  portno = atoi(argv[1]);
  server_addr.sin_family = AF_INET;
  server_addr.sin_addr.s_addr = INADDR_ANY;
  server_addr.sin_port   = htons(portno);

  retcode = bind(server_socket,
                 (struct sockaddr *)&server_addr,
                 sizeof(server_addr));
  if(retcode == -1)
  {
    perror("Errore bind");
    exit(3);
  }
  listen(server_socket,1);
  printf("Server: attendo connessione su porta %d\n",portno);
  client_addr_len = sizeof(client_addr);
  connect_socket = accept(server_socket,
                          (struct sockaddr *) & client_addr,
                          &client_addr_len);
   if (connect_socket < 0) {
perror("Errore accept");
exit(4);
}


  printf("Server: accettata nuova connessione\nApro file locale %s",argv[2]);
  fd = open(argv[2],O_WRONLY|O_TRUNC|O_CREAT,0644);
  if(fd == -1)
  {
    perror("Errore open");
    return(5);
  }
  do {
    retcode = read(connect_socket,line,MAXBUF);
    if(retcode != -1)
      write(fd,line,retcode);
  } while(retcode > 0);
  close(fd);
  printf("\nFine del messaggio, chiusura della connessione\n");
  close(connect_socket);

  printf("Chiusura dei lavori ... \n");
  close(server_socket);
  return(0);
}
```

Client2.c

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

```
#include <netinet/in.h>
#include <netdb.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#define MAXBUF     8192

int main(int argc, char * argv[])
{
  int client_socket,fd,portno;
  int retcode,letti;
  struct sockaddr_in  server_addr;
  char message[MAXBUF];
  char *nomehost,*filename;

  if(argc != 4)
  {
    printf("Uso:\n%s nomefile nomehost portno\n",argv[0]);
    return(1);
  }
  filename = argv[1];
  nomehost = argv[2];
  printf("Client (%d): fase di inizializzazione\n",getpid());
  client_socket = socket(AF_INET,SOCK_STREAM,0);
  if(client_socket == -1)
  {
    perror("Errore socket");
    return(2);
  }

  portno = atoi(argv[3]);
  server_addr.sin_family = AF_INET;
  server_addr.sin_port   = htons(portno);
  memcpy(&server_addr.sin_addr,(gethostbyname(nomehost)->h_addr),
 sizeof(server_addr.sin_addr));
  retcode = connect(client_socket,
 (struct sockaddr *)&server_addr,
 sizeof(server_addr));
  if(retcode == -1)
  {
    perror("Errore connect");
    return(3);
  }
  else
  printf("Client (%d): connesso al server\n",getpid());
```

```
  fd = open(filename,O_RDONLY);
  if(fd == -1)
  {
    perror("Errore open");
    return(4);
  }
  do {
    letti = read(fd,message,MAXBUF);
    if(letti > 0) { /* solo se la lettura ha avuto buon fine */
      retcode = write(client_socket,message,letti);
      if(retcode == -1)
      {
perror("Errore write");
return(5);
      }
    }
  } while (letti > 0);

 printf("Client (%d): file %s inviato\n",getpid(),filename);


  close(fd);
  close(client_socket);
  return(0);
}
```

Esercizio 3:

Si progetti in ambiente Unix/C la seguente interazione tra il processo server Ps e i suoi client Pi:

1. il processo Ps viene eseguito sul server localhost alla porta specificata come argomento di invocazione del programma;
2. i processi clienti Pi (anch'essi vengono eseguiti su localhost) inviano richieste di servizio arbitrarie (nel caso specifico il server risponde aggiungendo un valore casuale al numero inviato dal client) al server Ps attraverso socket di tipo STREAM;
3. Ps attiva un processo figlio per ogni nuova richiesta di servizio accettata (**server concorrente**) ;

Utilizzo

./server1 <numero porta>

./client1 localhost <numero porta>

Utilizzare come <numero porta> 10000+ le quattro cifre meno significative del proprio numero di matricola.

Server3.c

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


typedef struct _RICHIESTA_MSG
{
  int req;
} RICHIESTA_MSG;

typedef struct _RISPOSTA_MSG
{
  int answ;
} RISPOSTA_MSG;


main(int argc, char* argv[])
{
  int sock,length,portno;
  struct  sockaddr_in  server,client;
  int pid,s,msgsock,rval,rval2,i;
  struct hostent *hp,*gethostbyname();
  RICHIESTA_MSG request;
  RISPOSTA_MSG  answer;


  if (argc !=2) {
    printf("Usage: %s <port_number>\n", argv[0]);
    exit(1);
  }

  /* Crea la  socket STREAM */
  sock= socket(AF_INET,SOCK_STREAM,0);
  if(sock<0)
  {
    perror("opening stream socket");
    exit(2);
  }

  portno = atoi(argv[1]);

  server.sin_family =  AF_INET;

  /* Utilizzo della wildcard INADDR_ANY per accettare connessioni
```

```
   ricevute da qualunque interfaccia di rete del sistema */
server.sin_addr.s_addr= INADDR_ANY;
server.sin_port = htons(portno);
if (bind(sock,(struct sockaddr *)&server,sizeof(server))<0)
{
  perror("binding stream socket");
  exit(3);
}

length= sizeof(server);
if(getsockname(sock,(struct sockaddr *)&server,&length)<0)
{
  perror("getting socket name");
  exit(4);
}

printf("Socket port #%d\n",ntohs(server.sin_port));

/* Pronto ad accettare connessioni */

listen(sock,2);

do {
  /* Attesa di una connessione */

  msgsock= accept(sock,(struct sockaddr *)&client,(socklen_t *)&length);

  if(msgsock ==-1)
    perror("accept");
  else
  {
    printf("Connection from %s, port %d\n",
           inet_ntoa(client.sin_addr), ntohs(client.sin_port));
// SERVER CONCORRENTE
    if((pid = fork())== 0)
    {
      close(sock);
      read(msgsock,&request,sizeof(request));

      /* Esecuzione del servizio */
      answer.answ = request.req + (rand()%100);
      write(msgsock,&answer,sizeof(answer));
      close(msgsock);
      exit(0);
    }
    else
    {
      if(pid == -1) /* Errore nella fork */
```

```
        {
          perror("Error on fork");
          exit(5);
        }
        else
{
          /* OK, il padre torna in accept */
          close(msgsock);
        }
      }
    }
  }
  while(1);
  exit(0);
}
```

Client3.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


typedef struct _RICHIESTA_MSG
{
  int req;
} RICHIESTA_MSG;

typedef struct _RISPOSTA_MSG
{
  int answ;
} RISPOSTA_MSG;

main(int argc,char* argv[])
{
  int i,s,sock,rval,rval2,portno;
  struct  sockaddr_in  server;
  struct hostent *hp,*gethostbyname();
  int tsum;
  float avg;
  time_t nsec;
  unsigned short nmil;
  RICHIESTA_MSG request;
  RISPOSTA_MSG  answer;
```

```
if(argc != 3)
{
  fprintf(stderr,"Uso: %s hostname portno\n\n",argv[0]);
  exit(-1);
}

srand(getpid());

/* Crea una  socket di tipo STREAM per il dominio TCP/IP */
sock= socket(AF_INET,SOCK_STREAM,0);

if(sock<0)
{
  perror("opening stream socket");
  exit(1);
}

/* Ottiene l'indirizzo del server */
server.sin_family = AF_INET;
hp= gethostbyname(argv[1]);

if(hp==0)
{
  fprintf(stderr,"%s: unknown host",argv[1]);
  exit(2);
}

memcpy((char *)&server.sin_addr, (char *)hp->h_addr, hp->h_length);

/* La porta e' sulla linea di comando */
portno = atoi(argv[2]);
server.sin_port= htons(portno);

/* Tenta di realizzare la connessione */
printf("Connecting...\n");
if(connect(sock,(struct sockaddr *)&server,sizeof(server)) <0)
{
  perror("connecting stream socket");
  exit(1);
}

printf("Connected.\n");

request.req = rand() %100;
write(sock,&request,sizeof(request));

read(sock,&answer,sizeof(answer));
```

```
  printf("Sent %d - server answer is : %d\n",request.req,answer.answ);

  close(sock);

  exit(0);
}
```

Esercizio 4:

Si realizzi un server TCP sulla porta <numero porta> che ritorni ai clienti la data e l'ora in ASCII.
Utilizzo:
./server4 <numero porta>
Da un altro shell invocare il comando *telnet localhost <numero porta>*.
Utilizzare come <numero porta> 10000+ le quattro cifre meno significative del proprio numero di matricola.

Server4.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>


main(int argc, char *argv[])
{
  char buf[26];
  time_t ticks;
  int sock,length,msgsock,rval,portno;
  struct sockaddr_in server;
  struct hostent *hp;


  if (argc !=2) {
    printf("Usage: %s <port_number>\n", argv[0]);
    exit(-1);
  }

  printf("\n*-----------------------------------------------*\n");
  printf("*                                               *\n");
  printf("*         SERVER data e ora correnti            *\n");
  printf("*                                               *\n");
  printf("* ^c per terminare!                             *\n");
```

```
  printf("*                                                *\n");
  printf("*------------------------------------------------*\n\n");

  sock= socket(AF_INET,SOCK_STREAM,0);    /* Crea la  socket STREAM */
  if(sock<0)
  {
    perror("opening stream socket");
    exit(1);
  }

  portno = atoi(argv[1]);

  server.sin_family = AF_INET;

/* Utilizzo della wildcard INADDR_ANY per accettare connessioni
   da ogni interfaccia di rete del sistema  */
  server.sin_addr.s_addr= INADDR_ANY;

  server.sin_port = htons(portno);
  if (bind(sock,(struct sockaddr *)&server,sizeof(server))<0)
  {
    perror("binding stream socket");
    exit(1);
  }

  length= sizeof(server);
  if(getsockname(sock,(struct sockaddr *)&server,&length)<0)
  {
    perror("getting socket name");
    exit(1);
  }

  printf("Socket port #%d\n\n",ntohs(server.sin_port));

  listen(sock,2); /* Pronto ad accettare connessioni */

  do {
    /* Attesa di una connessione */
    msgsock= accept(sock,(struct sockaddr *)0,(int *)0);

    if(msgsock ==-1)
      perror("accept");
    else
    { // SERVER NON CONCORRENTE : in questo caso è appropriato,
      // dato che il servizio è elementare e il server non deve
      // attendere alcun input dal client
      ticks = time(NULL);        /*ottiene data e ora corrente*/
      strcpy(buf,ctime(&ticks)); /*la converte in stringa e la memorizza in buf[26]*/
```

```
      /* Invio dell'informazione */
      if((rval = write(msgsock,buf,sizeof(buf)))<0)
        perror("writing on stream socket");
      printf("%d byte scritti\n",rval);
    }
    close (msgsock);
  }
  while(1);

  exit(0);
}
```

Per verificare il funzionamento del server utilizzare come client il comando *telnet*, eseguendolo con *telnet localhost numeroporta*. *Esercizio proposto*:

Si realizzi un client che si colleghi al server dell'esercizio precedente e riceva e visualizzi data e ora corrente.

3 Socket Datagram

Esercizio 5:

Esempio di ping pong su socket DATAGRAM. Utilizzo ./servdgram <numero porta>
./clientdgram localhost <numero porta>
Il client misura il ritardo temporale che intercorre tra la trasmissione e la ricezione dei datagrammi. Nel caso in cui la porta specificata risulti in uso modificare il codice (suggerimento: creare il numero della porta in modo casuale)

servdgram.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/timeb.h>
#include <string.h>

#define BYTES_NR 64
#define MSG_NR 512


char buf[BYTES_NR];
```

```
main(argc,argv)
     int argc;char *argv[];
{
int sock,length;
struct  sockaddr_in  server,client;
int rval,i;
struct hostent *hp,*gethostbyname();


if(argc !=2)
  {
    fprintf(stderr,"Usage: %s port\n",argv[0]);
    exit(1);
  }

/* Create socket */
sock= socket(AF_INET,SOCK_DGRAM,0);
if(sock<0)
{
perror("opening stream socket");
exit(2);
}

/* Name socket using wildcards */
server.sin_family =  AF_INET;
server.sin_addr.s_addr= INADDR_ANY;
server.sin_port = htons(atoi(argv[1]));
if (bind(sock,(struct sockaddr *)&server,sizeof(server))<0)
{
perror("binding stream socket");
exit(3);
}

/* Find out assigned port and print out */
length= sizeof(server);
if(getsockname(sock,(struct sockaddr *)&server,&length)<0)
{
perror("getting socket name");
exit(1);
}

printf("Socket port #%d\n",ntohs(server.sin_port));


while(1)
  {
    do
{
```

```
  bzero(buf,sizeof(buf));
    if((rval = recvfrom(sock,buf,sizeof(buf), 0,
(struct sockaddr *)&client, (socklen_t *)&length ))<0)
perror("reading stream message");
  i=0;
  if(rval==0)
printf("Ending connection\n");
  else
{
    printf("Message received: sending back\n");
  strcat(buf,"*");
  if(sendto(sock,buf,sizeof(buf),0,(struct sockaddr *)&client,sizeof(client))<0)
   perror("writing on stream socket");
}
} while(rval !=0);

  }
exit(0);
}
```

clientdgram.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <unistd.h>




#define BYTES_NR 64
#define MSG_NR 512


char buf[BYTES_NR];
char buf2[BYTES_NR];

char msg[MSG_NR][BYTES_NR];
char answ[MSG_NR][BYTES_NR];

struct timeval xstime[MSG_NR];
struct timeval xftime[MSG_NR];
```

```
int main(int argc, char *argv[])
{
int i,sock,rval,length;
unsigned long delay;
struct  sockaddr_in  server,client;
struct hostent *hp,*gethostbyname();

if(argc !=3)
  {
    fprintf(stderr,"Usage: %s servername serverport\n",argv[0]);
    exit(1);
  }

for(i=0;i<MSG_NR;i++)
{
sprintf(&msg[i][0],"%d",i);
}

/* Create socket */
sock= socket(AF_INET,SOCK_DGRAM,0);
if(sock<0)
{
perror("opening stream socket");
exit(2);
}


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

if (bind(sock,(struct sockaddr *)&client,sizeof(client)) <0)
{
perror("binding error");
exit(3);
}

length= sizeof(client);

if(getsockname(sock,(struct sockaddr *)&server,&length)<0)
{
perror("getting socket name");
exit(4);
}
printf("Socket port #%d\n",ntohs(client.sin_port));

hp = gethostbyname(argv[1]);
```

```
if (hp == 0)
{
fprintf(stderr,"%s :unknow host",argv[1]);
exit(5);
}

bcopy( (char *)hp ->h_addr, (char *)&server.sin_addr,hp ->h_length);
server.sin_family = AF_INET;
server.sin_port = htons(atoi(argv[2]));

for(i=0;i<MSG_NR;i++)
{
strcpy(buf,msg[i]);

gettimeofday(&xstime[i],NULL);
if(sendto(sock,buf,sizeof(buf),0,(struct sockaddr *)&server,sizeof(server))<0)
perror("sendto problem");

if((rval = read(sock,buf2,sizeof(buf2)))<0)
perror("reading stream message");

strcpy(answ[i],buf2);

gettimeofday(&xftime[i],NULL);
}
close(sock);

for(i=0;i<MSG_NR;i++)
{
delay= (xftime[i].tv_sec-xstime[i].tv_sec)*1000000. +
(xftime[i].tv_usec-xstime[i].tv_usec);
printf("msg %d [%s]: %0.3f ms\n",i,answ[i],delay/1000.);
}

exit(0);
}
```

4 Server concorrente di gestione di un contatore

Esercizio 6:
Si realizzi un **server concorrente** che gestisca un contatore incrementato secondo i valori interi inviati dai client.

Si tratta di una tipica situazione in cui la gestione di una risorsa condivisa, in assenza di memoria condivisa (**i nostri processi UNIX non condividono memoria, dato che servirebbero system call che non abbiamo visto a lezione**), deve essere realizzata nel modello a scambio di messaggi mediante un processo gestore oppure più semplicemente operando su un messaggio contenuto in una pipe. Il server seguente è una soluzione basata

su una pipe per ospitare il valore del contatatore. Per verificare il funzionamento del server utilizzare come client il comando *telnet*, eseguendolo con *telnet localhost numeroporta*.

server6.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>



int main(int argc, char* argv[])
{
  int sock,length,portno;
  struct  sockaddr_in  server,client;
  int pid,s,msgsock,rval,rval2,i;
  struct hostent *hp,*gethostbyname();
  int pipecont[2],counter_value,value;
  char buffer[256],answer[256];
  char *hellomsg = "Immettere un incremento intero per il contatore condiviso\n";

  if (argc !=2) {
    printf("Usage: %s <port_number>\n", argv[0]);
    exit(-1);
  }

  /* Crea la  socket STREAM */
  sock= socket(AF_INET,SOCK_STREAM,0);
  if(sock<0)
  {
    perror("opening stream socket");
    exit(1);
  }

  portno = atoi(argv[1]);

  /* Utilizzo della wildcard per accettare le connessioni ricevute da ogni interfaccia di r
  server.sin_family =  AF_INET;
  server.sin_addr.s_addr= INADDR_ANY;
  server.sin_port = htons(portno);
  if (bind(sock,(struct sockaddr *)&server,sizeof server)<0)
  {
    perror("binding stream socket");
    exit(1);
  }
```

```
length= sizeof(server);
if(getsockname(sock,(struct sockaddr *)&server,&length)<0)
{
  perror("getting socket name");
  exit(1);
}

printf("Socket port #%d\n",ntohs(server.sin_port));

/* Creazione di una pipe per ospitare il messaggio con il valore corrente del contatore *
if (pipe(pipecont)<0) {
      perror("creating pipe");
      exit(2);
  }
counter_value = 0;
write(pipecont[1],&counter_value,sizeof(counter_value));


/* Pronto ad accettare connessioni */

listen(sock,2);

do {
  /* Attesa di una connessione */

  msgsock= accept(sock,(struct sockaddr *)&client,(socklen_t *)&length);

  if(msgsock ==-1)
    perror("accept");
  else
  {
    printf("Connessione da %s, porta %d\n",
           inet_ntoa(client.sin_addr), ntohs(client.sin_port));


    if((pid = fork())== 0)
    {
      write(msgsock,hellomsg,strlen(hellomsg)+1);
      read(msgsock,&buffer,sizeof(buffer));

      sscanf(buffer,"%d",&value);

      /* Uso della pipe per contenere il valore corrente del contatore */
      read(pipecont[0],&counter_value,sizeof(counter_value));
      counter_value  += value;
      write(pipecont[1],&counter_value,sizeof(counter_value));
```

```
        sprintf(answer,"Il contatore ora vale %d\n",counter_value);

        write(msgsock,answer,strlen(answer)+1);
        close(msgsock);
        exit(0);
      }
      else
      {
        if(pid == -1) /* Errore nella fork */
        {
          perror("Error on fork");
          exit(3);
        }
        else
{
          /* OK, il padre torna in accept */
          close(msgsock);
        }
      }
    }

  }
  while(1);
  exit(0);
}
```