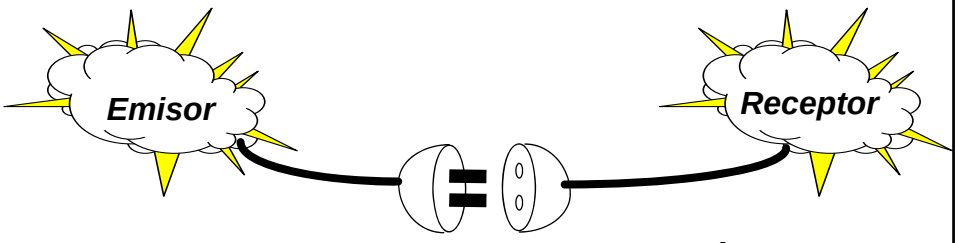
Los sockets de Unix



Funcionamiento y programación

Dr. Roberto Gómez Cárdenas
DCC del ITESM-CEM
rogomez@itesm.mx
http://webdia.cem.itesm.mx/ac/rogomez

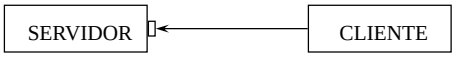
La comunicación

- Comunicación ocurre a través de un puerto
- Cliente y servidor comparten sistema de archivos, y están en misma máquina:

  puerto = memoria compartida, pipes, fifos.

- por otro lado en una red:

  puerto = socket o conexión TLI



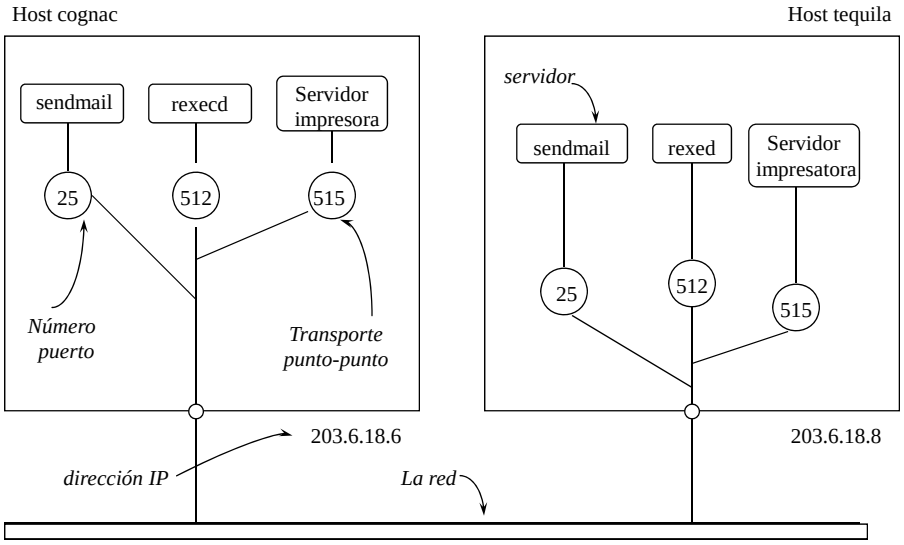
El concepto de puerto

- Cliente debe conocer la máquina a contactar:
  nivel alto: máquina identificada por un nombre
  nivel bajo: dirección red (IP: Internet Protocol)

- Dentro máquina existen varios “puntos finales de comunicación”, en los cuales los servidores están escuchando para una conexión.
  Puntos finales = número puerto

- Analogía:
  IP = Número teléfono; Número Puerto = extensión

Identificando un servidor



Ejemplo de Servicios y Números Puertos

daytime	13/ tcp	
daytime	13/ udp	
netstat	15/ tcp	
chargen	19/ tcp	ttytst source
chargen	19/ udp	ttytst source
ftp-data	20/ tcp	
ftp	21/ tcp	
telnet	23/ tcp	
smtp	25/ tcp	mail
time	37/ tcp	timserver
time	37/udp	timserver
. . .		
exec	512/ tcp	
login	513/ tcp	
shell	514/ tcp	cmd
printer	515/ tcp	spooler

¿Qué son los sockets?

- Punto de comunicación por el cual un proceso puede emitir o recibir información
- Es una interfaz con la entre capa de aplicación y el de transporte.
- En el interior de un proceso se identificará por un descriptor, parecido al usado para la identificación de archivos:
  - permite re-dirección de la entrada y salida estándar
  - permite utilización de aplicaciones estándar sobre la red
  - todo nuevo proceso, (*fork()*) hereda los descriptores de socket de su padre
- Permite, dado dos procesos que se comunican a través de ellos, despreocuparse de:
  - canal físico de comunicación, (capa física de la ISO-OSI)
  - forma de codificación de señales para disminuir probabilidad error en la transmisión (capa enlace)
  - nodos red por los cuales tiene que pasar, (capa red)
  - formar paquetes de la información a transmitir y buscar ruta que una la computadora origen con la destino, (capa transporte)

Los pasos en la comunicación

Servidor:
abre su puerto
espera por peticiones del cliente

Cliente:
abre su puerto
escribe su petición

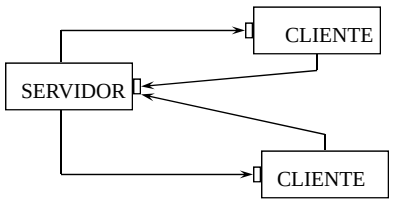
Servidor:
realiza el servicio

Excelente mientras solo exista un cliente y el cliente no requiera de un reply

Comunicación un servidor varios clientes

Si existe más de un cliente

Peticiones clientes diferentes deben diferenciarse
Establecer convención para enviar id del proceso cliente
Cuidado con la seguridad: posibilidad un proceso se haga pasar por otro.



Solución en sockets

Llamadas sistema *recvfrom( ) , listen()* y *accept()* permiten al servidor escuchar un ***socket*** conocido para verificar si hay peticiones

Cada petición identifica al emisor

Servidor usa la identificación para enviar respuesta usando *sendto( )*

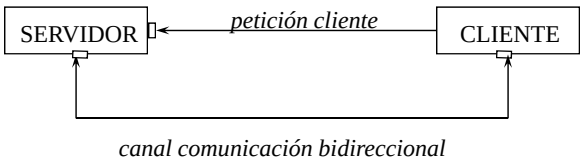
recvfrom( ) y *sendto( )* son la base del protocolo conectionless de sockets

Canal de comunicación bidireccional

Si cliente y servidor requieren de interacción adicional durante procesamiento de la petición:

útil contar con canal de comunicación de doble sentido

canal privado que no requiere intercambio de id de los procesos en cada envío de mensajes



canal comunicación bidireccional

Canal privado de comunicación

Canal:

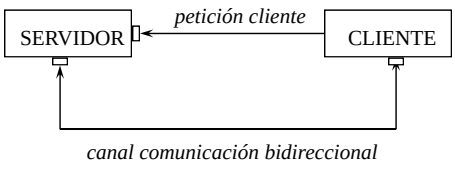
Ningún otro proceso puede aceptarlo
Protocolos orientados conexión:
mecanismo llamado *hand off*

Una vez que el canal fue establecido servidor debe decidir como manejar la petición

1. Estrategia servidor-serial (serial server)
2. Estrategia servidor-padre (father server)
3. Estrategia servidor-thread (thread server)

Estrategia servidor serial

Cuando servidor recibe una petición se dedica completamente a atender la petición antes que cualquier otra

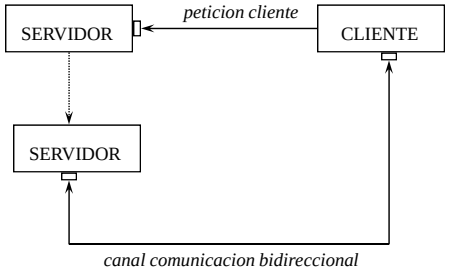


Pseudocódigo estrategia servidor serial

```
for ( ; ;  ) {
        escuchar petición cliente
        crear canal comunicación privado bidireccional
        while  (no error en canal de comunicación ) {
                leer  petición  cliente
                atender petición
                responder al cliente
        }
        cerrar canal de comunicación
}
```


Estrategia servidor padre

Servidor "forks" un hijo para que atienda la petición, mientras que el servidor se queda escuchando otras posibles peticiones

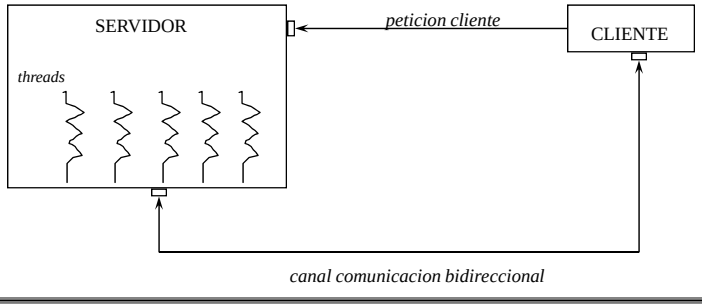


Pseudocódigo estrategia servidor padre

```
for ( ; ; ) {
        escuchar petición cliente
        crear canal comunicación privado bidireccional
        fork un hijo que atienda la petición
        cerrar el canal de comunicación
        limpiar zombies
}
```


Estrategia servidor thread

Alternativa de la estrategia anterior de bajo overhead
En lugar de hacer un fork de un hijo para atender la petición, el servidor crea un thread en su propio espacio de proceso
Ventaja: menos overhead y tratamiento más eficiente
Desventaja: posible interferencia entre peticiones multiples debido al espacio de direcciones compartido.



Transfiriendo información entre máquinas heterogéneas

Para comunicar dos computadoras no basta con poder conectarlas lógicamente también se tienen que poner de acuerdo en diferentes aspectos.

Entre los más importantes encontramos:

- Orden representación de bytes
- Operaciones sobre bytes
- Obtención nombre máquina y otros datos de la misma
- Representación de direcciones

Rutinas de ordenamiento de bytes

Funciones que manejan posibles diferencias en el orden de la representación de bytes entre diferentes arquitecturas de computadoras y diferentes protocolos.

```
#include <sys/types.h>
#include <netinet/in.h>
```

u_long htonl(u_long hostlong);
conversión entero largo de host a red

u_short htons(u_short hostshort);
conversión entero corto de host a red

u_long ntohl(u_long netlong);
conversión entero largo de red a host

u_short ntohs(u_short netshort);
conversión entero corto de red a host

Operaciones sobre bytes

Sistema 4.3BSD define funciones que funcionan sobre *user-defined byte strings*.
user-defined = no son strings de caracteres standard de C (que terminan en caracter nulo); pueden contener *null-bytes* dentro de ellos y *no significan* un fin de string; por esto se debe especificar el tamaño de cada string como parámetro.

bcopy(char *src, char *dest, int nbytes);
mueve el número de bytes especificado de *src* a *dest*
(difere del orden usado por la función strcpy())

bzero(char *dest, int nbytes);
escribe el número especificado de *null-bytes* en *dest*

int bcmp(char *ptr1, char *ptr2, int nbytes);
compara dos *bytes strings*
regresa cero si los dos byte strings son identicos, sino regresa un valor no-cero
(difiere del resultado aportado por la función strcmp())

NOTA: equivalente en sistema V: memcpy(), memset() y memcmp().

Rutinas conversión direcciones

Dirección Internet puede ser escrita en forma numérica: *148.241.61.25* o en forma de una cadena de caracteres: *puertorico.cem.itesm.mx*, pero es almacenada en una variable de tipo entero

Funciones permiten convertir de un formato u otro a un número entero

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

unsigned long inet_addr(char *ptr);
convierte una notación de string de caracteres en una notación de decimal de 32 bits.

char *inet_ntoa(struct in_addr inaddr);
convierte una dirección decimal en una de string de caracteres.

Obtención nombre máquina

Cuatro formas obtener nombre:

1. Comando *uname:*
   $***uname -n***
   puertorico
   $
2. Comando *hostname*
   $***hostname***
   puertorico
   $
3. Archivo */etc/hostname.le0*
   $more /etc/hostname.le0
   puertorico
   $

`comandos`

4. Llamada sistema *gethostname(name, nlen)*
   *char *name;*
   int nlen;

Ejemplo programa

```
$ cat maquina.c
main()
{
  char host[30];

  if ( gethostname(host,30) < 0 )
    strcpy(host,“desconocido”);

  printf(“Proceso %d “, getpid() );
  printf(“ejecutandose en %s \n”, host);

}
$ gcc maquina.c –o maquina
$ maquina
Proceso 7891 ejecutandose en cognac
$
```

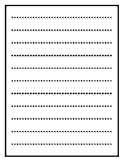

Obtención datos de otra máquina: archivo /ect/hosts

- Archivo de configuración de red TCP/IP
- Contiene la lista de sitios en la red local
- Debe contener al menos dos entradas: dirección de ciclo y la dirección con la que el sitio sera conocido en la red.
- Ejemplos archivos:

```
rogomez@brasil:9>more /etc/hosts
127.0.0.1       localhost
148.241.61.15   brasil
148.241.32.40   mailhost
```

```
rogomez@paises:9>more /etc/hosts
# loopbak address
127.1             loopbak
#  red  europea
192.1.3.1         italia
192.1.3.2         inglaterra  uk england
192.1.3.3         francia
192.1.3.252       portugal   #gateway para la LAN 4
# red america
192.1.4.1         nicaragua
192.1.4.2         venezuela
192.1.4.3         brasil
192.1.4.26        mexico   #gateway para la LAN 3
```

Obtención datos de otra máquina: los resolvers



gethostbyname(const char* host)
gethostbyaddr(const char *_addr_, int _len_, int _type_)
gethostent(void)

Archivo: */etc/hosts*

estructura host

```
struct hostent {
    char    *h_name;      The official name of the host.
    char   **h_aliases;   A zero-terminated array of alternative names for the host.
    int      h_addrtype;  The type of address; always AF_INET at present
    int      h_length;    The length of the address in bytes.
    char   **h_addr_list  A zero-terminated array of network addresses
                          for the host in network byte order
}
#define h_addr  h_addr_list[0] The first address in h_addr_list for backward compatibility.
```



```
 if ( (sd_actual = accept(sd, (struct sockaddr *)&pin, &addrlen)) == -1) {
            perror("accept");
            exit(1);
 }
dirlen = sizeof(pin.sin_addr.s_addr);     /* casi siempre tiene un valor de 4 */
host_clt = gethostbyaddr((void *)&pin.sin_addr.s_addr,  dirlen ,  AF_INET);
if (host_clt == NULL)
      nombre_clt = "desconocido";
else
      nombre_clt = host_clt->h_name;
printf("El mensaje [%d] fue recibido de: %s \n",pet, nombre_clt);
printf("con direccion:  %d \n",(int)&claddr.sin_addr.s_addr);
```

Ejemplo uso gethostbyname (1/2)

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>

struct hostent *he;
struct in_addr a;

int main (int argc, char **argv)
{
    if (argc != 2) {
        fprintf(stderr, "uso: %s hostname\n", argv[0]);
        return 1;
    }
    he = gethostbyname (argv[1]);
    if (he) {
     printf("Nombre Oficial (h__name): %s\n", he->h_name);
     printf("Tipo direccion (h_addrtype): %d \n", he->h_addrtype);
     printf("Longitud direccion en bytes (h_length): %d \n",
                                                 he->h_length);
```


Ejemplo uso gethostbyname (2/2)

```
while (*he->h_aliases)
            printf("Alias (h_aliases): %s\n", *he->h_aliases++);
      while (*he->h_addr_list)
      {
         bcopy(*he->h_addr_list++, (char *) &a, sizeof(a));
         printf("Direcciones (h_addr_list): %s\n", inet_ntoa(a));
      }
    }
    else
        herror(argv[0]);
    return 0;
}
```

Corrida:

```
$ quien-es whdh
quien-es: Unknown host
$ quien-es webdia
Nombre Oficial (h__name): webcem01.cem.itesm.mx
Tipo direccion (h_addrtype): 2
Longitud direccion en bytes (h_length): 4
Alias (h_aliases): webdia.cem.itesm.mx
Direcciones (h_addr_list): 10.48.6.164
$
```

Usando sockets para transmisión datos

+Interfaz capa transporte no esta totalmente aislada de capas inferiores.

al trabajar con sockets es necesario conocer detalles sobre estas capas

+Es necesario conocer:

1. La *familia* o *dominio* de la conexión
   Familia agrupa todos aquellos sockets con características comunes, (protocolos, formatos direcciones, convenios para los nombres, etc)

2. *Tipo* de conexión.
   Tipo de circuito que se va a establecer entre los dos procesos.

   2.1 *Circuito virtual*: orientado conexión

   2.2 *Datagrama:* orientado no conexión

Familias direcciones sockets

1. Unix domain addres

```
struct sockaddr_un {
   short     sun_family;
   char      sun_path[108];
}
```

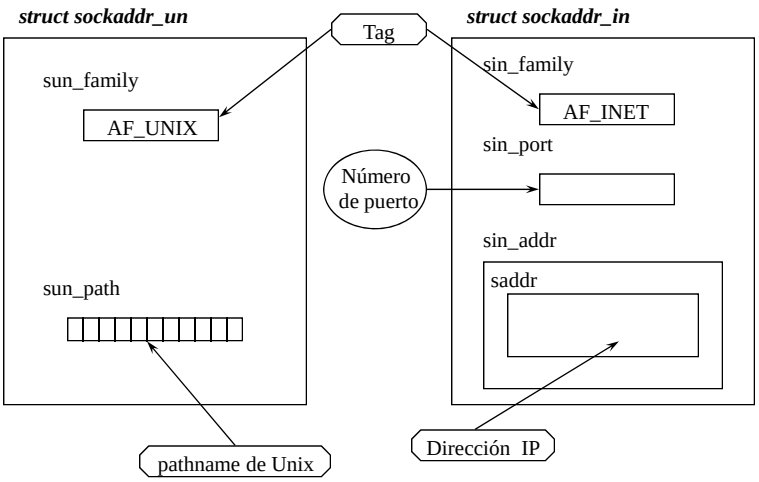
2. Internet domain addres

```
struct sockaddr_in {
   short             sin_family;
   u_short           sin_port;
   struct in_addr    sin_addr;
   char              sin_zero;
}

struct in_addr {
   u_long    s_addr;
}
```

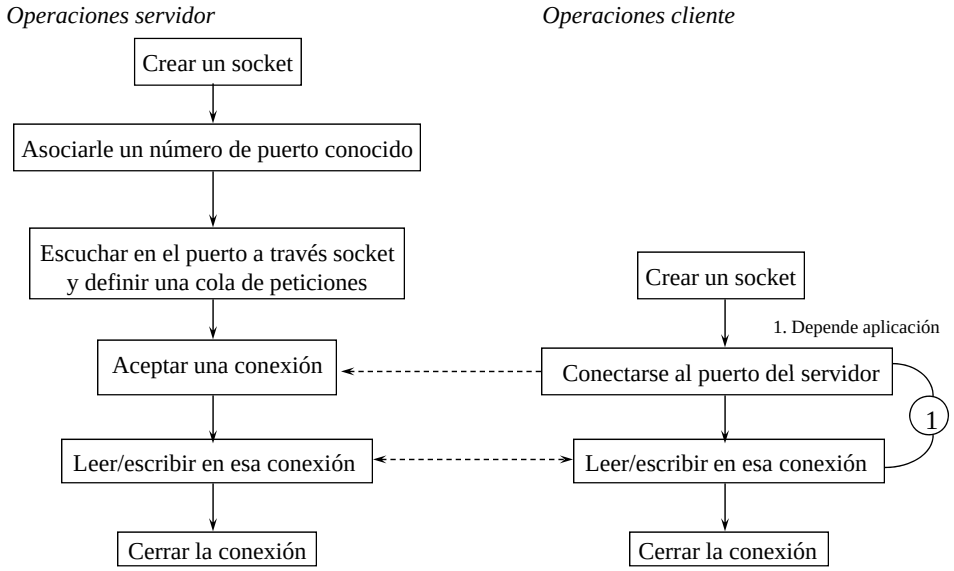
3. Generic socket addres

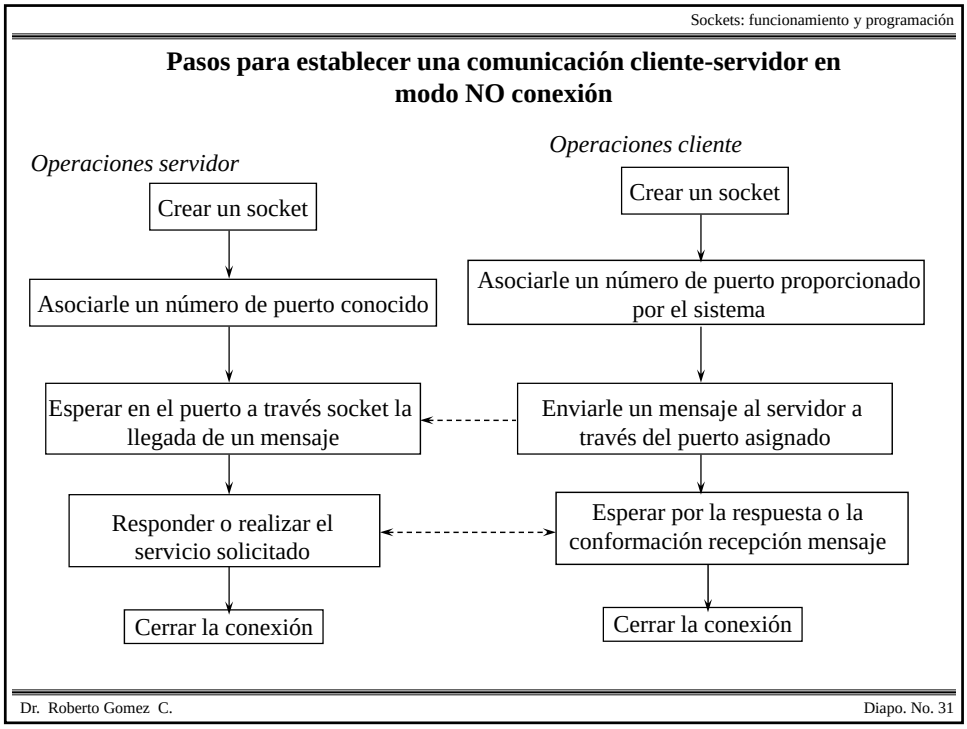
```
struct sockaddr{
   u_short    sa_family;
   char       sa_data[14];
}
```

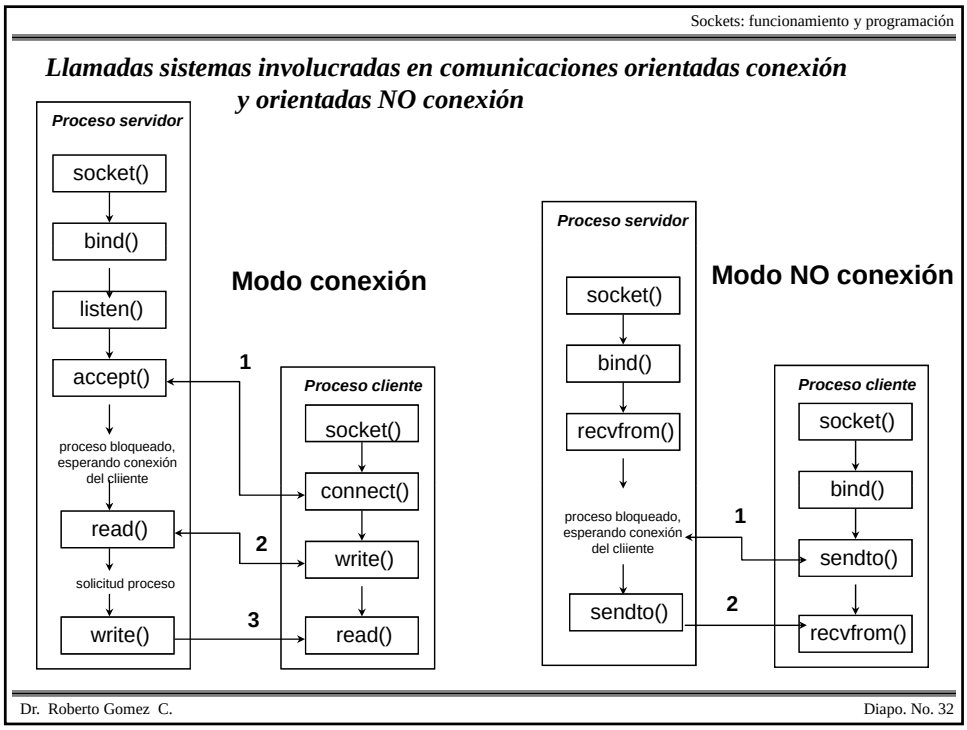
Comparando familias



Pasos para establecer una comunicación cliente-servidor en modo conexión




Sockets: funcionamiento y programación
Pasos para establecer una comunicación cliente-servidor en modo NO conexión
Operaciones servidor
Crear un socket
Asociarle un número de puerto conocido
Esperar en el puerto a través socket la llegada de un mensaje
Responder o realizar el servicio solicitado
Cerrar la conexión
Operaciones cliente
Crear un socket
Asociarle un número de puerto proporcionado por el sistema
Enviarle un mensaje al servidor a través del puerto asignado
Esperar por la respuesta o la conformación recepción mensaje
Cerrar la conexión
Dr. Roberto Gomez C.
Diapo. No. 31


Sockets: funcionamiento y programación
Llamadas sistemas involucradas en comunicaciones orientadas conexión y orientadas NO conexión
Proceso servidor
socket()
bind()
listen()
accept()
proceso bloqueado, esperando conexión del cliiente
read()
solicitud proceso
write()
Modo conexión
1
2
3
Proceso cliente
socket()
connect()
write()
read()
Proceso servidor
socket()
bind()
recvfrom()
proceso bloqueado, esperando conexión del cliiente
sendto()
Modo NO conexión
1
2
Proceso cliente
socket()
bind()
sendto()
recvfrom()
Dr. Roberto Gomez C.
Diapo. No. 32

Creación socket: *socket()*

int socket(familia, tipo, protocolo)

Regresa un valor entero
Parecido descriptor de archivos: descriptor socket *sockfd*

int familia
familia de direcciones:
AF_UNIX protocolos internos UNIX
AF_INET protocolo de internet
AF_NS protocolo de la XEROX NS
AF_IMPLIK IMP link layer (interface Multiple Layer)

int protocolo
tipo de protocolo a utilizar
O: el sistema elige su protocolo.

int tipo
tipo de socket:

SOCKET_STREAM	socket tipo stream
SOCKET_DGRAM	socket tipo datagrama
SOCKET_RAW	socket tipo raw
SOCKET_SEQPAKET	socket paquete secuencial
SOCKET_RDM	realible delivered message socket (no implementado)

Protocolos para diversos tipos y familias

	AF_UNIX	***AF_INET***	***AF_NS***
SOCK_STREAM	yes	TCP	SPP
SOCK_DGRAM	yes	UDP	IDP
SOCK_RAW		IP	yes
SOCK_SEQPACKET			SPP

yes: validos pero sin handy acronyms.
vacías: no implementadas

Ejemplo:

```
        :
int sock;
        :
sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock < 0) {
    fprintf(stderr, “Error en la creacion del socket \n”);
    exit(1);
}
```

Asociando un puerto al socket: *bind()*

int bind(sockfd, myaddr, addrlen)

Asocia una dirección a un socket:
Cualquier mensaje que llega a esa dirección tiene que dárselo al proceso servidor

int sockfd
descriptor del socket (obtenido a partir de ***socket( )***)

struct sockaddr *myaddr
apuntador a la dirección de un socket

int addrlen
tamaño de la dirección myaddr

Ejemplo uso *bind()*

```
#define  NUM_PUERTO          12345

int sock;
struct sockaddr_in   server;
                :
sock = socket(……);
                :
server.sin_family           = AF_INET;
server.sin_addr.s_addr      = htonl(INADDR_ANY);
server.sin_port             = htons(NUM_PUERTO);

if  ( bind(sock, (struct sockaddr*) &server, sizeof(server)) < 0) {
        fprintf(stderr,"Error en el binding del socket \n");
        exit(1);
}
```

Escuchando el puerto: *listen()*

int listen(sockfd, backbg)

Usada por el servidor en una conexión para indicar que desea recibir conexiones.
Ejecutada después de una llamada *socket()* y *bind()*, y antes de una llamada *accept()*

int sockfd
descriptor del socket

int backbg
especifíca cuantas demandas de conexión puede formar el sistema antes de atender una
generalmente este parámetro se deja con un valor de 5.

Ejemplo:

```
int sock;
        :
sock = socket(……);
        :
bind(sock, ……..);
        :
listen(sock, 5);
```


Aceptando una conexión: *accept()*

int accept(sockfd, peer, addrlen)

- Acepta la primera demanda de conexión en la cola, y crea otro socket con las mismas propiedades que *sockfd.*
- Regresa el identificador del descriptor creado
- Si no hay ninguna demanda esta llamada se bloquea hasta que llegue una.

int sockfd
descriptor del socket

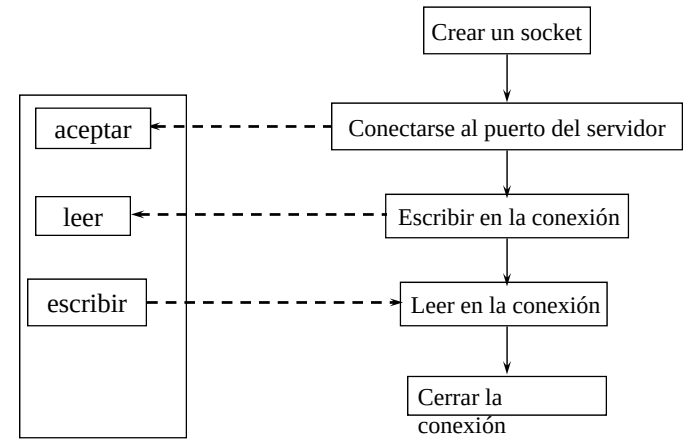
struct sockaddr *peer
regresar la dirección del proceso peer (*husmeador* -cliente, proceso con el que se hizo la conexión: comunicación *peer-to-peer*)

int *addrlen
- argumento de resultado: el cliente que llama deja un valor antes de llamar a *accept()*, y la llamada de sitema guarda un resultado en esa variable.
- generalmente el cliente lo utiliza para dar a conocer el tamaño de un buffer, y la llamada regresa la cantidad de información almacenada en el buffer
- en este caso se almacena el numero de bytes almacenados en *peer*.

Ejemplo uso llamada *accept()*

```
int sock, tam_cliente;
struct sockaddr_in cliente;
                :
    sock = socket(……);
                :
    bind(sock, ……..);
                :
    listen(sock, 5);
    while (1) {
        tam_cliente = sizeof(cliente)
        if ( (fd = accept(sock, (struct sockaddr *) &client, &tam_client)) < 0) {
                fprintf(stderr,"Error en la aceptacion de la conexión \n");
                exit(1);
        }
         else {

                        <Tratamiento de la conexión>
        }
    }
```


Operaciones cliente para establecer una conexión



Conectandose al puerto: *connect()*

int connect(sockfd, servaddr, addrlen)

Usada por el cliente para conectarse a un descriptor de socket
Se utiliza después de realizar una llamada socket()

int sockfd
descriptor del socket

struct sockaddr *servaddr
apuntador a la dirección del socket distante (servidor)

int addrlen
tamaño de la dirección.

Ejemplo uso *connect()* y otras funciones

```
#define SERV_HOST_ADDR          "192.43.235.6"
#define SERV_TCP_PORT           6543

int                     sockfd;
struct sockaddr_in      servidor;

bzero((char *) &servidor, sizeof(servidor));
servidor.sin_family             = AF_INET;
servidor.sin_addr.s_addr        = inet_addr(SERV_HOST_ADDR);
servidor.sin_port               = htons(SERV_TCP_PORT);

sockfd = socket(AF…….)

if ( connect(sockfd,  (struct sockaddr *) &servidor, sizeof(servidor)) <  0) {
        fprintf(stderr, "Error: cliente no pudo establecer una conexión \n");
        exit(1);
}
```

Emitiendo/recibiendo información: *write()* y *read()*

int write(int fd, char *buff, unsigned int nbytes);

Para enviar información a través de un socket basta con escribir la información a enviar en el descriptor que regresa la función ***socket()***, o la función ***accept()***.

```
#define SIZEBUF  1024
char *info;
sprintf(info," Probando 1,2,3….");

sock = socket(…….…..);          newfd = accept(…….……..);
write(sock, info, sizeof(info));  write(newfd, info, SIZEBUF);
```

int read(int fd, char *buffer, unsigned int nbytes);

Para recibir información basta con leer la información del descriptor obtenido de la llamada ***socket()*** o ***accept()***

```
char *mensaje;
read(sock, mensaje, sizeof(mensaje));   read(newfd, mensaje, SIZEBUF);
```


Otras rutinas de comunicación

Estas llamadas de sistema son similares a las llamadas *read()* y *write()*, pero requieren de otros argumentos:

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

int send(int sockfd, char *buff, int nbytes, int flags);

int sendto(int sockfd, char *buff, int nbytes, int flags, struct sockaddr *to, int addrlen);

int recv(int sockfd, char *buff, int nbytes, int flags);

int recvfrom(int sockfd, char *buff, int nbytes, int flags, struct sockaddr *from, int *addrlen);

Los tres primeros argumentos: *sockfd, buff* y *nbytes* son similares a los tres primeros de *read()* y *write()*

Las parámetros de las funciones anteriores

El parámetro **flag** puede ser cero, o un or de las siguientes constantes:

MSG_OOB	enviar o recibir datos out-of-band
MSG_PEEK	permite al que cliente, (el que llamó), ver la información que esta disponible para lee, sin descartar los datos despúés de que las llamadas ***recv()*** o ***recvfrom()*** regresaron.
MSG_DONTROUTE	ruteo bypass, (***send()*** o ***sendto()***)

Argumento **to** de ***sendto()*** indica la dirección especifica de donde se enviar los datos.

Llamada ***recvfrom()*** llena en la dirección especifica **from** la información recibida.

Todos regresan el tamaño de los datos que se escribieron o leyeron.

Ejemplos funciones

```
#define MAXLINE   512
int sockfd, n;
struct sockadrr  *pserv_addr;          /* ptr a la estructura sockaddrr_xx apropiada */
int     servlen;                       /* tamaño actual de pserv_addr   */
char sendline[MAXLINE], recvline[MAXLINE +1];

while (fgets(sendline, MAXLINE, fp) != NULL)  {
        n = strlen(sendline);
        if (sendto (sockfd, sendline, n, 0, pserv_addr, servlen) != n) {
                fprintf(stderr,"Error el enviar datos en el socket \n");
                exit(1);
        }
        /* lectura de un mensaje del socket y escritura en la salida standard */
        n = recvfrom(sokfd, recvline, MAXLINE, 0, (struct sockaddr *) 0,  (int *)  0);
        if (n < 0) {
                fprintf(stderr,"Error el enviar datos en el socket \n");
                exit(1);
        };
        recvline[n] =0;  fputs(recvline, stdout);
                        :
```

Cerrando la conexión: *close()*

int close(sockfd)

Cierra el socket
Parecido al cierre de un archivo

int sockfd
descriptor del socket.

Ejemplo:

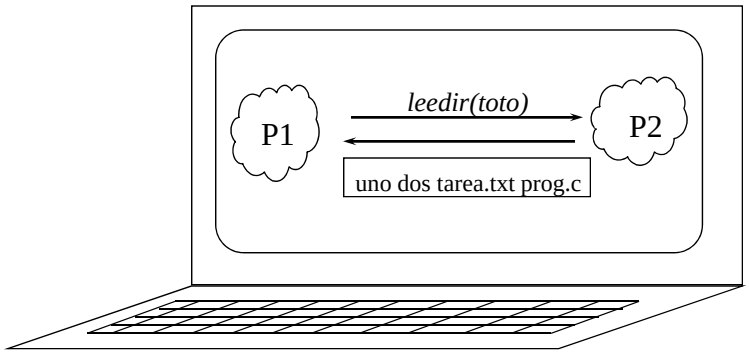
```
int sock
        :
sock = socket(……..);
        :
        :
close(sock);
```


Ejemplo del uso de sockets

Tres tipos:

1. Unix domain addressing
2. Internet domain addressing with TCP
3. Internet domain addressing with UDP

Ejemplo uso sockets familia Unix: *lectura de un directorio*



Los dos procesos en la misma máquina

Lectura local de un directorio
archivo: lee_dir.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <errno.h>

lee_dir(dir)
    char *dir;
{
    DIR *dirp;
    struct dirent *direntp;

    /* abriendo el directorio */
    dirp = opendir(dir);
    if (dirp == NULL)
        return((int)NULL);

    /* copiando los nombres archivos en el buffer dir */
     dir[0] = '\0'
     while ( (direntp= readdir(dirp)) != NULL)
         sprintf(dir, "%s%s\n", dir, direntp->d_name);

    /* regresando el resultado */
      closedir(dirp);
      return((int)dir);
}
```

Versión orientada conexión con sockets tipo Unix
Código del Servidor

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

#define  DIRSIZE   8192          /* tamaño máximo parámetro entrada/salida*/

main()
{
        char  dir[DIRSIZE];      /* parámetro entrada y salida */
        int     sd,   sd_actual; /* descriptores de sockets */
        struct sockaddr_un   sin;  /*  direcciones socket */
```



```
        /* obtención de un socket tipo AF_UNIX   */
                if ( (sd = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
                        perror("socket");
                        exit(1);
                }

        /* llenando los campos de la estructura de dirección unix   */
                strcpy(sin.sun_path, "./socket");
                sin.sun_family = AF_UNIX;

        /* asociando el socket al archivo ./socket */
                if ( bind(sd, (struc sockaddr *)&sin, sizeof(sin)) == -1 ) {
                        perror("bind");
                        exit(1);
                }

        /* ponerse a escuchar a través del socket */
                if ( listen(sd, 5) == -1 ) {
                        perror("listen");
                        exit(1);
                }
```

```
/* esperando que un cliente solicite un servicio */
        if ( (sd_actual = accept(sd, 0, 0)) == -1) {
                perror("accept");
                exit(1);
        }

/* tomar un mensaje del cliente */
        if ( read(sd_actual, dir, sizeof(dir) ) == -1) {
                perror("read");
                exit(1);
        }

/* realizando servicio solicitado: leyendo el directorio */
        lee_dir(dir);
```



```
/* enviando la respuesta del servicio */
        if ( write(sd_actual, dir, sizeof(dir) ) == -1) {
                perror("write");
                exit(1);
        }

/* cerrar los dos sockets */
        close(sd_actual);   close(sd);

/* no olvidar realizar un poco de limpieza */
        unlink("./socket");

}
```

Versión orientada conexión con sockets tipo Unix
Código del Cliente

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

#define  DIRSIZE   8192              /* tamaño máximo parámetro entrada/salida */

main(argc,  argv)
int argc; char *argv[];
{
        char  dir[DIRSIZE];          /* paramétro entrada y salida */
        int     sd;                  /* descriptor del socket */
        struct sockaddr_un   sin;    /*  direcciones socket */
```



```
/* verificando número parámetros entrada */
        if (argc != 2) {
                fprintf(stderr,"Error, uso: %s <directorio> \n",argv[0]);
                exit(1);
        }

/* obtención de un socket tipo AF_UNIX  (igual que el servidor) */
        if ( (sd = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
                perror("socket");
                exit(1);
        }

/* llenando los campos de la estructura de dirección unix, mismos datos servidor */
        strcpy(sin.sun_path, "./socket");
        sin.sun_family = AF_UNIX;

/* conectandose  al archivo ./socket */
        if ( connect(sd, (struct sockaddr *)&sin, sizeof(sin)) == -1) {
                perror("connect");
                exit(1);
        }
```

```
/* enviar mensaje al servidor */
        strcpy(dir, argv[1]);
        if ( write(sd, dir, sizeof(dir)) == -1) {
                perror("write");
                exit(1);
        }

/* esperar por la respuesta */
        if ( read(sd, dir, sizeof(dir) ) == -1) {
                perror("read");
                exit(1);
        }

/* imprimir los resultados y cerrando la conexion del socket */
        printf("%s \n", dir);
        close(sd);
}
```


Compilando y ejecutando

```
rogomez@armagnac:1>cc servicio.c lee_dir.c -lnsl -lsocket -o servicio
rogomez@armagnac:2>cc cliente.c -lnsl -lsocket  -o cliente
rogomez@armagnac:3>servicio &
[1]    2606
rogomez@armagnac:4>cliente .
.
..
socket
servicio
lectura_dir
cliente.c
cliente
servicio.c
[1]    Exit  1                 servicio
rogomez@armagnac:5>ls -l
.
..
servicio
lectura_dir
cliente.c
cliente
servicio.c
rogomez@armagnac:6>
```

Versión "resumida" del servidor

```
1   main()
2   {
3           sd = socket(AF_UNIX, SOCK_STREAM, 0);
4           strcpy(sin.sun_path, "./socket");
5           sin.sun_family = AF_UNIX;
6           bind(sd, (struc sockaddr *)&sin, sizeof(sin));
7           listen(sd, 5);
8           sd_actual = accept(sd, 0, 0);
9           read(sd_actual, dir, sizeof(dir) );
10          lee_dir(dir);
11          write(sd_actual, dir, sizeof(dir));
12          close(sd_actual);
13          close(sd);
14          unlink("./socket");
15  }
```


Versión servidor serial

```
1   main()
2   {
3           sd = socket(AF_UNIX, SOCK_STREAM, 0);
4           strcpy(sin.sun_path, "./socket");
5           sin.sun_family = AF_UNIX;
6           bind(sd, (struc sockaddr *)&sin, sizeof(sin));
7           listen(sd, 5);
8           while (1) {
9                   sd_actual = accept(sd, 0, 0);
10                  read(sd_actual, dir, sizeof(dir) );
11                  lee_dir(dir);
12                  write(sd_actual, dir, sizeof(dir));
13                  close(sd_actual);
14          }
15          close(sd);
16          unlink("./socket");
17  }
```

Versión servidor padre con regreso de valores numéricos

```
main()
{
    sd = socket(AF_UNIX, SOCK_STREAM, 0);
    strcpy(sin.sun_path, "./socket");
    sin.sun_family = AF_UNIX;
    bind(sd, (struc sockaddr *)&sin, sizeof(sin));
    listen(sd, 5);
    while (1) {
            sd_actual = accept(sd, 0, 0);
            if ( fork() == 0 ) {
                    atencion(sd_actual);
                    exit(0);
            }
    }
   close(sd);
   unlink("./socket");
}
```

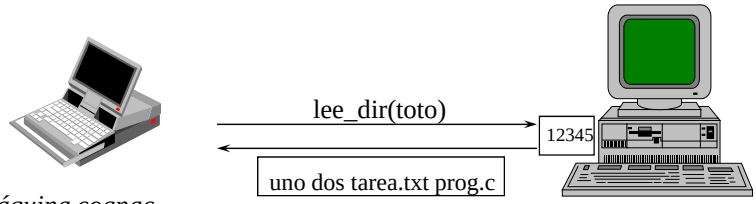
```
atencion(int sock)
{
    int pid;
    char res[8], pet[20];

    read(sock, pet, sizeof(pet) );
    if  (pet[0] == 'a')
       pid = getpid();
    else
       pid = getppid();
    sprintf(res,"%d", pid);
    write(sock, res, sizeof(res));
    close(sock);
}
```


Función de atención que regresa una estructura de datos

```
struct datos {
  int padre;
  int id;
  float numero;
  char letra;
};
```

```
void atencion(int sockfd)
{
  struct datos info;
  char mensaje[20];

  read(sockfd, &mensaje, sizeof(mensaje));

  info.id = getpid();
  info.padre = getppid();
  info.letra = mensaje[0];
  info.numero = srand();

  write(sockfd, &info, sizeof(info));
  close(sockfd);
}
```

Ejemplo uso sockets familia internet en modo conexión lectura de un directorio remoto



máquina cognac

máquina tequila

```
$ cliente  tequila toto
.
..
uno
dos
tarea.txt
prog.c
$
```

```
$ servidor
&
$
```


Versión orientada conexión usando TCP
Código del Servidor

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

#define  DIRSIZE   8192          /* tamaño máximo parámetro entrada/salida */
#define  PUERTO   12345           /* numero puerto arbitrario */

main()
{
        char  dir[DIRSIZE];                 /* paramétro entrada y salida */
        int     sd,   sd_actual;            /* descriptores de sockets */
        int     addrlen;                    /* tamaño de la dirección */
        struct sockaddr_in   sin, pin;      /*  direcciones socket cliente y servidor */
```

```
/* obtención de un socket tipo internet */
        if ( (sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                perror("socket");
                exit(1);
        }
/* asignar direcciones en la estructura de direcciones */
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = INADDR_ANY;      /* INADDR_ANY = 0x00000000 = yo mismo*/
        sin.sin_port = htons(PUERTO);           /* convirtiendo a formato red*/

/* asociando del socket al número de puerto */
        if ( bind(sd, (struct sockaddr *)&sin, sizeof(sin)) == -1) {
                perror("bind");
                exit(1);
        }

/* ponerse a escuchar a través del socket */
        if (listen(sd, 5) == -1) {
                perror("listen");
                exit(1);
        }
```



```
/* esperando que un cliente solicite un servicio */
        if ( (sd_actual= accept(sd, (struct sockaddr *)&pin, &addrlen)) == -1) {
                perror("accept");
                exit(1);
        }
/* tomar un mensaje del cliente */
        if (recv(sd_actual, dir, sizeof(dir), 0) == -1) {
                perror("recv");
                exit(1);
        }
/* leyendo el directorio */
        lee_dir(dir);

/* enviando la respuesta del servicio */
        if (send(sd_actual, dir, sizeof(dir), 0) == -1) {
                perror("send");
                exit(1);
        }
/* cerrar los dos sockets */
        close(sd_actual);   close(sd);
}
```

Versión orientada conexión usando sockets tipoTCP
Código del Cliente

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

#define  DIRSIZE   8192              /* tamaño máximo parámetro entrada y salida */
#define  PUERTO   12345               /* numero puerto arbitrario */

main(argc, argv)
  int argc;  char *argv[];
{
        char  dir[DIRSIZE];                 /* paramétro entrada y salida */
        int     sd,                         /* descriptor socket */
        struct hostent          *hp;        /* estructura del host */
        struct sockaddr_in   sin, pin;      /*  direcciones socket cliente, servidor */
        char  *host;                        /* nombre del host */
```



```
        /* verificando número parámetros entrada */
                if (argc != 3) {
                        fprintf(stderr,"Error, uso: %s <host> <directorio> \n",argv[0]);
                        exit(1);
                }

        /* encontrando todo lo referente acerca de la máquina host */
                host = argv[1];
                if ( (hp = gethostbyname(host)) == 0) {
                        perror("gethosbyname");
                        exit(1);
                }
        /* llenar la estructura de direcciones con la información del host */
                pin.sin_family = AF_INET;
                pin.sin_addr.s_addr = ((struct in_addr *) (hp->h_addr))->s_addr);
                pin.sin_port = htons(PUERTO);

        /* obtención de un socket tipo internet */
                if ( (sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                        perror("socket");
                        exit(1);
                }
```

```
/* conectandose al PUERTO en el HOST  */
        if (connect(sd, (struct sockaddr *)&pin, sizeof(pin)) == -1) {
                perror("connect");
                exit(1);
        }

/* enviar mensaje al PUERTO del  servidor en la máquina HOST */
        strcpy(dir, argv[2]);
        if (send(sd, dir, sizeof(dir), 0) == -1) {
                perror("send");
                exit(1);
        }

/* esperar por la respuesta */
        if (recv(sd, dir, sizeof(dir), 0) == -1) {
                perror("recv");
                exit(1);
        }

/* imprimir los resultados y cerrando la conexion del socket *
        printf("%s \n", dir);   close(sd);
}
```


Compilando y ejecutando

Máquina servidora: cognac

```
rogomez@cognac:6>cc servicio.c lee_dir.c  -lnsl -lsocket -o servicio
rogomez@cognac:7>cc cliente.c -lnsl -lsocket  -o cliente
rogomez@cognac:8>servicio &
[1]    2606
rogomez@cognac:9>
```

Máquina cliente: tequila

```
rogomez@tequila:27>cliente cognac toto
.
..
servicio
lectura_dir
cliente.c
cliente
servicio.c
rogomez@tequila:28>
```

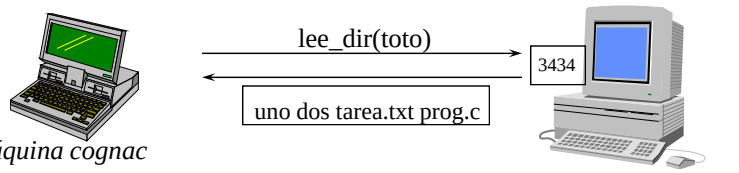
Versión “resumida” servidor

```
#define  DIRSIZE   8192
#define  PUERTO   12345
main()
{
        sd = socket(AF_INET, SOCK_STREAM, 0);
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = INADDR_ANY;
        sin.sin_port = htons(PUERTO
        bind(sd, (struct sockaddr *)&sin, sizeof(sin))
        listen(sd, 5)
        sd_actual= accept(sd, (struct sockaddr *)&pin, &addrlen))
        recv(sd_actual, dir, sizeof(dir), 0
        lee_dir(dir);
        send(sd_actual, dir, sizeof(dir), 0)
        close(sd_actual);
        close(sd);
}
```


Versión “resumida” cliente

```
#define  DIRSIZE   8192
#define  PUERTO   12345

main(int argc, char * argv[ ])
{
        host = argv[1];
        hp = gethostbyname(host);
        pin.sin_family = AF_INET;
        pin.sin_addr.s_addr = ((struct in_addr *) (hp->h_addr))->s_addr);
        pin.sin_port = htons(PUERTO);
        sd = socket(AF_INET, SOCK_STREAM, 0);
        connect(sd, (struct sockaddr *)&pin, sizeof(pin));
        strcpy(dir, argv[2]);
        send(sd, dir, sizeof(dir), 0);
        recv(sd, dir, sizeof(dir), 0);
        printf(“%s \n”, dir);
        close(sd);
}
```

Ejemplo uso sockets familia internet en modo NO conexión lectura de un directorio remoto



máquina cognac

máquina tequila

```
$ cliente  tequila 3434  toto
.
..
uno
dos
tarea.txt
prog.c
$
```

```
$ servidor 3434 &
$
```


Version orientada no-conexión usando sockets tipo UDP

Código servidor (vers. simple)

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

#define  DIRSIZE        8192        /* tamaño parámetro entrada/salida */

int  puerto;                        /* número de puerto */
int sd;                             /* descriptor del socket */
int addrlen;                        /* tamaño dirección */
int cc;                             /* regreso llamada recvfrom() */
struct sockaddr_in   myaddr;        /* mi dirección internet, servidor*/
struct sockaddr_in   claddr;        /*  dirección internet del cliente */
char  dir[DIRSIZE];                 /* paramétro entrada y salida */
```

```
main (argc, argv)
   int argc;
   char *argv[];
{
        /* verificando número parámetros entrada */
        if (argc != 2) {
                fprintf(stderr,"Error, uso: %s <puerto> \n",argv[0]);
                exit(1);
        }
        puerto = atoi(argv[1]);

        /* llenar la estructura de direcciones con la información del host */
        myaddr.sin_family = AF_INET;
        myaddr.sin_addr.s_addr = INADDR_ANY;
        myaddr.sin.port = puerto;

        /* obtención de un socket tipo internet */
        if ( (sd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
                perror("socket");
                exit(1);
        }
```



```
        /* asociando del socket al número de puerto */
        if (bind(sd, (struct sockaddr *)&myaddr, sizeof( struct sockaddr_in)) == -1) {
                perror("bind");
                exit(1);
        }
        addrlen = sizeof(struct sockaddr_in);
        /* tomar el mensaje cuando este es enviado */
        if ( (cc = recvfrom(sd, dir, sizeof(dir), 0, (struct sockaddr *)&claddr, &addrlen))=-1);
                perror("recv");
                exit(1);
        }
        /* realizar el servivio */
        lee_dir(dir);
        /* enviar la respuesta al cliente */
        addrlen=sizeof(claddr);
        if ( sendto(sd, dir, sizeof(dir), 0, (struct sockaddr *)&claddr, addrlen) == -1) {
                perror("sendto");
                exit(1);
        }
        /* conecctionless - no es necesario el accept() no hay socket extra que cerrar */
        close(sd);
}
```

Versión orientada no-conexión con sockets tipo UDP

Código cliente (versión simple)

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

#define  DIRSIZE            8192            /* tamaño parámetro entrada/salida */

main(argc, argv)
  int argc;  char *argv[];
{
        char            dir[DIRSIZE];   /* paramétro entrada y salida */
        char            *host;          /* nombre del host */
        int             puerto;         /* número de puerto */
        int             sd;             /* descriptor del socket */
        int             addrlen;        /* tamaño de la dirección */
        struct hostent  *hp;            /*  descriptor del host*/
```



```
        struct sockaddr_in  myaddr;     /*  mi dirección internet (cliente) */
        struct sockaddr_in  svaddr;     /*  dirección internet del emisor */
        struct sockaddr_in  claddr;     /*  dirección internet de la respuesta, (servidor)*/

        /* verificando número parámetros entrada */
        if (argc != 4) {
                fprintf(stderr,"Error, uso: %s <host> <puerto> <directorio> \n",host);
                exit(1);
        }

        /¨* obteniendo datos del servidor */
        host = argv[1];
        puerto=atoi(argv[2]);
        if  ( (hp = gethostbyname(host)) == NULL) {
                perror("gethostbyname");
                exit(1);
        }

        /* llenando la información del servidor */
        svaddr.sin_family = AF_INET;
        svaddr.sin_addr.s_addr = ((struct in_addr*) (hp->h_addr))->s->addr;
        svaddr.sin_port = htons(puerto);
```

```
/* obtención de un socket tipo internet */
if ( (sd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("socket");
        exit(1);
}

/* añadiendo informacion sobre el cliente */
myaddr.sin_family = AF_INET;
myaddr.sin_port = 0;
myaddr.sin_addr.s_addr = INADDR_ANY;

/* asociando el socket al puerto  */
if (bind(sd,  (struct sockaddr  *)&myaddr, sizeof( struct sockaddr_in)) == -1) {
        perror("bind");
        exit(1);
}
/* se envía la petición del servicio */
strcpy(dir, argv[3]); addrlen = sizeof(svaddr);
if (sendto(sd, dir, sizeof(dir), 0, (struct sockaddr *)&svaddr, addrlen) == -1) {
        perror("sendto");
        exit(1);
}
```



```
/* recepción de la respuesta del servicio demandado */
        if  (recvfrom(sd, dir, sizeof(dir), 0, (struct sockaddr *)&claddr, &addrlen) == -1) {
                perror("recvfrom");
                exit(1);
        }

        printf("%s  \n", dir);
        close(sd);
}
```

Compilando y ejecutando

Máquina servidora: cognac

```
rogomez@cognac:6>cc servicio.c lee_dir.c  -lnsl -lsocket -o servicio
rogomez@cognac:7>cc cliente.c -lnsl -lsocket  -o cliente
rogomez@cognac:8>servicio 3434 &
[1]    2606
rogomez@cognac:9>
```

Máquina cliente: tequila

```
rogomez@tequila:27>cliente cognac 3434 toto
.
..
servicio
lectura_dir
cliente.c
cliente
servicio.c
rogomez@tequila:28>
```


Versión “resumida” servidor

```
#define  DIRSIZE 8192
main(int argc, char * argv[])
{
        puerto = atoi(argv[1]);
        myaddr.sin_family = AF_INET;
        myaddr.sin_addr.s_addr = INADDR_ANY;
        myaddr.sin.port = puerto;
        sd = socket(AF_INET, SOCK_DGRAM, 0);
        bind(sd, (struct sockaddr *)&myaddr, sizeof( struct sockaddr_in));
        addrlen = sizeof(struct sockaddr_in);
        recvfrom(sd, dir, sizeof(dir), 0, (struct sockaddr *)&claddr, &addrlen);
        lee_dir(dir);
        addrlen=sizeof(claddr);
        sendto(sd, dir, sizeof(dir), 0, (struct sockaddr *)&claddr, addrlen);
        close(sd);
}
```

Versión “resumida” cliente

```
#define  DIRSIZE 8192
main(int argc, char * argv[])
{
        host = argv[1];
        puerto=atoi(argv[2]);
        hp = gethostbyname(host);
        svaddr.sin_family = AF_INET;
        svaddr.sin_addr.s_addr = ((struct in_addr*) (hp->h_addr))->s->addr;
        svaddr.sin_port = htons(puerto);
        socket(AF_INET, SOCK_DGRAM, 0);
        myaddr.sin_family = AF_INET;
        myaddr.sin_port = 0;
        myaddr.sin_addr.s_addr = INADDR_ANY;
        bind(sd,  (struct sockaddr  *)&myaddr, sizeof( struct sockaddr_in);
        strcpy(dir, argv[3]);   addrlen = sizeof(svaddr);
        sendto(sd, dir, sizeof(dir), 0, (struct sockaddr *)&svaddr, addrlen)
        recvfrom(sd, dir, sizeof(dir), 0, (struct sockaddr *)&claddr, &addrlen)
        printf("%s  \n", dir);
        close(sd);
}
```


Mejorando el sistema

- Busqueda automática del servicio
  - El archivo /etc/hosts
  - El archivo /etc/services
  - Obtención información de hosts y servicios
- Manejo perdida de mensajes

El archivo /etc/hosts

- Archivo de configuración de red TCP/IP
- Contiene la lista de sitios en la red local
- Debe contener al menos dos entradas: dirección de ciclo y la dirección con la que el sitio sera conocido en la red.
- Ejemplos archivos:

```
rogomez@brasil:9>more /etc/hosts
127.0.0.1       localhost
148.241.61.15   brasil
148.241.32.40   mailhost
```

```
rogomez@paises:9>more /etc/hosts
# loopbak address
127.1             loopbak
#  red  europea
192.1.3.1         italia
192.1.3.2         inglaterra  uk england
192.1.3.3         francia
192.1.3.252       portugal   #gateway para la LAN 4
# red america
192.1.4.1         nicaragua
192.1.4.2         venezuela
192.1.4.3         brasil
192.1.4.26        mexico   #gateway para la LAN 3
```


El archivo /etc/services

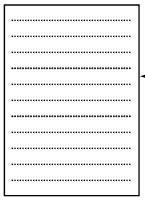
Continene la lista de servicios y los puertos reservados asociados
Cada línea esta formada del nombre del servicio, puerto asociado y protocolo
Ejemplo archivo */etc/services*

```
# Networks services, Internet style
tcpmux    1/tcp
echo      7/tcp
ftp       21/tcp
telnet    23/tcp
smtp      25/tcp    mail
time      37/tcp    timeserver
time      37/udp    timeserver

# Host specific functions

tftp      69/udp
rje       77/tcp
finger    79/tcp
link      87/tcp    ttylink
```

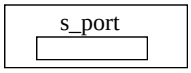
```
# Unix specific services
# these are not officially assigned
exec      512/tcp
login     513/tcp
shell     514/tcp   cmd       # no passwd used
printer   515/tcp   spooler   # line printer spooler
courier   530/tcp   rpc       # experimental
uucp      540/tcp   uucpd     # uucp deamon
biff      512/udp   comsat
who       513/udp   whod
talk      517/udp
kerberos  750/udp   kdc       # Kerberos by server
kerberos  750/tcp   kdc       # Kerberos by server
lockd     4045/udp            # NFS lock daemon
lockd     4045/tcp
fs        7100/tcp            # Font server
```

Obteniendo información servicio (puerto, protocolo, etc.)

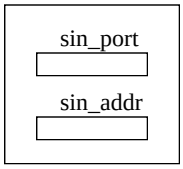
Archivo: */etc/services*



getservbyname(serv, proto)
getservbyport(puerto)
getservent()



estructura socket



estructura servidor

```
struct servent {
   char     *s_name;
   char    **s_aliases;
   int       s_port;
   char    **s_proto;
}
```

```
struct sockaddr_in {
    short   sin_family;
    u_short   sin_port
   struct  in_addr   sin_addr;
   char   sin_zero[8];
}
```


Ejemplo uso getservbyname()

```
#include <errno.h>
#include <string.h>
#include <netdb.h>

int main(int argc, char *argv[])
{
  struct  servent        *sp;

  if (argc != 3) {
    printf("Error, uso: %s servicio protocolo \n",argv[0]);  exit(1);
  }
  if ((sp=getservbyname(argv[1],argv[2])) == NULL) {
     printf("No hay servicio asociado con %s y protocolo %s \n",argv[1],argv[2]);   exit(1);
  }

  printf("Valores asociados con %s y %s:\n",argv[1],argv[2]);
  printf(" Nombre del servicio: %s \n",sp->s_name);
  while (*sp->s_aliases)
     printf(" Alias servicio: %s\n", *sp->s_aliases++);
  printf(" Puerto servicio: %d \n",ntohs(sp->s_port));
  printf(" Protocolo servicio: %s \n",sp->s_proto);
  return 0;
}
```

Contenido archivo /etc/services de los ejemplos

```
$ cat /etc/services
tcpmux          1/tcp
tcpmux          1/udp
        :
ftp-data        20/tcp
ftp-data        20/udp
ftp             21/tcp
ftp             21/udp          fsp fspd
ssh             22/tcp
ssh             22/udp
telnet          23/tcp
telnet          23/udp
smtp            25/tcp          mail
smtp            25/udp          mail
time            37/tcp          timserver
time            37/udp          timserver
        :
fido            60179/tcp
fido            60179/udp
toto            62123/udp  uno dos
```


Ejecución ejemplo getservbyname( )

```
$ svname ftp tcp
Valores asociados con ftp y tcp:
 Nombre del servicio: ftp
 Puerto servicio: 21
 Protocolo servicio: tcp
$ svname toto udp
Valores asociados con toto y udp:
 Nombre del servicio: toto
 Alias servicio: uno
 Alias servicio: dos
 Puerto servicio: 62123
 Protocolo servicio: udp
$ svname unservicio tcp
No hay servicio asociado con unservicio y protocolo tcp
$
```

Ejemplo uso getservbyport()

```
#include <errno.h>
#include <string.h>
#include <netdb.h>

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

   struct  servent   *sp;      int puerto;
   if (argc != 3) {
     printf("Error, uso: %s puerto protocolo \n",argv[0]); exit(1); }
   puerto = htons(atoi(argv[1]));
   if ((sp=getservbyport(puerto,argv[2])) == NULL) {
      printf("No hay servicio asociado con puerto %s y protocolo %s \n",argv[1],argv[2]);  exit(1); }

   printf("Valores asociados con %s y %s:\n",argv[1],argv[2]);
   printf(" Nombre del servicio: %s \n",sp->s_name);
   while (*sp->s_aliases)
      printf(" Alias servicio: %s\n", *sp->s_aliases++);
   printf(" Puerto servicio: %d \n",ntohs(sp->s_port));
   printf(" Protocolo servicio: %s \n",sp->s_proto);
   return 0;
}
```


Ejecución ejemplo getservbyport( )

```
$ svport 21 tcp
Valores asociados con 21 y tcp:
 Nombre del servicio: ftp
 Puerto servicio: 21
 Protocolo servicio: tcp
$ svport 62123 udp
Valores asociados con 62123 y udp:
 Nombre del servicio: toto
 Alias servicio: uno
 Alias servicio: dos
 Puerto servicio: 62123
 Protocolo servicio: udp
$ svport 36761 udp
No hay servicio asociado con puerto 36761 y protocolo udp
$
```

Manejo perdida mensajes

- Llamada de sistema: *signal(señal, manejador)*

  signal(SIGALRM, haz_algo);

- Llamada de sistema: *alarm(tiempo)*

  alarm(5);

- La sentencia *goto*

  etiqueta <código cualquiera>
  :
  :
  if (pasa algo)
  goto *etiqueta*

- Implementando un time-out o timer:

```
signal(SIGALARM, nada);
again:  ejecuta_algo();
        alarm(5);
        ejecuta_otra_cosa()
        if (llego_alarma)
                goto again
```


Esquema general sistema mejorado

máquina cognac

$ cliente toto
.
..
uno
dos
tarea.txt
prog.c
$

lee_dir(toto)

uno dos tarea.txt prog.c

/etc/services

lee_direc *xxyy*

máquina tequila

$ servidor &
$

Versión orientada no conexión con sockets tipo UDP

Código del Servidor (versión mejorada)

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

 /* se debe añadir este servicio al archivo  /etc/services      */
#define  UDP_SERVICE                “lee_direc”
#define  DIRSIZE                    8192   /* máximo tamaño parámetro ent/sal */

main()
{
        char                dir[DIRSIZE];      /* paramétro entrada y salida */
        int                 sd;                /* descriptores de sockets */
        int                 addrlen;           /* tamaño dirección */
        int                 cc;                /* valor regreso recvfrom() */
        struct sockaddr_in  myaddr;            /* mi dirección internet (servidor)*/
        struct sockaddr_in  claddr;            /*  dirección internet del cliente */
        struct servent      *sp;               /* estructura datos del servicio */
```



```
/* llenar la estructura de direcciones con la información del host */
        myaddr.sin_family = AF_INET;
        myaddr.sin_addr.s_addr = INADDR_ANY;
/* buscando en /etc/services por el servicio  */
        sp = getservbyname(UDP_SERVICE, “udp”);
        if (sp == NULL) {
                printf(“Imposible encontrar %s en /etc/services \n”, UDP_SERVICE);
                exit(1);
        }
        myaddr.sin.port = sp->sin_port;
/* obtención de un socket tipo internet */
        if ( (sd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
                perror(“socket”);
                exit(1);
        }
/* asociando del socket al número de puerto */
        if (bind(sd, (struct sockaddr *)&myaddr, sizeof( struct sockaddr_in)) == -1) {
                perror(“bind”);
                exit(1);
        }
```

```
/* tomar el mensaje cuando este es enviado */
        addrlen = sizeof(struct sockaddr_in);
        cc = recvfrom(sd, dir, sizeof(dir), 0, (struct sockaddr *)&claddr, &addrlen);
        if ( cc == -1) {
                perror("recv");
                exit(1);
        }

/* realizar el servivio */
        lee_dir(dir);

/* enviar la respuesta al cliente */
        addrlen = sizeof(claddr);
        if (sendto(sd, dir, sizeof(dir), 0, (struct sockaddr *)&claddr, adrrlen) == -1) {
                perror("sendto");
                exit(1);
        }

/* conecctionless - no es necesario el accept() no hay socket extra que cerrar */
        close(sd);
}
```


Versión orientada no-conexión con sockets UDP

Código del Cliente (versión mejorada)

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <errno.h>
#include <signal.h>
#include <netinet/in.h>
#include <netdb.h>

#define  UDP_SERVICE  "lee_dir"   /* servicios añadidos al archivo  /etc/services del servidor */
#define  HOST         "tequila"   /* nombre de la máquina servidora */
#define  DIRSIZE      8192        /* tamaño máximo del parámetro de entrada/salida */


/*  Función de desplegado de espera del cliente */
void  t_nop() {
        fprintf(stderr, "esperando por un servicio....");
}
```

```
main(argc, argv)
  int argc;  char *argv[];
{
        char                dir[DIRSIZE];     /* parámetro entrada y salida */
        int                 sd;               /* descriptor del socket */
        int                 retry;            /* número intentos petición servicio */
        int                 addrlen;          /* tamaño dirección */
        struct hostent      *hp;              /*  descriptor del host remoto */
        struct servent      *sp;              /*  descriptor del ervicio */
        struct sockaddr_in  myaddr;           /*  mi direccion (cliente) */
        struct sockaddr_in  svaddr;           /* direccion internet emisor */
        struct sockaddr_in  claddr;           /*  dirección internet respuesta (servidor) */

/* verificando  parámetro de entrada, (solo uno HOST y puertos ya predefinidos) */
        if (argc != 2) {
                fprintf(stderr,"Error, uso: %s <directorio> \n");
                exit(1);
        }
```



```
/* llenar la estructura de direcciones con la información del host */
        svaddr.sin_family = AF_INET;

        /* obteniendo información sobre el host servidor */
        hp = gethostbyname(HOST);
        if (hp == NULL) {
                fprintf(stderr,"No se encontro %s en /etc/hosts \n",HOST);
                exit(1);
        }
        svaddr.sin_addr.s_addr = ((struct in_addr *)(hp->h_addr))->s_addr;

        /* obteniendo número de puerto del servicio */
        sp = getservbyname(UDP_SERVICE, "udp");
        if (sp == NULL) {
                fprintf(stderr, "No se encontro %s en /etc/hosts \n",UDP_SERVICE);
                exit(1);
        }
        svaddr.sin_port = sp->s_port;
```

```
/* obtención de un socket tipo internet */
        if ( (sd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
                perror("socket");
                exit(1);
        }

/* añadiendo informaci{on sobre el cliente */
        myaddr.sin_family = AF_INET;
        myaddr.sin_port = 0;
        myaddr.sin_addr.s_addr = INADDR_ANY;

/* asociado el socket al puerto */
        if (bind(sd,  (struct sockaddr  *)&myaddr, sizeof( struct sockaddr_in)) == -1) {
                perror("bind");
                exit(1);
        }
```



```
/* intenta enviar un mensaje al servidor */
        signal(SIGALRM, t_nop);      retry = 5;     addrlen = sizeof(struct sockaddr_in);
again:  if (sendto(sd, argv[1], sizeof(dir), 0, (struct sockaddr *)&svaddr, addrlen) == -1) {
                perror("sendto");
                exit(1);
        }
        alarm(5);                       /* señal SIGALRM es repetida cada 5 segundos */
        if  (recvfrom(sd, dir, sizeof(dir), 0, (struct sockaddr *)&claddr, &addrlen) == -1) {
                if (errno == EINTR) {
                        if  (--retry)
                                goto again;
                        else{
                                perror("servidor no responde");
                                exit(1);
                        }
                }
                else {
                      perror("recvfrom");
                      exit(1);
                }
        }
```

```
        alarm(0);       /*resetear la alarma */
    /* en caso de éxito: se imprimen resultados y se cierra todo */
        printf("%s  \n", dir);
        close(sd);
    }
```


Compilando y ejecutando

Máquina servidora: cognac

```
rogomez@cognac:6>cc servicio.c lee_dir.c  -lnsl -lsocket -o servicio
rogomez@cognac:7>cc cliente.c -lnsl -lsocket  -o cliente
rogomez@cognac:8>servicio &
[1]    2606
rogomez@cognac:9>
```

Máquina cliente: tequila

```
rogomez@tequila:27>cliente toto
.
..
servicio
lectura_dir
cliente.c
cliente
servicio.c
rogomez@tequila:28>
```

Primer versión timeout

```
alarm(segs);
signal(SIGALRM, t_nop);

 /* se envia el mensaje */
 addrlen = sizeof(svaddr);
 if ( sendto(sd, &pet, sizeof(pet), 0, (struct sockaddr *)&svaddr, addrlen) == -1) {
            perror("sendto");
            exit(1);
 }

 /* se espera por la respuesta */
 if  ( recvfrom(sd, &resp, sizeof(resp), 0, (struct sockaddr *)&claddr, &addrlen) == -1) {
            perror("recvfrom");
            exit(1);
 }

 /*resetear la alarma */
 alarm(0);
```

```
void t_nop()
 {
   printf("Servidor no responde, abortando... \n");
   exit(2);
 }
```


Segunda versión timeout

```
intentos = 1;
 siginterrupt(SIGALRM,1);
 DE_NUEVO: signal(SIGALRM, t_nop);
             alarm(5);
addrlen = sizeof(svaddr);
  if ( sendto(sd, &pet, sizeof(pet), 0, (struct sockaddr *)&svaddr, addrlen) == -1) {
            perror("sendto");
            exit(1);
  }
  r = recvfrom(sd, &resp, sizeof(resp), 0,0,0);
  if (r == -1) {
    if (errno == EINTR) {
      if (intentos == n) {
        printf("Abortando, servidor no responde despues de %d mensajes enviados \n",intentos);
        exit(1);
      }
       intentos++;
       goto DE_NUEVO;
    }
    else {
            perror("recvfrom");
            exit(1);
    }
  }
 alarm(0);
```

```
void t_nop()
 {
   printf("Servidor no responde, se envia el “);
   printf(“mensaje de nuevo (%d) \n",intentos);
 }
```

La llamada siginterrupt

- Sintaxis: int siginterrupt(int sig, int flag);
- Cambia el comportamiento de reanudación cuando una señal interrumpe una llamada al sistema.
  - Dependiendo del valor de flag y de si se ha realizado transferencia de datos o no la llamada al sistema se relanzará, se interrumpirá simplemente, o se interrumpirá devolviendo la cantidad de datos transmitida
- Si el argumento flag es falso (0), entonces las llamadas al sistema se reanudarán si han sido interrumpidas por la señal especificada en sig.
- Si el argumento flags es verdad (distinto de cero, 1 por ejemplo) y no se han transferido datos, cuando una señal sig interrumpe una llamada al sistema, ésta devolverá -1 y la variable global errno contendrá el valor EINTR.