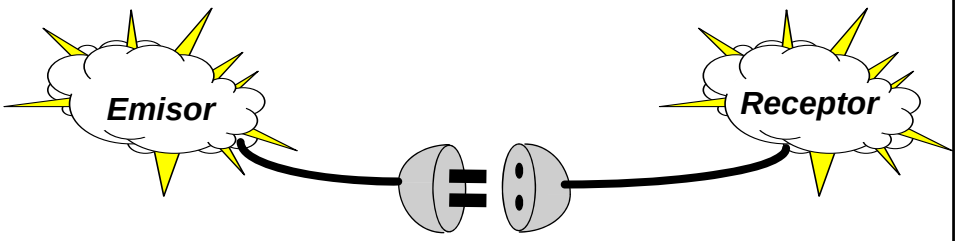
Los sockets de Unix



Dr. Roberto Gómez Cárdenas
ITESM-CEM
rogomez@itesm.mx
http://homepage.cem.itesm.mx/rogomez

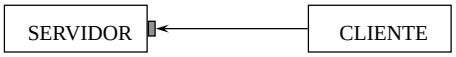
La comunicación

- 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



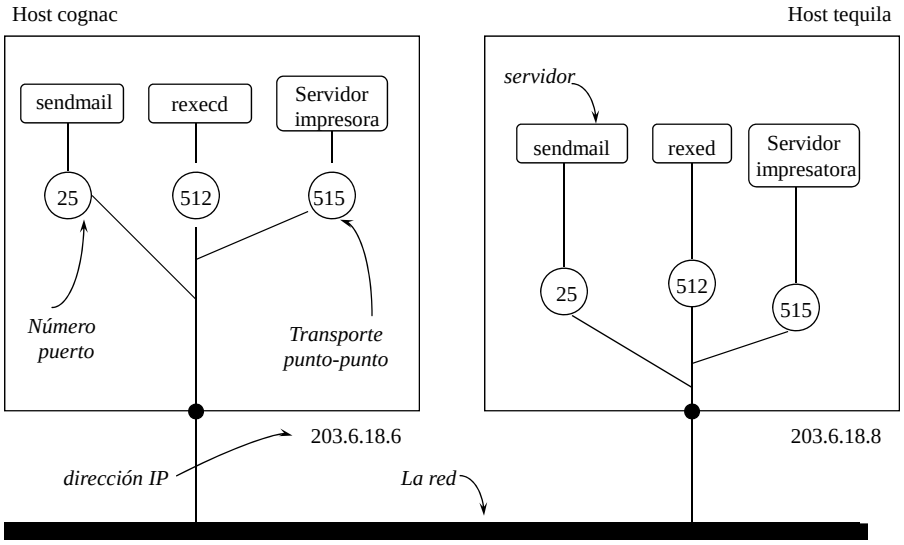
El concepto de puerto

- 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

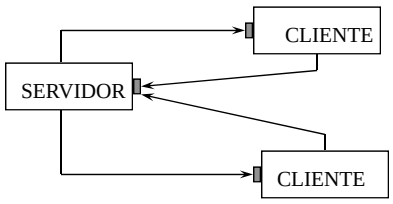
Servidor:
realiza el servicio

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( )*

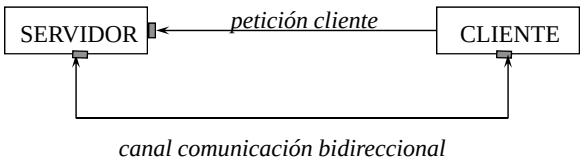
recvfrom( ) y *sendto( )* son la base del protocolo conectionless de sockets

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:

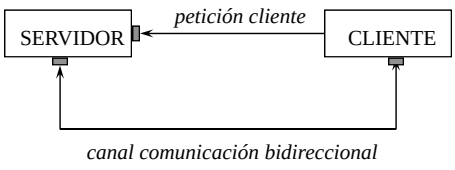
Ningún otro proceso puede aceptarlo
Protocolos orientados conexión:
mecanismo llamado *hand off*

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

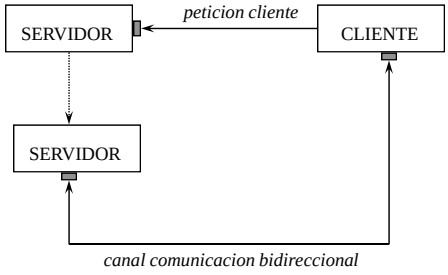


Pseudocódigo estrategia servidor serial

```
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

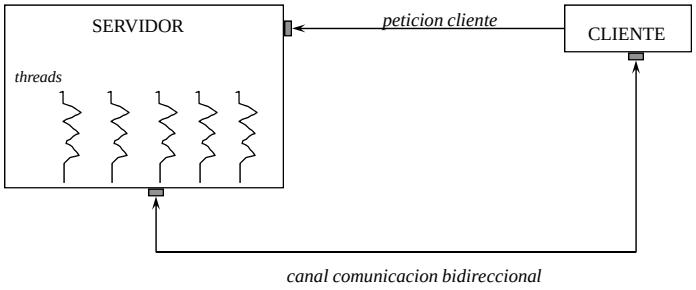


Pseudocódigo estrategia servidor padre

```
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
$
```

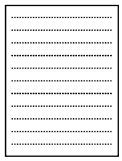

Obtención datos de otra máquina: archivo /ect/hosts

- 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.*

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
$
```


Ejemplo uso gethostbyaddr (2/2)

```
 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);
```

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

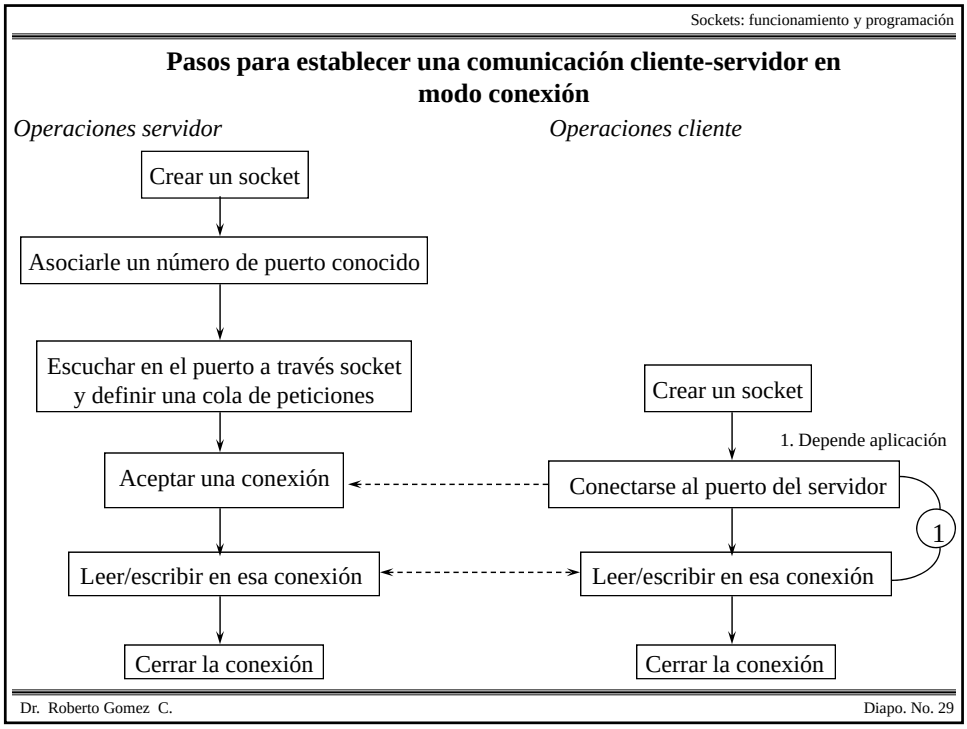
Internet domain addres

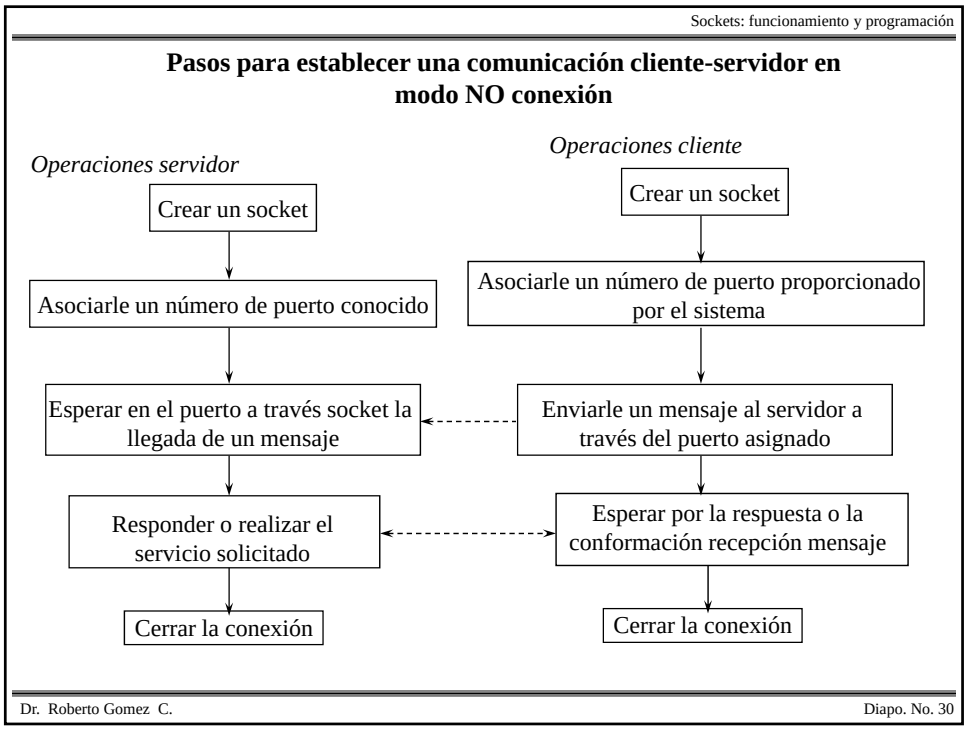
```
struct sockaddr_in {
   short          sin_family;
   u_short        sin_port;
   struct in_addr sin_addr;
   char           sin_zero;
}

struct in_addr {
   u_long   s_addr;
}
```

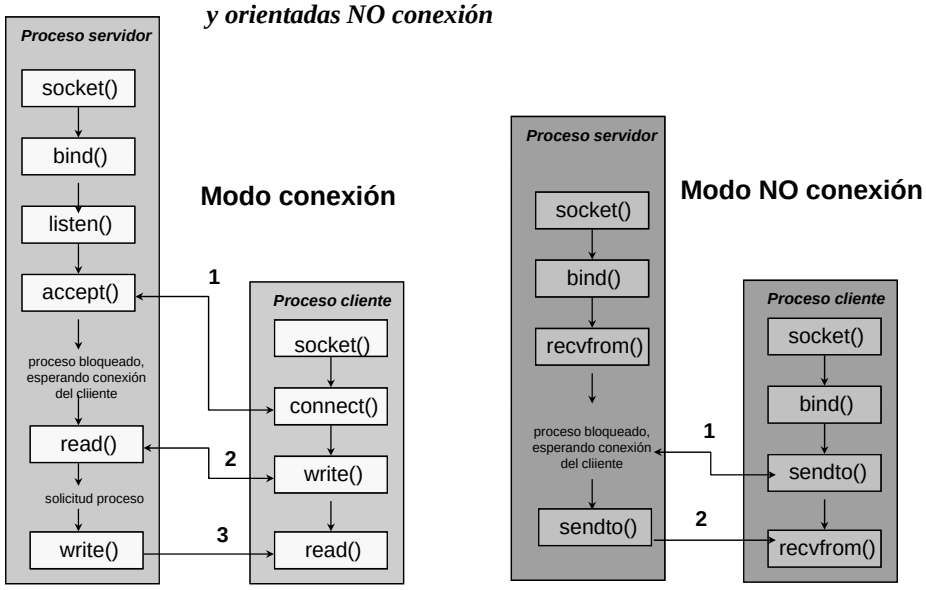
Generic socket addres

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


Sockets: funcionamiento y programación
Pasos para establecer una comunicación cliente-servidor en modo conexión
Operaciones servidor
Operaciones cliente
Crear un socket
Asociarle un número de puerto conocido
Escuchar en el puerto a través socket y definir una cola de peticiones
Crear un socket
1. Depende aplicación
Aceptar una conexión
Conectarse al puerto del servidor
1
Leer/escribir en esa conexión
Leer/escribir en esa conexión
Cerrar la conexión
Cerrar la conexión
Dr. Roberto Gomez C.
Diapo. No. 29


Sockets: funcionamiento y programación
Pasos para establecer una comunicación cliente-servidor en modo NO conexión
Operaciones servidor
Operaciones cliente
Crear un socket
Crear un socket
Asociarle un número de puerto conocido
Asociarle un número de puerto proporcionado por el sistema
Esperar en el puerto a través socket la llegada de un mensaje
Enviarle un mensaje al servidor a través del puerto asignado
Responder o realizar el servicio solicitado
Esperar por la respuesta o la conformación recepción mensaje
Cerrar la conexión
Cerrar la conexión
Dr. Roberto Gomez C.
Diapo. No. 30

Llamadas sistemas involucradas en comunicaciones orientadas conexión y orientadas NO conexión



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 sd;
        :
sd = socket(AF_INET, SOCK_STREAM, 0);
if  (sd < 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 sd;
struct sockaddr_in   server;
                  :
sd = socket(AF_INET, SOCK_STREAM, 0);
                  :
server.sin_family            = AF_INET;
server.sin_addr.s_addr   = htonl(INADDR_ANY);
server.sin_port              = htons(NUM_PUERTO);

if  ( bind(sd, (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 sd;
      :
sd = socket(AF_INET, SOCK_STREAM, 0);
      :
bind(sd, (struct sockaddr*) &server, sizeof(server)) );
      :
listen(sd, 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*)

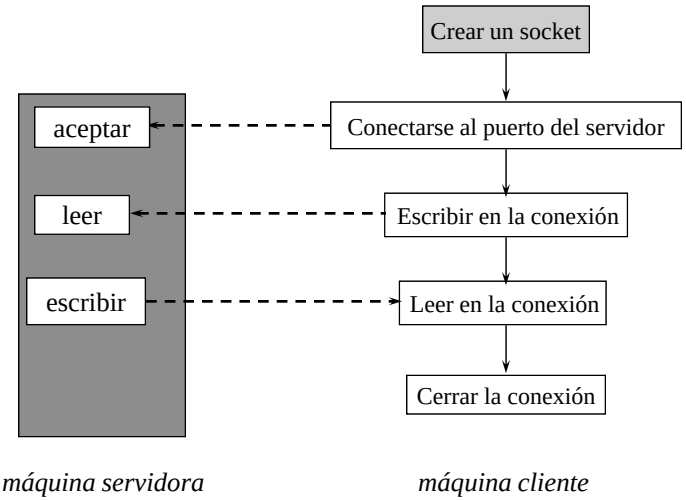
int *addrlen
- argumento de resultado: el cliente que llama deja un valor antes de llamar a *accept()*, y la llamada de sitema guarda un resultado en esa variable.
- generalmente el cliente lo utiliza para dar a conocer el tamaño de un buffer, y la llamada regresa la cantidad de información almacenada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

Ejemplo uso llamada *accept()*

```
int sd, tam_cliente;
struct sockaddr_in  server, cliente;
                :
sd= socket(AF_INET, SOCK_STREAM, 0);
                :
bind(sd, (struct sockaddr*) &server, sizeof(server)) );
                :
listen(sd, 5);
tam_cliente = sizeof(cliente)
if ( (fd = accept(sd, (struct sockaddr *) &cliente, &tam_cliente)) < 0) {
      fprintf(stderr,"Error en la aceptacion de la conexión \n");
      exit(1);
}
```

<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 HOST            "192.43.235.6"
#define PUERTO          6543

int                     sd;
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_INET, SOCK_STREAM, 0)

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);
```

Ejemplo de uso *write()* y *read()*

```
int sd, tam_cliente;
struct sockaddr_in  server, cliente;
                :
sd= socket(AF_INET, SOCK_STREAM, 0);
                :
bind(sd, (struct sockaddr*) &server, sizeof(server)) );
                :
listen(sd, 5);
tam_cliente = sizeof(cliente)
fd = accept(sd, (struct sockaddr *) &cliente, &tam_cliente);

 read(fd, &peticion, sizeof(peticion));

 respuesta = servicio(peticion);

 write(fd, &peticion, sizeof(respuesta));
```


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 revform( ) y sendto( )

```
int sd, tam_cliente;
struct sockaddr_in  ego, cliente;
                :
sd= socket(AF_INET, SOCK_STREAM, 0);
                :
bind(sd, (struct sockaddr*) &ego, sizeof(ego));


 tam_cliente = sizeof(cliente)
 recvfrom(fd, &peticion, sizeof(peticion), 0, (struct sockaddr*) &cliente, &tam_cliente);

 respuesta = servicio(peticion);

 sendto(fd, &respuesta, sizeof(respuesta), 0, (struct sockaddr*) &cliente, tam_cliente);
```

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);
```