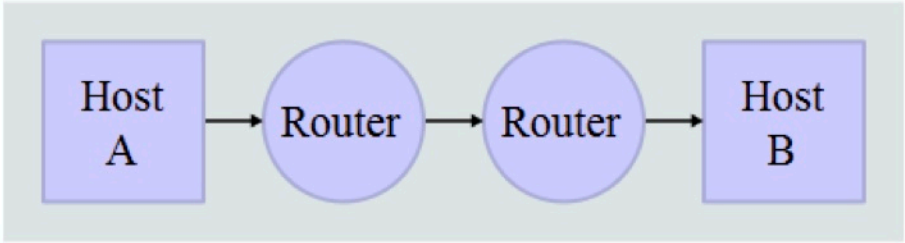
Socket Programming

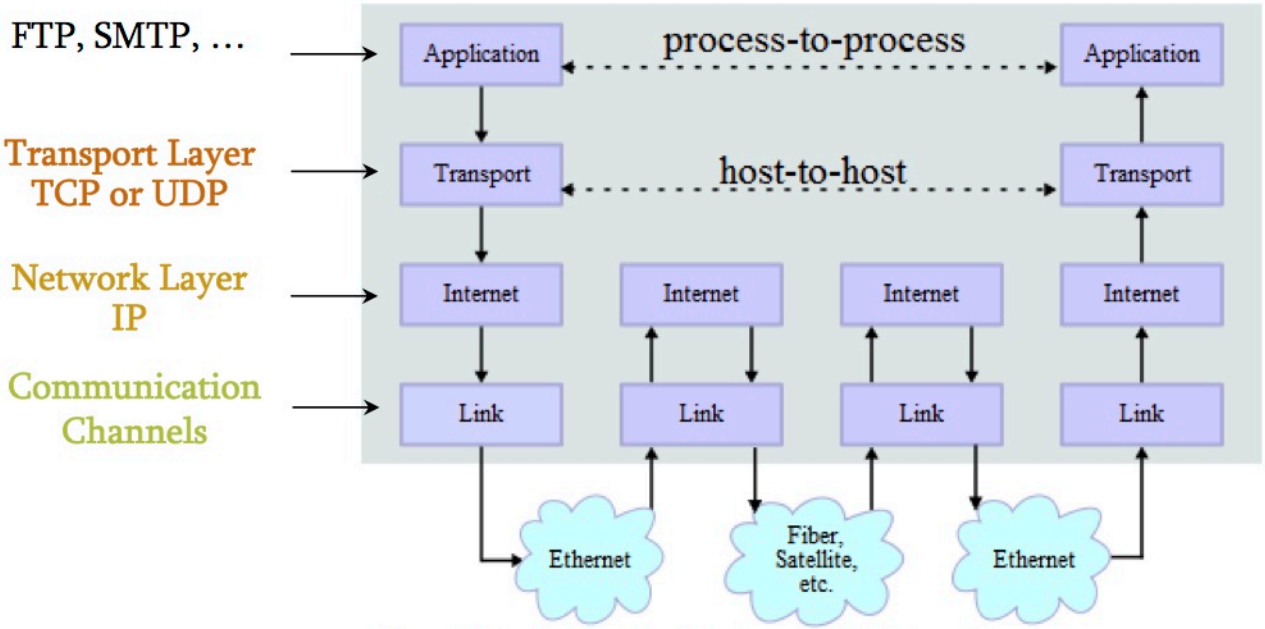
R. Hasimoto

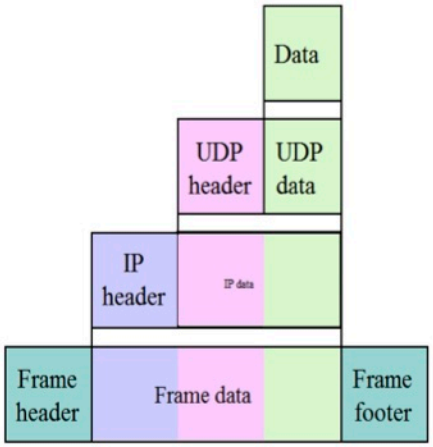
TCP/IP

Network Topology *



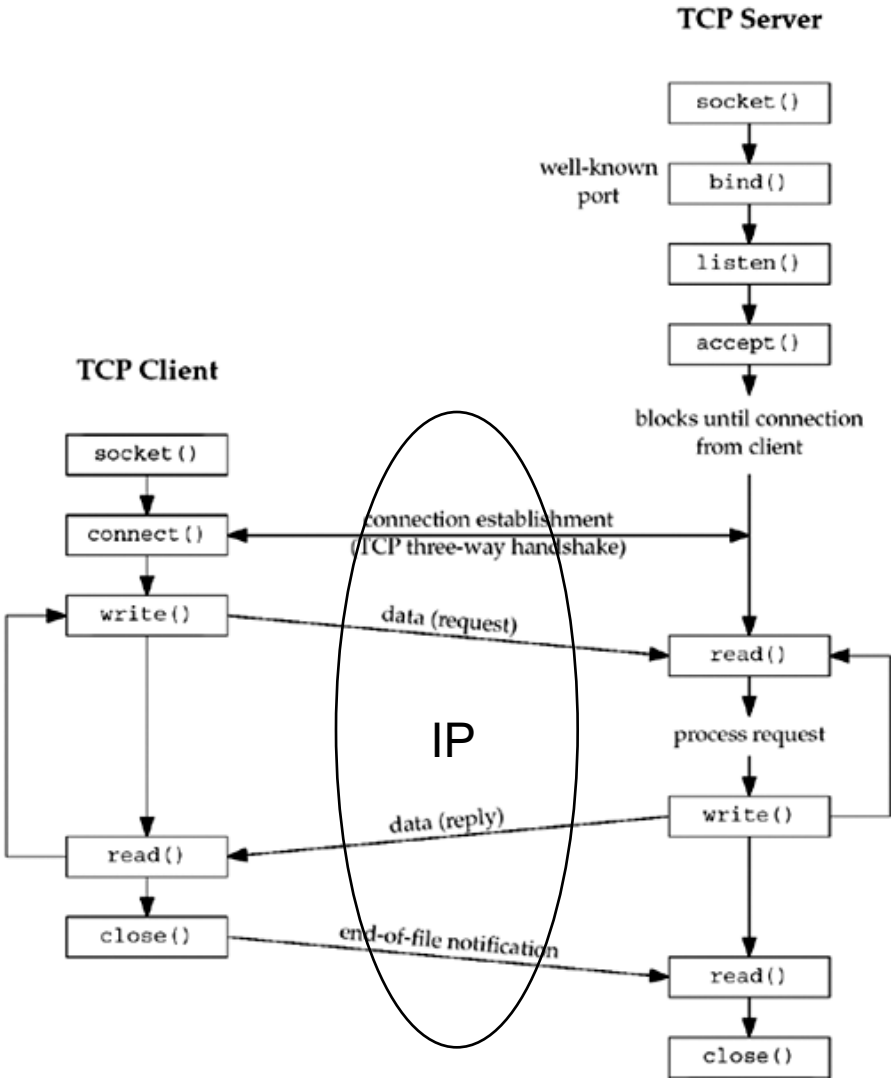
Data Flow





* image is taken from "http://en.wikipedia.org/wiki/TCP/IP_model"

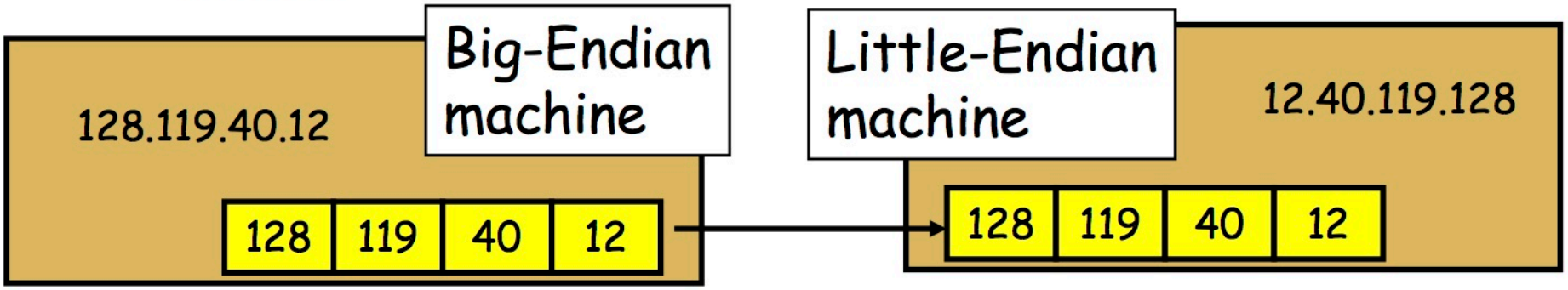
Basic socket call for client-server



Constructing Messages - Byte Ordering

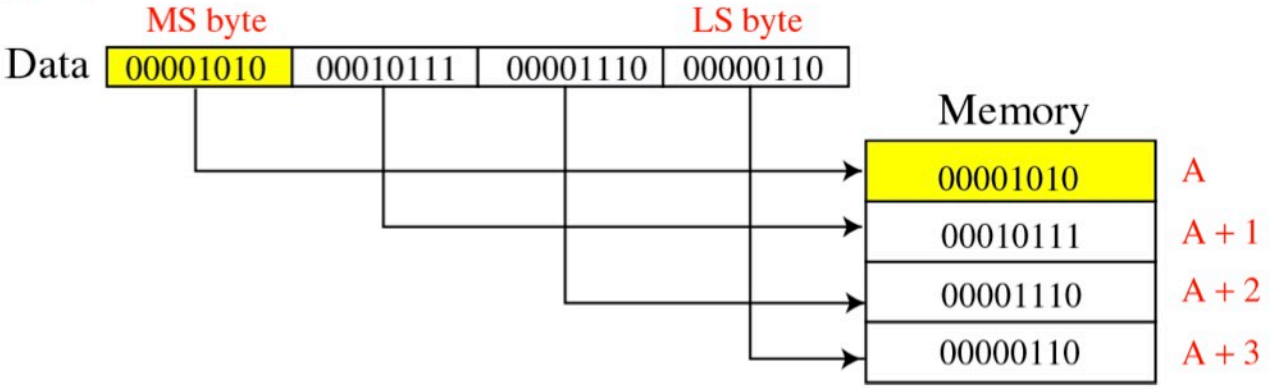
- Address and port are stored as integers
  - u_short sin_port; (16 bit)
  - in_addr sin_addr; (32 bit)

- Problem:
  - different machines / OS's use different word orderings
    - little-endian: lower bytes first
    - big-endian: higher bytes first
  - these machines may communicate with one another over the network

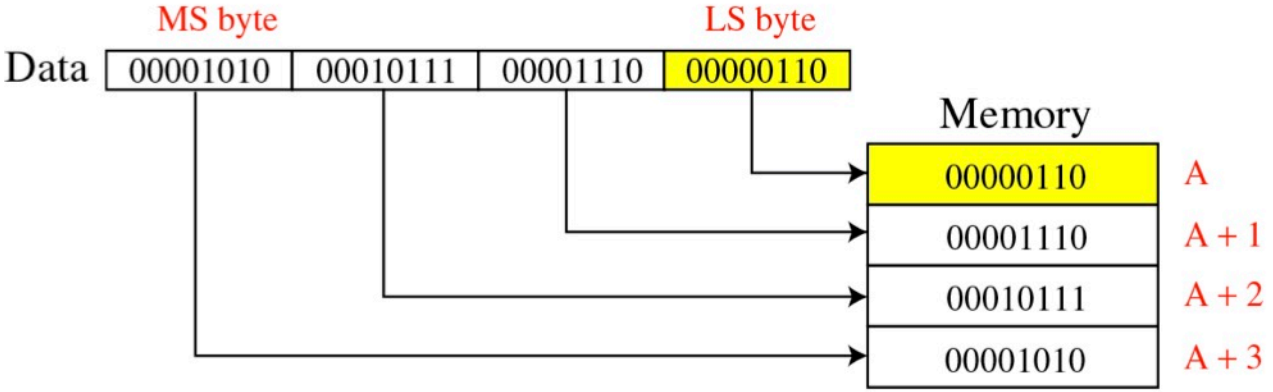


Constructing Messages - Byte Ordering

- Big-Endian:



- Little-Endian:

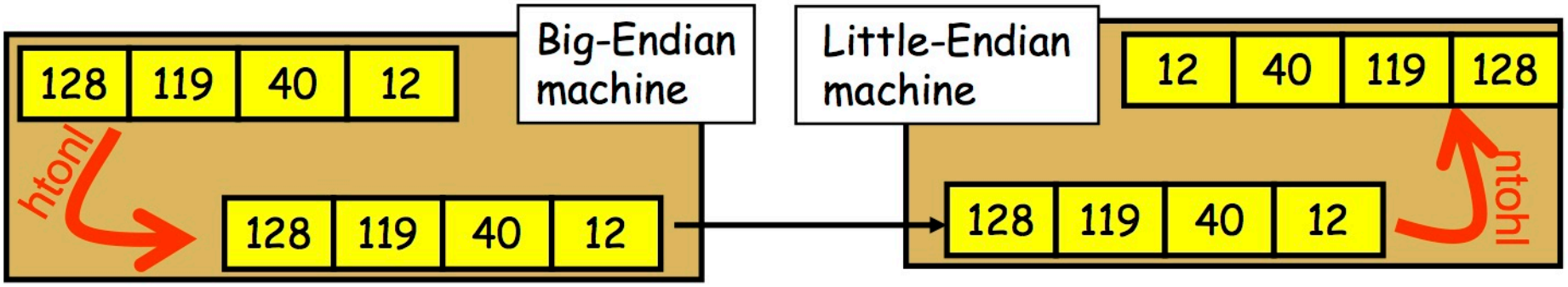


Constructing Messages - Byte Ordering - Solution: Network Byte Ordering

- Host Byte-Ordering: the byte ordering used by a host (big or little)
- Network Byte-Ordering: the byte ordering used by the network – always big-endian

- `u_long htonl(u_long x);`
- `u_short htons(u_short x);`
- `u_long ntohl(u_long x);`
- `u_short ntohs(u_short x);`

- On big-endian machines, these routines do nothing
- On little-endian machines, they reverse the byte order



Client Program

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

int main( void )
{
   struct sockaddr_in peer;
   int s;
   int rc;
   char buf[ 1 ];

   peer.sin_family = AF_INET;
   peer.sin_port = htons( 7500 );
   peer.sin_addr.s_addr = inet_addr( "127.0.0.1" );
   s = socket( AF_INET, SOCK_STREAM, 0 );
   if ( s < 0 )
   {
              perror( "socket call failed" );
              exit( 1 );
   }
```

```
rc = connect( s, ( struct sockaddr * )&peer,
                   sizeof( peer ) );
if ( rc )
{
              perror( "connect call failed" );
              exit( 1 );
}
rc = send( s, "1", 1, 0 );
if ( rc <= 0 )
{
              perror( "send call failed" );
              exit( 1 );
}
rc = recv( s, buf, 1, 0 );
if ( rc <= 0 )
              perror( "recv call failed" );
else
              printf( "%c\n", buf[ 0 ] );
exit( 0 );
}
```

Server Program

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

int main( void )
{
   struct sockaddr_in local;
   int s;
   int s1;
   int rc;
   char buf[ 1 ];

   local.sin_family = AF_INET;
   local.sin_port = htons( 7500 );
   local.sin_addr.s_addr = htonl( INADDR_ANY );
   s = socket( AF_INET, SOCK_STREAM, 0 );
   if ( s < 0 )
   {
                perror( "socket call failed" );
                exit( 1 );
   }
  rc = bind( s, ( struct sockaddr * )&local, sizeof( local
) );
  if ( rc < 0 )
  {
                perror( "bind call failure" );
                exit( 1 );
  }
  rc = listen( s, 5 );
  if ( rc )
  {
                perror( "listen call failed" );
                exit( 1 );
  }
  s1 = accept( s, NULL, NULL );
  if ( s1 < 0 )
  {
                perror( "accept call failed" );
                exit( 1 );
  }
  rc = recv( s1, buf, 1, 0 );
  if ( rc <= 0 )
  {
                perror( "recv call failed" );
                exit( 1 );
  }
  printf( "%c\n", buf[ 0 ] );
  rc = send( s1, "2", 1, 0 );
  if ( rc <= 0 )
                perror( "send call failed" );
  exit( 0 );
  }
```

Socket creation: socket()

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

socket socket(*int family, int type, int protocol*);

Returns: socket descriptor on success, or -1 on failure.

family: AF_INET (Internet communication)
AF_LOCAL (for Interprocess communication--IPC)
type: SOCK_STREAM (full-duplex connection oriented, in TCP/IP)
SOCK_DGRAM (unreliable, best-effort service, in UDP)
SOCK_RAW (provide access to ICMP messages)
protocol: IPPROTO_TCP, IPPROTO_UDP,
usually set to 0 (i.e., default protocol).

sockfd = socket(AF_INET*, SOCK_STREAM, 0*);

socket() errors

- EACCES Permission to create a socket of the specified type and/or protocol is denied.

- EAFNOSUPPORT
  The implementation does not support the specified address family.

- EINVAL Unknown protocol, or protocol family not available.

- EMFILE Process file table overflow.

- ENFILE The system limit on the total number of open files has been reached.

- ENOBUFS or ENOMEM
  Insufficient memory is available. The socket cannot be created until sufficient resources are freed.

- EPROTONOSUPPORT
  The protocol type or the specified protocol is not supported within this domain.

Puertos

Port numbers: 16-bit integer numbers

Servidores típicos: usan puertos conocidos

-- puerto 21 → ftp

-- puerto 80 → web servers

Clientes: usan puertos éfímeros, asignados por TCP o UDP.

Los puertos son divididos en 3 rangos:

1. *Well-known ports*: 0-1023 asignados por IANA (para servidores).
2. *Registered ports*: 1024-49151, no son controlados por IANA pero sí registrados (6000-6063 son para servidores X windows).
3. *Dynamic or private ports*: 49152-65535, son llamados puertos efímeros.

IANA: Internet assigned numbers authority

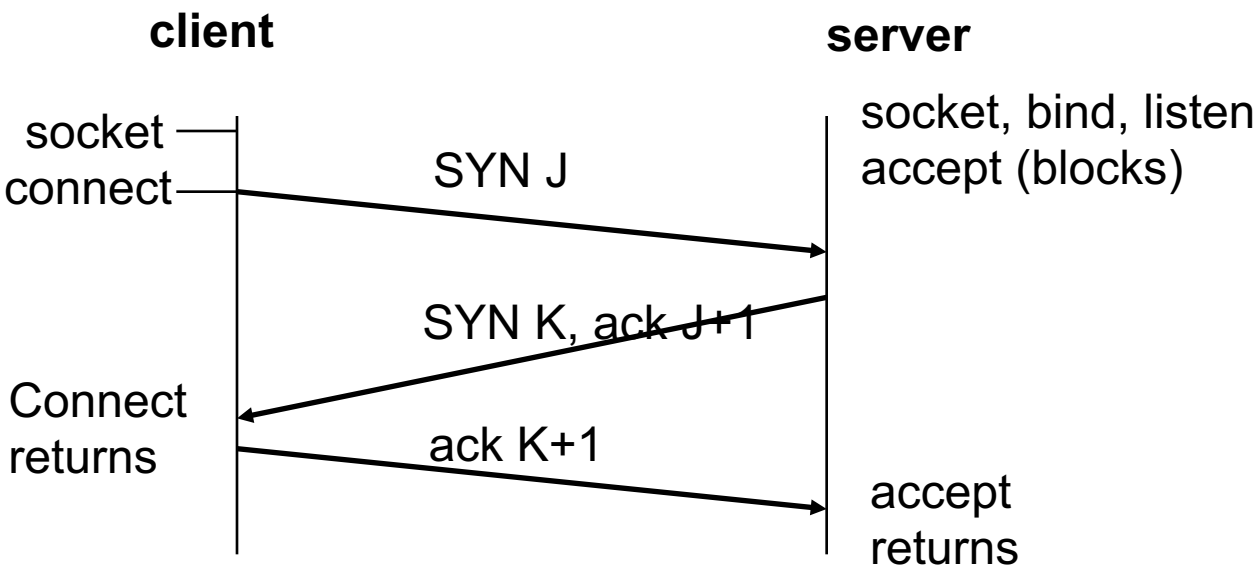
Program description

#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);

Returns: 0 on success, -1 on failure.

Se bloquea hasta que se efectúa la conexión.

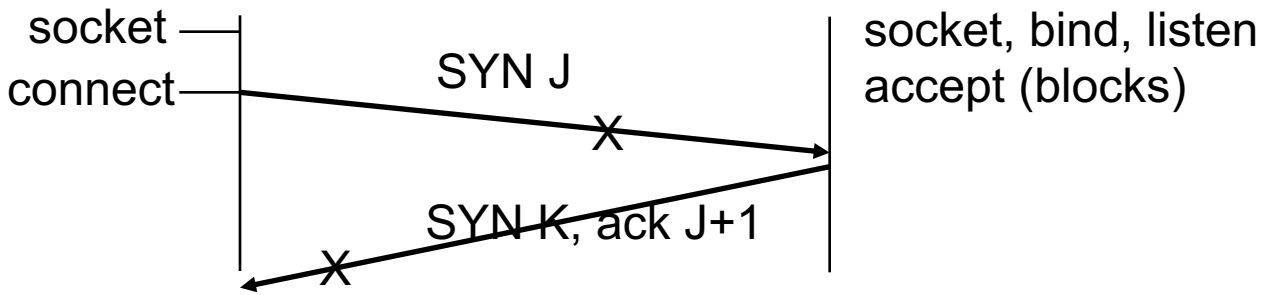


TCP three-way handshake

Program description

Fuentes de error:

1. ETIMEDOUT: cliente no recibió la respuesta a su conexión (SYN J). Llama a connect() 6 seg, 24. Si en 75seg no recibe respuesta, regresa con error (-1).



2. ECONNREFUSED: Servidor responde con RST (reset conection). El servidor no tiene ningún procesos esperando por conexión en el puerto especificado.
3. ENETUNREACH: error en la IP o nodo inalcansable (espera 75seg). Error ICMP es regresado.
4. Otros: EAGAIN (se acabaron local ports), EINTR, EISCONN, EINPROGRESS (non-blocking).

Connect… errors

- EACCES For Unix domain sockets, which are identified by pathname: Write permis-
- sion is denied on the socket file, or search permission is denied for
- one of the directories in the path prefix. (See also path_resolu-
- tion(2).)

- EADDRINUSE
- Local address is already in use.

- EAFNOSUPPORT
- The passed address didnât have the correct address family in its sa_fam-
- ily field.

- EAGAIN No more free local ports or insufficient entries in the routing cache.
- For PF_INET see the net.ipv4.ip_local_port_range sysctl in ip(7) on how
- to increase the number of local ports.

- EALREADY
- The socket is non-blocking and a previous connection attempt has not yet
- been completed.

- EBADF The file descriptor is not a valid index in the descriptor table.

- EFAULT The socket structure address is outside the userâs address space.

- EINPROGRESS
- The socket is non-blocking and the connection cannot be completed imme-
- diately. It is possible to select(2) or poll(2) for completion by
- selecting the socket for writing. After select(2) indicates writability,
- use getsockopt(2) to read the SO_ERROR option at level SOL_SOCKET to
- determine whether connect() completed successfully (SO_ERROR is zero) or
- unsuccessfully (SO_ERROR is one of the usual error codes listed here,
- explaining the reason for the failure).

- EINTR The system call was interrupted by a signal that was caught.

- EISCONN
- The socket is already connected.

- ENETUNREACH
- Network is unreachable.

- ENOTSOCK
- The file descriptor is not associated with a socket.

Address structure

Figure 3.1 The Internet (IPv4) socket address structure: sockaddr_in Posix.1g).

```
#include <netinet/in.h>
struct in_addr {
in_addr_t s_addr;            /* 32-bit IPv4 address */
                             /* network byte ordered */
};

struct sockaddr_in {
   uint8_t sin_len;           /* length of structure (16) */
   sa_family_t sin_family;    /* AF_INET */
   in_port_t sin_port;        /* 16-bit TCP or UDP port number */
                              /* network byte ordered */
   struct in_addr sin_addr;   /* 32-bit IPv4 address */
                              /* network byte ordered */
   char sin_zero[8];          /* unused */
};
```

Inicializarla en zero antes de usarla

Puertos y direcciones IP son en network byte order

Generic socket address structure

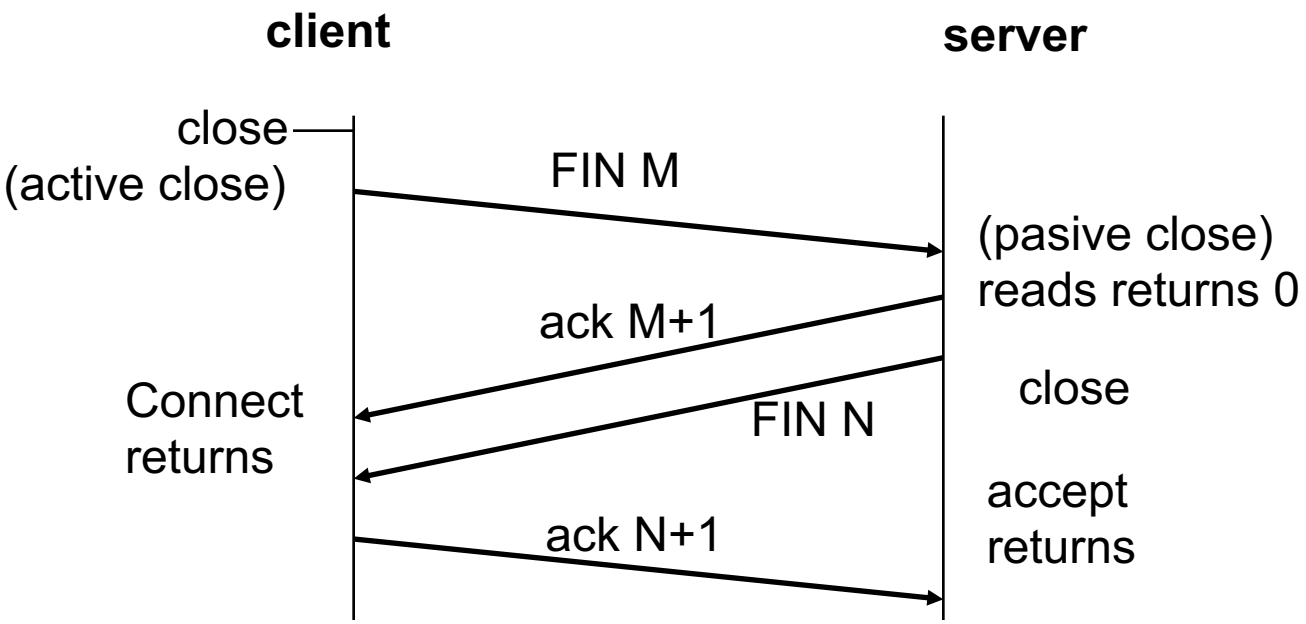
- Direcciones IP se pasan como referencia en argumentos de funciones. Por cuestiones historicas, en lugar de pasar la dirección como * void, se usa la siguiente estructura.
- #include <sys/socket.h>

```
struct sockaddr {
   uint8_t sa_len;
   sa_family_t sa_family;      /* address family: AF_xxx value */
   char sa_data[14];           /* protocol-specific address */
};
```

Ver funciones que tienen como argumento direcciones IP.

Nota: La longitud del socket se pasa dependiendo si va o viene del kernel.

TCP connection termination



Server program description

#include <sys/socket.h>
int bind (int sockfd, const struct sockaddr *localaddr, int addrlen);
Returns: 0 on success, -1 on failure.
Addrlen: longitud de la estructura *struct sockaddr.*
El socket, puerto y la dirección son unidas mediante la llamada a *bind.* En el caso de que el servidor tenga mas de una dirección IP, se puede usar la macro INADDR_ANY.

Distintas especificaciones de bind:

IP addr	Port	
INADDR_ANY	0	:Kernel escoge ambas
INADDR_ANY	no cero	:Kernel escoge IP addr
Local IP	0	: Kernel escoge puerto
Local IP	no cero	: proceso escoge IP y puerto.

Server program description

- Puede el cliente ejecutar bind?
- Puede el Servidor no ejecutar bind?
- *“Si bind no se ejecuta el kernel escoje los puertos efímeros cuando se llama a connect o listen.”*

- ERRORS
- EACCES The address is protected, and the user is not the superuser.

- EADDRINUSE
- The given address is already in use.

- EBADF sockfd is not a valid descriptor.

- EINVAL The socket is already bound to an address.

- ENOTSOCK
- sockfd is a descriptor for a file, not a socket

Server program description

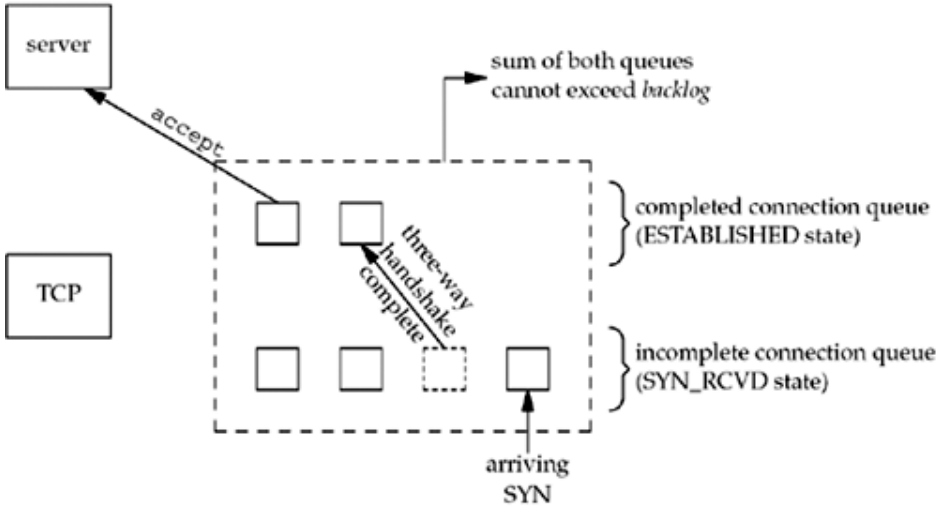
int listen (int sockfd, int backlog);

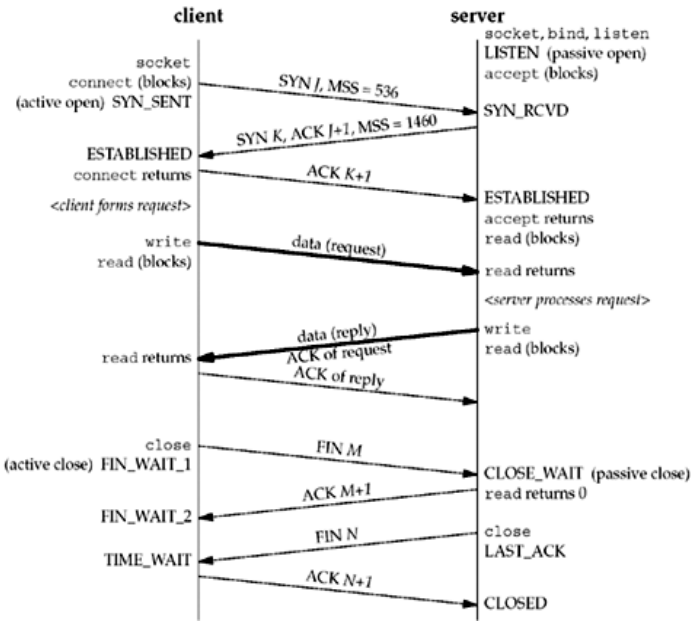
Returns: 0 on success, -1 on failure.

Backlog: defines the maximum length the queue of pending connections may grow to. (ver LISTENQ en unp.h).

Backlog = suma de las dos colas

- ERRORS
- EADDRINUSE
- Another socket is already listening on the same port.
- EBADF The argument sockfd is not a valid descriptor.
- ENOTSOCK
- The argument sockfd is not a socket.
- EOPNOTSUPP
- The socket is not of a type that supports the listen() operation.





Server program description

socket accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
Returns: a connected socket or -1 on failure. Regresa la dirección del cliente en **addr**. El kernel actualiza addrlen. Este es el socket que se usa para la transferencia de datos.

- ERRORS
- accept() shall fail if:

- EAGAIN or EWOULDBLOCK
- The socket is marked non-blocking and no connections are present to be
- accepted.

- EBADF The descriptor is invalid.

- ECONNABORTED
- A connection has been aborted.

- EINTR The system call was interrupted by a signal that was caught before a
- valid connection arrived.

- EINVAL Socket is not listening for connections, or addrlen is invalid (e.g., is
- negative).

- EMFILE The per-process limit of open file descriptors has been reached.

- ENFILE The system limit on the total number of open files has been reached.

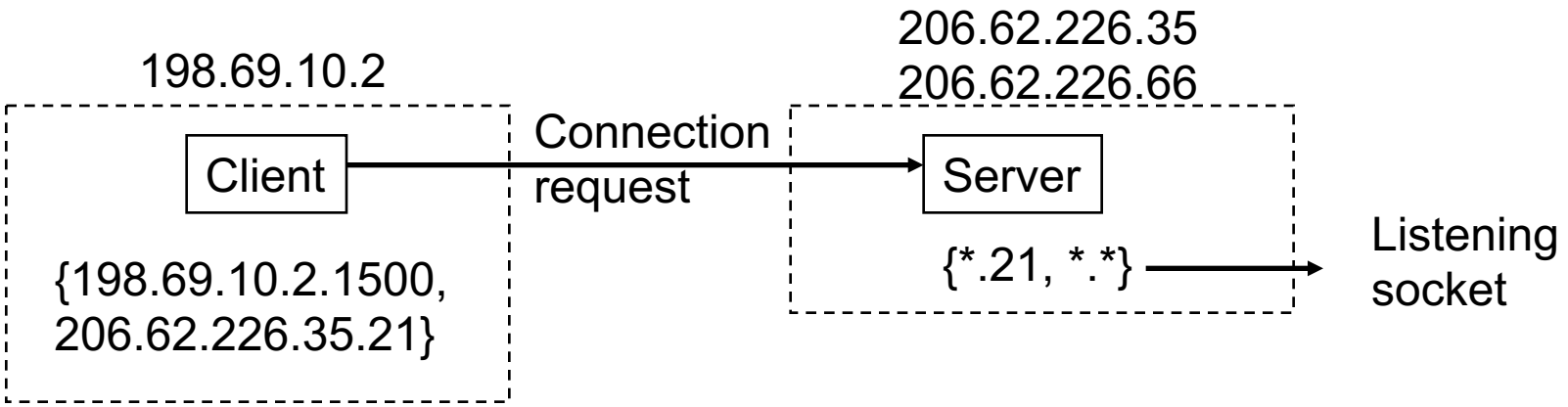
- ENOTSOCK
- The descriptor references a file, not a socket.

- EOPNOTSUPP
- The referenced socket is not of type SOCK_STREAM.

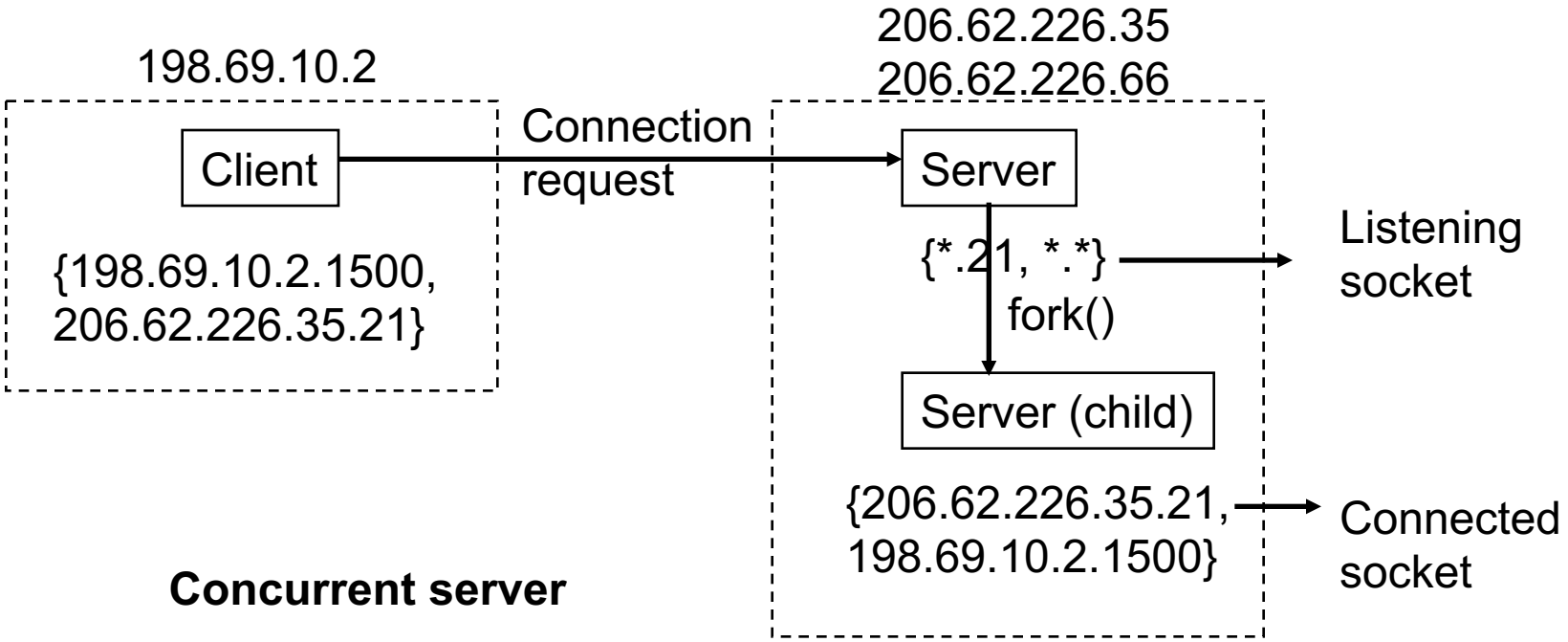
Pares de socket (socket pairs) TCP connection

- Una conexión se define por una 4-tupla: dirección local, puerto local TCP, dirección y puerto externo.
- Los dos valores que definen la conexión en cada extremo (dirección y puerto) se le llama ***socket***.

Concurrent servers

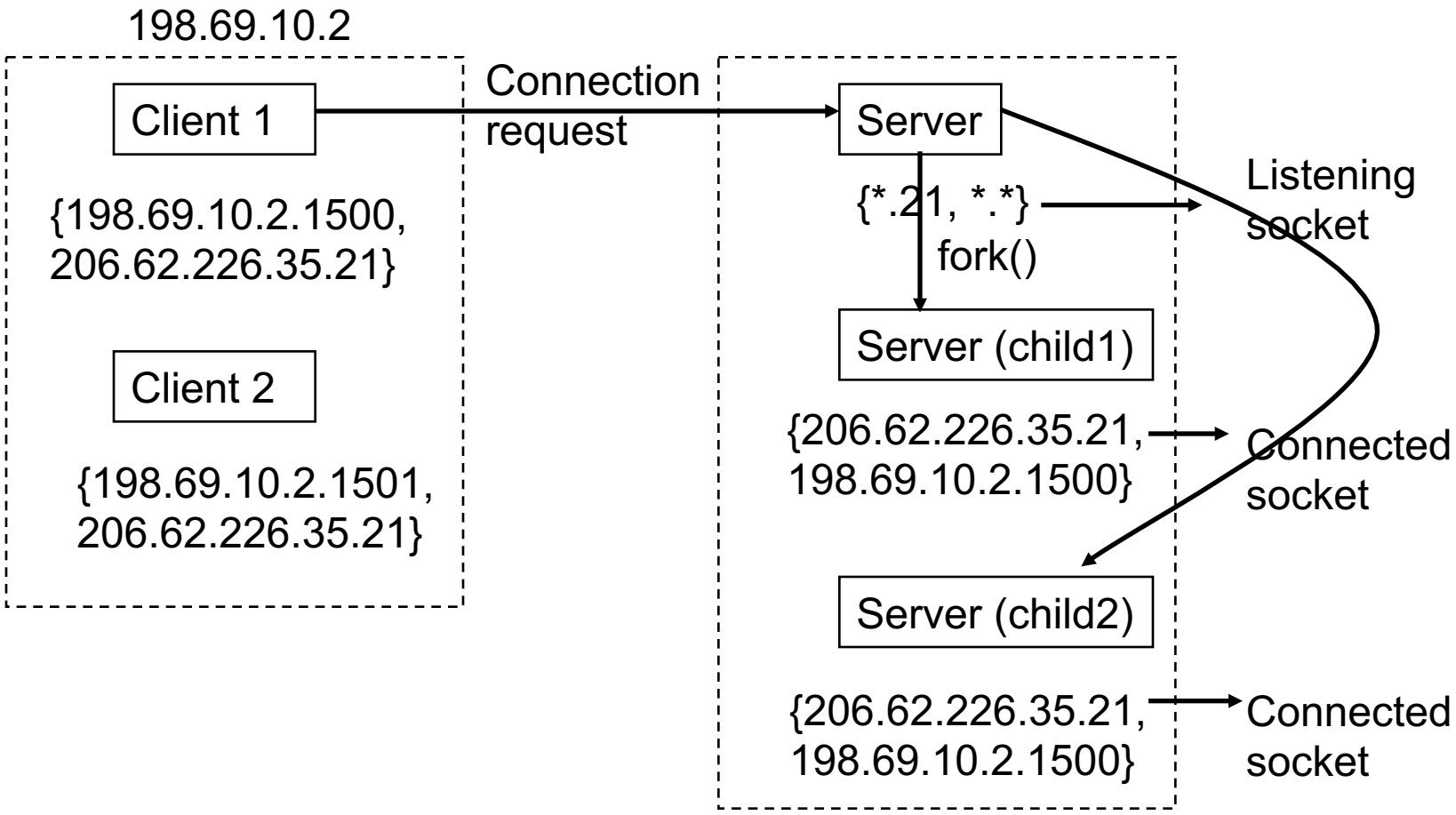


Connection request from client to server.



Concurrent server

Second client connection to same server



Concurrent server

Comentarios adicionales

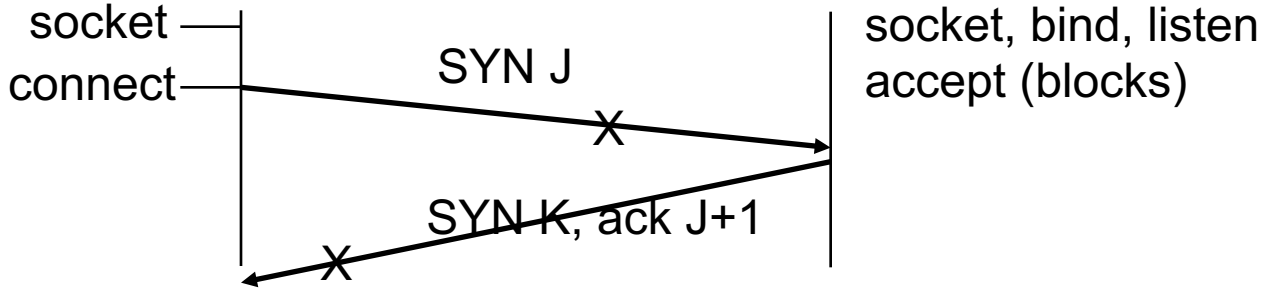
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);

Returns: 0 on success, -1 on failure. Se bloquea hasta que se efectúa la conexión.

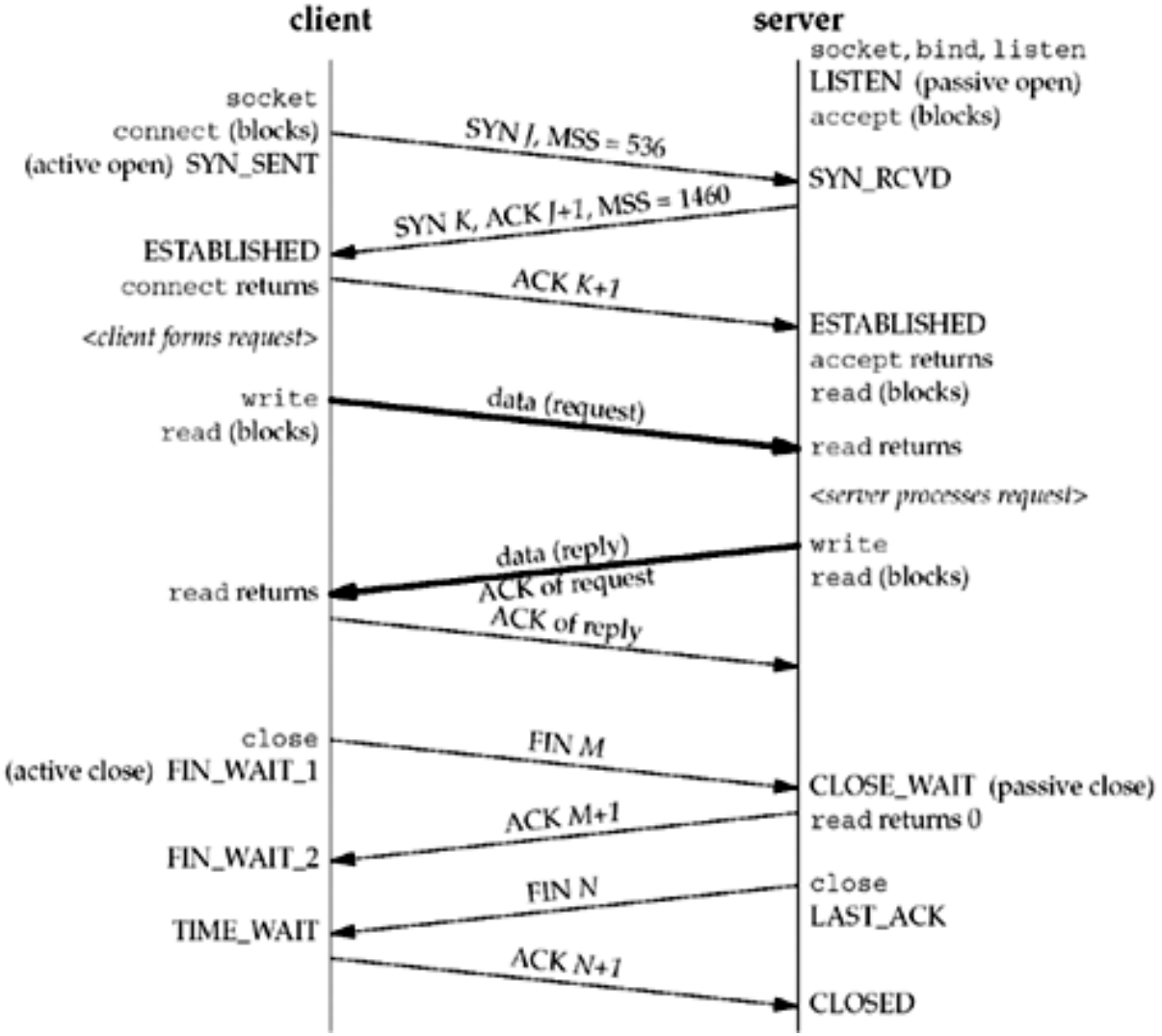
Fuentes de error:

1. ETIMEDOUT: cliente no recibió la respuesta a su conexión (SYN M). Llama a connect() 6 seg, 24. Si en 75seg no recibe respuesta, regresa con error (-1).



2. ECONNREFUSED: Servidor responde con RST (reset conection). El servidor no tiene ningún procesos esperando por conexión en el puerto especificado.
3. ENETUNREACH: error en la IP o nodo inalcansable (espera 75seg). Error ICMP es regresado.
4. Otros: EAGAIN (se acabaron local ports), EINTR, EISCONN, EINPROGRESS (non-blocking).

Comentarios adicionales



Readn/Writen bytes Functions

```
ssize_t readn(int fd, void *vptr, size_t n)
{
    size_t nleft;
    ssize_t nread;
    char *ptr;

    ptr = vptr;
    nleft = n;
   while (nleft > 0) {
        /* Read "n" bytes from a descriptor. */
        if ( (nread = read(fd, ptr, nleft)) < 0) {
            if (errno == EINTR)
                nread = 0;
            else
                return (-1);
          } else if (nread == 0)
                break;
          nleft -= nread;
          ptr+= nread
     } // while
     return (n - nleft);
   }
```

```
ssize_t writen(int fd, const void *vptr, size_t n)
{
    size_t nleft;
    ssize_t nwritten;
    const char *ptr;

    ptr = vptr;
    nleft = n;
    while (nleft > 0) {
        if ( (nwritten = write(fd, ptr, nleft)) <= 0) {
            if (nwritten < 0 && errno == EINTR)
                nwritten = 0;  // call write again
            else
                return(-1);
         }
         nleft -= nwritten;
         ptr += nwritten;
     } // while
     return (n);
}
```

Comentarios adicionales

- Que pasa cuando las colas estan llenas al arribo de un SYN?...solamente se ignora
- Los datos que llegan al servidor antes de que se complete 3WH, son almacenados en el buffer del socket.

socket accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
Returns: a connected socket or -1 on failure.
Regresa la dirección del cliente en **addr**. El kernel actualiza addrlen.

Toma la siguiente conexión establecida de la cola de listening soket.

Funciones complementarias

IP protocols usan big-endian

```
#include <netinet/in.h>
uint16_t htons(uint16_t host16bitvalue) ;
uint32_t htonl(uint32_t host32bitvalue) ;
```

Both return: value in network byte order

```
uint16_t ntohs(uint16_t net16bitvalue) ;
uint32_t ntohl(uint32_t net32bitvalue) ;
```

Both return: value in host byte order

```
#include <arpa/inet.h>
int inet_aton(const char *strptr, struct in_addr *addrptr);
                    Returns: 1 if string was valid, 0 on error
in_addr_t inet_addr(const char *strptr);   /* OBSOLETA*/
                    Returns: 32-bit binary network byte ordered IPv4 address;
                                INADDR_NONE if error
char *inet_ntoa(struct in_addr inaddr);  /*no reentrante*/
                    Returns: pointer to dotted-decimal string
```

Funciones complementarias

Trabajan en ambas plataformas: IPv4 y IPv6

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

p= presentation = ascii string
n= numeric

```
int inet_pton(int family, const char *strptr, void *addrptr);
        Returns: 1 if OK, 0 if input not a valid presentation format, -1 on error

const char *inet_ntop(int family, const void *addrptr, char *strptr, size_t len);
        Returns: pointer to result if OK, NULL on error
```

family: AF_INET o AF_INET6
len: al menos

```
#define INET_ADDRSTRLEN 16 /* for IPv4 dotted-decimal */
#define INET6_ADDRSTRLEN 46 /* for IPv6 hex string */
```

Ejemplos:

```
char sptr[INET_ADDRSTRLEN], saddr[ ] ={“192.34.12.34”}
inet_pton(AF_INET,saddr,&addr.sin_addr);
Inet_ntop(AF_INET,&addr.sin_addr,sptr,sizeof(sptr));
```

Funciones complementarias

- const char *inet_ntop(int family, const void *addrptr, char *strptr, size_t len);
- Returns: pointer to result if OK, NULL on error

  errno set to EAFNOSUPPORT if ***family*** was not set to a valid address family, or to ENOSPC if the converted address string would exceed the size of ***strptr*** given by the ***len*** argument

- Ejemplos:
- char sptr[INET_ADDRSTRLEN], saddr[ ] ={“192.34.12.34”}
- inet_pton(AF_INET,saddr,&addr.sin_addr);
- Inet_ntop(AF_INET,&addr.sin_addr,sptr,sizeof(sptr));

Funciones complementarias

4.2BSD

#include <strings.h>

void **bzero**(void *dest, size_t nbytes);

void **bcopy**(const void *src, void *dest, size_t nbytes);

int **bcmp**(const void *ptr1, const void *ptr2, size_t nbytes);

Returns: 0 if equal, nonzero if unequal

ANSI C

#include <string.h>

void ***memset**(void *dest, int c, size_t len); //**constant byte c**

regresa un apuntador a dest

void ***memcpy**(void *dest, const void *src, size_t nbytes);

int **memcmp**(const void *ptr1, const void *ptr2, size_t nbytes);

Returns: 0 if equal, <0 or >0 if unequal

Funciones de E/S exclusivas para sockets orientados a la conexión

- read and write se usan normalmente.
- ssize_t **send**(int sockfd, void *buf, size_t len, int flags);
- ssize_t **recv**(int sockfd, void *buf, size_t len, int flags);

  -sockfd: socket descriptor.

  -se leen (reciben) hasta *len* bytes en (de) *buf*.

  -flags (0 no actions):

  MSG_DONTWAIT: non-blocking operation.

  MSG_PEEK: lee datos sin eliminarlos del buffer.

  MSG_DONTROUTE: send to host in current network

  MSG_MORE : use piggybacking

  y varios más.

Errores in send and recv

On success, these calls return the number of characters sent or received. On error, -1 is returned, and errno is set appropriately.

errno para send:
-EAGAIN or EWOULDBLOCK
The socket is marked non-blocking and the requested operation would block.
-ECONNRESET
Connection reset by peer.
-EINTR A signal occurred before any data was transmitted.
- ENOBUFS
The output queue for a network interface was full.

Errno para recv:
If no messages are available at the socket, the receive calls wait for a message to arrive, unless the socket is nonblocking (see fcntl(2)), in which case the value -1 is returned and the external variable errno set to EAGAIN. The receive calls normally return any data available, up to the requested amount, rather than waiting for receipt of the full amount requested.
The return value will be 0 when the peer has performed an orderly shutdown.
-EAGAIN
-EINTR

Ver ejemplos: read and write (lectura archivos y chat).

Funciones complementarias

```
#include <sys/socket.h>
int getsockname(int sockfd, struct sockaddr *localaddr,
    socklen_t *addrlen);
int getpeername(int sockfd, struct sockaddr *peeraddr,
    socklen_t *addrlen);
```

Both return: 0 if OK, -1 on error