Facultad de Ingeniería en Electricidad y Computación

Programación de Sistemas CCPG1051

Federico Domínguez, PhD.

Unidad 5 – Sesión 4: Uso de sockets en Linux

Agenda

- Introducción a sockets
- Estructura de datos usadas por la interface sockets
- Conexión desde el cliente
- Conexión desde el servidor
- Demostración

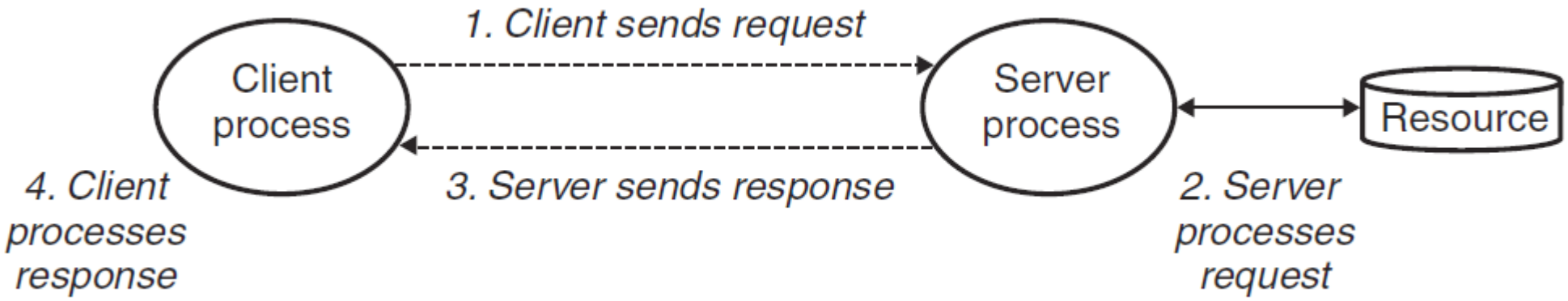
Modelo cliente-servidor

La gran mayoría de las aplicaciones en red usan el modelo cliente-servidor.

Una aplicación cliente-servidor consiste en un proceso *servidor* y uno o varios procesos *cliente*.

El proceso **servidor** custodia y administra un **recurso**.

El proceso **cliente** requiere y consume un **recurso**.

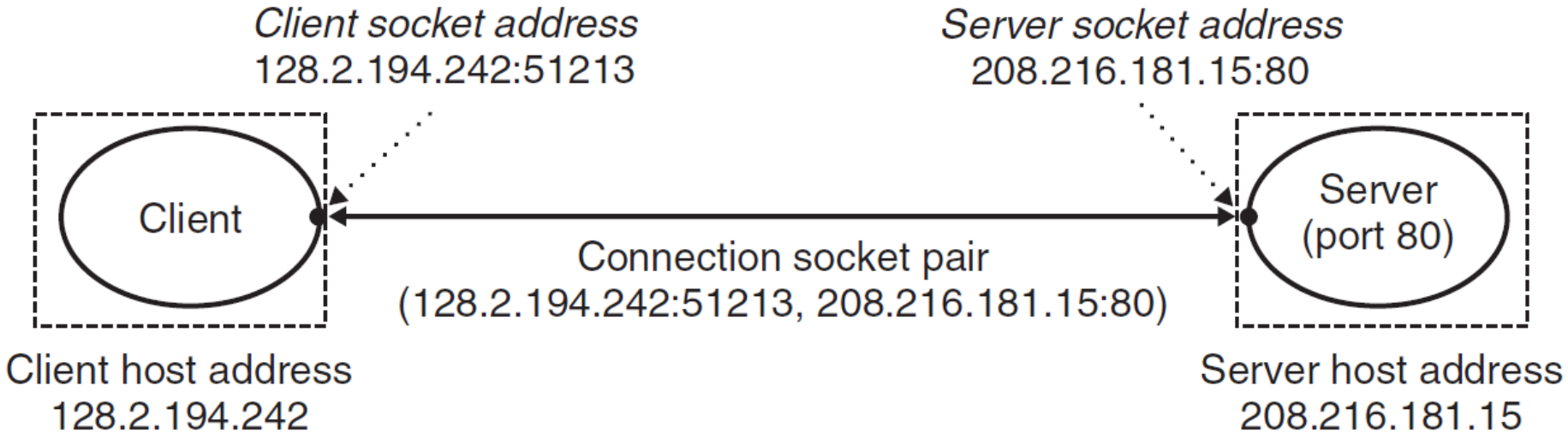


Transacción cliente - servidor

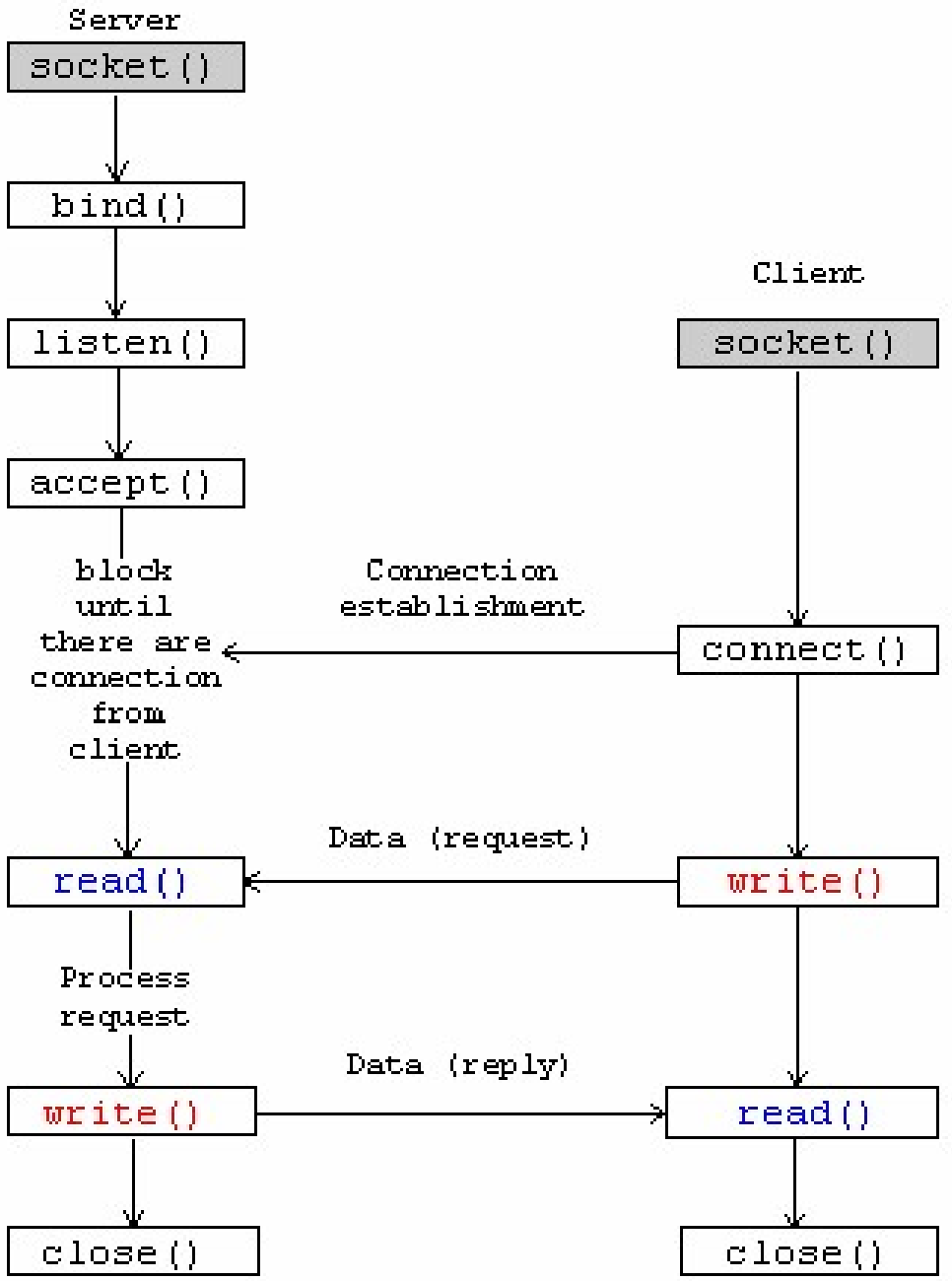
La interface de sockets en Linux

Es un grupo de funciones que son usadas en conjunto con Unix I/O para comunicación entre procesos y aplicaciones de red.

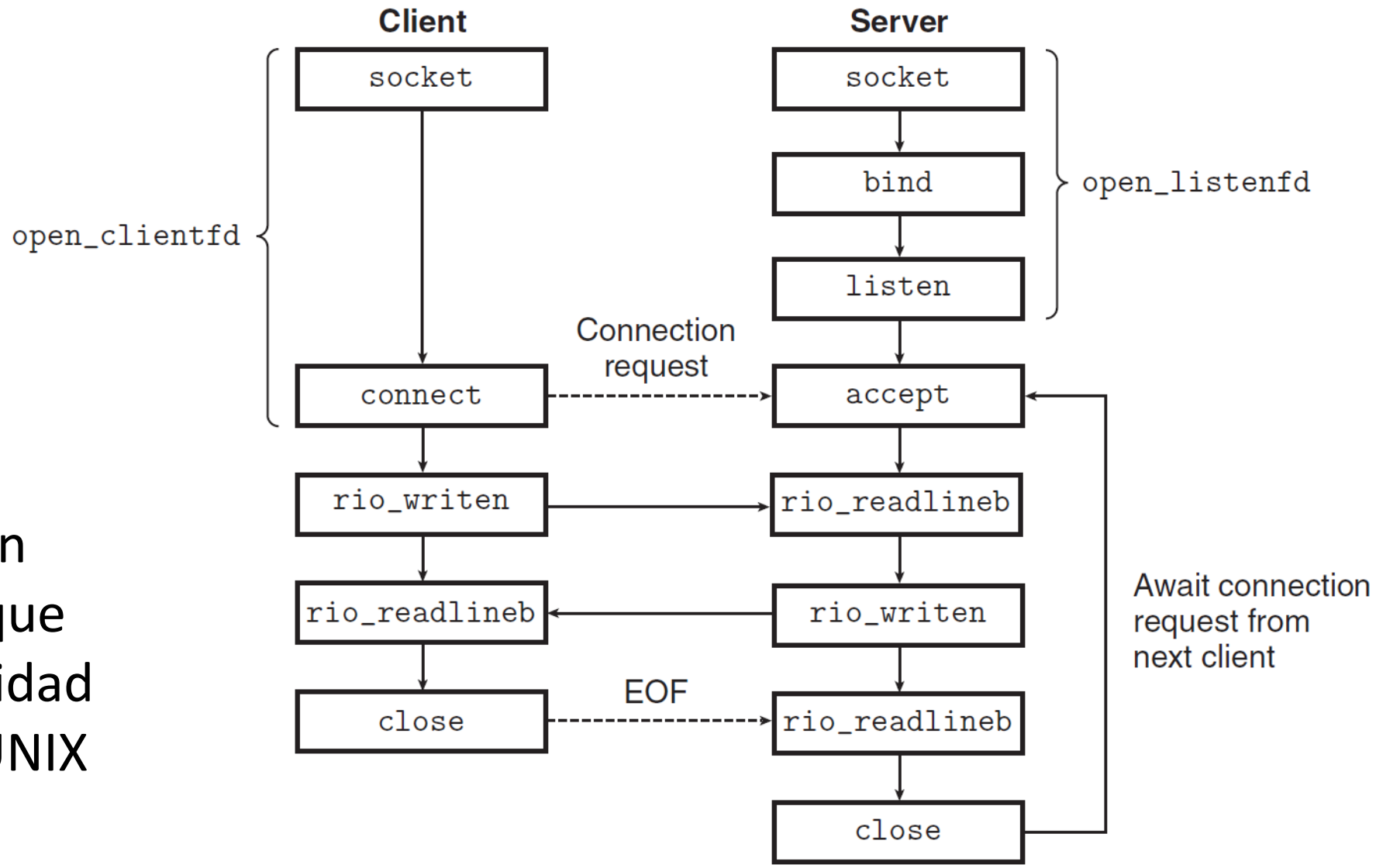
Ha sido implementada en todas las variantes UNIX, incluyendo Linux. También ha sido implementada en Windows y MacOS.



La interface de sockets usa el modelo cliente-servidor para implementar comunicación entre procesos.



Usualmente se proveen funciones “wrapper” que encapsulan la complejidad del uso de funciones UNIX I/O.



Representaciones de datos en interface sockets

Por razones históricas, la dirección IP es representada como una estructura la cual contiene un entero sin signo.

El protocolo TCP/IP es *big-endian*, por lo tanto la dirección IP es almacenada en ese formato.

netinet/in.h

```
/* Internet address structure */
struct in_addr {
    unsigned int s_addr; /* Network byte order (big-endian) */
};
```

netinet/in.h

Ejercicio:

La IP 200.10.150.37, ¿cómo estaría representada en esta estructura?

Estructura para nombres de dominio

netdb.h

```
/* DNS host entry structure */
struct hostent {
    char    *h_name;        /* Official domain name of host */
    char    **h_aliases;    /* Null-terminated array of domain names */
    int     h_addrtype;     /* Host address type (AF_INET) */
    int     h_length;       /* Length of an address, in bytes */
    char    **h_addr_list;  /* Null-terminated array of in_addr structs */
};
```

netdb.h

La estructura de un socket tiene una versión genérica y una especifica al protocolo IP. Ambas son de 16 bytes.

```
/* Generic socket address structure (for connect, bind, and accept) */
struct sockaddr {
    unsigned short  sa_family;   /* Protocol family */
    char            sa_data[14]; /* Address data.  */
};

/* Internet-style socket address structure */
struct sockaddr_in  {
    unsigned short  sin_family;  /* Address family (always AF_INET) */
    unsigned short  sin_port;    /* Port number in network byte order */
    struct in_addr  sin_addr;    /* IP address in network byte order */
    unsigned char   sin_zero[8]; /* Pad to sizeof(struct sockaddr) */
};
```

Ambos procesos, cliente y servidor, empiezan llamando a la función socket. Esta función retorna un descriptor.

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

int socket(int domain, int type, int protocol);
```

Returns: nonnegative descriptor if OK, −1 on error

```
clientfd = socket(AF_INET, SOCK_STREAM, 0);
```

El cliente inicia una conexión con un proceso servidor usando *connect*.

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

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
                                                    Returns: 0 if OK, −1 on error
```

Esta función bloquea el proceso hasta que la conexión este establecida.

La función *open_clientfd* es un conveniente "wrapper" para un proceso cliente.

```
#include "csapp.h"

int open_clientfd(char *hostname, int port);
```

Returns: descriptor if OK, −1 on Unix error, −2 on DNS error

code/src/csapp.c

```
1  int open_clientfd(char *hostname, int port)
2  {
3      int clientfd;
4      struct hostent *hp;
5      struct sockaddr_in serveraddr;
6
7      if ((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
8          return -1; /* Check errno for cause of error */
9
10     /* Fill in the server's IP address and port */
11     if ((hp = gethostbyname(hostname)) == NULL)
12         return -2; /* Check h_errno for cause of error */
13     bzero((char *) &serveraddr, sizeof(serveraddr));
14     serveraddr.sin_family = AF_INET;
15     bcopy((char *)hp->h_addr_list[0],
16           (char *)&serveraddr.sin_addr.s_addr, hp->h_length);
17     serveraddr.sin_port = htons(port);
18
19     /* Establish a connection with the server */
20     if (connect(clientfd, (SA *) &serveraddr, sizeof(serveraddr)) < 0)
21         return -1;
22     return clientfd;
23 }
```

code/src/csapp.c

El servidor usa las funciones *bind, listen y accept*.

La función *bind* le dice al *kernel* que asocie el respectivo socket a la dirección proporcionada.

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

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

Returns: 0 if OK, −1 on error

La función listen convierte al socket en pasivo, es decir en este momento se le notifica al *kernel* que el socket va a actuar como servidor.

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

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

Returns: 0 if OK, −1 on error

El argumento `backlog` define la longitud máxima a la que puede crecer la cola de conexiones pendientes para `sockfd`.

En el proceso servidor, las funciones *socket, bind y listen* pueden ser combinadas en un conveniente “wrapper”.

```
#include "csapp.h"

int open_listenfd(int port);
```

Returns: descriptor if OK, −1 on Unix error

code/src/csapp.c

```
1  int open_listenfd(int port)
2  {
3      int listenfd, optval=1;
4      struct sockaddr_in serveraddr;
5
6      /* Create a socket descriptor */
7      if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
8          return -1;
9
10     /* Eliminates "Address already in use" error from bind */
11     if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
12                    (const void *)&optval , sizeof(int)) < 0)
13         return -1;
14
15     /* Listenfd will be an end point for all requests to port
16        on any IP address for this host */
17     bzero((char *) &serveraddr, sizeof(serveraddr));
18     serveraddr.sin_family = AF_INET;
19     serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
20     serveraddr.sin_port = htons((unsigned short)port);
21     if (bind(listenfd, (SA *)&serveraddr, sizeof(serveraddr)) < 0)
22         return -1;
23
24     /* Make it a listening socket ready to accept connection requests */
25     if (listen(listenfd, LISTENQ) < 0)
26         return -1;
27     return listenfd;
28 }
```

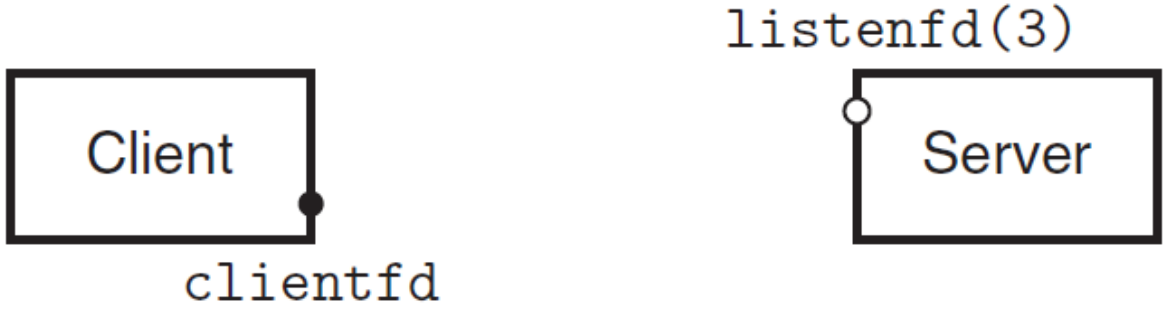
code/src/csapp.c

Finalmente, en el proceso servidor, la función *accept* es usada para obtener un descriptor conectado y responder al cliente.

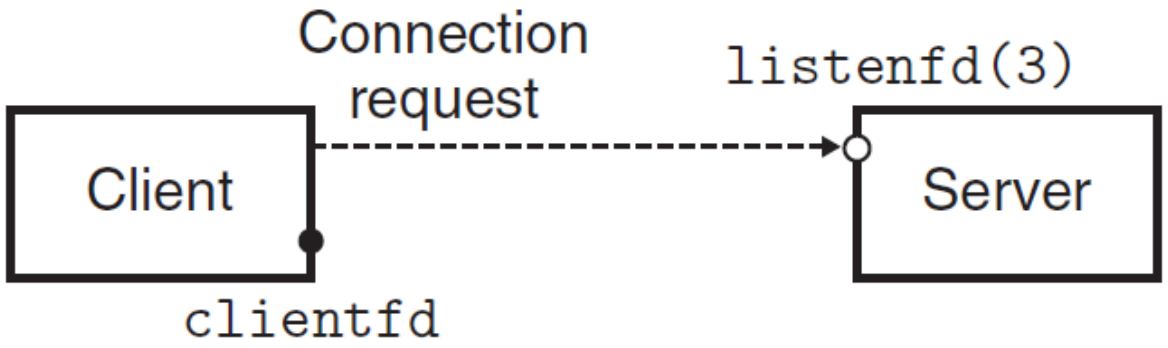
```
#include <sys/socket.h>

int accept(int listenfd, struct sockaddr *addr, int *addrlen);
```

Returns: nonnegative connected descriptor if OK, −1 on error

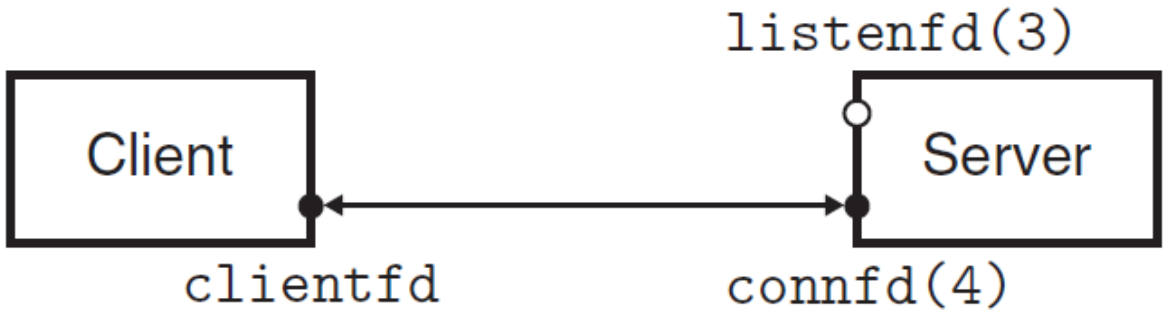
Esta función bloquea el proceso servidor hasta que el *kernel* reciba una conexión de un cliente por el socket.



1. Server blocks in `accept`*, waiting for connection request on listening descriptor* `listenfd`*.*



2. Client makes connection request by calling and blocking in `connect`*.*



3. Server returns `connfd` *from* `accept`*. Client returns from* `connect`*. Connection is now established between* `clientfd` *and* `connfd`*.*

Demostración

Aplicación “eco” cliente-servidor

Referencias

Libro guía Computer Systems: A programmers perspective. Sección 11.4