Programación con Sockets

INTRODUCCIÓN

¿Qué son los Sockets ?

- Son un mecanismo de comunicación entre procesos:
  - Dentro de una misma máquina.
  - Distribuidos en diferentes máquinas.
- Que pueden ser identificados mediante un nombre.
- Proporcionan un **interfaz** estándar para el acceso a diferentes protocolos de comunicaciones.
- Soportan, entre otros, la pila de protocolos **TCP/IP.**
  - Acceso a servicios de transporte orientados a conexión (**TCP**)
  - Acceso a servicios de transporte orientados a datagrama (**UDP**)
  - Acceso a servicios de capa de red (**IP**)

Tipos de Sockets

- Stream (SOCK_STREAM):
  - Flujo de datos bidireccional, fiable, ordenado y sin duplicados.
  - En la recepción no se conservan marcas relacionadas con la forma en que fueron enviados.
- Datagrama (SOCK_DGRAM):
  - Flujo de datos bidireccional, sin garantías de fiabilidad, orden o evitación de duplicados.
  - Los datos se recibes en los mismos bloques en que fueron enviados.
- Otros tipos:
  - Raw (SOCK_RAW)
  - Paquetes Secuenciados (SOCK_SEQPAQUET)

Dominios de Sockets

- Cada socket está asociado a un dominio de comunicaciones.
- Un dominio es una abstracción introducida para encapsular propiedades comunes de los sockets involucrados en la comunicación:
  - Esquema usado para nombre el socket.
  - Tipos de sockets disponibles dentro del dominio.
  - Protocolos de comunicaciones utilizados.
  - ...
- La familia que soporta la pila de protocolos TCP/IP es:

PF_INET o AF_INET

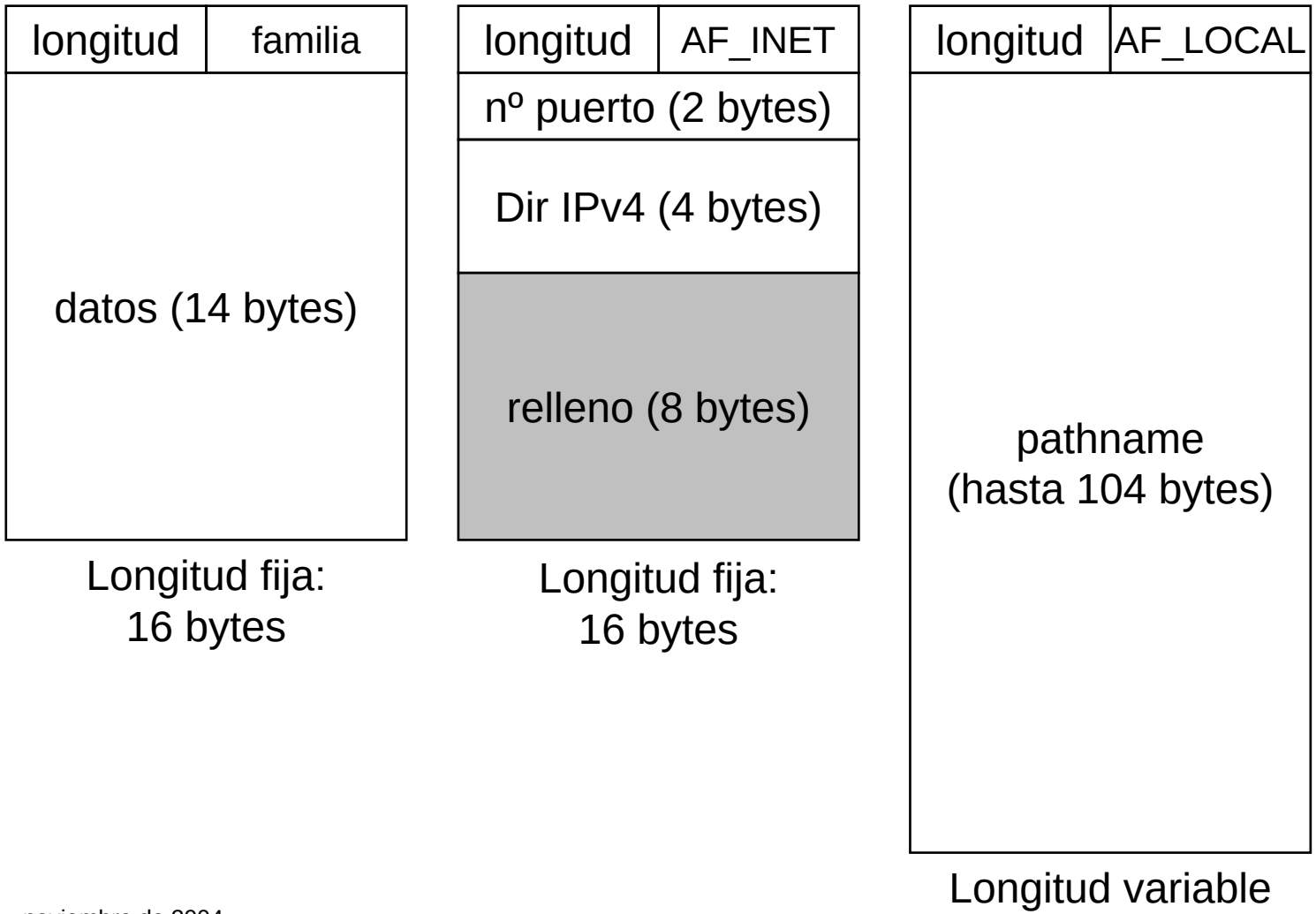
Nombres (I)

- La estructura del nombre o dirección de un socket depende de la familia a la que pertenece.
- La API utiliza una estructura genérica en la declaración de las funciones.

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

- Proporciona estructuras particularizadas para cada familia de sockets soportada.
  - struct sockaddr_in
  - struct sockaddr_un

Nombres (II)


longitud
familia
datos (14 bytes)
Longitud fija:
16 bytes
longitud
AF_INET
nº puerto (2 bytes)
Dir IPv4 (4 bytes)
relleno (8 bytes)
Longitud fija:
16 bytes
longitud
AF_LOCAL
pathname
(hasta 104 bytes)
Longitud variable

Rellenando la una dirección:

Ejemplo I:

```
sock_addr.sin_addr.s_addr = htonl(INADDR_ANY);
sock_addr.sin_port = htons(0);
sock_addr.sin_family = AF_INET;
```

Ejemplo II:

```
sock_addr.sin_addr.s_addr = inet_addr(“128.59.69.7”);
sock_addr.sin_port = htons(5115);
sock_addr.sin_family = AF_INET
```

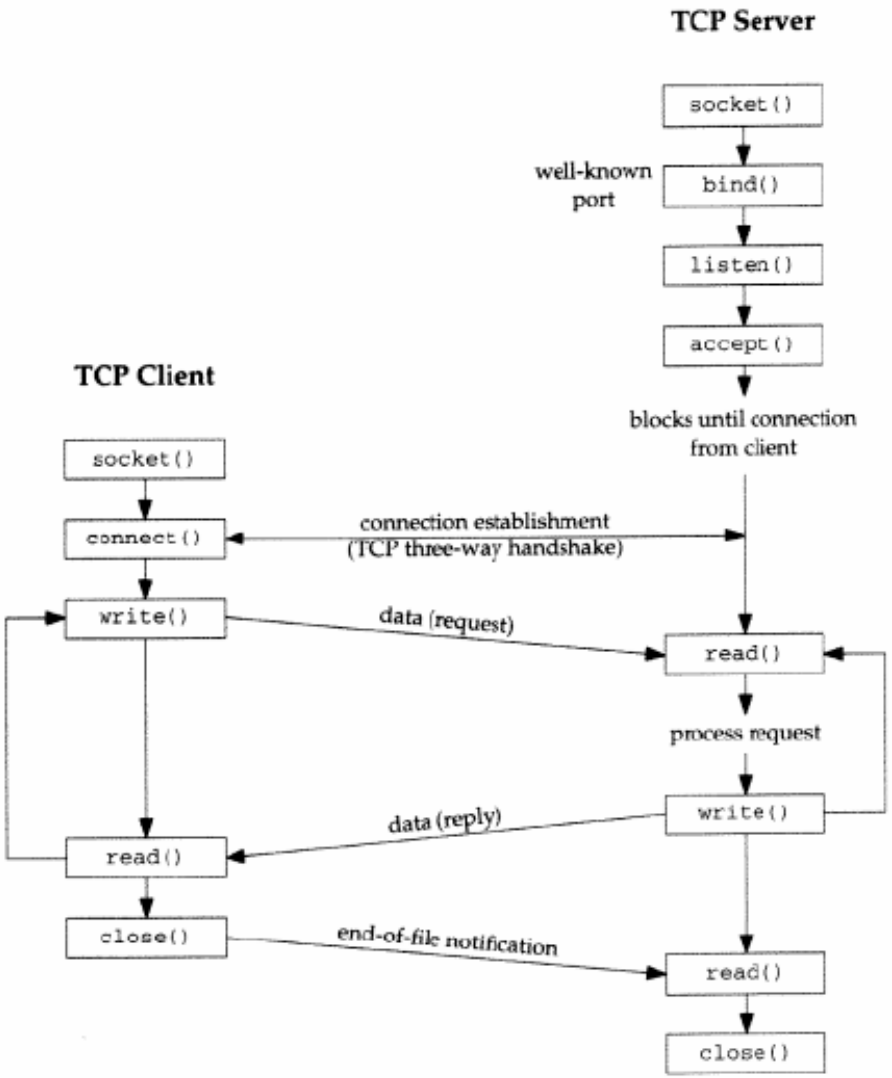


Figure 4.1 Socket functions for elementary TCP client–server.

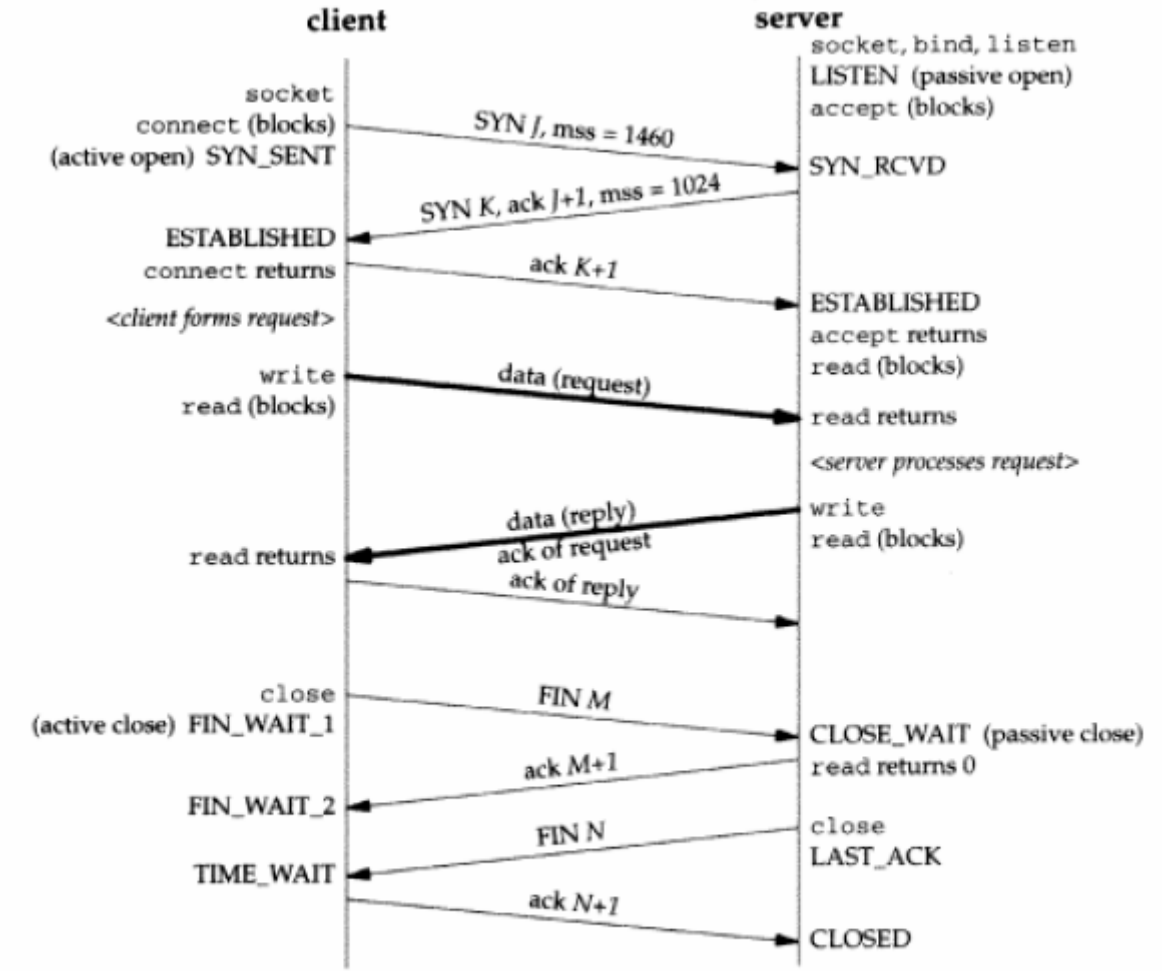


Figure 2.5 Packet exchange for TCP connection.

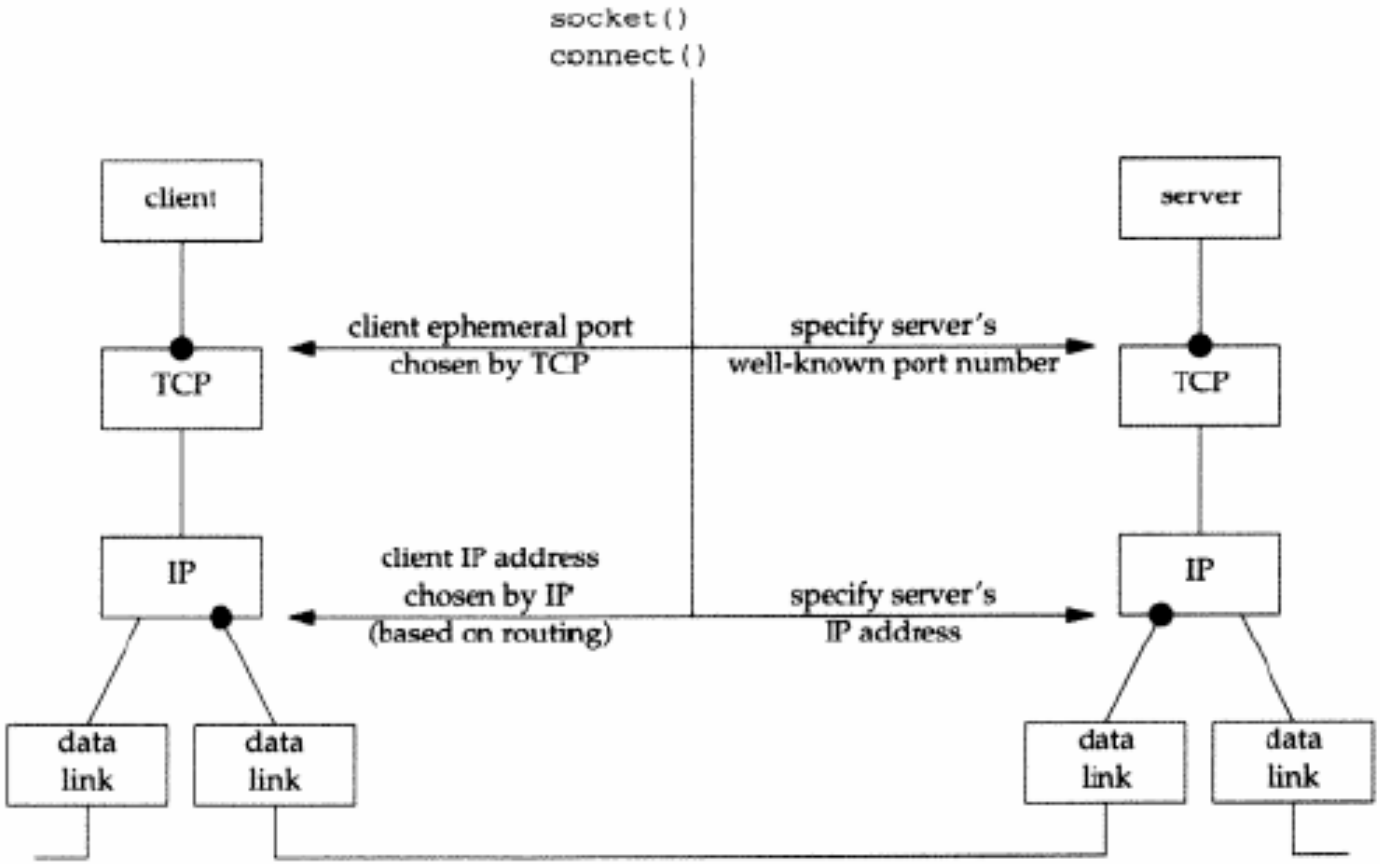


Figure 5.15 Summary of TCP client–server from client's perspective.

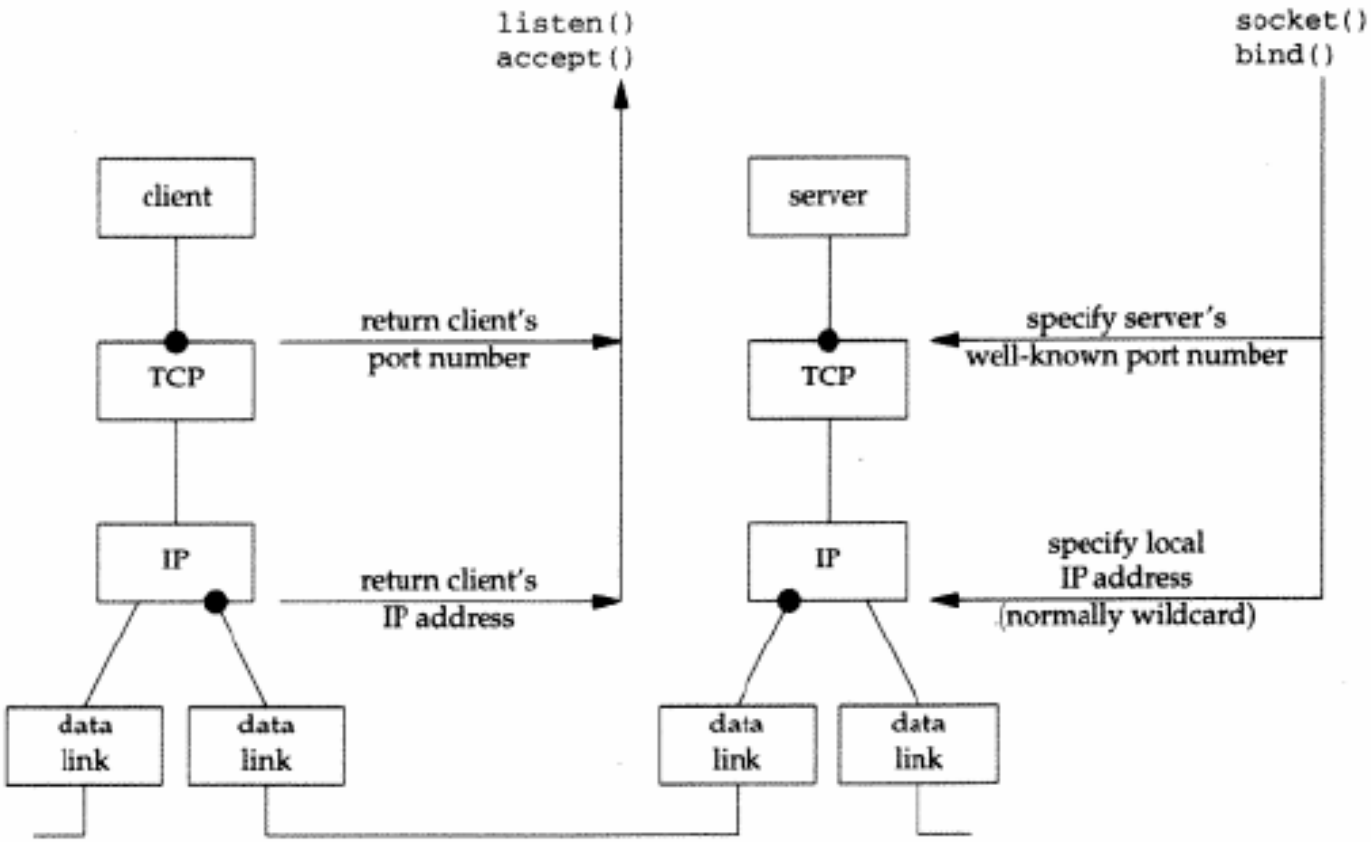


Figure 5.16 Summary of TCP client–server from server's perspective.

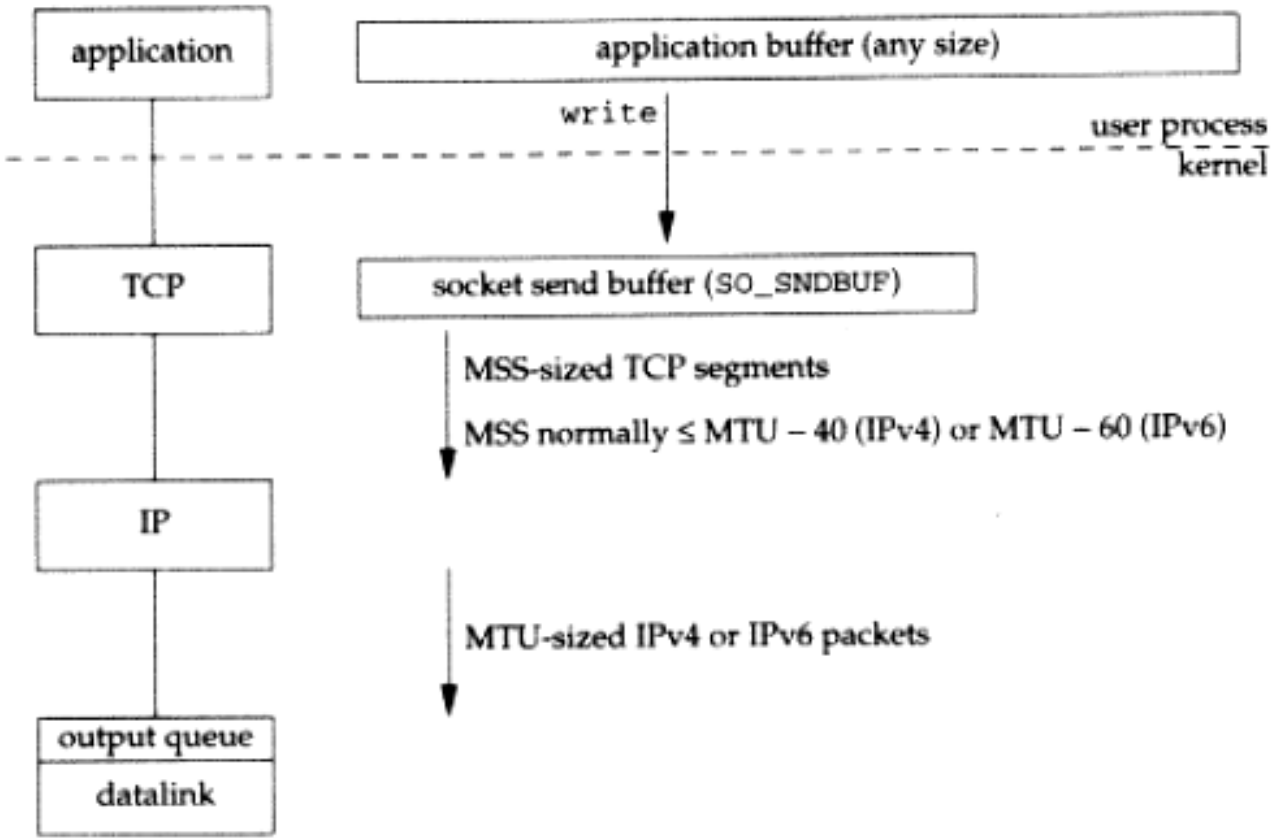


Figure 2.11 Steps and buffers involved when application writes to a TCP socket.

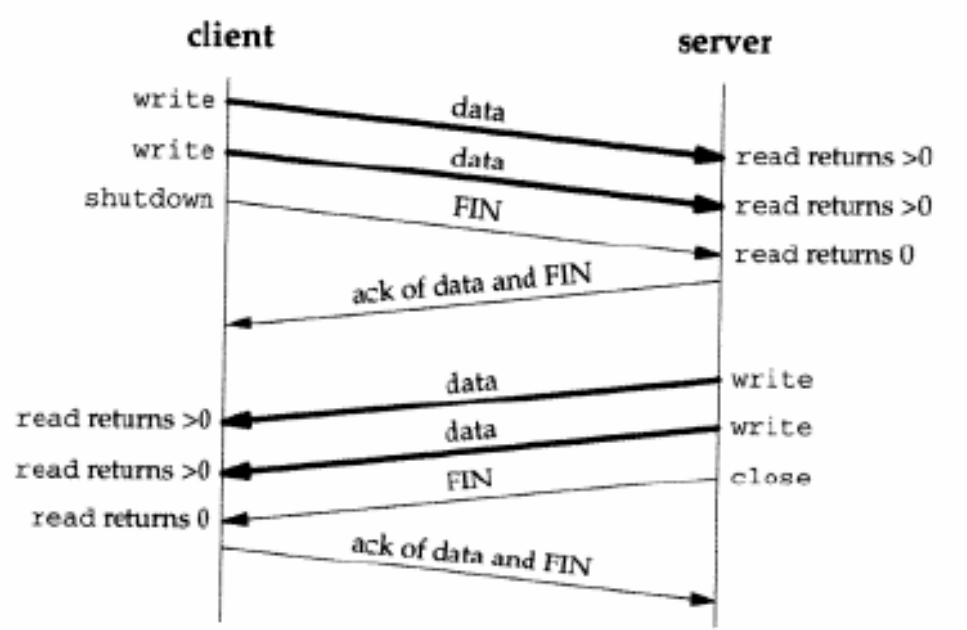


Figure 6.12 Calling `shutdown` to close half of a TCP connection.

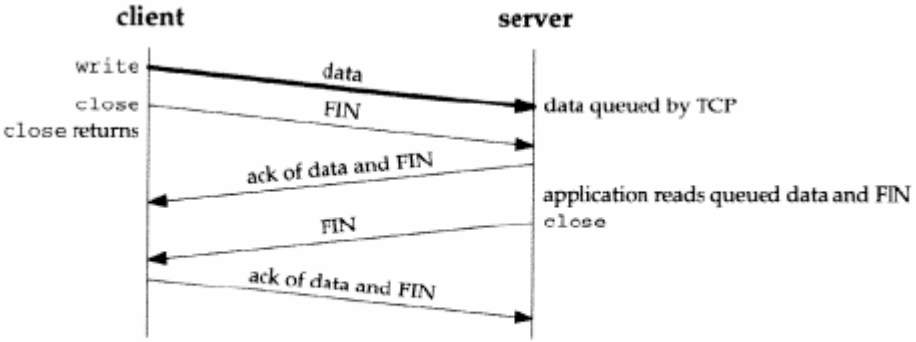


Figure 7.6 Default operation of `close`: it returns immediately.

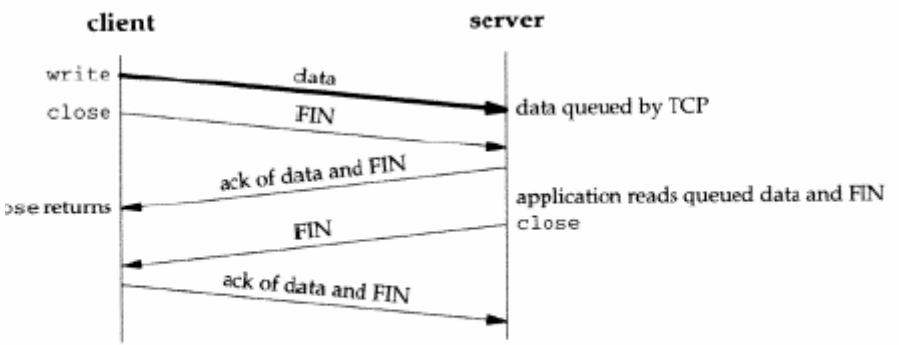


Figure 7.7 `close` with `SO_LINGER` socket option set and `l_linger` a positive value.

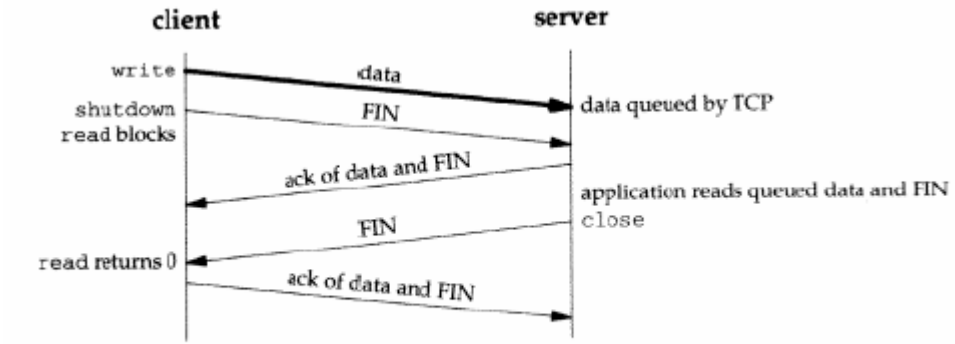


Figure 7.8 Using `shutdown` to know that peer has received our data.

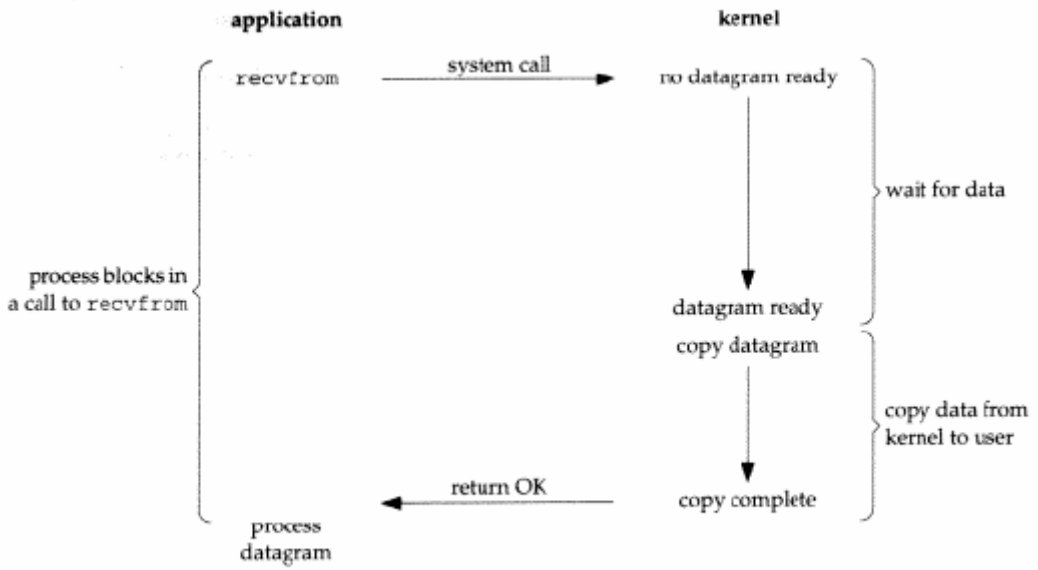


Figure 6.1 Blocking I/O model.

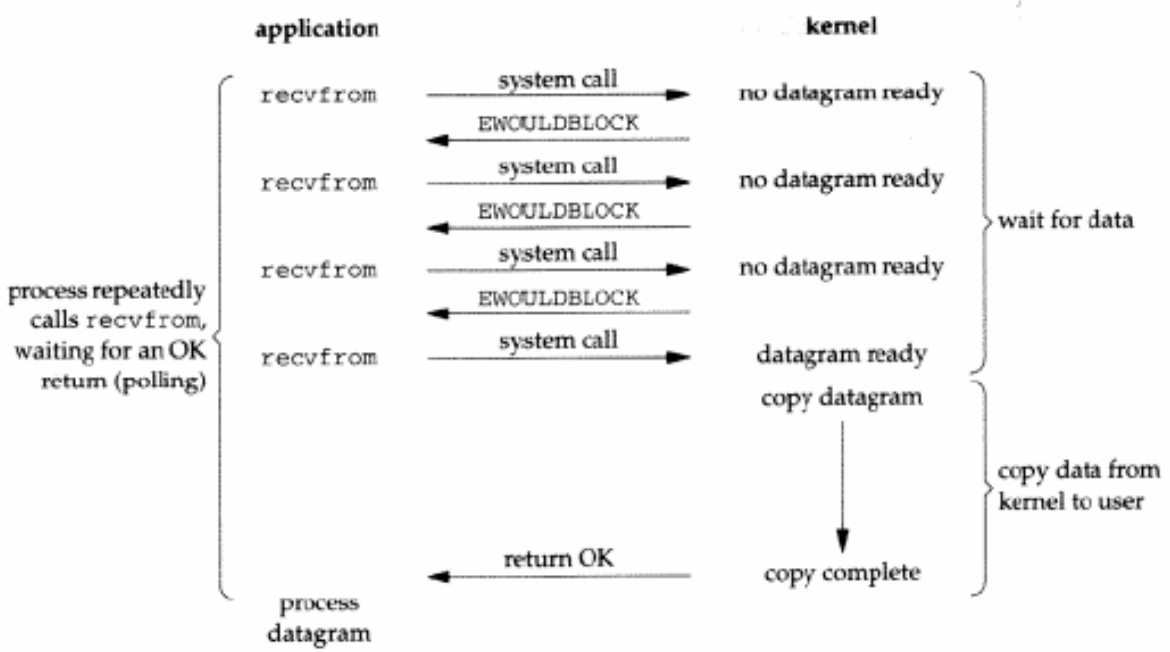


Figure 6.2 Nonblocking I/O model.

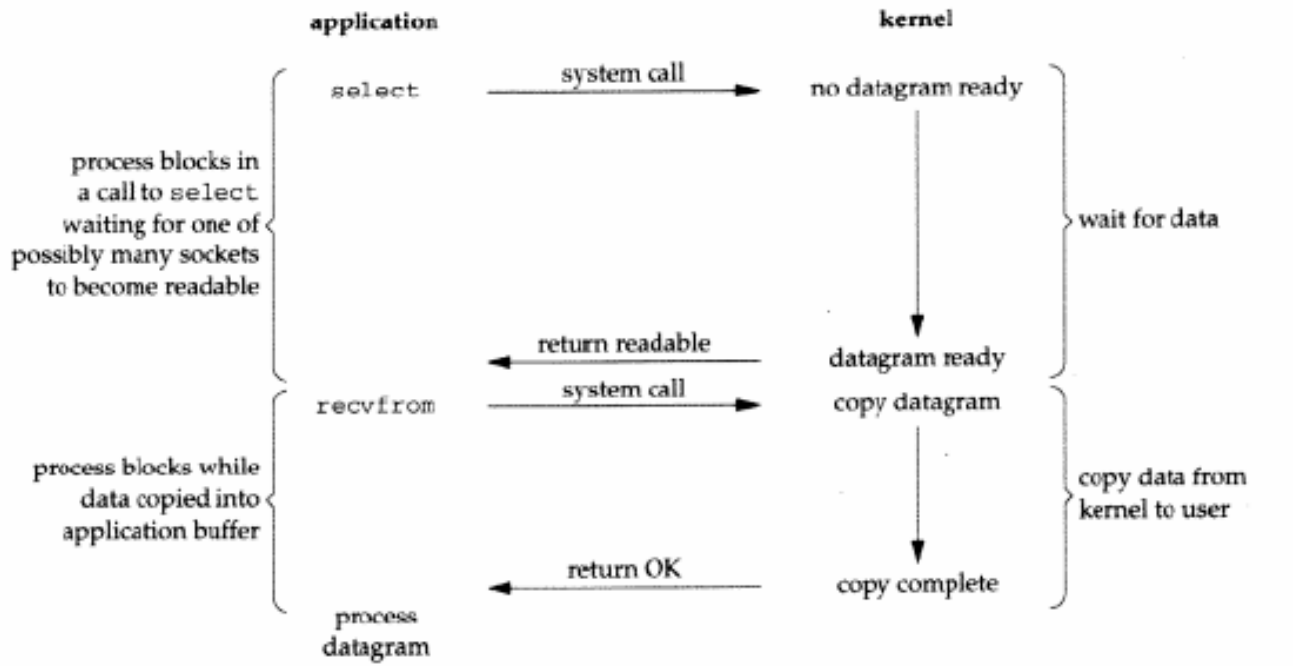


Figure 6.3 I/O multiplexing model.

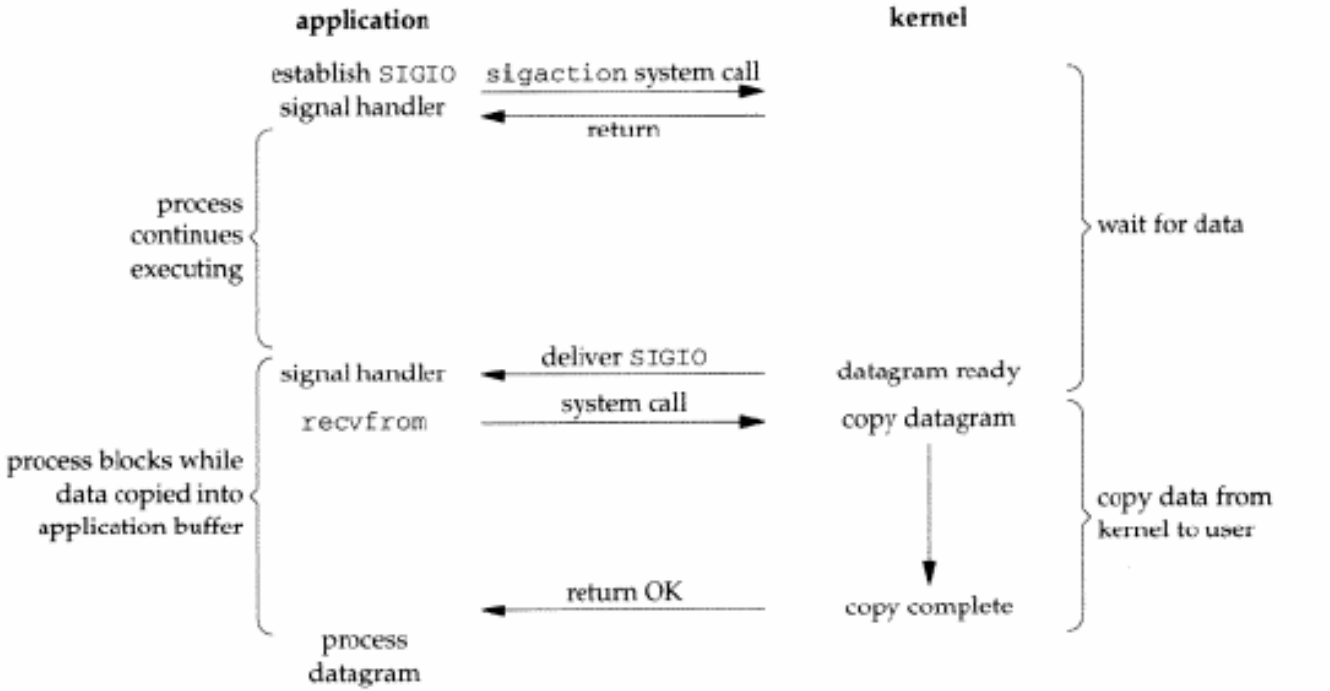


Figure 6.4 Signal Driven I/O model.

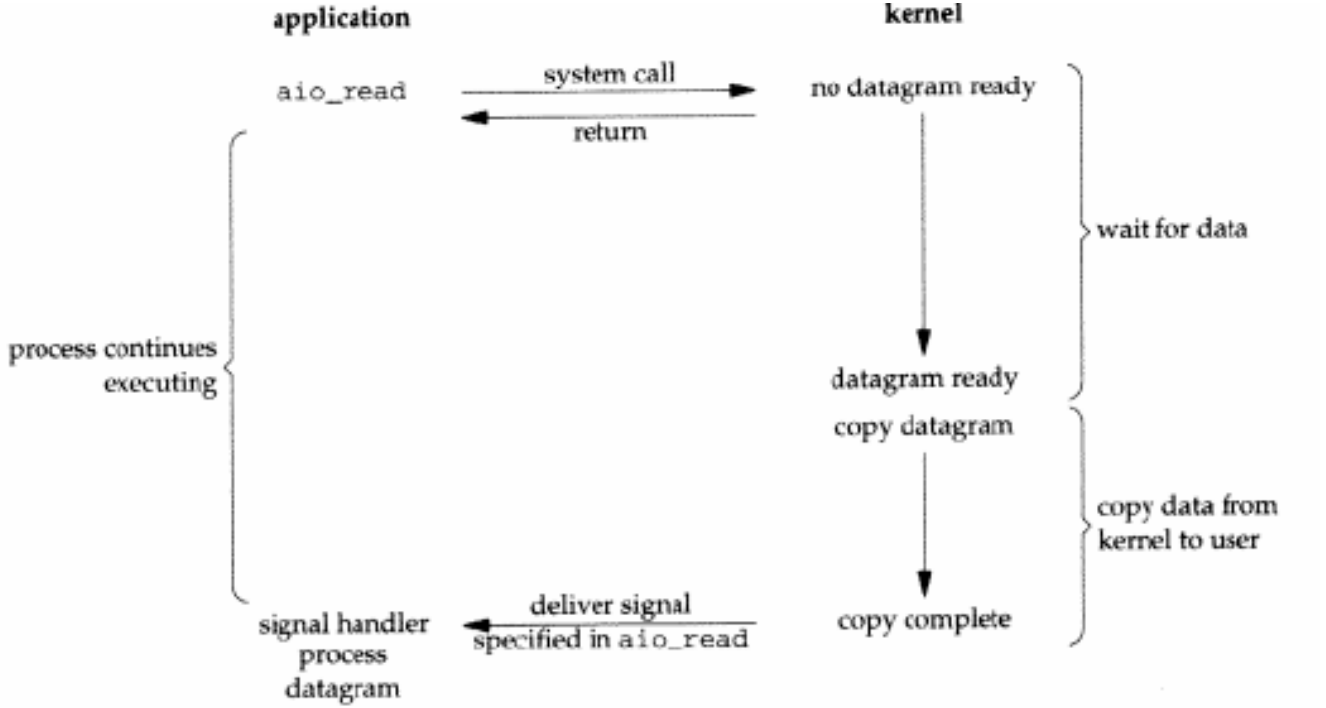


Figure 6.5 Asynchronous I/O model.

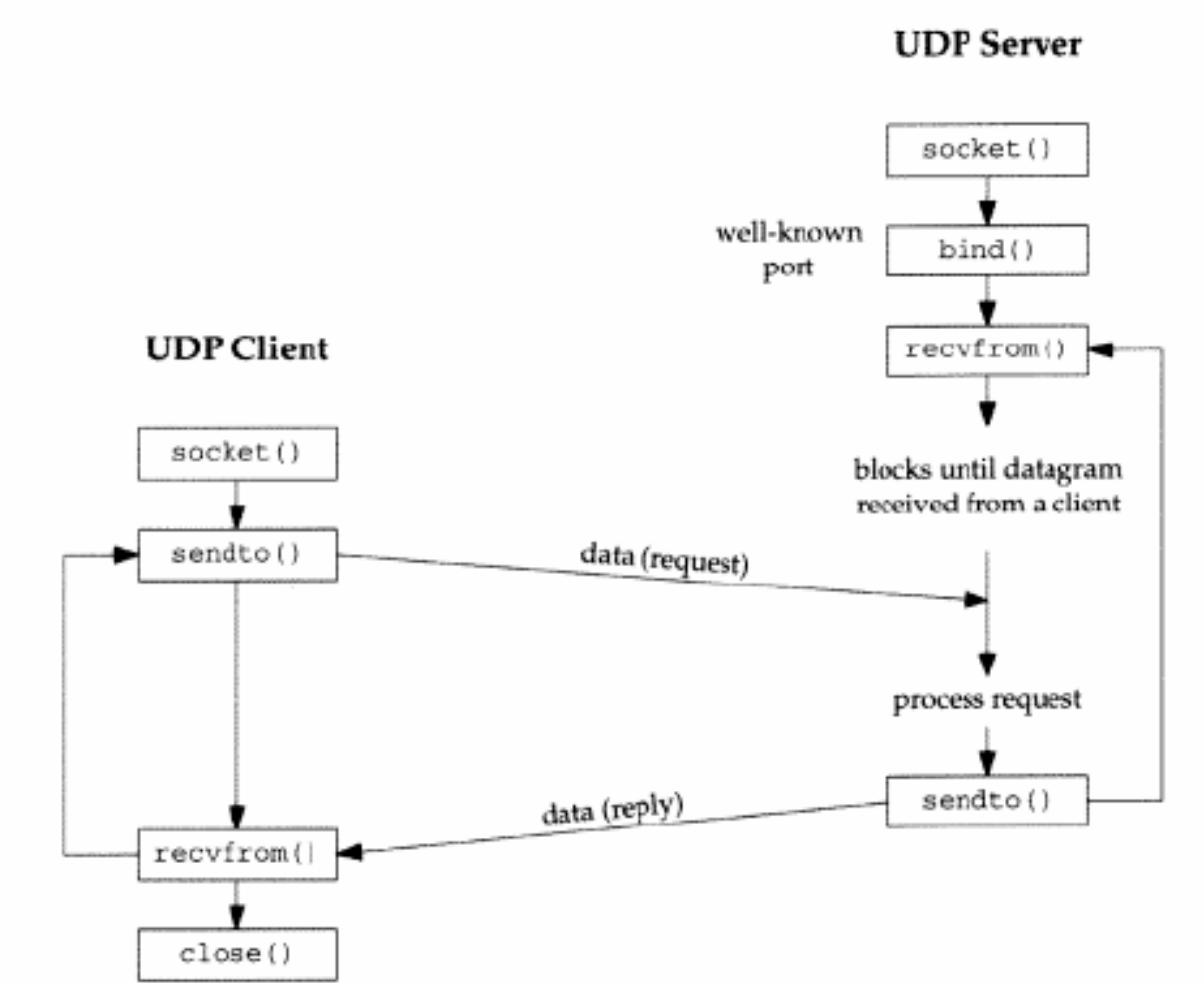


Figure 8.1 Socket functions for UDP client–server.

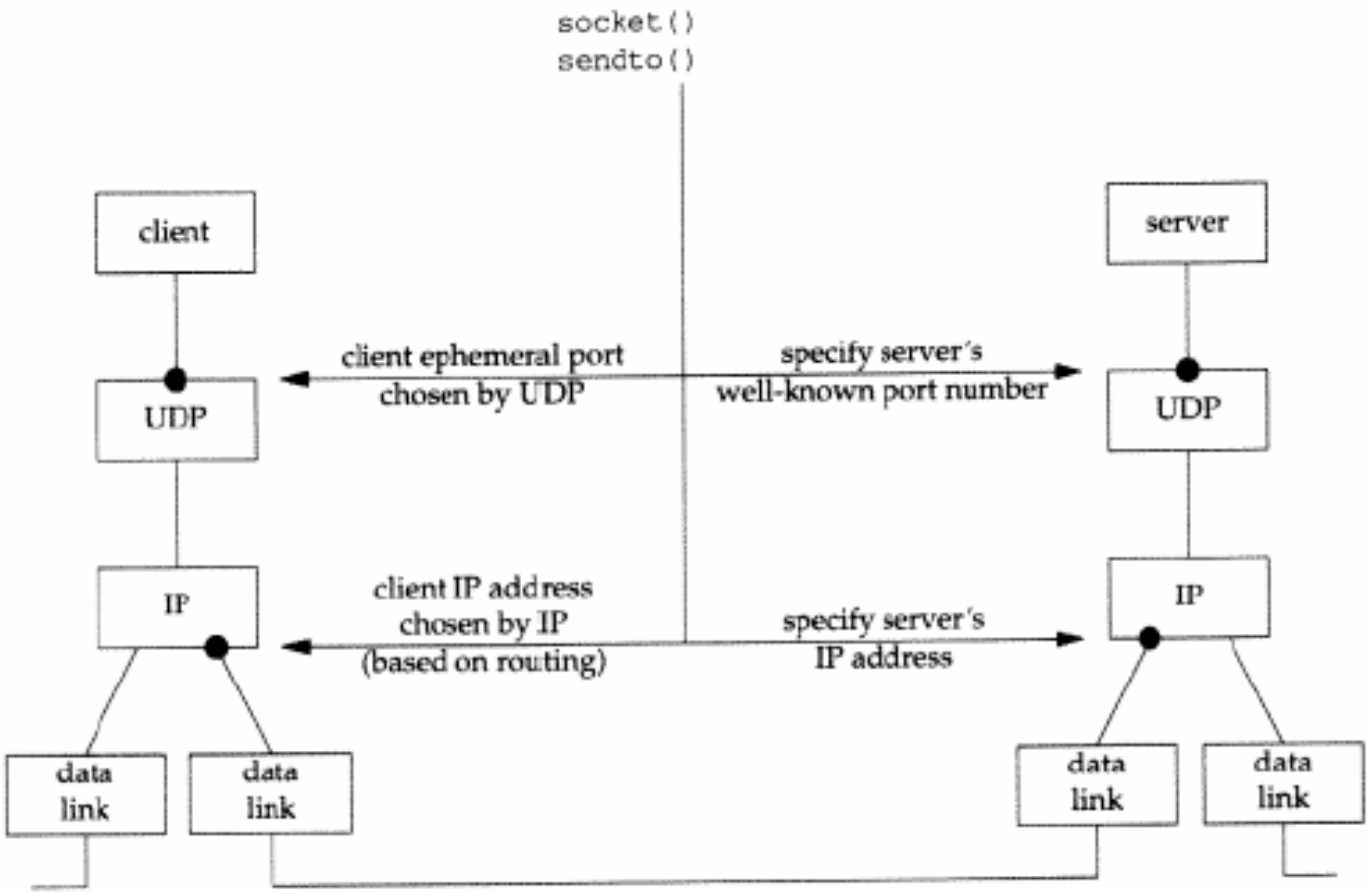


Figure 8.11 Summary of UDP client–server from client's perspective.

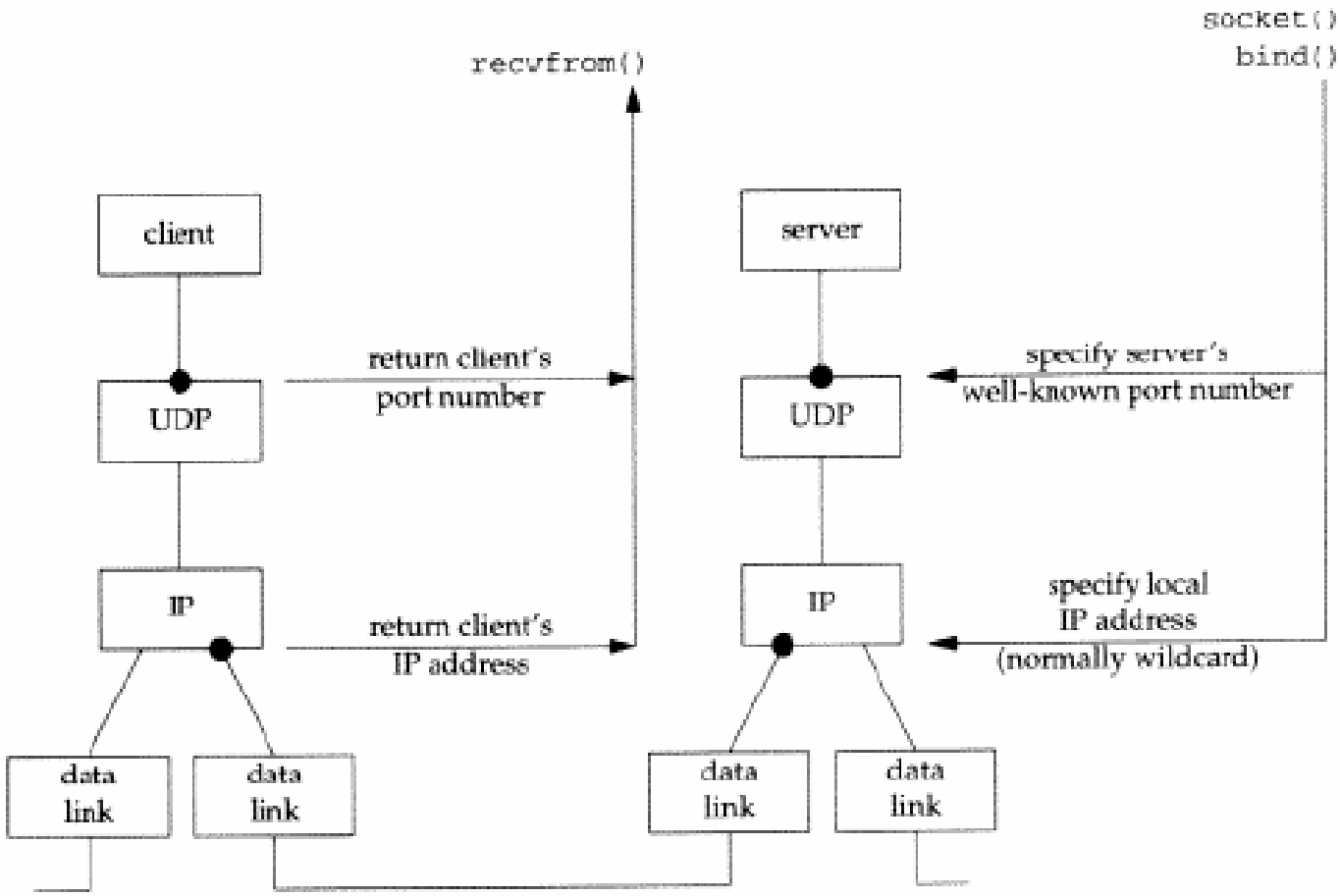


Figure 8.12 Summary of UDP client–server from server's perspective

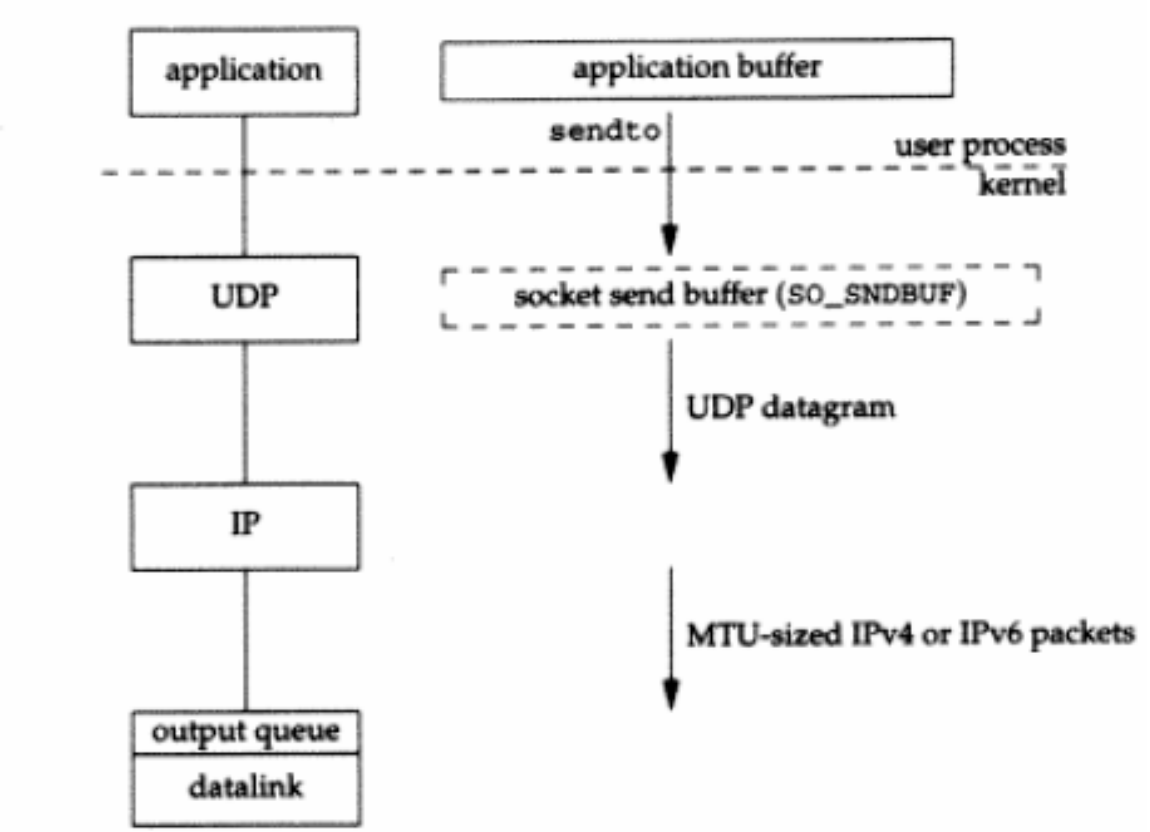


Figure 2.12 Steps and buffers involved when application writes to a UDP socket

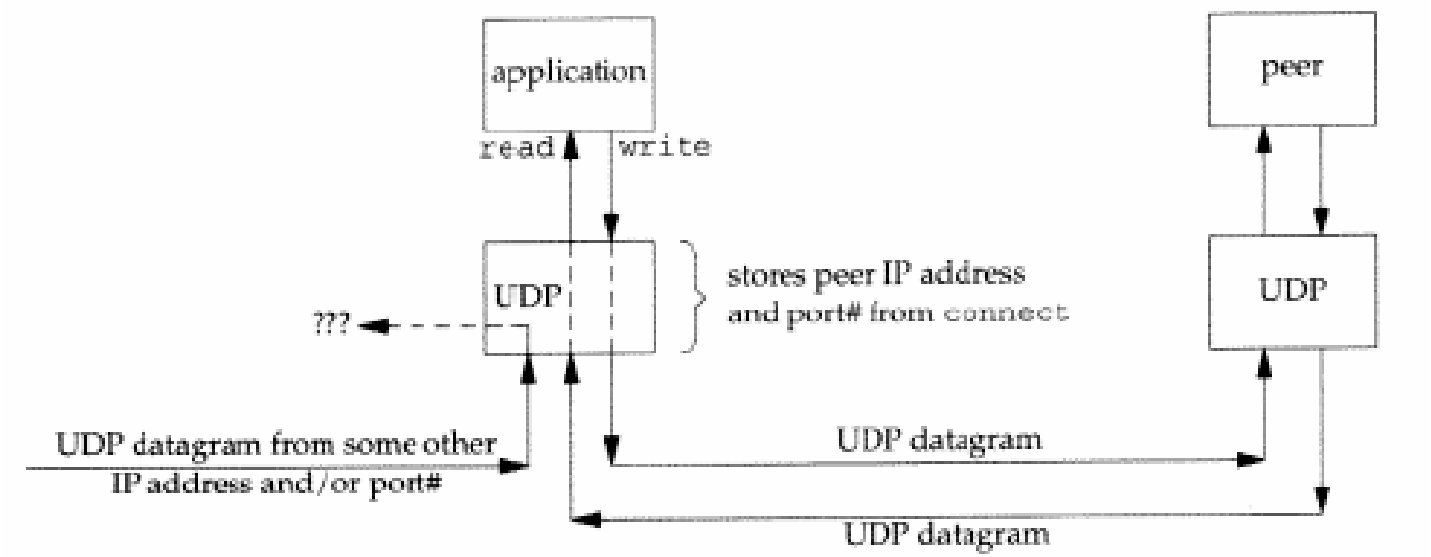


Figure 8.15 Connected UDP socket.

Programación con Windows Sockets (I)

- Tipo de dato del socket: SOCKET
  - La validez de un socket se comprueba comparando con INVALID_SOCKET

Estilo BSD:

```
s = socket(...);
if (s == -1)   /* or s < 0 */
{...}
```

Estilo recomendado:

```
s = socket(...);
if (s == INVALID_SOCKET)
{...}
```

Programación con Windows Sockets (II)

- Función Select() y FD_*
  - La sintaxis y funcionalidad se mantiene, aunque:
    - Sólo es aplicable a sockets.
    - El número máximo de descriptores a comprobar no se emplea
  - Los conjuntos de sockets se representan con el tipo fd_set, aunque se implementan como un array de sockets.
  - Para evitar problemas de compatibilidad se recomienda emplear las macros FD_XXX para:
    - Activar
    - Inicializar
    - Borrar
    - Comprobar

    dichos conjuntos.

Programación con Windows Sockets (III)

- Códigos de Error
  - La causa de un error no está disponible a través de la variable errno.
  - Para acceder a los códigos de error debe usarse la función WSAGetLastError().
  - Para mantener compatibilidad a nivel de código con BSD, se puede hacer la siguiente definición:

```
En Windows:
    #define errno WSAGetLastError()
En UNIX:
 int WSAGetLastError() {return errno}
```

Programación con Windows Sockets (IV)

- Códigos de Error (cont.)

Estilo BSD:

```
r = recv(...);
if (r == -1 && errno == EWOULDBLOCK)
{...}
```

Estilo recomendado:

```
r = recv(...);
if (r == -1 && WSAGetLastError()== EWOULDBLOCK)
{...}
```

Programación con Windows Sockets (V)

- Cambio de nombre en funciones:
  - En dos casos ha sido necesario cambiar el nombre de las funciones usadas por la librería Berkeley Sockets:
    - close() ⇔ closesocket()
    - ioct() ⇔ ioctlsocket() / WSAIoctl()
- Número máximo de sockets soportados
  - El número máximo de sockets que puede usar una determinada aplicación depende de la implementación de Winsock empleada y se conoce a través de WSAStartUp()
  - Además existe la constante FD_SETSIZE que fija el tamaño máximo de las estructuras fd_set. Por defecto es 64.

Programación con Windows Sockets (VI)

- Fichero de inclusión:
  - Winsock2.h
- Comprobación del fallo de una función:

Estilo BSD:

```
r = recv(...);
if (r == -1 && errno == EWOULDBLOCK)
{...}
```

Estilo recomendado:

```
r = recv(...);
if (r == SOCKET_ERROR && WSAGetLastError()==
EWOULDBLOCK)
{...}
```

Programación con Windows Sockets (VII)

- Orden de los bytes:
  - El orden de los bytes en la arquitectura Intel es diferente del empleado por la red.
  - Cualquier referencia a direcciones IP o números de puerto pasadas desde o hacia una función WinSock debe estar en orden de red.
- Inicialización y terminación de los programas:
  - Es necesario usar la función WSAStartup() antes de hacer ninguna llamada a las funciones de sockets para inicializar la DLL compartida (winsock.dll)
  - Es necesario ejecutar la función WSACleanup() al finalizar para liberar los recursos ocupados en la DLL