


Sockets orientados a conexiones no bloqueantes

Lechuga Canales Hector Jair
Calderon Zimbron Luis Ángel
Añorve Gómez Axel Israel






Las conexiones no bloqueantes son aquellas en las que las operaciones de lectura y escritura no esperan indefinidamente a que los datos estén disponibles o se envíen completamente. En lugar de eso, el programa puede realizar otras tareas mientras espera la disponibilidad de datos en el socket. Esto se logra mediante el uso de funciones o métodos que no bloquean el hilo de ejecución principal.

Introduccion

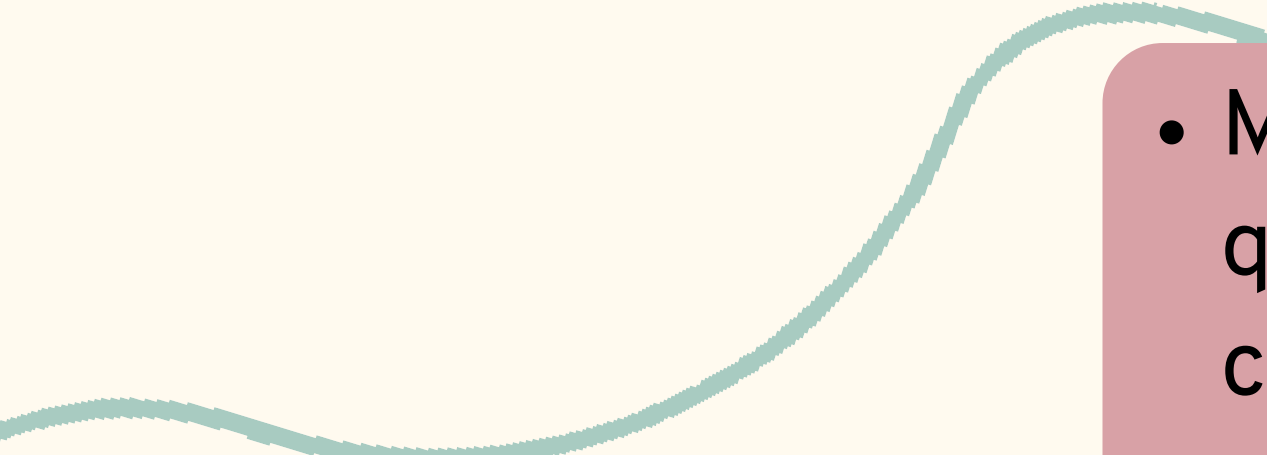
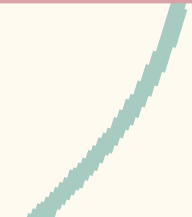


Uso y Ventajas

- Multiplexación: Puedes manejar múltiples conexiones de red a través de un único proceso o hilo, usando técnicas como `select()`, `poll()` o `epoll()` en Linux. Estas funciones permiten que tu programa monitoree múltiples sockets a la vez para ver si están listos para ser leídos o escritos.



Eficiencia: Al evitar el bloqueo, tu aplicación puede realizar otras tareas mientras espera que los datos de red estén listos. Esto es especialmente útil en aplicaciones de servidor donde múltiples clientes pueden conectarse simultáneamente, como servidores web, juegos en línea o aplicaciones de chat.

- 
- Mejor Escalabilidad: Los sockets no bloqueantes permiten que los servidores manejen un mayor número de conexiones simultáneamente, ya que un solo hilo puede manejar muchas conexiones sin quedarse esperando operaciones de I/O de ninguna de ellas.
- 

Desarrollo

Por defecto, los sockets son bloqueantes, lo que significa que si una llamada de socket no puede completarse de inmediato, el proceso se duerme esperando que la condición necesaria se cumpla.

Las llamadas a sockets que pueden bloquear se dividen en cuatro categorías principales:

- Operaciones de entrada
- Operaciones de salida

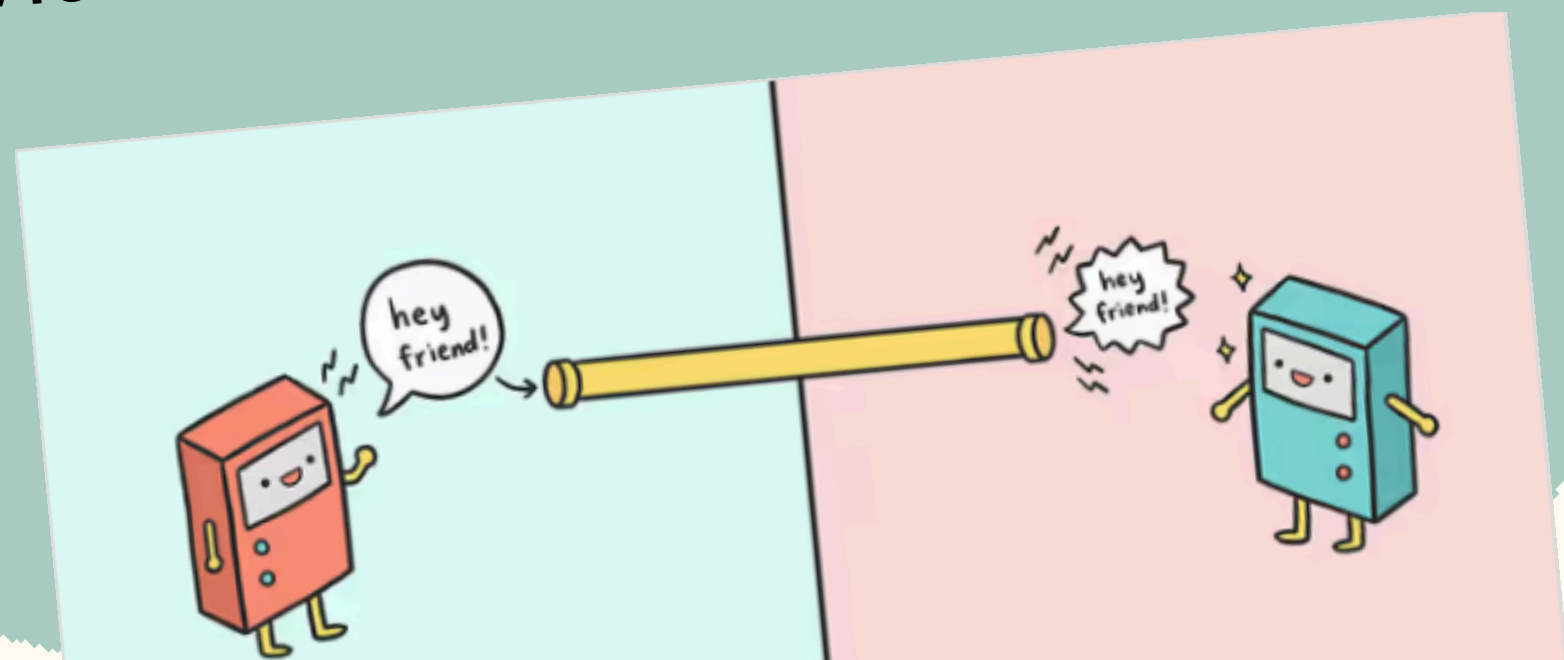
Operaciones de entrada

En el caso de un socket TCP bloqueante, si no hay datos disponibles en el buffer de recepción del socket, el proceso se suspenderá hasta que llegue algún dato. Para TCP, que es un protocolo de flujo de bytes, el proceso se reanuda cuando llega "algún" dato, que puede ser un solo byte o un segmento completo de TCP. En el caso de UDP, que usa datagramas, el proceso se suspende hasta que llegue un datagrama completo.

Tiene funciones como: read, readv, recv, recvfrom y recvmsg

También con un socket no bloqueante, si la operación de entrada no puede satisfacerse de inmediato (al menos un byte de datos para un socket TCP o un datagrama completo para un socket UDP), retorna inmediatamente con un error de EWOULDBLOCK. 435.

Este error indica que una operación en un socket no se puede realizar de manera inmediata porque haría que el proceso se bloqueara, lo cual no es deseado si el socket se ha configurado en modo no bloqueante. En otras palabras, EWOULDBLOCK se devuelve cuando una operación de red, como leer o escribir datos, no puede completarse inmediatamente sin esperar a que otros datos lleguen o sean enviados.



Operaciones de salida

Para un socket TCP, los datos de la aplicación se copian en el buffer de envío del socket. Si no hay espacio en el buffer de envío de un socket bloqueante, el proceso se suspende hasta que haya espacio. Con un socket TCP no bloqueante, si no hay espacio en el buffer de envío, se retorna inmediatamente con un error de EWOULDBLOCK. Si hay algo de espacio, el valor de retorno será el número de bytes que el kernel pudo copiar en el buffer.

Operaciones de salida

En el caso de UDP, no hay un buffer de envío de socket UDP como tal; el kernel simplemente copia los datos de la aplicación y los mueve hacia abajo en la pila, añadiendo las cabeceras UDP e IP. Por lo tanto, una operación de salida en un socket UDP bloqueante no se bloqueará por la misma razón que un socket TCP, pero es posible que las operaciones de salida se bloqueen en algunos sistemas debido al buffering y control de flujo que ocurre dentro del código de red en el kernel.⁴³⁵

Incluye funciones como ``write``, ``writev``, ``send``, ``sendto``, y ``sendmsg``.



Para un socket UDP, no existe un "buffer de envío" real dentro del socket como ocurre con los sockets TCP.

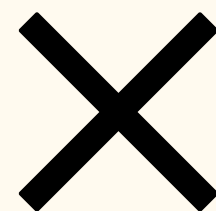
Lo que sucede es que el núcleo del sistema operativo simplemente toma los datos de la aplicación y los mueve hacia abajo en la pila de protocolos, añadiendo las cabeceras de UDP e IP antes de proceder al envío. Esto significa que, bajo circunstancias normales, una operación de salida en un socket UDP bloqueante no debería bloquearse esperando que haya espacio.⁴³⁶



Aceptación de Conexiones Entrantes



Se aborda la función de aceptación de conexiones entrantes en un servidor.



En un socket bloqueante, si no hay nuevas conexiones disponibles, el proceso se suspende.



Se muestra cómo en un socket sin bloqueo, se devuelve un error de `EWOULDBLOCK` si no hay nuevas conexiones disponibles.



Inicio de Conexiones Salientes

The diagram illustrates the three steps of a TCP connection establishment process. On the left, a large blue circle is partially visible. A light green wavy line connects the three steps, which are each contained within a pink rounded rectangle. The steps are: 1. Initialization of the connection, 2. Server response, and 3. Client confirmation.

Inicialización de la conexión

El cliente envía una solicitud de conexión al servidor TCP. Esto se hace enviando un paquete SYN (Synchronize) al servidor. Este paquete contiene información como los números de secuencia iniciales y las opciones de configuración.

Respuesta del servidor

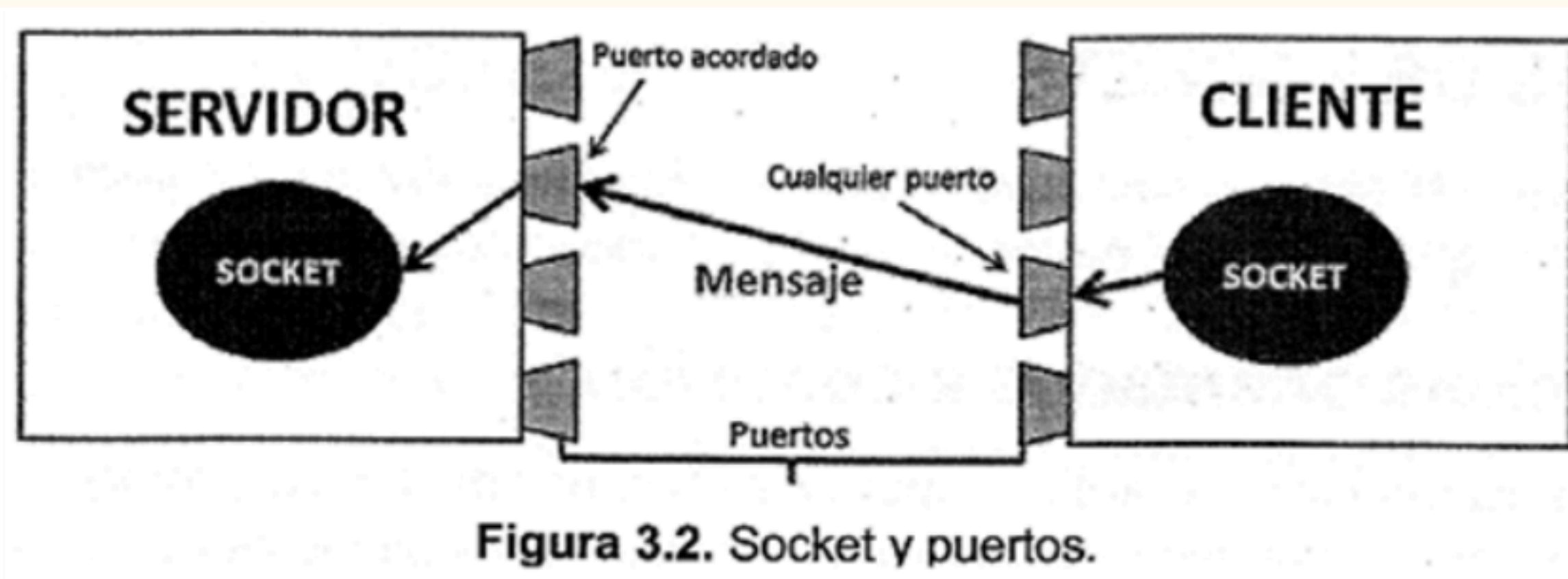
El servidor recibe el paquete SYN y responde con un paquete SYN-ACK (Synchronize-Acknowledgement). Este paquete confirma la solicitud de conexión y también indica que el servidor está dispuesto a establecer la conexión. El número de secuencia de este paquete es incrementado por uno.

Confirmación del cliente

El cliente recibe el paquete SYN-ACK y envía un paquete ACK (Acknowledgement) de vuelta al servidor. Este paquete confirma la recepción del paquete SYN-ACK y completa el proceso de establecimiento de la conexión.

Como funcionan?

El funcionamiento de los sockets no bloqueantes se basa en el uso de llamadas al sistema o funciones que permiten a la aplicación verificar si un socket tiene datos listos para ser leídos o si se puede escribir en él sin bloquear el hilo de ejecución principal.





- `select()` y `poll()` son funciones que permiten a la aplicación monitorear múltiples sockets y esperar hasta que uno o más de ellos estén listos para realizar operaciones de lectura o escritura.
- Estas funciones devuelven inmediatamente cuando al menos uno de los sockets está listo, evitando bloquear el hilo de ejecución.
- `select()` es una función más antigua y presenta limitaciones en cuanto al número de sockets que puede manejar eficientemente, mientras que `poll()` proporciona una solución más escalable.

Funciones



- `epoll()` es una interfaz más reciente y eficiente para manejar eventos de entrada/salida (E/S) en sistemas Linux.
- Utiliza una estructura de datos más eficiente para almacenar y manejar grandes cantidades de descriptores de archivos (incluidos los sockets).
- Proporciona un mejor rendimiento en comparación con `select()` y `poll()` para aplicaciones con un gran número de conexiones.

Conclusión

En conclusión, los Sockets orientados a conexiones no bloqueantes ofrecen una solución flexible y eficiente para el manejo de comunicaciones en entornos donde la capacidad de respuesta y la escalabilidad son críticas. Al permitir que múltiples conexiones se manejen de manera concurrente sin bloquear el hilo principal de ejecución, estos Sockets ofrecen la capacidad de gestionar cargas de trabajo pesadas y responder de manera eficiente a eventos simultáneos.

Referencias

01

Stevens, W. Richard, Fenner, Bill, & Rudo, Andrew M. (2004).
"UNIX Network Programming: The Sockets Networking API"
(3rd ed.). Addison-Wesley Professional.
Páginas 436-437.