GEOLOCALIZADOR EN TIEMPO REAL

Control mediante Real-Time Linux

Jose Ángel Gumiel

Índice

Introducción

Se ha decidido implementar una aplicación distribuida (cliente-servidor) que permita obtener las coordenadas de un dispositivo móvil y enviarlas a un servidor en tiempo real. Para ello se han usado sockets UDP, no orientados a la conexión, lo que permite la concurrencia con un único proceso.

Para el servidor se utiliza una Raspberry Pi con sistema operativo Raspbian, al que se le ha modificado el Kernel para incluir un parche de RT Preempt. Este cambio permite transformar Linux en un sistema operativo de tiempo real.

Como el servidor va a estar dedicado exclusivamente a responder a las peticiones de los clientes, se desea que el programa tenga prioridad sobre otros procesos que pudieran estar corriendo en segundo plano.

Protocolo

Cuando se desarrolla una aplicación distribuida se pretende comunicar un cliente con un servidor. Ambos tienen que tener un mismo lenguaje para entenderse, es decir, un protocolo de comunicación.

Todos los comandos y sus respuestas serán cadenas de caracteres ASCII. Los comandos serán de 3 caracteres y las respuestas de 2. Los parámetros de latitud (LAT) y longitud (LON) se envían utilizando el estándar IEEE 754, que es el estándar de la coma flotante. Esto es debido a que ocupa menos bytes que si se enviara una cadena de caracteres. Al ser una comunicación, cuanto menor sea el tamaño de datos a enviar, más rápida y eficiente será.

El cliente enviará su ID asociada en cada comando que mande, ya que se asume que cada petición será un servicio inmediato sin establecer ningún tipo de sesión.

Los comandos irán seguidos de los parámetros con un espacio como separación entre cada uno de los argumentos.

Comandos:

Comando	Parámetros	Descripción/Semántica
SAV	LAT + LON + ID	Orden de guardar la posición junto con la ID.
LST	ID + N	Solicita las N últimas posiciones guardadas del dispositivo con el ID establecido. N está limitado a un máximo de 10.
LSL	ID	Solicita la última posición almacenada con el ID establecido.
RST	ID	Vacía toda la lista de posiciones guardadas.

Respuestas:

Comando	Parámetros	Descripción/Semántica
OK	LAT + LON + FECHA [1..N] o nada	Respuesta positiva. Todo ha ido correctamente.
ERR	Código de error	Respuesta negativa. Ha habido algún problema.

Procedimientos:

En cualquier momento el servidor está esperando a recibir comandos, si el comando es desconocido el servidor responderá con un código de error 1.

Solicitud de almacenamiento de coordenadas.

El dispositivo enviará el comando SAV seguido de la latitud (LAT), longitud (LON) y el ID del usuario. En caso de que se envíe y almacene correctamente devolverá una respuesta positiva (OK). En caso de error de almacenamiento el servidor responderá negativamente con un error. En el caso de que fuera el ID el incorrecto responderá con un error. No hay que comprobar que la latitud y la longitud mandada sean correctas ya que las manda el dispositivo y suponemos que no cometerá ningún error.

Solicitud de listado de coordenadas.

El dispositivo envía el comando LST con la ID y el número N que es el número de posiciones que desea que el servidor le devuelva. El dispositivo responderá positivamente si no ha habido ningún problema seguido de la latitud (LAT), longitud (LON). Hemos confirmado que todo esto cabe usando paquetes UDP. Si el ID fuera incorrecto o desconocido el servidor respondería negativamente (ER) con un error. Si hubiera algún problema en leer de la base de datos este devolvería la respuesta negativa con un error.

Solicitud de última coordenada.

El dispositivo envía el comando LSL (LiSt Last) con la ID del dispositivo. Este responderá positivamente (OK) con la última posición guardada en forma de latitud (LAT), longitud (LON). Si el ID fuera incorrecto devolvería un error y si hubiera habido un problema al leer de la base de dato respondería con otro error.

Solicitud de borrado de la lista de posiciones.

El dispositivo envía el comando RST seguido del ID del dispositivo para borrar la lista de posiciones guardadas hasta ahora, algo así como para reiniciar la cuenta/dispositivo a la hora de por ejemplo, revender el dispositivo. Si la lista se borra y no ha habido ningún inconveniente, el servidor responderá con un OK. En el caso de que hubiera un error borrando la base de datos se devolvería una respuesta negativa con error referente a problemas con la base de datos. En caso de que el ID del dispositivo fuera incorrecto o desconocido se respondería con un código de error 3.

Aplicación

Como se ha anteriormente, se trata de una aplicación para el registro de geoposicionamiento. Es una aplicación distribuida que conecta N clientes con un servidor de forma remota a través de la interfaz de sockets. Además está programada para que tenga una prioridad y pueda responder en tiempo real a las peticiones que le llegan al servidor.

La aplicación permite realizar las siguientes acciones:

1. Guardar la localización actual de un usuario.
2. Mostrar las últimas localizaciones.
3. Mostrar la última localización.
4. Resetear la lista de posiciones almacenadas.
5. Salir de la aplicación.

Funcionamiento de la aplicación

El cliente no está programado en tiempo real, ya que en un principio se desconoce si el dispositivo en el que se ejecute va a tener un kernel RT o no. De todas formas, será la primera parte a explicar, ya que así será más fácil de comprender la estructura del servidor.

En este caso el cliente se ha hecho para probar la comunicación y poder efectuar simulaciones en un entorno controlado.

Para ejecutar el cliente, primero tiene que haber un servidor en ejecución. En esta prueba se está trabajando dentro de la misma máquina, así que la comunicación será en localhost (127.0.0.1). La ejecución del cliente tiene que ir asociada a la IP del servidor, por lo que está parametrizada.

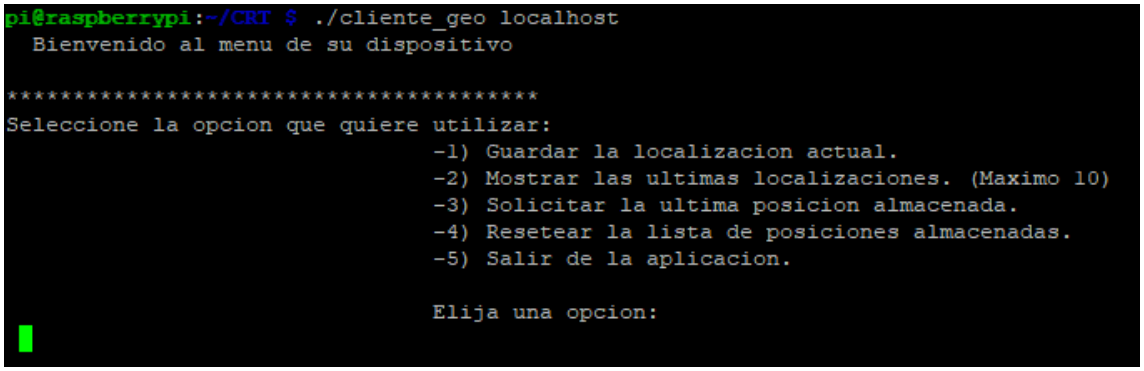

Imagen 1. Menú principal del cliente. Se puede elegir una de las 5 opciones.

Menú principal:

El menú principal permite mostrar al usuario las acciones que podrá realizar. El usuario deberá seleccionar una de las opciones disponibles mediante números. Se ha limitado el rango entre 1 y 5. En el caso de introducir un valor erróneo o no aceptado por la aplicación se mostrará un mensaje y se instará a que el usuario inserte un valor válido. En la Imagen 1 se puede ver su apariencia y estructura.

Guardar la localización actual:

En esta opción el cliente tiene que enviar al servidor la localización en la que se encuentra para que la almacene. El resultado en la aplicación se representa de esta forma:

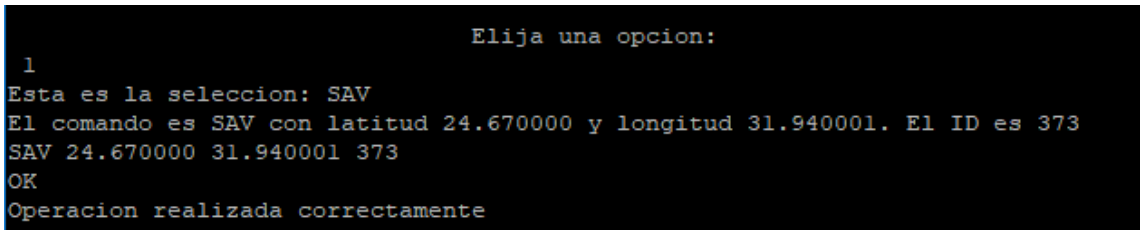

Imagen 2. Se ha elegido la opción 1. Añade una localización.

A modo de prueba, se imprime la información del cliente y el comando que se prepara para ser enviado al servidor. En este caso se recibe la respuesta “OK” del servidor, que es tratada en el cliente, confirmando así que la operación ha sido satisfactoria.

Mostrar las últimas localizaciones:

Permite al usuario de la aplicación ver cuáles han sido las últimas localizaciones que ha guardado en el sistema. La aplicación cliente solicitará un número comprendido entre 1 y 10 para que se muestren esas últimas localizaciones.

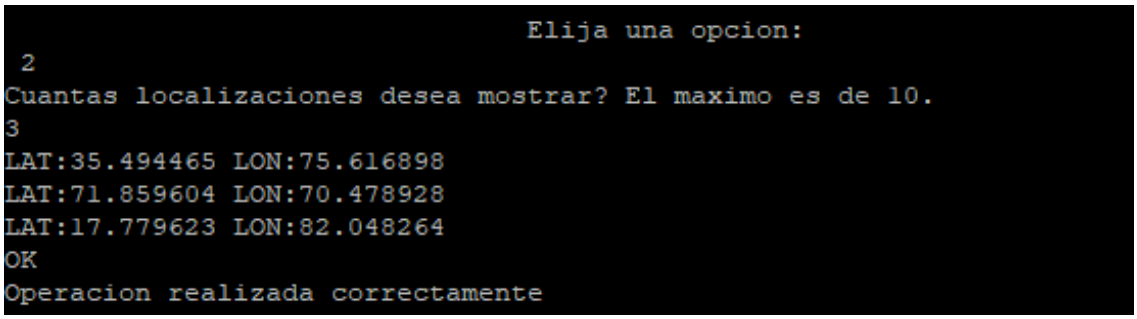

Imagen 3. Se ha elegido la opción 2 para mostrar las últimas localizaciones del usuario.

En esta captura de pantalla del programa se solicita al usuario que inserte un número de elementos a mostrar. A continuación, el cliente recibe las diferentes localizaciones e informa del resultado de la operación. No ha habido errores.

Mostrar la última localización:

Muestra la última localización registrada por el servidor. Tiene similitud con la operación anterior, pero en este caso sólo se solicita un único resultado.

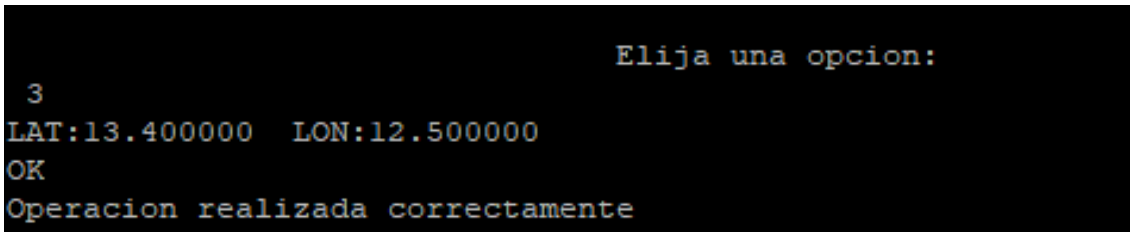

Imagen 4. Se ha elegido la opción 3. Muestra la última localización.

Se comprueba que recibe un mensaje con una latitud y longitud válida y que además recibe el mensaje de "OK" que indica que la operación ha sido exitosa.

Resetear la lista de posiciones almacenadas:

Este comando permite, dado un identificador, borrar del servidor todas las localizaciones del usuario.

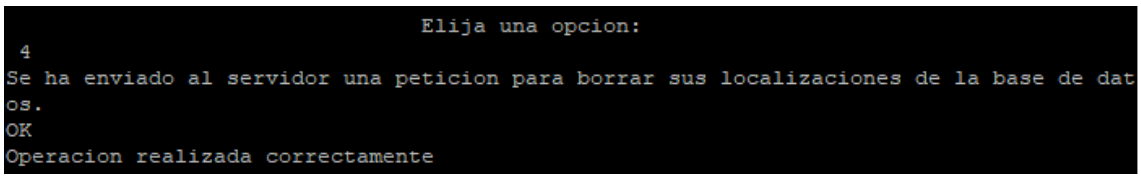

Imagen 5. Se ha elegido la opción 4. Se borra la BD.

Salir:

No está relacionada con el protocolo. Cierra la comunicación y la aplicación de una forma controlada.

Comprobaciones:

El servidor no tendría que mostrar nada, si acaso tener un fichero de tipo “log” que vaya apuntando las acciones realizadas o incidencias. De todas formas, para probar la comunicación y lo que recibe, se han imprimido por pantalla los comandos del protocolo que tiene que leer.

```
pi@raspberrypi:~/CRT $ ./servidor_geo
He recibido una peticion del cliente,es SAV 24.670000 31.940001 373
He recibido una peticion del cliente,es LST 373 3
He recibido una peticion del cliente,es LSL 373
He recibido una peticion del cliente,es RST 373
He recibido una peticion del cliente,es SAV 24.670000 31.940001 373
He recibido una peticion del cliente,es RST 373
He recibido una peticion del cliente,es LSL 373
He recibido una peticion del cliente,es LSL 373
He recibido una peticion del cliente,es RST 373
```

Imagen 6. Comandos recibidos en el servidor.

Es una forma de depurar el código o de ver que todo funciona bien de forma textual.

Implementación:

Cliente:

En las siguientes líneas se explica el funcionamiento interno del cliente.

Al arrancar el cliente se cargan en primer lugar las funciones auxiliares. Esto en C es relevante, porque no se puede llamar a una subrutina que haya sido definida después de la función principal.

En el “main” se comprueba en primer lugar que los parámetros del programa sean correctos, la ejecución requiere de una dirección IP o en su defecto de nombre asociado a una IP.

A continuación se hacen las llamadas correspondientes a las funciones relativas a la interfaz de sockets. En este caso se crea un socket de tipo UDP y se guarda la dirección IP del servidor y su puerto en una estructura de datos. Una vez que realizado se llama a la función generarId(), que proporciona el cliente un identificador basado en un número aleatorio, e imprimirMenu(), que muestra por pantalla las opciones disponibles.

Seguidamente aparece el bucle principal de la aplicación.

Inicialmente el programa se queda a la espera de que el usuario introduzca un número por la entrada estándar, que tiene que estar comprendido entre 1 y 5. En caso de introducir un valor erróneo se instará al usuario para que vuelva a introducir un número.

Conviene explicar cómo se trata la preparación, envío y recepción de datos para cada una de las funcionalidades que ofrece la aplicación:

- **Guardar la localización actual.** Esta aplicación se basa en simulaciones. En éste caso se están usando unos valores para ID, latitud y longitud ya definidos previamente. El mensaje se prepara con la función sprintf, que permite imprimir dentro de un array de caracteres.
  Una vez preparado el mensaje se procede con el envío, que se hace con la función sendto. Por último se espera la recepción de un mensaje por parte del servidor, que será “OK” si ha ido bien o “ER” si ha habido algún problema, esto se hace con la función

recvfrom. Se analiza la respuesta recibida y se informa al usuario sobre el estado de la operación.

- **Mostrar las últimas localizaciones.** Para usar correctamente esta función se necesitan saber cuántas localizaciones desea ver el usuario, con la limitación del máximo de 10. La función que pregunte al usuario por el número de localizaciones debe de ejecutarse en el cliente, sería un fallo de diseño hacerlo en el servidor, ya que se estaría haciendo una petición de comunicación innecesaria.
  En cuanto al envío de mensajes, una vez preparado el buffer, hay que quedarse a la espera de recibir todos los mensajes y además un último mensaje con la respuesta de "OK" o "ER". En esta implementación no existe, ya que está simulado, pero sería conveniente que si el servidor no tuviera "N" datos mandara un "ER", y que en el lado del cliente se analizase el mensaje para poder salir de ese bucle. (Por ejemplo, si se solicita un número de entradas mayor que el de las almacenadas en la BD para ese usuario).
- **Mostrar la última localización.** No añade ninguna complejidad adicional sobre los anteriores. En este caso hay que preparar el comando y esperar dos respuestas, una con la última localización y otra con el estado de la operación. Podría ser interesante aplicar el mismo control de errores que se ha sugerido en la acción anterior, ya que podría ocurrir un caso en el que no hubiera ninguna localización almacenada para ese ID.
- **Resetear la lista de posiciones almacenadas.** Tampoco añade ninguna novedad al respecto. Se prepara el mensaje y en este caso sólo se espera una única respuesta, por parte del servidor, al igual que en la primera operación.
- **Salir.** Permite el cierre de la aplicación de forma controlada, primero cerrando el socket y luego la aplicación. El servidor seguirá activo para recibir nuevas peticiones en el futuro.

Servidor:

En este apartado se explica el funcionamiento interno del servidor. Para que no haya conflictos con otros puertos, suele ser recomendable usar un puerto superior a 50000.

En primera estancia, se hacen los #defines de los 4 comandos que se pueden tratar, además de definir el array en el que se buscarán los comandos, junto con una ID fija predefinida por nosotros (ID = 373).

Tras esto, prosigue la función auxiliar buscar_substrings, la cual se encargará de la búsqueda de los comandos especificados (SAV, LST, LSL, RST).

Después de estas declaraciones ya se puede dar paso al programa principal, es decir, el "*main*".

Las variables locales del *main*, que aparecen nada más empezar el código de este, servirán tanto para realizar las operaciones necesarias en el protocolo UDP, como para realizar la interacciones adecuadas al protocolo implementado, así como crear latitudes o longitudes o recibir latitudes, longitudes, identificadores de ID y números enteros.

Principalmente se crea el socket que se va a encargar de la comunicación, mediante la función "socket", indicando que va a ser socket para el protocolo UDP -> **SOCK_DGRAM**.
A continuación, se realizan las asignaciones necesarias para realizar a posteriori el "*bind*", el cual le asigna una dirección al socket creado anteriormente.

Finalmente se llega al el bucle principal, en el cual se realizarán las operaciones más significativas.

Nada más entrar en el bucle se queda a la espera de recibir un mensaje del cliente mediante "*recvfrom*", para poder empezar a tratar lo recibido. Si todo ha salido bien habrá que buscar el comando recibido en la lista de comandos anteriormente especificada, pero en caso de no encontrarlo aparecerá un error. Si todo ha ido bien, se pasa a un "*switch"*.

En el "*switch*" se tratará el comando recibido por el cliente. Dependiendo de cuál sea se hace una operación u otra. A continuación se explica cada caso:

- "**SAV**". Lo primero que hace es escanear lo recibido en el buffer, para poder obtener de esta manera el ID del cliente, si la ID no coincide con la predefinida en el servidor se enviará un mensaje de error "ER" al cliente mediante "sendto".
  En caso de que todo vaya bien se enviará un mensaje correspondiente a lo especificado en el protocolo, que en este caso será un "OK".
- "LST". Escanea lo recibido en el buffer, para obtener nuevamente la ID del cliente, si la ID no coincide con la predefinida en el servidor se enviará un mensaje de error "ER" al cliente mediante "sendto". En caso contrario, se obtienen del buffer las "N" latitudes y longitudes especificadas por el cliente, este número indicará cuántas latitudes y longitudes desea recibir. Como se trata de una simulación, se crea dentro de un bucle una latitud y una longitud aleatorias, después se imprimen en el buffer, dejándolo prepara do para enviárselo a continuación al cliente. Este proceso se repetirá N veces, una vez transcurridas las iteraciones se enviará un mensaje "OK", para indicar al cliente que todo ha salido correctamente.
- "LSL". Al igual que en las anteriores, se comprueba la ID del cliente. En el caso de que coincida, se realizan las operaciones oportunas al comando. Para realizar el envío de la última latitud y longitud, se predefinen dos float, y se imprimen en el buffer, para después enviárselo al cliente mediante un "sendto", finalmente se envía al cliente el mensaje "OK" indicando que todo ha salido correctamente.
- "RST". Se vuelve a escanear en busca del UD. Si la ID no coincide con la predefinida en el servidor se enviará un mensaje de error "ER" al cliente mediante "sendto".
  En caso de que todo vaya bien se enviará un mensaje correspondiente a lo especificado en el protocolo, que en este caso será un "OK". Indica de este modo que se han borrado las entradas correspondientes a ese ID de la base de datos. Al ser una simulación, el borrado es ficticio.

Justificación para el uso de Real-Time

Lo que se ha implementado es una simulación, pero puede ser fácilmente escalable. Se ha creado un protocolo de comunicación propio para una aplicación distribuida. Con unas pequeñas modificaciones del servidor y del cliente puede ser perfectamente funcional.

¿Qué cambios habría que hacer?

- El servidor tendría que tener una base de datos con dos campos como mínimo, la ID de los usuarios y sus coordenadas. La forma más básica de implementarlo podría ser mediante un fichero de texto (por ejemplo con una estructura sencilla XML), la lectura de estos archivos es muy sencilla en C. Hay ya funciones predefinidas para el acceso a lectura y escritura de ficheros.
- Al tener una base de datos, la aplicación puede tener acceso a añadir nuevas entradas de un usuario con un ID determinado, leer esas entradas y también borrarlas.
- A los clientes distribuidos, habría que asignarles un ID único.
- Si se desea que los datos sean reales, se podrían recoger las coordenadas de un módulo GPS, en caso de tener acceso a ese Hardware y poder hacer la lectura correspondiente.
- Se podría hacer un bucle en el cliente que fuera enviando una nueva ubicación cada cierto tiempo utilizando "*hard-real-time*". De este modo un usuario podría estar grabando una ruta.

¿Por qué Real-Time?

Uno de los usos comunes de un sistema de geolocalización es conocer una ruta efectuada. En caso de querer medir velocidades medias entre diferentes tramos conviene que la adquisición de datos sea rápida y el envío también, así será más fiel a la realidad.

Por otra parte, si se desea que al servidor se conecte más de un dispositivo y sólo se quiere que haga esa rutina, conviene que tenga preferencia sobre otras tareas del SO que pudieran ejecutarse. En este caso se le ha asignado una política de "*scheduling*" de tipo "*Round-Robin*". Aunque tenga preferencia sobre tareas, no significa que vaya a haber inanición (independientemente de usar RR o FIFO). Para evitar que un proceso en bucle y con máxima prioridad tome el control del sistema y no haya forma de recuperarlo sin reiniciar, el kernel de Linux limita de forma global cuanto tiempo se le asigna a un "*scheduling*" en tiempo real. El tiempo restante se asigna para que se puedan ejecutar procesos con el tipo de "*scheduling*" "SCHED_OTHER".

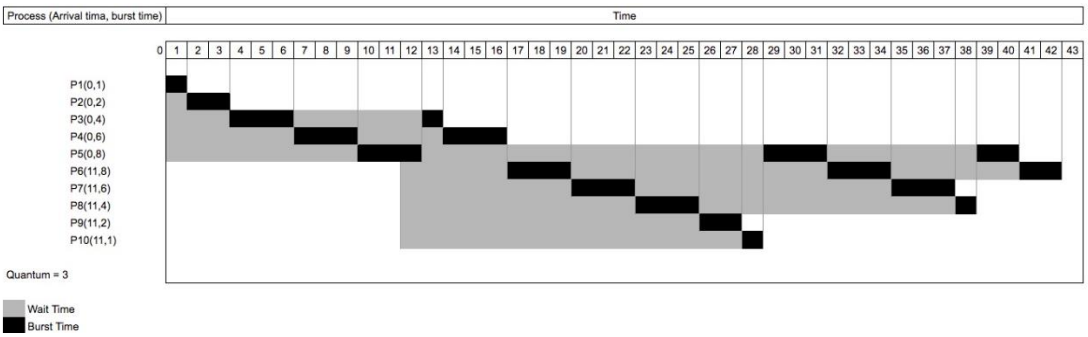

Imagen 7. Ejemplo de "*scheduling*" de tipo "*Round-Robin*".

Al programa se le ha dado una alta prioridad. En Linux la prioridad va de 1 a 99, se le ha asignado 85. Este proceso tendrá mayor prioridad que otros y se le proporcionará un "slice" mayor para terminar la operación antes de que entre otra rutina, mientras que para otras el "slice" será menor.

Bibliografía

Operating System Concepts - Silberschatz A.,Galvin P.B., Gagne G. (2008)

Operating Systems: Internals and Design Principles - Stallings W. (2011)

Understanding Linux Scheduling – Sojan James

Computer Networking: A Top-Down Approach - J.F. Kurose, K.W. Ross

Beej's Guide to Network Programming (Programación de Sockets en C)