3.6.1 Sockets

Un **socket** se define como un punto terminal de una comunicación. Una pareja de procesos que se comunican a través de una red emplea una pareja de *sockets*, uno para cada proceso. Cada *socket* se identifica mediante una dirección IP concatenada con un número de puerto. En general, los *sockets* usan una arquitectura cliente-servidor: el servidor espera a que entren solicitudes del cliente, poniéndose a la escucha en un determinado puerto. Una vez que se recibe una solicitud, el servidor acepta una conexión del *socket* cliente y la conexión queda establecida. Los servidores que implementan servicios específicos (como telnet, ftp y http) se ponen a la escucha en puertos bien conocidos (un servidor telnet escucha en el puerto 23, un servidor ftp escucha en el puerto 21 y un servidor web o http escucha en el puerto 80). Todos los puertos por debajo de 1024 se consideran *bien conocidos* y podemos emplearlos para implementar servicios estándar.

Cuando un proceso cliente inicia una solicitud de conexión, la computadora *host* le asigna un puerto. Este puerto es un número arbitrario mayor que 1024. Por ejemplo, si un cliente en un *host* X con la dirección IP 146.86.5.20 desea establecer una conexión con un servidor web (que está escuchando en el puerto 80) en la dirección 161.25.19.8, puede que al *host* X se le asigne el puerto 1625. La conexión constará de una pareja de *sockets*: (146.86.5.20:1625) en el *host* X y (161.25.19.8:80) en el servidor web. Esta situación se ilustra en la Figura 3.18. Los paquetes que viajan entre los *hosts* se suministran al proceso apropiado, según el número de puerto de destino.

Todas las conexiones deben poderse diferenciar. Por tanto, si otro proceso del *host* X desea establecer otra conexión con el mismo servidor web, deberá asignarse a ese proceso un número de puerto mayor que 1023 y distinto de 1625. De este modo, se garantiza que todas las conexiones dispongan de una pareja distintiva de *sockets*.

Aunque la mayor parte de los ejemplos de programas de este texto usan C, ilustraremos los *sockets* con Java, ya que este lenguaje proporciona una interfaz mucho más fácil para los *sockets* y dispone de una biblioteca muy rica en lo que se refiere a utilidades de red. Aquellos lectores que estén interesados en la programación de *sockets* en C o C++ pueden consultar las notas bibliográficas incluidas al final del capítulo.

Java proporciona tres tipos diferentes de *sockets*. Los ***sockets* TCP orientados a conexión** se implementan con la clase `Socket`. Los ***sockets* UDP sin conexión** usan la clase `DatagramSocket`. Por último, la clase `MulticastSocket` (utilizada para multidifusión) es una subclase de `DatagramSocket`. Un *socket* multidifusión permite enviar datos a varios receptores.

Nuestro ejemplo describe un servidor de datos que usa *sockets* TCP orientados a conexión. La operación permite a los clientes solicitar la fecha y la hora actuales al servidor. El servidor escucha en el puerto 6013, aunque el puerto podría usar cualquier número arbitrario mayor que 1024. Cuando se recibe una conexión, el servidor devuelve la fecha y la hora al cliente.

En la Figura 3.19 se muestra el servidor horario. El servidor crea un `ServerSocket` que especifica que se pondrá a la escucha en el puerto 6013. El servidor comienza entonces a escuchar en

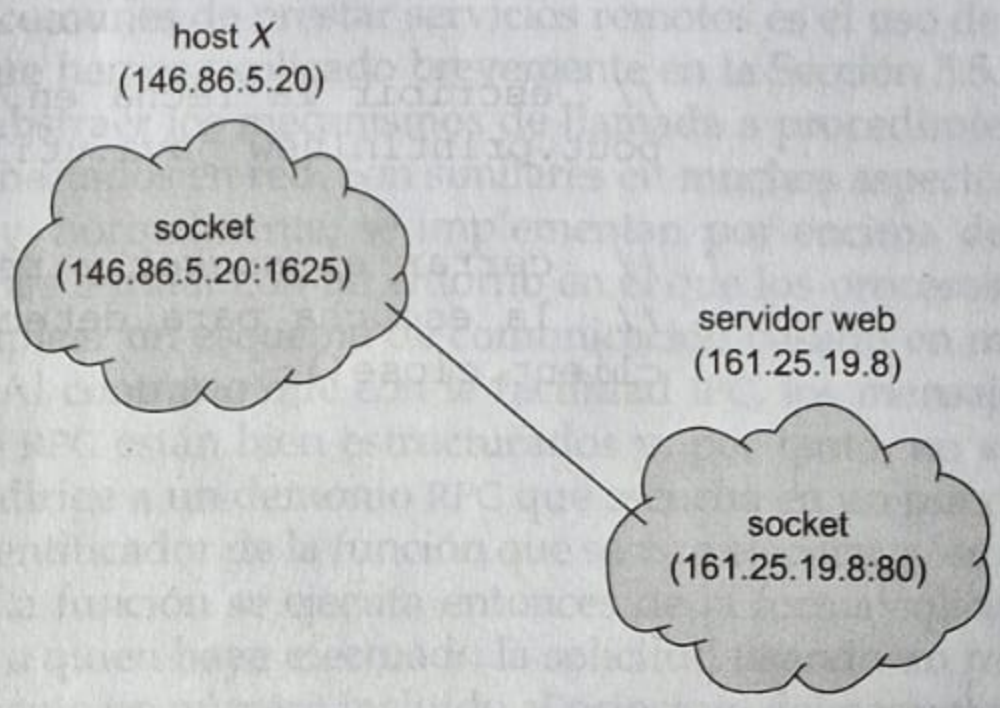

Figura 3.18 Comunicación usando *sockets*.