# 9 Servidor de dominio Internet tipo UDP

## Introducción

En la actualidad Internet ha sido uno de los principales motores para el auge de las aplicaciones distribuidas. La gran mayoría de estas aplicaciones implementan una arquitectura cliente servidor, donde una de las computadoras funciona como solicitante de servicios (cliente) y otra como suministradora de servicios (servidor). Este tipo de aplicaciones no sería posible sin la conexión física y lógica de sistemas de cómputo diversos que tienen inclusive sistemas operativos distintos.

UNIX en su versión BSD implementó un conjunto de funciones conocidas como *sockets*, los cuales permiten a los procesos comunicarse. Sin embargo, a diferencia de la memoria compartida o las colas de mensajes, los *sockets* comunican procesos que pueden encontrarse en sistemas de cómputo distintos.

Dado que los *sockets* se encuentran implementados en una arquitectura de capas, nos es posible el uso de las funciones para la comunicación sin tener que preocuparnos por otros detalles. Por ejemplo, no importa si los sistemas de cómputo se encuentran conectados mediante fibra óptica, cable coaxial o de forma inalámbrica la interfaz seguirá siendo la misma y por consiguiente nuestras aplicaciones no requerirán modificación alguna.

Existen dos principales modos para conectar procesos en computadoras diferentes. Estos se conocen como orientado a conexión (TCP) y no orientado a conexión (UDP).

En el presente capítulo nos concentraremos en crear un servidor que utiliza el protocolo de datagrama de usuario (UDP), el cual, al no ser orientado a conexión, no garantiza que los mensajes emitidos sean recibidos, ni tampoco que el orden en que se envían los paquetes sea el mismo que cuando se reciben. Sin embargo tiene la ventaja sobre el protocolo TCP en que es más rápido y genera menor tráfico de red si se implementa correctamente.

# 9 Servidor de dominio Internet tipo UDP

## Introducción

En la actualidad Internet ha sido uno de los principales motores para el auge de las aplicaciones distribuidas. La gran mayoría de estas aplicaciones implementan una arquitectura cliente servidor, donde una de las computadoras funciona como solicitante de servicios (cliente) y otra como suministradora de servicios (servidor). Este tipo de aplicaciones no sería posible sin la conexión física y lógica de sistemas de cómputo diversos que tienen inclusive sistemas operativos distintos.

UNIX en su versión BSD implementó un conjunto de funciones conocidas como *sockets*, los cuales permiten a los procesos comunicarse. Sin embargo, a diferencia de la memoria compartida o las colas de mensajes, los *sockets* comunican procesos que pueden encontrarse en sistemas de cómputo distintos.

Dado que los *sockets* se encuentran implementados en una arquitectura de capas, nos es posible el uso de las funciones para la comunicación sin tener que preocuparnos por otros detalles. Por ejemplo, no importa si los sistemas de cómputo se encuentran conectados mediante fibra óptica, cable coaxial o de forma inalámbrica la interfaz seguirá siendo la misma y por consiguiente nuestras aplicaciones no requerirán modificación alguna.

Existen dos principales modos para conectar procesos en computadoras diferentes. Estos se conocen como orientado a conexión (TCP) y no orientado a conexión (UDP).

En el presente capítulo nos concentraremos en crear un servidor que utiliza el protocolo de datagrama de usuario (UDP), el cual, al no ser orientado a conexión, no garantiza que los mensajes emitidos sean recibidos, ni tampoco que el orden en que se envían los paquetes sea el mismo que cuando se reciben. Sin embargo tiene la ventaja sobre el protocolo TCP en que es más rápido y genera menor tráfico de red si se implementa correctamente.

## Ejemplo práctico

Un *socket* representa un extremo en la comunicación bidireccional entre dos procesos y ofrece una interfaz para acceder a los servicios de red en la capa de transporte. Los dos procesos que se desean comunicar deben crear un *socket* cada uno. Existen dos dominios de comunicación:

- El dominio UNIX (`AF_UNIX`), para comunicar procesos dentro de la misma computadora.
- El dominio Internet (`AF_INET`), para comunicar procesos en computadoras distintas utilizando el protocolo TCP/IP.

Existen dos tipos de *sockets*:

- *Sockets Stream* (`SOCK_STREAM`), donde el protocolo utilizado es orientado a conexión.
- *Sockets* tipo datagrama (`SOCK_DGRAM`), donde el protocolo utilizado es no orientado a conexión.

El direccionamiento se logra dependiendo del dominio de comunicación utilizado:

- AF_UNIX para el dominio UNIX. Utiliza como dirección el nombre de un archivo local.
- AF_INET para el dominio Internet. Utiliza la dirección IP de la máquina y un número de puerto dentro de la máquina.

El programa 9-1 muestra el código mínimo de un servidor, el cual, si se ejecuta, no realizará ninguna acción hasta que no se conecte un cliente ( éste se elabora en el siguiente capítulo). Las funciones que lo componen se describen a continuación.

**Programa 9-1**
Servidor UDP que recibe dos enteros y devuelve el resultado de su suma

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <netdb.h>

int puerto = 7200;

int main(void)
{
   int num[2];
   int s, res, clilen;
1  struct sockaddr_in server_addr, msg_to_client_addr;
```

continúa

```
2  s = socket(AF_INET, SOCK_DGRAM, 0);
   /* se asigna una dirección al socket del servidor*/
   bzero((char *)&server_addr, sizeof(server_addr));
   server_addr.sin_family = AF_INET;
   server_addr.sin_addr.s_addr = INADDR_ANY;
   server_addr.sin_port = htons(puerto);

3  bind(s, (struct sockaddr *)&server_addr,
   sizeof(server_addr));
   clilen = sizeof(msg_to_client_addr);

   while(1) {

4     recvfrom(s, (char *) num, 2*sizeof(int), 0, (struct
   sockaddr *)&msg_to_client_addr, &clilen);

      res = num[0] + num[1];

      /* envía la petición al cliente. La estructura
   msg_to_client_addr contiene la dirección socket del cliente
   */
5     sendto(s, (char *)&res, sizeof(int), 0, (struct sockaddr
   *)&msg_to_client_addr, clilen);
      }
}
```

**Línea 1** Estructura que almacena la dirección genérica de un *socket* y se encuentra definida como sigue para el caso UNIX:

```
struct sockaddr_un
{
   short sun_family;        /*AF_UNIX*/
   char sun_path[108];  /*Ruta */
};
```

para el caso de socket internet se usa la estructura:

```
struct sockaddr_in
{
   short       sin_family;  /* AF_INET */
   u_short       sin_port;    /* 16 bits con el número de .....  */
   struct in_addr sin_addr;       /* 32 bits con la identificación
de la red y del nodo */
   char              sin_zero[8]; /* 8 bytes no usados */
}
```

donde:

```
struct in_addr
{
   u_long s_addr;    /* 32 bits que contienen la identificación de
la red y del nodo */
};
```

**Pregunta 9.1** Las librerías para el compilador gcc se ubican en `/usr/include`, busque en los archivos `include` de este programa la definición de la estructura `sockaddr_in` y diga que se almacena en el miembro `sin_port`.

Existen más dominios para otras aplicaciones específicas además de `AF_UNIX` y `AF_INET` definidos en `/linux/socket.h`
Mediante el puerto se puede identificar un único servicio en una computadora, los puertos inferiores a 1024 están reservados para servicios del sistema. Los puertos que se gestionan automáticamente por el sistema operativo, van desde el 1024 hasta el 5000. El máximo número de puerto utilizable es el 65535, de donde podemos inferir que se ocupan 2 *bytes* para su almacenamiento.

En el protocolo TCP/IP los números que se transmiten se apegan al formato big-endian. Pero como no todas las computadoras lo usan, se utilizan las siguientes funciones para traducir formatos:

```
u_long htonl(u_long hostlong);
u_short htons(u_sort hostsort);
u_long ntohl(u_long netlong);
u_short ntohs(u_long netshort);
```

la primera función traduce un número de 32 *bits* en el formato de la computadora a un formato big-endian.
La segunda función traduce un número de 16 *bits* en el formato de la computadora a un formato big-endian.
Las dos últimas funciones realizan el trabajo inverso.

**Línea 2** Un *socket* se crea mediante la función:

```
int socket(int dominio, int tipo, int protocolo);
```

el parámetro `protocolo` se pone en 0 para dejarlo en manos del sistema. El *socket* creado no tiene asociada ninguna dirección.

**Pregunta 9.2** En este caso, ¿cuál es el dominio del socket y cuál es su tipo? ¿Para qué se usa la función `bzero`? ¿Cuál es el tamaño en bits de la variable puerto? Imprima el valor de la variable puerto en hexadecimal `(%x)` antes y después de usar `htons` y explique detalladamente qué operaciones realiza la función `htons` *byte* por *byte*.

**Pregunta 9.3** En el programa se declaran dos variables del tipo `struct sockaddr_in`, ¿cuál de las dos variables se inicializa en el programa servidor y con qué valores? Si la otra variable no se inicializa, ¿qué valor deben tener sus miembros?

**Línea 3** Para asignar una dirección IP y un puerto a un *socket* para que se pueda ubicar el punto de conexión, se utiliza la función `bind`:

```
int bind(int socket, struct sockaddr *dir, int long);
```

El primer argumento es el identificador devuelto por `socket( )`. El segundo especifica la dirección que se va a asignar al *socket*, la cual se debe encontrar en la estructura definida en la línea 1. El tercer argumento especifica el número de *bytes* que ocupa la dirección en la estructura.

**Línea 4** Para recibir datos en una conexión de tipo datagrama se utiliza:

```
int recvfrom(int socket, char *mensaje, int len, int flags, struct
sockaddr *dir, int *long);
```

`len` representa el máximo número de *bytes* que se pueden escribir en la memoria referenciada por `mensaje`. El miembro `flags` se inicializa con 0, `dir` almacena la dirección del *socket* del que se han recibido los datos y `long` la longitud que ocupa dicha dirección en bytes.
La estructura `sockaddr` es utilizada por muchas llamadas al sistema y tiene la forma:

```
struct sockaddr
{
   u_short sa_family;
   char sa_data[14];
}
```

Cabe aclarar que el programa se quedará bloqueado en esta línea y no se ejecutará la siguiente línea de código hasta que llegue un datagrama proveniente de otra computadora. Esta tarea la llevaremos a cabo con el programa cliente del siguiente capítulo. Para terminar el proceso sólo hay que oprimir CTRL-C.

**Línea 4** La función `recvfrom` devuelve el número de *bytes* leídos.

**Línea 5** Para enviar datos en una conexión de tipo datagrama se utiliza:

```
int sendto(int socket, char *mensaje, int long, int flags, struct
sockaddr *dir, int long);
```

El argumento `dir` representa la dirección del socket remoto al que se quieren enviar los datos y `long` la longitud en bytes que ocupa dicha dirección.

Como puede observar dentro del ciclo `while`, el servidor se bloquea esperando un mensaje de solicitud. Por lógica dicho mensaje debe contener además de los números que se quieren sumar la dirección IP y el puerto del remitente, de modo

que el servidor tenga la información necesaria para que pueda enviarle al cliente la respuesta.

**Pregunta 9.4** ¿En el programa 9-1, cuáles son las variables donde se almacenarían la dirección IP y el puerto de algún cliente que le haga una solicitud al servidor? En su apreciación ¿cuál es el propósito de inicializar los miembros de la variable `server_addr`?

# 10 Cliente Internet tipo UDP

## Introducción

Dado que en los *sockets* UDP se tienen dos instrucciones principales para la comunicación, una de ellas para enviar mensajes (`sendto`) y otra para recibir mensajes (`recvfrom`), es importante considerar que debe existir un proceso que espera recibir un mensaje, antes de que otro proceso se lo envíe. Ésta es una de las razones principales por las que primero se ejecutan los procesos servidores y después los procesos clientes. Si se ejecutase en orden inverso, el cliente debiera emitir un mensaje de error al no encontrar al proceso servidor.

Siguiendo la misma lógica, deberá existir una instrucción `recvform` en el cliente, antes de que el servidor ejecute un `sendto`. De modo que se estará presentando una sincronización (un proceso envía y el otro proceso recibe en un momento dado) definida por el propio mecanismo de comunicación. Es por esta razón que al servidor del capítulo anterior se le conoce como interactivo.

Para que un mensaje en el dominio de Internet pueda llegar a su destino, él cual podría estar ubicado en cualquier nodo de la red de redes, es imprescindible direccionarlo. Es decir, el paquete debe contener, además de la información que se requiere enviar, una dirección única en la red (dirección IP) y un identificador del proceso al cual esté destinado el mensaje (puerto). Básicamente ésta es la información que se tiene que proporcionar a un *socket* para establecer la comunicación con cualquier otro proceso en la Internet.

## Ejemplo práctico

Compile el programa 10-1 (cliente), posteriormente ejecute el servidor del capítulo anterior y en otra terminal de la misma computadora ejecute el cliente.

Podrá observar en este código que el cliente envía los enteros 2 y 5 hacia el servidor, para posteriormente recibir el resultado de la suma. De modo que

tendremos dos mensajes: el primero, que viaja del cliente hacia el servidor y contiene los dos enteros que se quieren sumar, el segundo, que va del servidor hacia el cliente y que contiene el resultado de la suma.

**Programa 10-1** Cliente envía dos enteros hacia el servidor

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <netdb.h>

int puerto = 7200;

int main(void)
{
   struct sockaddr_in msg_to_server_addr, client_addr;
   int s, num[2], res;

   s = socket(AF_INET, SOCK_DGRAM, 0);

   /* rellena la dirección del servidor */
   bzero((char *)&msg_to_server_addr,
sizeof(msg_to_server_addr));
   msg_to_server_addr.sin_family = AF_INET;
     msg_to_server_addr.sin_addr.s_addr =
inet_addr("127.0.0.1");
   msg_to_server_addr.sin_port = htons(puerto);

   /* rellena la dirección del cliente*/
   bzero((char *)&client_addr, sizeof(client_addr));
   client_addr.sin_family = AF_INET;
   client_addr.sin_addr.s_addr = INADDR_ANY;
   /*cuando se utiliza por número de puerto el 0, el sistema se
encarga de asignarle uno */
   client_addr.sin_port = htons(0);

   bind(s, (struct sockaddr *)&client_addr,sizeof(client_addr));

   num[0] = 2;
   num[1] = 5; /*rellena el mensaje */

   sendto(s, (char *)num, 2 * sizeof(int), 0, (struct sockaddr
*) &msg_to_server_addr, sizeof(msg_to_server_addr));

   /* se bloquea esperando respuesta */
   recvfrom(s, (char *)&res, sizeof(int), 0, NULL, NULL);

   printf("2 + 5 = %d\n", res);

   close(s);
}
```

Por otra parte, aunque existen dos variables `struct sockaddr_in` en ambos códigos, cada una tiene un uso diferente, como podemos observar en la figura 10.1

Podemos darnos cuenta de que tanto en el cliente como en el servidor, la función `bind` enlaza la variable inicializada con el *socket* para recepción de mensajes.

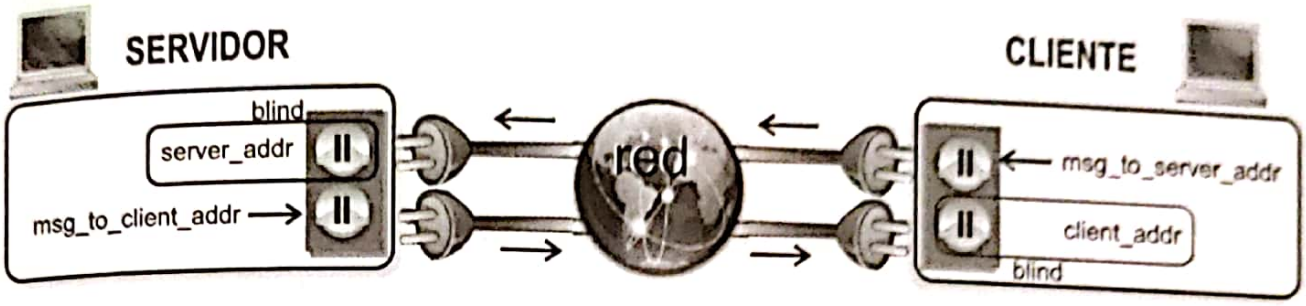


**Figura 10.1** Comunicación por sochets en una red

**Pregunta 10.1** Investigue cual es la función que realiza `inet_addr("127.0.0.1")` en el programa 10-1.¿Porque se la pasa la dirección 127.0.0.1 como argumento?

**Ejercicio 10.1** Añada al programa 10-1 las líneas de código necesarias, para imprimir dígito por dígito el contenido de la variable:

```
msg_to_server_addr.sin_addr.s_addr.
```

(Recomendación: haga una copia hacia una cadena de caracteres mediante la función `memcpy`)

**Ejercicio 10.2** Modifique el programa 10-1 para que imprima inmediatamente después de la función `recvfrom` en el programa del servidor y en el programa del cliente, la dirección IP y el puerto de quien le ha enviado el datagrama (remitente).

**Ejercicio 10.3** Haga las modificaciones necesarias para ejecutar el código cliente y el servidor en distintas computadoras. Al programa cliente se le debe pasar la IP del servidor en la línea de comandos (observación: en LINUX, para obtener la IP de una computadora se utiliza el comando `ifconfig` ubicado en el directorio /sbin. La localización de cualquier archivo ejecutable se realiza mediante el comando `whereis`, pruébelo para este caso).