Los Sockets son una interfaz que permiten que las aplicaciones puedan acceder a los servicios que brinda el software TCP/IP, permitiendo la comunicación entre procesos en el mismo equipo o en equipos diferentes.

Introducción a los Sockets : Introducción bastante completa sobre la definición y las principales funciones para el manejo de Sockets

Funciones de Sockets : Funciones para trabajar con sockets

Modelo Cliente/Servidor Streams Sockets : Códigos fuentes de la implementación de Streams Sockets Cliente/Servidor

Modelo Cliente/Servidor Datagram Sockets : Códigos fuentes de la implementación de Datagram Sockets

- El código fuente de los ejemplos de este tutorial **descarguenlo aquí**

Sockets: Entablando conversaciones...

Los Sockets son una interfaz que permiten que las aplicaciones puedan acceder a los servicios que brinda el software TCP/IP, permitiendo la comunicación entre procesos en el mismo equipo o en equipos diferentes.

La Interfaz Socket proporciona funciones generalizadas que dan soporte a las comunicaciones en red empleando para ello muchos de los protocolos disponibles hoy en día. Los llamados sockets hacen referencia a todos los protocolos TCP/IP como una única familia. Las llamadas permiten al programador especificar el tipo de servicio requerido, en vez de el nombre de un protocolo específico.

Los sockets pueden ser:

- Basados en la conexión (connection based) o Independientes de la conexión (connectionless): ¿La conexión se ha definido antes de la comunicación o cada paquete definirá su destino?

- Basados en paquetes (packet based) o basados en flujos (streams based): ¿Están definidos los límites del mensaje o es un flujo?

- Fiable (reliable) o inestable (unreliable): ¿Puede perderse, reproducirse, solicitarse de nuevo o corromperse un mensaje?

Los Sockets se caracterizan por un dominio, un tipo y un protocolo de comunicación.

Los dominios más comunes son:

El dominio de comunicación nos dice como ser va a realizar la comunicación de los procesos que se van a intercomunicar, o mejor dicho, en que ambiente.
Si los procesos se comunicarán bajo la forma de un único sistema (tipo root de Unix), el dominio de comunicación será AF_UNIX, si los procesos se comunicarán como sistemas independientes y estos se hallan unidos mediante una red TCP/IP, el dominio de comunicación será AF_INET.

Cabe aclarar que existen otros dominios de comunicación.

Los sockets no se han diseñado solamente para TCP/IP. La idea original fue que se usase la misma interfaz también para distintas familias de protocolos.

En esta introducción solo trataremos el dominio AF_INET.

Algunos dominios:

- AF_INET (mediante una red TCP/IP).

- AF_UNIX (en el mismo sistema).

- Otros dominios.

Los tipos más comunes son:

Sockets Stream: son los más utilizados, hacen uso del protocolo TCP, el cual nos provee un flujo de datos bidireccional, secuenciado, sin duplicación de paquetes y libre de errores.

Sockets Datagram: hacen uso del protocolo UDP, el cual nos provee un flujo de datos bidireccional, pero los paquetes pueden llegar fuera de secuencia, pueden no llegar o contener errores.
Por lo tanto el proceso que recibe los datos debe realizar re-secuenciamiento, eliminar duplicados y asegurar la confiabilidad.
Se llaman también sockets sin conexión, porque no hay que mantener una conexión activa, como en el caso de sockets stream.
Son utilizados para transferencia de información paquete por paquete. Ejemplo: dns, tftp, bootp, etc.
Entones podríamos preguntar, Cómo hacen estos programas para funcionar si pueden perder datos ?
Ellos implementan un protocolo encima de UDP que realiza control de errores.

Sockets SeqPacket: Establece una conexión fiable bidireccional con un tamaño de mensaje máximo definido. (Este tipo puede no estar habilitado en algunos sistemas.)

Sockets Raw: no son para el usuario común, son provistos principalmente para aquellos interesados en desarrollar nuevos protocolos de comunicación o para hacer uso de facilidades ocultas de un protocolo existente.

Cada tipo de Sockets tiene uno o más protocolos:

TCP/IP (Streams)
UDP (Datagram)

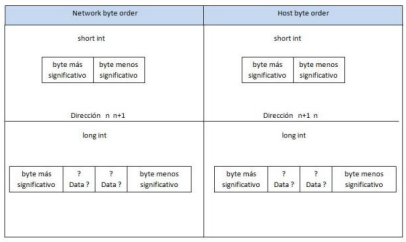
Uso de Sockets:

Los sockets basados en la conexión son cliente-servidor: el servidor espera por una conexión del cliente.
Los sockets Independientes de la conexión son de igual a igual (peer-to-peer): cada proceso es simétrico.

Almacenamiento de datos (Byte order)

Network byte order y Host byte order son dos formas en las que el sistema puede almacenar los datos en memoria.
Está relacionado con el orden en que se almacenan los bytes en la memoria RAM.
Si al almacenar un short int (2 bytes) o un long int (4 bytes) en RAM, y en la posición más alta

se almacena el byte menos significativo, entonces está en network byte order, caso contrario es host byte order.



Esto depende del microprocesador que se esté utilizando, podríamos estar programando en un sistema host byte order o network byte order, pero cuando enviamos los datos por la red deben ir en un orden especificado, sino enviaríamos todos los datos al revés. Lo mismo sucede cuando recibimos datos de la red, debemos ordenarlos al orden que utiliza nuestro sistema. Debemos cumplir las siguientes reglas:

- Todos los bytes que se transmiten hacia la red, sean números IP o datos, deben estar en network byte order.

- Todos los datos que se reciben de la red, deben convertirse a host byte order.

Para realizar estas conversiones utilizamos las funciones que se describen a continuación.

Funciones de conversión

htons() - host to network short - convierte un short int de host byte order a network byte order.

htonl() - host to network long - convierte un long int de host byte order a network byte order.

ntohs() - network to host short - convierte un short int de network byte order a host byte order.

ntohl() - network to host long - convierte un long int de network byte order a host byte order.

Puede ser que el sistema donde se esté programando almacene los datos en network byte order y no haga falta realizar ninguna conversión, pero si tratamos de compilar el mismo código fuente en otra plataforma host byte order no funcionará. Como conclusión, para que nuestro código fuente sea portable se debe utilizar siempre las funciones de conversión.

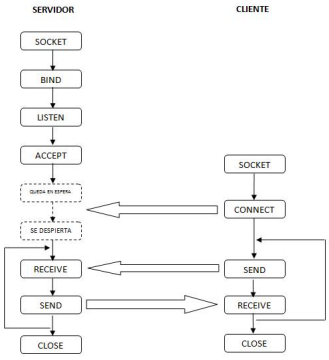
Uso de puertos

Note que para iniciar un tipo de comunicación mediante un socket se requiere designar un puerto de comunicaciones.
Esto significa que algunos puertos deben reservarse para éstos usos bien conocidos .

0-1023: Estos puertos pueden ser sólo enlazados por el sistema
1024-5000: Puertos designados a aplicaciones conocidas
5001-64K-1: Puertos disponibles para usos particulares

Modelo Simple Cliente-Servidor: Conexión Stream

En el siguiente gráfico se muestran los procesos que se ejecutan cuando se realiza una conexión Simple Cliente-Servidor:



Teoría de conexión:

En el servidor:

- Se crea un socket de comunicación mediante la función socket().

- El socket está formado por una dirección ip + un puerto de conexión, para ello hay que asociar esa información al socket mediante la función bind().

- La parte servidor de la aplicación debe establecer una cola de conexiones entrantes para atender a los clientes según el orden de llegada, esto se hace mediante la función listen().

- Ahora el servidor deberá atender las conexiones entrantes por el puerto que establecimos mediante la función accept().

- Tras establecer una comunicación entre un cliente y un servidor se enviarán datos mediante la función send() y se recibirán mediante la función recv().

- Al finalizar la comunicación deberemos cerrar el socket de comunicación.

En el cliente:

- Se crea un socket de comunicación mediante la función socket().

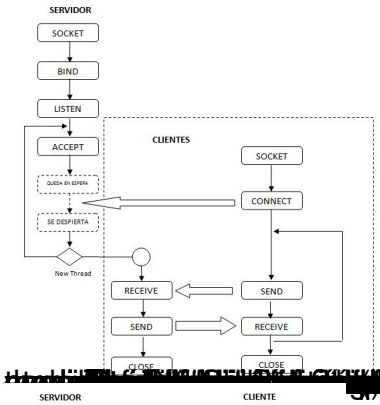
- El servidor está aceptando conexiones entrantes, así que nos conectamos al servidor mediante la función connect().

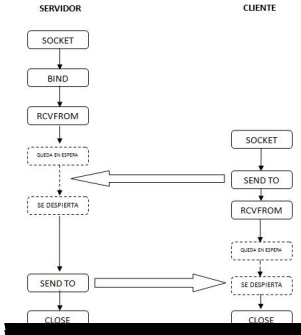
- Recibimos y enviamos datos, pues ya estamos en contacto con el equipo remoto, mediante las funciones recv() y send().

- Al finalizar la comunicación deberemos cerrar el socket de comunicación.

Modelo Concurrente Cliente-Servidor: Conexión Stream

En el siguiente gráfico se muestran los procesos que se ejecutan cuando se realiza una conexión Concurrente Cliente-Servidor:


SERVIDOR
SOCKET
BIND
LISTEN
ACCEPT
QUEDA EN ESPERA
SE DESPIERTA
New Thread
CLIENTES
SOCKET
CONNECT
RECEIVE
SEND
SEND
RECEIVE
CLOSE
CLOSE


SERVIDOR
CLIENTE
SOCKET
BIND
RCVFROM
QUEDA EN ESPERA
SE DESPIERTA
SEND TO
CLOSE
SOCKET
SEND TO
RCVFROM
QUEDA EN ESPERA
SE DESPIERTA
CLOSE