PRÁCTICA 2:

Introducción de Sockets en C.

- La Interfaz Socket es una API para redes TCP/IP que se compone de funciones o rutinas.

- Originalmente se construyó a principios de los 80 para el sistema operativo UNIX , aunque hoy en día también la utilizan otros sistemas operativos como Linux, Microsoft Windows, Mac, OS2, etc...

- Para que se dé la comunicación en una red, el programa requiere un Socket en cada extremo del proceso de comunicación.
- Las llamadas al sistema de E/S en UNIX se basan en el proceso de **open-read-write-close** (abrir-leer-escribir-cerrar).
- La interfaz de Sockets para comunicarse con una red TCP/IP, abre primero una conexión con la red, se leen y escriben datos a través de ella y una vez terminados los procesos se cierra la conexión.

- Las aplicaciones desarrolladas en Sockets están basadas en la arquitectura *Cliente-Servidor.*

  – Una aplicación, el ***servidor***, permanece a la espera de que otras aplicaciones deseen sus servicios:
    - Utilizando un determinado protocolo de red y de transporte.
    - En una determinada dirección de red.
    - En un determinado número de puerto.

  – Otra aplicación, el ***cliente***, solicita los servicios de la aplicación servidor:
    - Utilizando el mismo protocolo de red y de transporte.
    - Una dirección de red.
    - Un número de puerto.

- El primer paso para que cualquier aplicación ***cliente*** o ***servidor*** pueda comunicarse es crear un Socket.

- Se realiza con la función ***socket().***
  - Sus parámetros son:
    - ***Dominio.***
    - ***Tipo.***
    - ***Protocolo.***
  - El valor devuelto es un entero:
    - >=0 si el socket es creado correctamente.
    - <0 si se produce un error en la creación.

- ***Dominio:*** Indica el dominio de comunicación que se desea utilizar:
  - AF_INET
    - o *(PF_INET)* : Protocolos de Internet.
  - AF_UNIX
    - o *(PF_UNIX)* : Comunicación local (conexión entre aplicaciones del propio ordenador).

PF: Familia de protocolos
AF: Dirección de protocolos
(En la práctica AF=PF)

- ***Tipos***
  - **Stream** (*SOCK_STREAM*):
  - – Orientado a conexión ***(TCP).***
  - – Fiable, se asegura el orden de entrega de mensajes.
  - – No mantiene separación entre mensajes (*stream*).

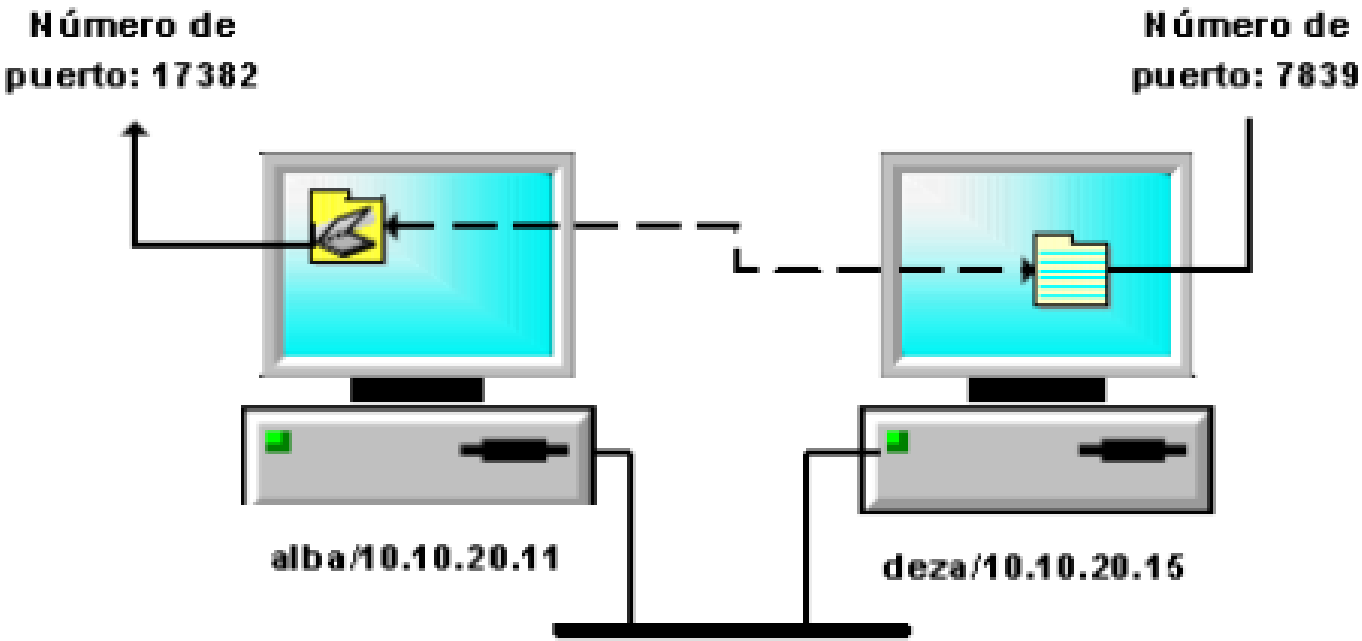
  - **Datagrama** (*SOCK_DGRAM*):
  - – Sin conexión ***(UDP).***
  - – No fiable, no se asegura el orden en la entrega.
  - – Mantiene la separación entre mensajes.

- ***Protocolo***
  - – IPPROTO_TCP, IPPROTO_UDP.
  - – Si es 0, el valor por defecto es ***TCP*** para SOCK_STREAM y ***UDP*** para SOCK_DGRAM.

- Una transmisión está caracterizada por varios parámetros:
  - Dirección *host* y puerto origen.
  - Dirección *host* y puerto destino.
  - Protocolo de transporte (UDP o TCP).

- Una dirección destino viene determinada por:
  - Dirección del *host*: 32 bits.
  - Puerto de servicio: 16 bits (Reservados: 0..1023) (Espacio de puertos TCP y UDP independientes)

- Los usuarios manejan direcciones en dos formatos:
  - decimal-punto: 10.10.20.11
  - dominio-punto: atc2.uah.es


Número de puerto: 17382
Número de puerto: 7839
alba/10.10.20.11
deza/10.10.20.15

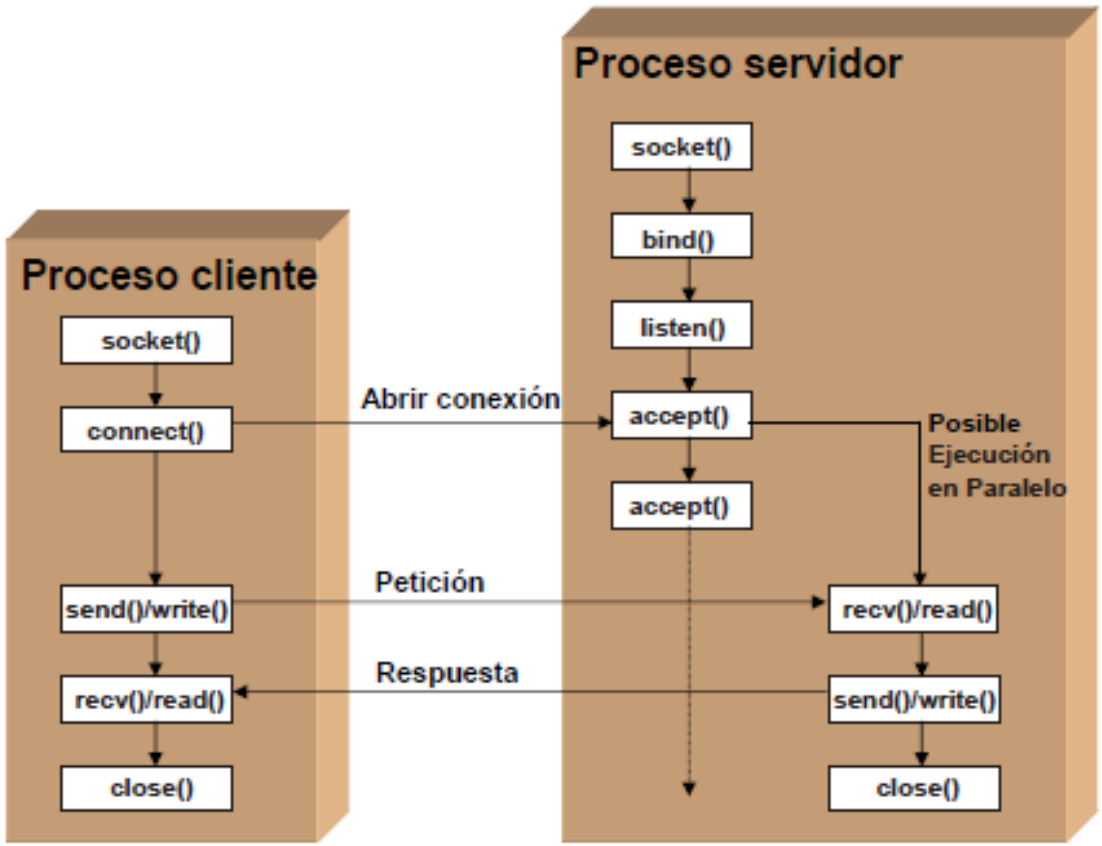
- Para asignarlos se utiliza una estructura genérica de datos que almacena la dirección de la *familia, el tipo de socket y el protocolo de dirección:*
  - En **AF_INET** es (***struct sockaddr_in***).
    - Miembros de la estructura:
      - **sin_family**: dominio (*AF_INET*).
      - **sin_port**: puerto.
      - **sin_addr**: dirección del *host*.
    - Debe inicializarse previamente a 0 con la función ***bzero()***.

- Cada Socket debe tener asignada una dirección única. La asignación de una dirección a un Socket ya creado se realiza con la función ***bind().***

- Un Socket servidor asociado a una dirección y puerto es puesto en escucha con la función ***listen()***, que indica al socket que atienda las conexiones entrantes y que confirme las solicitudes de conexión.

- El cliente realiza solicitud de comunicación por medio de la función ***connect().***
  - Un Socket *stream* sólo permite un único *connect* durante su vida.
  - Para conectarse con el mismo u otro hay que crear un nuevo Socket.

- El servidor acepta las peticiones de conexión con la función ***accept().***



Llamadas al sistema para Sockets en un protocolo orientado a la conexión (TCP)

- Cuando se produce la conexión, el servidor obtiene:
  - La dirección del Socket del cliente.
  - Un nuevo descriptor (Socket) conectado al Socket del cliente.

- Después de conexión quedan activos 2 Sockets en el servidor:
  - El original para aceptar nuevas conexiones
  - El nuevo para enviar/recibir datos por la conexión establecida.

- A continuación se procederá al envío y recepción de datos:
  - Se realiza con la función ***read()*** para leer datos y ***write()*** para escribir datos en transmisiones *TCP.*
  - Se realiza con las funciones ***recvfrom()*** o ***recv()*** para leer datos y ***sendto()*** para escribir datos en transmisiones *UDP*.

- Para cerrar la conexión:
  - ***close()****:* Cierra ambos tipos de Sockets (*UDP y TCP*). Si el socket es de tipo *stream* cierra la conexión en ambos sentidos.
  - **shutdown():** Para cerrar la conexión en un único extremo:

- Otras funcionalidades:

  • Obtener la dirección a partir de un descriptor:
  - ***getsbyname()****:* Obtiene la dirección *(en Network Byte Order)* que corresponde al nombre del host.
  - ***gethostbyaddr():*** Obtiene la dirección a través una estructura *hostent* con la información del host dado en binario.

  • Transformación de valores:
  - De formato *host* a red (*Network Byte Order):*
    • Enteros largos: ***htonl().***
    • Enteros cortos: ***htons().***
  - De formato de red (*Network Byte Order)* a *host*:
    • Enteros largos: ***ntohl().***
    • Enteros cortos: ***ntohs().***

- Múltiples clientes con *streams:*

  - **Con servidor iterativo no concurrente**
    - ***Si 1 conexión por petición.***
      - Se intercalan peticiones de los clientes (1 por iteración).
    - ***Si varias peticiones de cliente usan misma conexión.***
      - No se trata a otro cliente hasta que no termine el actual.

  - **Con servidor concurrente.**
    - ***Si 1 conexión por petición.***
      - Se sirve una petición y se termina.
    - **Si varias peticiones de cliente usan misma conexión.**
      - Se sirven peticiones hasta que cliente cierra el socket.