Unidad 3. Programación Distribuída

- Características de un sistema distribuido: Esta formado por más de un elemento computacional que son independientes, no comparten memoria interna, no están sincronizados, cada uno tiene su propio estado y a través de la red se comunican entre sí para realizar una tarea en común.

Conceptos fundamentales de un proceso de comunicación:

1.- Mensaje: Información que se intercambia entre aplicaciones.
2.- Emisor: Aplicación que envía el mensaje.
3.- Receptor: Aplicación que recibe el mensaje.
4.- Paquete: La unidad básica de información que se intercambia entre dos dispositivos.
5.- Canal de comunicación: Medio por el cual se trasmiten los paquetes.
6.- Protocolo de comunicaciones: Conjunto de reglas que determinan o definen como se intercambian los paquetes.

Pila de protocolos: Contiene todos los elementos hardware y software que son necesarios para la comunicación y establece como trabajan juntos. La más utilizada es la **pila IP (Internet Protocol).**

Niveles del protocolo IP:

1.- Nivel de red: Nivel más bajo, está compuesto por los elementos de hardware. Se encarga de trasmitir los paquetes de información, y puede estar construido en LAN o WAN.

2.- Nivel de Internet: Por encima del de red y compuesto por los elementos software que se encargan de dirigir los paquetes por la red.

3.- Nivel de Transporte: Por encima del de Internet y compuesto los los elementos software encargados de crear el canal de comunicación, descomponer el mensaje en paquetes y gestionar su trasmisión entre receptor y emisor.

- Protocolo **TCP** (orientado a conexión): Garantiza que los datos no se pierdan, garantiza que los mensajes llegaran en orden y es un **protocolo orientado a conexión** (es decir, primero se establece la conexión hasta que una de las dos aplicaciones la cierre, luego se envía el mensaje, y por último se cierra la conexión).

- Protocolo **UDP** (No orientado a conexión): No garantiza que los mensajes lleguen siempre, no garantiza que los mensajes lleguen en el mismo orden que fueron enviados, los mensaje no pueden exceder de los 64KB, los mensajes en UDP se llaman datagramas, y es un **protocolo no orientado a conexión** (es decir, es más rápido que el TCP ya que se crea un canal de comunicación independiente para cada mensaje y por lo tanto no hay establecimiento de conexión ni cierre)

4.- Nivel de Aplicación: Nivel más alto y esta compuesto por las aplicaciones que forman el sistema distribuido.

SOCKETS:

Proporcionan un mecanismo de abstracción de la pila de protocolos, y ofrecen una interfaz sencilla para que las diferentes aplicaciones de un sistema distribuido puedan intercambiar mensajes.

Un socket enchufe es el extremo de un canal de comunicación.

Para que dos aplicaciones puedan enviar y recibir mensajes entre si necesitan crear sus respectivos sockets y conectarlos entre sí para que de esta forma se creé una tubería privada.

Tipos de SOCKETS:

1.-SOCKETS Stream o SOCKETS Tcp: Se utiliza para comunicarse siempre con el mismo receptor manteniendo el canal de comunicación abierto hasta que se termine la conexión, la comunicación se realiza por medio del protocolo TCP. Los procesos que se van a comunicar deben establecer antes una conexión mediante un **stream** (secuencia ordenada de unidades de información que puede fluir en dos direcciones). Una de las aplicaciones debe ejercer el papel e proceso servidos y otra de proceso cliente, de esta forma el proceso servidor crea el socket y se pone a la espera a que el cliente se conecte.

En los Sockets TCPS las clases a las que pertenecen los objetos a transmitir entre las aplicaciones deben implementar la interfaz Serializable

Operaciones desde el proceso cliente:

1 – Creación del socket: Se crea y se le asigna un puerto, y se le llama socket cliente.
2 – Conexión: Se localiza el sockect del proceso servidor y se crea el canal de comunicación que une ambos procesos.
3 – Envió y Recepción del mensaje: El proceso cliente puede realizar las operaciones sobre el socket.
4 – Cierre de la conexión: El proceso cliente puede cerrar su socket si desea terminar la comunicación.

Operaciones desde el proceso servidor:

1 – Creación del socket: Se crea y se le asigna un puerto, y se le llama socket servidor.
2 – Asignación de dirección y puerto: La operación bind asigna una dirección IP y un número de puerto al socket.
3 – Escuchar: Se configura en la operación listen que se escuche por el puerto asignado.
4 – Aceptación de conexiones: En la operación accept se crea un nuevo socket dentro del proceso servidor el cual establece un canal de comunicación privado con el cliente dejando el socket servidor libre a la espera de nuevas conexiones.
5 – Envió y Recepción de mensajes: El proceso servidor puede realizar operaciones sobre su nuevo socket.
6 – Cierre de la conexión: En la operación close el proceso servidor puede cerrar su nuevo socket si desea terminar la conexión. El socket servidor sigue disponible a la escucha de nuevas conexiones.

2.-SOCKETS DATAGRAM: No son orientados a conexión ya que un mismo socket puede usarse para enviar mensajes a distintos receptores. Se usa un canal temporal para cada envió, no son fiables, no garantiza el orden de entrega de los mensajes y hacen uso del protocolo UDP cuando son usados sobre la pila de protocolos IP. No se diferencia el papel del proceso servidor y del cliente.

Secuencia de operaciones desde cualquier proceso que use sockets datagram:

1 – Creación del socket.

2 – Asignación de dirección y puerto: La operación bind es optativa, solo debe realizarse se se desea usar el socket para recibir mensajes.
3 – Envió y Recepción de mensajes: La operación send enviá el datagrama y necesita que se le especifique un dirección IP y un puerto, y la operación receive recibe el datagrama.
4 – Cierre de la conexión: Se pude cerrar el socket mediante la operación close.

Direcciones y puertos:

Existen dos versiones del protocolo IPv4 e Ipv6.

En IPv4 está formada por secuencias de 32 bits, llamadas palabras y cada palabra está dividida en 4 grupos de 8 bits.

Dirección IP fija: Se asignan permanentemente a un dispositivo y son útiles para servidores.
Dirección IP dinámica: Se asigna temporalmente a un dispositivo ya que se establece cada vez que la máquina se conecta.

Métodos:

Métodos clase Socket

Socket()	Socket	Crea un socket stream cliente.
Socket(String Host,int port)	Socket	Crea un socket y lo conecta al número de puerto y al nombre de host especificados .
Socket(InetAddress address, int port)	Socket	Crea un socket y lo conecta al puerto y dirección IP especificados.
connect(SocketAddress adr)	void	Establece la conexión con la dirección y puerto de destino, con la aplicación servidor .
getInputStream()	InputStream	Flujo de entrada que se usa para realizar operaciones de lectura de bytes para leer desde el socket.
getOutputStream()	OutputStream	Flujo de salida, que se usa para realizar operaciones de escritura de bytes en el socket .
getInetAddress()	InetAddress	Devuelve la dirección IP a la que el socket está conectado .
getLocalPort()	int	Devuelve el puerto local al que está enlazado el socket .
getPort()	int	Devuelve el puerto remoto al que está conectado el socket .
close()	void	Cierra el socket .

Métodos clase ServerSocket

ServerSocket()	ServerSocket	Crea un socket stream servidor.
ServerSocket(int port, int maximo)	ServerSocket	Crea un socket servidor y lo enlaza al puerto local especificado, el parámetro máximo indica el número máximo de peticiones de conexión que se pueden mantener en la cola.
ServerSocket(String host, int port)	ServerSocket	Recibe la dirección IP y el puerto que se desea asignar al socket .
bind(SocketAddress bindpoint)	void	Asigna al socket creado una dirección y número de puerto determinado.
accept()	Socket	El proceso servidor escucha por el socket servidor, esperando conexiones por parte de clientes. Cuando llega una conexión devuelve un nuevo objeto de la clase Socket, crea el canal, este nuevo socket queda conectado al cliente.
getInputStream()	InputStream	Flujo de entrada que se usa para realizar operaciones de lectura de bytes sobre él, para leer del socket.
getOutputStream()	OutputStream	Flujo de salida, que se usa para realizar operaciones de escritura en él, escribir bytes en el socket
getLocalPort()	int	Devuelve el puerto local al que está enlazado el ServerSocket
close()	void	Cierra el socket

Clases para los Sockets TCP: ObjectInputStream y ObjectOutputStream.

Ejemplo:

```
//preparando flujo de salida para escribir en él objetos.

ObjectOutputStream outObjeto= new ObjectOutputStream(socket.getOutpuStream());

//preparando flujo de entrada para leer de él objetos

ObjectInputStream inObjeto= new ObjectInputStream(socket.getInputStream());
```

Clases para los Sockets UDP: DatagramPacket y DatagramSocket.

Ejemplo:

```
// Construimos datagrama a enviar indicando el host de destino y puerto

DatagramPacket envio= new DatagramPacket(mensaje, mensaje.lenght,destino, port);
DatagramSocket socket= new DatagramSocket(34567);
socket.send(envio);

// En el otro extremo, para recibir el datagrama

DatagramSocket socket= new DatagramSocket(12345);

//construimos datagrama a recibir

DatagramPacket recibo=new DatagramPacket(buffer, buffer.length);
 socket.receive(recibo); // recibo datagrama
int bytesrecibidos=recibo.getLength();
String paquete= new String(recibo.getData());
System.out.println("número de bytes recibidos: "+bytesrecibidos);
System.out.println ("contenido del paquete: "+paquete.trim());
System.out.println("Puerto origen del mensaje: "+recibo.getPort());
System.out.println("IP de origen: "+recibo.getAddress().getHostAddress());
System.out.println("Puerto destino del mensaje: "+socket.getLocalPort());
```

Métodos clase DatagramPacket

DatagramPacket(byte[] buf,int length)	DatagramPacket	Constructor para datagramas recibidos.
DatagramPacket(byte[] buf,int length, InetAddress addrss, int port)	DatagramPacket	Constructor para el envío de datagramas.
getAddress()	InetAddress	Devuelve la dirección IP del host al cual se envía el datagrama o del que se recibió el datagrama.
getData()	byte[]	Obtiene el mensaje contenido en el datagrama.
getLength()	int	Obtiene la longitud de los datos
getPort()	int	Obtiene el número de puerto de la máquina remota a la que se le va a enviar el datagrama o de la que se recibió el datagrama.
setAddress(InetAddress addr)		Estable la dirección IP de la máquina a la que se le envía el datagrama .
setData(byte[] buf)		Establece el buffer de datos para el paquete datagrama.
setLength(int length)		Almacena la longitud de este paquete
setPort(int port)		Almacena el número de puerto

		del host remoto al que este datagrama se eviará

Métodos clase DatagramSocket

DatagramSocket(int port, InetAddress ip)	DatagramaSocket	Construye un socket para datagramas y establece el puerto local y la dirección local a la que se va asociar el socket
receive(DatagramPacket paquete)		Recibe un DatagramPacket del socket y llena el paquete con los datos que recibe.
send(DatagramPacket paquete)		Envía un DatagramPacket a través del socket.
getLocalPort()	int	Devuelve el número de puerto en el host local al que el socket está enlazado, -1 si el socket está cerrado y 0 sino está enlazado a ningún puerto
getPort()	int	Devuelve el número de puerto al que está conectado el socket,-1 sino está conectado.
connect(InetAddress address, int port)		Conecta el socket a un puerto remoto y a una dirección IP concretos, el socket sólo podrá enviar y recibir mensajes a/desde esa dirección.
setSoTimeout(int timeout)		Permite establecer un tiempo de espera límite. Entonces el método receive se bloquea durante el tiempo fijado.
close()		Cierra el socket .

Métodos de la clase InetAddres

getLocalHost()	InetAddress	Devuelve un objeto InetAddress que representa la dirección IP de la máquina donde se está ejecutando el programa
getByName(String host)	InetAddress	Devuelve un objeto InetAddress que representa la dirección IP de la máquina que se especifica como parámetro (host).
getAllByName(String host)	InetAddress[]	Devuelve todas las direcciones IP que tenga asignada una máquina en particular.
getHostAddress()	String	Devuelve la dirección IP de un

		objeto InetAddress en forma de cadena.
getHostName()	String	Devuelve el nombre del host de un objeto InetAddress.
getCanonicalHostName()	String	Obtiene el nombre canónico completo de un objeto InetAddress.