FIEC

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

PROGRAMACIÓN APLICADA A LA AUTOMATIZACIÓN

M.Sc. Alexander Prieto León

Conferencia XIII

Unidad IV Comunicación y manejo de datos en automatización.

4.1 Mecanismo de Intercambio de datos a través de la red.

OBJETIVOS

Objetivos:

- Elaborar aplicaciones que incluyan comunicación con otros dispositivos y manejo de Base de Datos.

BIBLIOGRAFÍA

Bibliografía:

- Zhi Eng, Lee; Rischpater, Ray. Application Development with Qt Creator: Build cross-platform applications and GUIs using Qt 5 and C++, 3rd Edition. (2020). Packt Publishing. ISBN-10: 1789951755, ISBN-13: 978-1789951752.
- http://www.qtrac.eu/C++-GUI-Programming-with-Qt-4-1st-ed.zip
- Qt 5.14.2 Reference Documentation. Qt Creator Help.
- https://doc.qt.io/qtcreator/index.html
- https://doc.qt.io/qt-5/qtdesigner-manual.html

En la clase anterior:

- ¿De que clase se debe heredar para hacer un widget personalizado en Qt?
- ¿Qué se debe volver a implementar en una clase de un widget personalizado?
- ¿Con qué se dibuja la apariencia visual de los widgets en Qt?

INTRODUCCIÓN

- **¿Qué son las aplicaciones distribuidas?**

¿Qué son las aplicaciones distribuidas?

- Las aplicaciones distribuidas son aquellas que se dividen de modo que partes de la misma aplicación puedan ejecutarse en varias computadoras y plataformas. Hacen que dos o más computadoras trabajen con el objetivo de realizar un conjunto concertado de operaciones.

- Ejemplos: los navegadores de Internet, los programas de charla interactiva (chat), los programas para el correo electrónico, los juegos de computadora, las aplicaciones bancarias u otros servicios, y otras aplicaciones que se utilizan en entornos de red, Internet, o usan comunicación telefónica.

Protocolo TCP/IP:

- Entre los modelos para crear aplicaciones distribuidas encontramos al protocolo TCP/IP.
  - Este es un protocolo de comunicación que nos permitirá escribir aplicaciones que se comuniquen a través de una red.
  - TCP/IP brinda una capa de transporte entre aplicaciones pero no impone ningún tipo de arquitectura particular para la aplicación distribuida, lo cual permite que una capa o parte de la aplicación esté en una plataforma y la otra capa de la aplicación esté en otra (ejemplos de plataformas son los sistemas operativos como Windows, Linux, Unix, etc.).

Protocolo TCP/IP:

- El protocolo TCP no es determinístico en el tiempo de envío, o sea, puede variar en el tiempo de envío. Sin embargo, sí intenta garantizar la llegada de la información pues si un paquete o trama no llega, el mismo protocolo establece volver a enviarlo hasta que se confirme su recepción.

- En Qt también existe el protocolo UDP (buscar **QUdpSocket** en la ayuda de Qt) que, contrario al TCP, sí garantiza los tiempos de envío pero no garantiza la recepción de lo enviado.

Nota: En la actualidad se han creado nuevos protocolos que hacen una mezcla inteligente de los principios de ambos para una comunicación más efectiva en tiempo real.

Protocolo TCP/IP:

- El desarrollo y la rápida difusión del uso de Internet ha creado un medio donde la mayoría de las computadoras tienen algún tipo de acceso TCP/IP, lo cual simplifica notablemente la realización y posterior instalación de una aplicación distribuida con TCP/IP.

- Las aplicaciones que usan TCP/IP pueden ser aplicaciones distribuidas basadas en mensajes, como por ejemplo, las que utilizan protocolos de transmisión de hipertexto (HTTP), o pueden ser aplicaciones objeto distribuidas , como por ejemplo, las aplicaciones distribuidas de bases de datos que se comunican usando sockets de Windows.

- El método más básico para adicionar la funcionalidad del TCP/IP a una aplicación es usar **sockets cliente y/o servidor** como se verá más adelante en Qt.

Interconexión universal en Internet:

- Un **host** es una computadora que ejecuta aplicaciones que permiten realizar intercambios de información con otras computadoras. Host significa que hospeda o alberga. Lo que hospeda en este caso son las aplicaciones de este tipo y sus datos relacionados.

- Un **router** o enrutador, es un dispositivo que conecta redes de computadoras y por tanto puede encaminar correctamente la información que pasa a través de él (es como un policía dirigiendo el tránsito).

- Un sistema de comunicaciones proporciona servicio universal de comunicaciones si permite que cualquier **host se comunique con cualquier otro host**. Esto **requiere un método de identificación** de los host conectados al sistema que sea aceptado de forma global.

Interconexión universal en Internet:

- Los identificadores de host se clasifican en:
  - **nombres**: lo que un objeto es (preferido por los humanos)
  - **direcciones**: donde está (usados por las computadoras)
  - **rutas**: como se llega a él (usados por los enrutadores)

Direcciones IP:

- En Internet se emplea como **identificador de host** el término de **direcciones IP** (Internet Protocol)
- Cada host para conectarse a Internet y ser reconocida debe tener asociada una dirección IP, que es un número binario de 32 bits (4 bytes).
- Esta dirección IP de 32 bits, es agrupada en octetos y se representa a través de 4 números decimales separados por puntos.

Ej: 169.158.144.1

Interconexión universal en Internet:

- Los identificadores de host se clasifican en:
  - **nombres**: lo que un objeto es (preferido por los humanos)
  - **direcciones**: donde está (usados por las computadoras)
  - **rutas**: como se llega a él (usados por los enrutadores)

Direcciones IP (IPv4):

- En Internet se emplea como **identificador de host** el término de **direcciones IP** (Internet Protocol)
- Cada host para conectarse a Internet y ser reconocida debe tener asociada una dirección IP, que es un número binario de 32 bits (4 bytes).
- Esta dirección IP de 32 bits, es agrupada en octetos y se representa a través de 4 números decimales separados por puntos.

Ej: 169.158.144.1

Componentes de Direcciones IP:

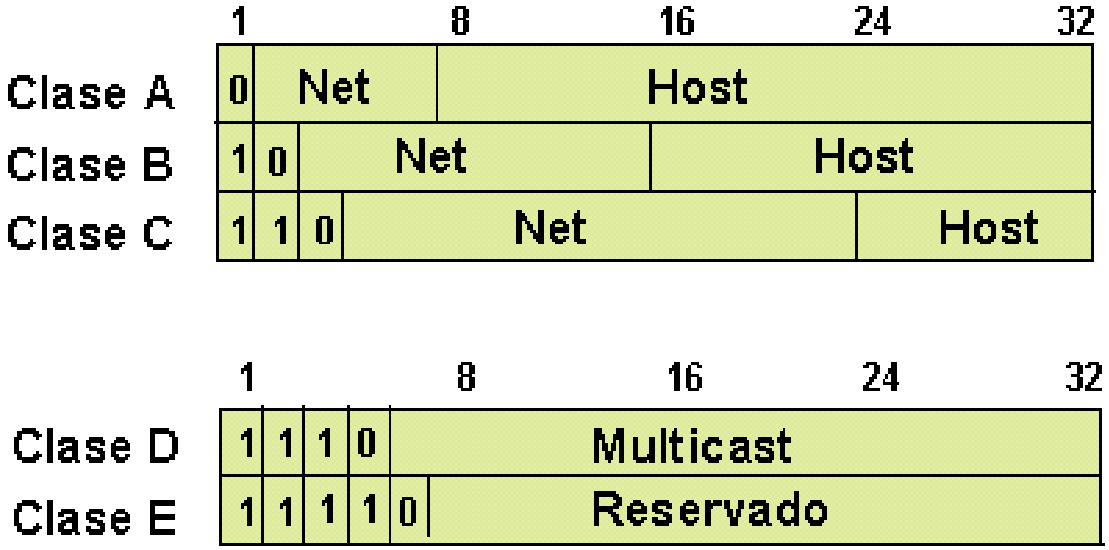
- Cada dirección es un par (**netid, hostid**), donde netid identifica una red y hostid identifica un host sobre esa red. Con esto, una dirección IP codifica a una red y a un host sobre esa red. **No especifica a una computadora en particular, sino una conexión a la red.**

- Las host multi-homed (que tiene varias direcciones o "sitios" de Internet) y los routers requieren de varias direcciones IP pues cada dirección corresponde con una conexión de ese dispositivo a la red.

Componentes de Direcciones IP(IPv4):

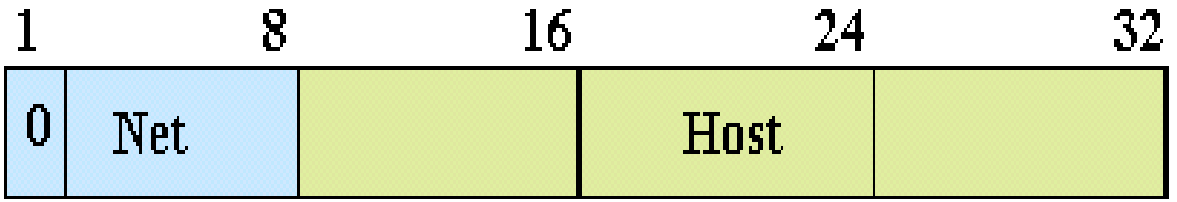
- Existen varias formas normalizadas de hacer esta división de direcciones. En dependencia de esto se han establecido varias clases de direcciones.

Clases de Direcciones en Internet



Componentes de Direcciones IP(IPv4):

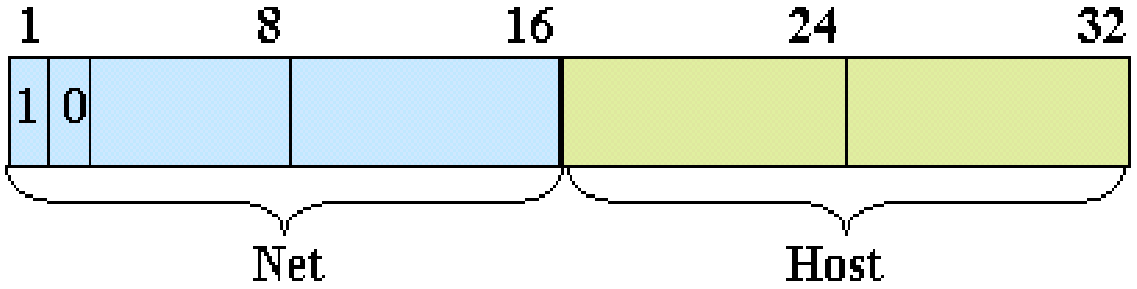
- Direcciones para Clase A:



- Se usan para redes grandes (cada una con alrededor de 2^{24} (16,777,216) hosts).
- Forma general de la dirección : [1 a 127] . x . x . x

Componentes de Direcciones IP(IPv4):

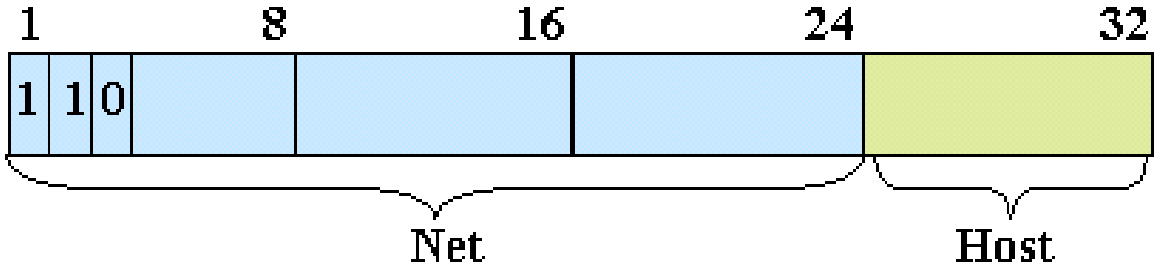
- Direcciones para Clase B:



- Se usa para redes de dimensiones intermedias de hasta 2^{16} (65,536) hosts. Como tenemos 14 bits disponibles para Net, entonces tendremos 2^{14} = 16,384 redes.
- Forma general de la dirección : **[128 a 191] . x . x . x**

Componentes de Direcciones IP(IPv4):

- Direcciones para Clase C:



- Se usa para redes de pocos hosts, menos de 2^8 (max 254) host. 2^{21} redes = 2'097,152 redes.

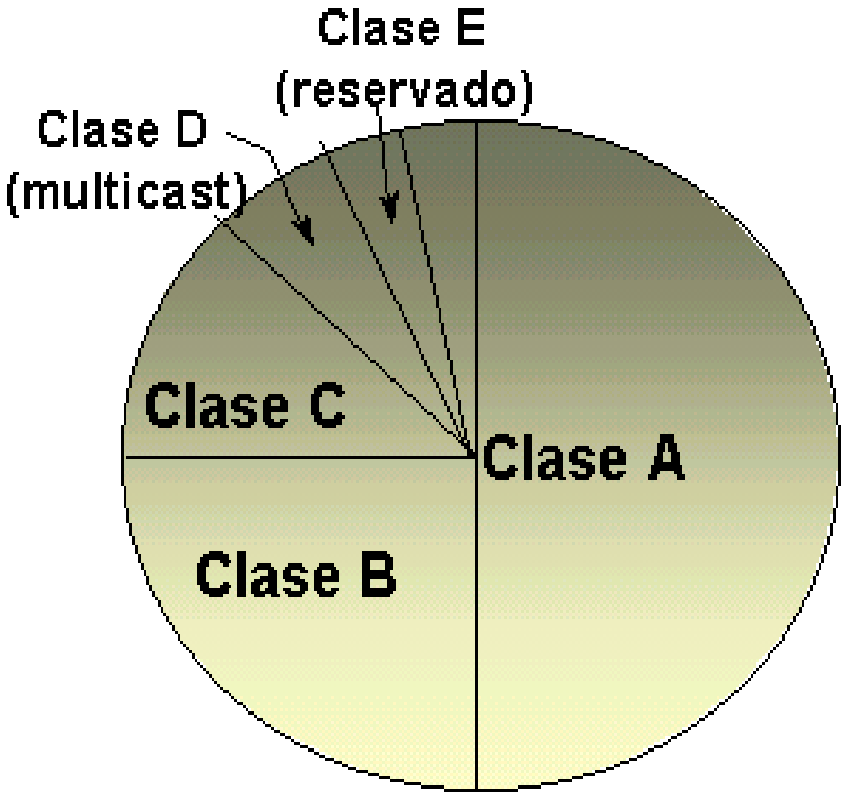
- Forma general de la dirección : **[192 a 223] . x . x . x**

Componentes de Direcciones IP(IPv4):

- Clases D y E:

❖ Clase D: Usada para definir grupos de hosts. (Direcciones Multicast)

❖ Clase E: Direcciones reservadas para uso posterior.



Componentes de Direcciones IP(IPv4):

- Máscaras de Subred.

Mácaras de Subred según Clases de Direcciones en Internet

Class A: 1 a 126

Network Mask 255.0.0.0

Network Address

0xxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
11111111 00000000 00000000 00000000

Class B: 128 a 191

Network Mask 255.255.0.0

Network Address

10xxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
11111111 11111111 00000000 00000000

Class C: 192 a 223

Network Mask 255.255.255.0

Network Address

110xxxxx xxxxxxxx xxxxxxxx xxxxxxxx
11111111 11111111 11111111 00000000

Componentes de Direcciones IP(IPv4):

- Ejemplo: Una red está compuesta por dispositivos y un enrutador:
  - Dirección de los dispositivos (generalmente computadoras, también pueden ser impresoras, etc.):
    - 192.168.4.3 (computadora 1)
    - 192.168.4.4 (computadora 2)
    - 192.168.4.5 (computadora 3)
    - 192.168.4.6 (computadora 4)
    - 192.168.4.7 (computadora 5)
    - 192.168.4.8 (computadora 6)
  - Enrutador:
    - 192.168.4.1 (dirección IP privada LAN), para comunicarse con la red local.
    - dirección IP pública (WAN, Internet), para comunicarse con otra red, generalmente asignada automáticamente por Dynamic Host Configuration Protocol (DHCP), aunque puede ser fijada por el proveedor de servicios de Internet (ISP).

Componentes de Direcciones IP(IPv4):

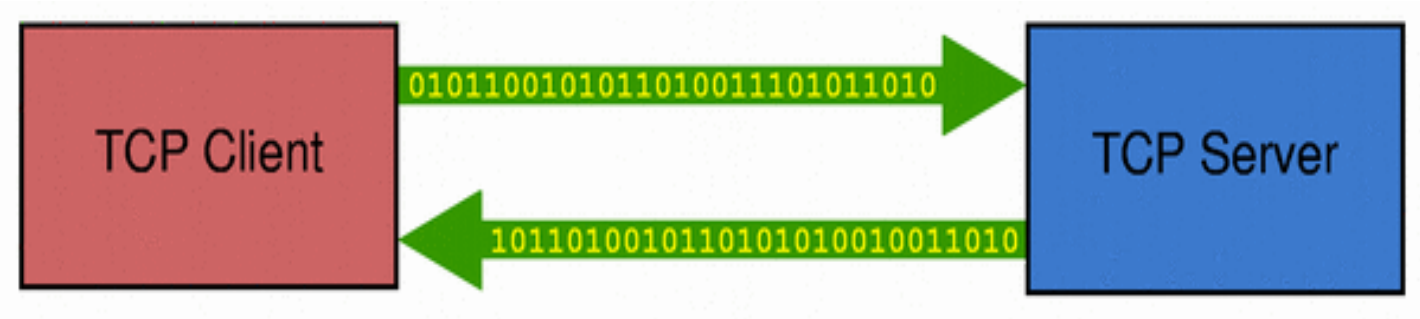
- Ejemplo: Una red está compuesta por dispositivos y un enrutador:
  - Máscara de subred:
    - 255.255.255.0
  - Se pueden usar las direcciones IP desde 192.168.4.1 hasta 192.168.4.254.
  - Las direcciones 192.168.4.0 y 192.168.4.255 están reservadas para usos especiales (dirección de red y dirección de broadcast de la red respectivamente).

Componentes de Direcciones IP(IPv4):

- Dirección de Loopback
  - Es la dirección que se usa para la realización de pruebas del TCP/IP y para la comunicación de los procesos internos en una máquina. Al usar esta dirección, lo que ocurre es que los datos regresan a la computadora origen sin generar tráfico en ninguna red.
  - La dirección de loopback es: 127.0.0.0 (Tipo A)
- Se especifica que:
  - 1- Un paquete enviado a esta dirección de red nunca debe aparecer en ninguna red.
  - 2- Un host o un router nunca deben difundir información de ruteo o de accesibilidad para el número de red 127, pues no es una dirección de red.

Usando TCP:

- TCP (Protocolo de control de transmisión) es un protocolo de red de bajo nivel usado por la mayoría de los protocolos de internet, incluyendo HTTP y FTP, para transferir los datos.
- Para que dos aplicaciones se comuniquen usando el protocolo TCP/IP una debe ser servidor y la otra cliente. Una aplicación servidor es aquella que permite que varias aplicaciones se conecten a ella para solicitar cualquier tipo de servicio. Una aplicación cliente es aquella que inicia la comunicación con otra (conocida como servidor).:



Clases asociadas a la comunicación TCP/IP en Qt:

QObject
↑
QIODevice
↑
QAbstractSocket
QHostAddress peerAddress()const
void setPeerAddress(const QHostAddress& address)
qint16 peerPort() const
void setPeerPort(qint16 Port)
qint64 bytesAvailable()const
void connectToHost(const QHostAddress& address, qint16 port, OpenMode = ReadWrite)
bool waitForConnect(int msecs = 30000)
qint64 readData(char* data, qint64 maxSize)
qint64 writeData(char* data, qint64 size)
Señales:
void connected()
void disconnected()
↑
QTcpSocket

QObject
↑
QTcpServer
bool listen(const QHostAddress & address = QHostAddress::Any, quint16 port = 0)
QTcpSocket * nextPendingConnection ()
Int maxPendingConnections () const
void close()
bool waitForNewConnection (int msec = 0, bool * timedOut = 0)
Señales
void newConnection()

Usando TCP con QTcpSocket y QTcpServer:

- La clase **QTcpSocket** provee una interfaz para TCP. Una conexión TCP debe ser establecida hacia un host remoto y un puerto antes de que la transferencia de cualquier dato comience. Una vez que la conexión ha sido establecida la dirección IP y el puerto del servidor están disponibles a través de **QTcpSocket::peerAddress()** y **QtcpSocket::peerPort()**. En cualquier instante el servidor puede cerrar la conexión y la transferencia de datos terminar inmediatamente.

- **QTcpSocket** trabaja asincrónicamente y emite señales para reportar el cambio de estado y errores.

- Los datos se pueden enviar al servidor a través del socket mediante **QTcpSocket::write(…)** y **QTcpSocket::writeData(…)** y leer con **QTcpSocket::read(…)** y **QTcpSocket::readData(…) QTcpSocket** representa dos flujos de datos independientes: uno para lectura y otro para escritura. Como **QTcpSocket** hereda de de **QIODevice** este puede usarse con **QTextStream** y **QDataStream.** Cuando se van a leer los datos desde **QTcpSocket** hay que estar seguro que la cantidad exacta de estos están disponibles llamando a la función **QTcpSocket::byteAvailable().**

Usando TCP con QTcpSocket y QTcpServer:

- Cuando se quieren implementar aplicaciones servidoras se usa la clase **QTcpServer**. Llamando a la función **QTcpServer::listen()** se activa al servidor.

- Cuando una conexión se establece se emite la señal **QTcpServer::newConnection().**

- En el slot al que se conecte la señal llamar a **QTcpServer::nextPendingConnection()** para aceptar la conexión y usar el **QTcpSocket** retornado para establecer la comunicación con el cliente.

Implementación TCP con QTcpSocket y QTcpServer:

- A continuación se tratarán los principales aspectos que debemos tener en cuenta en Qt para desarrollar una aplicación que haga uso de la comunicación TCP/IP. Ello se llevará a cabo a través de varios ejemplos de código.

Implementación TCP con QTcpSocket y QTcpServer:

El fichero de proyecto (.pro)

- Cualquier aplicación en Qt que haga uso del módulo de red debe incluir en el fichero del proyecto la siguiente línea.

```
QT += network
```

- Si su aplicación es visual probablemente ya tenga en el fichero .pro lo siguiente:

```
QT += core gui
```

- Por lo que si queremos incluir el módulo de red debe quedar de la siguiente manera:

```
QT += core gui, network
```

- Esto es importante pues de no hacerlo la aplicación no reconocería las clases de Qt que sirven para trabajar con la red, incluso aunque hayan incluido sus librerías (.h).

Implementación TCP con QTcpSocket y QTcpServer:

Conectando un cliente a un servidor

- Para que un cliente se conecte a un servidor es necesario conocer la dirección IP del servidor y por cuál puerto el servidor brinda el servicio que el cliente quiere. Teniendo eso en cuenta veamos que hay que hacer.

- 1. Incluir las librerías que contienen las clases que usaremos

```
#include <QTcpSocket>
#include <QHostAddress>
```

- 2. Crear un objeto cliente de tipo **QTcpSocket**:

```
QTcpSocket* client;
```

- 3. Establecer la conexión

```
client->connectToHost(QHostAddress(IP) ,Port);
```

Implementación TCP con QTcpSocket y QTcpServer:

Conectando un cliente a un servidor

- En este caso suponemos que la IP es un **QString** que contiene la dirección IP, p.ej: 127.0.0.1 y la variable Port es un entero que contiene el puerto por el cual se va a dar el servicio, p.ej: 965.

- Una vez dado estos tres pasos se le ha solicitado una conexión a la aplicación servidora, que se encuentra en la dirección IP. Para que la conexión se pueda establecer se necesita que ocurran 2 cosas: que la aplicación servidora se encuentre activa y que acepte la solicitud de conexión.

- Si la conexión se pudo establecer entonces se emite la señal **conected()**, indicando que ya se estableció la conexión.

Implementación TCP con QTcpSocket y QTcpServer:

Conectando un cliente a un servidor

- Otra forma de esperar a que se establezca la conexión es llamando a la función:
  bool **QAbstractSocket::waitForConnected**(int msecs = 30000)

- la cual espera a que se establezca la conexión un tiempo máximo que se le pasa como parámetro. Devuelve verdadero si se establece la conexión, falso en caso contrario.

- Ejemplo:

```
client->connectToHost(QHostAddress(IP) ,Port);
if(client->waitForConnected(1000))
{
    // estamos conectados..
}
```

Implementación TCP con QTcpSocket y QTcpServer:

- Activando el servidor.

1. Incluir las librerías que contienen las clases que usaremos

```
#include <QTcpServer>
```

2. Crear un objeto de tipo **QTcpServer**:

```
QTcpServer *tcpServer;
```

3. Activar el servidor para esperar por las conexiones

```
QHostAddress IP;
int Port;
.....
tcpServer->listen(IP,Port);
```

Implementación TCP con QTcpSocket y QTcpServer:

- Activando el servidor.

  - En este caso la variable IP es la dirección IP de la PC donde se encuentra ejecutándose el servidor y la variable Port es el puerto por el que el servidor va a brindar el servicio.

  - La función **listen**(..) se encarga de activar el servidor y esperar por solicitudes de conexiones. Esta función devuelve verdadero si se pudo activar el servidor y falso en caso contrario. Dicha información puede utilizarse para el chequeo de errores.

  - Hasta este punto lo único que se ha hecho es activar el servidor, pero aún no se ha aceptado ninguna solicitud de conexión por parte de un cliente. Veamos que sucede cuando un cliente solicita conectarse al servidor.

Implementación TCP con QTcpSocket y QTcpServer:

- Servidor aceptando solicitud de conexión de un cliente.
  - Cuando un cliente solicita conectarse a un servidor el objeto de la clase **QTcpServer** emite la señal **newConnection().** Por lo tanto, conectando un slot a dicha señal se puede procesar la solicitud de conexión. Los pasos que hay que hacer para aceptar la conexión se explican a continuación. Supongamos que la siguiente función es el slot que se activa cuando se emite la señal **newConnection().**

```
QTcpSocket *clientConnection;
...
void XXXX::newClient()
{
   clientConnection = tcpServer->nextPendingConnection();
....
}
```

  - En este caso para aceptar la solicitud de conexión solo basta con llamar a la función **nextPendingConnection()** de la clase **QTcpServer.** Dicha función devuelve la dirección de un objeto de tipo **QTcpSockect** que caracteriza la conexión con el cliente. A través del objeto de **QTcpSockect** es que se puede comunicar el servidor con el cliente.

Implementación TCP con QTcpSocket y QTcpServer:

- Enviando datos cliente-servidor y servidor-cliente.

  - Para el envío de información entre el cliente y el servidor lo normal es que el cliente sea quien comience la comunicación y el servidor responda. No obstante, el servidor puede comenzar la comunicación si, como se explicó en la sección anterior, se retuvo al objeto que caracteriza la conexión del cliente.

  - En cualquier caso, los pasos para el envío de datos son los mismos y siempre es usando las funciones de la clase **QTcpSocket**.

Implementación TCP con QTcpSocket y QTcpServer:

- Cliente enviando datos.
  - Para que el cliente envíe datos solo basta utilizar la función **write(...)** de la clase **QIODevice** que es clase base de **QTcpSocket. OJO:** esta función debe ser revisada por ustedes pues tiene varias homonimias. Aquí solo veremos una de ellas:

```
qint64 QIODevice::write(const char* data, qint64 maxSize)
```

  - Ejemplo.

```
QTcpSocket* client;

...

QString data = "Hola";

client->write(data.toAscii());
```

Implementación TCP con QTcpSocket y QTcpServer:

- Servidor recibiendo datos.
  - Cuando un cliente envía datos al servidor y llega a este, el objeto de la clase **QTcpSocket** que caracteriza la conexión del cliente en el servidor emite la señal **readyRead().** Esta señal es necesaria conectarla a un slot para poder recibir los datos que envía el cliente. **OJO: la señal solo podrá ser conectada a un slot después que se haya aceptado la conexión del cliente y el objeto de QTcpSocket que representa dicho cliente haya sido retenido.**
  - Ejemplo.

```
....
clientConnection = tcpServer->nextPendingConnection();
connect(clientConnection, SIGNAL(readyRead()), this, SLOT(leyendoDatos()));
...
void XXXX::leyendoDatos(){
        QTcpSocket *clientConnection = (QTcpSocket*)sender();
        qint64 bytesCount = clientConnection->bytesAvailable();
        QString  dato;
        dato = QString::fromUtf8(clientConnection->read(bytesCount));
...}
```

Implementación TCP con QTcpSocket y QTcpServer:

- Servidor recibiendo datos.
  - El slot leyendo datos que se implementa en el servidor. Para leer los datos lo fundamental es utilizar la función **read(...)** de la clase **QIODevice**.
  - Para que el servidor envíe datos a un cliente hay que realizar lo mismo, solo tener en cuenta que hay que usar el objeto de **QTcpSocket** que caracteriza al cliente que queremos enviarle datos. Puede que existan más de un cliente conectado al servidor por lo tanto habrá más de un objeto de **QTcpSocket** retenido. Además recordar que entonces el slot **leyendoDatos()** habrá que implementarlo en el cliente.

CONCLUSIONES

- De esta manera ya conocen los aspectos básicos a tener en cuenta para desarrollar una aplicación cliente y otra servidor. No obstante, estas no son las únicas cosas que se pueden hacer en este tipo de aplicaciones, quedan muchas otras que ustedes pueden estudiar por su cuenta.

- En el laboratorio…