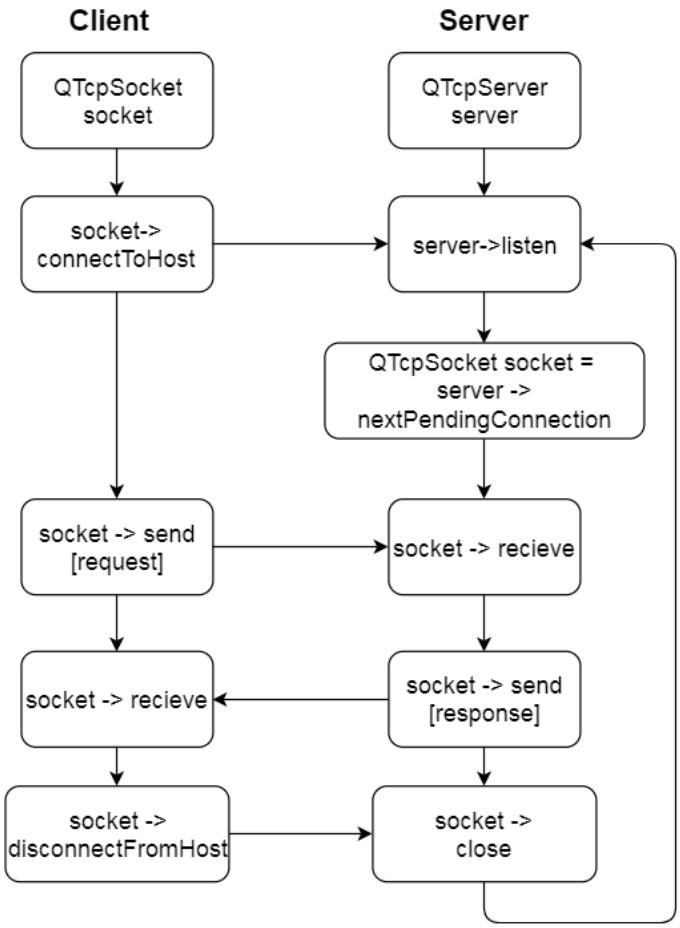
14

Klijent-server komunikacija

14.1. TCP/IP

TCP/IP (eng. *Transmission Control Protocol/Internet Protocol*) je skup protokola koji služe da obezbede komunikaciju između umreženih uređaja. Ovi protokoli se koriste za razmenu podataka putem interneta kao i u lokalnoj mreži. TCP protokol definiše način na koji aplikacije stvaraju kanale za komunikaciju među umreženim računarima, kao i način na koji se poruka deli na manje delove (pakete) pre nego što se krene sa slanjem. IP protokol definiše adresiranje i rutiranje svakog paketa kako bi stigao na pravu lokaciju. Svaki ruter u mreži proverava adresu paketa kako bi ga preusmerio odgovarajućem uređaju. Iako postoje druge vrste protokola kao što je UDP/IP (koji se koristi za npr. *stream*-ovanje, prenos videa u realnom vremenu i drugim slučajevima kada nije obavezan prenos podataka bez gubitaka), u tipičnoj internet komunikaciji je TCP/IP najzastupljeniji zato što obezbeđuje siguran prenos podataka.

Većina komunikacija putem TCP/IP protokola se odvija pomoću *klijent-server* modela. To su dve aplikacije na odvojenim uređajima koji će međusobno komunicirati. Jedna od te dve aplikacije (klijent) inicira komunikaciju sa drugom aplikacijom (serverom). Kao inicijator komunikacije klijent mora da zna osnovne podatke o serveru kao što je njegova IP adresa, dok server ne mora znati ništa o klijentu pre uspostavljanja komunikacije, već samo čeka da mu se neki klijent prvi obrati. Kako bi se uspostavila komunikacija, klijent i server moraju napraviti sopstveni *socket* (jedan kraj komunikacionog kanala). Nakon što se uspostavi veza, server i klijent mogu međusobno slati podatke u oba smera (*full duplex*).



14.2. Klasa QTcpSocket

Kako bi se u Qt razvojnom okruženju dobio pristup klasama za internet komunikaciju, u project datoteci (*.pro*) je potrebno dodati sledeću liniju:

```
QT += network
```

QTcpSocket klasa pravi objekat koji predstavlja jedan kraj komunikacionog kanala i inicijalno je u nepovezanom stanju. Kako bi se uspostavio komunikacini kanal sa klijentske strane potrebno je pozvati metodu ***connectToHost*** *(const QHostAddress &, quint16 port, QIODevice::OpenMode).* QHostAddress je IP adresa uređaja, a *port* je

broj porta na koji se klijent povezuje, dok je *QIODevice::OpenMode* tip uređaja i može primiti jedan od nabrojivih tipova: *ReadOnly, WriteOnly, ReadWrite...*

Kada se uspostavi konekcija sa serverom, QTcpSocket objekat emituje signal ***connected().*** *QTcpSocket* poseduje metodu bool **waitForConnected**(int msec=3000) koja čeka na signal *connected()* maksimalno *msec* milisekundi nakon čega se program nastavlja izvršavati. Uspešnim povezivanjem metoda vraća vrednost *true*, a ukoliko se uređaj ne uspe povezati za dati vremenski period, povratna vrednost je *false*.

Jednostavno uspostavljanje konekcije sa klijentske strane bi izgledalo kao u sledećem kodnom fragmentu:

```
QTcpSocket *socket = new QTcpSocket();
socket -> connectToHost("127.0.0.1",8080);
if(socket->waitForConnected(1000))
      //uspesno uspostavljena veza
else
      //problem pri uspostavljanju veze
```

Umesto lokalne adrese 127.0.0.1 pokazalo se da je potrebno upisati konkretnu IP adresu servera (npr. 10.1.220.234).

Slanje podataka preko kanala se vrši jednom od datih *write* metoda. Podaci se upisuju u odlazni bafer nakon čega se bajt po bajt šalju pomoću internet protokola. Kada se svi podaci iz odlaznog bafera pošalju, generiše se **bytesWritten**(qint64 *bytes*) signal. QTcpSocket poseduje metodu za čekanje na ovaj signal: bool **waitForBytesWritten** (int msec=3000).

```
qint64 write(const char *data, qint64 maxSize)
qint64 write(const char *data)
qint64 write(const QByteArray &byteArray)

Jednostavan primer slanja podataka:
      socket->write("Hello from client");
      if(socket->waitForBytesWritten(1000))
            //uspešno poslata poruka
      else
            //neuspešno poslata poruka
```

Kada pristignu novi podaci u prijemni bafer generiše se ***readyRead()*** signal. QTcpSocket objekat poseduje metodu za čekanje na ovaj signal ***waitForReadyRead****(int msec=3000)*. U svakom trenutku je moguće proveriti broj bajtova u prijemnom baferu

pomoću metode *qint64* ***bytesAvailable()***.Nakon generisanja signala, podaci se mogu pročitati iz prijemnog bafera pomoću jedne od sledećih metoda:

```
qint64 read(char *data, qint64 maxSize)
QByteArray read(qint64 maxSize)
QByteArray readAll()
qint64 readLine(char *data, qint64 maxSize)
QByteArray readLine(qint64 maxSize = 0)
```

Jednostavan primer prijema podataka:

```
if(socket->waitForReadyRead(1000))
      QByteArray recieved = socket->readAll();
else
      //problem pri prijemu
```

Kada se završi sa razmenom podataka, potrebno je prekinuti vezu pomoću metode ***disconnectFromHost***(). Nakon što se pozove ova metoda, ukoliko nema (nepročitanih ili neposlatih) podataka u baferima, emitovaće se ***disconnected***() signal. Metoda bool ***waitForDisconnected*** (int msecs =1000) čeka na *disconnected* signal:

```
socket -> disonnectFromHost()
if(socket->state() == QTcpSocket::ConnectedState)
socket->waitForDisconnected(1000))
// uredjaj se iskljucio sa veze
```

Za pregled svih metoda klase QTcpSocket kao i njihov detaljan opis,čitalac se upućuje na oficijalnu Qt dokumentaciju https://doc.qt.io/

14.3. Klasa QTcpServer

Ova klasa ima realizovane metode koje olakšavaju implementaciju serverske aplikacije. Prvo je potrebno napraviti pokazivač na objekat ove klase *QTcpServer* ****server*** *= new QTcpServer()* a zatim pozvati metodu *bool* ***listen*** *(const QHostAddress &address= QHostAddress::Any, quint16 port = 0).* Ova metoda govori serveru da čeka na konekciju od uređaja sa IP adresom i brojem porta prosleđenim putem parametara. Često se adresa postavlja na QHostAddress::Any kako bi server primio konekciju od uređaja sa bilo kojom IP adresom.

Primer jednostavnog servera koji prima konekcije sa svih IP adresa na portu 1234:

```
if(server->listen(QHostAddress::Any,1234)
      //Server čeka na konekciju
else
      //Server nije uspešno pokrenut
```

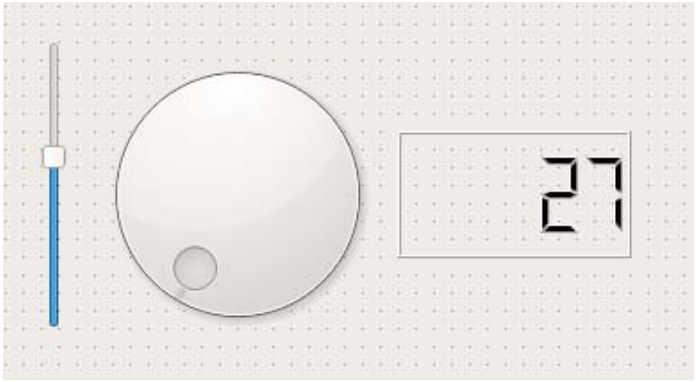
Dodatna stvar po kojoj se server razlikuje od klijenta je način uspostavljanja konekcije. Potrebno je povezati signal **newConnection**() na proizvoljan slot, a zatim pozvati metodu **nextPendingConnection**() koja će vratiti pokazivač na objekat tipa QTcpSocket koji je već povezan sa klijentom.

```
connect(server,SIGNAL(newConnection()),this,SLOT(newConnect
ion()));

void MyClass::newConnection()
{
    QTcpSocket *socket = server -> nextPendingConnection();
    //komunikacija sa klijentom pomoću objekta socket
}
```

Zadatak

YL-40 pločica se nalazi na RPi serveru koji je povezan na lokalnu mrežu putem ethernet kabla. Napraviti GUI i napisati program koji prikazuje vrednosti sva tri senzora (osvetljaj, temperatura i napon na trimer potenciometru). Koristiti klase QDial za napon trimer pot. (samo prikaz), QSlider za osvetljaj (vertikalni, samo prikaz položaja) i QLCDNumber. Za dobijanje informacija sa YL-40 pločice je potrebno kontaktirati server. Nakon uspostavljanja komunikacije sa serverom potrebno je poslati jednu od tri instrukcije "temperatura", "osvetljaj" ili "potenciometar" na koje će server odgovoriti tako što će pročitati vrednost sa odgovarajućeg AD kanala i vratiti vrednost u opsegu 0-255 u vidu stringa.
Potrebno je zatvoriti komunikacioni kanal nakon svake razmene.



Slika 14.1. Detalji GUI