Modje Niroo Co: Programing Assignment

Due on Tuesday, October 06 2020

Farshad Gholami

Project 1

Implement a simple SCADA system using Qt

The simplified SCADA system is composed of three sections:

The input module: This module has a simple graphics with several input boxes which are corresponding to each element of the elements represented in the graphic module.

Server: The server receives the values with a special protocol from the input module and sends them to the graphics module for display after processing. Selection of the protocol between the input module and the server as desired.

Graphical module: A simple graphic has to be designed by using Qt and it must have several elements as shown in Figure 1. The squares in the 1 represent the breaker or power-switch and the three gauges to show Voltage, active and reactive power of the transformer. The blue segments are represented a 63 kV voltage level and the yellow segments represent a 20 kV voltage level. To describe the breakers, only the values of zero and one are valid, and the graphic module displays the closed state by receiving 1 and the open state by receiving 0 value for the breakers. Gauges should be also able to show any values with decimals precision after introducing the graphical module.

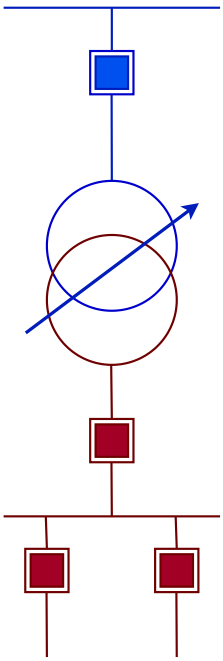

Figure 1: Graphical module representation.

Implement a simple SCADA system using Qt: Server/main.cpp

```
#include <QCoreApplication>
#include "mytcpserver.h"
int main(int argc, char *argv[])
```

Project 1 continued on next page...

```
{
    QCoreApplication a(argc, argv);
    myTcpServer* server = new myTcpServer();

    return a.exec();
}
```

Implement a simple SCADA system using Qt: Server/mytcpserver.h

```
#ifndef MYTCPSERVER_H
#define MYTCPSERVER_H
#include <QDebug>
#include <QTcpServer>
#include<QTcpSocket>
#include <QObject>
#include <QFile>
#include <QJsonObject>
#include <QJsonArray>
#include <QJsonDocument>
#include <QtGlobal>
#include <QTextStream>



class myTcpServer : public QObject
{
    Q_OBJECT
public:
    explicit myTcpServer(QObject *parent = nullptr);
private:
    QTcpServer* myServer;
    QByteArray* myQJosnByetedInPut;
    QByteArray* myQJosnByetedOutPut;
    void prepareOutPut();
    ~myTcpServer();

signals:

public slots:
    void newConnection();
};

#endif // MYTCPSERVER_H
```

Implement a simple SCADA system using Qt: Server/mytcpserver.cpp

```
#include "mytcpserver.h"

myTcpServer::myTcpServer(QObject *parent) : QObject(parent)
{
```

Project 1 continued on next page...

```
    myServer = new QTcpServer(this);
    myQJosnByetedInPut= new QByteArray();
    myQJosnByetedOutPut= new QByteArray();
    connect(myServer,SIGNAL(newConnection()), this, SLOT(newConnection()));
    if(!myServer->listen(QHostAddress::Any, 8585))
    {
        qDebug() << "Server could not start!";
    }
    else
    {
        qDebug() << "Server started!";
    }
}

void myTcpServer::prepareOutPut()
{
    QFile fin("recivedDataFromServer.json");
    fin.open(QIODevice::ReadOnly);
    QString readJsonFile = fin.readAll();
    fin.close();
    QJsonDocument readJsonFileDoc = QJsonDocument::fromJson(readJsonFile.toUtf8());
    QJsonObject readJsonObj = readJsonFileDoc.object();
    int Switch_1_status = 0;
    int Switch_2_status = 0;
    int Switch_3_status = 0;
    int Switch_4_status = 0;
    if(readJsonObj.value("Switch_1").toVariant().value<bool>()==true){
        Switch_1_status=1;
    }
    if(readJsonObj.value("Switch_2").toVariant().value<bool>()==true){
        Switch_2_status=1;
    }
    if(readJsonObj.value("Switch_3").toVariant().value<bool>()==true){
        Switch_3_status=1;
    }
    if(readJsonObj.value("Switch_4").toVariant().value<bool>()==true){
        Switch_4_status=1;
    }
    double outPut1= Switch_1_status*1500.48+Switch_2_status*1500+Switch_3_status*250+45;
    double outPut2= Switch_2_status*1500.47+Switch_3_status*1400+23;
    double outPut3= Switch_4_status*900.51+Switch_3_status*1400+12;
    readJsonObj["outPut1"]=outPut1;
    readJsonObj["outPut2"]=outPut2;
    readJsonObj["outPut3"]=outPut3;
    *myQJosnByetedOutPut = QJsonDocument(readJsonObj).toJson();
    QFile outPutFile("serverOutPut.json");
    outPutFile.open(QIODevice::WriteOnly);
    outPutFile.write(*myQJosnByetedOutPut);
    outPutFile.close();
```

Project 1 continued on next page...

```
}

myTcpServer::~myTcpServer()
{
    delete myServer;
    delete myQJosnByetedInPut;
    delete myQJosnByetedOutPut;

}

void myTcpServer::newConnection()
{
    QTcpSocket* myServerSocket= myServer->nextPendingConnection();
    myServerSocket->waitForReadyRead(30000);
    *myQJosnByetedInPut = myServerSocket->readAll();

    QString str("Im HMI");
    QByteArray bytes = str.toLatin1();
    if(*myQJosnByetedInPut ==bytes ){
        prepareOutPut();
        myServerSocket->write(*myQJosnByetedOutPut);
        myServerSocket->flush();
        myServerSocket->waitForBytesWritten(3000);
    }else {
        QFile fin("recivedDataFromServer.json");
        fin.open(QIODevice::WriteOnly);
        fin.write(*myQJosnByetedInPut);
}

    myServerSocket->close();

}
```

Implement a simple SCADA system using Qt: Client/main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Implement a simple SCADA system using Qt: Client/mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
```

Project 1 continued on next page...

```
#include <QFile>
#include <QJsonObject>
#include <QJsonArray>
#include <QJsonDocument>
#include <QtGlobal>
#include <QTextStream>

#include <mysocket.h>



#include <QMainWindow>
#include <QString>
#include <QDebug>
namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    QByteArray* returningJson;
    mySocket* socket;
    quint16 PortNumber;
private slots:
    void reportConnectionStatus(bool);
    void reportSendingStatus(bool);

    void on_switch_1_toggled(bool checked);
    void on_switch_2_toggled(bool checked);
    void on_switch_3_toggled(bool checked);
    void on_switch_4_toggled(bool checked);

    void on_pushButtonSetPort_clicked();

private:
    Ui::MainWindow *ui;
    QString* portNumber_s;


    bool* switchStat1;
    bool* switchStat2;
```

Project 1 continued on next page...

```
    bool* switchStat3;
    bool* switchStat4;

    void prepareQJson();


};

#endif // MAINWINDOW_H
```

Implement a simple SCADA system using Qt: Client/mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QIntValidator>
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
   PortNumber=0;
   returningJson = new QByteArray();
   switchStat1 = new bool(false);
   switchStat2 = new bool(false);
   switchStat3 = new bool(false);
   switchStat4 = new bool(false);
   ui->setupUi(this);
   ui->lineEditPortNumber->setValidator( new QIntValidator(0, 99999, this) );
   ui->textEditReporter->setTextColor( QColor( "red" ) );
   ui->textEditReporter->append("At First you must set Port number.");
   ui->textEditReporter->setTextColor( QColor( "black" ) );
   socket = new mySocket(this);
   connect(socket, SIGNAL(connectionSignal(bool)), this, SLOT(reportConnectionStatus(bool)));
   connect(socket, SIGNAL(writenSignal(bool)), this, SLOT(reportSendingStatus(bool)));

}


MainWindow::~MainWindow()
{
    delete ui;
    delete returningJson  ;
    delete switchStat1   ;
    delete switchStat2   ;
    delete switchStat3   ;
    delete switchStat4   ;
    delete socket;
}

void MainWindow::reportConnectionStatus(bool _connectionStatus)
{
```

```
    if(_connectionStatus){
        ui->textEditReporter->setTextColor( QColor( "green" ) );
        ui->textEditReporter->append("The connection is successfully established.");
        ui->textEditReporter->setTextColor( QColor( "black" ) );
    }else {

        ui->textEditReporter->setTextColor( QColor( "red" ) );
        ui->textEditReporter->append("The connection is not successfully established.");
        ui->textEditReporter->setTextColor( QColor( "black" ) );
    }
}

void MainWindow::reportSendingStatus(bool _isWiten)
{
    if(_isWiten){
        ui->textEditReporter->setTextColor( QColor( "green" ) );
        ui->textEditReporter->append("The data is successfully written.");
        ui->textEditReporter->setTextColor( QColor( "black" ) );
    }else {

        ui->textEditReporter->setTextColor( QColor( "red" ) );
        ui->textEditReporter->append("The data is not successfully written.");
        ui->textEditReporter->setTextColor( QColor( "black" ) );
    }
}




void MainWindow::on_switch_1_toggled(bool checked)
{
    if(checked){
       *switchStat1 = true;
       ui->textEditReporter->append("Switch_1 satat is changed to \" Connected\". ");
    }else {
       *switchStat1=false;
       ui->textEditReporter->append("Switch_1 satat is changed to \" Disconnected\". ");
    }
    prepareQJson();
    socket->doConnect(PortNumber,returningJson);
}

void MainWindow::on_switch_2_toggled(bool checked)
{
    if(checked){
        *switchStat2 = true;
        ui->textEditReporter->append("Switch_2 satat is changed to \" Connected\". ");
    }else {
        *switchStat2=false;
```

```
        ui->textEditReporter->append("Switch_2 satat is changed to \" Disconnected\". ");
    }
    prepareQJson();
    socket->doConnect(PortNumber,returningJson);
}

void MainWindow::on_switch_3_toggled(bool checked)
{
    if(checked){
        *switchStat3 = true;
        ui->textEditReporter->append("Switch_3 satat is changed to \" Connected\". ");
    }else {
        *switchStat3=false;
        ui->textEditReporter->append("Switch_3 satat is changed to \" Disconnected\". ");
    }
    prepareQJson();
    socket->doConnect(PortNumber,returningJson);
}

void MainWindow::on_switch_4_toggled(bool checked)
{

    if(checked){
        *switchStat4 = true;
        ui->textEditReporter->append("Switch_4 satat is changed to \" Connected\". ");
    }else {
        *switchStat4=false;
        ui->textEditReporter->append("Switch_4 satat is changed to \" Disconnected\". ");
    }
    prepareQJson();
    socket->doConnect(PortNumber,returningJson);
}


void MainWindow::on_pushButtonSetPort_clicked()
{
    QString report = "Connecting o port "+ ui->lineEditPortNumber->text();
    PortNumber =  ui->lineEditPortNumber->text().QString::toUShort();
    ui->textEditReporter->append(report);
    prepareQJson();
    socket->doConnect(PortNumber,returningJson);


}

void MainWindow::prepareQJson()
{
    QJsonObject root;
    root["Switch_1"] = (*switchStat1);
```

Project 1 continued on next page...

```
    root["Switch_2"] = (*switchStat2);
    root["Switch_3"] = (*switchStat3);
    root["Switch_4"] = (*switchStat4);
    *returningJson= QJsonDocument(root).toJson();
    QFile outPutFile("clientSwitchsState.json");
    outPutFile.open(QIODevice::WriteOnly);
    outPutFile.write(*returningJson);
}
```

Implement a simple SCADA system using Qt: Client/mysocket.h

```
#ifndef MYSOCKET_H
#define MYSOCKET_H

#include <QObject>
#include <QTcpSocket>
#include<QDebug>
#include <QByteArray>
class mySocket : public QObject
{
    Q_OBJECT
public:
    explicit mySocket(QObject *parent = nullptr);
    void doConnect(quint16, QByteArray*);
    void writeMyJson();
    bool isConnected;
    ~mySocket();
private:
    QTcpSocket* myclientSoket;

signals:
    void connectionSignal( bool isConnected);
    void writenSignal(bool iswriten);
public slots:

};

#endif // MYSOCKET_H
```

Implement a simple SCADA system using Qt: Client/mysocket.cpp

```
#include "mysocket.h"

mySocket::mySocket(QObject *parent) : QObject(parent)
{
    myclientSoket= new QTcpSocket(this) ;
}

void mySocket::doConnect(quint16 _portNumber, QByteArray* _data)
{
```

Project 1 continued on next page...

```
    myclientSoket->connectToHost("127.0.0.1",_portNumber);

   if(myclientSoket->waitForConnected(1000)){
       isConnected =true;
       emit connectionSignal(isConnected);
       myclientSoket->write(*_data);
       if(myclientSoket->waitForBytesWritten(1000)){
        emit writenSignal(true);
       }else {
        emit writenSignal(false);
       }
   }else {
       isConnected =false;
       emit connectionSignal(isConnected);
   }
}

mySocket::~mySocket()
{
    delete myclientSoket;
}
```

Implement a simple SCADA system using Qt: HMI/main.cpp

```
#include "mainwindow.h"
#include <QApplication>
#include<mythread.h>
#include <QObject>
#include<QByteArray>
#include<QEvent>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MyThread* thread = new MyThread();
    MainWindow w;
    thread->start();
    w.show();
    while (true) {
        QCoreApplication::processEvents();
        w.checkData();
    }
    return 0;//a.exec();
}
```

Implement a simple SCADA system using Qt: HMI/mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
```

Project 1 continued on next page...

```
#include <QMainWindow>
#include <QDir>
namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    void checkData();
    void setData();
    bool sw_1_pre;
    bool sw_2_pre;
    bool sw_3_pre;
    bool sw_4_pre;
    double outPut1_pre;
    double outPut2_pre;
    double outPut3_pre;

    bool sw_1;
    bool sw_2;
    bool sw_3;
    bool sw_4;
    double outPut1;
    double outPut2;
    double outPut3;
    QDir directory;
    QString sefidPath;
    QString redPath;
    QString bluePath;
    QString backgroundPath;



private:
    Ui::MainWindow *ui;

};

#endif // MAINWINDOW_H
```

Implement a simple SCADA system using Qt: HMI/mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <myqsoket.h>
```

Project 1 continued on next page...

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    directory= QDir::current();
    sefidPath=directory.filePath("sefid.png");
    redPath=directory.filePath("redSwitch.png");
    bluePath=directory.filePath("blueSwitch.png");
    backgroundPath=directory.filePath("background.png");

    ui->setupUi(this);
    ui->label->setPixmap(QPixmap(backgroundPath));
    ui->lcdNumberkV->setDecMode();
    ui->lcdNumberMW->setDecMode();
    ui->lcdNumberMVA->setDecMode();
    ui->lcdNumberMW->setPalette(Qt::black);
    ui->lcdNumberkV->setPalette(Qt::black);
    ui->lcdNumberMVA->setPalette(Qt::black);
    sw_1_pre=false;
    sw_2_pre=false;
    sw_3_pre=false;
    sw_4_pre=false;
    outPut1_pre=0;
    outPut2_pre=0;
    outPut3_pre=0;

    sw_1=false;
    sw_2=false;
    sw_3=false;
    sw_4=false;
    outPut1=0;
    outPut2=0;
    outPut3=0;

}



MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::checkData()
{
    QFile fin("recivedDataFromHMI.json");
    fin.open(QIODevice::ReadOnly);
    QString readJsonFile = fin.readAll();
```

Project 1 continued on next page...

```
    fin.close();
    QJsonDocument readJsonFileDoc = QJsonDocument::fromJson(readJsonFile.toUtf8());
    QJsonObject readJsonObj = readJsonFileDoc.object();
    sw_1=readJsonObj.value("Switch_1").toVariant().value<bool>();
    sw_2=readJsonObj.value("Switch_2").toVariant().value<bool>();
    sw_3=readJsonObj.value("Switch_3").toVariant().value<bool>();
    sw_4=readJsonObj.value("Switch_4").toVariant().value<bool>();
    outPut1=readJsonObj.value("outPut1").toVariant().value<double>();
    outPut2=readJsonObj.value("outPut2").toVariant().value<double>();
    outPut3=readJsonObj.value("outPut3").toVariant().value<double>();
    if((sw_1!=sw_1_pre || sw_2!=sw_2_pre || sw_3!=sw_3_pre || sw_4!=sw_4_pre || outPut1!=outP
        setData();
        sw_1_pre=sw_1;
        sw_2_pre=sw_2;
        sw_3_pre=sw_3;
        sw_4_pre=sw_4;
        outPut1_pre=outPut1;
        outPut2_pre=outPut2;
        outPut3_pre=outPut3;
    }
}


void MainWindow::setData()
{

    if(sw_1){
        ui->label_SW1 ->setPixmap(QPixmap(redPath));
    }else {
        ui->label_SW1 ->setPixmap(QPixmap(sefidPath));
    }
    if(sw_2){
        ui->label_SW2 ->setPixmap(QPixmap(redPath));
    }else {
        ui->label_SW2 ->setPixmap(QPixmap(sefidPath));
    }
    if(sw_3){
        ui->label_SW3 ->setPixmap(QPixmap(redPath));
    }else {
        ui->label_SW3 ->setPixmap(QPixmap(sefidPath));
    }
    if(sw_4){
        ui->label_SW4 ->setPixmap(QPixmap(bluePath));
    }else {
        ui->label_SW4 ->setPixmap(QPixmap(sefidPath));
    }


    ui->lcdNumberkV->display(outPut1);
```

Project 1 continued on next page...

```
    ui->lcdNumberMW->display(outPut2);
    ui->lcdNumberMVA->display(outPut3);



}
```

Implement a simple SCADA system using Qt: HMI/myqsoket.h

```
#ifndef MYQSOKET_H
#define MYQSOKET_H

#include <QObject>
#include <QtDebug>
#include <QString>
#include <QTcpSocket>


#include <QFile>
#include <QJsonObject>
#include <QJsonArray>
#include <QJsonDocument>
#include <QtGlobal>
#include <QTextStream>



class myQSoket : public QObject
{
    Q_OBJECT
public:
    explicit myQSoket(QObject *parent = nullptr);
    void doConnect();
    ~myQSoket();
private:
    QTcpSocket* myHMISoket;
    QByteArray* myQJosnByetedInPut;
signals:
    void dataIsRecived();
public slots:
};

#endif // MYQSOKET_H
```

Implement a simple SCADA system using Qt: HMI/myqsoket.cpp

```
#include "myqsoket.h"
#include <QThread>
myQSoket::myQSoket(QObject *parent) : QObject(parent)
```

Project 1 continued on next page...

```
{
    myHMISoket = new QTcpSocket(this);
    myQJosnByetedInPut= new QByteArray();

}

void myQSoket::doConnect()
{
    myHMISoket->connectToHost("127.0.0.1",8585);
   if(myHMISoket->waitForConnected(50000)){
       myHMISoket->write("Im HMI");
       myHMISoket->flush();
       myHMISoket->waitForBytesWritten(50000);
       myHMISoket->waitForReadyRead(30000);
       *myQJosnByetedInPut = myHMISoket->readAll();
       qDebug()<<*myQJosnByetedInPut;
       QString newData= *myQJosnByetedInPut;
       QJsonDocument newReadJsonFileDoc = QJsonDocument::fromJson(newData.toUtf8());
       QJsonObject newReadJsonObj = newReadJsonFileDoc.object();
       bool mustUpdat= false;
       if(newReadJsonObj.value("outPut1").toVariant().value<double>()>0){

           mustUpdat= true;
       }

       if(*myQJosnByetedInPut->data() & mustUpdat){
           mustUpdat= false;
           QFile fin("recivedDataFromHMI.json");
           fin.open(QIODevice::WriteOnly);
           fin.write(*myQJosnByetedInPut);
           fin.close();

       }
   }

}

myQSoket::~myQSoket()
{
    delete myHMISoket;
    delete myQJosnByetedInPut;
}
```

Implement a simple SCADA system using Qt: HMI/mythread.h

```
#ifndef MYTHREAD_H
#define MYTHREAD_H

#include <QObject>
#include <QThread>
```

Project 1 continued on next page...

```
#include <myqsoket.h>
class MyThread : public QThread
{
public:
    void run();
    MyThread();
    ~MyThread();
    myQSoket *socket;
public slots:
    void readyToShow();
signals:
    void emitToGUI();

};

#endif // MYTHREAD_H
```

Implement a simple SCADA system using Qt: HMI/mythread.cpp

```
#include "mythread.h"

void MyThread::run()
{
    while (true) {
         socket->doConnect();
    }


}

MyThread::MyThread()
{
socket = new myQSoket();
connect(socket,SIGNAL(dataIsRecived), this , SLOT(readyToShow));
}

MyThread::~MyThread()
{
    delete socket;
}
```