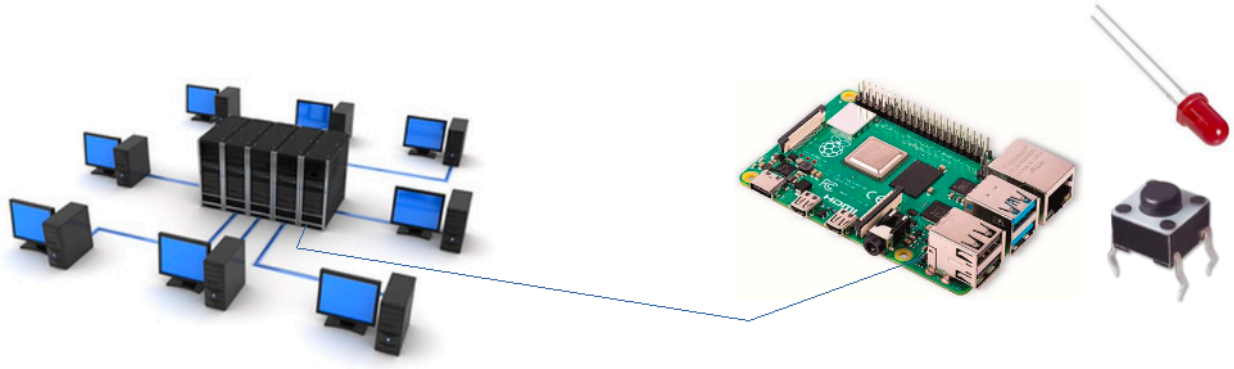
Création d'une application IOT : *Internet of things* (Internet des objets) avec Qt Creator

---

But : Apprendre à réaliser une application Qt mettant en œuvre une liaison réseau UDP entre un PC et une Raspberry Pi. Gérer un protocole de communication simple.

1 Présentation du TP

Le tutoriel propose de réaliser un logiciel de gestion d'un Raspberry connecté à un PC par réseau.



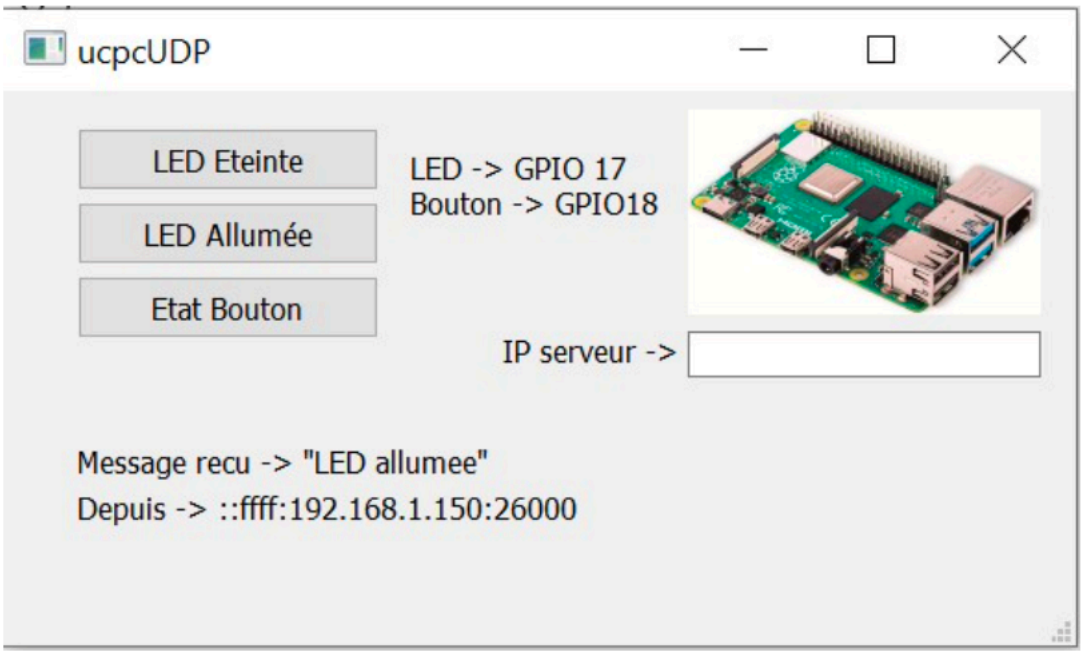
Matériel :

- Un Raspberry Pi
- Une led de signalisation
- Un bouton poussoir

Le logiciel gère ces éléments en fonctions du protocole de communication avec un PC suivant, chaque commande étant suivie d'un tiret d'un message et d'un passage à la ligne :

Périphérique	Action	Message ASCII	Réponse
LED	Allumage	'1'	LED allumée
LED	Extinction	'0'	LED Éteinte
Bouton	Demande état bouton	'2	0 ou 1

Exemples :
Pour allumer la led le PC transmet : ‘1’
Pour connaître l’état du bouton, le PC envoie ‘2’ , le Raspberry répond ‘1’ ou ‘0’



Coté serveur – Raspberry Pi

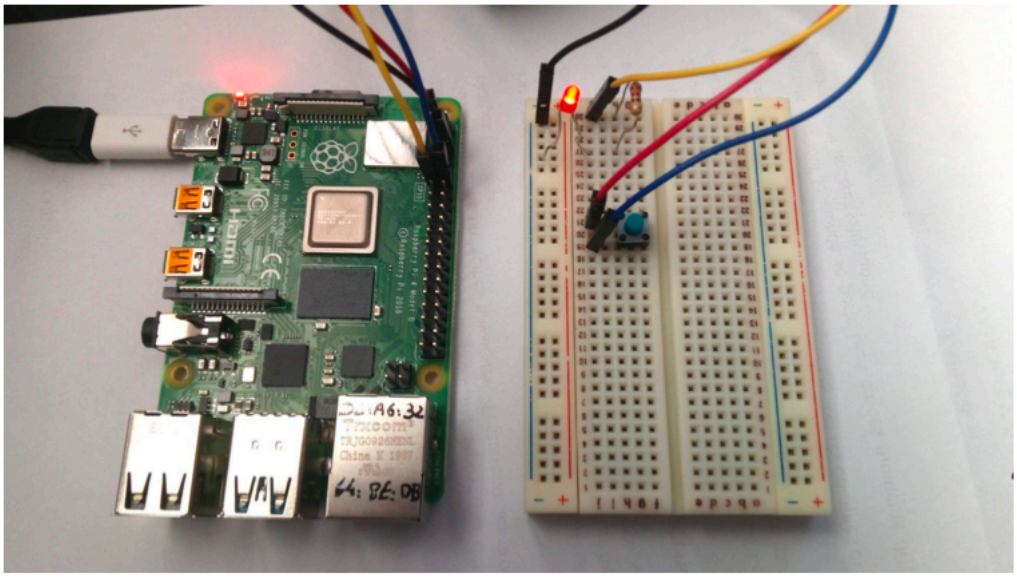
Le serveur écoute les communications UDP sur le port 26000.

```
/*
simple serveur UDP sur RPi  (LINUX DEBIAN)
compilateur g++
g++ -Wall -o srvUDP srvUDP.cpp -lwiringP
*/
// bibliothèques C
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h> // conversion types reseau ASCII
#include <sys/types.h>
#include <sys/socket.h> // gestion des sockets
#include <netinet/in.h> // definition des protocoles internet
#include <string.h>
#include <netdb.h>  // definitions de in_port_t, in addr_t de la structure
hostent
#include <unistd.h> // pour les systèmes POSIX
// bibliothèques C++
#include <iostream>
#include <string>
// wiringPi pour gérer les périphériques de Raspberry Pi
#include <wiringPi.h>
using namespace std;

#define portEcoute 26000
#define led 0   // GPIO 17
#define btn 1   // GPIO 18
```

```
void error(const char *msg); void infoServeur(void);

int main(void)

{
cout<<"Serveur UDP..."<<endl;
cout<<"Reception ASCII 0 -> extinction LED"<<endl; cout<<"Reception ASCII 1 ->
allumage LED"<<endl; cout<<"Reception ASCII 2 -> retourne etat du
bouton"<<endl; cout<<"-------------------------------------------"<<endl;
int sock, n;
socklen_t tailleClient;
struct sockaddr_in server; // structure de données IP du serveur struct
sockaddr_in client; // structure de données IP du client char buf[1024];

//  creation du socket
sock = socket(AF_INET, SOCK_DGRAM, 0);
if (sock < 0) error("ouverture socket");
```

```
else cout<<"socket ouvert"<<endl; server.sin_family = AF_INET;

IP)

    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(portEcoute);
//  activation ecoute
// enregistrement des infos IP serveur (AF_INET) pour
// adresse du serveur
// port

if (bind(sock, (struct sockaddr *)&server, sizeof(server)) < 0) error("binding");

tailleClient = sizeof(struct sockaddr_in); // Affichage infos serveur

    infoServeur();
    // configuration peripheriques
    wiringPiSetup();   //numeros broches
    pinMode(led,OUTPUT); // LED en sortie
    pinMode(btn,INPUT);  // bouton en entrée
while (1)
{
string mess="";

    // attente requete du client (bloquant)
n = recvfrom(sock, buf, 1024, 0, (struct sockaddr *)&client, &tailleClient);
if (n < 0) error("reception depuis client");

    // message recu
cout<<"Reception message de
"<<inet_ntoa(client.sin_addr)<<":"<<client.sin_port<<" -> "; cout<<buf<<endl;
// analyse du message
// le premier caractère de buff defini la tâche


switch (*buf) {
```

```
case '0': mess="LED eteinte"; digitalWrite(led,LOW);

break;
case '1': mess="LED allumee";

digitalWrite(led,HIGH); break;

case '2': if(digitalRead(btn)) mess="1 (bouton relache)"; else mess="0 (bouton
appuye)";

break;
default : mess="commande inconnue";


}

        // emission de la réponse
n = sendto(sock, mess.c_str(), 17,0, (struct sockaddr *)&client,
tailleClient); if (n < 0) error("sendto");

else cout<<"Reponse -> "<<mess<<endl; }

return 0; }

void infoServeur()

{

cout<<IPbuff<<" ecoute sur port "<<portEcoute<<endl; }

void error(const char *msg)

{
    perror(msg);
exit(0);

}
```



Coté client – Application PC

Ajouter la ligne suivante dans le fichier .pro

QT += network **main.cpp**

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

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

}
```

ucpcudp.h

```
#ifndef UCPCUDP_H #define UCPCUDP_H

#include <QMainWindow>
#include <QUdpSocket>
QT_BEGIN_NAMESPACE
namespace Ui { class ucpcUDP; } QT_END_NAMESPACE

class ucpcUDP : public QMainWindow {

Q_OBJECT

public:
ucpcUDP(QWidget *parent = nullptr); ~ucpcUDP();

public slots:
void on_Mess_a_lire();
void on_pushButton_clicked();

private slots:
void on_btn0_clicked();

void on_btn1_clicked();

void on_lineEdit_returnPressed();

private:
    Ui::ucpcUDP *ui;
    QUdpSocket  *MaSocket;
```

```
    QString IPserveur="192.168.1.150";
};
#endif // UCPCUDP_H
```

ucpdudp.cpp

```
#include "ucpcudp.h" #include "ui_ucpcudp.h"

#include <QDebug>
#include <QLabel>
#include <QLineEdit>
#include <QString>
//#define IPserveur "192.168.1.150"
#define portServeur 26000
ucpcUDP::ucpcUDP(QWidget *parent) : QMainWindow(parent)
, ui(new Ui::ucpcUDP)

{
    ui->setupUi(this);
    qDebug() <<"Debut du programme"<<endl;
    MaSocket=new QUdpSocket(this);
    if (!MaSocket->bind(QHostAddress::Any, 26000))
    {
        qDebug("Impossible de créer le socket en écoute");
        exit(EXIT_FAILURE);
    }
    qDebug() <<"bind OK";
    if (!connect(MaSocket,SIGNAL(readyRead()),this,SLOT(on_Mess_a_lire())))
    {
        qDebug("Impossible de créer le slot");
        exit(EXIT_FAILURE);
    }
    qDebug() <<"Slot UDP ok";
}
ucpcUDP::~ucpcUDP() {

MaSocket->close();

delete ui; }

void ucpcUDP::on_Mess_a_lire() {

    QHostAddress senderAddress;
    quint16 senderPort;
    QString msg;
     while (MaSocket->hasPendingDatagrams())
```

```
    {
      QByteArray datagram;
      datagram.resize(MaSocket->pendingDatagramSize());
      if (MaSocket->readDatagram(datagram.data(), datagram.size(),
&senderAddress, &senderPort) == -1)
        {
MaSocket->close();

         exit(EXIT_FAILURE);
        }
      msg = datagram.data();
      qDebug() << "Reception depuis : " << senderAddress.toString() << ':' <<
senderPort;
      qDebug() << "Message recu-> " << msg;
      ui->lblReception->setText(tr("Depuis ->
%1:%2").arg(senderAddress.toString()).arg(senderPort));
      ui->lblEmission->setText(tr("Message recu -> \"%1\"").arg(msg));
    }
}

void ucpcUDP::on_pushButton_clicked()

{
 QByteArray datagram="2";
   if (MaSocket->writeDatagram(datagram, QHostAddress(IPserveur), portServeur)
== -1)
{
qDebug("Émission du message modifié impossible"); MaSocket->close();
exit(EXIT_FAILURE);

}
 qDebug() << "Emission du message : " << datagram.data();
void ucpcUDP::on_btn0_clicked() {

   QByteArray datagram="0";
     if (MaSocket->writeDatagram(datagram, QHostAddress(IPserveur),
portServeur) == -1)
       {
void ucpcUDP::on_btn1_clicked()

{
   QByteArray datagram="1";
     if (MaSocket->writeDatagram(datagram, QHostAddress(IPserveur),
portServeur) == -1)
{
qDebug("Émission du message modifié impossible"); MaSocket->close();
exit(EXIT_FAILURE);
```

```
}
 qDebug() << "Emission du message : " << datagram.data();
}

void ucpcUDP::on_lineEdit_returnPressed() {

    IPserveur=ui->lineEdit->text();
}
```