Transformación de Operaciones

Santiago León Ortiz
Carlos Antonio Bulnes Domínguez

23 de junio de 2016

1. Cliente

```
#include <QtWidgets>

#include "editor_client.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    EditorCliente ed;
    ed.show();

    return a.exec();
}
```

```
#include "editor_client.h"
#include <stdlib.h>

#include <QString>
#include <iostream>

using namespace std;

```

```
9  struct Transform
10 {
11     qint32 pos;
12     quint8 c;
13     qint32 priority;
14 };
15
16 QDataStream &operator<<(QDataStream &out, const Transform
   ↪ &transform)
17 {
18     out << transform.pos << transform.c << transform.priority;
19     return out;
20 }
21
22 QDataStream &operator>>(QDataStream &in, Transform &transform)
23 {
24     in >> transform.pos >> transform.c >> transform.priority;
25     return in;
26 }
27
28 EditorCliente::EditorCliente(QWidget *parent)
29     : QWidget(parent)
30 {
31     writing_to_box = false;
32     connect(&m_textEdit, SIGNAL(textChanged()), this,
   ↪ SLOT(onTextChanged()));
33     connect(&m_textEdit, SIGNAL(cursorPositionChanged()), this,
   ↪ SLOT(onCursorPositionChanged()));
34     //connect(&sock, SIGNAL(readyRead()), this,
   ↪ SLOT(m_read()));
35     //connect(&sock, SIGNAL(bytesWritten()), this,
   ↪ SLOT(m_read()));
36     connect(&sock, SIGNAL(readyRead()), this, SLOT(m_read()));
37
38     m_cursor = m_textEdit.textCursor();
39
40     QStringList args = QCoreApplication::arguments();
41
```

```
42      if (args.count() == 1) {
43          m_label.setText(QString("Cliente 1"));
44          start("127.0.0.1", 2347);
45      } else {
46          m_label.setText(QString("Cliente 2"));
47          start(args.at(1), 2348);
48          //start("127.0.0.1", 2346);
49      }
50
51      m_layout.addWidget(&m_textEdit);
52      m_layout.addWidget(&m_label);
53
54      setLayout(&m_layout);
55  }
56
57  EditorCliente::~EditorCliente(){
58    sock.close();
59  }
60
61  void EditorCliente::onTextChanged(){
62      if (writing_to_box)
63          return;
64
65      Transform new_transform;
66
67      QByteArray block;
68      QDataStream sendStream(&block, QIODevice::ReadWrite);
69
70      t = m_textEdit.toPlainText();
71
72      new_transform.pos =
    ↪ m_textEdit.textCursor().positionInBlock() - 1;
73      new_transform.c = t[new_transform.pos].toLatin1();
74      new_transform.priority = 0;
75      sendStream << new_transform;
76      sock.write(block);
77
78      cout << "Paquete enviado"<< endl;
```

```
79      cout << "Posicion: " ;
80      cout << new_transform.pos << endl;
81      cout << "Caracter: " ;
82      cout << new_transform.c << endl;
83      cout << "Prioridad: " << new_transform.priority << endl;
84  }
85
86  void EditorCliente::onCursorPositionChanged(){
87  }
88
89  void EditorCliente::start(QString address, quint16 port)
90  {
91    QHostAddress addr(address);
92    sock.connectToHost(addr, port);
93  }
94
95  void EditorCliente::m_read(/* arguments */) {
96      Transform transform;
97      QTcpSocket *tcpSocket = (QTcpSocket*)sender();
98
99      if (tcpSocket->bytesAvailable() < 9) {
100         return;
101     }
102
103     QByteArray block = tcpSocket->read(9);
104     QDataStream sendStream(&block, QIODevice::ReadWrite);
105     sendStream >> transform;
106
107     cout << "Respuesta del servidor:"<< endl;
108     cout << "Posicion: " ;
109     cout << transform.pos << endl;
110     cout << "Caracter: " ;
111     cout << transform.c << endl;
112     cout << "Prioridad: " << transform.priority << endl;
113
114     cout << "*****Caracter: " << caracter << endl;
115
116     t = t.insert(transform.pos, transform.c);
```

```
117
118     QTextCursor tmp_cursor = m_textEdit.textCursor();
119     int cur_position;
120     if (transform.pos < m_textEdit.textCursor().position())
121         cur_position = m_textEdit.textCursor().position() + 1;
122     else
123         cur_position = m_textEdit.textCursor().position();
124
125     writing_to_box = true;
126     m_textEdit.setText(t);
127     writing_to_box = false;
128
129     tmp_cursor.setPosition(cur_position);
130     m_textEdit.setTextCursor(tmp_cursor);
131 }
```

2. Servidor

```
1 #include "server.h"
2 #include <QApplication>
3
4 int main(int argc, char** argv)
5 {
6   QApplication app(argc, argv);
7   Server server;
8   return app.exec();
9 }
```

```
1 // server.cc
2 #include "server.h"
3 #include "mythread.h"
4 #include <iostream>
5
6 using namespace std;
7
```

```
8  QDataStream &operator<<(QDataStream &out, const Transform
   ↪  &transform)
9  {
10     out << transform.pos << transform.c << transform.priority;
11     return out;
12 }
13
14 QDataStream &operator>>(QDataStream &in, Transform &transform)
15 {
16     in >> transform.pos >> transform.c >> transform.priority;
17     return in;
18 }
19
20 Server::Server(QObject* parent):
   ↪  QTcpServer(parent)//QObject(parent)
21 {
22   num_clients = 0;
23   num_transformaciones = 0;
24
25   connect(&server1, SIGNAL(newConnection()),
26     this, SLOT(acceptConnection1()));
27   server1.listen(QHostAddress::Any, 2347);
28
29   connect(&server2, SIGNAL(newConnection()),
30     this, SLOT(acceptConnection2()));
31   server2.listen(QHostAddress::Any, 2348);
32 }
33
34 Server::~Server()
35 {
36   server1.close();
37   server2.close();
38 }
39
40 void Server::acceptConnection1()
41 {
42   num_clients++;
43   clients[1] = server1.nextPendingConnection();
```

```
44    quint16 port = clients[1]->localPort();
45    QString port_s = QString::number(port);
46    cout << "Puerto: ";
47    cout << port << endl;
48
49    connect(clients[1], SIGNAL(readyRead()),
50      this, SLOT(read_from_client_1()));
51  }
52
53  void Server::acceptConnection2()
54  {
55    num_clients++;
56    clients[2] = server2.nextPendingConnection();
57    quint16 port = clients[2]->localPort();
58    QString port_s = QString::number(port);
59    cout << "Puerto: ";
60    cout << port << endl;
61
62    connect(clients[2], SIGNAL(readyRead()),
63      this, SLOT(read_from_client_2()));
64  }
65
66  void Server::write_to_client (Transform transform, int
    ↪  cli_number) {
67      QByteArray block;
68      QDataStream sendStream(&block, QIODevice::ReadWrite);
69      sendStream << transform;
70      clients[cli_number]->write(block);
71  }
72
73  Transform Server::operat_transformation (Transform t1,
    ↪  Transform t2){
74      //mutex.lock();
75
76      Transform res;
77      res.c = t1.c;
78      res.priority = t1.priority;
79      if (t1.pos < t2.pos ||
```

```
80              (t1.pos==t2.pos && t1.c!=t2.c &&
    ↪ t1.priority<t2.priority)) {
81          res.pos = t1.pos;
82      } else if (t1.pos > t2.pos ||
83              (t1.pos==t2.pos && t1.c!=t2.c &&
    ↪ t1.priority>t2.priority)) {
84          res.pos = t1.pos+1;
85      } else {
86          res.priority = -1;
87      }
88      return res;
89  }
90
91  void Server::read_from_client_1()
92  {
93      if (num_clients !=2)
94          return;
95
96      QTcpSocket *tcpSocket = (QTcpSocket*)sender();
97      Transform transform;
98
99      cout << "(1) bytes: " << tcpSocket->bytesAvailable() <<
    ↪ endl;
100
101     if (tcpSocket->bytesAvailable() < 9)
102         return;
103
104     QByteArray block = tcpSocket->read(9);
105     QDataStream sendStream(&block, QIODevice::ReadWrite);
106     sendStream >> transform;
107
108     transform_client1 = transform;
109
110     mutex.lock();
111     num_transformaciones++;
112     mutex.unlock();
113
114     transform.priority = 2;
```

```
115
116     int cont = 0;
117     while(cont < /*9999999*/INT_MAX){
118         cont++;
119     }
120
121     if(num_transformaciones == 2){
122         transform = operat_transformation(transform_client1,
    ↪  transform_client2);
123     }
124
125     cout << "Posicion: " ;
126     cout << transform.pos << endl;
127     cout << "Caracter: " ;
128     cout << transform.c << endl;
129     cout << "Prioridad: " << transform.priority << endl;
130
131     write_to_client (transform, 2);
132
133     num_transformaciones--;
134 }
135
136 void Server::read_from_client_2()
137 {
138     if (num_clients !=2)
139         return;
140
141     QTcpSocket *tcpSocket = (QTcpSocket*)sender();
142     Transform transform;
143
144     cout << "(2) bytes: " << tcpSocket->bytesAvailable() <<
    ↪  endl;
145
146     if (tcpSocket->bytesAvailable() < 9)
147         return;
148
149     QByteArray block = tcpSocket->read(9);
150     QDataStream sendStream(&block, QIODevice::ReadWrite);
```

```
151     sendStream >> transform;
152 
153     transform_client2 = transform;
154 
155     mutex.lock();
156     num_transformaciones++;
157     mutex.unlock();
158 
159     transform.priority = 2;
160 
161     int cont = 0;
162     while(cont < /*9999999*/INT_MAX){
163         cont++;
164     }
165 
166     if(num_transformaciones == 2){
167         transform = operat_transformation(transform_client2,
    ↪  transform_client1);
168     }
169 
170     cout << "Posicion: " ;
171     cout << transform.pos << endl;
172     cout << "Caracter: " ;
173     cout << transform.c << endl;
174     cout << "Prioridad: " << transform.priority << endl;
175 
176     write_to_client (transform, 1);
177 
178     num_transformaciones--;
179 }
180 
181 
182 void Server::incomingConnections(int socketDescriptor)
    ↪  //Incoming connections
183 {
184   MyThread *thread = new MyThread(socketDescriptor,this);
185 
```

```
186   connect(thread, SIGNAL(finished()), thread,
      ↪  SLOT(deleteLater()));
187
188   //Start a new thread for the connection
189   thread->start();    //Which will cause the run() function
190 }
```

```
1  #include "mythread.h"
2
3  MyThread::MyThread(int ID, QObject *parent):
4    QThread(parent)
5  {
6    this->socketDescriptor = ID;            //Get the socket ID
     ↪  number
7  }
8
9
10 void MyThread::run()
11 {
12   //thread starts here
13
14   socket = new QTcpSocket();
15
16   if ( !socket->setSocketDescriptor(this->socketDescriptor) )
     ↪  //Here we set the socket ID
17   {
18     emit error (socket->error());         //emit the error
     ↪  signal
19     return;
20   }
21
22   connect(socket, SIGNAL(readyRead()), this,
     ↪  SLOT(readyRead()),Qt::DirectConnection );  //Make a direct
     ↪  connection to the thread
23   connect(socket, SIGNAL(disconnected()), this,
     ↪  SLOT(disconnected()));
24
```

```
25    //client is connected
26
27    //IMPORTANT
28    //This function will cause the thread to stay alive until
      ↪  we tell it to close
29    //Otherwise the run() function will end and the thread will
      ↪  be dropped / destroyed
30    exec();
31  }
32
33
34  void MyThread::readyRead()
35  {
36    QByteArray Data = socket->readAll();    //Get all the
      ↪  information from the connected client
37
38    //Send the info back, (echo server)
39    socket->write(Data);
40  }
41
42
43  void MyThread::disconnected()
44  {
45    socket->deleteLater();
46    exit(0);
47  }
```