STL
Boost
Qt

Boost
STL
Qt

A példaprogram logikáját így követi

Boost
STL
Qt
Boost

ha lesz idő

chat program n+1

Példaprogram

Architektúra

Server:

- Csak relay, nem ismeri a protokolt
- Mindenkinek elküld mindent.

Kliensek:

- Számon tartják a többi klienst.
- PING üzenet: "Még itt vagyok"
- MSG üzenet: "Chat üzenet"

Protokol

Szöveges

Minden üzenet egy sor

PING <ID> <NÉV>

MSG <ID> <ÜZENET>

Boost

Ami az STL-ből kimaradt

Előnyök

- Platform független
- Jól meg van írva
- Nagyon robusztus
- Hálózat!
- Szálak!
- Folyamatosan átszivárognak belőle dolgok a szabvány c++-ba
- Nagyon jó dokumentáció

Hátrányok

- Template alapú (főleg a konténerek)
  - A logika a headerben van -> Lassan fordul
  - Nehezen értelmezhető hibaüzenetek(gcc-vel)
  - Az IDE-k autokiegészítője nehezen bír vele
- Nagyon robusztus - az egyszerű dolgokat is körülményes lehet megoldani
- Hatalmas API - meg lehetne egyszerűen oldani, ha megtaláltam volna rá a függvényt

Lemaradt egy '*'

g++ -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"Peer.d" -MT"Peer.d" -o "Peer.o" "../Peer.cpp"

In file included from /usr/include/boost/asio/read_until.hpp:902:0,

from /usr/include/boost/asio.hpp:82,

from ../Peer.h:11,

from ../Peer.cpp:3:

/usr/include/boost/asio/impl/read_until.hpp: In instantiation of ‘void boost::asio::detail::read_until_delim_op<AsyncReadStream, Allocator, ReadHandler>::operator()(const boost::system::error_code&, std::size_t, int) [with AsyncReadStream = boost::asio::basic_stream_socket<boost::asio::ip::tcp>*; Allocator = std::allocator<char>; ReadHandler = boost::_bi::bind_t<void, boost::_mfi::mf2<void, Peer, const boost::system::error_code&, long unsigned int>, boost::_bi::list3<boost::_bi::value<Peer*>, boost::arg<1> (*)(), boost::arg<2> (*)()> >; std::size_t = long unsigned int]’:

/usr/include/boost/asio/impl/read_until.hpp:495:3:   required from ‘void boost::asio::async_read_until(AsyncReadStream&, boost::asio::basic_streambuf<Allocator>&, char, const ReadHandler&) [with AsyncReadStream = boost::asio::basic_stream_socket<boost::asio::ip::tcp>*; Allocator = std::allocator<char>; ReadHandler = boost::_bi::bind_t<void, boost::_mfi::mf2<void, Peer, const boost::system::error_code&, long unsigned int>, boost::_bi::list3<boost::_bi::value<Peer*>, boost::arg<1> (*)(), boost::arg<2> (*)()> >]’

../Peer.cpp:25:2:   required from here

/usr/include/boost/asio/impl/read_until.hpp:409:11: error: request for member ‘async_read_some’ in ‘((boost::asio::detail::read_until_delim_op<boost::asio::basic_stream_socket<boost::asio::ip::tcp>*, std::allocator<char>, boost::_bi::bind_t<void, boost::_mfi::mf2<void, Peer, const boost::system::error_code&, long unsigned int>, boost::_bi::list3<boost::_bi::value<Peer*>, boost::arg<1> (*)(), boost::arg<2> (*)()> > >*)this)->boost::asio::detail::read_until_delim_op<boost::asio::basic_stream_socket<boost::asio::ip::tcp>*, std::allocator<char>, boost::_bi::bind_t<void, boost::_mfi::mf2<void, Peer, const boost::system::error_code&, long unsigned int>, boost::_bi::list3<boost::_bi::value<Peer*>, boost::arg<1> (*)(), boost::arg<2> (*)()> > >::stream_’, which is of pointer type ‘boost::asio::basic_stream_socket<boost::asio::ip::tcp>*’ (maybe you meant to use ‘->’ ?)

make: *** [Peer.o] Error 1

**** Build Finished ****

Amiről szó lesz

Hálózat:
boost::asio

Szálak:
boost::thread

Mellékesen **boost::bind** és **boost::tokenizer**

boost::bind

Függvénypointer-szerűséget hozhatunk létre.

Nagyon hasznos mindenütt, ahol függvényt kell átadni:

- Szálak
- Callbackek

Már benne van a C++11-ben

boost::bind

Nagyon egyszerű használat:
boost::bind(<fv. pointer>, param1, param2,...)

Megy objektumok metódusaival is.
Kerülhetnek bele placeholderek is.

PI:

```
void ConsoleApplication::start() {
    helloThread = new boost::thread(
        boost::bind(
            &MainClass::sayHello,
            this,
            "hello"
        )
    );
```

PI:

```
void ConsoleApplication::start() {
    helloThread = new boost::thread(
        boost::bind(
            &MainClass::sayHello,
            this,
            "hello"
        )
    );
```

boost::bind

PI:

```
void ConsoleApplication::start() {
    helloThread = new boost::thread(
        boost::bind(
            &MainClass::sayHello,
            this,
            "hello"
        )
    );
```

Függvény pointer

PI:

```
void ConsoleApplication::start() {
    helloThread = new boost::thread(
        boost::bind(
            &MainClass::sayHello,
            this,
            "hello"
        )
    );
```

Ezen fog meghívódni a függvény. Nem feltétlen this, mutathat bármilyen MainClass objektumra.

PI:

```
void ConsoleApplication::start() {
    helloThread = new boost::thread(
        boost::bind(
            &MainClass::sayHello,
            this,
            "hello"
        )
    );
```

Egyéb paraméter. Akárhány lehet még.

boost::thread

- Új libek linkelésével jár
- Bindolt eljárással vagy functorral
- join - Mint javában. Blokkol, amíg kilép a szál.
- interrupt - Nem elegáns
- sleep - Speciális, megadott ideig alszik
- ...

mutex, lock, társaik

Hogy ne akadjanak össze a szálak.
Következő előadáson részletesen

RelayServer projectben benne van

boost::asio

- TCP
- UDP
- Aszinkron hívások
- Timerek
- Streamek támogatása

Az IO-ról

1. Minden művelet blokkol. A párhuzamos műveletek külön szálon.
2. select és társai. Blokkoló hívás de összefog több műveletet. Lassú.
3. Aszinkron hívások

Összehasonlítás:

Apache: minden kapcsolat külön szál

nginx: Aszinkron model

Az IO-ról

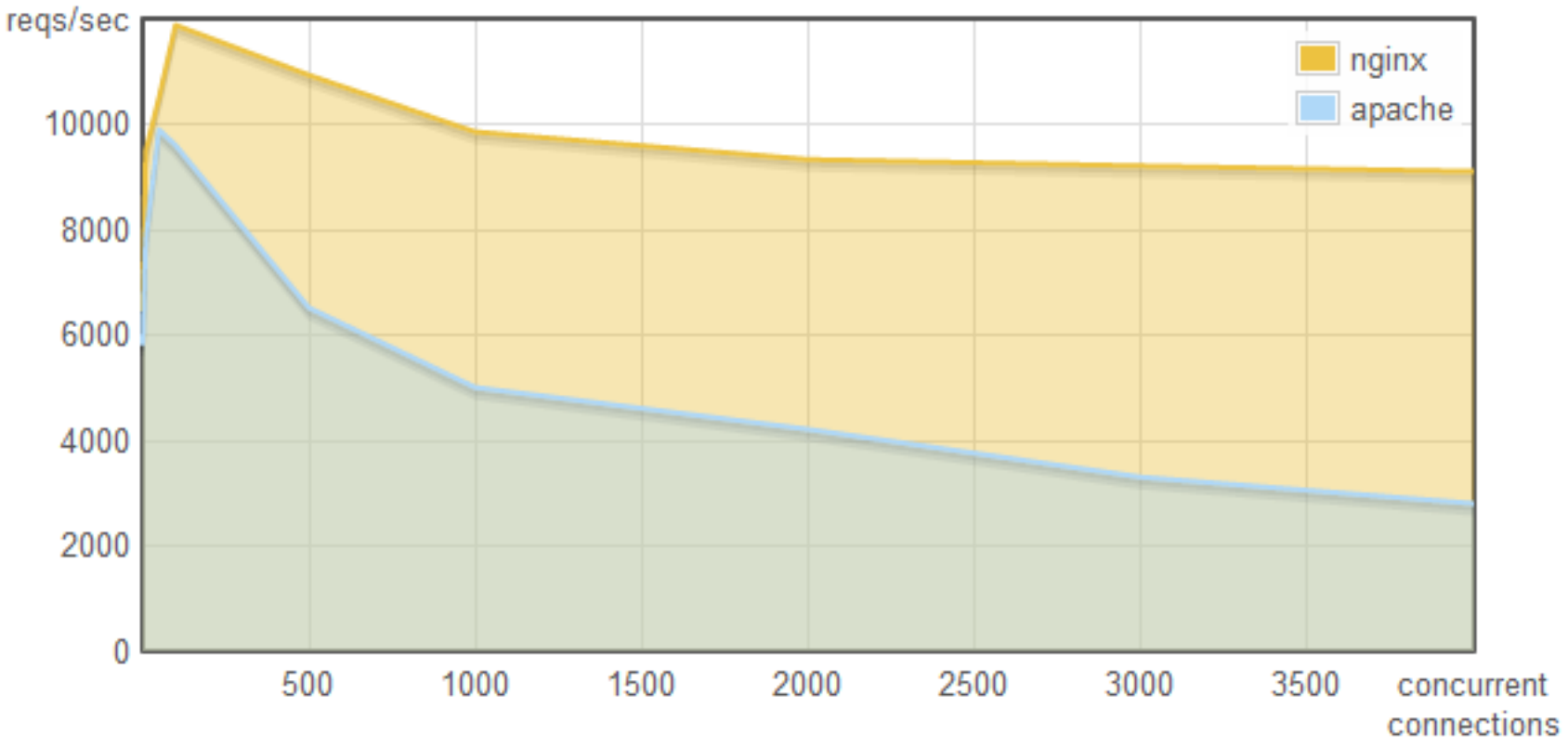
1. Minden művelet blokkol. A párhuzamos műveletek külön szálon.
2. select és társai. Blokkoló hívás de összefog több műveletet. Lassú.
3. Aszinkron hívások

Összehasonlítás:

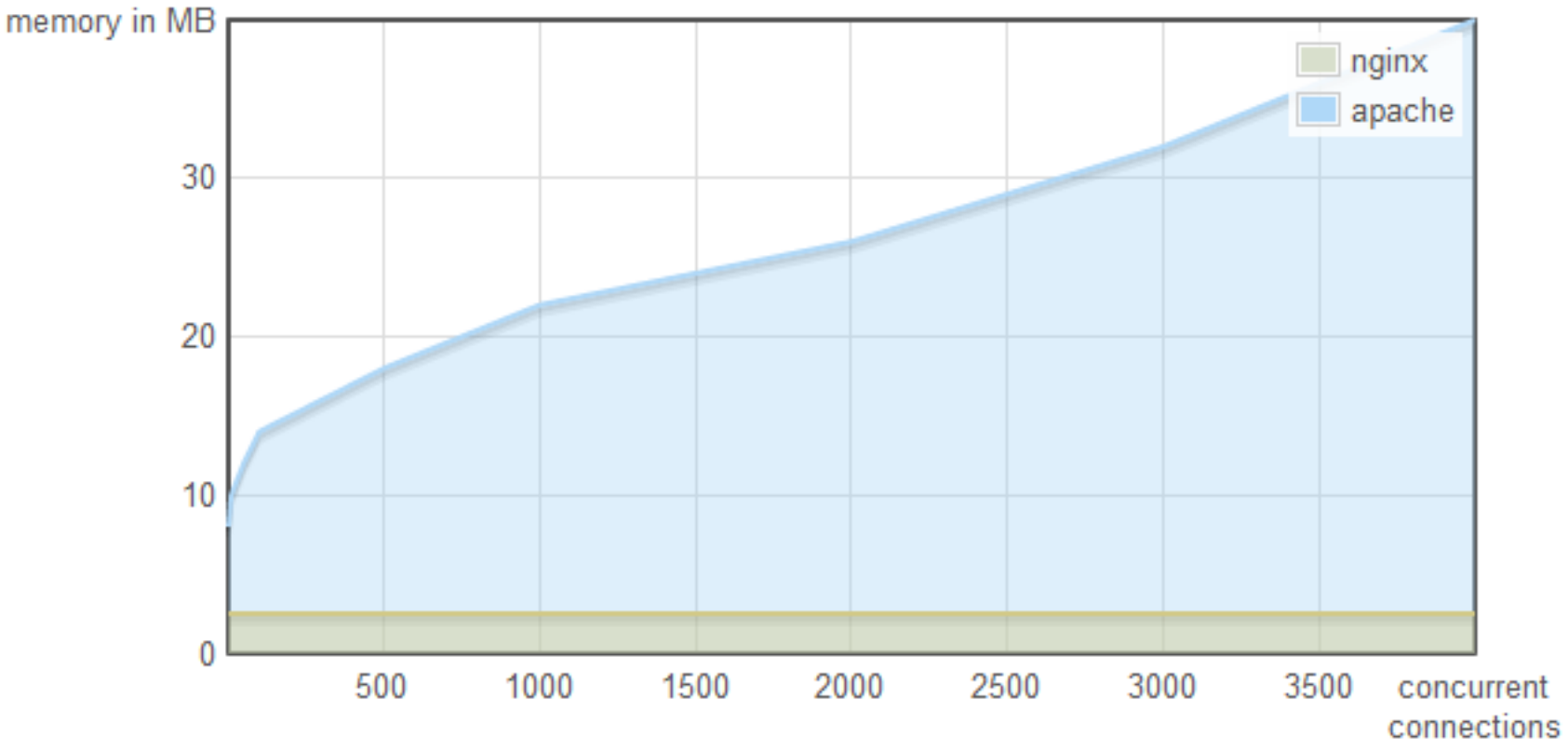
Apache: minden kapcsolat külön szál

nginx: Aszinkron model

Apache vs nginx



Apache vs nginx



boost::asio
fontos fogalmak

ioservice:
Ez az objektum beszél az oprendszerrel.
Konstruktoroknak gyakran kell.

bufferek:
const, mutable, streambuffer
A legtöbb hívás ilyeneket vár.
Könnyen átalakíthatók STL-es konténerekké és vissza.

Tipikus menet

IO service, az oprendszerhez:
boost::asio::io_service io_service()
Socket létrehozása, nem csatlakozik még
boost::asio::ip::tcp::socket socket(io_service)
Csatlakozás a megadott címhez
socket.connect(boost::asio::ip::tcp::end_point)
Küldés:
socket.send(boost::asio::buffer)
Fogadás:
socket.receive(boost::asio::buffer)

Címek

boost::asio::ip::address

+

port

=

boost::asio::ip::tcp::endpoint

```
string ip("127.0.0.1");
boost::asio::ip::address::from_string(ip.c_str());
```

buffer

String becsomagolás:

```
string message("kacsa");
boost::asio::buffer(message)
```

példa: `BoostNet::sendPing`

Konténer becsomagolás:

```
vector<char> buf;
boost::asio::buffer(buf)
```

Hasznos

`boost::asio::read_until()`

Meg lehet adni egy karaktert/stringet/regexpet/..., hogy meddig olvasson

Könnyen lehet vele readline-t csinálni

Példa: `BoostNet::receiveThreadFunc`

Server socket

`boost::asio::ip::tcp::acceptor`

io_service és egy endpoint kell neki létrehozásnál

accept()-el fogad kapcsolatot

Kell neki egy tcp::socket& amit kitölt.

Névfeloldás

DNS névfeloldó, io_service kell neki:

`boost::asio::ip::tcp::resolver`

Egy query kell neki:

`boost::asio::ip::tcp::resolver::query`

Névfeloldás

Query használat

query("github.com", "http")->204.232.175.90:80

Névfeloldás

Query használat

query("github.com", "12312") -> 204.232.175.90:12312

Névfeloldás

Query használat

query("github.com", "smtp")->204.232.175.90:25

Névfeloldás

Query használat

query("github.com", "")->204.232.175.90:0

Névfeloldás

```
tcp::resolver::iterator it =
resolver.resolve(query);

tcp::resolver::iterator endit;

for (; it != endit; ++it) {
   // Az iterator tcp::endpoint-okat ad
}
```

Gyakorlat

Megírni a kapcsolódást a szerverhez.
serverIP, serverPort adott member változóban

ip->endpoint->socket.connect

Gyakorlat

Hálózati fogadó szál.
receiveThreadFunc legyen a függvény, amit hív

Ez az I/O-nál az első verzió. Minden blokkoló hívásnak egy szál.

Könnyen elrontható

- '&' a függvény neve elé
- nem kell () a függvény után
- a boost általában mindent referenciaként vár

Gyakorlat

Pingelő szál

Induljon el, és időközönkét küldjön pinget a másik usernek. Kilépésnél várja be ezt a szálat az alkalmazás.

Könnyen elrontható

- '&' a függvény neve elé
- nem kell () a függvény után
- a boost általában mindent referenciaként vár

boost::tokenizer

Hasonlít a java string tokenizerhez.

```
boost::char_separator<char> separator(" ");
```

Ez lesz az elválasztó.

boost::tokenizer

Hasonlít a java string tokenizerhez.

```
char_separator<char> separator(" ");
tokenizer<char_separator<char> > tokenizer
(line, separator);
```

Létrehozzuk magát a tokenizert

boost::tokenizer

Hasonlít a java string tokenizerhez.

```
char_separator<char> separator(" ");
tokenizer<char_separator<char> > tokenizer
(line, separator);

.... it = tokenizer.begin();
```

Iterátorral lehet bejárni

STL

STL - Annyira nem nagy

- C hedörök std namespace alatt
  - pl. cstdlib, cstdio, cstring, cmath, ...
- Konténerek
  - Alapból 8 fajta
- I/O
  - streamek, 9 header
- Egyebek
  - algorithm
  - string
  - ...

Annyira nem nagy

- C hedörök std namespace alatt
  - pl. cstdlib, cstdio, cstring, cmath, ...
- Konténerek
  - Alapból 8 fajta
- I/O
  - streamek, 9 header
- Egyebek
  - algorithm
  - string
  - ...

<algorithm>

http://www.cplusplus.com/reference/algorithm/

iterator alapú
kollekciókon művelet

Gyakorlat

Kliensek rendezése név szerint, miután megérkezett egy új.

Nem pingelő kliensek eltávolítása.

Ezeknek kell egy új maintenance szál. (A headerben ott van). Kilépéskor várja be.

Qt

A (nemcsak) C++ (és nemcsak) GUI SDK

Qt bevezető

http://qt-project.org/downloads

- Opensource (Trolltech -> Nokia -> Digia)
- Windows, Linux, OS X, N9, BlackBerry, Android, Linux Embedded
- QtCore, QtGui, QtWidgets, QtNetwork, QtTest, QtOpenGL, QtWebKit, QtSql, QtXml, ...
- C++, QML, CSS & JavaScript, ...
- Qt Creator, Designer

Chat++ GuiApplication

- GuiApplication
  - az alkalmazás vezérlője és GUI-ja
- INetworkAdapter
  - összekapcsolja a különböző hálózati protokollokat az alkalmazást
- LoopbackNetworkAdapter
  - egyszerű visszacsatolás
- PingnetNetworkAdapter
  - kezeli a PING/MSG hálózati protokollt kezelő libeket
- UserListModel
  - a bejelentkezett felhasználók listáját mutató GUI adatai

.pro file

```
QT       += core gui network
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = GuiApplication
TEMPLATE = app

SOURCES += main.cpp GuiApplication.cpp ...
HEADERS  += GuiApplication.h ...
FORMS    += GuiApplication.ui

LIBS += -L$$OUT_PWD/../QtNet/ -lQtNet
INCLUDEPATH += $$PWD/../QtNet
DEPENDPATH += $$PWD/../QtNet
```

.pro file - modulok

```
QT       += core gui network
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = GuiApplication
TEMPLATE = app

SOURCES += main.cpp GuiApplication.cpp ...
HEADERS  += GuiApplication.h ...
FORMS    += GuiApplication.ui

LIBS += -L$$OUT_PWD/../QtNet/ -lQtNet
INCLUDEPATH += $$PWD/../QtNet
DEPENDPATH += $$PWD/../QtNet
```

A használt Qt modulok felsorolása (include, link)

.pro file - végeredmény

```
QT       += core gui network
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = GuiApplication
TEMPLATE = app

SOURCES += main.cpp GuiApplication.cpp ...
HEADERS  += GuiApplication.h ...
FORMS    += GuiApplication.ui

LIBS += -L$$OUT_PWD/../QtNet/ -lQtNet
INCLUDEPATH += $$PWD/../QtNet
DEPENDPATH += $$PWD/../QtNet
```

A termék típusa és neve

.pro file - forrás

```
QT       += core gui network
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = GuiApplication
TEMPLATE = app

SOURCES += main.cpp GuiApplication.cpp ...
HEADERS  += GuiApplication.h ...
FORMS    += GuiApplication.ui

LIBS += -L$$OUT_PWD/../QtNet/ -lQtNet
INCLUDEPATH += $$PWD/../QtNet
DEPENDPATH += $$PWD/../QtNet
```

A forrás fájlok és GUI leíró

.pro file - külső libek

```
QT       += core gui network
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = GuiApplication
TEMPLATE = app

SOURCES += main.cpp GuiApplication.cpp ...
HEADERS  += GuiApplication.h ...
FORMS    += GuiApplication.ui

LIBS += -L$$OUT_PWD/../QtNet/ -lQtNet
INCLUDEPATH += $$PWD/../QtNet
DEPENDPATH += $$PWD/../QtNet
```

Egyéb külső függőségek

main.cpp

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

    QtNet net("127.0.0.1", 50002);
    PingnetNetworkAdapater adapter(net);

    User me(777, "User");
    GuiApplication w(me, adapter);
    w.show();

    return a.exec();
}
```

main.cpp - EventLoop

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

    QtNet net("127.0.0.1", 50002);
    PingnetNetworkAdapater adapter(net);

    User me(777, "User");
    GuiApplication w(me, adapter);
    w.show();

    return a.exec();
}
```

Eseménykezelő fő szál és más erőforrások

main.cpp

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

    QtNet net("127.0.0.1", 50002);
    PingnetNetworkAdapater adapter(net);

    User me(777, "User");
    GuiApplication w(me, adapter);
    w.show();

    return a.exec();
}
```

A hálózati protokoll megvalósítás Qts változata

main.cpp

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

    QtNet net("127.0.0.1", 50002);
    PingnetNetworkAdapater adapter(net);

    User me(777, "User");
    GuiApplication w(me, adapter);
    w.show();

    return a.exec();
}
```

A hálózati protokollt illeszti az alkalmazáshoz

main.cpp

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

    QtNet net("127.0.0.1", 50002);
    PingnetNetworkAdapater adapter(net);

    User me(777, "User");
    GuiApplication w(me, adapter);
    w.show();

    return a.exec();
}
```

Beégetett felhasználói név és azonosító :)

main.cpp

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

    QtNet net("127.0.0.1", 50002);
    PingnetNetworkAdapater adapter(net);

    User me(777, "User");
    GuiApplication w(me, adapter);
    w.show();

    return a.exec();
}
```

A GUI és a vezérlő példányosítása

GuiApplication

- Az alkalmazás főablaka, a QWidget-ből származik. (QObject)
  - A QObjectek headerjéből a MOC (meta object compiler) készít egy moc_*.cpp állományt. Ez is a projecthez fordul.
- #include "ui_GuiApplication.h"
  - A .ui leíróból készíti fordításkor az UIC program
  - Nem érdemes szerkeszteni, inkább a .ui-t Qt Desingerrel
- Összeköti a GUI és a hálózati eseményeket (signal és slot)

Signal/Slot

```
class GuiApplication : public QWidget
{
    Q_OBJECT
.
.
signals:
    void login(const User& user);
    void logout(const User& user);
    void sendMessage(const Message &message);

public slots:
    void onUserLogin(const User& user);
    void onUserLogout(const User& user);
    void onMessageReceived(const Message& message);

private slots:
    void on_connectButton_clicked(bool checked);
.
.
};
```

Signal/Slot

- Közvetlenül is hívhatók
- A callback-ek hasonló
- Az esemény bekövetkezésekor (emit signal) meghívódik a kezelő (slot).
- Egy "emit login(m_user);" utasítás kiadásakor mintha "m_networkAdapter.login(m_user);" utasítást írtuk volna le.

```
connect(    this,                   SIGNAL(login(const User&)),
            &m_networkAdapter,      SLOT(login(const User&)));
```

Signal/Slot

- Közvetlenül is hívhatók
- A callback-ek hasonló
- Az esemény bekövetkezésekor (emit signal) meghívódik a kezelő (slot).
- Egy "emit login(m_user);" utasítás kiadásakor mintha "m_networkAdapter.login(m_user);" utasítást írtuk volna le.

```
connect(    this,                   SIGNAL(login(const User&)),
            &m_networkAdapter,      SLOT(login(const User&)));
```

A signalt küldő objektum.

Signal/Slot

- Közvetlenül is hívhatók
- A callback-ek hasonló
- Az esemény bekövetkezésekor (emit signal) meghívódik a kezelő (slot).
- Egy "emit login(m_user);" utasítás kiadásakor mintha "m_networkAdapter.login(m_user);" utasítást írtuk volna le.

```
connect(    this,                   SIGNAL(login(const User&)),
            &m_networkAdapter,      SLOT(login(const User&)));
```

A signal függvény.

Signal/Slot

- Közvetlenül is hívhatók
- A callback-ek hasonló
- Az esemény bekövetkezésekor (emit signal) meghívódik a kezelő (slot).
- Egy "emit login(m_user);" utasítás kiadásakor mintha "m_networkAdapter.login(m_user);" utasítást írtuk volna le.

```
connect(    this,                   SIGNAL(login(const User&)),
            &m_networkAdapter,      SLOT(login(const User&)));
```

A fogadó objektum

Signal/Slot

- Közvetlenül is hívhatók
- A callback-ek hasonló
- Az esemény bekövetkezésekor (emit signal) meghívódik a kezelő (slot).
- Egy "emit login(m_user);" utasítás kiadásakor mintha "m_networkAdapter.login(m_user);" utasítást írtuk volna le.

```
connect(    this,                   SIGNAL(login(const User&)),
            &m_networkAdapter,      SLOT(login(const User&)));
```

A slot függvény

Signal/Slot

- Paramétereknek meg kell egyezni (slotnak lehet kevesebb)
- A kapcsolat lehet NxM.
- A paraméterek típusához tartozni kell Qt-s MetaType-nak.
  - beépített és Qt-s típusoknak van
  - ha az osztályunk van publikus copy és paraméter nélküli konstruktora akkor mi is írhatunk.
  - Ez nem Qt-s szálról hívott emit esetén érdekes.

UserListModel

- Modell alapú és elem alapú megjelenítők (ListView és ListWidget, ...)
- A modellben vannak az adatok a gui elem csak a megjelenítést vezérli.
- Az UserListModel egy saját, felhasználók adatainak listáját tartalmazó adattároló
- QAbstractListModel-ből származik.
- Csak két kötelezően megvalósítandó függvény: rowCount, data

PingnetNetworkAdapater

- Az alkalmazás szempontjából érdekes esemény:
  - felhasználó ki- és belépés
  - üzenet küldés és fogadás
- A hálózati protokoll rejtve maradhat, lecserélődhet
- PingnetNetworkAdapater ismeri, hogy a NetInterface mögött rejtőző hálózati libet hogyan kell használni

QtNet - egy másik hálózati lib

- Megvalósítja a NetInterface osztályt, így ugyan azt várhatjuk el tőle mint a boostNettől
- Nincs új szál, a QTcpSocket már használ szálakat!

```
QObject::connect(
    &m_socket,   SIGNAL(readyRead()),
    this,                SLOT(analyze()));
```

- De nem signal/slot alapú a callback. Nem ezt mondja az interface.

```
m_callback->onPing(id, payload.toStdString())
```

boost::asio - aszinkron IO

- RelayServer project
- Minden hívásnak van egy async_* megfelelője.
- Meg kell adni bindolt eljárásokat, amik a callbackek lesznek.
- Általában error code is jön, ezeknek kell placeholder.
- kell egy io_service.run() hogy működjön

boost::asio - aszinkron IO

waitForNewConnection():

```
Peer* newPeer = new Peer((Callback*)this,
peerSocket); // Socket és endpoint tároló

serverSocket->async_accept(
   *newPeer->getSocket(),
   *newPeer->getEndpoint(),
   bind(&RelayServer::peerAccepted,
        this,
        newPeer,
        asio::placeholders::error)
   );
```

boost::asio - aszinkron IO

```
asio::async_read_until(
   *socket,
   streamBuf, //boost::asio::streambuf
   '\n',
   bind(
      &Peer::onRead,
      this,
      asio::placeholders::error,
      asio::placeholders::bytes_transferred
   )
);
```

Források

http://blog.webfaction.com/2008/12/a-little-holiday-present-10000-reqssec-with-nginx-2/

www.boost.org

www.cplusplus.com

http://qt-project.org/doc/

http://www.wikipedia.org/