Training Advices

- Interact / Interrupt
- If I’m not clear stop me and ask to repeat
- No question is stupid!
- Two way conversation is better
- Qt documentation is really good
  - I’m not going to read it! DIY :)
- **Strong emphasis on code**
- Excuse my French!
- Contact info: s.borghese@netresults.it

Network Programming

With 

Sergio Borghese
NetResults Srl

The Qt Company

NetResults
Building the digital society

Agenda

- NetResults Srl :: a short intro
- Create TCP and UDP Sockets
- Sending and Receiving data
- Encrypted connections
  - QSslSocket
- A simple QSslServer
- The NetworkManager
- QHostInfo class

NetResults Srl :: a short intro

- Established in 2006
- Spin-off University of Pisa
  - Telecommunication Network Research group
- Strong skill on VoIP/MoIP, netwoking, network performance testing
- Employees: 20 and growing
  - With degree 16/20
  - With PhD 2/16

Quick Miscellanea

- Create Project in QtCreator
  - subprojects
- Kits
  - Different versions & platforms
- Code Snippet

TCP & UDP

- TCP & UDP are L4 protocols used on top of IP
- TCP
  - Connection oriented
  - Stream oriented
  - Reliable (higher latency)
- UDP
  - Message (datagram) oriented
  - Connectionless
  - Non reliable (lower latency)

Qt & sockets

- Enable network module in .pro
  - **QT += network**
- `QAbstractSocket`
  - `QTcpSocket`
  - `QUdpSocket`
  - `QSslSocket`
- API support for **sync** or **async** operations
  - *Do NOT Mix and Match*
- Sockets have an event-loop
  - **Do not move sockets between threads**

QUdpSocket Async API

- Create the udp socket
- Bind the socket
  - Useful for **multi-homed** systems
- Write data to socket
  - **`writeDatagram()`**
  - **`byteWritten()`** signal
- Read data on **`readyRead()`** signal
  - **`hasPendingDatagrams()`**
  - **`readDatagram()`**
    - MUST call this at least one time to get signaled again

UDP :: Send & Receive Data

TX

- `new QUdpSocket()`
- `bind()`
- `writeDatagram()`

RX

- `new QUdpSocket()`
- `bind()`
- → `readyRead()`
  - `readDatagram()`

QUdpSocket Sync API

- Create the udp socket
- Bind the socket
  - Useful for **multi-homed** systems
- Write data to socket
  - **`waitForBytesWritten(int msec)`**
- Read data
  - **`waitForReadyRead(int msec)`**
- **`-1`** → `no timeout`
- `! error ~= timeout`

Low Level Socket

- **socketDescriptor() : qintptr**
- **setSocketDescriptor(qintptr, SocketState)**

```
int val = IP_PMTUDISC_DO;
setsockopt(m_socket,
           SOL_IP,
           IP_MTU_DISCOVER,
           &val,
           sizeof(val));
```

TCP :: Send & Receive Data

Client

- new QTcpSocket
- bind() [opt.]
- Connect sig/slot
  - connected()
  - disconected()
  - error()
- connectToHost()
- close()

Server

- new QTcpServer
- Connect sig/slot
  - newConnection()
- listen()
- → newConnection()
  - NextPendingConnection() : QTcpSocket

Please Note...

- **MyObject.connect() != MyObject.connectToHost()**
- BUT MAINLY
  - **MyObject.disconnect() != MyObject.disconnectFromHost()**

Let’s encrypt it

- QsslSocket
  - Both client & server
- Two handshake modes
  - Immediate
  - delayed
- setProtocol()
- setCiphers()
- connectToHostEncrypted()
- Signals
  - connected()
  - encrypted()

Network Access Manager (1)

- Used to manage Request / Reply messages over the network
- Support to **HTTP(S)**, **FTP** and **local file** protocols
  - **`http:// https:// ftp:// file://`**
- Support for cache, cookies, proxy
- **Async** API
- Three main objects:
  - `QNetworkAccessManager`
  - `QNetworkRequest`
  - `QNetworkReply`
- Bonus Object:
  - `QAuthenticator`

Network Access Manager (2)

- `QNetworkReply` object MUST be deleted by the user
  - **Do not delete** it **inside the `finished()` slot**: use `deleteLater()` instead
- `QObject::deleteLater` schedule the object to be delete
  - Object will be deleted when the control returns to the event loop
  - A deleted object emit **`destroyed()` signal**

Can I get you number?

- `QHostInfo` is a helper class
- Used to perform host name lookups (**DNS**)
  - **Reverse lookup** is supported too
- API is both **sync** and **async**
- API uses the underlaying **O.S. resolver**
  - It is *not possible* to specify a DNS server
- Async API uses assign an ID to each request
- abortHostLookup(ID) to abort a pending request

Multi-Threading Programming

With 

Sergio Borghese
NetResults Srl

The Qt Company

NetResults
Building the digital society

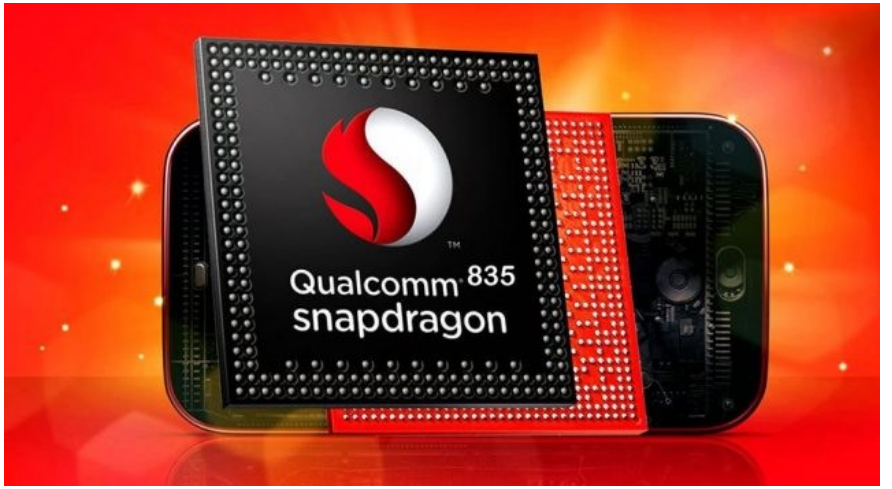
Agenda

- Concetti base sui thread
- Perché la programmazione Multithread
- Introduzione alla gestione dei thread
- Classi per creare e gestire i thread all'interno di una stessa applicazione
- Thread Pool
- Meccanismi di sincronizzazione (mutex & semafori)
- Coding tips:
  - Uso di QSocket, QTimer con i thread
  - Riconoscere i thread: dare un nome alle cose!

What is Multithreading?

- Definition
- Main characteristics
  - Concurrent execution *on multi-core hw*
    - Algorithms speedup
  - Light context switch
  - Simple communication
    - Shared state and resources
  - Contention (synchronization)
  - *Complexity*
  - *Hard Debugging*

Why MultiThreading?



Why Multi Threading ?

- Intel Core i9
  - 18 physical cores
  - 32 logical cores
- AMD Ryzen 2nd Gen
  - 32 physical cores
  - 64 logical cores
- Huawei Kirin 970
  - 8 cores
- Qualcomm Snapdragon 835
  - 8 cores

Reentrant != Thread Safe

- Thread safe **!**→ Reentrant
- Reentrant **!**→ Thread Safe
  - But...
- Qt Documentation specifies whether a class is:
  - Reentrant
  - Thread safe
  - None

Reentrant

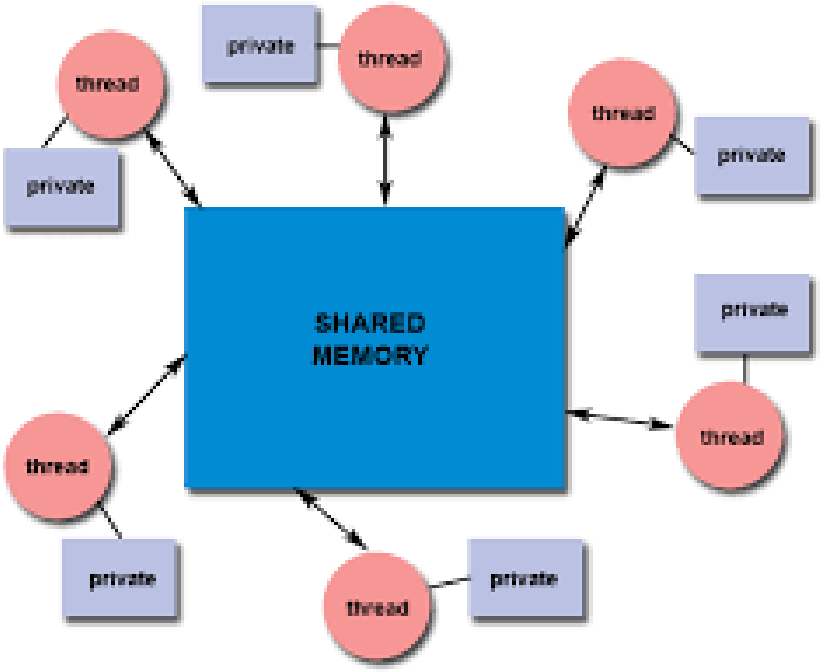
- *Can be interrupted in the middle of its execution and then safely be called again ("re-entered") before its previous invocations complete execution.*
- **Do not use static or global variables** in your function since those may be changed by time your function resumes
- Function must **not modify its own code** (e.g. some low level graphic routines may have "habit" to generate itself)
- **Do not call** any **non-reentrant functions**
- When to use re-entrant function?
  - Functions executed in interrupt context must be re-entrant.
  - Functions that will be **called from multiple threads/tasks must be re-entrant**

Thread Safe

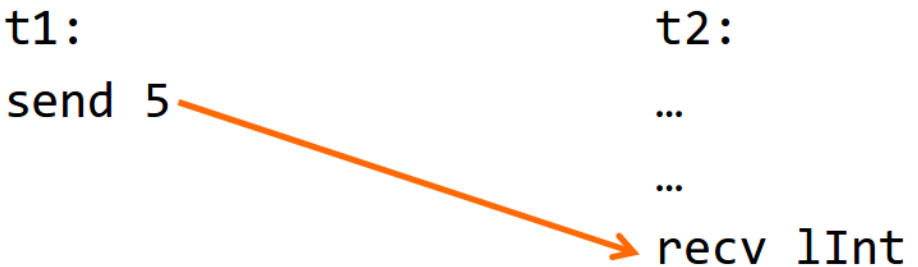
- Can be performed from multiple threads safely, even if the calls happen simultaneously on multiple threads.

Thread IPC Models

- Shared Memory



- Message Passing



The Qt Company

NetResults
Building the digital society

Parallel Algorithms

- Array Sum
- Fibonacci
- Merge Sort
- Matrix Multiplication

Know Your Threads

- **`QThread::currentThread()`**
- **`QThread::currentThreadId()`**
- Naming things
  - **`QObject::setObjectName()`**
  - **`QObject::objectName()`**

Using Threads in Qt

- Qt 5.6
- **QThread**
  - ***Subclass*** QThread class
  - Create worker object(s) and ***move it to the QThread***
- Qthread event loop
  - `exec()`
- Qthread Signals
  - `start()` → `started()` signal
  - `exit()` → `finished()` signal
- Each thread has a unique Id
  - `Qthread::currentThreadId()`

Subclassing Qthread (1/2)

- Subclass QThread
- Re-implement the method `QThread::run()`
  - Call `exec()` to start the thread event loop
- Instantiate the thread object(s)
- Start the thread object(s)
  - `QThread::start()`

Subclassing Qthread (2/2)

- QThread instance lives inside the thread that instatiated it
- **Only the `run()` method is executed in the new thread context**
- QThread slots are executed in the “old” thread
- start() & finished()
  - Should be used to communicate with other objects
- Call `exec()` to create an event loop in the thread

MoveToThread (1/2)

- Create a QThread
- Create a worker object (QObject)
- Move worker to thread
- Start the thread
- *Simple as that! Is it?*

MoveToThread (2/2)

- *Which code is executed from the worker object?*
- *When is the code executed?*

ThreadPool (1/2)

- Managed collection of pre-allocated thread
  - Performance
  - Simple thread management
- Each **`QApplication`** has a global thread pool
  - Retrive it with **`globalInstance()`**
- More ThreadPool could be created
- How to use it?
  - Subclass **`QRunnable`**
  - Start the runnable object

ThreadPool (2/2)

- How does it work?
  - start() adds the runnable object to a queue
  - When a thread from the pool is available it gets the next runnable object from the queue
    - Use priority variable to tune the queue

Synchronization via Mutex

- Shared resources MUST be protected
- Critical section: **`QMutex`**
  - Only one thread at a time
- Start critical section → **`QMutex::lock()`**
- Exit critical section → **`Qmutex::unlock()`**
- Deadlocks & how to avoid them
  - **`QMutexLocker`**
- `Mutex cannot be copied!`

Semaphores

- *The Little book of semaphores*
- Generalized mutex
  - A mutex is a semaphore intialized to 1
- Semaphore can be locked multiple times
  - How many time depends on the semaphore init value
- Semaphore operations
  - `acquire(n)` → decrement by n the semaphore's value;
    - ***blocks if n == 0***
  - `release(n)` → increment by n the semaphore's value
- acquire(n) → P(n) → wait()
- release(n) → V(n) → signal()

References

- https://www.cs.princeton.edu/courses/archive/fall10/cos597C/docs/threads1.pdf
- https://doc.qt.io/qt-5.6/index.html
- Mastering Qt5, Guillaume Lazar, Robin Penea, Packt Publishing
- http://blog.qt.io/blog/2010/06/17/youre-doing-it-wrong/
- Programmazione concorrente e distribuita, Paolo Ancilotti, Maurelio Boari, McGraw-Hill
- http://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf

Training Advices

- Interact / Interrupt
- If I’m not clear stop me and ask to repeat
- No question is stupid!
- Two way conversation is better
- Qt documentation is really good
  - I’m not going to read it! DIY :)
- **Strong emphasis on code**
- Excuse my French!
- Contact info: s.borghese@netresults.it

The Qt Company

NetResults
Building the digital society

Network Programming

With Qt

Sergio Borghese
NetResults Srl

The Qt Company

NetResults
Building the digital society

Regular gray #868482
Green #80c342

Agenda

- NetResults Srl :: a short intro
- Create TCP and UDP Sockets
- Sending and Receiving data
- Encrypted connections
  - QSslSocket
- A simple QSslServer
- The NetworkManager
- QHostInfo class

NetResults
Building the digital society

NetResults Srl :: a short intro

- Established in 2006
- Spin-off University of Pisa
  - Telecommunication Network Research group
- Strong skill on VoIP/MoIP, netwoking, network performance testing
- Employees: 20 and growing
  - With degree 16/20
  - With PhD 2/16

Qt The Qt Company

NetResults
Building the digital society

Quick Miscellanea

- Create Project in QtCreator
  - subprojects
- Kits
  - Different versions & platforms
- Code Snippet

TCP & UDP

- TCP & UDP are L4 protocols used on top of IP
- TCP
  - Connection oriented
  - Stream oriented
  - Reliable (higher latency)
- UDP
  - Message (datagram) oriented
  - Connectionless
  - Non reliable (lower latency)

Qt & sockets

- Enable network module in .pro
  - **QT += network**
- `QAbstractSocket`
  - `QTcpSocket`
  - `QUdpSocket`
  - `QSslSocket`
- API support for **sync** or **async** operations
  - *Do NOT Mix and Match*
- Sockets have an event-loop
  - **Do not move sockets between threads**

NetResults
Building the digital society

- create a project and show the QT += network

QUdpSocket Async API

- Create the udp socket
- Bind the socket
  - Useful for **multi-homed** systems
- Write data to socket
  - **`writeDatagram()`**
  - **`byteWritten()`** signal
- Read data on **`readyRead()`** signal
  - **`hasPendingDatagrams()`**
  - **`readDatagram()`**
    - MUST call this at least one time to get signaled again

UDP :: Send & Receive Data

TX

- `new QUdpSocket()`
- `bind()`
- `writeDatagram()`

RX

- `new QUdpSocket()`
- `bind()`
- → `readyRead()`
  - `readDatagram()`

The Qt Company

NetResults
Building the digital society

70-udp-socket/udp-tx-rx

01-write-datagram
02-read-datagram

QUdpSocket Sync API

- Create the udp socket
- Bind the socket
  - Useful for **multi-homed** systems
- Write data to socket
  - **`waitForBytesWritten(int msec)`**
- Read data
  - **`waitForReadyRead(int msec)`**
- **`-1`** → `no timeout`
- `! error ~= timeout`

Low Level Socket

- **socketDescriptor() : qintptr**
- **setSocketDescriptor(qintptr, SocketState)**

```
int val = IP_PMTUDISC_DO;
setsockopt(m_socket,
           SOL_IP,
           IP_MTU_DISCOVER,
           &val,
           sizeof(val));
```

TCP :: Send & Receive Data

Client

- `new QTcpSocket`
- `bind() [opt.]`
- `Connect sig/slot`
  - `connected()`
  - `disconected()`
  - `error()`
- `connectToHost()`
- `close()`

Server

- `new QTcpServer`
- `Connect sig/slot`
  - `newConnection()`
- `listen()`
- `→ newConnection()`
  - `NextPendingConnection() : QTcpSocket`

The Qt Company

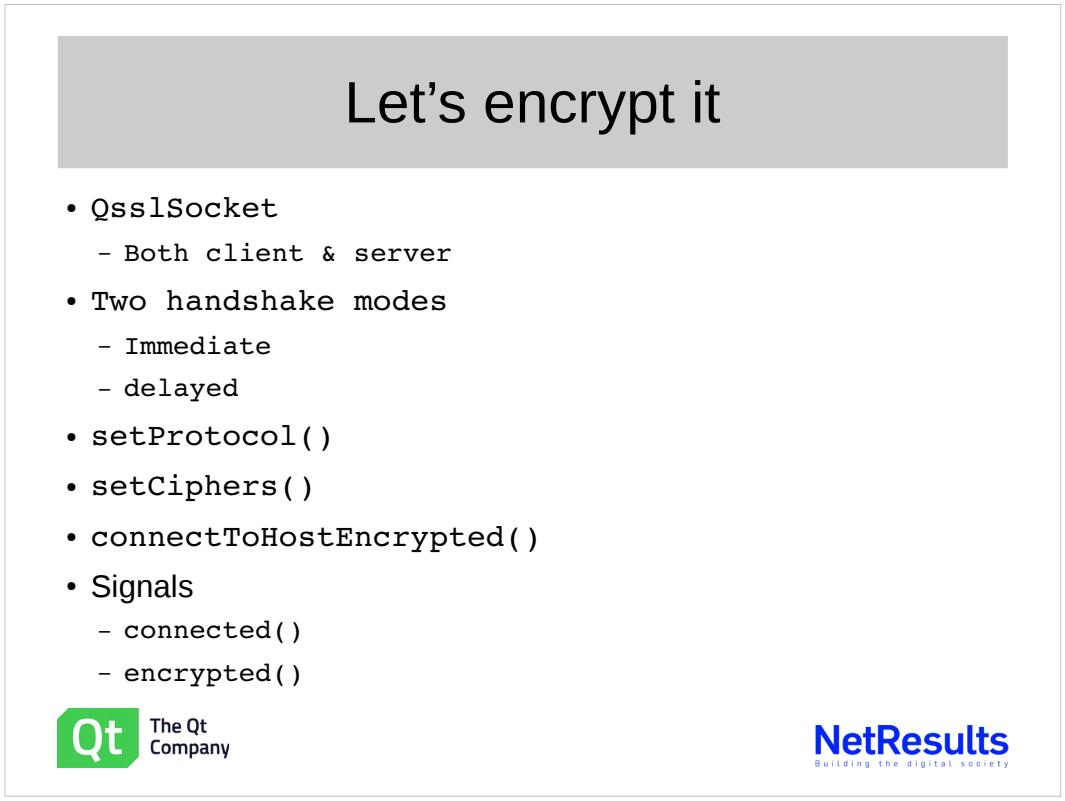
NetResults
Building the digital society

90-tcp-socket

01-buggy-tcp-send
02-tcp-sender-fixed
03-ts-server

Please Note...

- **MyObject.connect() != MyObject.connectToHost()**
- BUT MAINLY
  - **MyObject.disconnect() != MyObject.disconnectFromHost()**



openssl s_server -accept 5432 -cert selfsigned.crt -key selfsigned.key

- can write data before encrypted() has been emitted
  - data are buffered

90-tcp-socket

04-ssl-socket-errors
05-print-ssl-errors
06-ignore-ssl-errors

Network Access Manager (1)

- Used to manage Request / Reply messages over the network
- Support to **HTTP(S)**, **FTP** and **local file** protocols
  - **`http:// https:// ftp:// file://`**
- Support for cache, cookies, proxy
- **Async** API
- Three main objects:
  - `QNetworkAccessManager`
  - `QNetworkRequest`
  - `QNetworkReply`
- Bonus Object:
  - `QAuthenticator`

Qt The Qt Company

NetResults
Building the digital society

- it is the responsibility of the user to delete the QNetworkReply object at an appropriate time.
- Do not directly delete it inside the slot connected to finished().
- You can use the deleteLater() function.

50-qnetwokmanager

01-http-request
02-ftp-request
03-network-reply-error
04-url-authentication
05-authenticator
06-ssl-errors
07-ignore-ssl-errors

Network Access Manager (2)

- `QNetworkReply` object MUST be deleted by the user
  - **Do not delete** it **inside the `finished()` slot**: use `deleteLater()` instead
- `QObject::deleteLater` schedule the object to be delete
  - Object will be deleted when the control returns to the event loop
  - A deleted object emit **`destroyed()` signal**

Can I get you number?

- `QHostInfo` is a helper class
- Used to perform host name lookups (**DNS**)
  - **Reverse lookup** is supported too
- API is both **sync** and **async**
- API uses the underlaying **O.S. resolver**
  - It is *not possible* to specify a DNS server
- Async API uses assign an ID to each request
- abortHostLookup(ID) to abort a pending request

60-qhostinfo

01-sync-resolve
02-reverse-lookup
03-async-resolution

Multi-Threading Programming

With Qt

Sergio Borghese
NetResults Srl

The Qt Company

NetResults
Building the digital society

Regular gray #868482
Green #80c342

Agenda

- Concetti base sui thread
- Perché la programmazione Multithread
- Introduzione alla gestione dei thread
- Classi per creare e gestire i thread all'interno di una stessa applicazione
- Thread Pool
- Meccanismi di sincronizzazione (mutex & semafori)
- Coding tips:
  - Uso di QSocket, QTimer con i thread
  - Riconoscere i thread: dare un nome alle cose!

What is Multithreading?

- Definition
- Main characteristics
  - Concurrent execution *on multi-core hw*
    - Algorithms speedup
  - Light context switch
  - Simple communication
    - Shared state and resources
  - Contention (synchronization)
  - *Complexity*
  - *Hard Debugging*

Qt The Qt Company

NetResults
Building the digital society

https://en.wikipedia.org/wiki/Thread_(computing)

In computer science, a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system.[1] The implementation of threads and processes differs between operating systems, but in most cases a thread is a component of a process. Multiple threads can exist within one process, executing concurrently and sharing resources such as memory, while different processes do not share these resources. In particular, the threads of a process share its executable code and the values of its variables at any given time.

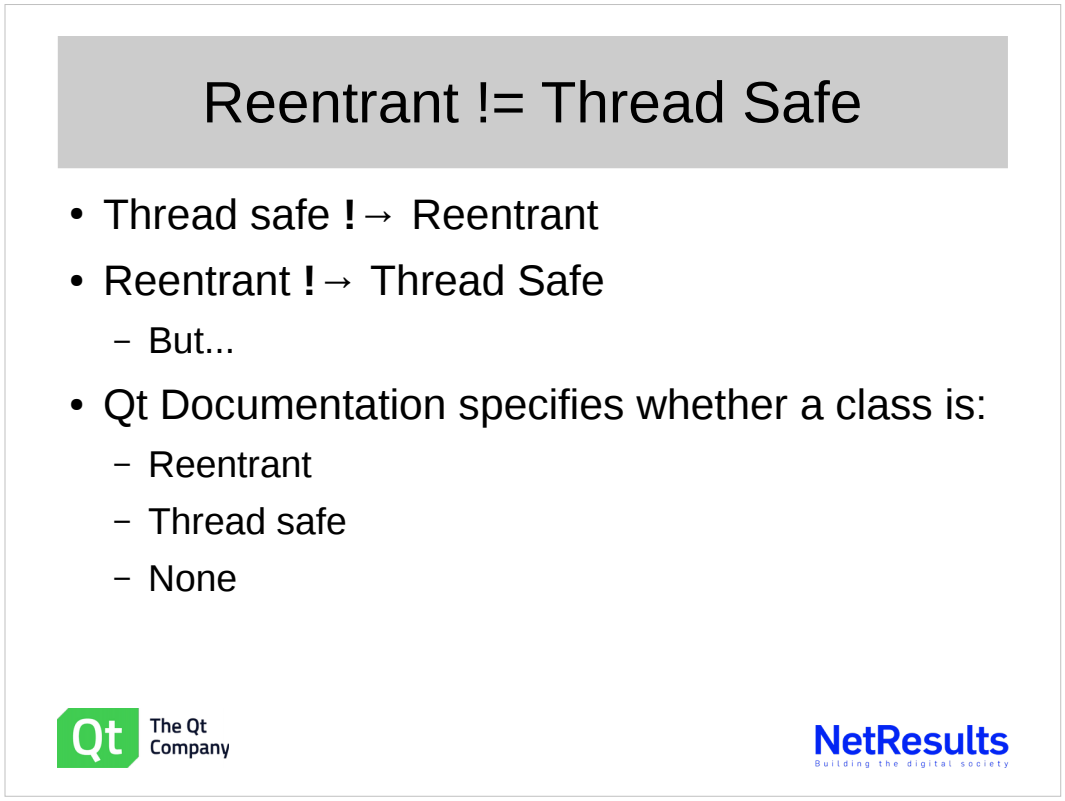
Why MultiThreading?

Qt The Qt Company

NetResults
Building the digital society

Why Multi Threading ?

- Intel Core i9
  - 18 physical cores
  - 32 logical cores
- AMD Ryzen 2nd Gen
  - 32 physical cores
  - 64 logical cores
- Huawei Kirin 970
  - 8 cores
- Qualcomm Snapdragon 835
  - 8 cores



subroutine is called reentrant if it can be interrupted in the middle of its execution and then safely be called again ("re-entered") before its previous invocations complete execution. Once the reentered invocation completes, the previous invocations will resume correct execution.

This definition of reentrancy differs from that of thread-safety in multi-threaded environments. A reentrant subroutine can achieve thread-safety,[1] but being reentrant alone might not be sufficient to be thread-safe in all situations. Conversely, thread-safe code does not necessarily have to be reentrant

An operation is "thread-safe" if it can be performed from multiple threads safely, even if the calls happen simultaneously on multiple threads.

An operation is re-entrant if it can be performed while

Reentrant

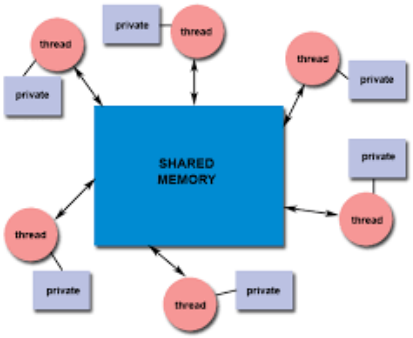
- *Can be interrupted in the middle of its execution and then safely be called again ("re-entered") before its previous invocations complete execution.*
- **Do not use static or global variables** in your function since those may be changed by time your function resumes
- Function must **not modify its own code** (e.g. some low level graphic routines may have "habit" to generate itself)
- **Do not call** any **non-reentrant functions**
- When to use re-entrant function?
  - Functions executed in interrupt context must be re-entrant.
  - Functions that will be **called from multiple threads/tasks must be re-entrant**

Thread Safe

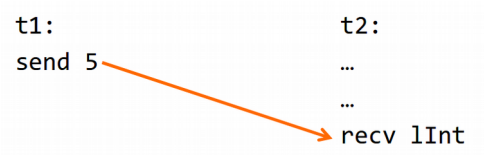
- Can be performed from multiple threads safely, even if the calls happen simultaneously on multiple threads.

Thread IPC Models

- Shared Memory



- Message Passing



The Qt Company

NetResults
Building the digital society

Parallel Algorithms

- Array Sum
- Fibonacci
- Merge Sort
- Matrix Multiplication

Know Your Threads

- **`QThread::currentThread()`**
- **`QThread::currentThreadId()`**
- Naming things
  - **`QObject::setObjectName()`**
  - **`QObject::objectName()`**

The Qt Company

NetResults
Building the digital society

01-LookMa_Threads

01-thread-id
02-thread-names

Using Threads in Qt

- Qt 5.6
- **QThread**
  - ***Subclass*** QThread class
  - Create worker object(s) and ***move it to the QThread***
- Qthread event loop
  - `exec()`
- Qthread Signals
  - `start()` → `started()` signal
  - `exit()` → `finished()` signal
- Each thread has a unique Id
  - `Qthread::currentThreadId()`

Subclassing Qthread (1/2)

- Subclass QThread
- Re-implement the method `QThread::run()`
  - Call `exec()` to start the thread event loop
- Instantiate the thread object(s)
- Start the thread object(s)
  - `QThread::start()`

02-ThreadSubclass

01-work-inside-started
02-work-inside-run
03-timer-started-from-wrong-thread
04-timer-direct-connection
05-timer-direct-queued

Subclassing Qthread (2/2)

- QThread instance lives inside the thread that instatiated it
- **Only the `run()` method is executed in the new thread context**
- QThread slots are executed in the “old” thread
- start() & finished()
  - Should be used to communicate with other objects
- Call `exec()` to create an event loop in the thread

MoveToThread (1/2)

- Create a QThread
- Create a worker object (QObject)
- Move worker to thread
- Start the thread
- *Simple as that! Is it?*

The Qt Company

NetResults
Building the digital society

10-MoveToThread/

one-thread-one-worker
one-thread-two-async-workers
one-thread-two-sync-workers

MoveToThread (2/2)

- *Which code is executed from the worker object?*
- *When is the code executed?*

ThreadPool (1/2)

- Managed collection of pre-allocated thread
  - Performance
  - Simple thread management
- Each **`QApplication`** has a global thread pool
  - Retrive it with **`globalInstance()`**
- More ThreadPool could be created
- How to use it?
  - Subclass **`QRunnable`**
  - Start the runnable object

Qt The Qt Company

NetResults
Building the digital society

20-threadPool

autodelete-off
autodelete-ownership-delete
tp-crash
tp-no-crash
tp-no-crash-logs

ThreadPool (2/2)

- How does it work?
  - start() adds the runnable object to a queue
  - When a thread from the pool is available it gets the next runnable object from the queue
    - Use priority variable to tune the queue

Synchronization via Mutex

- Shared resources MUST be protected
- Critical section: **`QMutex`**
  - Only one thread at a time
- Start critical section → **`QMutex::lock()`**
- Exit critical section → **`Qmutex::unlock()`**
- Deadlocks & how to avoid them
  - **`QMutexLocker`**
- `Mutex cannot be copied!`

NetResults
Building the digital society

30-Syncronization

01-unprotected-shared-resource
02-mutex-crash
03-global-mutex
04-mutex-deadlock
05-mutex-locker
06-talking-mutex-locker
07-mutex-copy
08-semaphore-101
09-rendevouz

Semaphores

- *The Little book of semaphores*
- Generalized mutex
  - A mutex is a semaphore intialized to 1
- Semaphore can be locked multiple times
  - How many time depends on the semaphore init value
- Semaphore operations
  - `acquire(n)` → decrement by n the semaphore's value;
    - ***blocks if n == 0***
  - `release(n)` → increment by n the semaphore's value
- acquire(n) → P(n) → wait()
- release(n) → V(n) → signal()

References

- https://www.cs.princeton.edu/courses/archive/fall10/cos597C/docs/threads1.pdf
- https://doc.qt.io/qt-5.6/index.html
- Mastering Qt5, Guillaume Lazar, Robin Penea, Packt Publishing
- http://blog.qt.io/blog/2010/06/17/youre-doing-it-wrong/
- Programmazione concorrente e distribuita, Paolo Ancilotti, Maurelio Boari, McGraw-Hill
- http://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf