

Programación Concurrente

# Qt y concurrencia

Lourdes Mas Lillo

# Qt

Framework multiplataforma orientado a objetos usado ampliamente para desarrollar programas que utilicen interfaz gráfica, pero también se usa para otros tipos de herramientas para línea de comandos y consolas de servidores sin necesidad de interfaz gráfica.

# Desarrollo

En 1991 TrollTech comienza su desarrollo.

Nokia, Digia

The Qt Company, Qt Project  
comunidad

# Licencia

Software libre y de código abierto,  
GPL 2.0 y 3.0  
LGPL 3.0

# Qt

Funciona en mayoria de plataformas de escritorio  
Moviles  
Sistemas embebidos  
Interfaces con aspecto nativo  
Soporta varios compiladores incluyendo gcc C++  
y Visual Studio

# Módulos

Qt Core	Modulo básico
Qt GUI	Módulo central para interfaces
Qt Widgets	Contiene clases para interfaces basadas en widgets
Qt QML	Modulo para QML y lenguajes Javascript
...	

# Módulo Qt Core

Para poder trabajar la concurrencia con Qt  
necesitamos el módulo Qt Core

Meta-objetos  
Concurrencia  
Threading  
contenedores

# Aplicaciones que usan Qt

Autodesk Maya (Autodesk Maya 2011 creada usando Qt por Nokia)

Google Earth (Qt WebKit)

TeamViewer (interfaz nativa Qt)

VirtualBox (Qt frontend)

Agencia Espacial Europea (solución multiplataforma para generar mapas digitales de Marte)



# Qthread

Para usar hilos en Qt: Qthread

- `#include <Qthread>`
- Qmake: `QT += core`
- Heredar de la clase `Qthread`

# Objeto Qthread: inicio

- Tiene un hilo de control
- `start(Priority prioridad)`: Comienza la ejecución llamando a `run()`, la gestión de la prioridad dependerá del SO
- `Run()`: inicio de ejecución de Qtreads
- `Exec()`: se llama desde `run()` y espera a que sea llamado `exit()`, devolviendo el valor que devuelve esa función

# Objeto Qthread: inicio

Prioridades:

- 0 IdlePriority
- 1 LowestPriority
- 2 LowPriority
- 3 NormalPriority
- 4 HighPriority
- 5 HighestPriority
- 6 TimeCriticalPriority
- 7 InheritPriority

Por defecto, como hilo padre

# Objeto Qthread: estado

QThread notifica mediante señal cuando el hilo empieza y termina:

- Started(): señal de inicio (justo antes del run())
- Finished(): señal de final (después de todos los eventos de hilo y antes de su eliminación)

Consulta:

- IsFinished()
- IsRunning()

## Objeto Qthread: fin

Se puede parar el hilo de diversas formas:

- `Exit(int codigo)`: el hilo termina devolviendo a través de `exec()`, por convención, un código de 0 indica éxito
- `Quit()`: indica al hilo que termine con código de 0
- `Terminate()`: Termina la ejecución del hilo, haya o no terminado inmediatamente, dependiendo del SO, para asegurar el buen funcionamiento, usar `wait()`. Al terminar todos los hilos que esperaran serán despertados
- `SetTerminationEnabled(bool enabled = true)`: se utiliza para habilitar o deshabilitar `terminate()`.

## Objeto Qthread: espera

- Wait(unsigned long tiempo = ULONG\_MAX):  
bloquea al hilo hasta que otro hilo termina la ejecución o hasta que un determinado tiempo en milisegundos ha pasado.
- sleep(unsigned long segundos)
- msleep(unsigned long milisegundos)
- usleep(unsigned long microsegundos)

# Compilación

gcc – o name file.cpp

```
[lourdes@lourdes-pc ProgramacionConcurrente]$ gcc -o sem semaforos.cpp
semaforos.cpp:1:10: fatal error: QThread: No such file or directory
#include <QThread>
         ^~~~~~
compilation terminated.
```

# Solución

Crear .pro y correr “qmake” para crear un  
makefile automáticamente  
Finalmente correr “make”



# Ejemplo: wait y QMutex

waits.cpp

# Ejemplo: semáforo

semaforos.cpp

# Hilos sin candados ni Mutex

Mutex funciona correctamente, pero lento por el candado:

- Bloquea hilos
- SO: más cambios de contexto, caros por limpiar de las caches los procesos

# Hilos sin candados ni Mutex

Ejemplo: programa dividido en procesos, para que la app no caiga si uno falla...

... si falla mientras está en la S.C puede causar un deadlock en la app

# Ejecución sin candado

Ejecucion desordenada: CPU ejecuta bloques de instrucciones de forma eficiente sin contar con el orden, esto puede causar inconsistencia.

# Hilos sin candados ni Mutex

Solución: operaciones atómicas

Qt : QatomicInt QatomicPointer

- Clases que envuelven un int o puntero para realizar operaciones atómicas sobre el.
- 
- Implementadas en ensamblador

# QAtomic

## Operaciones Básicas

- FetchAndAdd: incrementar valor
- FetchAndStore: Leer-modificar-escribir
- TestAndSet: comparar e intercambiar

# QAtomic

Ejemplo:

```
QatomicPointer<int> p;  
int x;  
x = 1;  
p.fetchAndStoreRelease(&x);
```

¿Por qué Release?



# QAtomic

Porque x debe tener el valor asignado para que podamos realizar la asignación atómica correctamente.

Uso de vallas.

# QAtomic

## Vallas

- Acquire: no se adelantarán a la operación atómica otras operaciones de lectura u escritura.
- Release: no se atrasarán a la operación atómica otras operaciones de lectura u escritura.
- Ordered: Una mezcla de las dos anteriores, más segura.
- Relaxed: Sin vallas.

# Qt Concurrent

Permite escribir programas multihilo sin usar primitivas de bajo nivel como mutex, waits, o semaforos.

Ajusta automáticamente el número de hilos según los procesadores disponibles, lo que permite que la aplicación escale.

# Qt Concurrent

Incluye APIs para procesamiento de listas en paralelo, usando programación funcional incluyendo:  
MapReduce y FilterReduce para memoria compartida

# Qt Concurrent

Qfuture: representa el resultado de una computación asincrona

Podemos trabajar con ella usando

- QFutureIterator: Permite iterar sobre los resultados de una QFuture
- QFutureWatcher:permite monituzar QFuture usando señales
- QFutureSynchronizer: sincroniza automáticamente varias QFuture

# Qt Concurrent

Es compatible con STL pero trabaja mejor con iteradores de acceso random como QList y QVector.

# Qt Concurrent

```
void hola(QString nombre){
    qDebug() << "Hola" << nombre << "soy" <<
    QThread::currentThread();
}

Int main(int argc, char **argv){
    QApplication app(argc,argv);
    QFuture<void> hola1 = run(hola, QString("Maria"));
    QFuture<void> hola2 = run(hola, Qstring("Pablo"));
    hola1.waitForFinished();
    hola2.waitForFinished();
}
```

# Bibliografía

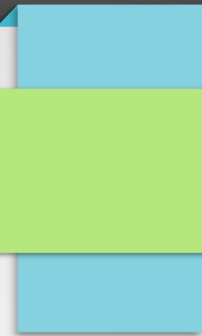
Qt documentación:

- <http://doc.qt.io/>

Artículos:

- <https://web.archive.org/web/20110723145736/http://qt.nokia.com/qt-in-use/autodesk/>
- <http://googlesystem.blogspot.com/2010/06/google-earth-includes-web-browser.html>
- <https://community.teamviewer.com/t5/Linux/Update-TeamViewer-13/td-p/24537>
- <https://www.virtualbox.org/wiki/VBoxMainLogging>
- <https://web.archive.org/web/20110723145813/http://qt.nokia.com/qt-in-use/story/customer/esa-european-space-agency>



A decorative graphic consisting of two light blue squares, one above the other, positioned on the right side of the slide.

# Muchas gracias

Lourdes Mas Lillo

Contacto:

[Lourdes.mas.lillo@gmail.com](mailto:Lourdes.mas.lillo@gmail.com)