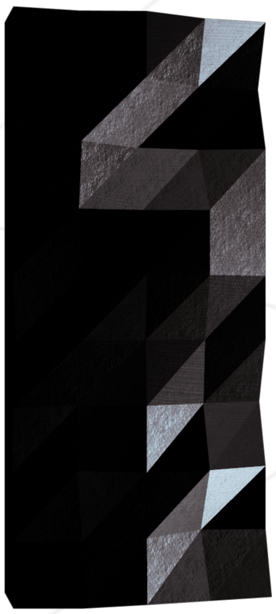


Multitasking

Threading in Qt

Qt training training@qt.io
42 Pedago Staff pedago@42.fr

Summary: Multitasking is a way to prevent the application from blocking event handling or freezing UI. There should exist no application doing these things in the first place. Smooth animates may require updates for each frame, though user may tolerate much longer delays (about 250 ms), before a GUI is expected to react to an input event. Qt has basically three approaches for multi-tasking: ***QThread*** *and* ***QRunnable*** *classes and the* ***QtConcurrent*** *namespace.*

Contents

Chapter I

General instructions

Unless explicitely specified, the following rules will apply every day of this Piscine.

- This subject is the one and only trustable source. Don't trust any rumor.
- This subject can be updated up to one hour before the turn-in deadline.
- The assignments in a subject must be done in the given order. Later assignments won't be rated unless all the previous ones are perfectly executed.
- Be careful about the access rights of your files and folders.
- You must follow the `turn-in process` for each assignment. The url of your `GIT` repository for this day is available on your intranet.
- Your assignments will be evaluated by your Piscine peers.
- In addition to your peers evaluation, a program called the "Moulinette" will also evaluate your assignments. Fully automated, The Moulinette is tough and unforgiving in its evaluations. As a consequence, it is impossible to bargain your grade with it. Uphold the highest level of rigor to avoid unpleasant surprises.
- You must not leave in your turn-in repository any file other than the ones explicitly requested by the assignments.
- You have a question? Ask your left neighbor. Otherwise, try your luck with your right neighbor.
- Every technical answer you might need is available in the `mans` or in the Qt Documentation, which is available both in `QtCreator` IDE and on `http://doc.qt.io`.
- Remember to use the Piscine forum of your intranet and also Slack!
- You must read the examples thoroughly. They can reveal requirements that are not obvious in the assignment's description.
- Use the latest Qt version. Qt version 5.10 or newer is recommended.

- Many Qt classes are available as standard C++ classes. Qt classes should be preferred to standard classes, as you are supposed to learn Qt.
- Basic Qt coding style should be used, so read `http://wiki.qt.io/Qt_Coding_Style` before writing any assignments.
- Deprecated macros, functions or classes must not be used.
- Any build system, such as `qmake`, `cmake` or Qt Build System, can be used in the assignments. Instructions are based on `qmake`.
- Any editor or IDE, supporting Qt, can be used. However, `QtCreator` usage is strongly recommended.
- By Thor, by Odin! Use your brain!!!

Chapter II

Assignment 00: Concurrent tasks

Implement a function, which calculates Fibonacci numbers. Use `QtConcurrent` and your function to calculate Fibonacci value for 5, 6, 7, 8, 9, and 10 concurrently. Use `QDebug` to print the calculated values on the debug console after each task of running the Fibonacci function, has finished. Use `QCoreApplication` instead of `QApplication`, as no widgets are needed in the assignment. After all tasks have finished, the application should exit without any user interaction.

Chapter III

Assignment 01: Concurrent container manipulation

Convert source code files or any text files concurrently, so that the files contain only lower case characters. Use the `main.cpp` skeleton in the `res` folder.

- The program must be able to convert several files concurrently. However, preventing concurrent manipulation of the same file, i.e., mutual exclusion, is not required.
- File content must be converted using threads in the thread pool. `QtConcurrent` provides the functionality you need.
- Read the file content to `QByteArray` and implement required changes to manipulate file content concurrently. Write the content back to a file.

`May be wise to write the output to a new file.`

Chapter IV

Assignment 02: Runnables

Use `QRunnable` to invert `QImage` pixels as shown below. Note that some white images on the left column are not shown.

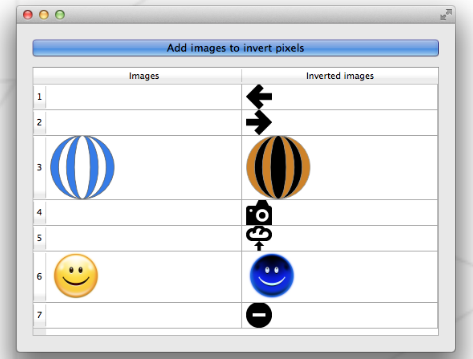


Figure IV.1: Images with inverted pixels

- The program template is given in the `res` folder.
- Subclass `QRunnable` and use `QImage` API to invert the pixels.
- Store the new image names to the model, so that they are shown in the table view.

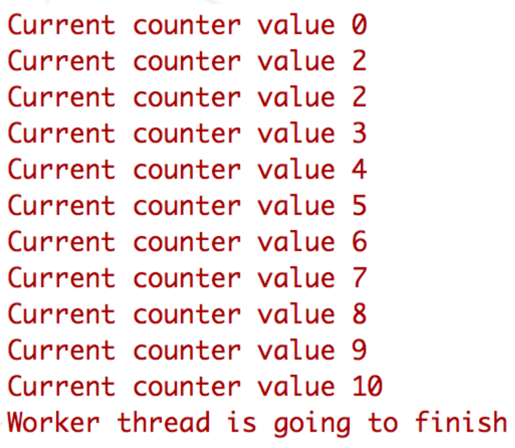
> `Obviously, you must not call any model member directly from another thread.`

Chapter V

Assignment 03: Mutual exclusion

Make the `ReentrantClass` in the `res` folder thread-safe.

- Create a simple widget application with a single `QPushButton`,
- When the button is clicked, increment the counter in the reentrant class. You may log the value into the debug console.
- Subclass `QThread` and reimplement the `run()` function. The thread should run an infinite loop, in which it increments and decrements the counter value in the reentrant class.
- If the counter is larger than 10, the thread should finish.
- The thread must be gracefully finished in case the application window is closed.
- The reentrant class must be accessed using `QSharedPointer` both in the main and child threads.
- The logged counter values may skip some numbers and log other ones twice, as the logging may take place between increment and decrement statements in the other thread.



```
Current counter value 0
Current counter value 2
Current counter value 2
Current counter value 3
Current counter value 4
Current counter value 5
Current counter value 6
Current counter value 7
Current counter value 8
Current counter value 9
Current counter value 10
Worker thread is going to finish
```

Figure V.1: Logged values in one example run

Chapter VI

Assignment 04: Thread synchronisation

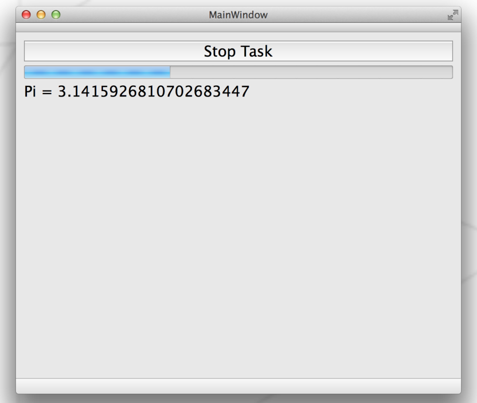
Global variables in C++ should be avoided. In Qt, there is a `Q_GLOBAL_STATIC` macro, which allows global variable creation in a thread-safe way. Use the macro to share data and objects between two threads. You are asked to implement a clasic producer-consumer application. One thread produces the data and the other one consumes the produced data. There is a `semaphores` example in Qt SDK, what you are allowed to use as a skeleton.

- Subclass the producer and consumer from `QThread`. Implement the classes in separate header and source code files. You must not implement the producer and consumer threads in `main.cpp`.
- Use two semaphores to control reading and writing to a shared buffer. The buffer content must be a random DNA alkali sequence, consisting of characters ACGT.
- Read and write semaphores as well as the shared buffer must be global static objects. You must use `Q_GLOBAL_STATIC`.
- Define a maximum buffer size of at least 8. The producer thread should finish, after it has filled the buffer 8 times. Obviously the producer must not write to the buffer, if it is full, before the consumer has read at least one item from the buffer. The consumer should finish after it has read the buffer 8 times.

Chapter VII

Assignment 05: Thread management

Implement a program, which calculates pi digits using a background thread. The calculation progress is shown in the progress bar and the value in the label as shown in the figure below.



```
Pi = 3.1415926810702683447
```

Figure VII.1: Pi calculator

The project skeleton is given in the `res` folder. Your task is to complete the functionality by completing the implementation of `MainWindow` and `PiCalculator`.

- When the user clicks on the push button, the calculation is started in a background thread, After the calculation is finished, you must cleanup the thread and create a new one, if the user re-starts the calculation.

- In the `PiCalculator` constructor, the observer is a progress bar and the label is the label in the UI.

- The thread must be gracefully finished and cleanup in all cases: when it finishes itself, when the window is closed or when the user stop the thread. You must not subclass `QThread`.

- In `PiCalculator`, exit from the infinite loop and the thread, if the calculation is finished. It can be finished in three ways. Either the user stop the calculation by clicking on the button, the value of `k` used in the calculation exceeds the maximum value or a timer expires.

- Notify the progress bar about the calculation.
- Set the pi value to a label. `QString::arg()` may be very useful function.
- Take care that the timer event is handled as well.