Multithreading in Qt

Doing it wrong, debugging it, doing it right

David Faure <david.faure@kdab.com>
Outline

• QThread
• Debugging race conditions
• Debugging deadlocks
• Unit-testing for thread-safety
How to use QThread

- A busy run()
- A default run()
- A wrapper
- Move the object
- Not using it
QThread – a busy run()

- Subclass QThread
- Reimplement run()
- Heavy calculation, or blocking on I/O
- WARNING: no slots called from other threads
QThread – a default run()

- Subclass QThread
- run() calls exec()
- Objects created by this thread, execute slots in it
- WARNING: not the QThread subclass itself!
- Too dangerous, prefer another solution

- Example: qthread_timer_wrong.cpp
QThread – a wrapper

- Solution: separate thread and worker object
- KDThreadRunner, from KDTools
- Worker created from run()
- Semaphores for synchronization both ways

- Example: qthread_timer.cpp + threadrunner.h
QThread – move the object

• Solution: separate thread and worker object
• Documentation changed in Qt 5
• Applies to Qt 4 too
• No QThread subclass
• Move worker to thread

• Example: qthread_timer_worker.cpp
What if I don't use QThread?

- CORBA, Rhapsody, boost, etc. create threads
- Will Qt handle events posted to QObjects in these threads?
  YES
  - Example: qobject_in_non_qt_thread.cpp
- What if all of Qt is used in a secondary thread, can we create widgets?
  YES, if main thread has no Qt.
  - Example: qt_in_thread.cpp
Race conditions

- What's a race condition
- How to detect race conditions?
  - Reading the code (when expert)
  - Frequently unreliable results (when lucky)
  - helgrind (everyone else)

- Example: RaceConditionExample, with 10 and with 100000
• Helgrind isn't perfect yet, especially for Qt code
  • Lock order detection (AB/BA) hits bug 243232 due to QOrderedMutexLocker.
  • glib has its own issues

alias helgrind=
"QT_NO_GLIB=1 valgrind --tool=helgrind --track-lockorders=no"
Setting up Qt for helgrind

- Qt code isn't perfect yet, especially for helgrind
  - qFlagLocation() race
    - apply http://www.davidfaure.fr/kde/qflaglocation-fix.diff
  - QEventLoop::exec() races with exit()
    - to be ported to an atomic data type
  - QFuture race in waitForResult
    - https://codereview.qt-project.org/38025
  - Qt5 atomics are seen as racy
    - apply http://www.davidfaure.fr/kde/qatomics-helgrind.diff

(work in progress)
• What's wrong with this code?

```cpp
bool MyClass::acceptString(const QString& str)
{
    QReadLocker locker(&m_lock);
    return m_regExp.exactMatch(str);
}
```

Example: qregexp_race.cpp

Very unreliable results. Memcheck says clean! Helgrind says clean, initially...

Discussion: reentrant vs thread-safe
Debugging deadlocks

- Deadlock!
- `gdb appname <pid>`
- `thread apply all bt`

Example: qmutex_order.cpp
Race prevention

- Testing code for thread-safety
- QtConcurrent::run in unit tests
- Case at hand: using a QUrl in multiple threads
- Unit test addition in tst_qurl.cpp
- export MALLOC_CHECK_=1 (or 3)
- repeat 10 ./tst_qurl testThreading
- gdb doesn't help [works, or deadlocks]
- helgrind doesn't help [warns in QFuture only]
Making helgrind see it

- Runnables finish too early, so they get reused
- See activeThreadCount()
- Helgrind needs to see concurrent threads!
- Solution: add qSleep(10)
- 100 concurrent threads
- Finally, helgrind finds the issue
- Implicit sharing + on-demand parsing
Conclusion

- Careful with subclassing QThread
- Test your library code with QtConcurrent
- Use helgrind on your multithreaded code
- Compile your code on linux, to use valgrind
- Help me making Qt helgrind-clean
- Questions?