Lecture 8. Multithreading Support in Qt

Cross-Platform Application Development

November 10, 2017

Multithreading Applications

Main Fields of Application

- Parallel algorithms.
- Networking applications, asynchronous input/output.
- User interface.

Thread Creation (Windows API)

Example

```
#include <windows.h>

DWORD WINAPI MyThreadProc(
  LPVOID pvData)
{
  // ...
  return 0;
}

int main()
{
```

Example (end)

```
  HANDLE hThread = CreateThread(
    NULL,          // attributes
    0,             // stack size
    MyThreadProc, // function
    NULL,          // parameter
    0,             // flags
    NULL);         // address of id
  // ...
  WaitForSingleObject(
    hThread, INFINITE);
  CloseHandle(hThread);
}    // main()
```

Multithreading Support in Qt

Level	Facilities
Lower	`QThread`, `QThreadPool`
Middle	`QtConcurrent::run()`
Upper	Qt Concurrent (Map, Filter, Reduce)

Table 1: Levels of multithreading support

Multithreading Technologies

Facility	`QThread`	Middle	Upper
Setting a thread priority	✔		
Starting a message processing loop	✔		
Receiving signals	✔		
Controlling a thread using signals	✔		✔
Tracking the state using `QFuture`		partially	✔
Internal facilities of starting/resuming/ canceling			✔

Table 2: comparison of multithreading technologies in Qt

QThread Class

Methods (Open Slots)

- **void** start(QThread::Priority nPriority = QThread::InheritPriority);
- **void** terminate();
- **void** quit(); (∼ exit(0);)

Methods (Basic Operations)

- **bool** wait(**unsigned long** ulTime = ULONG_MAX);
- **void** requestInterruption();
- **void** exit(**int** nReturnCode = 0);

QThread Class

Methods (Open Slots)

- `void start(QThread::Priority nPriority = QThread::InheritPriority);`
- `void terminate();`
- `void quit();` (∼ `exit(0);`)

Methods (Basic Operations)

- `bool wait(unsigned long ulTime = ULONG_MAX);`
- `void requestInterruption();`
- `void exit(int nReturnCode = 0);`

QThread Class (cont.)

Methods (Protected Operations)

- `int exec();`
- `virtual void run();`

Methods (properties)

- `bool isRunning() const;`
- `bool isFinished() const;`
- `bool isInterruptionRequested() const;`
- `QThread::Priority priority() const;/`
  `void setPriority(QThread::Priority nPriority);`
- `uint stackSize() const;/void setStackSize(uint uStackSize);`

QThread Class (cont.)

Methods (Protected Operations)

- `int exec();`
- `virtual void run();`

Methods (properties)

- `bool isRunning() const;`
- `bool isFinished() const;`
- `bool isInterruptionRequested() const;`
- `QThread::Priority priority() const;/`
  `void setPriority(QThread::Priority nPriority);`
- `uint stackSize() const;/void setStackSize(uint uStackSize);`

QThread Class (end)

Methods (Signals)

- `void started();`
- `void finished();`

Methods (Static Functions)

- `QThread *currentThread();`
- `int idealThreadCount();`
- `void sleep(unsigned long ulSecs);` (`msleep()`, `usleep()`)
- `void yieldCurrentThread();`

QThread Class (end)

Methods (Signals)

- **void** started();
- **void** finished();

Methods (Static Functions)

- QThread *currentThread();
- **int** idealThreadCount();
- **void** sleep(**unsigned long** ulSecs); (msleep(), usleep())
- **void** yieldCurrentThread();

Example

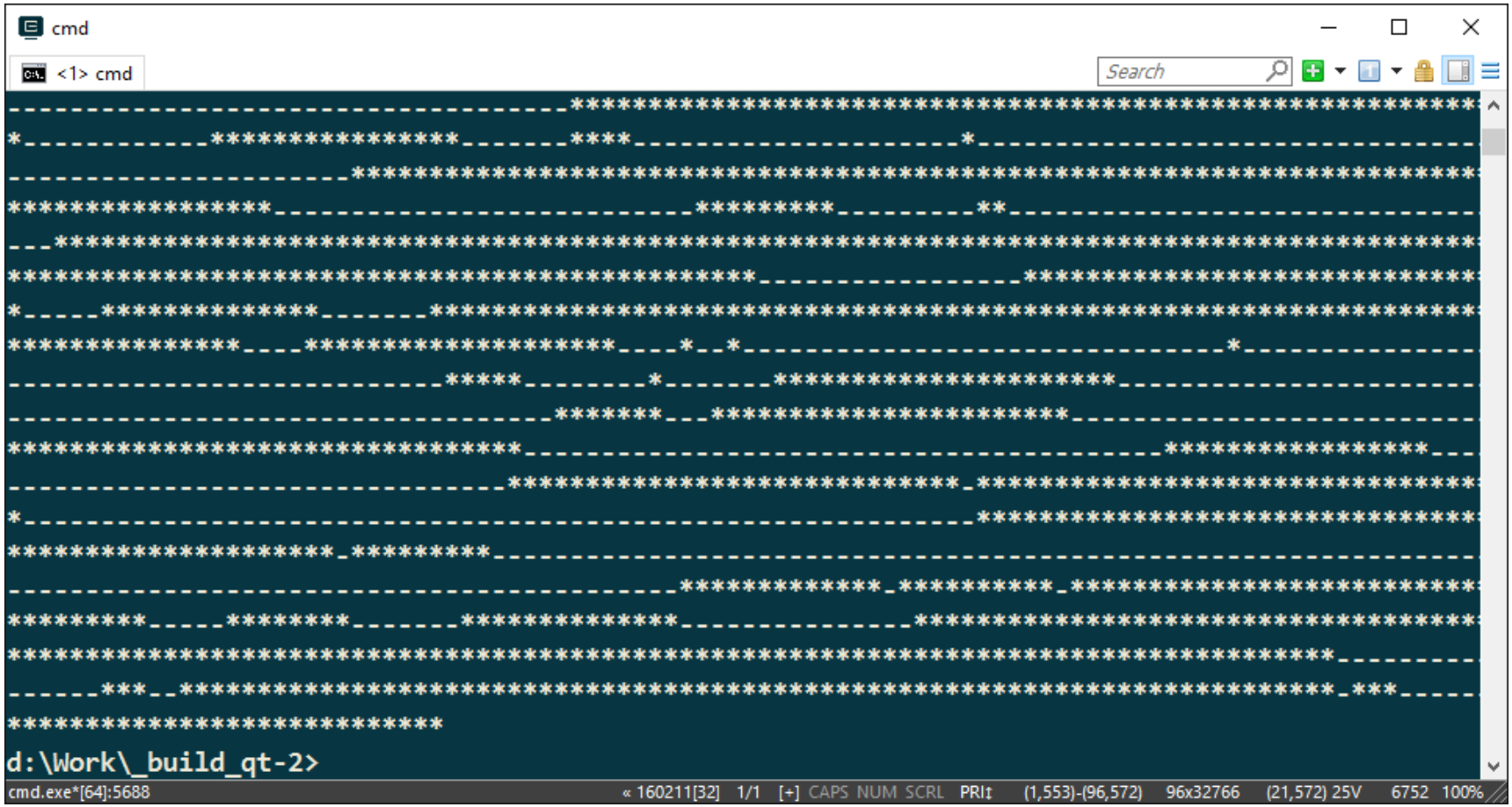

Figure 1: output of a simple console multithreaded program

Example (cont.)

Example (example-14.cpp)

```
#include <QThread>

#include <iostream>

class Thread : public QThread
{
public:
  //
  explicit Thread(char chData);
  //
```

Example (cont.)

```
protected:
  //
  virtual void run();
  //
private:
  //
  char m_chData;
};    // class Thread
```

Example (cont.)

Example (cont.)

```
Thread::Thread(char chData)
  : QThread(0),
    // QThread(QObject *pParent)
    m_chData(chData)
{
  //
}
```

Example (cont.)

```
void Thread::run()
{
  forever
  {
    std::cerr << m_chData;
    if (isInterruptionRequested())
      break;
  }
}
```

Example (end)

Example (cont.)

```
int main()
{
  Thread threadA('-');
  Thread threadB('*');
  //
  threadA.start();
  threadB.start();
  //
  QThread::sleep(5);
  //
```

Example (end)

```
  threadA.requestInterruption();
  threadB.requestInterruption();
  //
  threadA.wait();
  threadB.wait();
}    // main()
```

Thread Safety

Definition

Thread Safety: a property of a code assuming its correct functioning on simultaneous execution from within several threads.

Basic Techniques

- Atomic operations;
- Mutual exclusion;
- Reentrancy;
- Thread local storage.

Reentrancy

Definition

Reentrancy: a property of a function or program assuming its correct recurring call during execution (for example, recursively).

Calling Safety from within Different Qt Threads

- Reentrant functions — for different data;
- Thread-safe functions — possibly for shared data.

Requirements for a Reentrant Function

Reentrant Function

- Must not work with static/global non-constant data;
- Must not return an address of such data;
- Must work only with data passed from the caller part;
- Must not rely on locks of global resources;
- Must not modify its code;
- Must not call non-reentrant programs or subroutines.

Thread Local Storage

Definition

Thread Local Storage: (TLS) — a set of static/global variables that are local relative to a thread that uses them.

Restrictions on Qt Classes

Class Safety

- `QObject::connect()`, `QCoreApplication::postEvent()` functions, etc., all classes related to threads (`QMutex`, etc.) are thread-safe.
- `QObject` class and most of the classes not related to user interface are reentrant.
- Message-driven objects (timers, `Network` module. etc.) may be used only within one thread (otherwise, serialization to access is needed using mutexes, etc.)
- Objects of user interface classes (`QWidget`, etc.), `QCoreApplication::exec()` function, etc. are non-reentant and may be used only within the main thread.

Restrictions on Qt Classes (end)

Restrictions on `QObject`

- The instances of `QObject` and its subclasses must not be created before `QApplication`.
- The child objects for `QObject` etc. must be created in the same thread where their parent was created ($\Rightarrow$ their parent cannot be their thread — `QThread`).
- All instances of `QObject` etc. must be deleted before deleting the thread (`QThread`) in which they were created (for ex., the local variables of `QDerivedThread::run()`).
- The objects must be deleted in the thread that owns them. Alternatively, `QObject::deleteLater()` could be used, or `QObject::moveToThread()`.

Sending Signals

Constant	Thread	Call Moment	Blocked
`DirectConnection`	of a signal	Immediately after the signal	✔
`QueuedConnection`	of a receiving object	When the processing loop reaches	
`BlockingQueuedConnection`	of a receiving object	When the processing loop reaches	✔
`AutoConnection`	(Default) As either `Qt::DirectConnection`, or `Qt::QueuedConnection`, selected automatically.		

Table 3: types of signal-slot connections

Example

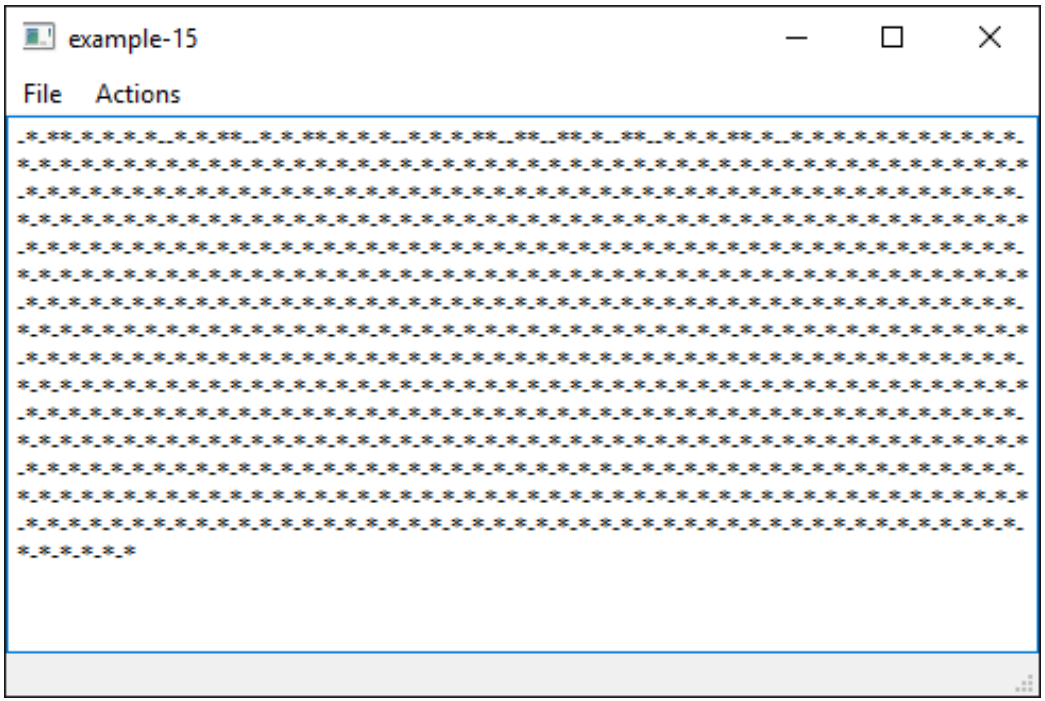

Figure 2: GUI application with two worker objects

Example

Example (worker.h)

```
class Worker : public QObject
{
  Q_OBJECT
  //
public:
  //
  Worker(
    char chData,
    volatile bool &rvbFinish);
  //
signals:
  //
```

Example (end)

```
  void resultReady(char chResult);
  void finish();
  //
public slots:
  //
  void onDoWork();
  //
private:
  //
  volatile bool &m_rvbFinish;
  char m_chData;
};
```

Example (cont.)

Example (worker.cpp)

```
#include "worker.h"

#include <QThread>

Worker::Worker(
  char chData, volatile bool &rvbFinish)
  : m_chData(chData),
    m_rvbFinish(rvbFinish)
{
  //
}
```

Example (cont.)

Example (worker.cpp, end)

```
void Worker::onDoWork()
{
  for (int i = 0; i < 1000; ++ i)
  {
    QThread::msleep(50);
    emit resultReady(m_chData);
    if (m_rvbFinish)
      break;
  }
  //
  emit finish();
}
```

Example

Example (main-window.h)

```
#include "ui_main-window.h"

#include <QMainWindow>
#include <QThread>

class MainWindow :
  public QMainWindow,
  public Ui::MainWindow
{
  Q_OBJECT
  //
```

Example (cont.)

```
public:
  //
  MainWindow(QWidget *pParent = 0);
  ~MainWindow();
  //
private slots:
  //
  void on_m_pActionStart_triggered();
  void onHandleResultReady(
    char chResult);
  void onHandleFinish();
```

Example (cont.)

Example (main-window.cpp, end)

```
  //
private:
  //
  volatile bool m_vbFinish;
  QThread m_aWorkThreads[2];
};    //  class MainWindow
```

Example (cont.)

Example (main-window.cpp)

```
#include "main-window.h"
#include "worker.h"

MainWindow::MainWindow(QWidget *pParent)
  : QMainWindow(pParent),
    //
    m_vbFinish(false)
{
  setupUi(this);
  //
```

Example (cont.)

Example (main-window.cpp, cont.)

```
setCentralWidget(m_pTextEdit);
setAttribute(Qt::WA_DeleteOnClose);
connect(
  m_pActionExit, SIGNAL(triggered()),
  this, SLOT(close()));
//
m_pTextEdit->setWordWrapMode(
  QTextOption::WrapAnywhere);
//
```

Example (cont.)

Example (main-window.cpp, cont.)

```
char achData[] = {'-', '*'};
for (int i = 0; i < 2; ++ i)
{
  Worker *pWorker = new Worker(
    achData[i], m_vbFinish);
  pWorker->moveToThread(&m_aWorkThreads[i]);
  connect(
    &m_aWorkThreads[i], &QThread::finished,
    pWorker, &QObject::deleteLater);
  connect(
    m_pActionStart, &QAction::triggered,
    pWorker, &Worker::onDoWork);
```

Example (cont.)

Example (main-window.cpp, cont.)

```
    connect(
      pWorker, &Worker::resultReady,
      this, &MainWindow::onHandleResultReady);
    connect(
      pWorker, &Worker::finish,
      this, &MainWindow::onHandleFinish);
    m_aWorkThreads[i].start();
  }    // for (int i = 0; i < 2; ++ i)
}    //  MainWindow::MainWindow()
```

Example (cont.)

Example (worker.cpp, cont.)

```
MainWindow::~MainWindow()
{
  m_vbFinish = true;
  for (int i = 0; i < 2; ++ i)
  {
    m_aWorkThreads[i].quit();
    m_aWorkThreads[i].wait();
  }
}
```

Example (cont.)

Example (worker.cpp, cont.)

```
void MainWindow::on_m_pActionStart_triggered()
{
  m_pStatusBar->showMessage("Working...");
}

void MainWindow::onHandleResultReady(char chResult)
{
  m_pTextEdit->setPlainText(
    m_pTextEdit->toPlainText() + chResult);
  m_pTextEdit->moveCursor(QTextCursor::End);
}
```

Example (end)

Example (worker.cpp, end)

```
void MainWindow::onHandleFinish()
{
  m_pStatusBar->showMessage("Finished", 5000);
}
```

Example

Example (example-16.cpp)

```
#include <QThread>
#include <QMutex>
#include <QMutexLocker>

#include <iostream>

class Thread : public QThread
{
public:
  //
  Thread();
  //
```

Example (cont.)

```
public:
  //
  int getData() const;
  void incData();
  void decData();
  //
protected:
  //
  virtual void run();
  //
```

Example (cont.)

Example (cont.)

```
private:
  //
  int m_nData;
  QMutex m_Mutex;
};    // class Thread

Thread::Thread()
  : m_nData(0)
{
  //
}
```

Example (cont.)

```
int Thread::getData() const
{
  return m_nData;
}

void Thread::incData()
{
  QMutexLocker lock(&m_Mutex);
  ++ m_nData;
}
```

Example (cont.)

Example (cont.)

```
void Thread::decData()
{
  QMutexLocker lock(&m_Mutex);
  -- m_nData;
}
```

Example (cont.)

```
void Thread::run()
{
  for (int i = 0; i < 1000; ++ i)
  {
    incData();
    std::cerr << getData() << ' ';
  }
}
```

Example (end)

Example (example-16.cpp, end)

```
int main()
{
  Thread thread;
  thread.start();
  //
  for (int i = 0; i < 1000; ++ i)
  {
    thread.decData();
    std::cerr << thread.getData() << ' ';
  }
  //
  thread.wait();
}    // main()
```

A Reader/Writer Problem

Problem Formulation

- Reading data is possible by any number of readers simultaneously.
- Writing data is possible by only one writer simultaneously.
- During the writing process, no one reader has access to data.
- Reading data is impossible if, at least, one writer has manifested its intention of writing.

Example

Example (example-17.cpp)

```
#include <QThread>
#include <QReadWriteLock>
#include <QReadLocker>
#include <QWriteLocker>
#include <QScopedArrayPointer>

#include <iostream>
#include <set>
```

Example (cont.)

```
typedef std::set <int> IntSet;

IntSet g_Bitcoins;
QReadWriteLock g_Lock;

inline bool divides(
  int n1, int n2)
{
  return (n1 % n2 == 0);
}
```

Example (cont.)

Example (cont.)

```
class Thread : public QThread
{
protected:
  //
  virtual void run();
};    // class Thread
```

Example (cont.)

```
void Thread::run()
{
  int n = 2;
  forever
  {
    ++ n;
    bool bFound = true;
    {
      QReadLocker lock(&g_Lock);
```

Example (cont.)

Example (cont.)

```
    IntSet::const_iterator
      i = g_Bitcoins.begin(),
      e = g_Bitcoins.end();
    for (; i != e; ++ i)
      if (divides(n, *i))
      {
        bFound = false;
        break;
      }
  }
  //
```

Example (cont.)

```
  if (bFound)
  {
    QWriteLocker lock(&g_Lock);
    IntSet::const_iterator
      i = g_Bitcoins.begin(),
      e = g_Bitcoins.end();
    while (i != e)
      if (divides(*i, n))
        g_Bitcoins.erase(i ++);
      else
        ++ i;
```

Example (cont.)

Example (example-17.cpp, cont.)

```
      //
      g_Bitcoins.insert(n);
    }    // if (bFound)
    //
    if (isInterruptionRequested())
      break;
  }    // forever
}    // Thread::run()
```

Example (cont.)

Example (example-17.cpp, cont.)

```
int main()
{
  g_Bitcoins.insert(2);
  //
  int nThreads = QThread::idealThreadCount();
  if (nThreads < 1)
    nThreads = 2;
  //
  QScopedArrayPointer <Thread> threads(new Thread[nThreads]);
  //
  for (int i = 0; i < nThreads; ++ i)
    threads[i].start();
```

Example (cont.)

Example (example-17.cpp, cont.)

```
//
QThread::sleep(10);
//
for (int i = 0; i < nThreads; ++ i)
  threads[i].requestInterruption();
//
for (int i = 0; i < nThreads; ++ i)
  threads[i].wait();
//
```

Example (end)

Example (example-17.cpp, end)

```
  IntSet::const_iterator
    i = g_Bitcoins.begin(),
    e = g_Bitcoins.end();
  for (; i != e; ++ i)
    std::cout << ' ' << *i;
  //
  std::cout << std::endl;
}    // main()
```

Circular Buffer

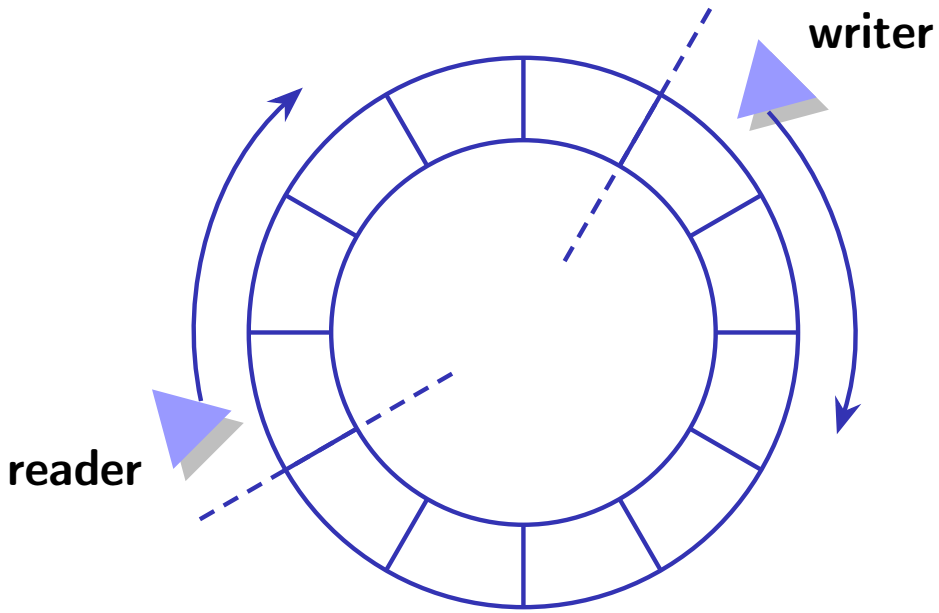

Figure 3: concept of circular buffer

Example

Example (example-18.cpp)

```
#include <QThread>
#include <QSemaphore>

#include <iostream>
#include <cmath>

const int NUM_BLOCKS = 10;
const int BLOCK_SIZE = 8;
const int AMPLITUDE = 100;
const int PERIOD = 30;
const int INTERVAL = 3 * PERIOD;
```

Example (cont.)

Example (example-18.cpp, cont.)

```
const int BUFFER_SIZE = NUM_BLOCKS * BLOCK_SIZE;

int g_anBuffer[BUFFER_SIZE];
QSemaphore g_FreeSpace(NUM_BLOCKS);
QSemaphore g_UsedSpace(0);

inline bool divides(
  int n1, int n2)
{
  return (n1 % n2 == 0);
}
```

Example (cont.)

Example (example-18.cpp, cont.)

```
class Thread : public QThread
{
protected:
  //
  virtual void run();
};    // class Thread
```

Example (cont.)

Example (example-18.cpp, cont.)

```
void Thread::run()
{
  for (int t = 0; t <= INTERVAL; ++ t)
  {
    int i = t % BUFFER_SIZE;
    if (divides(i, BLOCK_SIZE))
      g_FreeSpace.acquire();
    //
```

Example (cont.)

Example (example-18.cpp, cont.)

```
    g_anBuffer[i] =
      t == INTERVAL ?
      AMPLITUDE + 1 :
      AMPLITUDE * std::sin(1.0 * t / PERIOD);
    //
    if (t == INTERVAL || divides(i + 1, BLOCK_SIZE))
      g_UsedSpace.release();
  }    // forever
}    // Thread::run()
```

Example (cont.)

Example (example-18.cpp, cont.)

```
int main()
{
  Thread thread;
  thread.start();
  //
  int i = 0, nData;
  do
  {
    if (divides(i, BLOCK_SIZE))
      g_UsedSpace.acquire();
    //
    nData = g_anBuffer[i];
    std::cout << " (" << i << ", " << nData << ")" << std::flush;
```

Example (cont.)

Example (example-18.cpp, cont.)

```
    //
    i = (i + 1) % BUFFER_SIZE;
    //
    if (nData > AMPLITUDE || divides(i, BLOCK_SIZE))
    {
      g_FreeSpace.release();
      std::cout << " |" << std::flush;
    }
  }
  while (nData <= AMPLITUDE);
```

Example (end)

Example (example-18.cpp, end)

```
  //
  thread.wait();
  //
  std::cout << std::endl;
}    // main()
```

Spurious and Stolen Wake-ups

Definitions

Spurious Wakeup: a wakeup caused by accidental reasons instead of the signaling operation.

Stolen Wakeup: a wakeup after which the current thread was preempted and another thread executing before continuing the first thread changed the condition.

Example

Example (example-19.cpp)

```
const int MY_MAX_QUEUE = 50;
const int MY_NUM_PRODUCERS = 6;
const int MY_NUM_CONSUMERS = 3;

QQueue <int> g_Queue;
QMutex g_MutexQueue;
QWaitCondition g_CondNotEmpty;
QWaitCondition g_CondNotFull;
volatile bool g_vbRun = true;

// ...
```

Example (cont.)

Example (example-19.cpp, cont.)

```
void Producer::run()
{
  forever
  {
    QThread::msleep(std::rand() % 100);
    //
    QMutexLocker lock(&g_MutexQueue);
    while (g_Queue.size() == MY_MAX_QUEUE && g_vbRun)
      g_CondNotFull.wait(&g_MutexQueue);
    //
    if (!g_vbRun)
      break;    // lock.unlock()
```

Example (cont.)

Example (example-19.cpp, cont.)

```
    //
    static int s_nItem = 0;
    s_nItem = s_nItem % MY_MAX_QUEUE + 1;
    g_Queue.enqueue(s_nItem);
    std::cout <<
      "(p " << m_nNum << ", " << g_Queue.back() << ") " << std::flush;
    //
    g_CondNotEmpty.wakeOne();
  }    // forever
}    // Producer::run()
```

Example (cont.)

Example (example-19.cpp, cont.)

```
// ...

void Consumer::run()
{
  forever
  {
    QMutexLocker lock(&g_MutexQueue);
    while (g_Queue.empty() && g_vbRun)
      g_CondNotEmpty.wait(&g_MutexQueue);
    //
    if (!g_vbRun && g_Queue.empty())
      break;
```

Example (cont.)

Example (example-19.cpp, cont.)

```
    //
    std::cout <<
      "(c " << m_nNum << ", " << g_Queue.head() << ") " << std::flush;
    g_Queue.dequeue();
    //
    g_CondNotFull.wakeOne();
  }    // forever
}    // Consumer::run()
```

Example (cont.)

Example (example-19.cpp, cont.)

```
int main()
{
  QObject parent;
  QList <QThread *> threads;
  for (int i = 0; i < MY_NUM_PRODUCERS; ++ i)
    threads.append(new Producer(i + 1, &parent));
  //
  for (int i = 0; i < MY_NUM_CONSUMERS; ++ i)
    threads.append(new Consumer(i + 1, &parent));
  //
```

Example (cont.)

Example (example-19.cpp, cont.)

```
std::for_each(
  threads.begin(), threads.end(),
  std::bind2nd(
    std::mem_fun(&QThread::start), QThread::InheritPriority));
//
QThread::sleep(3);
//
g_vbRun = false;
g_CondNotEmpty.wakeAll();
g_CondNotFull.wakeAll();
```

Example (end)

Example (example-19.cpp, end)

```
  //
  std::for_each(
    threads.begin(), threads.end(),
    std::bind2nd(std::mem_fun(&QThread::wait), ULONG_MAX));
}    // main()
```

Thread Local Storage

Example

```
QThreadStorage <Data> g_Storage;

void setData(int n)
{
  if (!g_Storage.hasLocalData())
    g_Storage.setLocalData(Data());
  //
  g_Storage.localData().m_nVal = n;
}
```

Example

Figure 4: application with a working thread

Example

Example (main-window.h)

```
class MainWindow : public QMainWindow, public Ui::MainWindow
{
  Q_OBJECT
  //
public:
  //
  MainWindow();
  //
public slots:
  //
  void updateProgress(int nPercent);
  //
```

Example (cont.)

Example (main-window.h, cont.)

```
protected:
  //
  virtual void closeEvent(QCloseEvent *pEvent);
  //
private slots:
  //
  void on_m_pActionOpen_triggered();
  void on_m_pActionBlur_triggered();
  void on_m_pActionCancel_triggered();
  void onThreadStarted();
  void onThreadFinished();
  void onThreadCanceled();
```

Example (cont.)

Example (main-window.h, end)

```
  //
private:
  //
  QImage m_Image;
  WorkerThread *m_pThreadWork;
  bool m_bCanceled;
};    //  class MainWindow
```

Example (cont.)

Example (main-window.cpp)

```
MainWindow::MainWindow()
{
  setupUi(this);
  //
  m_pAreaImage->setBackgroundRole(QPalette::Dark);
  m_pAreaImage->setWidget(m_pLabelImage);
  m_pAreaImage->setWidgetResizable(true);
  setCentralWidget(m_pAreaImage);
  //
```

Example (cont.)

Example (main-window.cpp, cont.)

```
m_pAreaImage->addAction(m_pActionOpen);
m_pAreaImage->addAction(m_pActionExit);
//
m_pProgressBar->hide();
m_pButtonCancel->hide();
statusBar()->addWidget(m_pProgressBar);
statusBar()->addWidget(m_pButtonCancel);
//
```

Example (cont.)

Example (main-window.cpp, cont.)

```
  m_pThreadWork = new WorkerThread(this);
  connect(
    m_pThreadWork, SIGNAL(started()),
    this, SLOT(onThreadStarted()));
  connect(
    m_pThreadWork, SIGNAL(finished()),
    this, SLOT(onThreadFinished()));
  connect(
    m_pThreadWork, SIGNAL(canceled()),
    this, SLOT(onThreadCanceled()));
}    // MainWindow::MainWindow()
```

Example (cont.)

Example (main-window.cpp, cont.)

```
void MainWindow::updateProgress(int nPercent)
{
  m_pProgressBar->setValue(nPercent);
}

void MainWindow::closeEvent(QCloseEvent *pEvent)
{
  pEvent->accept();
  m_pThreadWork->stop();
}
```

Example (cont.)

Example (main-window.cpp, cont.)

```
void MainWindow::on_m_pActionOpen_triggered()
{
  QString fileName = QFileDialog::getOpenFileName(this);
  if (!fileName.isEmpty())
    m_pThreadWork->startLoadFile(fileName);
}

void MainWindow::on_m_pActionBlur_triggered()
{
  m_pThreadWork->startBlurImage(m_Image);
}
```

Example (cont.)

Example (main-window.cpp, cont.)

```
void MainWindow::on_m_pActionCancel_triggered()
{
  m_pThreadWork->stop();
}
```

Example (cont.)

Example (main-window.cpp, cont.)

```
void MainWindow::onThreadStarted()
{
  m_pActionOpen->setEnabled(false);
  m_pActionBlur->setEnabled(false);
  m_pProgressBar->setValue(0);
  m_pProgressBar->show();
  m_pButtonCancel->show();
  m_pActionCancel->setEnabled(true);
  QApplication::setOverrideCursor(Qt::WaitCursor);
  //
  m_bCanceled = false;
}
```

Example (cont.)

Example (main-window.cpp, cont.)

```
void MainWindow::onThreadFinished()
{
  m_pActionOpen->setEnabled(true);
  m_pActionBlur->setEnabled(true);
  m_pProgressBar->hide();
  m_pButtonCancel->hide();
  m_pActionCancel->setEnabled(false);
  QApplication::restoreOverrideCursor();
  //
```

Example (cont.)

Example (main-window.cpp, cont.)

```
  const QImage *pcImage = m_pThreadWork->getResultImage();
  if (pcImage)
  {
    m_Image = *pcImage;
    m_pLabelImage->setPixmap(QPixmap::fromImage(m_Image));
  }
  //
  m_pStatusBar->showMessage(
    m_bCanceled ? "Canceled" : "Finished");
}    // MainWindow::onThreadFinished()
```

Example (cont.)

Example (main-window.cpp, end)

```
void MainWindow::onThreadCanceled()
{
  m_bCanceled = true;
}
```

Example (cont.)

Example (worker-thread.h)

```
class WorkerThread : /* private */ public QThread
{
  Q_OBJECT
  //
public:
  //
  WorkerThread(MainWindow *pWindow);
  ~WorkerThread();
  //
public:
  //
  const QImage *getResultImage() const;
  //
```

Example (cont.)

Example (worker-thread.h, cont.)

```
public:
  //
  void startLoadFile(const QString &rcFilePath);
  void startBlurImage(const QImage &rcImage);
  void stop();
  //
signals:
  //
  void canceled();
  //
private:
  //
```

Example (cont.)

Example (worker-thread.h, end)

```
  enum Operation
  {
    LoadFile,
    BlurImage
  };
  //
  MainWindow *m_pWindow;
  Operation m_nOperation;
  QString m_FilePath;
  QScopedPointer <QImage> m_ptrInput, m_ptrResult;
  //
  virtual void run();
};    //  class WorkerThread
```

Example (cont.)

Example (worker-thread.cpp)

```
const QImage *WorkerThread::getResultImage() const
{
  assert(isFinished());
  //
  return m_ptrResult.data();
}
```

Example (cont.)

Example (worker-thread.cpp, cont.)

```
void WorkerThread::startLoadFile(const QString &rcFilePath)
{
  if (isRunning())
    return;
  //
  m_nOperation = LoadFile;
  m_FilePath = rcFilePath;
  m_ptrInput.reset(new QImage);
  QThread::start();
}
```

Example (cont.)

Example (worker-thread.cpp, cont.)

```
void WorkerThread::startBlurImage(const QImage &rcImage)
{
  if (isRunning())
    return;
  //
  m_nOperation = BlurImage;
  m_ptrInput.reset(new QImage(rcImage));
  QThread::start();
}
```

Example (cont.)

Example (worker-thread.cpp, cont.)

```
void WorkerThread::stop()
{
  requestInterruption();
}
```

Example (cont.)

Example (worker-thread.cpp, cont.)

```
void WorkerThread::run()
{
  switch (m_nOperation)
  {
    case LoadFile:
      //
      if (m_ptrInput->load(m_FilePath))
        m_ptrResult.reset(new QImage(*m_ptrInput));
      //
      break;
      //
```

Example (cont.)

Example (worker-thread.cpp, cont.)

```
case BlurImage:
  //
  if (m_ptrInput)
  {
    m_ptrResult.reset(new QImage(*m_ptrInput));
    //
    const int cm = m_ptrInput->width();
    const int cn = m_ptrInput->height();
    const int cnTotal = cm * cn;
    const int cnWindow = 5;
    int nCurrent = 0, nPercent, nPrev = 0;
    for (int i = 0; i < cm; ++ i)
      for (int j = 0; j < cn; ++ j)
```

Example (cont.)

Example (worker-thread.cpp, cont.)

```
{
  std::vector <int> reds, greens, blues;
  for (int k = i - cnWindow; k < i + cnWindow; ++ k)
    if (k >= 0 && k < cm)
      for (int l = j - cnWindow; l < j + cnWindow; ++ l)
        if (l >= 0 && l < cn)
        {
          QColor color = m_ptrInput->pixel(k, l);
          reds.push_back(color.red());
          greens.push_back(color.green());
          blues.push_back(color.blue());
        }
```

Example (cont.)

Example (worker-thread.cpp, cont.)

```
//
QColor color(
  std::accumulate(reds.begin(), reds.end(), 0) /
    reds.size(),
  std::accumulate(greens.begin(), greens.end(), 0) /
    greens.size(),
  std::accumulate(blues.begin(), blues.end(), 0) /
    blues.size());
m_ptrResult->setPixel(i, j, color.rgb());
//
```

Example (cont.)

Example (worker-thread.cpp, cont.)

```
if (isInterruptionRequested())
{
  m_ptrResult.reset();
  emit canceled();
  return;
}
//
```

Example (cont.)

Example (worker-thread.cpp, cont.)

```
++ nCurrent;
nPercent = (100 * nCurrent) / cnTotal;
if (nPercent > nPrev)
{
  nPrev = nPercent;
  QMetaObject::invokeMethod(
    m_pWindow,
    "updateProgress",    // Q_INVOKABLE void myMethod();
    Qt::QueuedConnection,
    // Q_RETURN_ARG(QString, retVal)
    Q_ARG(int, nPercent));
```

Example (end)

Example (worker-thread.cpp, end)

```
                //
                // emit onPercentChanged(nPercent)
              }
            }    // for (int j = 0; ...)
        }    // if (m_ptrInput)
        //
        break;
    }    // switch (m_nOperation)
}    // WorkerThread::run()
```

Example

Example

```
QFuture <void> future1 = QtConcurrent::run(f, arg1);
future1.waitForFinished();

Data data;
QFuture <QString> future2 = QtConcurrent::run(
  &data, &Data::method, arg1);
QString result = future2.result();
```

Example

Example

```
MainWindow::MainWindow(QWidget *pParent)
{
  // ...
  // QFutureWatcher <int> m_Watcher
  connect(
    &m_Watcher, SIGNAL(finished()), this, SLOT(onFinished()));
  connect(
    &m_Watcher, SIGNAL(started()), this, SLOT(onStarted()));
  // QFuture <int> m_Future;
  m_Future = QtConcurrent::run(f, arg1, arg2);
  m_Watcher.setFuture(m_Future);
}
```