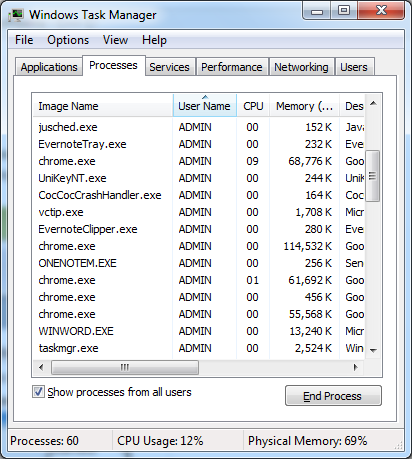
**Lập trình Qt tìm hiểu Thread**

Một process có thể có một hoặc nhiều Thread. Mỗi Thread sử dụng một dung lượng bộ nhớ khác nhau.



**Bài 1. Lập trình Qt chạy một thread**

Bước 1: Tạo project – và tạo MyThread Class

New Project > Application > Qt Console Application > nhập tên project.

Add New... > C++ > C++ Class > nhập tên class “MyThread”

Bước 2: Code

mythread.h

|  |
| --- |
| #include <QtCore>  class MyThread : public QThread  {  public:  MyThread();  void *run*();  }; |

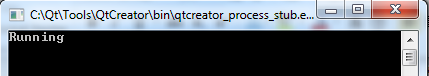
Mythread.cpp

|  |
| --- |
| #include <QtCore>  #include <QDebug>  MyThread::MyThread()  {  }  void MyThread::*run*() {  qDebug() << "Running";  } |

Main.cpp

|  |
| --- |
| #include <mythread.h>  int main(int argc, char \*argv[])  {  QCoreApplication a(argc, argv);  MyThread mThread;  mThread.start();  return a.exec();  } |

Kết quả chạy, mThread được khởi chạy và xuất ra màn hình console



**Bài 2. Lập trình Qt tạo 3 Thread chạy song song**

Mythread.h – thêm vào một trường public QString name

|  |
| --- |
| class MyThread : public QThread  {  public:  MyThread();  void *run*();  QString name;  }; |

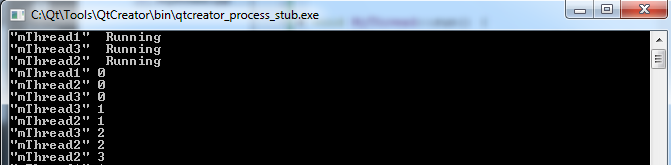
Main.cpp – Khởi chạy 3 thread

|  |
| --- |
| MyThread mThread1;  mThread1.name = "mThread1";  MyThread mThread2;  mThread2.name = "mThread2";  MyThread mThread3;  mThread3.name = "mThread3";  mThread1.start();  mThread2.start();  mThread3.start(); |

Mythread.cpp – sửa lại hàm run cho in ra tên thread và mỗi thread lần lượt in ra số nguyên từ 0->9

|  |
| --- |
| void MyThread::*run*() {  qDebug() << this->name << " Running";  for (int i = 0; i < 10; i++) {  qDebug() << this->name << i;  }  } |

Kết quả, các thread được sắp xếp chạy luân phiên.



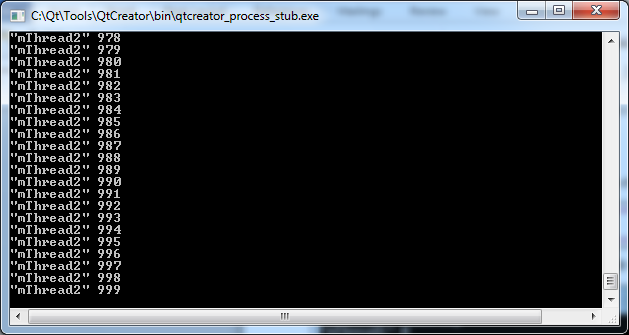
**Bài 3. Thread Priority**

Mỗi thread có một độ ưu tiên (priority) khác nhau.

Main.cpp – Thiết lập độ ưu tiên cho thread lúc khởi chạy

|  |
| --- |
| mThread1.start(QThread::HighestPriority);  mThread2.start();  mThread3.start(QThread::HighestPriority); |

Kết quả, mThread2 là thread có độ ưu tiên thấp nhất trong 3 thread. Nó kết thúc cuối cùng.



**Bài 4. QMutex**

Mythread.h – Thêm biến public bool Stop

|  |
| --- |
| public:  MyThread();  void *run*();  QString name;  bool Stop;  }; |

Main.cpp - Khởi tạo giá trị biến stop bằng false

|  |
| --- |
| int main(int argc, char \*argv[])  {  QCoreApplication a(argc, argv);  MyThread mThread1;  mThread1.name = "mThread1";  mThread1.Stop = false;  MyThread mThread2;  mThread2.name = "mThread2";  mThread2.Stop = false;  MyThread mThread3;  mThread3.name = "mThread3";  mThread3.Stop = false;  ... |

Mythread.cpp – Thêm vào Mutex trước và sau hàm kiểm tra giá trị biến Stop

|  |
| --- |
| void MyThread::*run*() {  qDebug() << this->name << " Running";  for (int i = 0; i < 10; i++) {  QMutex mutex;  mutex.lock();  if (this->Stop) {  break;  }  mutex.unlock();  this->sleep(1);  qDebug() << this->name << i;  }  } |

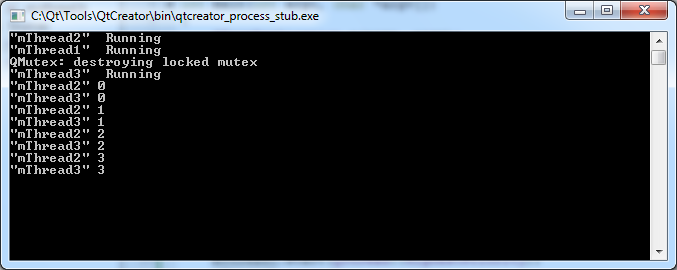
Có thể thấy nếu trong quá trình kiểm tra biến this->Stop, có khả năng biến này bị một thread khác thay đổi giá trị, làm sai lệch kết quả chạy.

Sử dụng Mutex.lock()để khóa giá trị biến Stop. Sau khi kiểm tra xong thì gọi hàm Mutex.unlock() để mở khóa.

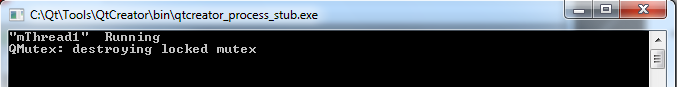
Main.cpp – Khởi chạy mThread1, sau đó thiết lập giá trị Stop của nó bằng true.

|  |
| --- |
| mThread1.Stop = true; |

Kết quả, mThread1 kết thúc ngay sau khi bắt đầu chạy.



Chú ý: nếu không khởi tạo lại giá trị mThread1.Stop = false, lỗi sau đây sẽ xuất hiện:

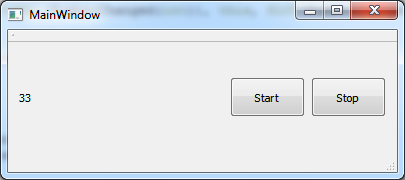


**Bài 5. Lập trình GUI với QThread**

Tạo giao diện GUI ở mainWindow:

+ Ấn nút Start sẽ khởi chạy thread, đếm tăng chu kì 100ms 1 đơn vị.

+ Ấn nút Stop dừng chạy thread.



Bước 1. Tạo project – và tạo MyThread Class

New Project > Application > Qt Widgets Application > nhập tên project.

Add New... > C++ > C++ Class > nhập tên class “MyThread” > Inherit QThread

Bước 2: Code

Mythread.h – Khai báo các biến và hàm, chú ý hàm MyThread()

|  |
| --- |
| #include <QThread>  class MyThread : public QThread  {  Q\_OBJECT  public:  explicit MyThread(QObject \*parent = 0);  void *run*();  bool Stop;  signals:  void NumberChanged(int);  ... |

Mythread.cpp – định nghĩa hàm run().

Với mỗi giá trị i, signal NumberChanged(i) được emit.

|  |
| --- |
| #include <QMutex>  MyThread::MyThread(QObject \*parent) :  QThread(parent)  {  Stop = false;  }  void MyThread::*run*() {  for (int i = 0; i < 1000; i++) {  QMutex mutex;  mutex.lock();  if (this->Stop) {  mutex.unlock();  break;  }  mutex.unlock();  emit NumberChanged(i);  this->msleep(100);  }  } |

Mainwindow.h – Khai báo con trỏ \*mThread và slots NumberChangedHandler để xử lý sự kiện khi có signal NumberChanged từ Thread.

|  |
| --- |
| class MainWindow : public QMainWindow  {  Q\_OBJECT  public:  MyThread \*mThread;  public slots:  void NumberChangedHandler(int);  ... |

Mainwindow.cpp

Kết nối sự kiện signal và slot

|  |
| --- |
| MainWindow::MainWindow(QWidget \*parent) :  QMainWindow(parent),  ui(new Ui::MainWindow)  {  ui->setupUi(this);  ui->pushButton\_start->setFixedHeight(40);  ui->pushButton\_stop->setFixedHeight(40);  mThread = new MyThread(this);  connect(mThread, SIGNAL(NumberChanged(int)), this, SLOT(NumberChangedHandler(int)));  }  void MainWindow::NumberChangedHandler(int Number) {  ui->label\_number->setText(QString::number(Number));  }  void MainWindow::on\_pushButton\_start\_clicked()  {  mThread->Stop = false;  mThread->start();  }  void MainWindow::on\_pushButton\_stop\_clicked()  {  mThread->Stop = true;  } |

Chú ý:

1. Nếu hàm MyThread viết không có QObject, quá trình biên dịch sẽ báo lỗi ở câu lệnh tạo mới mThread = new MyThread(this), kiểu như:

C2664 'webapi::fUser' : cannot convert paramter 1 from 'User \*const' to 'User'

=> Cần viết thêm QObject.

2. Phải có khởi tạo lại giá trị mThread->Stop = false trong hàm on\_pushButton\_start\_clicked(). Nếu ko Stop sẽ luôn bằng true sau lần chạy-dừng đầu tiên.

**Bài 6. How to do QThread correctly in Qt**

http://stackoverflow.com/questions/13878745/correct-way-of-threading-in-qt