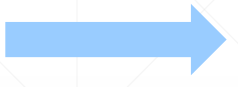
嵌入式工業機器視覺

- 課程宗旨：
  - 機器視覺應用原理介紹
  - 嵌入式系統架構
  - 程式編譯核心差異
  - 程式API開發與包裝
  - 工業相機取像API
  - 實作嵌入式視覺系統

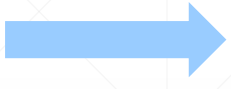
課堂練習

檔案路徑

Raw data



QImage



QPixmap
QGraphicsScene
QGraphicsView

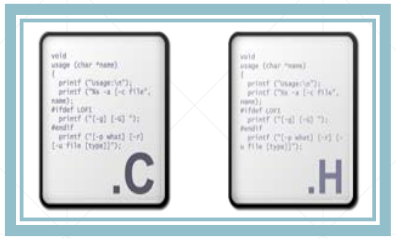
QImage
- bits()
- Scanline()

軟體架構參考

檔案路徑

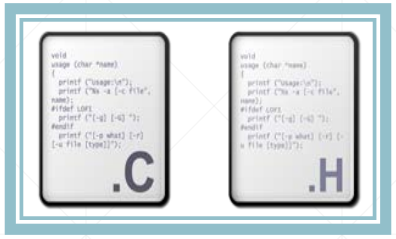
Raw data

MyBMPClass



\+

MyImgProcessing



QImage

UI

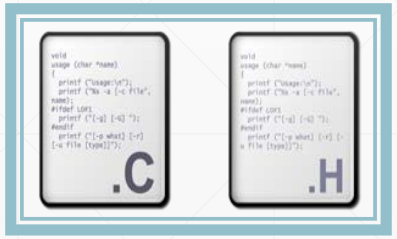
QPixmap
QGraphicsScene
QGraphicsView

Shared library is better.

檔案路徑

Raw data

MyBMPClass



QImage

or

CvImage

ImageProcessing

or

OpenCV

QImage

UI

QPixmap
QGraphicsScene
QGraphicsView

Shared library is better.

嵌入式工業機器視覺

- 即時多工系統
- 工業相機
  - 工業相機取像API
  - 基本操作

嵌入式工業機器視覺

- 多工 ➔ 主要的動機是希望能避免CPU 閒置，使其效能最大化。
- 分時(timesharing)
  - 把CPU 的使用時間進行分割，使用者輪流使用CPU。
  - 這樣每個使用者都會得到一些回應，感覺上就沒有等待那麼久。

- 即時系統
  - Cooling[1]：Real-time systems are those which must produce correct responses within a definite time limit. Should computer responses exceed these bounds then performance degradation and or malfunction results.
  - Levi, Agrawala[2]: A real-time system is a set of concurrently executed computations which interact with each other and adhere to some timing constraints.

1. Cooling, J.E., Software Design for Real-Time Systems, chapman & Hall, London (1991).
2. Shem-Tov Levi and Ashok K. Agrawala, Real-Time System Design, McGraw-Hill, New York (1990)

嵌入式工業機器視覺

- 即時系統
  - **硬即時（hard real-time）：** 機械手臂的控制系統、液面控制的閥門關閉動作
    工作必須在固定的時間內完成且絕無例外。
  - **軟即時（soft real-time）：** 自動提款機、電腦操作系統
    工作是無法在要求的時間內完成亦不會造成系統的傷害。

- 即時控制系統與外部同步動作：
  - **時間驅動工作（clock-driven task）**：週期性（periodic）資料採集
  - **事件驅動工作（event-driven task）**：緊急事件➔以最壞打算（the worst case）考量，
    盡量使延遲反應的時間縮短
  - **互動式工作（interactive task）**：操作人員界面（operator interface）

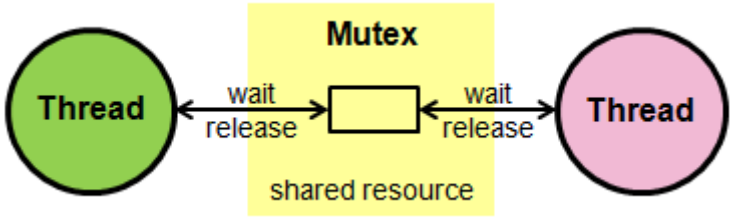
嵌入式工業機器視覺

- 軟體程式
  - **順序式程式（sequential program）：**
    - 容易撰寫、資源保護容易、軟體維護與擴充較無彈性。
  - **前景 / 背景式程式（foreground / background program）：**
    - 透過中斷處理程序進行前背景切換，背景難以作時間管理，軟體維護與擴充相較侷限，資源保護須留意。
  - **多工程式（multi-tasking program）：**
    - 將多個子工作分配到多CPU中，使其效能最高化，但子工作完全獨立是幾乎是不可能。通常的情況是各 CPU 之間仍需溝通資訊，或共用系統資源（如記憶體、I/O等）。
    - 多工設計
      1. 設計者完全以工作的性質來分類程式，不在遵循程式執行順序的限制。
      2. 各工作佔用系統資源的情況可作嚴密的管理與時間的分配。

嵌入式工業機器視覺

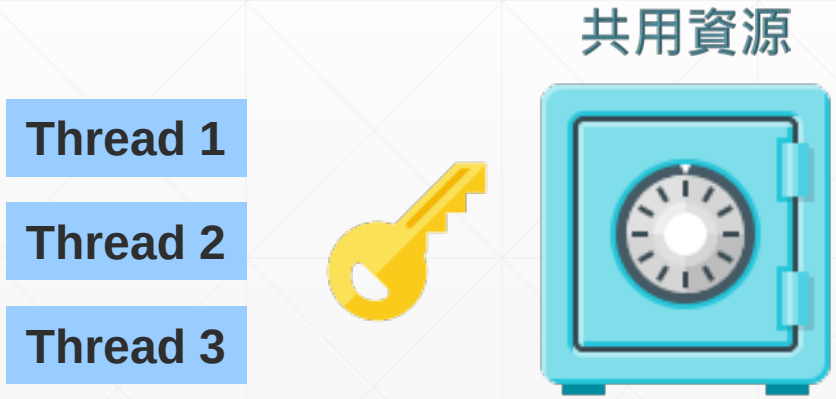
- 工作 (task) 與執行緒 (thread)
- 工作切換或內文切換 (Context Switching)
- 排程器 (Scheduler)
- 強取式(preemptive)與非強取式(non-preemptive)核心
- 程式之再進入性 (reentrancy)
  - 互斥 (mutual exclusion)
  - 號誌 (semaphore)
  - 死結 (Dead lock)
  - 同步(synchronization)

嵌入式工業機器視覺



- 互斥 (mutual exclusion)
- Mutex is a key to a toilet. One person can have the key - occupy the toilet -at the time. When finished, the person gives (frees) the key to the next person in the queue.
- A mutex object only allows one thread into a controlled section , forcing other threads which attempt to gain access to that section to wait until the first thread has exited from that section.

```
/* 任務1 */
   mutexWait (mutex_mens_room) ;
      //安全使用共享資源
   mutexRelease (mutex_mens_room) ;

/* 任務2  */
   mutexWait (mutex_mens_room) ;
      //安全使用共享資源
   mutexRelease (mutex_mens_room) ;
```



嵌入式工業機器視覺

- 號誌(semaphore)

Semaphore is the number of free identical toilet keys. Example, say we have four toilets with identical locks and keys. The semaphore count - the count of keys - is set to 4 at beginning (all four toilets are free), then the count value is decremented as people are coming in. If all toilets are full, ie. there are no free keys left, the semaphore count is 0. Now, when eq. one person leaves the toilet, semaphore is increased to 1 (one free key), and given to the next person in the queue.

- 控制共享資源的使用權(滿足互斥條件)
- 標誌某事件的產生
- 使兩個工作的行為同步

  Binary semaphore ≒ Mutex

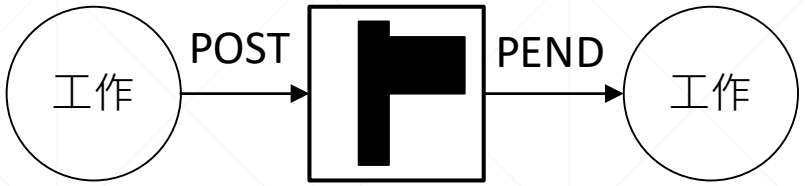
  Counting semaphore

```
/* Task 1 - Producer */
    semPost(sem_power_btn);    // Send the signal

/* Task 2 - Consumer */
    semPend(sem_power_btn);   // Wait for signal
```

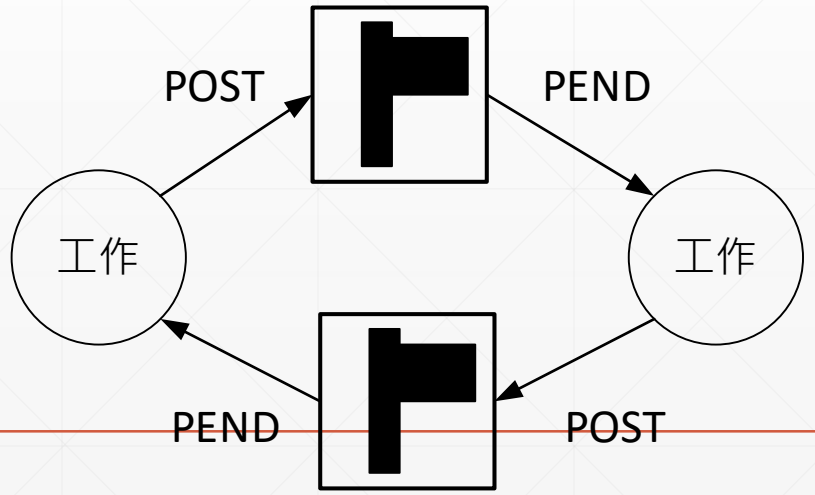
嵌入式工業機器視覺

- 同步(Synchronization)
  - 事件旗標 (Event flag)
  - 號誌：

單向同步



雙向同步




```
#Task 1
Task1()
{
     for(;;)
     {
          Perform operation;
          signal task #2; (Post)
          Wait for signal from task #2; (Pend)
          Continue operation;

     }
}

#Task 2
Task2()
{
     for(;;)
     {
          Perform operation;
          signal task #1; (Post)
          Wait for signal from task #1; (Pend)
          Continue operation;
     }
}
```

雙向同步

嵌入式工業機器視覺

- 嵌入式作業系統專家 Michael Barr[1]：

The correct use of a semaphore is for signaling from one task to another. A mutex is meant to be taken and released, always in that order, by each task that uses the shared resource it protects. By contrast, tasks that use semaphores either signal or wait—not both.

需要進入臨界區間(Critical section)時 ➔ Mutex ➔ 上鎖/解鎖永遠是同一個執行緒

需要處理訊號時， ➔ Semaphore ➔ 等待/通知通常是不同的執行緒

使用共用資源時用 Mutex

要互相通知時用 Semaphore

https://barrgroup.com/Embedded-Systems/How-To/RTOS-Mutex-Semaphore

嵌入式工業機器視覺

- **死結 (Dead lock)**

  可能發生狀況：

  (1) 具有多個共用資源

  (2) 執行緒鎖定一個共用資源時，還沒解除鎖定就想去鎖定另外一個共用資源。

  (3) 取得共用資源的順序不固定。

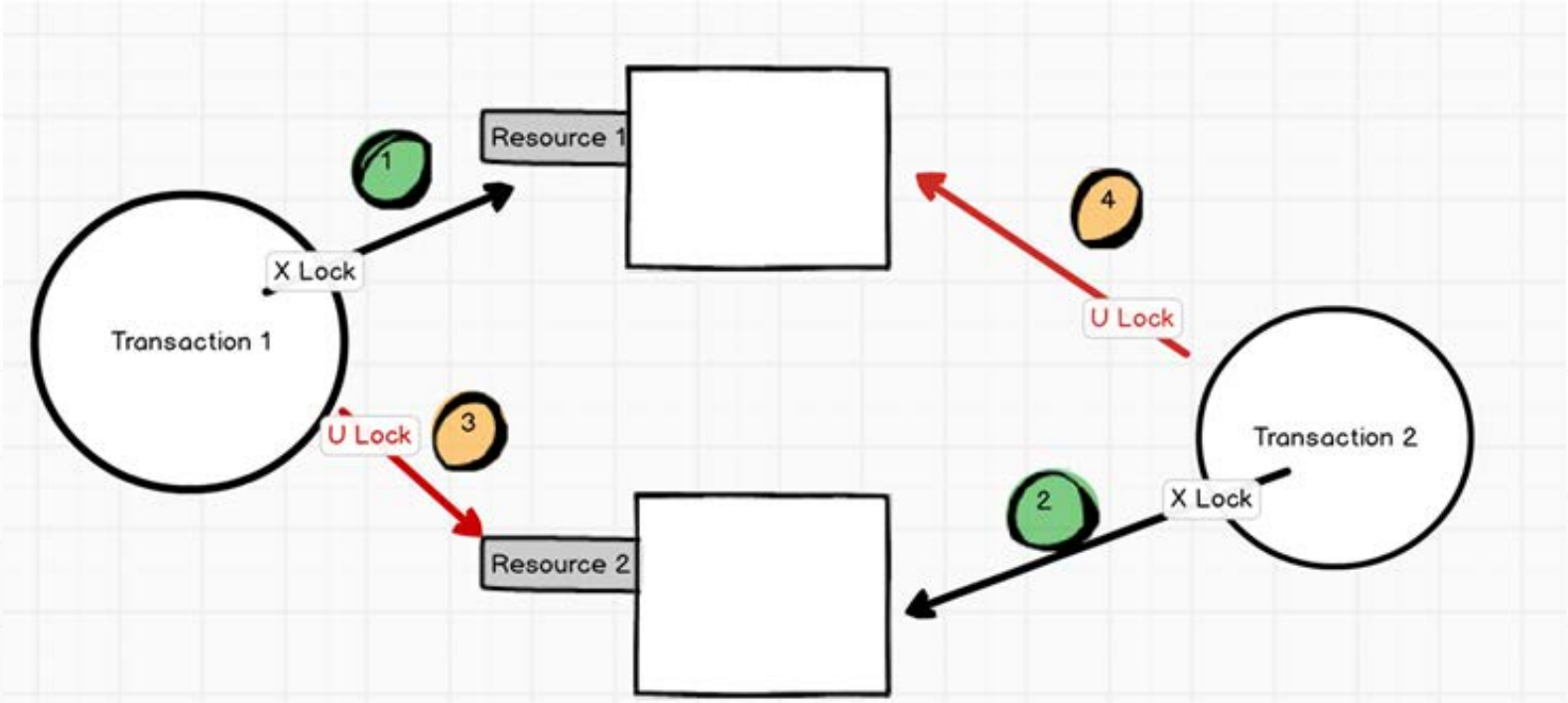
案例：

假設有兩個人(Ivy and Hank)在用餐，餐具只有兩個，分別是spoon以及fork， Ivy的習慣是左手拿spoon，接著右手拿fork才會開始用餐; Hank的習慣是左手拿fork ，接著右手拿spoon之後才開始用餐。

一個情況是： Ivy左手拿了spoon，等待fork，但Hank左手已經拿了fork，等待spoon.....

嵌入式工業機器視覺

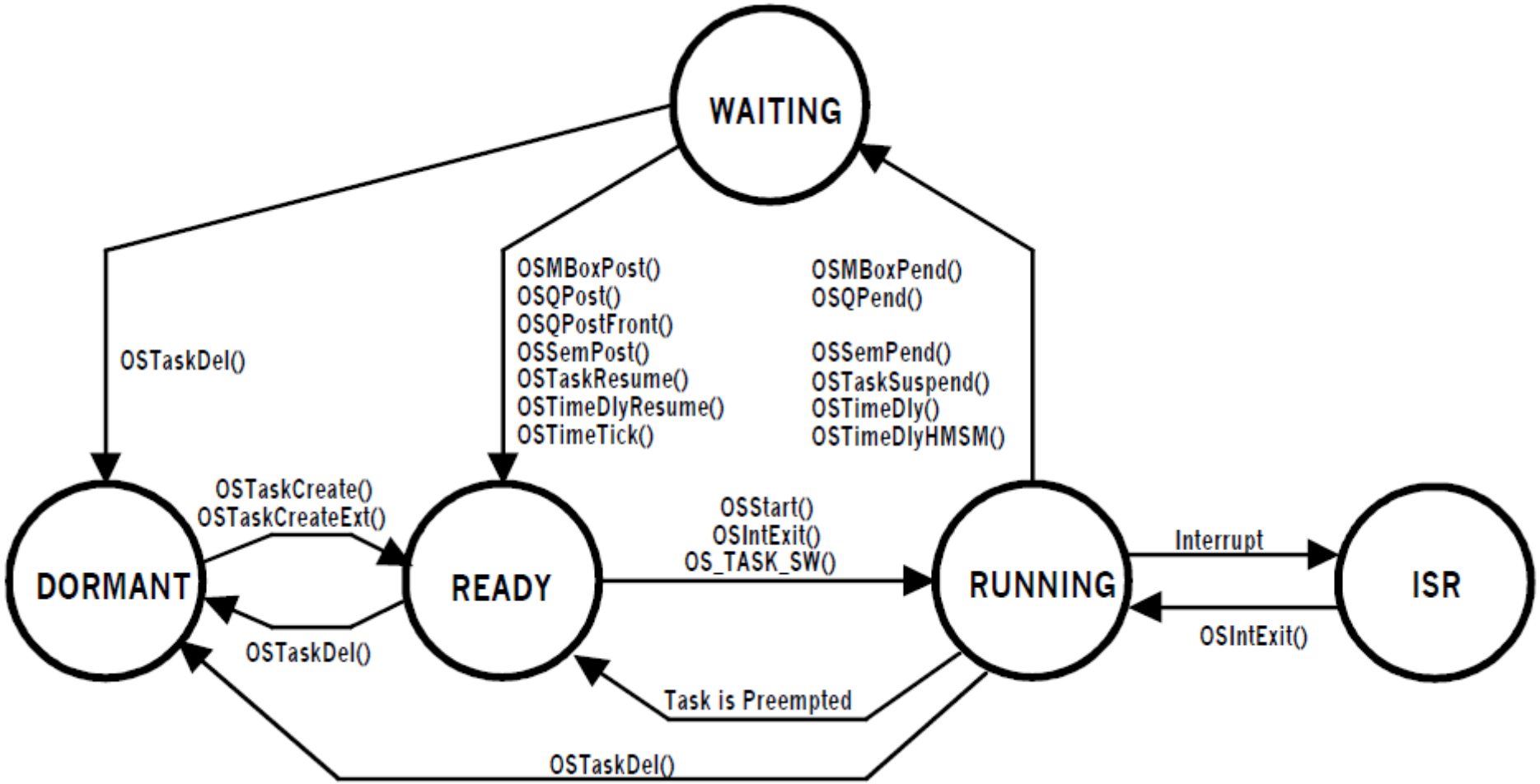
- **避免 死結(Dead lock)**
  - 得到全部需要的資源後，才開始做下一步動作。
  - 用相同的順序申請多個資源。
  - 用相反的順序釋放資源。
  - 呼叫請求訊號時，帶時間參數。



Dead lock 死結示意

嵌入式工業機器視覺

- 每個典型的工作都是一個無窮迴圈且處在以下五種狀態：
  - 休眠(Dormant)
  - 就緒(Ready)
  - 執行(Running)
  - 暫時停止(Waiting)



嵌入式工業機器視覺

- QThread
  - The QThread class provides a platform-independent way to manage threads.
  - A QThread object manages one thread of control within the program.
  - QThread begin executing in run(). By default, run() starts the event loop by calling exec() and runs a Qt event loop inside the thread.

繼承QThread來實現多執行緒有個新手陷阱 ➔

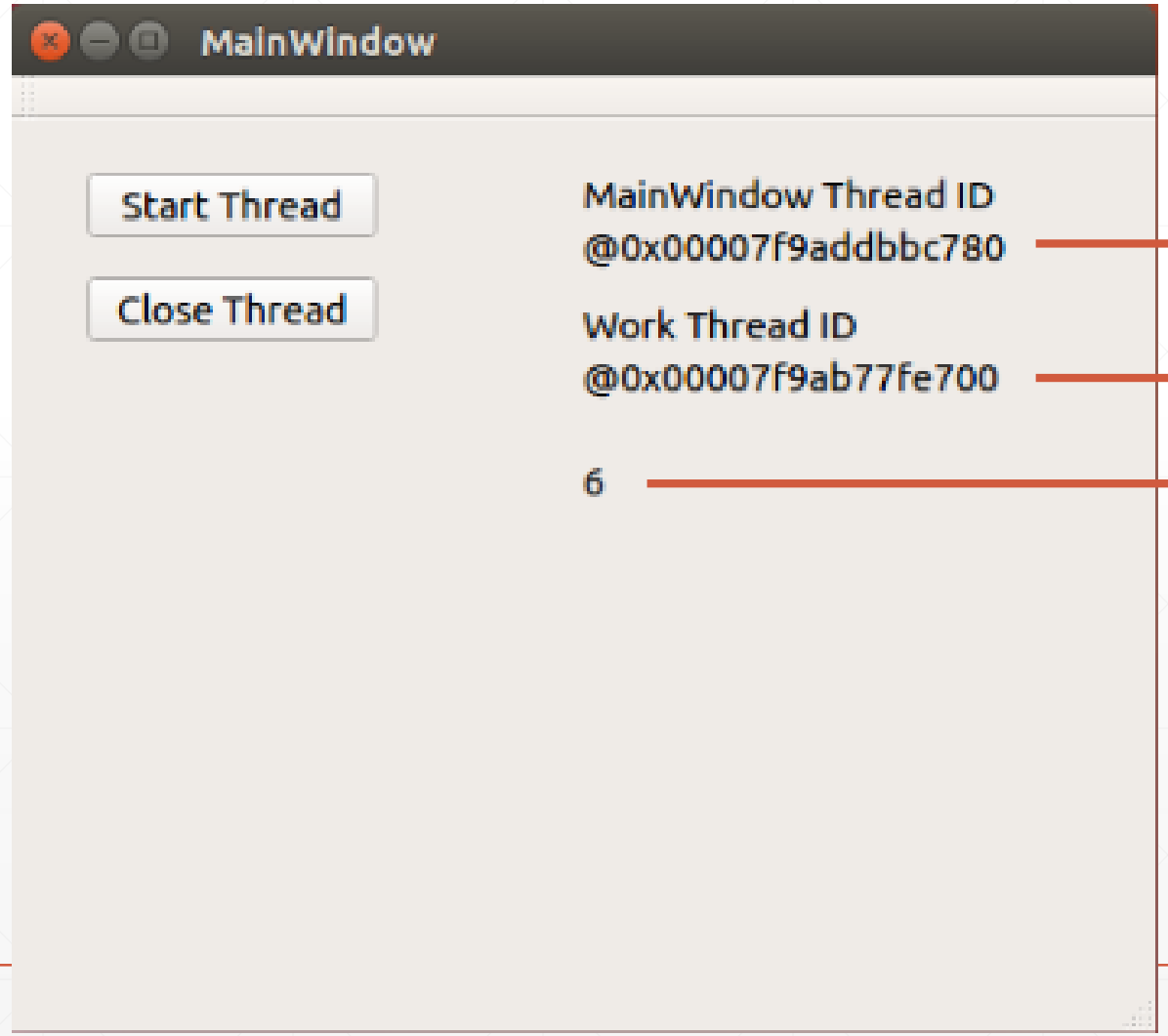
QThread本身並不存在run()函式所在的執行緒上，而是存在原屬執行緒中。

繼承QObject進行多執行緒開發 ➔ 多工執行內容

使用QThread進行多執行緒開發 ➔ 管理執行緒

嵌入式工業機器視覺

- 執行緒練習範例：



主執行緒ID位置
label_main_ID

子執行緒ID位置
label_work_ID

子執行緒：累加數值
label

嵌入式工業機器視覺

- 新建執行緒類別，繼承QObject ➔ 工作

workthread.h

```
#ifndef WORKTHREAD_H
#define WORKTHREAD_H

#include <QObject>

class WorkThread : public QObject
{
    Q_OBJECT
public:
    explicit WorkThread(QObject *parent = nullptr);
    void closeThread();

public slots:
    void slot_startThread();

private:
    volatile bool isStop;
};

#endif // WORKTHREAD_H
```

workthread.cpp

```
#include "workthread.h"
#include <QThread>

WorkThread::WorkThread(QObject *parent)
                        : QObject(parent)
{
    isStop = false;
}

void WorkThread::closeThread()
{
    isStop = true;
}

void WorkThread::slot_startThread()
{
    int i = 0;
    while (1)
    {
        if(isStop)
            return;

        i++;

        QThread::sleep(1);
        //QThread::usleep(1000);
    }
}
```

將執行緒執行內容宣告為槽函式

嵌入式工業機器視覺

- 新建執行緒類別
- 使用QThread類別 ➔ 管理

mainwindow.h

```
#include "workthread.h"
#include <QThread>
```

```
public:
    explicit MainWindow(QWidget *parent
    ~MainWindow();

    QThread *controlThread;
    WorkThread *myWorkThread;

private slots:
    void on_pushButton_clicked();   //Start Thread
    void on_pushButton_2_clicked(); //Stop Thread
```

mainwindow.cpp

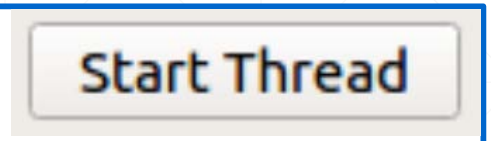
```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    controlThread = new QThread;  //contorll thread
    myWorkThread = new WorkThread; //work thread

    // move object based thread to controll thread
    myWorkThread->moveToThread(controlThread);
    //delete
    connect(controlThread,SIGNAL(finished()),
            controlThread,SLOT(deleteLater()));
    //start
    connect(controlThread,SIGNAL(started()),
            myWorkThread,SLOT(slot_startThread()));
}
```

嵌入式工業機器視覺

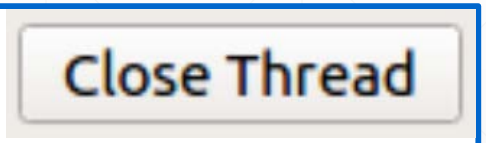
- 新建執行緒類別
- 使用QThread類別 ➔ 管理

mainwindow.cpp

```
void MainWindow::on_pushButton_clicked()
{
    controlThread->start();  //open thread slot
    QString threadText = QStringLiteral("@0x%1").arg(quintptr(QThread::currentThreadId()),
                                                     16, 16, QLatin1Char('0'));
    ui->label_main_ID->setText(threadText);
}
```




```
void MainWindow::on_pushButton_2_clicked()
{
    if(controlThread->isRunning())
    {
        myWorkThread->closeThread();  //close thread slot
        controlThread->quit();        //break from event loop
        controlThread->wait();        //release resource slot
    }
}
```



嵌入式工業機器視覺

- 新增介面互動接口 Signal & Slot

workthread

UI

workthread.h

```
signals:
    void signal_updateIndex(int index);
    void signal_updateThreadID(void *id);
```

workthread.cpp

```
void WorkThread::slot_startThread()
{
    int i = 0;
    while (1)
    {
        if(isStop)
            return;

        i++;
        emit signal_updateIndex(i);
        emit signal_updateThreadID(QThread::currentThreadId());

        QThread::sleep(1);
        //QThread::usleep(1000);
    }
}
```

嵌入式工業機器視覺

- 新增介面互動接口 Signal & Slot

workthread

UI

mainwindow.cpp

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    controlThread = new QThread;  //contorll thread
    myWorkThread = new WorkThread; //work thread

    // move object based thread to controll thread
    myWorkThread->moveToThread(controlThread);
    //delete
    connect(controlThread,SIGNAL(finished()),
            controlThread,SLOT(deleteLater()));
    //start
    connect(controlThread,SIGNAL(started()),
            myWorkThread,SLOT(slot_startThread()));

    //update
    connect(myWorkThread,SIGNAL(signal_updateIndex(int)),
            this,SLOT(slot_handleupdate(int)));
    connect(myWorkThread,SIGNAL(signal_updateThreadID(void*)),
            this,SLOT(slot_handleID(void*)));

}
```

嵌入式工業機器視覺

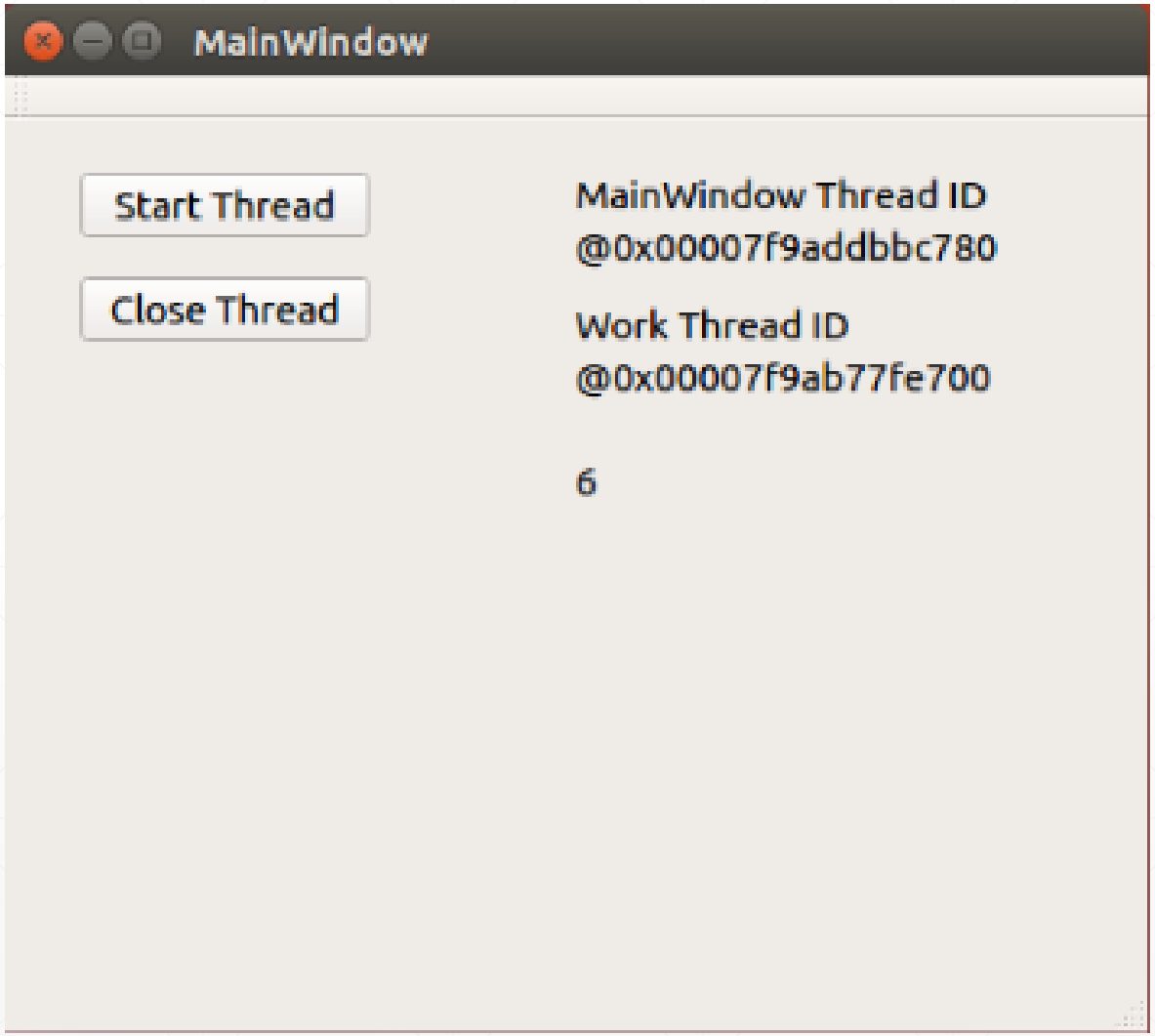
- 新增介面互動接口 Signal & Slot

workthread

UI

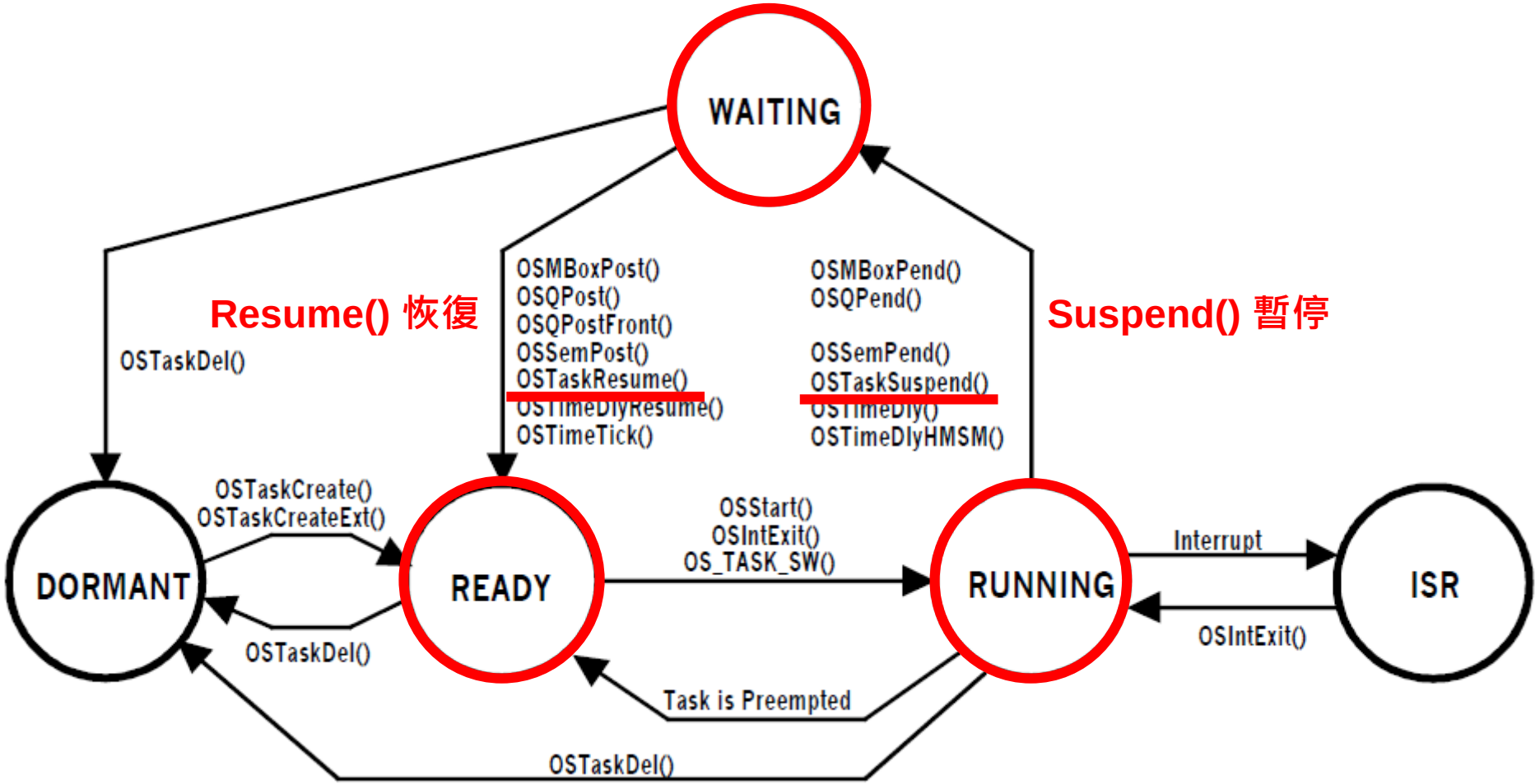
mainwindow.cpp

```
void MainWindow::slot_handleupdate(int index)
{
    ui->label->setText(QString::number(index));
}

void MainWindow::slot_handleID(void *id)
{
    QString threadText = QStringLiteral("@0x%1").arg(quintptr(id),
                                                     16, 16, QLatin1Char('0'));
    ui->label_work_ID->setText(threadText);
}
```


MainWindow
Start Thread
Close Thread
MainWindow Thread ID
@0x00007f9addbbc780
Work Thread ID
@0x00007f9ab77fe700
6

嵌入式工業機器視覺

- 每個典型的工作都是一個無窮迴圈且處在以下五種狀態：
  - 休眠(Dormant)
  - 就緒(Ready)
  - **執行(Running)**
  - **暫時停止(Waiting)**



- The QWaitCondition class provides a condition variable for synchronizing threads.

```
bool QWaitCondition::wait(QMutex *lockedMutex, unsigned long time = ULONG_MAX)
```

Releases the *lockedMutex* and waits on the wait condition.

The *lockedMutex* must be initially locked by the calling thread.

- Use wakeOne() to wake one randomly selected thread or wakeAll() to wake them all.

嵌入式工業機器視覺

workthread.h

```
private:
    volatile bool isStop;

    volatile bool isPause;
    QMutex mutex;
    QWaitCondition monitor;
    bool WaitOne(unsigned long time);
```

workthread.cpp

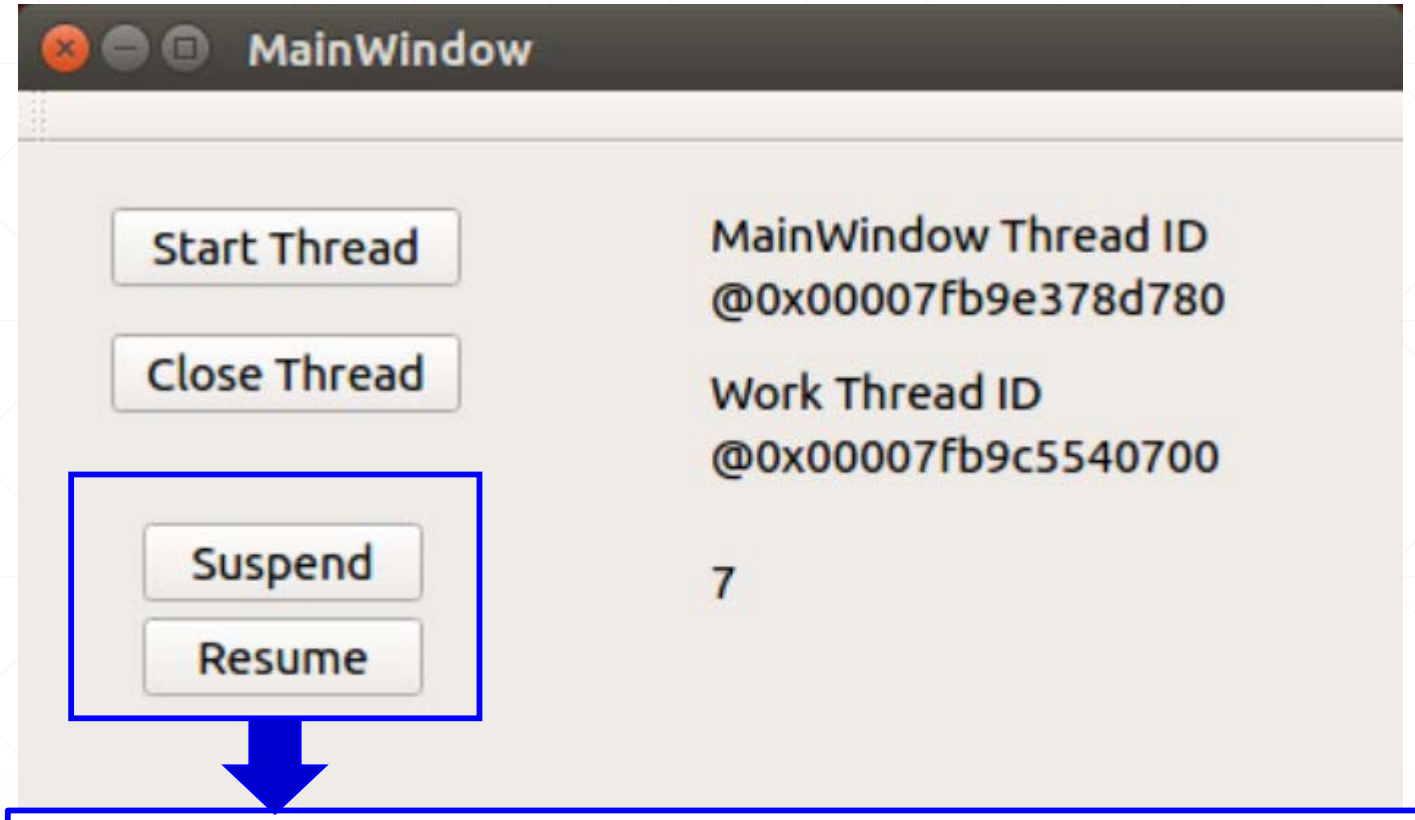
```
WorkThread::WorkThread(QObject *parent)
                    : QObject(parent)
{
    isStop = false;
    isPause = false;
}

void WorkThread::closeThread()
{
    resume();
    isStop = true;
}
```

workthread.cpp

```
void WorkThread::suspend()
{
    QMutexLocker locker(&mutex);
    isPause = true;
}

void WorkThread::resume()
{
    QMutexLocker locker(&mutex);
    isPause = false;

    monitor.wakeOne();
}

bool WorkThread::WaitOne(unsigned long time)
{
    QMutexLocker locker(&mutex);
    if(isPause) monitor.wait(&mutex, time);

    return isPause;
}
```

workthread.cpp

```
void WorkThread::slot_startThread()
{
    int i = 0;
    while (1)
    {
        if(isStop)
            return;

        WaitOne(INFINITY);

        i++;
        emit signal_updateIndex(i);
        emit signal_updateThreadID(QThread::currentThreadId());

        QThread::sleep(1);
        //QThread::usleep(1000);
    }
}
```




```
void MainWindow::on_pushButton_3_clicked()
{
    myWorkThread->suspend();
}

void MainWindow::on_pushButton_4_clicked()
{
    myWorkThread->resume();
}
```

嵌入式工業機器視覺

- 工業相機：
  - 性能強，穩定可靠，連續工作時間長。
  - 快門時間短，可全局曝光。
  - 幀率遠高於普通相機。
  - 直接輸出Raw Data，其光譜範圍相對普通相機寬。

- 工業相機的應用：
  - 基礎應用：開始、暫停、停止。
  - 相機曝光時間、幾何轉換...
  - 取像模式：連續、軟體觸發、外部觸發。
  - 閃光燈：同步、延遲。

嵌入式工業機器視覺

- 使用工業相機工作流程：

1）列舉裝置

2）建立控制代碼

3）開啟裝置

→ **相機初始化**

4）開始抓圖

5）獲取影像 ➔ 進行機器視覺演算法

→ **相機影像擷取執行緒Thread**
或
相機回呼函式 Call back

6）停止抓圖

7）關閉裝置

8）銷燬控制代碼

→ **相機反初始化**

嵌入式工業機器視覺

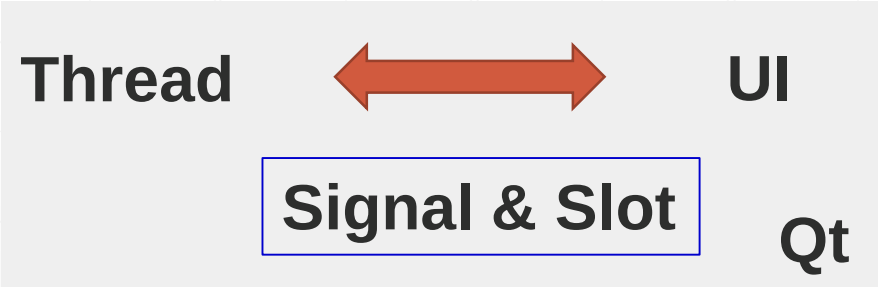

MainWindow
Initial Camera
Start Capture
Pause Capture
CameraPlay

嵌入式工業機器視覺

- 流程：
  - 新建專案 ➔ 載入相機API定義檔 ➔ 載入相機函式庫

初始化相機 ➔ 開始取像 ➔ 擷取影像資料 ➔ 顯示於介面

配置所需記憶體



停止取像 ➔ 反初始化相機

關閉程式前，釋放所有記憶體

嵌入式工業機器視覺

- 相機初始化

  1）列舉裝置

  2）建立控制代碼

  3）開啟裝置

```
typedef struct _cam_param
{
    int                     hCamera = -1;
    unsigned char           *pRawBuffer=NULL;
    unsigned char           *pRgbBuffer=NULL;

    tSdkFrameHead           tFrameHead;
    tSdkCameraCapbility     tCapability;

    int                     wid;
    int                     hei;
    long                    bufferSize;
    unsigned char           *rawBuffer=NULL;
}cam_param;
```

```
cam_param g_cam;
```

全域變數

```
int MainWindow::Init_Cam()
{
    int                     iCameraCounts = 1;
    int                     iStatus=-1;
    tSdkCameraDevInfo       tCameraEnumList[1];

    //sdk初始化  0 English 1中文
    CameraSdkInit(1);

    //列舉設備並建立清單
    CameraEnumerateDevice(tCameraEnumList,&iCameraCounts);

    //沒有找到設備
    if(iCameraCounts==0){
        return -1;
    }

    //相機初始化,初始化後才能使用相機操作函式
    iStatus = CameraInit(&tCameraEnumList[0],-1,-1,&g_cam.hCamera);

    //如果初始化失敗
    if(iStatus!=CAMERA_STATUS_SUCCESS){
        return -1;
    }

    //獲得相機特性參數結構
    CameraGetCapability(g_cam.hCamera,&g_cam.tCapability);
```

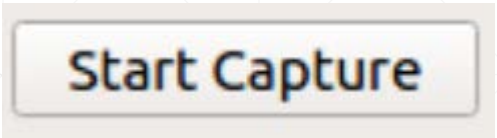
```
    //相機初始化,初始化後才能使用相機操作函式
    iStatus = CameraInit(&tCameraEnumList[0],-1,-1,&g_cam.hCamera);

    //如果初始化失敗
    if(iStatus!=CAMERA_STATUS_SUCCESS){
        return -1;
    }

    //獲得相機特性參數結構
    CameraGetCapability(g_cam.hCamera,&g_cam.tCapability);

    g_cam.pRgbBuffer = (unsigned char*)malloc(g_cam.tCapability.sResolutionRange.iHeightMax*
                                              g_cam.tCapability.sResolutionRange.iWidthMax*3);
    g_cam.rawBuffer = (unsigned char*)malloc(g_cam.tCapability.sResolutionRange.iHeightMax*
                                             g_cam.tCapability.sResolutionRange.iWidthMax*3);

    //設置圖像輸出格式
    if(g_cam.tCapability.sIspCapacity.bMonoSensor)
        CameraSetIspOutFormat(g_cam.hCamera,CAMERA_MEDIA_TYPE_MONO8);

    g_cam.wid = g_cam.tCapability.sResolutionRange.iWidthMax;
    g_cam.hei = g_cam.tCapability.sResolutionRange.iHeightMax;
    g_cam.bufferSize = sizeof(unsigned char) * g_cam.wid * g_cam.hei;

    return 0;
}
```


```
void MainWindow::on_pushButton_init_cam_clicked()
{
    if(g_cam.hCamera != -1)
        return;

    int err = Init_Cam();

    if(err == 0)
    {
       controlThread->start();  //open thread slot
        ui->graphicsView->setFixedSize(g_cam.wid + 4, g_cam.hei + 4);
    }

    if(err == 0)
        ui->label->setText("Init Cam OK");
    else
        ui->label->setText("Init Cam NG");
}
```

嵌入式工業機器視覺

- 相機反初始化

```
void MainWindow::Uninit_Cam()
{
    if(g_cam.rawBuffer!=NULL){
        free(g_cam.rawBuffer);
        g_cam.rawBuffer=NULL;
    }

    if(g_cam.pRgbBuffer!=NULL){
        free(g_cam.pRgbBuffer);
        g_cam.pRgbBuffer=NULL;
    }

    if(g_cam.hCamera>0){
        //相機反初始化,釋放資源
        CameraUnInit(g_cam.hCamera);
        g_cam.hCamera=-1;
    }
}
```




```
void MainWindow::on_pushButton_start_clicked()
{
    if(g_cam.hCamera == -1)
        return;

    /*讓相機進入工作模式，開始接收相機所擷取到的資料數據。
    如果當下是觸發模式則需要接收到觸發訊號後才會更新資料*/
    CameraPlay(g_cam.hCamera);

    ui->label->setText("CameraPlay");
}
```


```
void MainWindow::on_pushButton_pause_clicked()
{
    if(g_cam.hCamera == -1)
        return;

    CameraPause(g_cam.hCamera);

    ui->label->setText("CameraPause");
}
```

嵌入式工業機器視覺

- 獲取影像 ➔ 建立取像執行緒

capturethread.h

```
#ifndef CAPTURETHREAD_H
#define CAPTURETHREAD_H

#include <QObject>
#include <QMutex>
#include <QWaitCondition>
#include <QImage>

class CaptureThread : public QObject
{
    Q_OBJECT
public:
    explicit CaptureThread(QObject *parent = nullptr);

    void closeThread();

signals:
    void captured(QImage img);

public slots:
    void slot_startThread();

private:
    QVector<QRgb> grayColourTable;

private:
    volatile bool isStop;
};

#endif // CAPTURETHREAD_H
```

影像回傳訊號

嵌入式工業機器視覺

- 獲取影像 ➔ 建立取像執行緒

capturethread.cpp

```
#include "mainwindow.h"
#include "capturethread.h"
#include <QThread>
#include <QMutexLocker>
#include "CameraApi.h"

extern cam_param g_cam;

CaptureThread::CaptureThread(QObject *parent) : QObject(parent)
{
    isStop = false;

    for(int i = 0; i < 256; i++)
        grayColourTable.append(qRgb(i, i, i));

}

void CaptureThread::closeThread()
{
    isStop = true;
}
```

嵌入式工業機器視覺

- 獲取影像 ➔ 建立取像執行緒

capturethread. cpp

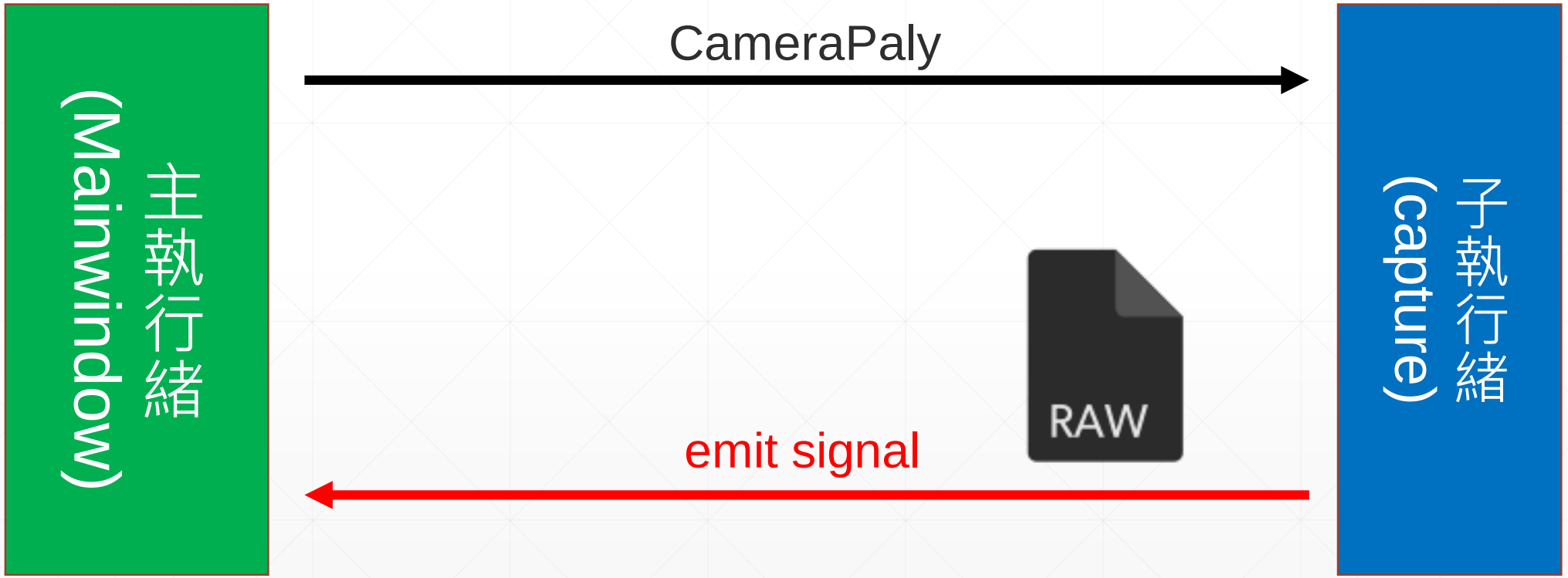
```
void CaptureThread::slot_startThread()
{
    while (1)
    {
        if(isStop)
            return;

        if (CameraGetImageBuffer(g_cam.hCamera,&g_cam.tFrameHead,&g_cam.pRawBuffer,2000)
            == CAMERA_STATUS_SUCCESS)
        {
            CameraImageProcess(g_cam.hCamera,g_cam.pRawBuffer,g_cam.pRgbBuffer,&g_cam.tFrameHead);
            if(g_cam.tFrameHead.uiMediaType==CAMERA_MEDIA_TYPE_MONO8)
            {
                memcpy(g_cam.rawBuffer,g_cam.pRgbBuffer,  g_cam.wid*g_cam.hei);
                QImage img(g_cam.rawBuffer, g_cam.wid, g_cam.hei, QImage::Format_Indexed8);
                img.setColorTable(grayColourTable);
                emit captured(img);
            }
            CameraReleaseImageBuffer(g_cam.hCamera,g_cam.pRawBuffer);
        }

        QThread::usleep(1000);
    }
}
```

g_cam.pRgbBuffer : Image raw data

建立Qimage，設定調色盤(灰階影像)

嵌入式工業機器視覺

- 顯示於介面



嵌入式工業機器視覺

- 顯示於介面

capturethread.h

```
signals:
    void captured(QImage img);
```

capturethread.cpp

```
emit captured(img);
```

capturehread

UI

mainwindow.h

```
private slots:
    void slot_handleCaputred(QImage img);
```

mainwindow.cpp

```
connect(myCaptureThread,SIGNAL(captured(QImage)),
        this, SLOT(slot_handleCaputred(QImage)));
```

```
void MainWindow::slot_handleCaputred(QImage img)
{
    QPixmap pixmap = QPixmap::fromImage(img);

    scene->clear();
    scene->setSceneRect(0,0,img.width(),img.height());
    scene->addPixmap(pixmap);
}
```

嵌入式工業機器視覺

- 注意CameraDemo 設定平台為PC 64bit
- 請自行更換適合自己的相機函示庫