Lab 4

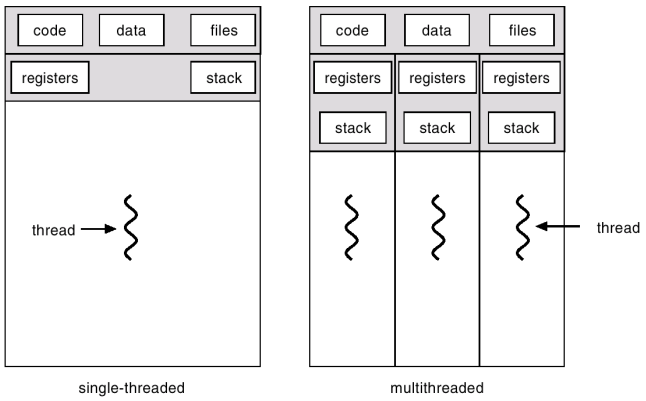
multithread

一、 實驗目的

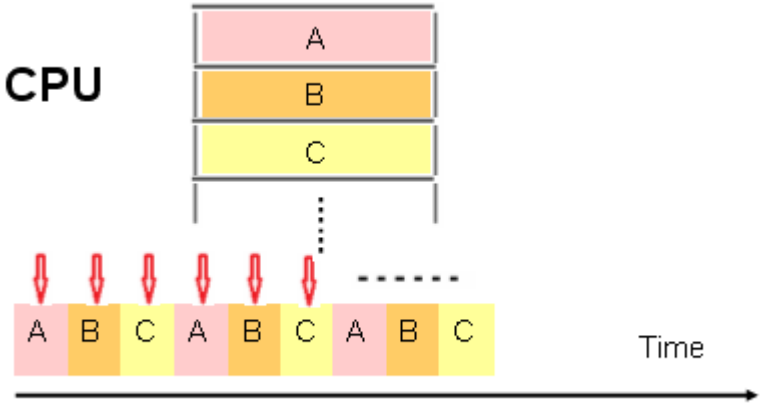
在上一次的實驗，我們成功地透過socket接收來自server的訊息，達到最簡易版瀏覽器的效果。那麼如果我們想要同時連上多個網站，並隨意執行任何一個socket的send 及 receive 函式，要如何實現呢？Visual C++ 本身提供了一個BackgroundWorker元件，這個元件本身即是一個以thread(執行緒)為基底所設計的元件，在本次的實驗中，我們實際來學習如何以BackgroundWorker來做為網路傳輸的背景設計。

二、 架構說明：

Process 中的一段程式碼執行軌跡稱為 thread，是電腦中最小的執行單位。

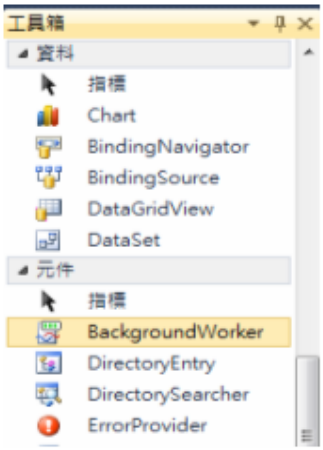


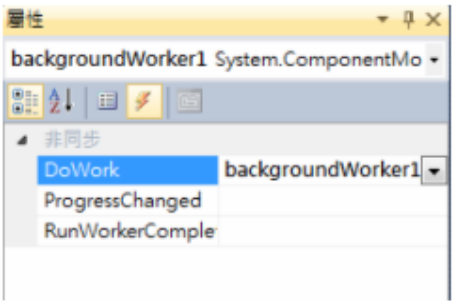
由於如果主程式內有迴圈的話，程式執行後會當住，且螢幕也不會顯示，因此，我們必須透過執行緒來解決此問題。利用time slicing 的概念，將時間軸切成好幾塊,每一小塊區間都執行不同的thread。



三、 實際操作：

1. 利用 visual studio 裡面的元件 Backgroundworker，並且把該 thread 要執行的事情寫在 DoWork 函數





2. 利用上次實驗學到的來建立連線並取得 html

```
private: System::Void backgroundWorker1_DoWork (System::Object^  sender, System::ComponentModel::DoWorkEventArgs^  e) {
    int i = j++;
    //for(int i=0;i<5;i++)
    //{
        mySocket[i] = gcnew Socket(AddressFamily::InterNetwork, SocketType::Stream, ProtocolType::Tcp);
         try
        {
            mySocket[i]->Connect("stunion.nctu.edu.tw", 80);
        }
        catch (Exception ^ee)
        {
            receivedata[i] = "主機無法連接 !! " + ee->Message;
            return;
        }
        if (mySocket[i]->Connected){
            //send command
            String ^strTest = "GET /AOOP/aoop_lab4.php HTTP/1.1\r\nHost: 140.113.150.12\r\nConnection: Close\r\n\r\n";
            array<Byte>^bytesSent = Encoding::ASCII->GetBytes( strTest );
            mySocket[i]->Send(bytesSent);
            //receive
            receivedata[i] = nullptr;
            int dataLength = 0;
            array <Byte> ^myBufferBytes=gcnew array <Byte>(256);
            do
            {
                dataLength  = mySocket[i]->Receive(myBufferBytes);
                receivedata[i] += Encoding::ASCII->GetString(myBufferBytes, 0, dataLength);
            }while( dataLength > 0 );
            //write
            file->WriteAllText("aoop_lab"+i+".html",receivedata[i]);
        }
    //}
    }
```

3. 呼叫 RunWorkerAsync()函式，觸發 DoWork 事件

```
private: System::Void button2_Click(System::Object^  sender, System::EventArgs^  e) {
             backgroundWorker1->RunWorkerAsync();
         }
```

4. A thread 無法更改 B thread 的物件，可以利用 global 變數當作溝通的橋樑

```
private:
    /// <Summary>
    /// 設計工具所需的變數。
    /// </Summary>
    int j;  //j = 0
    array <System::Net::Sockets::Socket ^> ^mySocket;
    array <System::String ^> ^receivedata;
    File ^file;
```

四、 動態產生 BackgroundWorker 物件：

快速複習動態一維陣列的產生方式

```
int *a;
a = new int[5];
```

當我們從工具箱產生一靜態的 BackgroundWorker 物件並決定好其 DoWork 事件的 event handler 後，程式碼便自行處理上述動作

1. **宣告一個名為 backgroundWorker1 的 BackgroundWorker pointer**

```
protected:
    /// <Summary>
    /// 清除任何使用中的資源。
    /// </Summary>
    ~Form1()
    {
        if (components)
        {
            delete components;
        }
    }
private: System::ComponentModel::BackgroundWorker^  backgroundWorker1;
protected:
```

2. **給予空間，呼叫 BackgroundWorker 的建構子**

```
#pragma region Windows Form Designer generated code
        /// <Summary>
        /// 此為設計工具支援所需的方法 - 請勿使用程式碼編輯器
        /// 修改這個方法的內容。
        /// </Summary>
        void InitializeComponent(void)
        {
            this->backgroundWorker1 = (gcnew System::ComponentModel::BackgroundWorker());
```

3. **使函式 backgroundWorker1_DoWork 與 backgroundWorker1 的 DoWorkEventHandler 作連結**

```
//
// backgroundWorker1
//
this->backgroundWorker1->DoWork += gcnew System::ComponentModel::DoWorkEventHandler(this, &Form1::backgroundWorker1_DoWork);
```

有了以上的觀念，我們便可以取代從工具箱產生的作法，在程式執行的狀態下，依據不同的情況，產生動態的 BackgroundWorker 一維陣列。