# Win32 行程通訊的觀念與技術

作者: 錢達智 from VCL-Team
本文原發表於微電腦傳真, 1997/11

---

視窗提供的永遠只是局部的景像。身? 視窗的製造者以及使用者的我們不可能不明白這個道理；對於視窗的使用者日益挑剔的品味，視窗的製造者所能提供的不僅止於視窗的大小，往往是視窗的數量。的確，探出窗去看得將更多一些，外頭天空地寬朗朗白日，別的視窗也許正有我們想要的景像。

這篇文章談的是 Interprocess Communication (IPC)，我將與你分享跨行程通訊的各項技術與資料交換的方法。

## ? 什? 需要 IPC?

### ? 什? 需要 Inter-process Communication ?

顯而易見的，沒有一個視窗應用程式可以包辦全部的工作。? 了避免資料重覆輸入的時間浪費與人? 錯誤，各應用程式間的資料會有互相交換的需求。首先的壓力將來自於使用者，甚至於很可能是你自己。先不說別的，在寫這篇文章時，我就曾剪貼原來以 Delphi 撰寫的程式到文書編輯軟體，同時，也利用抓圖軟體幫我拍下執行畫面，最後，這些文章與範例程式得用壓縮程式壓起來，然後 E-mail 寄給雜誌編輯。

使用 IPC 在某些情況下是不得不然的決定，有時候程式必須跨過機器邊界讓另一部機器內的程式明白該怎? 合作來共同完成工作，這同時也暗示我們可能面臨不同的作業系統的問題。

此外，IPC 有助於系統的安全與穩定。由於 Win32 各個行程彼此獨立的特性，一個行程死掉了，其他的行程還可以繼續跑下去，對於某些穩定性要求很高的系統而言，值得以額外的負擔(Overhead)交換系統的強固性(robustness)。嗯! 我的意思是說，因? 系統對穩定性的需求要求較高，值得拆開來做甚至額外的備援系統，既然工作拆成兩個以上，此時必然需要 IPC。

關於 IPC，一般人可能會對其有「執行效率緩慢」的印象，這當然不能說是錯誤的，但絕不是公平的評語。這? 說吧：一個主管親自去做一件事，往往會比先說明再授權下屬去做來得快，這是單一工作時的情況；然而如果管理者同時有好幾件事在手上，託付別人去做才能使得整個公司的效能提高。換句話說，如果能善

用 IPC，整體的系統效能不僅不會下降，反而可能因? 充分利用整個運算群的能力而有提升。

## 我們的第一個 IPC 例子

每個圖形介面的視窗應用程式都接受並處理訊息(Message)，因此，使用訊息佇列通知其他的行程是腦中很自然會浮現的第一個想法；換句話說，兩行程間彼此互相以 SendMessage() 或 PostMessage() 傳送訊息通知對方。

即然要互送訊息，就需要一個彼此都認得的訊息編號。於是，除了 Windows 標準的訊息編號之外，我們還需要額外定義一個(一些)訊息。

行程通訊間用來約定訊息編號常用的方法是呼叫 RegisterWindowMessage() API 函數。這個函數只有一個字串型別的引數，Windows 系統會檢查我們傳入的訊息名稱並傳回一個安全不重覆的訊息編號，假如傳入的訊息名稱早已經登記有案，則系統傳回的是稍早傳給那個行程的相同編號。

換句話說，兩支程式只要彼此都用相同的訊息名稱呼叫 RegisterWindowMessage() 註冊訊息，系統便會都給兩者一個相同的自訂訊息編號。

接下來要送出訊息了，可是，要送給誰呢？嗯，我在這 使用的方法是：第一次先用廣播的，每一個視窗程式都會收到通知，訊息的短三數(wParam)中寫明發訊視窗的 Handle 值，如果是同志，它自然明白這個訊息代表了什? ，並且也使用 SendMessage() 回送約定的訊息表示收到。同樣的，訊息的短三數注明自己的 Handle。於是，茫茫人海的小倆口終於得知對方的下落，以後就不再需要公開尋人可以透過 Handle 值直接與對方聯絡了。

除了訊息編號，訊息的 wParam，lParam 長短三數也可以用來進一步約定通訊的細節。事情進展得似乎十分順利，現在我們知道合作物件，也確信它明白我們的訊息代表什? 。雖然簡單，但是這種暗通款曲的方式是系統默許的。不過，我們還需要再多解決一個問題。

由於 SendMessage 只有 wParam，lParam 兩個 DWORD 型別的長短三數，攜帶的資料量十分有限。很顯然的，我們需要能夠一次傳送更多資料的方法。Windows 也的確提供了許多交換資料的機制，我在這篇文章中將會一一說明，其中最簡便的方法是使用 WM_COPYDATA 訊息，作法如下

- 將資料內容指定到 COPYDATASTRUCT 這個資料結構中。

- 必須使用 SendMessage()送出 WM_COPYDATA 訊息，訊息的短三數是發訊端視窗的 Handle 值，長三數的內容則是指向 COPYDATASTRUCT 的指標。

- 受訊端行程收到訊息時，以長三數提供的線索依址取回資料。

小倆口書信往返時系統是居中牽線的紅娘。就在發訊視窗送出 WM_COPYDATA 訊息,受訊視窗取得內容之間，系統在背後默默接管記憶體管理的瑣事。有關 WM_COPYDATA 的使用有一點需要提醒讀者的，收訊端應該視這塊記憶體是唯讀的，如果後來程式處理需要這些資料，應該要先將之拷貝出來。

多虧有了這項特殊的性質，使得 WM_COPYDATA 與訊息溝通模型成? Win32 平臺上少數同時支援 16-bit 與 32-bit 應用程式的 IPC 機制。你可以在 WM_COPYDATA 目錄找到範例程式 TwinApp 的完整原始程式。

## IPC 基本概念的討論

總結來說，上述的例子是兩個行程彼此利用 RegisterWindowMessage()註冊所得的編號對送訊息，並且利用訊息的長短三數進一步協定通訊的內容與細節，對於資料量比較大的資料則使用 WM_COPYDATA。

眼尖的讀者在檢視 TwinApp 時也許會察覺到一些 DDE 的影子。當然，比起 DDE 來說，TwinApp 範例程式的訊息溝通模型實在陽春，缺點也不少。不過我的用意本來就不在於一開始就寫一個大型程式出來嚇唬人；相反的，我打算提供一個簡單的例子，並且從這個例子支解出有關行程通訊的幾個重要的觀念與特性，這些特性並不是 TwinApp 所獨有的，對於其他 IPC 機制的討論也有相同的價值，等我們扣緊了對 IPC 的感覺，再陸續討論其他 Win32 平臺所支援的 IPC 機制。

話說內行的看門道，外行的看熱鬧。或許我算不得頂尖高手，但至少應該比看熱鬧的多看出一些東西來吧! :p 觀察 TwinApp 這個例子 --

- 行程之間彼此有共同的通訊協定

- 通訊的僅限於單機，稍候討論的 IPC 有些則是可以跨過機器邊界甚至網域.

- Process 在行程通訊中的角色扮演

一般來說，三與 IPC 的行程可以歸類成 Client 與 Server 兩類，所謂的 Server 指的是提供服務的行程；Client 指的是使用或向 Server 要求服務的行程。

真實的世界中，人的角色扮演是隨情境而變的。我們會是別人的子女，但也同時是別人的爸媽; 即使同樣是夫妻，居家生活與外出場合的行? 表現也有差異。界定某一程式是 Client 與 Server 的角色端視當時的情況而定並非絕對的。舉例來說，文書處理軟體可能向試算表要求庫存統計資料，此時試算表扮演的是 Server 的角色，但在試算表向庫存管理系統索取統計資料的場合，試算表則是 Client。

以我們的第一個例子 TwinApp 來說，彼此既接收訊息，同時也主動發出訊息。既可以是 Client 也可以是 Server，沒有明顯的主從之別，對於這樣的情況，有一個專有名詞叫「對等模式」(Peer-to-peer model) 。

- 同步與非同步的討論

  TwinApp 使用 SendMessage()送出訊息，程式會暫停在 SendMessage()那行等待訊息處理結束返回後再繼續下一列程式，這樣的情況屬於同步處理。同步(Synchronous)與非同步(Asynchronous)在 IPC 中是一個非常重要的論題，有必要先對這兩個名詞先做說明:

  假設程式 A 呼叫程式 B 時，若是 A 先暫停一直等到程式 B 結束返回後再繼續程式 A 的下一動作，我們稱其? 同步(Synchronous)；另一種情況是 -- 如果 A 呼叫 B 之後，不等 B 執行完，就直接進行 A 的下一動作，則是所謂的不同步。

  以提款機? 例，我們會先插入卡片，輸入密碼，鍵入金額，然後是內部安全與帳務查核，最後收回卡片及金額，列印交易明細，一動接一動按步就班；同樣是提款這件事，某位老闆可以交待會計小姐去提款，交待完之後他就逕自去忙別的事，等到會計小姐提款回來，再向老闆回報，這樣的程式是所謂的非同步。

  如果進一步觀察提款這個例子：會計小姐什? 時候出門什? 時候回來是算不得准的，假定這位老闆除了會計小姐之外，另外還交辦旁人其他工作，可以預見的，不一定那一件工作會先做完。由於執行的

次序無法預估，採用非同步方式設計的行程通訊將會多出許多協調與事件處理的工作，使得彼此之間總互相期待點什？。

- 三與通訊的行程個數，訊息資料的流向

  在 TwinApp 中，簡單的只有兩個端點。但在實際應用的場合，Server 通常得同時應付好幾個 Client 的要求，如何妥善照顧到每一個 Client 同時要兼顧系統執行的效能，是門很大的學問。

  當行程對行程搭起通訊的鵲撟時，這座撟是單行道或者是雙向通行，同樣也值得列入評估要素。不過有一點需要注意的: 不論選擇單工或雙工的 IPC 機制，並不構成我們建立雙向溝通無可跨越的天塹，話說山不轉路轉，蓋兩座單向的撟一樣可以有雙向通行的效果，不過就先天本質的特性來說，某些 IPC 機制確實比較容易作出雙工的效果，當然也有天生大嘴巴適合用來廣播的，例如本文稍後？述的 MailSlot。

- 資料的可視性與安全性

  交換的資料在行程之間當然必須是可見的，TwinApp 是用 WM_COPYDATA 交出資料。IPC 有些技術是可以讓行程共同存取資料的，稍候我們在 Shared memory 時將有討論.

- 是否需要有視窗或者純 Console Application 也能應用.

  TwinAPP 是以 SendMesasage()送出訊息，這表示需要有視窗才行得通。如果你設計的是純 Console Mode 的應用程式，那？，選用不需要視窗 Handle 也行得通的 IPC 機制(例如 pipe)會比較適合。

- 關於執行效能的討論

  許多人耽心 IPC 的執行效能，的確，先不說別的，光是？動另一個 Process 本身就比？動一個 Thread 的 Overhead 要高上很多。如果涉及協調的問題，建立一個 Mutex 的時間也比 Critical section 慢上不知多少倍。遺憾的是我們卻也別無選擇，因？ Critical section 在 Multi-Thread 中固然簡單好用，但是不能用在跨越行程邊界的場合。
  但是要說 IPC 一定使得系統效能降低，未免也太過

悲觀了；平視與俯看的視野是不同的。這年頭大家都將 Client/Server 挂在嘴邊，充分運用合理分配整個公司的運算資源才能提高整體的效能，我想 IPC 在這 自有其應用的價值與效益。

另外一個導致 IPC 執行效率不彰的元兇來自不良的設計，著名的例子是所謂的 Busy-loop 一個什？也不做只有一行程式不斷地期待的回圈。以稍早的老闆與會計小姐？例，如果老闆交辦事情之後卻將全部的事都停下來，來回踱步只？專心等著會計小姐回來，時間沒有花在刀口上的結果當然效率不彰。找出效率的瓶頸設法調校是件長期奮戰的工作，如同管理是持續不斷的合理化。

此處還有一個迷思也有待澄清，同步與非同步對於執行效能的影響是視情況而定的，並不能說非同步一定會比同步快，抽樣樣本很小或資料量偏小時，同步往往比非同步快。比較公允的說法應該是：同時有好幾件工作要處理時，整體來說「非同步」往往快一些。以剛才的提款的事情？例，老闆親自去提款未必比小姐慢，但是如果老闆同時有好幾件工作要處理時，非同步的好處就很明顯了。

## Win32 支援的 IPC 相關技術

上述的討論與其說是針對 TwinApp 的觀察，不如說是針對 IPC 的綜合討論。觀念的說明之後是技術層次的討論。接下來陸續介紹的是 Win32 API 支援的各項 IPC 機制 --

- Clipboard
- COM
- Dynamic Data Exchange (DDE)
- File Mapping
- Mailslots
- Pipes
- RPC
- Windows Sockets

- WM_COPYDATA

## 剪貼簿(Clipboard)

人，其實是最佳的 IPC 機制，十分的聰明也十分的有彈性。

剪貼簿幾乎是專? 人類而設的標準資料交換中心。它最大的特色除了使用者導向之外，任何應用程式都允許改寫其內容，同時它是可以跨越機器邊界，交換的範圍不僅限於單機內的各個行程。

由於它是純使用者導向，使用剪貼簿的程式有一項傳統是值得遵守的：如果不是基於使用者的操作，程式不應該主動去異動剪貼簿的內容；同樣的道理，我們也不應該假設剪貼簿中有我們程式想要的資料，哪怕是不久前才剛放進去的，因? ，使用者可能已經清除或改變其內容了。

剪貼簿幾乎可以容納任何的資料，除了標準支援的 CF_TEXT、CF_BITMAP...等資料格式，我們可以自行註冊登記其他格式的資料。但由於它的使用者導向，也由於任何程式都可以改寫其內容，除非使用者願意，不然坦白來說不太適合行程間的資料交換。這也使得應用設計 IPC 時，剪貼簿成? 每支應用程式都標準支援但卻也都適可而止的 IPC 機制。我們應該再多看看其他的資料交換方法。

## File Mapping

在早期 MS-DOS 時代還沒有現在這? 多 IPC 機制可供利用時，使用磁碟檔案來交換資料可說是一般應用程式的唯一選擇。時至今日，檔案不僅沒有從 IPC 領域中消失，反而是更加發揚光大了，然而觀念上早已不純粹界定在檔案系統的實體檔案。的確，資料位元於何處的份際如今是越來越模糊了，虛擬的記憶體實際上是檔案，虛擬的檔案結果是記憶體。

Win32 API 中有一個好玩的東西叫做 File-mapping；基本的觀念是開? 一個檔案並將之對映到某一塊記憶體，有趣的是，雖然程式是針對這塊記憶體操作，實際上改變的卻是檔案。
更好玩的是你不必真的在硬碟 開一個實體檔案，而是使用分頁置換檔(paging file)的一塊空間權充當作檔案。這個虛擬的檔案空間(或者你要說是記憶體)可以? 行程間共用，通常我們管它一個特別的名字叫 Share-memory，共用記憶體。

由於它的確不是真正的檔案，行程間不僅省去特定磁碟目錄檔案等約定，也毋須在意誰是最後走的要負責刪除檔案，當然啦，即使當機不會留下一些垃圾檔案。彼此分享的是正好是同一塊記憶體，資料一旦寫入，這項改變也立即反應到別的行程。

使用 ShareMemory 的大致步驟如下所述

- 呼叫 CreateFileMapping() API 函數建立 File-mapping 核心物件.

  CreateFileMapping()函數的第一個引數原本應該是 CreateFile()開檔所得的檔案物件 Handle，若是傳入 $FFFFFFFF 則是以分頁置換檔(paging file)的一部劃作共用記憶體。函數的最後一個引數是這塊區域的三考名稱，行程間彼此將根據此一相同的識別名稱三考同一塊共用記憶體。

  ```
  FHandle := CreateFileMapping(
   $FFFFFFFF, // Shared memory File，Handle 傳入 $FFFFFFFF
   nil, // 不設安全屬性
   PAGE_READWRITE, // 存取模式設定? 可讀寫以便行程交換資料
   0, // 使用 paging file 時一般將之設? 零
   Size, // 共用記憶體的大小
   pchar(name)); // 其他的行程將以此名稱三考到這塊共用記憶體
  ```

- 由於各個行程各有其邏輯定址空間，在正式存取這塊共用記憶體之前，我們得將其全部或部分映射回行程本身的位址空間中。呼叫 MapViewOfFile()的用意即是在此，該函數將傳回 mapped view 「視野」的起頭(就是指標啦)，接下來的就是用這個指標存取記憶體了。

  ```
  FFileView := MapViewOfFile(
   Fhandle, // File-mapping object 的 Handle 值
   FILE_MAP_ALL_ACCESS, // 設? FILE_MAP_ALL_ACCESS 開放存取
   0, // 模式以便順利存取共用記憶體
   0,
   Size); // 預備映射回來的 byte 數
  ```

- 最後，別忘了使用 UnmapViewOfFile()歸還指標並呼叫 CloseHandle()釋放 File-mapping 核心物件。

礙於篇輻，完整的程式碼請讀者三閱 ShareMem 目錄的 DemoSMem 專案。另外，？了方便使用，這些 CreateFileMapping(),MapViewOfFile()等函數已經包裝進 TSharedMem 這個類別。

## Mutex

Shared memory 的示範專案 DemoSMem 留下諸多懸疑待解，或許你也正有相同的疑問：既然兩個行程都利用這塊記憶體，那我們怎？知道什？時候資料改變了？此外，如何防止行程同時讀寫資料？

的確，行程通訊既是兩個以上的個體，協調是必然存在的負擔，要避免兩個行程同時使用關鍵資源，Mutex(互斥器)的使用是你必備的技術。

從字面上解釋，互斥意思是同一時間唯一；換句話說，同一時間最多只許握有 Mutex 的執行緒(Thread)有權使用關鍵資源，其他的執行緒若要使用只有等待。嗯! 在 Mutex 與 Event 這兩節我將暫時改口？執行緒，事實上這才是真正的 CPU 排程單位，由於每個行程至少有個 Thread(主執行緒)，這樣的稱呼應該是與本文行程通訊的主旨不相違背的。

就像是註冊訊息，共用記憶體一樣(甚至稍後的 Event，MailSlot，Pipe 都是)，在我們取得核心物件的 Handle 前，都是以「名稱」三考的，？生一個 Mutex 的 API 函數是:CreateMutex(), 以下範例采自本文所附的 ChienIPC 程式單元

```
constructor TMutex.Create(const name: string);
begin
 FHandle := CreateMutex(
 Nil, // 安全防護屬性, 暫時傳入 nil 採用預設值
 False, // 執行緒是否一開始就握有 mutex 的所有權
 pchar(name)); // Mutex 核心物件的名稱
 if FHandle = 0 then Abort;
end;
```

好極了，現在我們有了一個 Mutex，該怎？使用呢? 我用一個情節來說明：如果一群人在一起開會，每個人桌子前面各擺著一支麥克風，？了讓大家聽清楚彼此說什？，這些麥克風暫時都是關的，規定只有主席可以透過中央控制系統開？回路。要說話的得先舉手表示：「我要我要」，如果沒有別人舉手也沒人正在發言，主席便打開開關將發言權交給他，然後這個人的手放下開始講話。此時若是其他人也要講話，根據規則得先舉手，在別人講完交出發言權前只有繼續舉手等待的份。當然，排隊的人，可以選擇手一直舉著；或者他只打算等三分鐘，手酸了就放下來。

執行緒要求擁有 Mutex 的方法是呼叫 WaitForSingleObject()(我要我要，舉手等待)，此時程式將暫停(Blocking)在這列。倘若此時正好沒有別的執行緒擁有 Mutex (沒人講話)，系統會短暫的將 Mutex 設

? Signaled(激發狀態)，使得 WaitForSingleObject()正常返回，同時，系統也會將這個 Mutex 的所有權交給這個執行緒，然後程式繼續執行，握有 Mutex 所有權者開始使用關鍵資源，並儘快在事後以 ReleaseMutex()交出 Mutex 擁有權。

關於程式實入這部分請您三閱 DemoSMem 範例程式的讀取與寫入程式，同樣的，有關 Mutex 的 API 函數也已包裝進 TMutex 類別方便你的使用。

## Event

討論過行程之間以 Mutex 協調避讓的技術之後，Shared memory 的示範專案 DemoSMem 尚留下一個懸疑待解：既然兩個行程都利用這塊記憶體，那我們怎? 知道什? 時候資料被改變了呢? 以一個回圈定期不斷去抓資料回來比對不僅程式寫起來累人，執行效率也很低落。

當然，回到一開始提出的方法，寫入資料的行程用訊息一一個別通知其他合作夥伴是可以行得通，不過，事情該有更好的解決之道才是。Win32 的核心物件中有一種叫 Event(事件)物件，方便我們在某一事件發生時設定其狀態以便三與通訊的行程注意到某一件重要事情的發生。

? 生一個 Event 物件的方法是呼叫 CreateEvent() API 函數:

```
HANDLE CreateEvent(
 LPSECURITY_ATTRIBUTES lpEventAttributes,
 BOOL bManualReset, // flag for manual-reset event
 BOOL bInitialState,// flag for initial state
 LPCTSTR lpName // address of event-object name
);
```

同樣的，最後一個引數是執行緒在取得 Event Handle 前三考同一 Event 物件的識別名稱，如果相同名稱的 Event 物件稍早已經? 生而且三用次數尚未歸零消滅，並不會多? 生一個 Event 物件，系統只單純的將其三用次數加一，執行緒彼此得以三考到同一個物件。第三個引數用來設定 Event 物件的初值是否? Signaled(激發狀態) 。第二個引數用來設定事件的激發狀態是手動或自動；所謂手動與自動的分別在於事件的狀態變成 Signaled(激發狀態)時，要由系統自動幫我們重設回非激發狀態，或者由程式自行以 ResetEvent()將事件設成非激發狀態。

觀察 DemoSMem 的作法是這樣的：當某一個行程修改了 Shared memory 的內容時，該行程以 SetEvent() API 函數將 Event 物件的狀態設? Signaled(激發狀態)，三與行程通訊的各支程式在開跑之

初，除了以相同的識別名稱建立(三用)Event 物件之外，還特別分派另一個 Thread 專司偵測特定 Event 物件激發狀態的任務,一旦物件激發了，表示一定某一個行程修改了 Shared memory 的資料，此時我們知道該是重新讀取資料內容的時候了。

呼! 終於將 Shared memory 的範例程式 DemoSMem 講完了，下圖是它執行的畫面，彼此看來是毫無關聯，但是經由共同分享的記憶體與 Mutex，Event 兩種同步協調技術，彼此正在密切交換意見。

圖: DemoSMem 執行情形

## MailSlot

執行 DemoSMem 時如果讓你有廣播的感覺，接下來要說的 MailSlot 會讓你更有廣播的感覺，而且它是可以跨越機器邊界向網路廣播的。從字面上看來，這像是與寄信有關的通訊機制，實際上它的行？也的確與其名稱相符合。MailSlot 就像是你的信箱，只要知道地址，任何人都可以寄信給你，不過，只有你才可以打開信箱讀信。

MailSlot 是一種由系統維護的虛擬檔案，建立並擁有 Mailslot 的行程扮演 Server.的角色，其他的行程包含 MailSlot Server 本身的行程均可以開？ MailSlot 寫入訊息，不過，只有 MailSlot Server 可以讀取資料的內容。這是個單一 Server 多個 Client 的機制，同時，資料只允許由 Client 對 Server 單向傳送。

我想你可能也習慣了，要？生一個 MailSlot 物件大概也需要一個識別名稱吧! :p 說不定連 CreateMailSlot()函數名稱都猜得一字不差。不過，這次的名稱可不像先前那樣可以隨便高興取什？就取什？的，它具有以下的固定格式:

\\ServerName\mailslot\[path]name

我第一次看到時心想: 天哪! 這該怎？填呀? 邊舉例邊說明會比較容易懂

\\.\mailslot\MyMailSlotName MailSlot 的識別名稱一定從「\\」雙倒斜線開始。接下來的是機器的名稱或組群網域的名稱，這 的「.」句號代表的是行程所在的那部機器。再來是「\mailslot」，對於 MailSlot，一定是這個單字照抄就是了。最後則是你自訂的 MailSlot 名字。先前提到 MailSlot 實際上是特殊的虛擬檔案，所以，要當它是檔名應該也是說得通的。

的確，援引我們對於檔案系統的概念，MailSlot 的識別名稱就像路徑檔名一樣，可以經過適當的階層加分類管理，例如:
\\.\mailslot\Account\Note。最後再看一個例子:
*\mailslot\MyMailSlotName，其中「*」指的是群組內的所有機器。

說得夠多了，讓我們動手做做看吧! 首先是建立 MailSlot Server 的例子，取自本文所附的 ChienIPC 這個程式單元

```
procedure TMailSlotServer.Open;
var
 ASlotName: AnsiString;
begin
 if FActive then Exit;
 // 構成 Mailslot 識別名稱
 ASlotName := '\\' + FServerName + '\mailslot\' + FSlotName;
 FHandle := CreateMailslot(
 pchar(ASlotName), // MailSlot 識別名稱
 0, // 訊息長度的最大值，設? 零表示不限
 MAILSLOT_WAIT_FOREVER, // read time-out
 nil); // 安全屬性，先暫時採用預設值
 if FHandle = INVALID_HANDLE_VALUE then
 FActive := False
 else
 begin
 FActive := True;
 FWaitThread.Resume;
 end;
end;
```

再強調一次，只有 MailSlolt Server 才可以讀取資料，讀取的方法是先以 GetMailslotInfo()偵測訊息的長度與數量，然後以回圈逐一配置記憶體並以 ReadFile()讀出資料(別忘了 MailSlot 也是檔案)，以下是一則範例:

```
procedure TMailSlotServer.ReadFromMailSlot;
var
 NextSize: DWORD;
 MessageCount: DWORD;
 Result: BOOL;
 Buffer: pchar;
begin
 if FHandle = INVALID_HANDLE_VALUE then Exit;
 // 偵測 MailSlot 中是否有資料
 Result := GetMailslotInfo(Fhandle, nil,
```

```
 NextSize, @MessageCount, nil);
 if not Result or (NextSize = MAILSLOT_NO_MESSAGE) then
 Exit;
 // 如果還有資料 (MessageCount <> 0)，逐一讀出資料
 while Result and (MessageCount <> 0) do
 begin
 // 資料的長度
 Buffer := AllocMem(NextSize + 1);
 try
 // 讀出資料
 FileRead(Fhandle, Buffer^, NextSize);
 if Assigned(FOnDataAvailable) then
 FOnDataAvailable(Self, StrPas(Buffer));
 finally
 FreeMem(Buffer, NextSize + 1);
 end;
 // 繼續看看 MailSlot 中還有沒有資料
 Result := GetMailslotInfo(Fhandle, nil,
 NextSize, @MessageCount, nil);
 end;
end;
```

至於 MailSlot 的 Client 程式則沒有什？好說的，就當是檔案逕行開？與寫入即可：

```
procedure TMailSlotClient.Open;
var
 ASlotName: string;
begin
 if FActive then Exit;
 // MailSlot 的識別名稱
 ASlotName := '\\' + FServerName + '\mailslot\' + FSlotName;
 // 開？ MailSlot(檔案)
 FHandle := CreateFile(pchar(ASlotName),
 GENERIC_WRITE, // Client 端對於 MailSlot 只能寫入
 FILE_SHARE_READ, // 設定？可供分享讀取
 Nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
 FActive := FHandle <> INVALID_HANDLE_VALUE;
end;

function TMailSlotClient.WriteIntoMailSlot(
 const Data: string): integer;
begin
 Result := 0;
 if FHandle = INVALID_HANDLE_VALUE then Exit;
```

```
 Result := FileWrite(Fhandle, Data[1], Length(Data));
end;
```

稍早提到 MailSlot 適合於跨越機器邊界的網路廣播, 可是我也說明了只有 MailSlot Server 才可以讀取資料，那要怎? 廣播啊？答案在於 MailSlot 的名稱。別的機器如果也用相同的名稱建立 MailSlot Server，一旦任一個 Client 對某一個 MailSlot(也是經由名稱來三考)送出訊息，這份訊息會遊向網路節點上各個指定同名的 MailSlot, 這樣子就達成廣播的效果。至於訊息是怎? 流來流去的，就留給系統與網路底層去傷腦筋了，程式只管以檔案寫入資料的方式送出資料即可。

使用 MailSlot 時很可能你會遇到訊息重覆的問題；也就是說，雖然 MailSlot Client 端只寫了一個訊息，但相同的訊息 MailSlot Server 卻可能收到兩份。原因是這樣的：由於 Win32 多重通訊協定的緣故，MailSlot 在廣播時，並不知道到底該採用哪一條路徑，於是便各種可能的通路都傳了一份。情況有點像在發佈颱風警報，我們在電視，廣播與網路都同時會曉得有颱風要來的消息。解決的方法是在資料開頭處加上一些控制用的編號代碼，Server 據以判斷是否是相同的資料。

像 MailSlot 這樣的通訊機制可以應用在哪些場合呢? 著名的例子是 WinPopup，剛才我也寫了一支陽春的, 次圖是 MyWinpop.exe 執行的情況。由於 MailSlot 廣播的特性，十分適合網管時用來知會使用者重要的訊息，此外，MIS 系統也可以用它適時的報告異常狀況，各使用者如果在「開始? ? 動」中都放置這支小程式，彼此便可以之交換訊息，當訊號進來時，也會立即顯示訊息的內容。

圖: MyWinPop.exe 執行情形

當然，你還可以想得到其他的應用。像我就覺得它很適合用來作? 程式除錯工具，不僅可以將程式執行的過程與情況記錄下來，而且程式在網路上各節點的執行狀況也將源源而來，這是一般的測試方法所不容易達成的效果。

## Pipe

看過廣播式的 MailSlot 後，Pipe 則是點對點的通訊機制，資料允許單向或雙向於管子連接的兩端移動。pipe 可分? Anonymous pipe 與 Named pipe 兩種，Anonymous pipe 的資料只能單向流動，而且僅限於單機內使用，但卻是行程重導其標準輸出(Standard Output)成? 另一行程之標準輸入的方法；Named pipe 就如同先前討論的各項 IPC 機制，由於有一個識別名稱，其他的行程很容易可以依

照名稱找過來，通訊範圍不限於單機，同時，資料允許雙向流通。

## DDE

如同本文第一個 TwinApp 這個例子，DDE 也是建立在訊息通訊這個基礎上的，不過它的協定內容顯然嚴謹很多。

DDE 是由 Client 端以 WM_DDE_INITIATE 廣播訊息起拉開通話的序幕，Server 端受理後以 WM_DDE_ACK 回應，連通後則是一連串 Server 與 Client 間彼此互送 WM_DDE_DATA、WM_DDE_REQUEST、WM_DDE_ACK 等訊息。實際的資料並不是真的經由訊息傳遞，而是提供線索彼此利用 Atoms(由 Windows 系統提供的字串對照表)尋求 Application(應用程式), Topic(主題)與 Data(資料)等三個專案。最後，以 WM_DDE_TERMINATE 訊息結束對話。

行程間建立 DDE 連接時，當 Server 端的資料改變時，依資料交換的頻繁與 Client 的主動程度，其通道的形態可分？：

- Cold Link：來要才有；Client 端得主動要求傳送資料，如果沒有來要，即使 Server 的資料已經改變很多了，Server 對 Client 也置之不理。
- Hot Link：有變就給；當資料改變時，Server 端將主動通知 Client 改變的內容。
- Warm Link：更新通知；當資料改變時，Server 端只對 Client 端告知資料改變的消息，真正的資料要等 Client 提出要求才會送出。

由於 DDE 訊息通訊牽涉的實作細節頗多，？了使用方便起見，微軟也提供 DDE 管理函式庫(The DDE Management Library, 簡稱 DDEML), 使用上的最大差別在於使用 DDEML 的程式是用 Callback 函數處理 DDE 交易(Transaction)。另外，三大專案的 Application 改口叫做(Service name)服務。

時至今日，討論 DDE 的文獻已不在少數，的確，DDE 的使用應該是容易許多了，幾乎沒有一個 Windows 程式開發工具不提供一些元件或類別讓程式員更方便製作 DDE Server 或 Client 程式。當然，如果你的需求只是在行程間通知某些消息，自行設計一套訊息通訊協定倒也簡單得以完成任務，我想本文的第一個例子 TwinApp 是一個不錯的提示。

## 其他的 IPC 技術

EXE 通常呼叫 DLL 的輸出函數(exports function)，某些情況下 DLL 也會使用 EXE 事先預備的回呼(Callback)函數。函數呼叫這個觀念與想法如果移植到行程通訊中會發生什？事呢? 我的意思是說，讓一個行程呼叫另一個行程的函數。Ya! 這就是所謂的 RPC，行程之間屬於函數呼叫層級的合作。可以想見的，由於行程各有其定址空間，如同 OLE，要達到 RPC 確實需要額外標準的介面加以定義。

有關 IPC 的技術與觀念我們已經介紹得不少了，不論是訊息交換，剪貼簿，Shared memory，DDE，MailSlot，Pipe 等等，幾乎都是資料的交換或者 Client 與 Server「要求-回應」，三與通訊的行訊必須對於交換的資料有一定程度的瞭解與處理能力。換句話說，在我們以 DDE 向試算表軟體要求傳回資料後，這份資料到底代表什？得自己解釋；同樣的，如果要傳入資料到試算表軟體，即使透過現成元件的幫忙，仍然必須對試算表軟體有基本的認識。

話說回來了，只有試算表自己最清楚資料代表什？，不是嗎？那？，由它來處理資料應該才是適當的人選，強以外部程式去操作總有外行人指導內行人的遺憾。利用 OLE 技術將應用程式整合在一起工作確實是比較合理的作法，如果 COM 物件可以像電子 IC 一樣安插進我們的程式與我們的程式一同工作，那這種我們稱之？ OLE Control(ActiveX)，距離拉大到網路上，DCOM 這個名詞你一定聽說過.

想想看，終於我們可以用甲公司的統計圖表元件，然後用乙公司的元件將圖表傳真出去，這樣窗景真是美好。視窗確實只提供局部的景像，但是加裝了望遠鏡的視窗可是一個天文臺，加了風鈴的視窗所提供的就不只是景像了，還有悅耳的聲音。

不論是 RPC 或 OLE，我想這都是屬於本文應該討論但肯定是來不及討論的，這兩個主題甚至以單篇文章來談都不怎？夠用。事實上，有些地方(例如 DDE 這一節)我也沒有提到技術方面的實作細節，礙於篇輻(這篇文章已經太長了) 日後我們會在本專欄繼續以專文介紹 RPC 等主題。關於以 Winsock 作？ IPC 通訊機制這部分，本專欄的前一篇文章「走! 讓我們上 BBS 聊天去」才剛說明過，在此就不再重覆了。

## 應用 IPC 到你的程式中

各項 IPC 的技術往往以各種方式組合在一起。例如本文提供的 DemoSMem 範例程式就同時用了 ShareMem 交換資料，同步機制則採用 Mutex 與 Event。情況並不如想像中的複雜：既是行程通訊，那必然是兩個以上行程之間的事，既是分開的，中間一定有介面存在，定義這個介面的具體內容就是所謂的協定，留意資料

交換的位置與方式，需要協調避讓的採用合適的同步控制加以處理。這些重點把握住了，應該心 就已經有數了。

面對各式各樣的技術時，如果你正考慮應用 IPC 到你的程式中，首先得正視自己的需求，不妨提出類似以下的問題問問自己，最好將之寫下來

- 是否真的需要跨行程處理，成效何在？
- 技術實作的難易程度與所需付出的成本
- 資料的流向是單向或雙向，需不需反饋(feedback)的控制查核
- 這些工作只在單機完成，或者需要連上網路，範圍只在公司內部區域網路或者是廣域網路
- 三與通訊的行程最多與平均的數量是多少？
- 只在一種作業環境，或者可能同時要滿足不同的作業平臺
- 執行效能( performance)是不是關鍵需求.
- 應用程式使用 GUI 介面或者 console mode

接下來開始比較各項 IPC 的特性，哪些是與你列出的需求相符合的，有沒有哪些限制是你必須要排除而避免使用的，各項 IPC 經過與先前寫下的需求交叉評比的結果，積分高的自然是脫穎而出。最後，事情如果能簡單解決是最好，開發時程縮短成本自然降低，而且日後維護容易。

## 結語

技術是不斷推陳出新的，當各式各樣的 IPC 機制提出時，回顧行程之所以開始通訊合作的初衷是有必要的，唯有回到最初原始的簡單需求，才能看出技術演進過程的緣由與其修正的價值，不斷的變易之中我們可以粹化出一些不變的原則與觀念，而這些原則應該是與最初的需求互相吻合的。

以 IPC 這? 大的題目只寫一篇文章是件很痛苦的事，我不清楚有多少讀者會期待以一篇文章能將 IPC 的技術細節講清楚，不過已盡我所能的交待來龍去脈。觀念與說理太占篇幅，大部分的技術細節是隱藏在範例的原始程式中，這些程式日後如有修改或加強，您可以在我的網站(http://www.chih.com)找到更新後的版本，

**對於本文如有任何意見或評論，也歡迎您 E-mail 與我聯絡, 來信請寄** wolfgang@ms2.hinet.net **或** chien@chih.com。

---

## 三考資料


- Charles Petzold, Programming Windows 95
- Jim Beveridge & Robert Wiener **著, 侯俊傑譯**, Win32 **多緒程式設計**, Multithreading Application in Win32
- MSDN Library CD, 1997/7, SDK Documentation / Platform SDK / Windows base services / Interprocess Communication