第五章　系統軟體設計－軟體設計概念

本章將介紹系統軟體設計之概念。圖 5-1 為系統軟體設計之軟體設計概念架構圖，其中包括了應用程式開發流程，兩種元件的不同處理方式等。

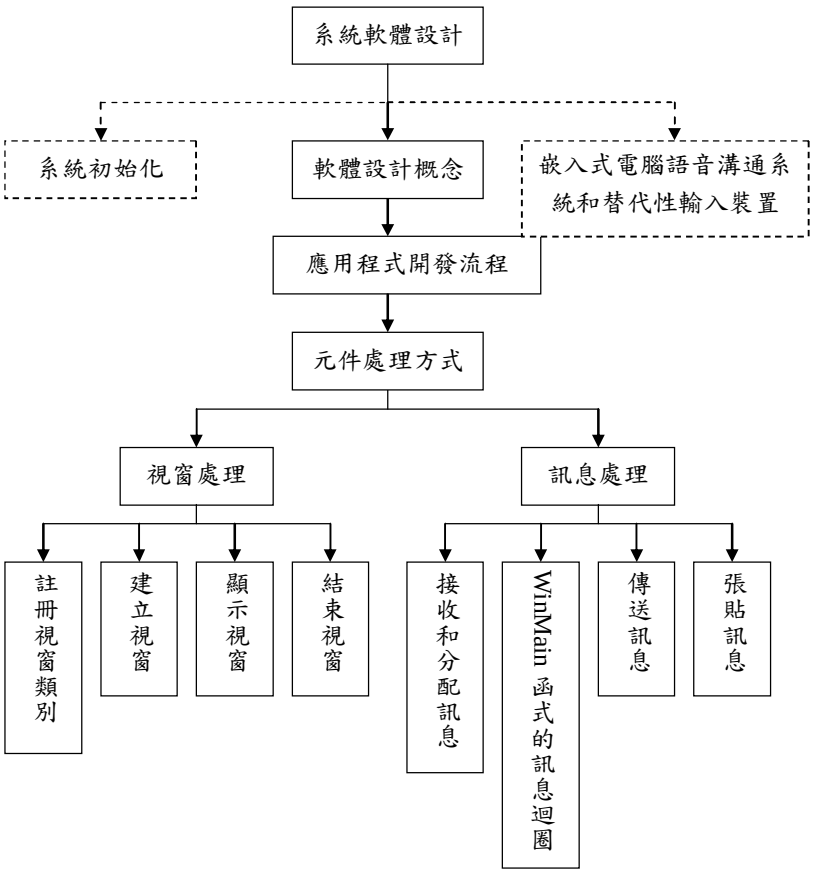

圖 5-1 系統軟體設計之軟體設計概念架構圖

由於本系統是針對學齡前兒童，特別是腦性痲痺症的小孩而設計，因此在圖片的選擇主要以小朋友生活周遭可能遇到的詞彙進行收集，並加以整理而製成最後的溝通系統版面。至於替代性輸入裝置的程式設計，則是利用對印表機埠的控制而達到此功能。

由於我們使用嵌入式系統進行開發，因此在程式軟體的編輯方面，我們必須

摒棄現成桌上型電腦常用之程式編輯軟體例如 BCB(Borland C++ Builder)、VC++(Visual c++)等，而採用 Evc++(eMbedded Visual C++)軟體作為程式設計的平台。

一、應用程式開發流程

建置應用程式涉及了前置處理器(preprocessor)，編譯器(compiler)和連結器(linker)。因此，在進行編譯(compile)之前，需讓前置處理器透過翻譯巨集、運算元和指示來準備編譯器(compiler)的來源檔(source file)。接著，編譯器就會建立一個含有機器碼、連結程式指示器節、外部參考和函式/資料名的物件檔(object file)。此時連結器(Linker)就會將編譯器創造的物件檔和函式庫(Libraries)連結，以產生執行檔(*.exe)。

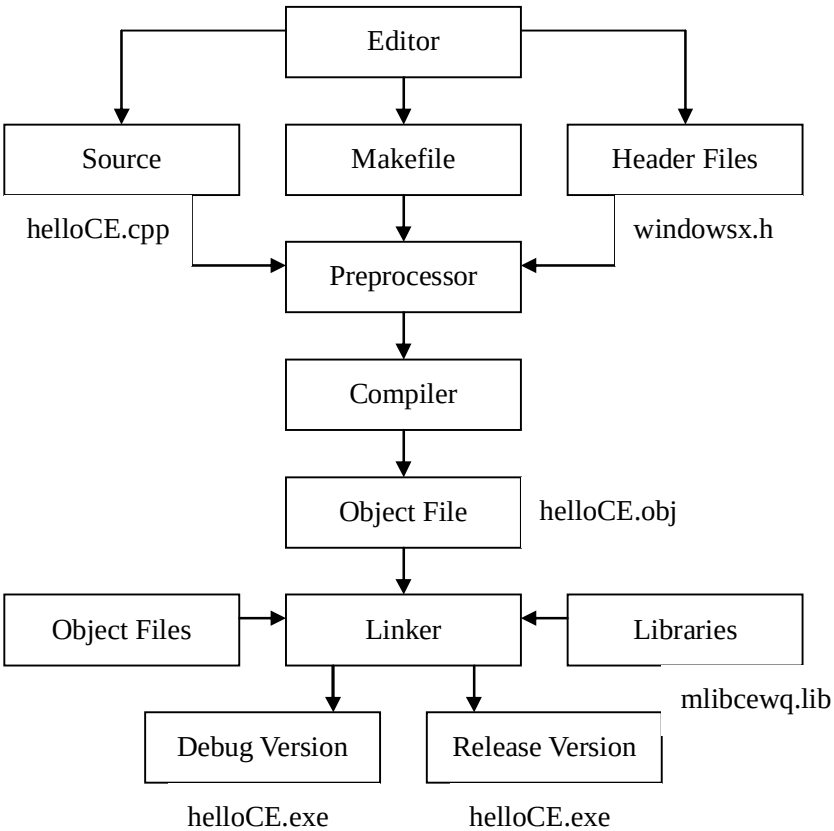

圖 5-2 eMbedded Visual C++開發應用程式流程圖

圖 5-2 列出了程式的建置流程，當程式在相關 editor 裡編輯時，包括的檔案主要有三種，一種是我們自己編輯之 Source Files，例如 helloCE.cpp 檔案；第二種是標頭檔(Header Files)，例如 windowsx.h，但這些標頭檔是系統提供之 include 檔，我們只需將程式所需之標頭檔 include 至程式內即可；第三種是 Makefile，主要是針對不同 Source Files 在進行編譯時其關聯性為何的檔案。由於我們利用 eMbedded Visual C++應用程式進行開發(而不是利用工作站編輯軟體例如 joe 或 vi 等)，因此不需自行定義 Makefile，只需設定正確，應用程式會自動將不同 Source Files 之關聯性加以串接并進行編譯。

應用程式的流程建置完成後，接著介紹 Windows-based 的應用程式。它與 MS-DOS 或 Unix-based 的程式是有很大的差別。所謂 MS-DOS 或 Unix-based 的程式是用 getc-及 putc-的方式，讀取從鍵盤輸入的資料，這些稱謂 pull-style。至於 Windows-based 程式則是用 push-style，即若輸入發生，作業系統就會透過傳送訊息(message)到應用程式視窗(application window)讓程式執行應該完成的步驟[29]。

二、元件處理方式

下面我們將介紹程式設計裡主要的元件功能處理方式，其中包括視窗處理和訊息處理。

（一）視窗處理

每個視窗都有一個唯一當作視窗的識別字(identifier)。當成功建立視窗

時，會傳回視窗的處理數(handle)，接下來在呼叫其他函式使用這個視窗時，必須傳入這個視窗的處理數。下列對視窗的建立做更詳盡的解釋[4]：

1. 註冊視窗類別

每個視窗都是一個視窗類別(window class)的成員。視窗類別是定義在該 class 裡一群通用特性屬性變數的模組。當開發視窗應用程式時，必須註冊所有被使用去建立視窗的視窗類別。為了簡化建立視窗的過程，Window CE 提供了一些系統定義的視窗類別；因為 Window CE 自動地註冊了這些類別，可以用它們馬上新建視窗，這部份的程式宣告包含在 InitInstance 裡的 WNDCLASS wc 裡。表 5-1、5-2 和 5-3 分別定義了 WNDCLASS 的結構、其宣告方式和 WNDCLASS 之變數功能描述。

表 5-1 WNDCLASS 結構定義

```
Type struct_WNDCLASS{
    UNIT style;
    WNDPROC lpfnWndProc;
    int cbClsExtra;
    int cbWndExtra;
    HANDLE hInstance;
    HICON hIcon;
    HCURSOR hCursor;
    HBRUSH hbrBackground;
    LPCTSTR lpszMennuName;
    LPCTSTR lpszClassName;
}WNDCLASS;
```

表 5-2 Window class 宣告方式

```
WNDCLASS wc;
wc.style = 0;                                          //Window style
wc.lpfnWndProc = MainWndProc;                          //Callback function
wc.cbClsExtra = 0;                                     //Extra class data
wc.cbWndExtra = 0;                                     //Extra window data
wc.hInstance = hInstance;                              //owner handle
wc.hIcon = NULL;                                       //Application icon
wc.hCursor = LoadCursor(NULL, IDC_ARROW);   //Default cursor
wc.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
wc.lpszMenuName = NULL;                                //Menu name
wc.lpszClassName = szAppName;                          //Window class name
```

表 5-3 Window class 變數功能描述

變　　數	功　　能　　描　　述
style	設定視窗類別的風格(style)。可設定多個風格，使用 OR(｜)運算子來結合這些風格。例如(CS_DBLCLKS|CS_GLOBALCLASS)。
lpfnWndProc	設定一個指標指向視窗處理程序(window procedure)。
cbClsExtra	設定視窗類別結構(window-class structure)的額外位元組個數。
cbWndExtra	設定視窗例子(window instance)的額外位元組個數。
hInstance	設定視窗處理程序所在的例子(instance)。
hIcon	Windows CE 不支援，設定為 NULL。
hCursor	設定游標資源(cursor resource)的處理數。當設定為 NULL 時，應用程式必須設定當滑鼠移到應用程式的視窗時的形狀。
hbrBackground	設定背景的筆刷資源(brush resource)處理數。可以設定一個實際存在的筆刷或設定成一個顏色。若設為 NULL，則應用程式必須在每次需要重畫背景時指定筆刷。應用程式可使用 WM_ERASEBKGND 這個訊息來確認是否需要重畫背景，或是也可檢查 BeginPaint 函式回傳的 PAINTSTRUCT 結構的 fErase 成員來判定是否需要重畫背景。
lpszMenuName	指向一個 NULL 結尾的字串，用來設定選單(menu)資源的名稱，也可使用 MAKEINTRESOURCE 巨集來指定選單資源的識別字。如果設定為 NULL，即表示視窗沒有預設的選單。
lpszClassName	可以設定一個以 NULL 結尾的字串來指定視窗類別的名字，或是指定一個 atom。

2. 建立視窗

在多執行緒(thread)的應用程式裡，任何子視窗建立執行緒之前，主要的執行緒必須先建立主應用程式。我們會使用 CreateWindow 函式來建立新視窗。新視窗類別的類別名稱必須是萬國碼(Unicode)字串。我們可以使用名為 TEXT 的巨集(macro)來將一個字串表達或轉型成 unicode 的字串，或是使用_T 巨集，例如 TEXT(“keyboard painting”)，或是_T(“keyboard painting”)。表 5-4、5-5 和 5-6 分別為 CreateWindow 函式之結構定義、函式宣告方式和視窗屬性之描述。

表 5-4 CreateWindow 函式結構定義

```
HWND CreateWindow(
    LPCWSTR lpClassName,
    LPCWSTR lpWindowName,
    DWORD dwStyle,
    int X,
    int Y,
    int nWidth,
    int nHeight,
    HWND hwndParent,
    HMENU hMenu,
    HINSTANCE hInstance,
    LPVOID lpParam);
```

表 5-5 CreateWindow 函式宣告方式

```
HWND hWnd;
hWnd = CreateWindow(szAppName,                          //Window class
                    TEXT("Keyboard Painting"),          //Window title
                    //Style flags
                    WS_VISIBLE | WS_CAPTION | WS_SYSMENU,
                    CW_USEDEFAULT,                      //x position
                    CW_USEDEFAULT,                      //y position
                    CW_USEDEFAULT,                      //Initial width
                    CW_USEDEFAULT,                      //Initial height
                    NULL,                               //Parent
                    NULL,                               //Menu, must be null
                    hInstance,                          //Application instance
                    NULL);                              //Pointer to create parameters

    if (!IsWindow (hWnd)) return 0;                     //Fail code if not created.
```

表 5-6 CreateWindow 函式視窗屬性之描述

視窗屬性	功　　能　　描　　述
Class name	每個視窗都屬於某個視窗類別,除了像控制項這種系統內建的視窗類別外，其他的視窗類別都必須先註冊才能建立這類別的視窗。lpClassName 參數即使用來指定建立視窗時要使用何種視窗類別做用模板(template)。
Window name	使用 lpWindowName 參數來指定視窗的本文(text)，例如像視窗(window)、對話盒(dialog box)或訊息盒(message box)等，本文會顯示在抬頭(title)，按鈕(button)、編輯(edit)和 static 等的控制項(control)則是顯示在中央，其他像 list box，combo box 或捲軸等則不會顯示。
Style	使用 dwStyle 參數設定視窗的風格(style)，例如 WS_BORDER 設定視窗有邊框(border)。
Horizontal and vertical coordinates	設定視窗在螢幕的位置，包含 x 軸和 y 軸兩個參數，即視窗在螢幕的最左上角那點的座標。
Width and height coordinates	使用 nWidth 和 nHeight 參數設定視窗的寬和高。
Parent	使用 hwndParent 參數來設定視窗的父親。
Menu	Windows CE 不支援，設為 NULL。
Instance handle	使用 hInstance 參數來設定建立定此視窗的應用程式的例子。
Creation data	當視窗被建立時，視窗的視窗處理程序會收到 WM_CREATE 訊息。可以使用 lpParam 參數來設定伴隨 WM_CREATE 訊息所傳入的資料，通常都傳入一個指向某種結構(structure)的指標。

3. 顯示視窗

系統建立視窗後，系統不會自動地顯示主視窗，而是應用程式的 WinMain 函式使用 ShowWindow 函式顯示視窗。透過使用 ShowWindow 函式可啟動或關閉視窗的 WS_VISIBLE 屬性，可以控制視窗的能見度(visibility)。

4. 結束視窗

結束視窗的同時也自動地結束其子視窗。可以呼叫 DestroyWindow 函式結束視窗。DestroyWindow 函式會先發送給起始視窗，再發給其他所有子視窗同樣的 WM_DESTROY 訊息。一般而言，視窗處理程序會在接收到 WM_DESTROY 或 WM_CLOSE 的視窗訊息(window messages)時呼叫 DestroyWindow 函式來結束視窗。

有了以上基本概念，對視窗程式的大略操作應該可以比較了解。接著我們會介紹 WinMain 函式。它是視窗應用程式的程式執行進入點(entry point)，同時它也是視窗應用程式的主要執行緒(primary thread)。WinMain 主要會完成以下的任務：

1. 註冊視窗類別(registering a window class)：

   呼叫 RegisterClass 函式來為已定義好的主視窗向視窗管理員註冊。

2. 建立主視窗(create main window)：

   透過呼叫 CreateWindow 函式來建立主視窗。

3. 建立及進入訊息迴圈(create and enter message loop)：

   建立訊息迴圈，及進入訊息迴圈中等待對主視窗有意義且重要的事件(events)，並對有意義且重要的事件作出相對應的動作。

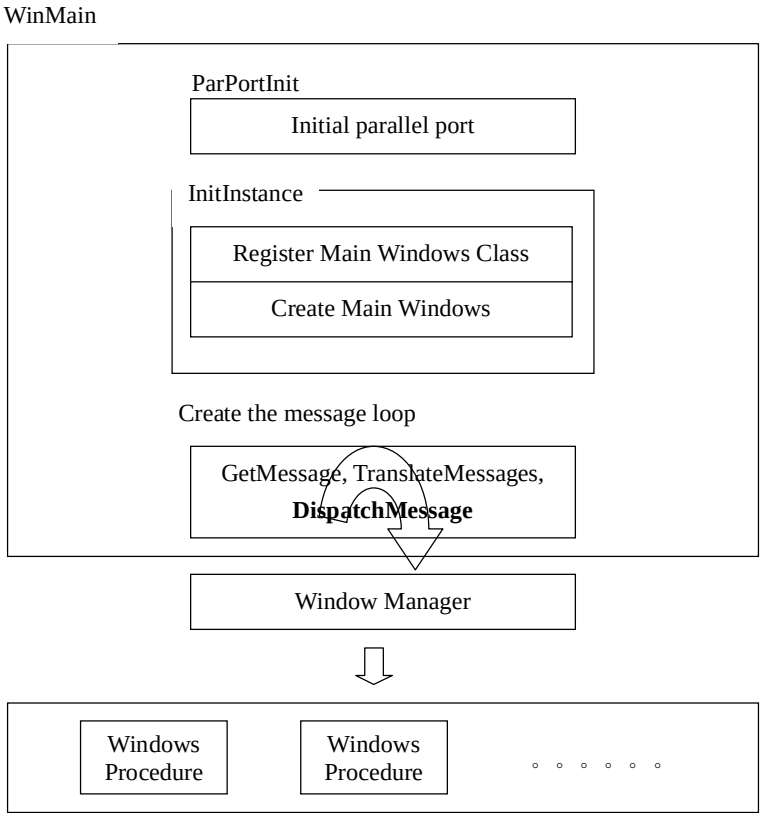

圖 5-3 WinMain 主要功能圖

圖 5-3 表 WinMain 主要功能。首先 WinMain 會進入 ParPortInit 副程式進行印表機埠的設定，接著利用 RegisterClass 函式來註冊視窗類別(Register Main Windows Class)。註冊完畢後，即可呼叫 CreateWindow 函式來建立主視窗(Create Main Windows)。接下來則進入到訊息迴圈裡等待事件(events)的發生而做出相對應的處理。

（二）訊息處理

WinMain 不需要親自去註冊一個視窗類別或者建立主要視窗，WinMain 可以呼叫其他已定義好的副程式去完成這些任務。但是，WinMain 必須自己

執行完成的一個任務便是建立訊息迴圈(message loop)。訊息迴圈會從一個執行緒訊息儲列(message queue)中取出(retrieve)訊息和發送(dispatch)訊息到適當視窗程序。同時會針對一特定執行緒的訊息列協調(coordinate)訊息的傳輸(transmission)。每個執行緒只能有一個訊息列，訊息被傳送給視窗時，訊息會被放在視窗執行緒的訊息列上，執行緒便從訊息列中接收並發送這個訊息。傳送訊息給視窗的執行緒的訊號列有兩種方式：貼出訊息(posting a message)和寄送訊息(sending message)。基本上，PostMessage 函式貼出訊息並立即返回，而 SendMessage 函式會等待來自傳送訊息的視窗的回應(當訊息的傳送端呼叫 SendMessage 的同時，將不會立即的產生呼叫返回值，直到接收端的執行緒完成此一訊息的處理並且會傳為止)。

Windows CE 是事件驅動的作業系統，所有的訊息都是使用一個名為 MSG 的結構來傳遞。MSG 結構含有六個欄位，下面表 5-7 顯示了這些欄位成員：

表 5-7 MSG 結構

成員	描述
hwnd	視窗處理數(window handle)
message	訊息代碼
wParam	額外的訊息資料
lParam	額外的訊息資料
time	指定訊息被投寄的時間
Pt	當訊息被投寄時，被動作所在的螢幕位置的座標

訊息處理主要包含了以下四種方式：

1. 接收和分配訊息(Receiving and Dispatching Messages)

   我們使用 GetMessage 函式來接收訊息。當應用程式呼叫 GetMessage 函式來接收訊息時，Windows CE 會到相關執行緒的訊息列(message queue)中取得訊息做相關處理，Window CE 會一直檢查訊息列中由 SendMessage 函式送出的訊息，並將這些訊息從訊息列中取出後配送給所有呼叫 GetMessage 函式在等待這些訊息的視窗處理序。這樣就能保證訊息接收列和訊息傳送列的同步。

2. WinMain 函式的訊息迴圈(message loop)

   先前提到 WinMain 主函式必須建立一個訊息迴圈來處理訊息，其中即是以呼叫 GetMessage 函式開始。當 GetMessage 函式接收到 WM_QUIT 訊息時會傳回 0，並使迴圈結束執行。另外，在 GetMessage 函式接收到訊息後，和呼叫 DispatchMessage 函式來分配訊息給相關的視窗處理程序前，可能需要對接收到的訊息做一些處理動作。例如呼叫 TranslateMessage、TranslateAccelerator 和 IsDialogMessage 等函式來做一些必要的處理動作，這些函式有時候會自己處理訊息的分派(dispatch)而不需要再呼叫 DispatchMessage 函式。通常會在呼叫 DispatchMessage 函式之前呼叫 TranslateMessage 函式，TranslateMessage 函式會確認鍵盤訊息(keyboard messages)所夾帶的

字元(characters)，並且張貼(post)這些字元到訊息儲列以便在下一次的迴圈時取出。另外，可使用 TranslateAccelerator 函式來解釋鍵盤訊息(keyboard messages)和產生選單命令(menu commands)，而 IsDialogMessage 函式可以用來確保 modeless 對話盒的動作正確無誤。不過在我們的程式片段裡只用了 TranslateMessage 此函式。

3. 傳送訊息(Sending Messages)

   我們可以呼叫 SendMessage 函式來傳送訊息到視窗。SendMessage 函式是一個同步(synchronous)函式，函式的呼叫會一直等到接收此訊息的視窗的視窗處理程序處理完這個送來的訊息後才會返回。

4. 張貼訊息(Posting Messages)

   相反地，PostMessage 函式是一個非同步(asynchronous)函式。呼叫 PostMessage 函式時，PostMessage 函式會送出訊息到訊息列後便馬上返回，而不會等到接收此訊息的視窗的視窗處理程序處理完這個送來的訊息後才返回。

瞭解了 Windows CE 應用程式的建置過程後，下一章我們將分析本系統的程式設計流程。