國立交通大學

電機學院通訊與網路科技產業研發碩士班

碩士論文

遠端桌面下建構多媒體串流服務

Multimedia Streaming Services On Remote Desktop

研究生：鄧如宏

指導教授：張文鐘　教授

中華民國九十八年八月

遠端桌面下建構多媒體串流服務
Multimedia Streaming Services On Remote Desktop

研 究 生：鄧如宏　　Student：Ju-Hung Teng

指導教授：張文鐘　　Advisor：Wen-Thong Chang

國 立 交 通 大 學

電機學院通訊與網路科技產業研發碩士班

碩 士 論 文

A Thesis

Submitted to College of Electrical and Computer Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Industrial Technology R & D Master Program on

Communication Engineering

August 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年八月

遠端桌面下建構多媒體串流服務

研究生：鄧如宏　　指導教授：張文鐘 博士

國立交通大學電機學院產業研發碩士班

摘要

本論文主要是利用串流技術，解決遠端桌面程式在播放影片，造成網路流量暴增的問題。當使用者點選影片播放，不直接在 Server 端電腦執行播放程式，而是由串流伺服器將影片串流。再由 Client 端接收串流資料並且播放影片。

我們採用遠端桌面程式 rdesktop 和多媒體播放器 MPlayer。由於 rdesktop 和 MPlayer 兩個程式的顯示視窗是獨立分開的，而使用者僅使用 rdesktop 來操作遠端電腦。所以必須透過 IPC 的方式，將 MPlayer 解碼後的影像資料傳給 rdesktop。rdesktop 在主視窗下建立一個子視窗，將影像資料顯示在上面。由於 Windows 遠端桌面服務程式是無法更改，所以透過外加自行撰寫的串流伺服器，完成影音控制和傳送的功能。最後將 Client 端的程式移植到以 Intel XScale PXA270 為核心的嵌入式平台。

Multimedia Streaming Services on Remote Desktop

Student：Ju-Hung Teng　　Advisor：Dr. Wen-Thong Chang

Industrial Technology R & D Master Program of
Electrical and Computer Engineering College
National Chiao Tung University

ABSTRACT

Our research uses streaming technology to solve the data burst during video playback on remote desktop. When the user clicks a video file, the server doesn't play the video file but passes the file to a streaming server. Then the client connects to the streaming server and plays the file.

We use a remote desktop client - rdesktop and a multimedia player - MPlayer. Rdesktop and MPlayer have their own display window, but the user only uses rdesktop to control the remote computer. So MPlayer passes the decoded frames to rdesktop through IPC. Rdesktop creates a child window and displays the decoded frames on it. Because that we can't modify Windows remote desktop service, we add our own streaming server to send the control command and transmit the streaming media. In the end, we port the client program to an Intel XScale PXA270 based embedded system.

致謝

碩士生涯這兩年，首先感謝的是我的指導教授張文鐘博士，感謝老師於學業上的教導，我才能夠順利完成碩士學位。同時也感謝范國清教授、黃仲陵教授及何文楨主任於口試時的指導，有了您的指導才使得這篇論文更趨完善。

另外，感謝實驗室同學們，志偉、盛如、弘達、琮壹、秉謙和建民，感謝你們陪我度過兩年碩士生涯，與你們一同修課研究的歲月裡是值得珍惜及懷念同時也感謝實驗室學弟妹，耀葦、明穎、怡如和雅嵐在研究上給予的支持，。

最重要的要感謝我的父母，有你們的支持，讓我在經濟上、生活上沒有任何負擔，有你們的陪伴，我才能無後顧之憂的完成學位。感謝我的女朋友姿萱，在我挫折時給我的鼓勵，在我失敗的時候給我支持。

最後，感謝這兩年曾給我協助、給我鼓勵的朋友們，謝謝你們。

誌於 2009.夏 新竹。交大

如宏

目錄

表目錄

圖目錄

第一章　緒論

1.1 研究背景

隨著電腦硬體技術和無線通訊技術的進步，行動上網裝置(Mobile Internet Decive)越來越普及。現代的人們可以透過行動上網裝置，隨時隨地的上網瀏覽網頁、收發E-mail、聊 MSN 或是分享影音資料。行動上網裝置的計算能力、儲存空間有限，可以透過遠端桌面程式，操控家中或是辦公室的電腦，將一切的資料處理和計算留在遠端電腦執行。行動上網裝置當成終端機，將鍵盤輸入、滑鼠操控訊息透過無線網路傳送到遠端的電腦，遠端的電腦再將處理完成的畫面結果傳回。

現今網路技術發達，加上多媒體影音壓縮的進步，透過網路分享影音資訊越來越普及。由於行動上網裝置的儲存空間有限，所以這些多媒體影音檔案大多存放在遠端的電腦。如果使用者在家中或是辦公室的電腦存放了許多的多媒體影音檔案，使用者最方便存取遠端電腦的檔案，就是透過遠端桌面程式，直接點選開啟。但是目前遠端桌面程式共同的問題－播放影片將造成網路流量暴增！

1.2 研究動機和目標

目前市面上有許多的遠端桌面程式，如 VNC、Microsoft 的 Remote Desktop Protocol(簡稱 RDP)、X Window…等。遠端桌面程式運作方式，是取得 Server 畫面的變動資料，然後將這些變動的畫面資料傳送給 Client。Client 將接收到的畫面資料更新到視窗上。對於一般操作模式，如文書處理和瀏覽網頁，並無太大問題，因為畫面更新的頻率和資料並不大。但是如果使用者點選影片播放，這個畫面更新率最高可能到每秒 30 個 frame。雖然 VNC 和 RDP 對於更新的畫面採用壓縮的方式傳送，但是那些壓縮方式僅對於單調的圖形有效。對於影片畫面劇烈變動的情況下，效果非常不理想。

表 1-1. VNC 和 RDP 播放影片的資料量

解析度	320*136	480*204	848*352
檔案大小(Kbytes)	5,691	16,353	43,141
位元深度(bpp)	24	24	24
影片長度(s)	135.18	135.18	135.18
更新率(fps)	23.976	23.976	23.976
資料率(kbit/s)	339.77	986.17	2610
VNC (Mbytes)	275.75	495.09	582.99
RDP (Mbytes)	269.23	609.52	889.27

表 1-1 為實際播放三種不同解析度的影片，分別為 320*136、480*204 和 848*352，在影片長度、網路環境和電腦硬體相同情況下所需要的頻寬大小。從表可以得知，隨著解析度的增加，所需要的頻寬越高。採用 RDP 播放解析度 848*352、長度 135.18 秒的影片，流量竟然高達 889.27Mbytes！平均每秒所需頻寬為 6.578Mbytes/s (約 53Mbps)，以目前家庭光纖網路頻寬下載 10Mbps/上傳 2Mbps，這頻寬是不能夠負荷。若是播放解析度 320*136，也是需要 1.992Mbytes/s (約 16Mbps)的速度。

表 1-2. VNC 和 RDP 播放影片的平均流量

解析度	320*136	480*204	848*352
VNC (Mbytes/s)	2.040	3.662	4.313
RDP (Mbytes/s)	1.992	4.509	6.578

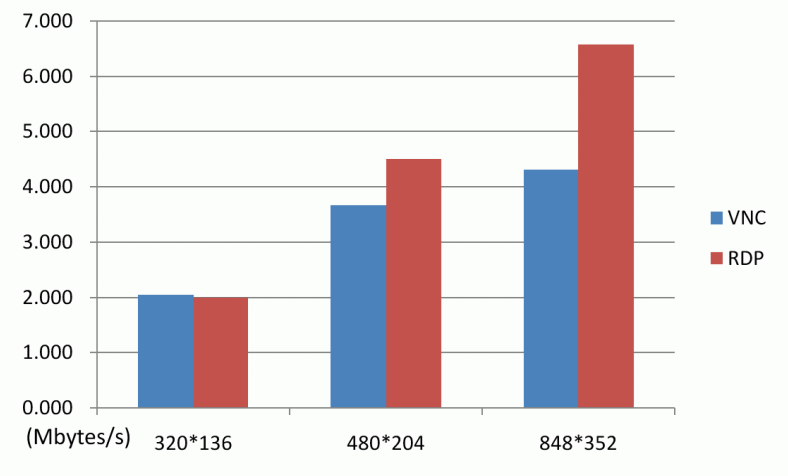

圖 1-1. VNC 和 RDP 播放影片的平均流量

使用 VNC 程式播放影片，頻寬需求比 RDP 少，但仍然很大。不論採用 RDP 或 VNC 播放影片，除了資料量暴增外，影片播放過程並不順暢。本研究主要目標是改善 RDP 遠端桌面程式播放影片的資料量暴增和順暢度的問題。最後將 Client 端程式移植到以 ARM 為核心的嵌入式平台。

1.3 研究方法和步驟

本研究提出的改善方法：利用影音串流方式，將 Server 的影片傳送到 Client，並由 Client 端的多媒體播放器播放影片。因此必須要先了解播放器的運作方式、遠端桌面程式視窗畫面運作和多媒體影音串流，如此才得以實現該系統。最後移植到以 ARM 為核心的嵌入式平台，這必須要了解將程式從 x86 移植到 ARM 的過程和方式。

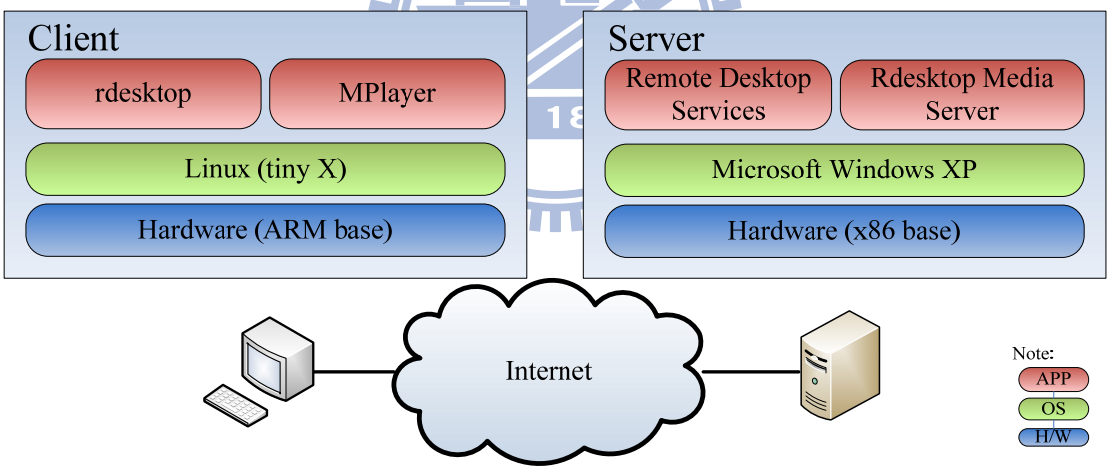

圖 1-2. 系統架構圖

圖 1-2 為本研究的系統架構，左邊為 Client 端，右邊為 Server 端，兩者以網際網路作溝通。在 Client 端採用 Linux 為作業系統，rdesktop 為 Remote Desktop Client，MPlayer 為多媒體播放器。在 Server 端，Remote Desktop Services 為 Microsoft Windows 的 RDP Server，Rdesktop Media Server 為影音串流伺服器。將細節約略分為以下步驟：

1.了解 MPlayer 播放器的程式架構：

- MPlayer 影像播放流程
- MPlayer 影像輸出模組

2.了解 rdesktop 的視窗運作：

- Linux x11 的視窗運作模式
- 利用 Named pipe 進行 IPC

3.系統實作

- 撰寫 MPlayer 影像輸出模組
- 新增播放器控制和影像顯示到 rdesktop 程式
- 撰寫影音串流伺服器
- 系統整合

4.程式移植

- 建置 ARM 平台的開發環境
- 編譯 ARM 用的程式庫
- 移植程式

1.4 論文架構

第三章將介紹 MPlayer 的開檔模組、影像濾波器和影像輸出模組。並且說明建立 video chain 流程，這將決定影像解碼器模組、影像濾波器和影像輸出模組。最後解說解碼後的圖像資料在模組間是如何的傳遞。第四章說明整個系統的實現，包括建立 MPlayer 的 rdp 影像輸出模組、修改 rdesktop 程式和撰寫 Server 端的串流伺服器。第五章分析 Progressive Download 串流方式如何利用 TCP 封包進行流量控制。最後比較 VNC、原本的 RDP 和本研究的 RMS 系統播放三種不同解析度影片所需要的網路流量。

第二章　相關知識及背景介紹

2.1 遠端桌面系統介紹

目前市面上有許多遠端桌面程式，在此就不全部介紹，僅選出較常用的三個來介紹，並淺要地分析他們的原理，和比較各項特點。

2.1.1 VNC

Virtual Network Computing[1](VNC)是現行的遠端桌面連線系統之一，最初發展是在 1995 年由英國劍橋的 Olivetti & Oracle 實驗室發展出來，公開且免費的通訊協定，至今已發展多種版本，為跨平台的遠端桌面連線程式。目前延伸的版本有 cotVNC、EchoVNC、RealVNC、TightVNC 和 UltraVNC。

VNC 透過 Remote Frame Buffer (RFB) Protocol 控制遠端的電腦，VNC 為一套跨平台的應用程式，在操作端安裝 VNC Viewer 就是我們所說的 Client，在遠端的電腦安裝 VNC Server，允許多台 Clients 在同一個時間連接同一個 Server，另外，也可透過 JAVA Applet 使用網路瀏覽器連接到安裝 VNC Server 的電腦。VNC 的架構可以分為 VNC Server 和 VNC Client(Viewer)，彼此溝通的通訊協定為 RFB Protocol。

- VNC Server：負責接收 Client 傳送過來的鍵盤或是滑鼠的輸入訊號後，並提供一套桌面分享機制，透過 TCP/IP 傳送畫面改變的資料到 Client。
- VNC Client：根據收到的資料，在將 Client 端顯示 Server 的畫面，並依照接收到的資料做更新。
- RFB Protocol：一個簡單的原則，將 Server 端某塊寬(W)、高(H)點陣圖(bitmap)的資料，放到螢幕指定的 X、Y 座標上。

2.1.2 RDP

Remote Desktop Protocol[2]，簡稱 RDP。RDP 是微軟根據 ITU T.120 協議系列所制定的一套未公開發表的數據通訊協議。透過網路連接 RDP Server 將應用程式顯示畫面傳送到 RDP Client，RDP Client 將滑鼠、鍵盤等輸入訊息傳送給 RDP Server。在畫面傳送上，是以命令(order)為操作方式，可以分為 primary order 和 secondary order，Primary order 主要的工作在於處理線條、矩形或是出現過的點陣圖和文字，Secondary order 則是傳遞首次出現的 bitmap 和文字。在終端服務的桌面圖形傳輸上，畫面中 bitmap 的傳送，可以說是主要的資料量。

RDP 的 bitmap 傳送方式有兩種方式：

1. RDP Server 先利用 secondary order 的 bitmap cache 將 bitmap 傳送到 RDP Client 暫存起來，隨後傳送一個 memory blt 將 cache 中 bitmap 資料，依據 cache id 和 cache index 取出指定的 bitmap，顯示在螢幕對應的位置上，主要畫面傳送都是利用這樣的方法。

2. RDP Server 直接傳送 bitmap 資料，並給予座標位置，RDP Client 收到後，直接將資料顯示在畫面上。傳遞的 bitmap 通常為視窗的外框或是工作列表邊緣線，因此，在畫面傳送上佔很小的比例，這部份的畫面更新方式稱為 bitmap updates。

2.1.3 X Window

X Window 系統[3](也常稱為 X11 或 X)是一種以點陣圖方式顯示的視窗系統。最初是 1984 年麻省理工學院的研究，之後變成 UNIX、Linux 等作業系統所一致適用的標準化軟體工具套件及顯示架構的運作協定，經過二十多年的演進，現今已成為工業標準。X Window 與一般的作業系統不同，在設計時就是以 Client/Server 為理念，整個 X Window 可以分為幾個部份：X Server、X Client、X Protocol 和 X Library。

- X Server：負責控制顯示卡將影像畫在顯示器上，並且管理鍵盤和滑鼠的事件，產生視窗、對應視窗及刪除視窗，這部份要特別說明，與一般的 Client/Server 名詞定義有點差異，X Server 定義為 Display Server，而應用程式端稱為 X Client。

- X Client：在 X Window 下的應用程式，要求特定的 X Server 作特定的動作。主要的工作為：1、向 X Server 提出需求，2、接收來自 X Server 的事件訊息，3、接收來自 X Server 的錯誤訊息。
- X Protocol：X Client 和 X Server 的通訊協定，定義 Requests、Reply、Error 和 Events，2.2.1 節有詳細說明。
- X Library：簡稱 X Lib，大部分 X window 上的應用程式以 X Library 來建立 GUI 元件，例如：按鈕(bottom)、目錄(menus)等等。

2.1.4 遠端桌面程式比較

根據先前介紹的遠端桌面系統，這節將對系統特性以及更新方式做個比較。

1. 畫面更新方式

Server 和 Client 之間的畫面更新方式可以分為以下幾種：

- RAW：所有的更新圖片，不經過壓縮編碼，每個 pixel 的資料依序傳送，可以想像資料量很大，假設傳送一張 640 x 480 大小的 16 位元高彩圖片，就需要 614.4K Bytes。VNC 有支援這樣的編碼方式機制。
- 2D draw primitives：利用區塊填色的方法將圖形編碼，通常是一種非失真編碼的方式，VNC 畫面編碼主要是以這樣的方法。
- Low level graphic：除了圖形編碼的方法之外，還利用一些簡單的指令，例如：字型、多邊形、線條等等的指令呼叫 Client 繪圖，RDP 便是利用這樣的更新方式進行畫面更新。
- High level graphic：除了 2D draw primitives 和 Low level graphic，還支援視窗的建立和管理，X Window 所利用的 X Protocol 就是屬於這類。

2. Server/Client 訊息傳遞方式

Server 和 Client 的訊息傳遞方式主要可以分為兩種，Server Push 就是由 Server 主動決定何時傳送更新畫面到 Client；Client Pull 就是由 Client 向 Server 要求傳送更新，使用 Server Push 的系統，其畫面更新較為平順但是相對資料量較大，而 Client Pull 則

是只有在使用者輸入訊息時，才會通知 Server 作畫面更新的傳送，但是其資料量較少。瞭解了畫面編碼方式及系統更新方式，根據這些特點我們將針對這幾個系統做分析比較，說明如表 2-1。

表 2-1. 遠端桌面系統比較

系統	畫面編碼方式	更新方式	壓縮方式	cache	license
VNC	Compressed Pixel Data	Client pull	2D draw primitives	Client frame buffer	GPL
RDP	Low level graphics	Server push	2D RLE	YES	proprietary
X window	High level graphics	Server push	none	NO	GPL

2.2 X Window 的 Client/Server 架構

2.2.1 X Window 的網路透明性

由於 X Window 系統採用網路訊息協定作為應用程式(X Client)和顯示介面(X Server)的溝通管道，而不是一般作業系統常見的函式呼叫，X Client 和 X Server 之間就是夠過可靠的位元組資料流作為溝通。一般來說，如果 X Client 和 X Server 在同一台電腦，就以內部程序通訊(IPC，InterProcess Communication)方式溝通;如果 X Client 和 X Server 在不同的機器上，利用程式介面 X Lib 透過可靠的網路協定(例如：TCP/IP)進行溝通。X Protocol 定義了四種封包做為溝通的機制。[9]

- 請求(Request)：客戶端請求伺服器的資訊，或者請求伺服器執行一個動作。
- 回應(Reply)：伺服器回應請求。但是並非所有的請求都會產生回覆。
- 事件(Event)：伺服器傳送事件給客戶端，例如，鍵盤或滑鼠的輸入，或移動、調整視窗。
- 錯誤(Error)：如果請求無效時，伺服器會傳送一個錯誤封包。

當 X Client 要向 X Server 發出要求(Request)時，可透過網路向執行的 X Server 發出要求，然後交給 X Server 處裡。X Server 向 X Client 通知事件(Event)時，也可以透過網路傳送。Request 和 Reply 的封包沒有長度限制，而 Event 和 Error 則有固定 32 bytes 的長度限制。網路透明化的性質，就是不同機器上的應用程式可以在同一台電腦上顯示結果，這樣的特性乃是 X Window 不同於其他系統的一大特點。

2.2.2 X Window 的視窗建立及繪圖功能

瞭解了 X window 的架構及網路透明性，我們的重點還是視窗的建立和影像畫面的繪製與傳遞，關於 X Window 的視窗建立和繪圖功能，X Window 提供了一些原則：

- X Server 將螢幕規劃為可重疊的視窗階層(window hierarchy)，每個應用程式可以使用多個視窗，依實際需要，重新規劃大小、位置。每個 GUI 中，視窗就是一個頂層視窗(Top Level Window)，除了根視窗(Root Window)以外，每個視窗都有一個 Parent Window，也就是說，Client 建立一個視窗就是在現存的視窗下建立一個子視窗，所以 Client 所建立的視窗都是以樹狀結構去建立的，樹狀結構的根就是 root window。
- X Server 的繪圖是立即性的，X Server 並沒有儲存繪圖序列的動作，而是收到資料後立即繪出畫面。X Window 繪圖是以位元對映(bit-mapped)導向，所有繪圖動作均以視窗定位，因此 X Client 可以在視窗內畫出東西，而不用考慮視窗在哪個地方。
- X Window 顯示圖片(images)，由於影像的資料量很大，X Lib 提供了一個 XImage 結構，允許使用者操作影像。

X Client 和 X Server 之間的交談，X Server 存放了資源(Resources)提供 X Client 使用識別碼(identifiers)操作，以下對各個資源做說明。

- 視窗(window)：這項資源是工作站上的矩型面積，使用者可利用視窗來觀看輸出結果，當應用程式產生需要顯示的圖形時，必須指定一個視窗來輸出。每個視窗都必須有一個根視窗(root window)，整個螢幕就是一個根視窗。

- 繪圖環境(graphical contexts,GC)：這部份是用來追蹤圖形的屬性，例如：前景顏色、背景顏色、線條寬度、字型等等。每一個圖形輸出的請求必須參照 GC，X Window 提供 X Client 請求產生、改變或是刪除一個 GC。
- 字型(fonts)：這項資源包括了在視窗上顯示文字有關的資料，描述一組文字的大小及樣式，這部份 X Server 通常有字型可供選擇，X Client 說明字元大小、字元空間、字體，利用編碼(ASCII、ISO Latin-1 等)由 X Server 顯示。
- 顏色對映表(color maps)：這項資源將應用程式所輸出的圖形資料轉換成視窗畫面上可見的顏色。
- 圖素對映表(pixmaps)：這樣資源和視窗很相似，主要的差別在於使用者無法在畫面上看到，對於 X Client 和 X Server 之間複製資料相當有用。

當 X Client 向 X Server 請求建立某個資源的時候，同時也指定了一個識別碼給特定的資源，例如：建立一個視窗的時候，X Client 指定了視窗的屬性和識別碼，這個識別碼便和這個視窗關聯。X Client 為了避免衝突，這些資源不可有相同的識別碼，資源建立後，識別碼就成為 X Client 和 X Server 溝通的特定識別。當建立資源的 X Client 關閉和 X Server 的連線後，資源就會正常解除(destroy)。

這些繪圖的功能與處理程序都交由 X Library 來實現，X Lib 是建立於 C 語言的函式，Client/Server 通訊協定都仰賴這個函式庫來完成，現行有些 X Window 高階的程式語言(例如：GTK+)，就是建構在 X Lib 函式庫的基礎上。

建立視窗的基本流程如下：

```
1.初始化流程(宣告結構、抓取環境變數)
Display* display;
char *display_name=getenv("DISPLAY");
2.連線到 X Server
display=XOpenDisplay(display_name);
3.X 相關初始化(設定視窗大小)
screen_num=DefaultScreen(display);
win=XCreateSimpleWindow(display, RootWindow(display, screen_num), x, y, width,
```

```
height, border_width, border, background)
4.開啟視窗
XMapWindow(display, win);
5.當未結束
    /*程式…*/
6.關閉視窗
XDestroyWindow(display, win);
7.關閉 X Server 的連線
XCloseDisplay(display);
8.執行清除函式
```

建立 XImage 結構並且將影像結合到視窗上的流程如下：

```
1.宣告變數
XImage *myximage;
2.讀取環境變數
int depth=DefaultDepth(display,screen);
Visual *visual=DefaultVisual(display,screen);
3.建立 XImage 並且宣告圖像記憶體空間
myimage=XCreateImage(display, visual, depth, format, offset, data, width, height,
bitmap_pad, bytes_per_line);
myximage->data=malloc(myximage->bytes_per_line * myximage->height);
4.將 XImage 顯示於視窗上
XPutImage(display, window, myximage, source X, source Y, destination X, destination
Y, width, height);
XSync(display, False);
```

擷取鍵盤滑鼠訊息流程如下：

```
1.設定擷取的訊息
int input= KeyPressMask | KeyReleaseMask | ButtonPressMask | ButtonReleaseMask;
2.註冊擷取訊息的視窗
XSelectInput(display, window, input);
3.處理訊息
XEvent an_event;
while(1){
    XNextEvent(display, &an_event);
    switch (an_event.type) {
```

```
            case: KeyPress:
                  /*處理鍵盤按鍵訊息*/
                  break;
            case: ButtonPress:
                  /*處理滑鼠按鍵訊息*/
                  break;
            default:
                  break;
      }
}
```

2.2.3 XImage 和 mp_image_t 結構

在 X11 操作影像的函式都使用 XImage 結構，在此結構中所支援的操作包括建立影像 XCreateImage()、刪除影像 XDestoryImage()、取得圖素 XGetPixel()、儲存圖素 XPutPixel()、取得影像中的次影像 XGetSubImage()和加一常數於影像中 XAddPixel()。XImage 資料結構的內容已定義完整，結構成員如圖 2-1。結構成員存放圖像的基本資訊，圖像的寬度、高度、色彩深度、每個像素所佔用的位元數和紅綠藍的元素遮罩。成員有一個指標指向圖像資料，如圖 2-2。

```
typedef struct XImage {
  int width, height;     /* size of image */
  int xoffset;           /* number of pixels offset in X direction */
  int format;            /* XYBitmap, XYPixmap, ZPixmap */
  char *data;            /* 指向圖像資料的指標 */
  int byte_order;        /* 位元組順序 LSBFirst, MSBFirst */
  int bitmap_unit;       /* quant. of scanline 8, 16, 32 */
  int bitmap_bit_order;     /* LSBFirst, MSBFirst */
  int bitmap_pad;           /* 8, 16, 32 either XY or ZPixmap */
  int depth;                /* 色彩 深度 */
  int bytes_per_line;       /* 一 行所佔用的位元組 */
  int bits_per_pixel;       /* 每 個像素所佔用的位元數 (ZPixmap) */
  unsigned long red_mask;   /* 紅色元素遮罩 */
  unsigned long green_mask;/* 綠色元素遮罩 */
  unsigned long blue_mask; /* 藍色元素遮罩 */
  XPointer obdata;   /* hook for the object routines to hang on */
  struct funcs {    /* image manipulation routines */
    XImage *(*create_image)();
    int (*destroy_image)();
    unsigned long (*get_pixel)();
    int (*put_pixel)();
    XImage *(*sub_image)();
    int (*add_pixel)();
  } f;
} XImage;
```

圖 2-1. XImage 結構成員

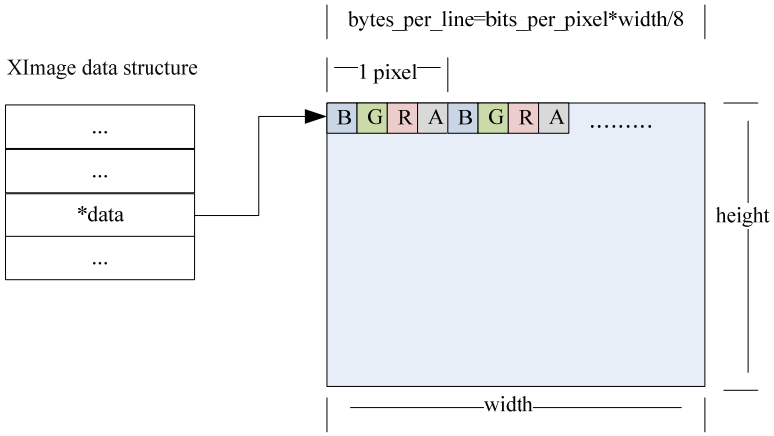

圖 2-2. XImage data 指向的圖像資料

mp_image_t 結構(簡稱 mpi)是 MPlayer 專屬存放圖像資料的結構，結構成員如 2-3。MPlayer 的模組間傳遞圖像資料都是用這個結構。成員有三個指標分別指向 YUV 三個圖像資料，如圖 2-4。XImage 和 mp_image_t 結構的最大差別就是前者是存放 packed RGB 格式，只需要一個指標指向圖像資料；後者是存放 planar YUV 格式，需要三個指標分別指向不同分量的資料。

```
typedef struct mp_image_s {
    unsigned short flags; // 圖像爲平面或打包格式
    unsigned char type;
    unsigned char bpp;      // 每個像素所需要的位元 並非 深度
    unsigned int imgfmt;    // 圖像格式
    int width,height;       // 圖像的寬度高度
    int x,y,w,h;            // 可見的寬度高度位置
    unsigned char* planes[MP_MAX_PLANES]; // 圖像資料的位址
    int stride[MP_MAX_PLANES];            // 圖像寬所佔用的位元組
    char * qscale;
    int qstride;
    int pict_type;          // 0->未知, 1->I, 2->P, 3->B
    int fields;
    int qscale_type;        // 0->mpeg1/4/h263, 1->mpeg2
    int num_planes;
    /* these are only used by planar formats Y,U(Cb),V(Cr) */
    int chroma_width;       // 色度的寬
    int chroma_height;      // 色度的高
    int chroma_x_shift;     // 色度水平位移
    int chroma_y_shift;     // 色度垂直位移
    /* for private use by filter or vo driver (to store buffer id or dmpi) */
    void* priv;
} mp_image_t;
```

圖 2-3. mp_image_t 結構成員

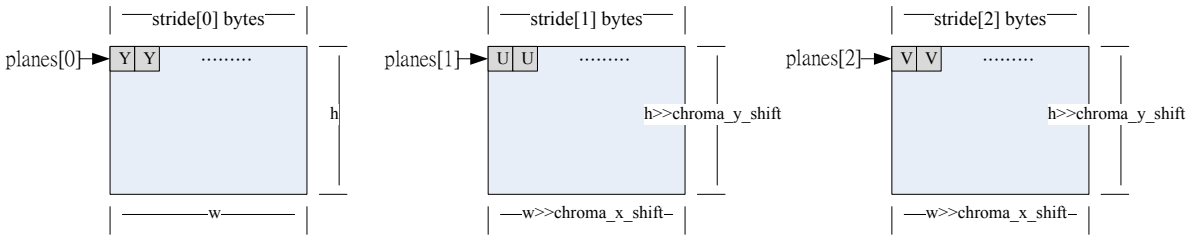

圖 2-4. mp_image_t plane[]指向的圖像資料

2.3 網際網路影音分享

近年來網路的蓬勃發展以及影音壓縮技術的進步，透過網際網路分享影音資訊已經非常普遍。在 56K Modem 時代影音分享大都是 Download and Play，將整個影片下載完，才進行播放。到了 Cable modem 和 ADSL 時代，下載速率的提升，符合影音串流服務的頻寬。而影音串流是將一連串的影像聲音壓縮，在網際網路上將資料分段傳送，以達到即時影像觀賞的服務。接下來介紹三種透過網際網路分享影音資訊的方法 True Streaming、Progressive Download 和 Download and Play。前兩種屬於串流技術，第三種僅是單純的下載播放。[4]

2.3.1 True Streaming

利用 RTSP 進行串流服務，RTSP 是種 stateful 的 protocol。從 client 第一次建立連線到最後和 server 中斷連線，這中間的時間 server 都要追蹤 client 的狀態。client 利用 PLAY、PAUSE 和 TEARDOWN 的命令和 server 告知目前狀態。client 和 server 建立程序後，server 才開始傳送 RTP 的影音封包。RTP 封包可以使用 UDP 或是 TCP 傳送。UDP 是非連結導向的傳輸，封包可能遺失，但是傳輸快；而 TCP 是連結導向的傳輸，封包遺失有重新傳送的機制，是種可靠傳輸。可以因應需求選擇 UDP 或是 TCP，如有些防火牆會濾除 UDP 封包，此時就會採用 TCP 的方式傳送。

server 即時的在傳送 RTP 封包給 client，傳送速率和影片的編碼率相同。例如編碼

率為 384kps 的影片，傳送速度大約是 384kps。client 緩衝器的資料約 1~10 秒的影片。當播放暫停 5 分鐘，僅需下載當時 1~10 秒的影片。

2.3.2 Progressive Download

Progressive Download 是介於 True Streaming 和 Download and Play 之間的方式，主要是採用 HTTP protocol。HTTP protocol 是種 stateless 的 protocol，client 和 server 要求資料，server 就回應所要求的資料，server 並不會知道 client 的狀態。每個 HTTP 的連線都是獨立的程序。當播放器從 server 獲得足夠的影片資訊和資料即開始播放影片，使用者不需要等待整個影片下載完成。在播放的同時，影片資料持續的下載到播放器的快取，而播放器再從快取取得影片資料播放。Progressive Download 和 True Streaming 有兩個主要差別。第一點，server 並非即時的在傳送檔案，當播放器快取的資料用完，而下載速度卻低於影片編碼率，這時候會造成畫面停格。使用者只能等待更多的資料下載完才可以繼續播放。第二點，最後完整的影片會被下載到使用者電腦。

2.3.3 Download and Play

將整個影片檔案下載到使用者的硬碟，再進行播放。使用者可以獲得高品質和順暢的播放效果，但是換來的是先前的等待影片下載完成。下載到使用者硬碟，這意味著無法防止非法複製，對於影片著作權有極大傷害。另外一點就是服務提供者的頻寬成本比較高，當使用者只觀賞 20%的影片，可是服務提供者卻支付了整個檔案的頻寬。

2.4 多媒體播放器介紹

目前市面上有許多播放器，在此列出比較常用的三種，VLC Media Player、MPlayer 和 Windows Media Player，並淺要描述它們的特點。

2.4.1 VLC Media Player

VLC[5]多媒體播放器是由 VideoLAN 計劃撰寫的免費開放原始碼的播放器。VLC 包含播放器、編碼器和串流器，並支援許多音訊和視訊解碼器和檔案格式。它可以透過網路將檔案串流或是將多媒體檔案轉成其他格式儲存。VLC 是 VideoLAN Client 的縮寫，但這原始意義已經不適用了。VLC 軟體許可是在 GPL 下。VLC 支援跨平台，提供 Microsoft Windows、Mac OS X、Linux、BSD 和 Solaris 的版本。

VLC 包含了許多免費的解碼和編碼程式庫。VLC 的許多 codec 是由 FFmpeg 計劃的 libavcodec 程式庫提供。libdvdcss 程式庫用來播放加密的 DVD 影片。

圖 2-5. VLC 0.9.9 (Windows)

- 支援許多影音格式，包含 MPEG-1、MPEG-2、MPEG-4、DivX、DVD/VCD、數位衛星頻道。
- 支援 RTSP、RTP、MMS 和 HTTP 協定。
- 預覽 eMule 或 BitTorrent 下載未完成的影片。
- 透過 IPv4 或 IPv6 的寬頻網路作為 unicast 或 multicast 的串流伺服器。
- 提供 telnet 或是 HTTP 操作。
- 可以錄製桌面畫面。

2.4.2 MPlayer

MPlayer[6]是個免費開放原始碼的播放器。這跨平台軟體，支援 Linux、Unix-like 系統、Microsoft Windows 和 Mac OS X。MPlayer 提供許多影音格式(MPEG-1, MPEG-2,

MPEG-4, RealVideo,WMV, AAC, AC3, MP3)，也可以將串流資料存成檔案。

MPlayer 是個系統命令模式的程式，但是有隨作業系統提供不同 GUI 版本。如 gMplayer (Unix-like 系統)、MPlayer OS X (Mac OS X)和 MPUI (Windows)。

圖 2-6. gMplayer (Linux)

- 支援多種驅動程式，VDPAU、X11、OpenGL、DirectX、Quartz、VESA 和 Framefbuffer。
- tv://channel 可以顯示影像擷取卡的電視畫面。
- radio://channel 可以播放廣播。
- 支援 RTP、RTSP、HTTP、FTP、MMS 和 SMB 協定。

2.4.3 Windows Media Player

Windows Media Player[7] (簡寫 WMP)是由 Microsoft 發展的多媒體播放器。WMP 執行在 Microsoft 作業系統底下，用來播放影片音樂和瀏覽圖片。WMP 可以將音樂 CD 複製到硬碟、燒錄音樂 CD 或是和攜帶型音樂設備同步音樂。

圖 2-7. Windows Media Player 11(Windows)

- 支援本地播放、播放串流和漸進式下載。
- 支援使用 DirectShow 濾波器的 codec。
- 多媒體管理，提供搜尋和分類功能。
- Video Smoothing 功能，對於低 frame-rate 影片使用內差法增加 frame 數。
- 提供 ActiveX 嵌入網頁，開發者可以在網頁上播放多媒體檔案。
- 提供其他平台 Windows Mobile, Mac OS, Mac OS X, Palm-size PC, Handheld PC 和 Solaris。

2.4.4 播放器比較

VLC Media Player 和 MPlayer 都有使用 FFmpeg 的函式庫。Windows Media Player 主要是用 DirectShow。比較表如表 2-2。

表 2-2. 播放器比較

	Author	Cost	Software license	Based Framework	Written in
VLC Media Player	VideoLAN	Free	GPL	FFmpeg + original	C, C++

MPlayer	Árpád Gereöffy	Free	GPL	FFmpeg + original	C99
Windows Media Player	Microsoft	Free	Proprietary	DirectShow	C++(COM)

2.5 利用 Named Pipes 進行 IPC

Named pipes 可以讓兩個不相關的程序(process)互相通訊，亦稱做 FIFO(first-in, first out)是用來做單向的資料傳輸。目前 Windows、Unix 和 Linux 作業系統都支援 named pipes，但本節就 Unix 和 Linux 作業系統的操作進行說明。Named pipes 和一般檔案一樣有檔案名稱和路徑，都定義在存取端的檔案系統中。兩個不相關的程序可以開啟定義為 Named pipe 的檔案並且開始通訊。Named pipe 的檔案不會隨著程序結束而消失，需要透過文字命令模式刪除。

使用 named pipe 的方法，一個程序開啟 named pipe 的檔案為寫入狀態，另外一個程序開啟為讀取狀態。Named pipe 的使用預設為 blocking 模式，也可以利用 O_NONBLOCK 旗標設定成 non-blocking 模式。一個程序只能開啟為讀取或是寫入狀態，不能同時開啟讀取和寫入，這是因為 named pipe 為半雙工通訊(Half-Duplex Communication)。如果要進行全雙工通訊(Full-Duplex Communication)，就要使用兩個 named pipe。圖 2-8 為兩個程序使用單一 named pipe 進行半雙工通訊；圖 2-9 為兩個程序使用兩個 named pipe 進行全雙工通訊。因為 named pipe 讀寫是預設為 blocking 模式，所以進行全雙工通訊要避免死結(deadlock)產生。[8]

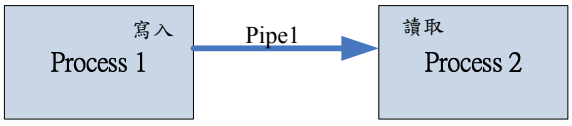

圖 2-8. 半雙工通訊

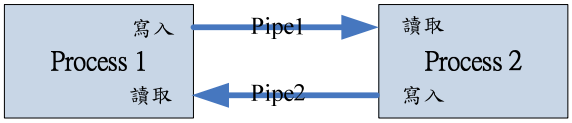

圖 2-9. 全雙工通訊

2.5.1 建立 Named Pipe

建立 named pipe 方式有兩種，一種是在系統命令模式建立，另一種是在程式呼叫 mkfifo()函式。假設我們要在當前目錄底下建立一個名為 pipe 的 named pipe 檔案，圖 2-10 為使用 mkfifo 或是 mknod 指令的範例。如果要在程式內建立，就要採用圖 2-11 的方式。

```
# mkfifo pipe      或      # mknod pipe p
```

圖 2-10. 文字命令模式建立 Named pipe

```
mkfifo("pipe", 0666)
```

圖 2-11. 程式內建立 Named pipe

2.5.2 開啟 Named Pipe

開啟方式和開啟一般檔案一樣，可以使用 open()或是 fopen()。使用 open()會回傳檔案描述，而使用 fopen()會得到 FILE 結構的指標。從使用者觀點，建立了 named pipe 後，操作方式和處理一般檔案一樣，可以開啟、讀取、寫入和刪除。

2.5.3 使用 Named Pipe

Named pipe 不能同時開啟為讀取或寫入模式，只能選擇其中一個模式。使用標準 C 的函式庫 read()或 write()，可以用來讀取或寫入 named pipe 的檔案。這些操作都是預設為 blocking 模式，當一個程序從 named pipe 讀取資料，但是卻沒有資料在裡面，此時會被 block 住。另一方面如果要寫入而沒有另一個程序讀取資料，也是會被 block。Named pipe 的檔案不能使用搜索(seek)的操作模式。

2.5.4 Named Pipe 的優缺點

1.Named Pipe 的優點

- 可以當成一般檔案操作。
- mkfifo()是個 thread-safe 的函式。
- 使用時不需要任何同步機制。
- write()到 named pipe 保證為 atomic，開啟為 non-blocking 亦是如此。
- 有操作權限限制，可以強化安全性。

2.Named Pipe 的缺點

- 只可以在同一台電腦上操作。
- 只可以建立在本地端的檔案系統，不能建立在 NFS(Network File System)。
- 因為 blocking 的機制，雙向通訊時要避免產生死結。
- Named pipe 是以位元組為單位的資料流，沒有任何資料被儲存。

2.6 嵌入式系統

嵌入式系統(Embedded System)是特地用途的微電腦系統，即將微電腦使用在特定系統中。例如數位相機、掌上型電視遊樂器和雷射印表機，在這類的應用中使用一般的微處理器當作中央處理器，並配合適當的記憶體和周邊裝置，完成所需要的系統。在這個系統中微處理器只做特定的工作，而微處理器搭配的周邊裝置，所構成的一個完整系統，稱為嵌入式系統。

嵌入式系統是以應用層面為設計的中心，以電腦技術為基礎，並且軟硬體可以依不同的需求做調整。該系統對功能、可靠性、成本、體積、功率消耗有嚴格要求。嵌入式系統和個人電腦有著本質上的區別，嵌入式系統本身的成本、適用性和可靠性和個人電腦不同。個人電腦設計上的要求是更好的處理效能和更快的處理速度。但是這不能運用於設計嵌入式系統，很多產品的成功不在於複雜的系統是否可以運作，而是否有優越的

性能價格比。

嵌入式系統受限於功能和環境，對外部事件需要在規定時間內反應，如汽車的安全氣囊。對於體積、重量的限制，如掌上型電視遊樂器。對於功率消耗和散熱必須符合環境要求，如 MID 裝置。

表 2-3 和表 2-4 為嵌入式系統和個人電腦的比較。[11]

表 2-3. 嵌入式系統和個人電腦硬體比較

設備名稱	嵌入式系統	個人電腦
CPU	ARM、MIPS	x86(Intel、AMD)
RAM	SDRAM	SDRAM、DDR
Storage	Flash	硬碟
Input	Touch Screen、按鍵	KeyBoard、Mouse
Output	LCD	顯示器
Sound	音效晶片	音效卡
Other	USB	主機板提供或擴充卡

表 2-4. 嵌入式系統和個人電腦軟體比較

	嵌入式系統	個人電腦
開機程式	Bootloader 引導，需要對不同開發板進行移植	主機板 BIOS 引導，不需更改
作業系統	WinCE、Linux、VxWork…等，需要移植	Windows、Linux…等，不需要移植
驅動程式	針對設備重新開發或是移植	作業系統內建或是由第三方提供
開發環境	交叉編譯器	本機開發
模擬器	需要	不需要

2.6.1 ARM

ARM 公司(Advanced RISC Machines Limited，簡稱為 ARM Limited)成立於 1990 年。1985 年 4 月 26 日，第一個 ARM 原型在英國劍橋的 Acorn 電腦有限公司誕生。目前 ARM 架構處理器已在高性能、低功耗、低成本的嵌入式應用領域佔有領先地位。

2.6.2 Linux

Linux 是一種基於 Linux 核心的類似 Unix 的作業系統。Linux 是自由軟體和開放原始碼最著名的例子。任何人遵守 GPL 的協議，都可以免費下載、重新更改或是發佈。Linux 主要用於伺服器，但是基於現今不同的電腦硬體，Linux 支援個人電腦、嵌入式裝置、手機甚至超級電腦，這些廣泛的電腦硬體包括 x86 、Power PC、ARM、Alpha、SRARC 等處理器。

Linux 核心是由 Linux Torvalds 在 1991 年所開發的。目前已經發展出許多 Linux 套件，最有名的是 Ubuntu。Linux 套件除了 Linux 核心還包含其他軟體如 Apache 網頁伺服器、X Windows 系統、KDE 桌面環境、GNOME 桌面環境或是 OpenOffice。

Linux 是個 Unix-like 的作業系統，重要特點如下。[10]

1.多程序(Multitasking)：

實現了優先權排程管理。當同時間有高優先權和低優先權程序，高優先權程序可以優先執行。

2.多使用者(Multi-user)：

提供了分時(time-sharing)多工系統，可以讓許多使用者同時使用電腦，同時提供資料隱私保護。

3.多處理器(Multi-processing)：

支援多處理器(Symmetric Multi-Processing)。

3.記憶體保護(Protected Memory)：

每個程序擁有自己的記憶體空間，而且不能直接存取其他程序的記憶體空間。這防止不同程序間相互破壞對方的記憶體資料。

4.檔案系統(File System)：

提供根目錄檔案系統，每個目錄和檔案都是從根目錄延伸，像是樹狀組織。除此之外還有提供兩項特別的功能如下。

(1)連結 (Links)－連結指向真正檔案的位置，而本身不是一個檔案。

(2)獨立裝置 I/O (Device-Independent I/O)－周邊的硬體裝置在 Linux 底下都是

一個檔案，對於程式而言寫入資料檔案和寫入資料到印表機是相同的。

如果要將 Linux 用於嵌入式環境，必須依需求做移植和修改。嵌入式 Linux 有以下的特點。

1.開放原始碼：

程式設計者可以針對 Linux 原始碼進行修改，根據實際應用對核心優化。

2.低成本：

基於 GPL 協議下，可以免費下載程式碼並對 Linux 核心做更改和重新發佈。大多數的嵌入式 Linux 開發工具也是遵守 GPL 協議，同樣也是可以免費獲得。如此可以節省大量的開發費用。

3.眾多軟體支援：

Linux 是個一完整功能強大的作業系統，提供各式各樣的應用軟體。利用 Linux 提供的軟體支援，可以迅速建構嵌入式應用的軟體環境。

2.7 開發環境

首先必須了解開發嵌入式系統所用到的兩種系統，一種是電腦主機(Host Computer)，我們在上面撰寫並偵錯程式，因為許多開發用的軟硬體工具支援通用型處理器，此處理器是桌上型電腦的一部份；另一種為目標系統(Target System)，此系統為開發中的嵌入式目標平台，我們在這個平台的處理器上執行我們撰寫的程式。

- 電腦主機(Host Computer)：標準(x86)平台用來開發目標系統的程式和軟體，並且透過 JTAG 線連結目標系統除錯。
- 目標系統(Target System)：開發中的嵌入式系統。
- 交叉開發(Cross-development)：建立在 host computer 的開發環境，利用交叉編譯器編譯目標系統的程式。

撰寫嵌入式系統的程式與撰寫在桌上型電腦執行的程式非常類似。一般而言，桌上型電腦上的應用程式設計流程首要步驟為撰寫原始碼，原始碼可能分成數個檔案。原始碼以編輯程式撰寫，再以編譯器(compiler)或組譯器(assembler)將各檔案的程式碼加以編譯或組譯，產生對應的二元程式碼檔案，再利用連結器(linker)將二元程式碼結合為執行檔，此為實作階段。透過偵錯程式測試該執行檔，為驗證階段。如果在驗證階段程式出現錯誤或是效能低落，回到實作階段改進原始碼，再編譯組譯。如此反覆，直到達到預期成效。

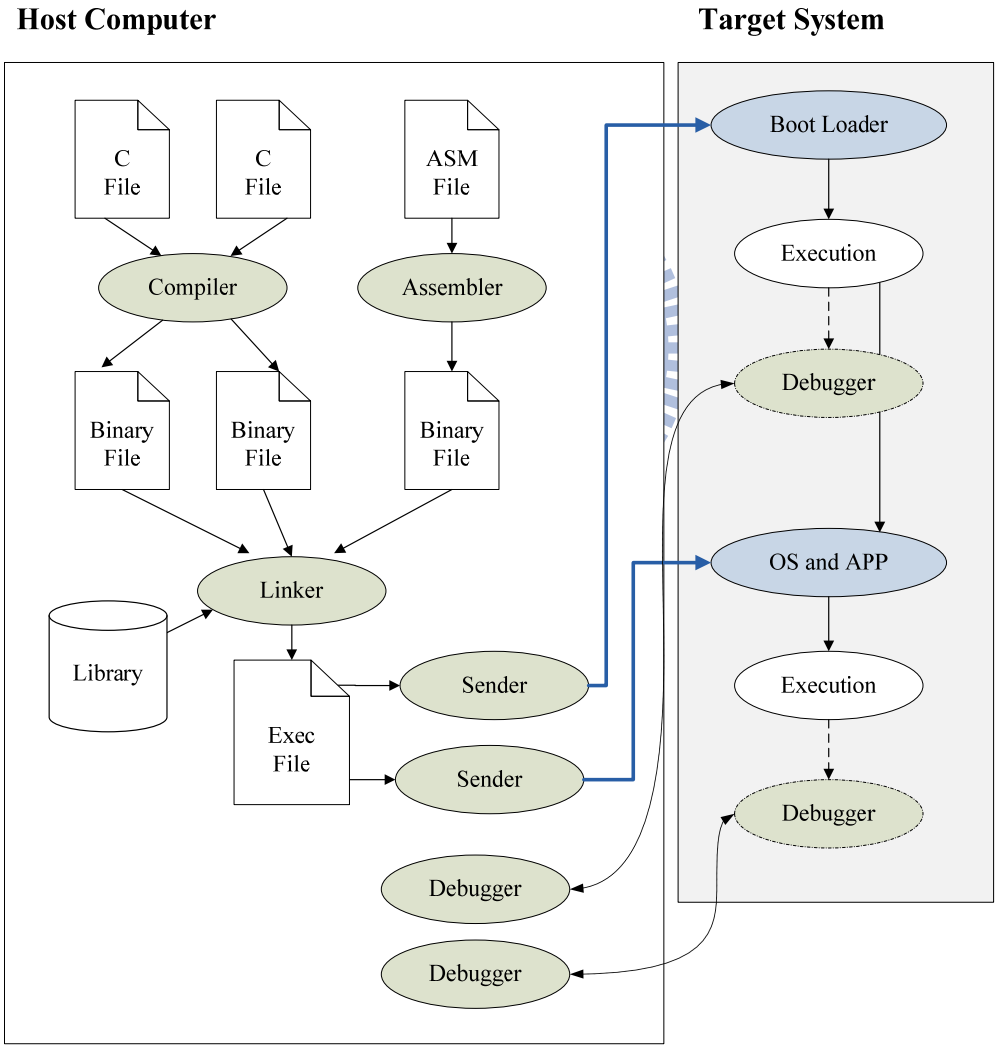

圖 2-12. 軟體開發流程

圖 2-12 為嵌入式系統軟體的開發流程，開發嵌入式系統的程式包括編輯、編譯、

組譯和連結。我們在開發用的電腦主機上使用交叉開發(cross-development)工具，包含交叉編譯器(cross complier)和交叉組譯器(cross assembler)來進行編譯和組譯工作。完成的目標執行檔，我們透過 JTAG、RS-232c 或是網路傳到目標系統上執行。驗證階段的工作會和開發桌上型電腦的程式有許多的不同。

偵錯和除錯需要透過額外的硬體電路或是連接器，將開發用的電腦主機和目標系統連接起來。如此在偵錯程式可以透過單步執行，觀察目標系統的處理器的記憶體位置和暫存器的值。這樣可以方便程式設計者加以除錯和改進程式。

2.7.1 硬體連接

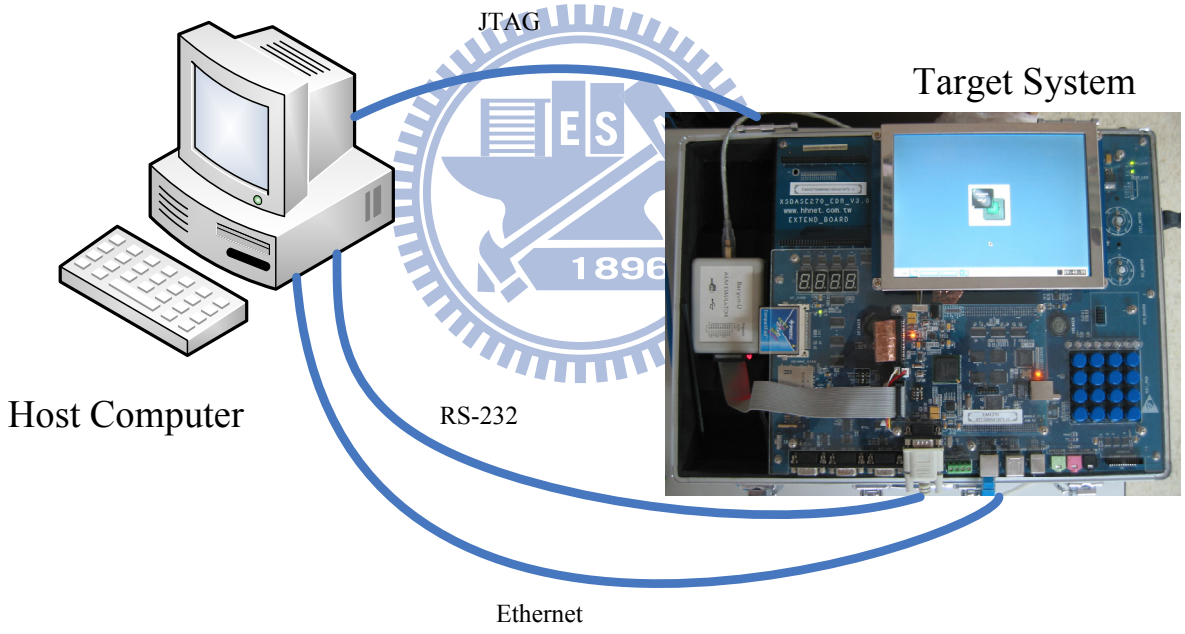

圖 2-13. 開發環境硬體連接

圖 2-13 為開發的時侯電腦主機和目標系統的硬體連接，透過這些連接線可以操控或是上傳檔案到目標系統。

- JTAG：用來偵錯或除錯，也可以用來燒寫目標系統的 Flash，一般是用來燒寫 Bootloader。

- RS-232：在超級終端機操作目標系統。開機後，bootloader 就會透過 RS-232 傳送文字選單，並且可以由使用者下達指令。進入 Linux 後將會顯示開機訊息，輸入登入帳號密碼就可以用命令模式操作整個目標系統。
- Ethernet：在 bootloader 的選單時，更新 Linux kernel 或是 Root Filesystem 都是透過 ethernet 傳送。

2.7.2 軟體架構

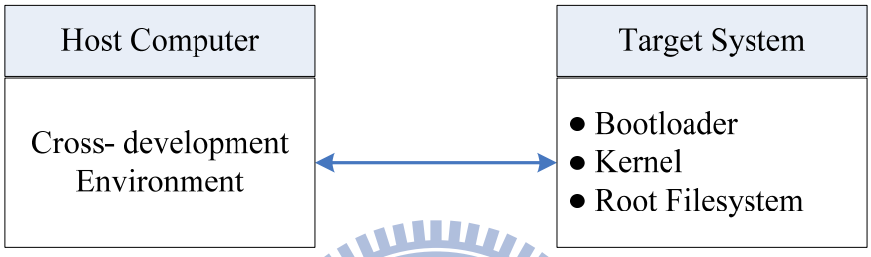

圖 2-14. 開發環境軟體架構

圖 2-14 為開發環境的軟體架構，目標系統通常含有三個部份，分別為 Bootloader、Kernel 和 Root Filesystem。

- Bootloader：

Bootloader 是在作業系統執行之前的一段小程式。透過這段小程式，我們可以初始化硬體設備、建立記憶體空間的映射表，從而建立適當的系統軟硬體環境，為最終呼叫作業系統做好準備

對於嵌入式系統，Bootloader 是建立在特定硬體平台來實現的。因此，不可能為所有的嵌入式系統建立一個通用的 Bootloader。Bootloader 依賴 CPU 的架構和嵌入式平台的周邊設備，所以每當開發新的平台，Bootloader 需要做適當的修改和移植。

- Kernel：

Kernel 是作業系統的核心，主要是進行硬體和軟體之間的溝通和資源的管理。硬體資源包含處理器、記憶體和輸入輸出裝置。硬體資源是有限的，核心必須決定

程式對硬體操作的時間。直接操作硬體是非常複雜的，所以核心提供一個抽象介面，可以讓程式更簡潔的方式操控硬體。

- Root Filesystem：

除了 kernel 之外，filesystem 是讓作業系統能順利運作的重要元素，它儲存作業系統在運作過程中所需要的程式和資料。

Root filesystem 是構成 filesystem 的最小集合，它包含所有 Linux 開機時需要的檔案及資料夾，如 initrd、init.d 裡的各項服務，/etc、/proc、/lib 等。Filesystem 的源頭為根目錄(/)，所有檔案和目錄都是從根目錄延伸。在嵌入式系統中， root filesystem 包含了一些基本使用工具，如查詢檔案的 ls、掛載裝置的 mount 等。若是內建的儲存空間不足，可以透過 CF 卡或是 SD/MMC 卡擴充儲存空間。

2.8 開發板

本研究採用華亨科技的 EELIOD 開發板，如圖 2-15，此開發板分兩個部份 XSBase270-Module 和 XSBase270-EDR。[12]

- XSBase270-Module：採用 Intel Xscale PXA270 處理器的開發平台，可以獨立使用。
- XSBase270-EDR：需與 XSBase270-Module 配套使用的介面擴充板，透過 2 個 120pin 排針與 XSBase270-Module 板連接一起工作，不可獨立使用。。

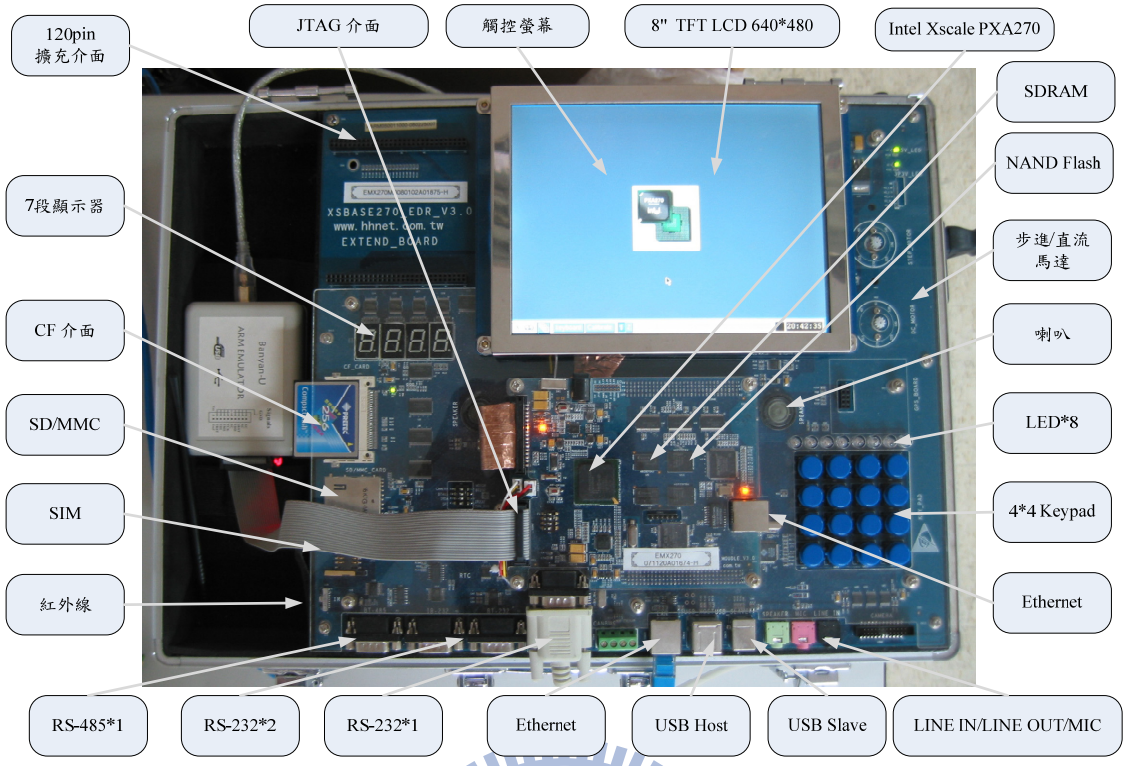

圖 2-15 華亨科技 EELIOD

表 2-5. XSBase270-Module 硬體配置

XSBase270-Module	
處理器	Intel XScale PXA270 520MHz
RAM	SDRAM 64MB
FLASH	NAND Flash 32MB
電源管理	LP3971
乙太網路	LAN91C113
音效晶片	UCB1400BE
液晶螢幕	Sharp 8 吋 TFT 640*480
觸控螢幕	8 吋 四線式觸控 UCB1400BE 控制
RS232	1
JTAG 介面	20pin
擴充介面	2 個 120pin

表 2-6. XSBase270-EDR 硬體配置

XSBase270-EDR	
紅外線	1
RTC	RTC4513

CF	接 CF 卡或是 802.11b 無線 CF 網卡
MMC/SD	1
SIM	1
乙太網路介面	1 (和 Module 共用)
串列埠	1 個 RS485、2 個 RS232
音效介面	LINE IN/LINE OUT/MIC
USB Host	2
USB Client	1
CMOS 介面	1
LED	8
七段顯示器	4
鍵盤	4*4
CAN Bus 介面	1
喇叭	2
步進馬達	1
直流馬達	1
擴充介面	120pin 可接 FPGA 板

表 2-7. EELIOD 軟體配置

Linux	
核心	Linux Kernel 2.4.21
系統引導程式	51Board Bootloader
檔案系統	JFFS2
GUI	Tiny-x

第三章　MPlayer 程式

MPlayer 的基本播放架構如圖 3-1 所示，使用者透過系統指令模式輸入開啟影音資料的來源、指定的 video/audio filter 和指定的 video/audio ouput。MPlayer 透過 Open Stream 函式選擇適當的開檔模組去讀取資料，並且放到 Cache2 緩衝器。然後透過合適的解多工器將資料分離出 video bitstream 和 audio bitstream。再利用合適的視訊解碼器和音訊解碼器分別對 video bitstream 和 audio bitstream 解碼成 video frame 和 audio frame。解碼完成後，透過使用者選定的濾波器分別 video frame 和 audio frame 做處理。最後再藉由 video output 和 audio output 將影像和聲音分別輸出到螢幕和喇叭。

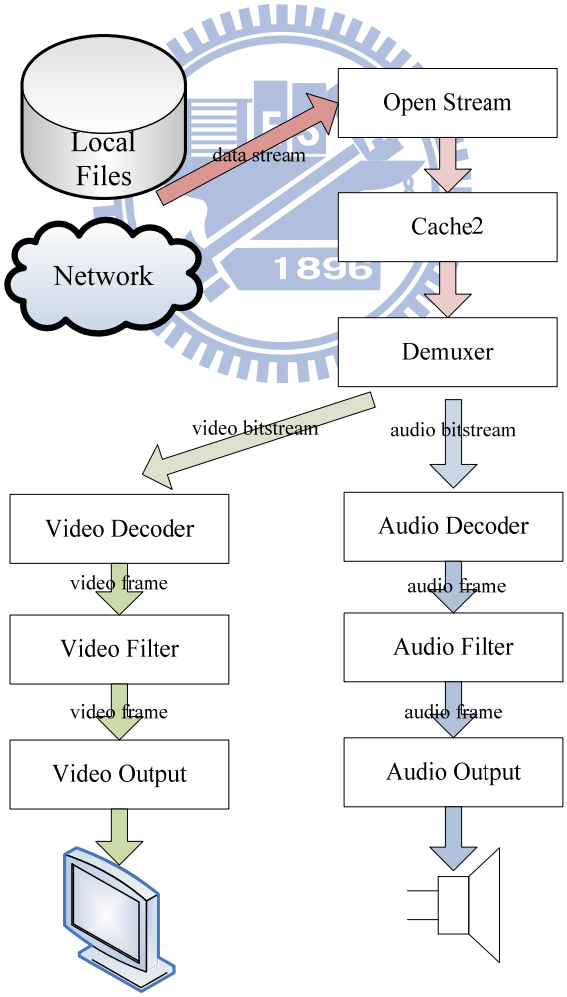

圖 3-1. 播放流程基本架構

這章將介紹 MPlayer 的 Open Stream、Cache2、Video Filter 和 Video Output。並且說明 MPlayer 建立 video chain 的流程，還有解碼完的一張張 frame 是怎麼通過 Video Filter 和 Video Output。

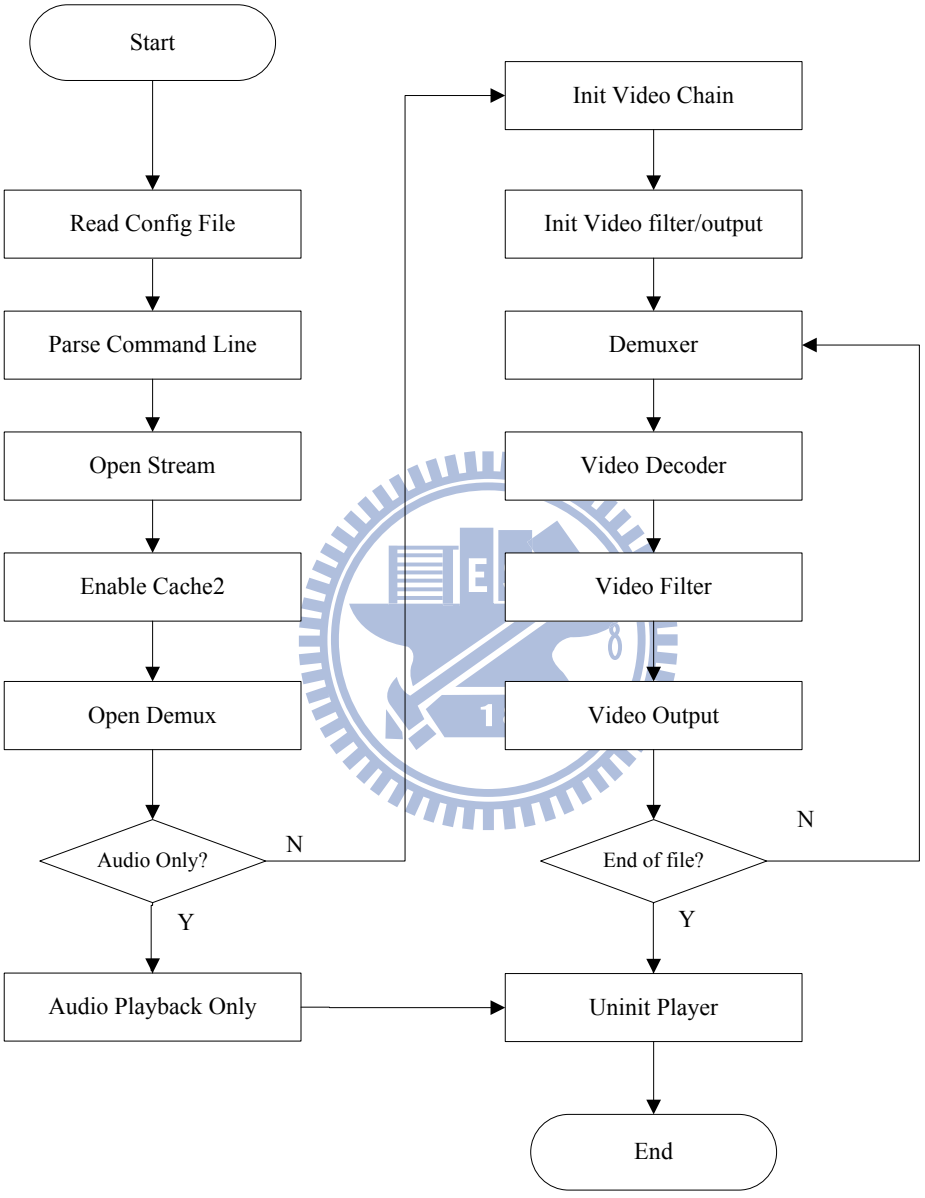

圖 3-2. MPlayer 播影像的基本流程

圖 3-2 為 MPlayer 播放影像的基本流程。3.1 節將說明 MPlayer 如何透過 open_stream() 函式開啟檔案。開啟檔案後會依需求開啟 Cache2，這是儲存影音資料的緩衝器。3.2 節說明 Cache2 的啟動機制。3.3 節介紹影像濾波器(Video Filter)，3.4 節介紹影像輸出模組

(Video Output)。影音資料經過解多工器分成視訊資料和音訊資料。MPlayer 在播放前需要建立 video chain，這包含決定影音解碼器、影音濾波器和影音輸出模組。3.5 節說明 video chain 建立的流程。3.6 節說明 video chain 中各模組的初始化和影像資料的傳遞。

3.1 Open Stream 函式

MPlayer 開啟檔案是採用 Open Stream 函式。開啟本地硬碟或光碟機的資料，或是透過 HTTP、RTSP、FTP 的 protocol 從網路開啟遠端檔案，都是使用 Open Stream 函式選擇合適的開檔模組。一個模組可以支援一種或多種的 protocol，如 RTSP 模組只支援 RTSP，而 HTTP 模組支援 HTTP、MMS、RTSP。目前 MPlayer 1.0rc2-4.1.2 包含 HTTP、RTSP、RTP、UDP、FTP、VCD、DVD…等。

表 3-1 為 stream_info_t 的結構成員，每個模組都要有這個結構，結構成員存放該模組對應的參數和模組開檔的函式指標。

表 3-1. stream_info_t 結構成員

變數	型態
info	const char *
name	const char *
author	const char *
comment	const char *
open	函式指標
protocols[MAX]	char *
opts	void *
opts_url	int

info：基本資訊。

name：模組名稱。

author：作者名字。

comment：建議事項。

open：指向模組開啟函式的指標，這個函式是模組的進入點。

protocols[]：型態為字串陣列，裡面存放模組所支援的 protocol。選擇模組會比對這裡面的字串，來判斷支援與否。

opts_url：如果為 1，會將檔案路徑當成選項字串解析。

圖 3-3 為 HTTP 的 stream_info_t 結構資料，最重要的是 protocols[]和 open()對應的參數。

```
stream_info_t stream_info_http1 = {
  "http streaming",                                          //info
  "null",                                                    //name
  "Bertrand, Albeau, Reimar Doeffinger, Arpi?",              //author
  "plain http",                                              //comment
  open_s1,                                                   //open()
  {"http", "http_proxy", "unsv", NULL},                      //protocols[]
  NULL,                                                      //opts
  0                                                          //opts_url
};
```

圖 3-3. HTTP 的 stream_info_t 結構資料

當 MPlayer 選定開檔模組後，利用 stream_t 結構來呼叫和使用模組內的函式和變數。而 stream_t 的部份結構資料會在 open()模組程式進入點指定值。表 3-2 為 stream_t 結構所包含的成員。這裡將這些結構成員的函式和變數，做詳加說明與介紹。

表 3-2. stream_t 結構成員

變數	型態	回傳型態
fill_buffer()	函式指標	int
write_buffer()	函式指標	int
seek()	函式指標	int
control()	函式指標	int
close()	函式指標	void
fd	int	-
type	int	-
flags	int	-
sector_size	int	-

buf_pos,buf_len	unsigned int	-
pos,start_pos,end_pos	off_t	-
eof	int	-
mode	int	-
cache_pid	unsigned int	-
cache_data	void *	-
priv	void *	-
url	char *	-
streaming_ctrl	streaming_ctrl_t *	-
buffer[SECTOR_SIZE]	unsigned char	-

fill_buffer ()：將 stream 的資料讀取到緩衝器。

write_buffer()：將緩衝器的資料寫到 stream。

seek()：搜索 stream 裡面的資料。

control()：控制介面。呼叫此函式需要傳入一個控制類別的旗標。該函式會針對被呼叫的類別，執行該段的程式碼。

close()：關閉 stream。

fd：開啟檔案的標號或是 TCP/UDP 開啟的標號。

type：stream 的類別，種類有 FILE、VCD、STREAM、DVD、CDDA…等。

sector_size：2048 或是 2324，2324 為 VCD 播放用。搜索和 Cache2 的大小會是 sector_size 的整數倍。

buf_pos,buf_len：buffer[SECTOR_SIZE]資料所存放的位置和大小。

pos,start_pos,end_pos：。

eof：所開啟的檔案是否結束(End Of File)。

mode： STREAM_READ 或是 STREAM_WRITE 模式。

cache_pid：當 Cache2 啟動後，cache2.c::cache_fill()子程序的 PID。

cache_data：指向 Cache2 所用 cache_vars_t 結構的指標。

streaming_ctrl：streaming_ctrl_t 結構，用於網路。

buffer[SECTOR_SIZE**]**：大小為 2048 或是 2324bytes。

圖 3-4 為 VCD 模組 open_s 程式進入點的部分程式碼。stream_t 部分結構資料在此指定，有函式操作的 fill_buffer、seek、close 函式指標，fd、type、sector_size、start_pos、end_pos 和 priv 變數資料。每個模組會依需求指定結構資料，並非所有變數都有使用。

```
static int open_s(stream_t *stream,int mode, void* opts, int* file_format) {
  struct stream_priv_s* p = (struct stream_priv_s*)opts;
  int ret,ret2,f,sect,tmp;
  mp_vcd_priv_t* vcd;

  ...

  stream->fd = f;
  stream->type = STREAMTYPE_VCD;
  stream->sector_size = VCD_SECTOR_DATA;
  stream->start_pos=ret;
  stream->end_pos=ret2;
  stream->priv = vcd;

  stream->fill_buffer = fill_buffer;
  stream->seek = seek;
  stream->close = close_s;
  *file_format = DEMUXER_TYPE_MPEG_PS;

  m_struct_free(&stream_opts,opts);
  return STREAM_OK;
}
```

指定 stream_t 的結構成員，這依開檔模組而有不同的指定參數

圖 3-4. VCD 模組 open_s()部分程式碼

streaming_ctrl_t 結構是專門用來處理網路資料，為 stream_t 結構成員之一，HTTP、RSTP、RTP 都有用到這個結構。streaming_ctrl_t 結構資料是在 MPlayer 選定開檔模組後指定值。表 3-3 為 streaming_ctrl_t 結構所包含的成員。這裡將這些結構成員的函式和變數，做詳加說明與介紹。

表 3-3. streaming_ctrl_t 結構成員

變數	型態
url	URL_t *
status	streaming_status
buffering	int
prebuffer_size	unsigned int
buffer	char *
buffer_size	unsigned int
buffer_pos	unsigned int
bandwidth	unsigned int

streaming_read	函式指標
streaming_seek	函式指標
data	void *

url：URL_t 的結構指標，其結構成員如表 3-4。這是存放解析網址變數，假設網址為 url=http://user:pass@140.113.13.88:6666/Video.avi 則解析出來的變數為 protocol=http、hostname=140.113.13.88、file=/Video.avi、port=6666、username=user 、password=pass。

表 3-4. URL_t 結構成員

變數	型態
url	char *
protocol	char *
hostname	char *
file	char *
port	unsigned int
username	char *
password	char *

status：目前狀態為 streaming_playing_e 或是 streaming_stopped_e。

buffering：開啟 Cache2 的旗標，設定為 1 代表開啟。

prebuffer_size：預先緩衝的資料大小，單位為 byte。

buffer：緩衝器，在 HTTP 模組是用來存放回傳的 body 資料。

buffer_size：緩衝器的大小。

buffer_pos：緩衝器的位置。

bandwidth：網路頻寬。

streaming_read：指到對應模組的讀取檔案的函式。

streaming_seek：指到對應模組的搜索檔案的函式。

data：用來存放任意型態的指標，如 HTTP 用來存放 HTTP_header_t 結構指標。

3.1.1 Open Stream 函式運作流程

圖 3-5 為 Open Stream 開啟的流程。當 main()要開啟檔案時，會呼叫 open_stream()函式並且傳入檔案路徑。open_stream()只負責檢查傳入的檔案路徑是否為 NULL，如果為 NULL 返回錯誤訊息，否則繼續呼叫 open_stream_full()。open_stream_full()是利用比對模組支援的 protocol 和檔案路徑的字串，來挑選適合的開檔模組。open_stream_plugin()是做後續的初始化和解析檔案路徑。當整個 Open Stream 函式開啟成功，會回傳一個 stream_t 結構的指標，這個 stream_t 的結構會由 main()傳給 demux_open()函式進行開啟解多工器。

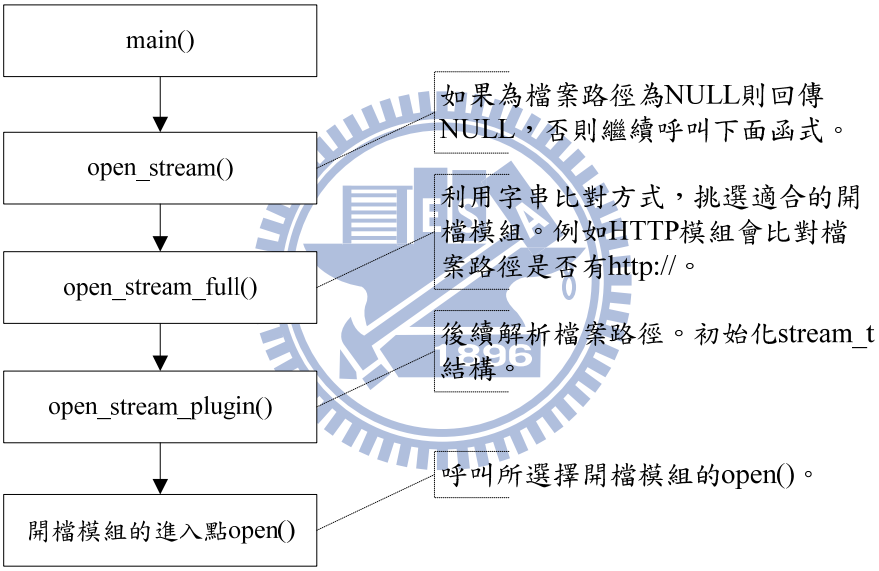

圖 3-5. Open Steam 流程圖

open()是利用函式指標指到 MPlayer 所選擇開檔模組的程式進入點，名稱可能有所不同，例如 HTTP 為 open_s1()，VCD 為 open_s()。open()函式，將會指定 stream_t 部份結構成員的資料，最重要的是指定操作函式指標到該模組的操作函式，如此可以用 stream_t 的結構直接操作該模組的函式。

```
stream_t* open_stream_full(char* filename,int mode, char** options, int* file_format) {
  int i,j,l,r;
  stream_info_t* sinfo;
  stream_t* s;
1 for(i = 0 ; auto_open_streams[i] ; i++) {
    sinfo = auto_open_streams[i];
    if(!sinfo->protocols) {
      mp_msg(MSGT_OPEN,MSGL_WARN, "Stream type %s has protocols == NULL, it's a bug\n", sinfo->name);
      continue;
    }
    for(j = 0 ; sinfo->protocols[j] ; j++) {
      l = strlen(sinfo->protocols[j]);
2     if((l == 0 && !strstr(filename, "://")) || ((strncmp(sinfo->protocols[j],filename,l) == 0)
                                                 && (strncmp("://",filename+l,3) == 0))) {
        *file_format = DEMUXER_TYPE_UNKNOWN;
3       s = open_stream_plugin(sinfo,filename,mode,options,file_format,&r);
        if(s) return s;
        if(r != STREAM_UNSUPPORTED) {
          mp_msg(MSGT_OPEN,MSGL_ERR, MSGTR_FailedToOpen,filename);
          return NULL;
        }
        break;
      }
    }
  }

  mp_msg(MSGT_OPEN,MSGL_ERR, "No stream found to handle url %s\n",filename);
  return NULL;
}
```

圖 3-6. open_stream_full()的程式碼

圖 3-6 為 open_stream_full()程式碼，這邊重點分三個部份解釋如下。

第 1 部分：auto_open_streams[]是 stream_info_t 的結構陣列，存放所有開檔模組的 stream_info_t 結構，利用 for 迴圈逐一讀取每個模組的 stream_info_t 結構，由第 2 部分比對。

第 2 部分：利用字串比對的方式，比對檔案路徑是否包含模組支援的 protocol 字串，這 protocol 字串是 stream_info_t 的成員變數。例如 HTTP 模組支援 http 和 rtsp 的 protocol，那就會比對檔案路徑開頭是否有“http://”和“rtsp://”的字串。如果都比對失敗，那回到第 1 部分讀取下一個模組的 stream_info_t 結構。如果最後都比對不出任何 protocol 字串，將由 FILE 開檔模組開啟檔案。MPlayer 1.0rc2-4.1.2 內建的開檔模組如表 3-5。

第 3 部分：將檔案名稱和指定的模組交由 open_stream_plugin()處理，open_stream_plugin()會做一些初始化，最後呼叫模組的函式 open()。這時候模組做開檔動作，進行模組初始化並且把 stream_t 結構的變數資料指派完成。當整個 open_stream_full()開啟成功，回傳 stream_t 結構的指標。

表 3-5. MPlayer 內建的開檔模組

開檔模組	protocols
stream_info_vcd	"vcd",
stream_info_cdda	"cdda", "cddb",
stream_info_netstream	"mpst",
stream_info_http1	"http", "http_proxy", "unsv",
stream_info_asf	"mms", "mmsu", "mmst", "http", "http_proxy", "mmshttp",
stream_info_pnm	"pnm",
stream_info_rtsp	"rtsp",
stream_info_sdp	"sdp",
stream_info_rtsp_sip	"rtsp", "sip",
stream_info_rtp	"rtp",
stream_info_udp	"udp",
stream_info_http2	"http", "http_proxy", "pnm", "mms", "mmsu", "mmst", "rtsp",
stream_info_dvb	"dvb",
stream_info_tv	"tv",
stream_info_radio	"radio",
stream_info_pvr	"pvr",
stream_info_ftp	"ftp",
stream_info_vstream	"tivo",
stream_info_smb	"smb",
stream_info_cue	"cue",
stream_info_dvd	"dvd",
stream_info_dvdnav	"dvdnav",
stream_info_null	"null",
stream_info_mf	"mf",
stream_info_file	"file", "",

3.1.2 HTTP 模組開檔流程

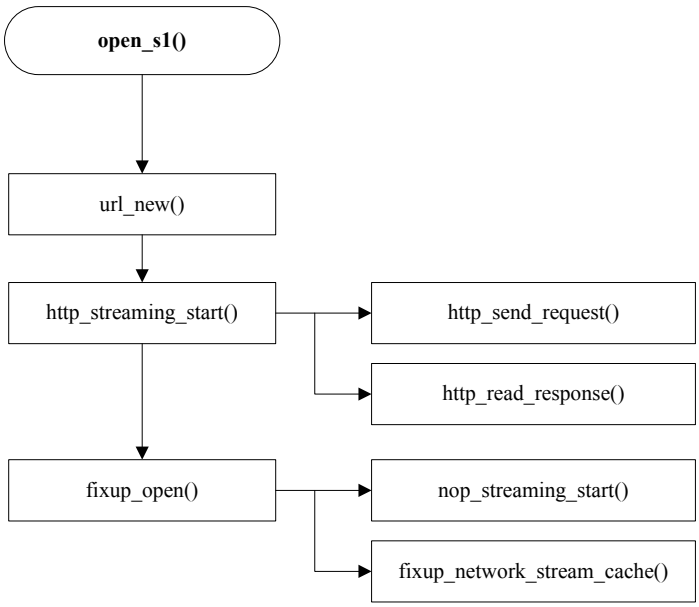

圖 3-7. HTTP 模組開檔流程

open_s1()是 HTTP 模組程式進入點。url_new()將網址分解成 URL_t 的結構格式，這表 3-3 有詳細說明。http_streaming_start()是重要的函式，包含了 http_send_request()和 http_read_response()。http_send_request()先建立 HTTP 請求訊息。與 Web Server 建立 TCP 連線後，發送 HTTP 請求訊息給 Web Server。http_read_response()負責接收 Web Server 回覆的資料，主要處理標頭部份，而不下載檔案資料(body)。http_send_request()和 http_read_response()處理完後，http_streaming_start()會去解析回覆資料。

```
GET /Video HTTP/1.0
Host: 140.113.13.81:667
User-Agent: MPlayer/1.0rc2-4.1.2
Icy-MetaData: 1
Connection: close
```

圖 3-8. HTTP 請求訊息

圖 3-8 為 HTTP 模組所發送的 HTTP 請求訊息。第一行是 GET 指令請求檔案為根目錄底下的 Video 檔案，Client 支援 HTTP 1.0。第二行是 Web Server 的位址和埠號。第三

行是發送請求的程式是 MPlayer1.0rc2-4.1.2。第四行是 ICY 的資料。第五行是說明這連線在檔案傳送完畢就立即斷線。

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Server: Rdesktop Media Server
Content-Type: application/octet-stream
Content-Length: 44176206
```

圖 3-9. Web Server 回覆訊息

圖 3-9 為 Web Server 回覆的訊息。第一行是 Server 支援 HTTP 1.1，代碼 200 為請求檔案隨回應傳送。第二行是判斷 Server 是否支援中斷點下載，在 Open Stream 中代表檔案來源是否可以快轉或倒退，也就是搜索的功能。回應 bytes 代表支援，none 則否。第三行是 Server 名稱。第四行是檔案的 MIME 類型，這會決定 MPlayer 預先設定的解多工器(demuxer)模組，application/octet-stream 是設定為 DEMUXER_TYPE_UNKNOWN，這會讓 MPlayer 自行選擇解多工器模組。第五行為檔案的大小，單位為 bytes。這邊最重要就三個欄位 Accept-Ranges、Content-Type 和 Content-Length。

表 3-6. MPlayer MIME 表

MIME 類型	
audio/mpeg	DEMUXER_TYPE_AUDIO
video/mpeg	DEMUXER_TYPE_UNKNOWN
video/x-mpeg	DEMUXER_TYPE_UNKNOWN
video/x-mpeg2	DEMUXER_TYPE_UNKNOWN
video/x-msvideo	DEMUXER_TYPE_AVI
video/quicktime	DEMUXER_TYPE_MOV
application/octet-stream	DEMUXER_TYPE_UNKNOWN
audio/x-pn-realaudio	DEMUXER_TYPE_REAL
application/x-ogg	DEMUXER_TYPE_OGG
video/nsv	DEMUXER_TYPE_NSV
video/x-flv	DEMUXER_TYPE_LAVF

fixup_open() 包含了兩個重要函式，nop_streaming_start() 和 fixup_network_stream_cache()。nop_streaming_start()初始化 streaming_ctrl_t 結構資料，並且指定函式指標 streaming_read 和 streaming_seek。如果設定了 streaming_ctrl_t 結構的 streaming_read 函式指標就不會設定 stream_t 結構的 fill_buffer 函式指標。fixup_network_stream_cache()則是用來調整 Cache2 的大小，詳細第 3.2 節將說明。

3.2 Cache2 機制

Open Stream 的模組支援很多 protocol，檔案來源可能是來自周邊設備、區域網路或是網際網路。這些檔案來源的速度不一定很穩定，而不穩定的資料量會造成播放斷續。播放斷斷續續會讓使用者有不好的影片欣賞經驗。Cache2 就是用來解決速度不穩定的問題，在影片播放前預先下載資料到快取中，再開始播放。而在影片播放的同時，Cache2 會持續從來源下載資料，以提供播放器穩定的資料。

開啟 Cache2 有兩種方法，第一種是在文字命令模式中加上-cache SIZE，SIZE 是任意大於等於 32 的數字，單位為 KB；第二種是 Open Stream 自動開啟，部分 Open Stream 模組有預設一個 prebuffer_size 的大小，如 HTTP 的 prebuffer_size 預設為 64KB，Cache2 調整為 320KB。

Cache2 為 prebuffer_size 的 5 倍大，這是因為 Cache2 會預先填滿 20%才開始播放影片。cache_vars_t 結構是 Cache2 在使用的。表 3-7 為 cache_vars_t 結構所包含的成員。這裡將這些結構成員的函式和變數，做詳加說明與介紹。

表 3-7. cache_vars_t 結構成員

變數	型態
buffer	unsigned char *
buffer_size	int
sector_size	int
back_size	int

fill_limit	int
seek_limit	int
eof	int
min_filepos	off_t
max_filepos	off_t
offset	off_t
read_filepos	off_t
stream	stream_t *

buffer：指向快取記憶體的指標。

buffer_size：快取的大小。

sector_size：單一區段的大小，2048 或是 2324。2324 是用於 VCD 播放。

back_size：已經播放過的資料大小，用來倒退搜索。

fill_limit：當緩衝器空間大於等於 fill_limit 才填充快取。

seek_limit：如果搜索的資料距離小於 seek_limit，繼續填充快取。

eof：檔案結束(End Of File)旗標，設定為 1 代表檔案結束。

min_filepos：快取所存的檔案的最小位置。

max_filepos：快取所存的檔案的最大位置，快取僅存放從 max_filepos–min_filepos 檔案的部分資料。

offset：快取和檔案位置的偏移量。

read_filepos：目前播放器讀取的位置。

stream：呼叫原本的 stream_t 結構變數。

3.2.1 HTTP Cache2 開啟流程

圖 3-10 為 HTTP 模組開啟檔案的程序，nop_streaming_start()預設的 prebuffer_size 為 64*1024 bytes、buffering 為 1。fixup_network_stream_cache()將 stream_cache_size 調整為 prebuffer_size 的 5 倍。注意的是 stream_cache_size 初始值為-1。開檔完成後，stream_cache_size 大小為 320KB，但是使用者可以利用-cache SIZE 指令變更大小。最後

main()呼叫 stream_enable_cache()，這時候 Cache2 啟動。

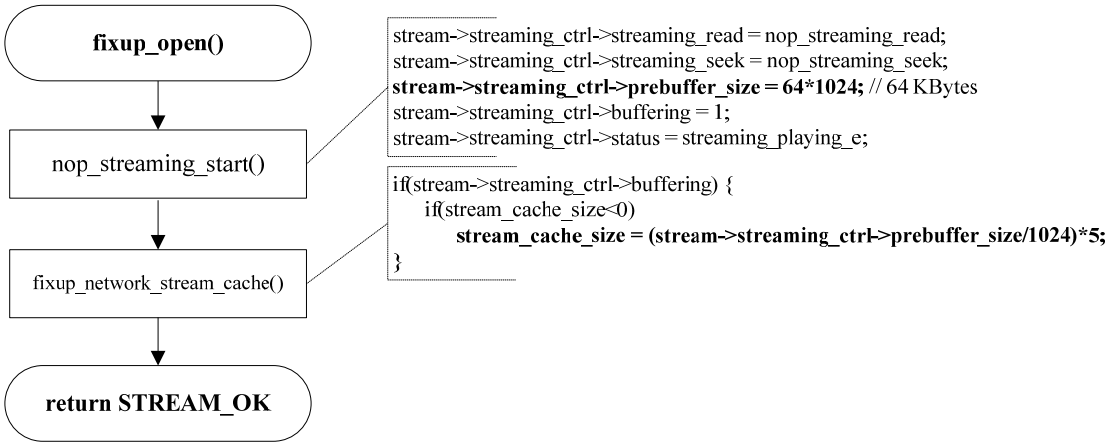

圖 3-10. HTTP Cache2 開啟流程

3.2.2 Cache2 運作流程

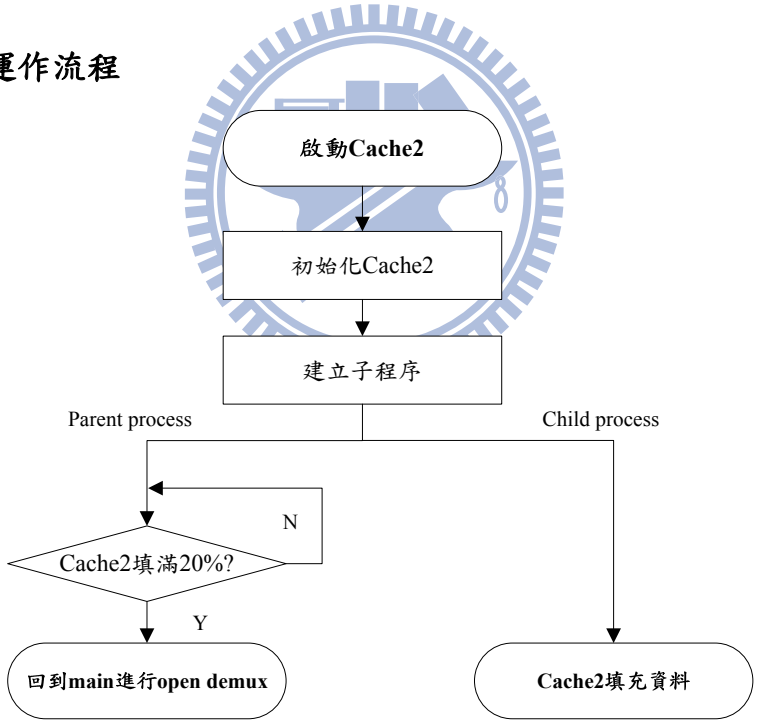

圖 3-11. Cache2 啟動流程圖

圖 3-11 為 Cache2 啟動流程圖，首先會先初始化 Cache2。主要是取得 Cache2 的記憶體空間，並且計算 fill_limit 和 back_size 兩個重要參數。當緩衝器空間小於 fill_limit 就會停止填充資料。back_size 是最多保留多少播放過的舊資料，如果設定為緩衝器的

1/4，那緩衝器的新資料會維持在 75%左右。fill_limit 是設定 8*sector_size，back_size 則是設定為緩衝器的 1/2，圖 3-12 為 Cache2 初始化的程式碼。

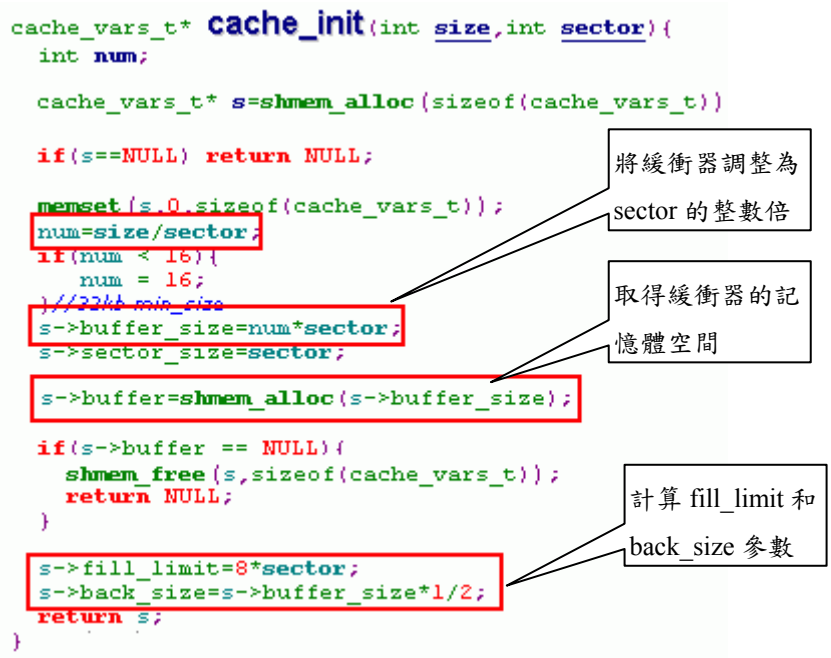

圖 3-12. Cache2 初始化的程式碼

Cache2 初始化完成後，呼叫 fork()建立一個子程序，此時子程序進行緩衝器填充。而母程序等待子程序將緩衝器填充到 20%才返回到 main()進行開啟解多工器。母程序和子程序都是使用 stream_t 的結構操作開檔模組，透過成員中的 cache_pid 判斷是哪個程序。如果 cache_pid 是 0 代表為子程序，就使用 stream_t 的 fill_buffer()或是 streaming_ctrl_t 的 streaming_read()將來源資料讀取到 stream_t 的 buffer[SECTOR_SIZE]然後再寫入 Cache2。如果 cache_pid 非 0 代表為母程序，此時使用 stream_t 的結構讀取資料，就會改讀取 Cache2 的資料。

當沒有開啟 Cache2 的時候，cache_pid 是預設為 0，所以此時會使用 stream_t 的 fill_buffer()或是 streaming_ctrl_t 的 streaming_read()從來源讀取資料。

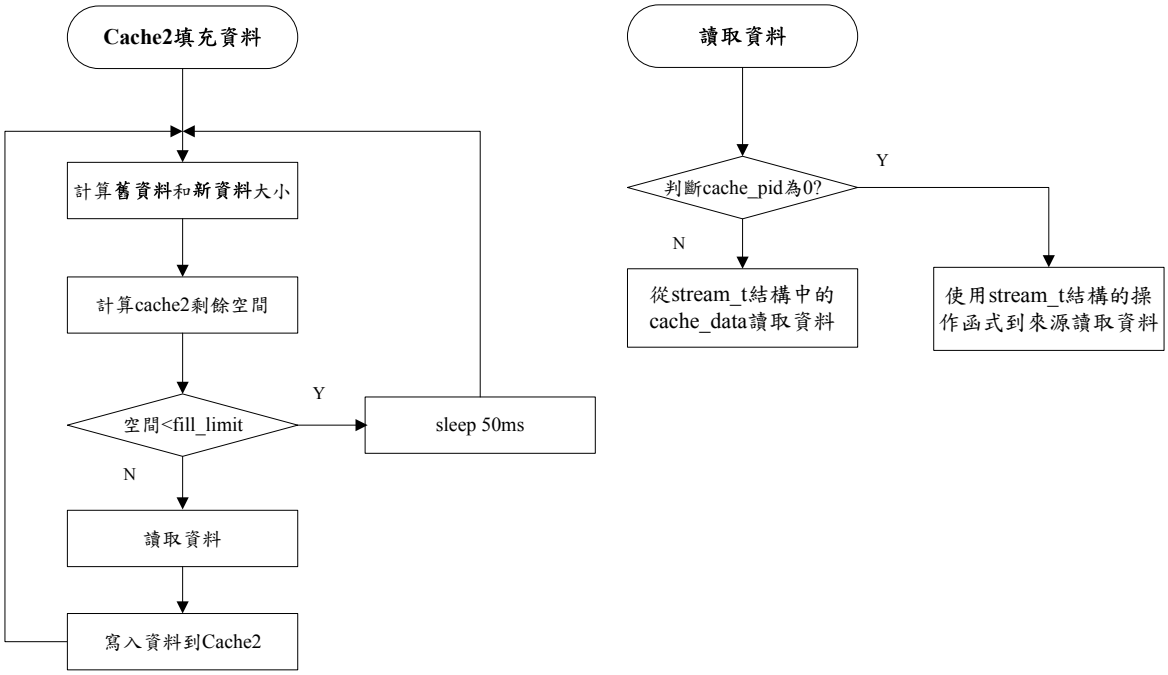

圖 3-13. Cache2 填充資料的流程

圖 3-14 為 Cache2 的運作流程，Min 為舊資料的起始位置，Max 為新資料的最終位置，Read：目前讀取資料的位置。舊資料大小為 Read-Min，新資料大小為 Max-Read，Cache2 剩餘空間為 Cache2 大小-(舊資料大小+新資料大小)。

(a)Cache2 最初狀態，Min、Max 和 Read 的位置都為 0。

(b)影片開始播放，子程序持續填充資料到 Cache2。

(c)當 Cache2 剩餘空間小於 fill_limit，子程序停止填充資料，母程序持續從 Cache2 取得資料。

(d)由於 back_size 設定為 Cache2 的 1/2，代表舊資料最多保存 Cache2 一半的大小。

(e)調整舊資料的起始位置，此時 Cache2 剩餘空間變大。子程序可以填充資料到 Cache2。

(f)當 Cache2 剩餘空間小於 fill_limit，子程序停止填充資料。當母程序讀取資料，Read 位置往前移，Min 的位置被調整，此時剩餘空間大於 fill_limit，子程序又可以填充資料，填充到剩餘空間小於 fill_limit 就停止。如此反復運作，新資料可以維持在 50%。圖 3-15 說明 cache 資料水位的變化。

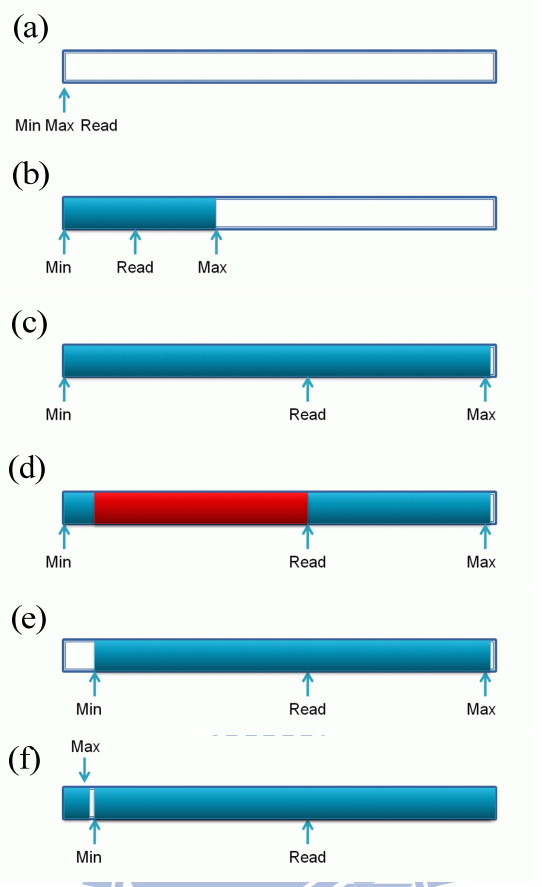

圖 3-14. Cache2 的運作流程

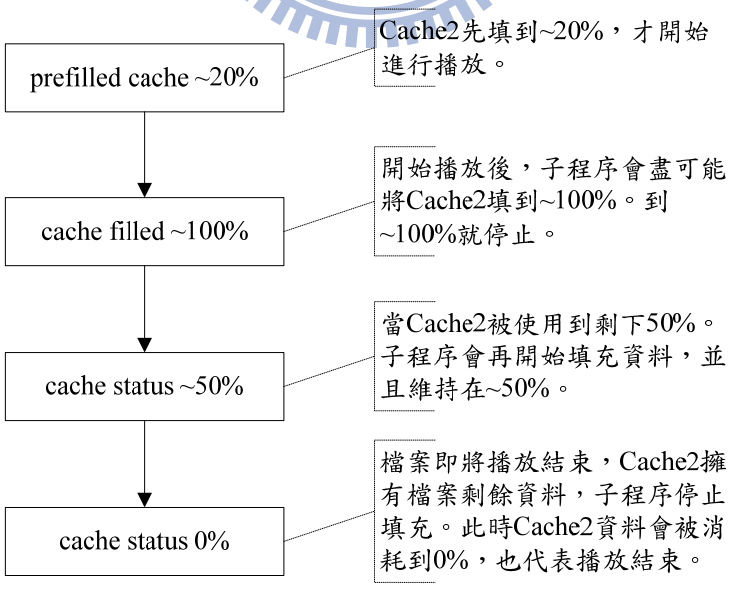

圖 3-15. Cache2 cache status

3.3 Video Filter 模組

MPlayer 1.0rc2-4.1.2 內建了許多影像濾波器(Video Filter)，包括 flip image upside-down、rotate、horizontal mirror、drop interlaced frames、screenshot to file…等，而每個影像濾波器都有獨立的 vf_info_t 結構資料。使用者在播放影片時可以同時開啟一個或多個影像濾波器，而多個影像濾波器串接起來稱為 filter chain。表 3-8 為 vf_info_t 的結構成員。

表 3-8. vf_info_t 結構成員

變數	型態
info	const char *
name	const char *
author	const char *
comment	const char*
open()	int
opts	void *

info：基本資訊為該濾波器完整描述，如 flip image upside-down。

name：模組名稱為開啟模組代碼，如 flip image upside-down 為 flip。

author：作者名字。

comment：建議事項。

open：指向模組開啟函式的指標，這個函式是模組的進入點。當 MPlayer 選定開啟的濾波器時，會呼叫這個函式。這個函式需要指定部分的 vf_instance_t 結構資料，將實現的函式指定到結構中的函式指標。

圖 3-16 為 flip 模組的 vf_info_t 結構資料，最重要的是 open()對應的參數。

```
vf_info_t vf_info_flip = {
    "flip image upside-down",    //info
    "flip",                      //name
    "A'rpi",                     //author
    "",                          //comment
    open,                        //open()
    NULL                         //opts
};
```

圖 3-16. flip 模組的 vf_info_t 結構資料

MPlayer 定義了 vf_instance_t 結構，用來操作選定的影像濾波器的函式，成員還有其他變數和結構存放圖像資料，操作每個影像濾波器需要獨立的 vf_instance_t 結構。在模組開啟的前置函式 vf_open_plugin()，將指定 vf_instance_t 結構的函式指標變數，這是每個模組都包含的要素，如圖 3-17 的程式碼。

圖 3-18 為 flip 模組的 open()函式，需要將模組中實現的函式指定到 vf_instance_t 的函式指標，如果先前已經有指定的結構變數，在此會被複寫。表 3-9 為 vf_instance_t 結構所包含的成員，這節將這些結構成員的函式和變數，做詳加說明與介紹。

```
vf_instance_t* vf_open_plugin(vf_info_t** filter_list, vf_instance_t* next, const char *name, char **args){
    vf_instance_t* vf;
    int i;
    for(i=0;;i++){
    if(!filter_list[i]){
        mp_msg(MSGT_VFILTER,MSGL_ERR,MSGTR_CouldNotFindVideoFilter,name);
        return NULL; // no such filter!
    }
    if(!strcmp(filter_list[i]->name,name)) break;
    }
    vf=malloc(sizeof(vf_instance_t));
    memset(vf,0,sizeof(vf_instance_t));
    vf->info=filter_list[i];
    vf->next=next;
    vf->config=vf_next_config;
    vf->control=vf_next_control;
    vf->query_format=vf_default_query_format;
    vf->put_image=vf_next_put_image;
    vf->default_caps=VFCAP_ACCEPT_STRIDE;
    vf->default_reqs=0;
```

所有影像濾波器的 vf_instance_t 結構都包含這些初始設定

圖 3-17. vf_open_plugin()函式的部分程式碼

```
static int open(vf_instance_t *vf, char* args){
    vf->config=config;
    vf->get_image=get_image;
    vf->put_image=put_image;
    vf->default_reqs=VFCAP_ACCEPT_STRIDE;
    return 1;
}
```

指定該模組實現的函數，這將複寫原本的初始設定

圖 3-18. flip 模組的 open()函式

表 3-9. vf_instance_t 結構成員

變數	型態	回傳型態
info	vf_info_t *	-
config()	函式指標	int
control()	函式指標	int
query_format()	函式指標	int
get_image()	函式指標	void
put_image()	函式指標	void
start_slice()	函式指標	void
draw_slice()	函式指標	void
uninit()	函式指標	void
continue_buffered_image	int 指標	-
default_caps	unsigned int	-
default_reqs	unsigned int	-
w,h	int	-
imgctx	vf_image_context_t	-
fmt	vf_format_context_t	-
next	vf_instance_t *	-
dmpi	mp_image_t *	-
priv	vf_priv_t *	-

info：影像濾波器模組的基本資訊。

config()：設定和初始化影像濾波器。

control()：控制介面。呼叫此函式需要傳入一個控制類別的旗標。該函式會針對被呼叫的類別，執行該段的程式碼。類別項目由下面做詳細介紹。

VFCTRL_SET_EQUALIZER

設定影像的亮度、對比和飽和度。

VFCTRL_GET_EQUALIZER

回傳影像的亮度、對比和飽和度。

VFCTRL_DRAW_OSD

繪製字幕和 OSD。

VFCTRL_FLIP_PAGE

通知影像輸出模組做影像翻轉。

VFCTRL_DUPLICATE_FRAME

複製影像畫面。

VFCTRL_SKIP_NEXT_FRAME

跳過下一個影像。

VFCTRL_FLUSH_FRAMES

清除延遲的影像畫面。

VFCTRL_SCREENSHOT

擷取單張影像畫面。

VFCTRL_SET_DEINTERLACE

設定去除交錯掃描的狀態。

VFCTRL_GET_DEINTERLACE

回傳去除交錯掃描的狀態。

query_format()：針對傳入的影像格式的旗標，回傳模組所支援模式的旗標。在此會針對所支援的格式，透過 vf_next_query_format()函式詢問下個影像濾波器所支援的模式。

get_image()：實現影像 direct rendering，進行立即(in-place)像素轉換。

put_image()：影像濾波器做影像處理的主要函式，傳入的參數為 mpi 影像結構。

draw_slice()：支援濾波器處理部分圖像資料。

uninit()：關閉影像濾波器模組，在此解放濾波器所使用的記憶體空間。

fmt：vf_format_context_t 結構。have_configured 為是否已經設定的旗標，orig_width 為影像原始寬度，orig_height 為影像原始高度，orig_fmt 為影像原始的格式。

表 3-10. vf_format_context_t 結構成員

變數	型態
have_configured	int
orig_width	int
orig_height	int
orig_fmt	int

next：指向下一個影像濾波器的 vf_instance_t 結構的指標。

dmpi：經過影像濾波器處理過後的影像(mpi_image_t)結構指標。

priv：vf_priv_t 結構指標，結構存放影像濾波器所使用的特殊變數，結構成員會依濾波器的需求而有所不同。

3.4 Video Output 模組

MPlayer 內建許多影像輸出模組，使用者可以依需求選擇模組，例如使用 JPEG file 的輸出模組，可以將畫面存成一張張的 JPEG 圖檔；或是使用 animated GIF ouput 的輸出模組，可以將畫面存成 GIF 動畫圖檔。MPlayer 為了要操作影像輸出模組，定義了 vo_functions_t 結構。所有的影像輸出模組都要這些結構成員，這些結構成員包含許多函式指標，而這些函式指標指向模組內特定函式，程式開發者可以針對每個函式撰寫對應的程式碼。MPlayer 1.0rc2-4.1.2 內建 X11 (XImage/Shm)、PNG file、JPEG file、animated GIF output…等。表 3-11 為 vo_functions_t 結構所包含的成員。這節將這些結構成員的函式，做詳加說明與介紹。

影像輸出模組的 vo_functions_t 結構和影像濾波器的 vf_instance_t 結構的差別，前者在模組內宣告指定所有結構成員的資料，後者是在模組開啟的時候才去指定結構成員的資料。圖 3-19 和圖 3-20 為影像輸出模組利用巨集的方式指定結構成員的資料。

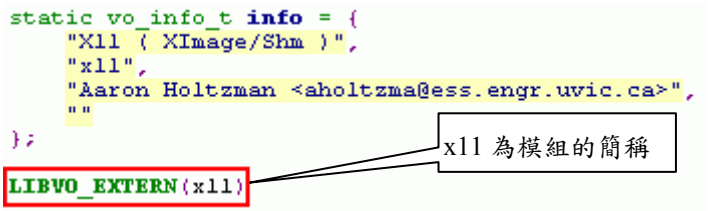

圖 3-19. x11 模組指定 vo_functions_t 結構成員資料

```
#define LIBVO_EXTERN(x) vo_functions_t video_out_##x =\
{\
    &info,\          //info
    preinit,\        //preinit()
    config,\         //config()
    control,\        //control()
    draw_frame,\     //draw_rame()
    draw_slice,\     //draw_slice()
    draw_osd,\       //draw_osd()
    flip_page,\      //flip_page()
    check_events,\   //check_events()
    uninit\          //uninit()
};
```

圖 3-20.利用巨集指定 vo_functions_t 結構成員資料

表 3-11. vo_functions_t 結構成員

變數	型態	回傳型態
info	vo_info_t *	-
preinit()	函式指標	int
config()	函式指標	int
control()	函式指標	int
draw_frame()	函式指標	void
draw_slice()	函式指標	int
draw_osd()	函式指標	void
flip_page()	函式指標	void
check_events()	函式指標	void
uninit()	函式指標	void

info：影像輸出模組的基本資訊，模組名稱(name)、模組簡稱(short_name)、模組撰寫作者(author)和建議事項(comment)。模組名稱為完整全名，如 X11 (XImage/Shm)。模組簡稱為開啟模組代碼，如 X11 (XImage/Shm)為 x11。

表 3-12. vo_info_t 結構成員

變數	型態
name	const char *
short_name	const char *
author	const char *
comment	const char *

```
static vo_info_t info = {
    "X11 ( XImage/Shm )",                                   //name
    "x11",                                                  //short_name
    "Aaron Holtzman <aholtzma@ess.engr.uvic.ca>",           //author
    ""                                                      //comment
};
```

圖 3-21. x11 模組的 vo_info_t 結構資料

preinit()：初始化影像輸出系統，如果初始化成功回傳零值，否則回傳非零值。

config()：設定影像輸出系統，在此傳入參數為影像的寬度和高度、輸出畫面的寬度和高度、視窗的標題、全螢幕的旗標和 fourcc 影像格式。

control()：控制介面。呼叫此函式需要傳入一個控制類別的旗標。該函式會針對被呼叫的類別，執行該段的程式碼。類別項目由下面做詳細介紹。

VOCTRL_QUERY_FORMAT

針對傳入的影像格式的旗標，回傳模組所支援模式的旗標。這控制類別可能在 config()前被呼叫，但是永遠在 preinit()和 uninit()之前執行。下面三個為最重要的旗標，所有模組都要回傳支援與否。

VFCAP_CSP_SUPPORTED 支援該影像格式的 colorspace

VFCAP_CSP_SUPPORTED_BY_HW colorspace 由硬體轉換

VFCAP_TIMER 由硬體控制時間同步

VOCTRL_GET_IMAGE

直接渲染介面(Direct Rendering Interface)模式，需要更新 mpi 結構。

VOCTRL_DRAW_IMAGE

取代 draw_slice()/draw_frame()函式。如果沒有實現該功能，draw_slice()或是 draw_frame()會被呼叫。

VOCTRL_RESET

重設播放器，有些裝置在重新填充緩衝器前，需清除先前儲存的資料。

VOCTRL_PAUSE

處理播放器在暫停的狀態。

VOCTRL_RESUME

處理播放器恢復播放的狀態。

VOCTRL_GUISUPPORT

回傳輸出模組是否支援圖形介面(GUI)。

VOCTRL_SET_EQUALIZER

設定輸出畫面的亮度、對比和飽和度。

VOCTRL_GET_EQUALIZER

回傳目前輸出畫面的亮度、對比和飽和度。

VOCTRL_ONTOP

設定播放器最上層顯示，僅支援 Linux 的 x11 和 Wiindows 的 directx,g12。

VOCTRL_BORDER

設定播放視窗無邊框，僅支援 Wiindows 的 directx。

draw_frame()：只支援包裝(packed)的 YUY2, RGB/BGR 格式。只支援更新全部圖像。如果有完成 VOCTRL_DRAW_IMAGE 功能，這個函式可以不用實現。

draw_slice()：支援平面(planar)格式，傳入 YUV 三個圖像的指標。如 YV12 格式，一個完整大小的 Y 和四分之一大小的 U,V。支援圖像部分更新。

draw_osd()：繪製字幕或 OSD 到影像緩衝器。

flip_page()：顯示緩衝器裡面的影像。如 x11 模組下，這個函式是將緩衝器的圖像資料，透過 Xlib 的 XPutImage()函式貼到畫面上。

check_events()：擷取播放視窗上的鍵盤和滑鼠控制訊息。

uninit()：關閉影像輸出系統，將系統回復到原先狀態。

3.4.1 vo Video Filter

MPlayer 1.0rc2-4.1.2 內建了許多影像濾波器，唯一比較特殊的是 libvo wrapper 這個模組(簡稱 vo 影像濾波器)。這個模組一定要開啟使用，它並沒有任何影像處理的功能而是用來連結影像濾波器和影像輸出模組。MPlayer 支援使用多個影像濾波器，這些多個影像濾波器開啟串接後，形成 filter chain。MPlayer 操作影像濾波器是使用 vf_instance_t 結構，並且利用 linked list 將開啟的影像濾波器的 vf_instance_t 串在一起。如此可以將影像做適當處理並將資料傳到下一個影像濾波器。而影像濾波器使用的 vf_instance_t 結構和影像輸出模組的 vo_functions_t 結構不同，所以 filter chain 的最後一個就是 vo 影像濾波器，最後將資料傳遞給影像輸出模組。

圖 3-22 為 vo 影像濾波器的 open()函式，除了指定 vf_instance_t 的結構成員，最重要是傳入選定的影像輸出模組的 vo_functions_t 結構的指標。並且將 vo_functions_t 指定到 vf->priv->vo 的指標變數(video_out 和 vf->priv->vo 是同一個變數)。如此 vo 影像濾波器才可以利用 vo_functions_t 的結構指標操作使用選定的影像輸出模組。

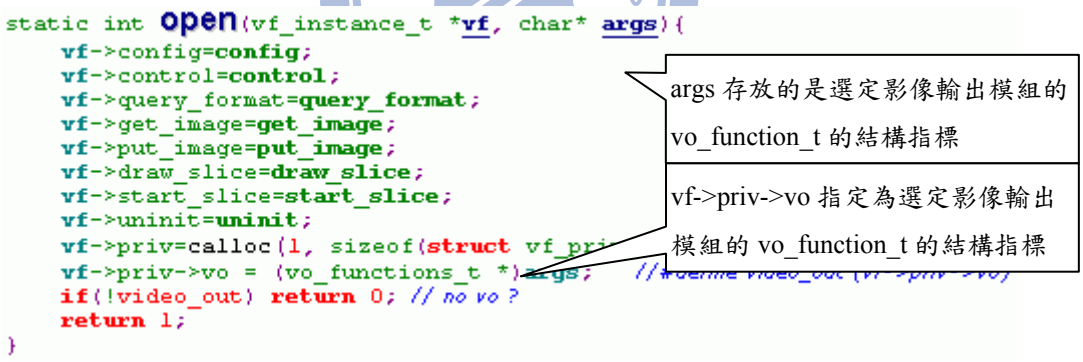

圖 3-22. vo 影像濾波器的 open()函式

我們就 vo 影像濾波器 query_format()、get_image()和 put_image()函式，說明如何透過這個模組和選定的影像輸出模組連結。當 query_format()函式被呼叫，這個函式再呼叫影像濾波器的 control()函式並且傳入對應的操作旗標 VOCTRL_QUERY_FORMAT 和變數 fmt(colorspace 的代碼)，如圖 3-23。而 get_image()函式則是呼叫影像濾波器的 control()函式並且傳入對應的操作旗標 VOCTRL_GET_IMAGE 和變數 mpi，如圖 3-24。

```
static int query_format(struct vf_instance_s* vf, unsigned int fmt){
    int flags=video_out->control(VOCTRL_QUERY_FORMAT,&fmt);
    if(flags)
        if(fmt==IMGFMT_YV12 || fmt==IMGFMT_I420 || fmt==IMGFMT_IYUV)
            flags|=VFCAP_ACCEPT_STRIDE;
    return flags;
}
```

圖 3-23. vo 模組的 query_format()函式

```
static void get_image(struct vf_instance_s* vf, mp_image_t *mpi){
    if(vo_directrendering && vo_config_count)
        video_out->control(VOCTRL_GET_IMAGE,mpi);
}
```

圖 3-24. vo 模組的 get_image()函式

put_image()函式在影像濾波器中是影像處理函式，但是在 vo 影像濾波器並不做影像處理，而是將圖像資料傳給影像輸出模組。圖 3-25，傳遞圖像資料有三個函式可以使用，但是只有一個函式會被執行，這必須看影像輸出模組是如何實現這部份的功能。第一種，影像輸出模組的 control()函式並傳入操作旗標 VOCTRL_DRAW_IMAGE 和變數 mpi。所以影像輸出模組可以得到 mpi 整個結構的資料。第二種，影像輸出模組的 draw_slice()並傳入圖像資料的指標、圖像寬度和高度。第三種，影像輸出模組的 draw_frame()並傳入圖像資料的指標。

```
static int put_image(struct vf_instance_s* vf, mp_image_t *mpi, double pts){
  if(!vo_config_count) return 0; // vo not configured?

  vf->priv->pts = pts;
1 if( video_out->control(VOCTRL_DRAW_IMAGE,mpi)==VO_TRUE ) return 1;

  if(!(mpi->flags&(MP_IMGFLAG_DIRECT|MP_IMGFLAG_DRAW_CALLBACK))){
    if(vf->default_caps&VFCAP_ACCEPT_STRIDE)
  2   video_out->draw_slice(mpi->planes,mpi->stride,mpi->w,mpi->h,mpi->x,mpi->y);
    else
  3   video_out->draw_frame(mpi->planes);
  }
  return 1;
}
```

圖 3-25. vo 模組的 put_image()函式

除了我們介紹的 query_format()、get_image()和 put_image()，還有其他對應的函式如表 3-13。uninit()函式並沒有任何對應函式，這是因為影像濾波器和影像輸出模組是分開關閉。

表 3-13. vo 影像濾波器函式對應表

vo 影像濾波器	影像輸出模組
config()	config()
control()	control()
query_format()	control()的 VOCTRL_QUERY_FORMAT 旗標
get_image()	control()的 VOCTRL_GET_IMAGE 旗標
put_image()	control()的 VOCTRL_DRAW_IMAGE 旗標 或 draw_slice()或 draw_frame()
start_slice()	control()的 VOCTRL_START_SLICE 旗標
draw_slice()	draw_slice()
uninit()	無

3.5 Video Chain 開啟流程

當影音資料透過 demuxer 分解成 video bitstream 和 audio bitstream，將 video bitstream 傳到影像解碼器。影像解碼器將資料解碼成 video frame，再透過影像濾波器做影像處理，最後透過影像輸出模組顯示到螢幕上。而影像解碼器、影像濾波器和影像輸出模組構成 video chain，如圖 3-26 虛線框部分。MPlayer 支援同時開啟多個影像濾波器，而這個多個影像濾波器構成 video filter chain，如圖 3-27 虛線框部分。前面我們談論了影像濾波器和影像輸出模組，但是這都是單獨的模組。這章節將說明 MPlayer 如何將這些獨立的模組串接起來。

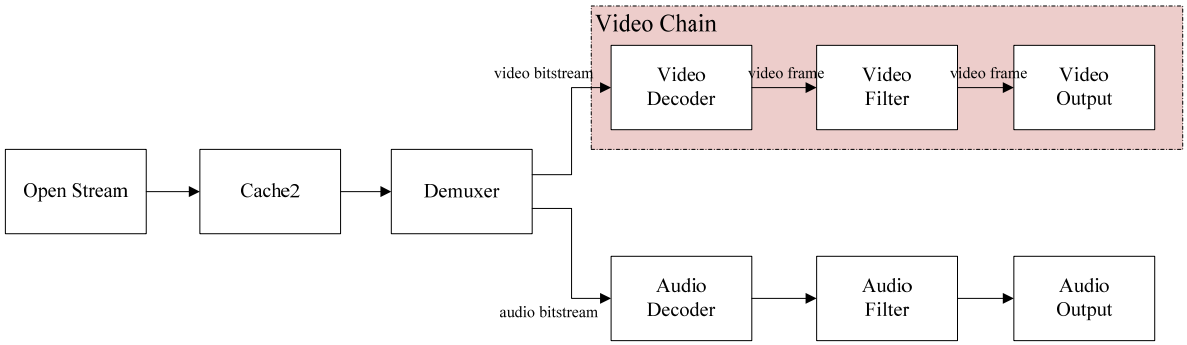

圖 3-26. Video chain 架構

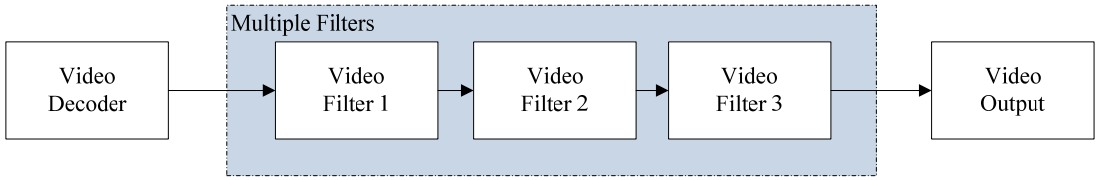

圖 3-27. Video Filter chain 架構

MPlayer 選定 demuxer 後，從檔案資料讀出標頭資訊。如果檔案有包含視訊標頭訊息，將建立 video chain，這將決定影像解碼器、影像濾波器和影像輸出模組，並且將它們串接起來。圖 3-28 為 video chain 建立流程，將分成四個階段說明。

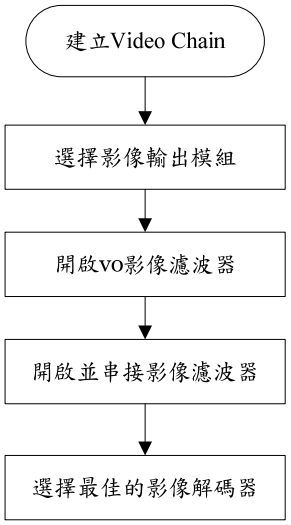

圖 3-28. Video chain 開啟流程

1.選擇影像輸出模組(init_best_video_out())

選擇最佳的影像輸出模組，並且回傳該模組的vo_function_t結構的指標，指定到mpctx->video_out指標變數，如圖3-29的第1部分。首先開啟使用者選定影像輸出模組，如果使用者沒有設定，那就會自動選擇影像輸出模組。選擇方式是依序呼叫表3-14的影像輸出模組的前置初始化函式(preinit())，只要這個函式初始化成功，就會選定該模組為影像輸出模組；如果初始化失敗就會呼叫列表中的下一個模組。圖3-30為第1階段video chain的狀態。(假設選定x11影像輸出模組)

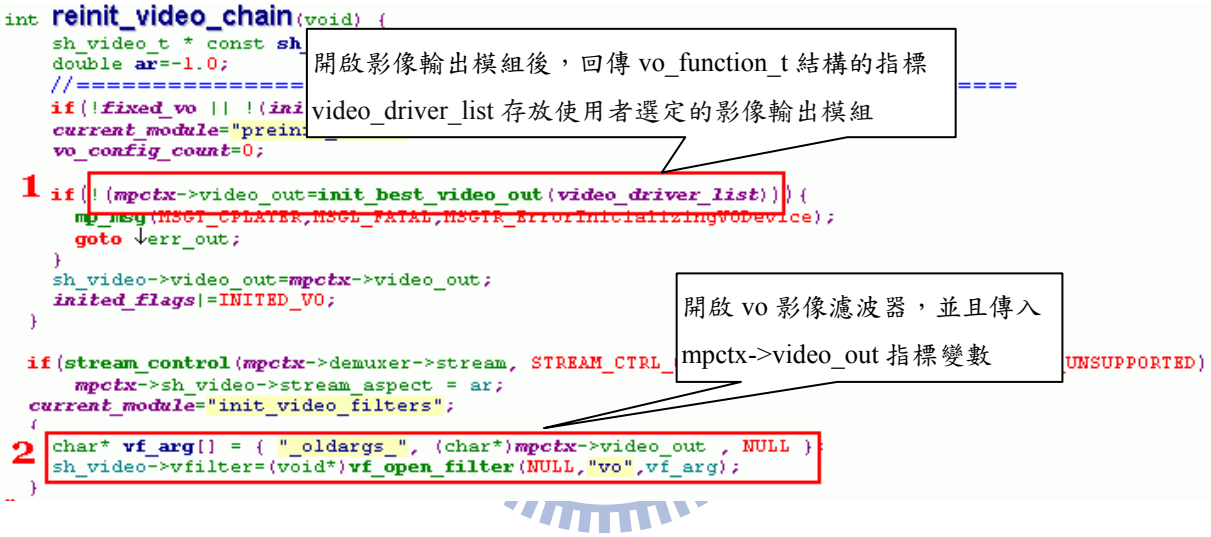

圖 3-29. reinit_video_chain()階段 1 和階段 2 的程式碼

表 3-14. MPlayer 內建的影像輸出模組

模組簡稱	說明
xv	X11/Xv
x11	X11 (XImage/Shm)
xover	General X11 driver for overlay capable video output drivers
gl	X11 (OpenGL)
gl2	X11 (OpenGL) - multiple textures version
dga	DGA (Direct Graphic Access V2.0)
sdl	SDL YUV/RGB/BGR renderer (SDL v1.1.7+ only!)
fbdev	Framebuffer Device
fbdev2	Framebuffer Device
v4l2	V4L2 MPEG Video Decoder Output

xvidix	X11 (VIDIX)
cvidix	console VIDIX
null	Null video output
rdp	rdesktop plugin output
mpegpes	Mpeg-PES to DVB card
yuv4mpeg	yuv4mpeg output for mjpegtools
png	PNG file
jpeg	JPEG file
gif89a	animated GIF output
tga	Targa output
pnm	PPM/PGM/PGMYUV file
md5sum	md5sum of each frame

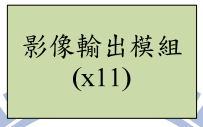

圖 3-30. Video chain 階段 1

2. 開啟vo影像濾波器(vf_open_filter())

vo影像濾波器是一個特殊的濾波器而且一定是filter chain的最後一個，它是影像濾波器和影像輸出模組的媒介(vo影像濾波器在第3.4.1節有詳細解說)。圖3-29的第2部分，傳入影像濾波器的名稱(vo)和mpctx->video_out指標變數(存放x11影像輸出模組的vo_function_t結構的指標)。vo影像濾波器開啟完成後，回傳vo影像濾波器的vf_instance_t結構的指標，指定到sh_video->vfilter指標變數。

圖3-31為vo影像濾波器的open()函式，此開啟動作將會把vf_instance_t結構資料指定完成。而vf->priv->vo也就是video_out，存放的是影像輸出模組(x11)的vo_function_t結構，透過這個結構成員的函式指標可以操作影像輸出模組的(x11)的函式。

```
static int open(vf_instance_t *vf, char* args){
    vf->config=config;
    vf->control=control;
    vf->query_format=query_format;
    vf->get_image=get_image;
    vf->put_image=put_image;
    vf->draw_slice=draw_slice;
    vf->start_slice=start_slice;
    vf->uninit=uninit;
    vf->priv=calloc(1, sizeof(struct vf_priv_s
    vf->priv->vo = (vo_functions_t *)args
    if(!video_out) return 0; // no vo ?
    return 1;
}
```

args 存放的是 x11 的 vo_function_t 的結構指標

vf->priv->vo 指定為 x11 的 vo_function_t 的結構指標

圖 3-31. vo 影像濾波器的 open()函式

圖3-32為video chain階段2，目前MPlayer已經開啟了vo影像濾波器和x11影像輸出模組。(假設選定x11影像輸出模組)

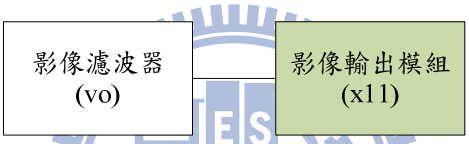

圖 3-32. Video chain 階段 2

3.開啟並串接影像濾波器(append_filters())

這個函式是將使用者選定的影像濾波器一一開啟並且串接。圖3-33的第3部分，傳入sh_video->vfilter的指標變數(目前是指到vo影像濾波器)，最後回傳的是video filter chain的第一個濾波器的vf_instance_t結構指標。開啟和串接都是由後往前，所以是從vo往前串接。

圖 3-33. reinit_video_chain()階段 3 和階段 4 的程式碼

圖3-34為append_filters()的程式碼。vf_settings[]存放使用者選定的影像濾波器，這資料是MPlayer解析指令的時候，將所有輸入的影像濾波器依序存放在這字串陣列。由於filter chain是由後往前串接，所以會從vf_settings[]的最後一個影像濾波器往前開啟。串接方式是用linked list，每個影像濾波器(除了vo影像濾波器)的操作結構都有存放指向下一個影像濾波器結構的指標。假設選定順序為flip、rotate，則由rotate先開啟，最後再開啟flip。

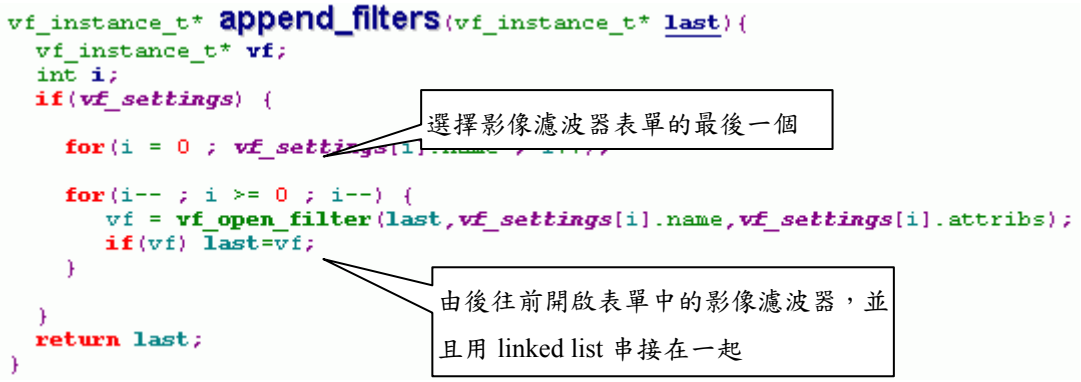

圖 3-34. append_filters()的程式碼

圖3-35為video chain階段3，目前MPlayer已經開啟了flip、rotate、vo影像濾波器和x11影像輸出模組。(假設選定flip和rotate影像濾波器)

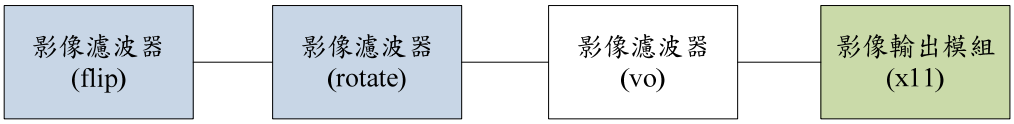

圖 3-35. Video chain 階段 3

4.選擇最佳影像解碼器(init_best_video_codec())

圖3-33的第4部分，傳入sh_video結構(存放視訊的標頭訊息)、video_codec_list(使用者選定的影像解碼器)和video_fm_list(使用者選定的影像解碼器模組)，如果使用者沒有選定影像解碼器和影像解碼模組，該函式將從視訊的標頭訊息比對選擇最佳的影像解碼器。圖3-36為video chain階段4，如此MPlayer已經將video chain整個建構完成。

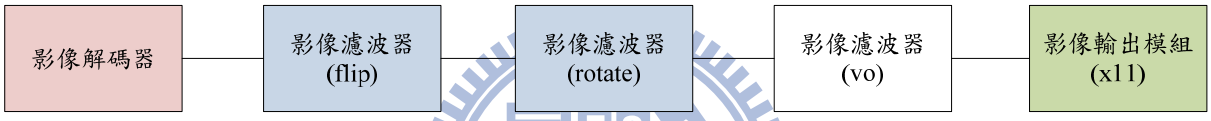

圖 3-36. Video chain 階段 4

3.6 Video Path 資料傳遞

Video chain建立完成後，已經找到影像解碼器、影像濾波器和影像輸出模組，MPlayer播放影片的時候，會將視訊資料依序通過video chain的每個部份，這資料的傳遞的路徑稱為video path。這章節將說明這視訊資料是如何在模組之間傳遞，還有在解碼前各模組之間的初始化動作。3.6.1將說明video chain各模組初始化的流程和3.6.2將說明影像播放時圖像資料在模組間的傳遞。

3.6.1 初始化

1.選擇最佳的 colorspace

(1)、影像解碼器在解出第一張frame之前，需要初始化影像輸出模組。透過呼叫外部函式(mpcodecs_config_vo())，並且傳入影像的寬度、高度和colorspace。

(2)、mpcodecs_config_vo()從選定的video codec所支援的colorspace，一一詢問影像解碼器、影像濾波器和影像輸出模組是否支援該colorspace。

影像解碼器模組(如ffmpeg)支援多種video codec，而這些video codec的基本資訊，全部都存放在/etc/codecs.conf檔案中。當MPlayer開啟的時候，處理video codec設定檔的函式將讀取設定值。這個函式會將codec支援的colorspace(存在設定檔特定videocodec的名稱out，如圖3-37)透過查表方式，轉成特定的代碼。例如選擇FFmpeg MPEG-4 codec，將會詢問YV12、I420和IYUV三個格式，而這三個格式代碼分別為0x32315659、0x30323449和0x56555949。

```
videocodec ffodivx
  info "FFmpeg MPEG-4"
  status working
  fourcc FMP4,fmp4
  fourcc DIVX,divx
  fourcc DIV1,div1 divx
  fourcc MP4S,mp4s  ; ISO MPEG-4 Video V1
  fourcc M4S2,m4s2
  fourcc xvid,XVID,XviD,XVIX
  fourcc DX50,dx50,BLZ0 DX50
  fourcc mp4v,MP4V
  format 0x4
  fourcc UMP4
  fourcc RMP4
  fourcc 3IV2,3iv2  ; 3ivx Delta 4
  fourcc DXGM
  fourcc SEDG ; diskless camcorder Samsung Miniket VP-M110
  fourcc SMP4,smp4 ; Samsung SMP4 video codec
  format 0x10000004  ; mpeg 4 es
  driver ffmpeg
  dll mpeg4 ;opendivx
  out YV12,I420,IYUV
```

圖 3-37. ffodivx 的 codecs.conf 訊息

詢問影像濾波器是透過呼叫該濾波器的函式(query_format())，並且傳入colorspace所對應的代碼，再利用查表的方式查詢是否支援。圖3-38的1，傳入要詢問的影像濾波器的操作結構指標和colorspace的代碼，最後回傳所支援的旗標，由於先前已經建立好video chain，所以在此會依序詢問linked list裡的影像濾波器。每個濾波器被呼叫的時候，都是利用查表方式判斷是否支援傳入的colorspace代碼，不支援就回傳0，支援就呼叫vf_next_query_format()。透過這個函式將會呼叫下一個影像濾波器的query_format()。回傳的flags變數是所支援的旗標，而最重要的旗標為VFCAP_CSP_SUPPORTED(支援該colorspace)和

VFCAP_CSP_SUPPORTED _BY_HW(colorspace不需要任何轉換)。

詢問影像輸出模組也是利用查表方式，這功能運作是呼叫該模組control()函式的VOCTRL_QUERY_FORMAT旗標。該影像輸出模組的control()函式利用查表方式執行旗標所對應的程式碼，然後該對應的程式碼又透過查表的方式查詢對colorspace支援與否。影像解碼器則是透過呼叫解碼器的control()函式的VDCTRL_QUERY_FORMAT，如圖3-38的2。

mpcodecs_config_vo()由VFCAP_CSP_SUPPORTED_BY_HW旗標找出影像解碼器、影像濾波器和影像輸出模組的最佳colorspace。如果都沒有任何符合的colorspace，將會嘗試插入“scale filter”在影像解碼器和第一個影像濾波器之間。如果再沒有符合的colorspace，影片就不能播放。

```
for(i=0;i<CODECS_MAX_OUTFMT;i++){
    int flags;
    out_fmt=sh->codec->outfmt[i];
    if(out_fmt==(unsigned int)0xFFFFFFFF)
        continue;
1
    flags=vf->query_format(vf,out_fmt);
    if((flags&VFCAP_CSP_SUPPORTED_BY_HW) || (flags&VFCAP_CSP_SUPPORTED && j<0)){
        sh->outfmtidx=j;
2   if(mpvdec->control(sh,VDCTRL_QUERY_FORMAT,&out_fmt)==CONTROL_FALSE){
            continue;
        }
        j=i;
        vo_flags=flags;
        if(flags&VFCAP_CSP_SUPPORTED_BY_HW)
            break;
    } else if(!palette && !(flags&(VFCAP_CSP_SUPPORTED_BY_HW|VFCAP_CSP_SUPPORTED))
            && (out_fmt==IMGFMT_RGB8||out_fmt==IMGFMT_BGR8))
    {
        sh->outfmtidx=j;
        if(mpvdec->control(sh,VDCTRL_QUERY_FORMAT,&out_fmt)!=CONTROL_FALSE)
        palette=1;
    }
} ? end for i=0;i<CODECS_MAX_OUTF... ?
```

圖 3-38. mpcodec_config_vo()部分程式碼

2.影像濾波器、影像輸出模組 config

colorspace決定好了，再來就是要通知filter chain中所有模組在播放過程會收到影像的寬度、高度和colorspace。並非每個濾波器會得到同一個參數，例如在rotate(影像順時針轉90度)會將收到的影像寬度和高度的參數對調，然後再通知下

一個影像濾波器。這是由mpcodecs_config_vo()呼叫vf_config_wrapper()進行設定，這函式將從filter chain第一個成員開始進行(呼叫該模組的config())，收到設定參數的模組要負責傳遞參數到下一個影像濾波器，直到最後一個成員設定完成為止。

3.取得 mpi 的圖像記憶體空間

(1)、取得影像解碼器的mpi記憶體空間：

在 video chain 的各模組間傳遞圖像資料是利用 MPlayer 的特殊格式 mp_image_t(簡稱mpi)結構。除了結構所需要的記憶體空間外，結構中有指向圖像資料的指標。而影像解碼器會透過呼叫外部函式(mpcodecs_get_image())取得這些圖像資料的記憶體空間。

如果這些圖像資料的記憶體空間不是由影像解碼器的內部取得。mpcodecs_get_image() 會嘗試使用 direct rendering，這將呼叫影像濾波器的 get_image()取得記憶體空間。如果成功MP_IMGFLAG_DIRECT旗標會被設定，在影像解碼器傳遞圖像資料到影像濾波器時，將省去一次記憶體資料複製的動作。

(2)、取得影像濾波器的mpi記憶體空間：

影像濾波器在做影像處理之前，需要取得mpi的圖像記憶體空間。這是用來存放影像處理過後的圖像資料。透過呼叫外部函式(vf_get_image())取得新的mpi記憶體空間。

vf_get_image()會嘗試使用direct rendering，這是透過呼叫下一個影像濾波器(get_image())取得記憶體空間。如果失敗將會回到一般記憶體取得方式。

3.6.2 影像播放流程

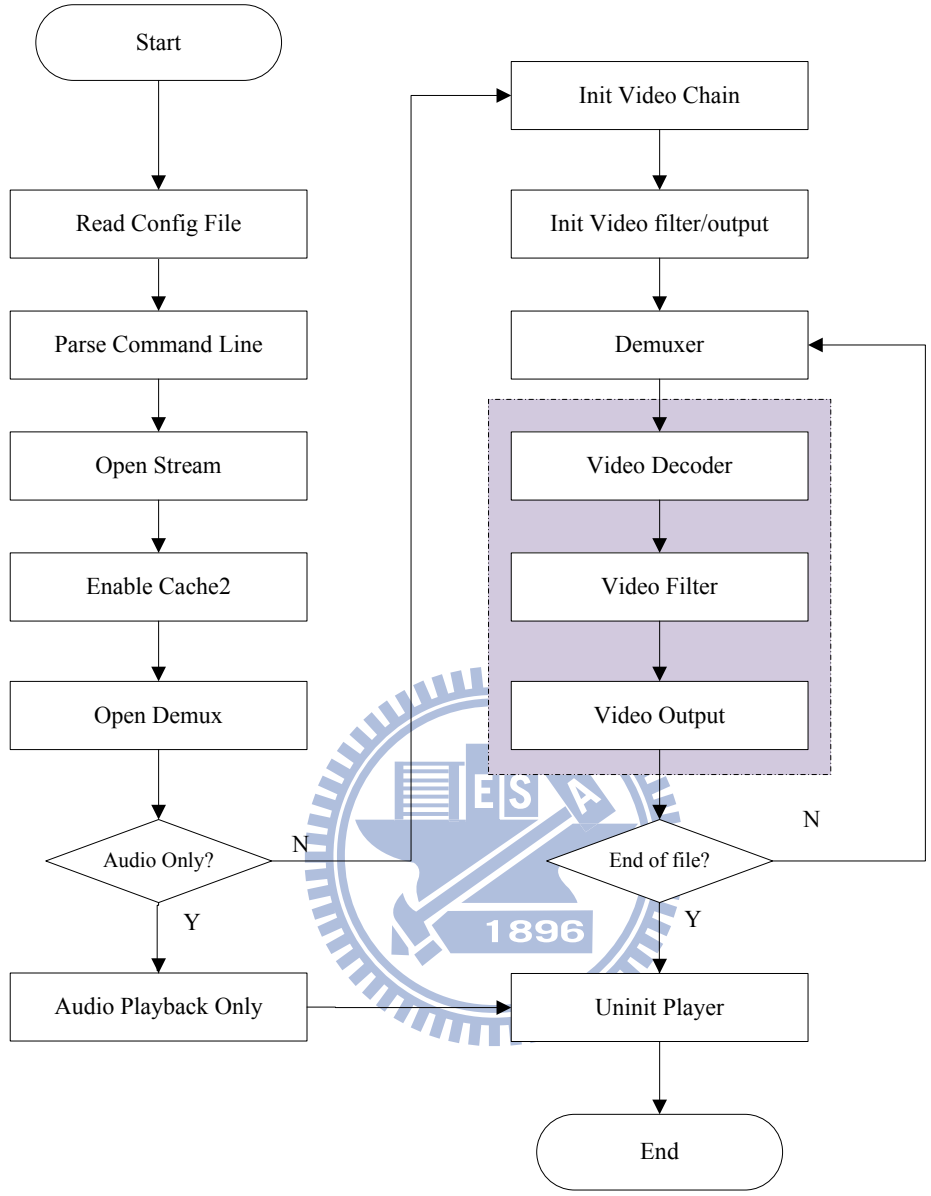

圖 3-39. 影像播放流程

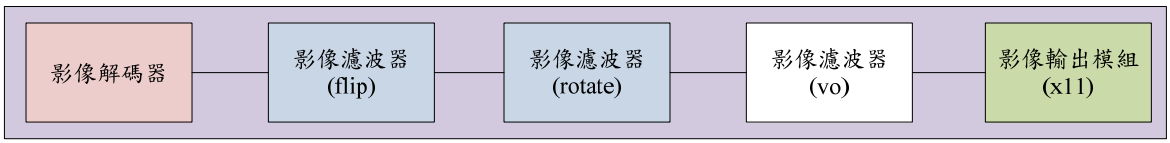

圖 3-40. Video Chain

圖3-39為MPlayer影像播放的流程，紫色區塊為video chain，我們將video chain中各模組間資料傳遞的流程詳加說明。假設已經建立的video chain如圖3-40。

1、當MPlayer要解碼一張video frame會將視訊的標頭資訊和video bitstream傳入先前已經選定的影像解碼器。

2、影像解碼器會使用適當的video codec將video bitstream解碼成一張frame存放到mpi結構中，並且回傳mpi結構指標。

3、MPlayer收到影像解碼器回傳的mpi結構指標，將這個mpi結構指標傳入影像濾波器(flip)的影像處理函式(put_image())。

4、影像濾波器(flip)將傳入的圖像資料做影像處理(上下翻轉)，將處理過後的圖像資料存放到另一個mpi結構中。

5、影像濾波器(flip)將處理過後的mpi結構傳到下一個影像濾波器(rotate)的影像處理函式(put_image())。

6、影像濾波器(rotate)將傳入的圖像資料做影像處理(順時針旋轉90度)，將處理過後的圖像資料存放到另一個mpi結構中。

7、影像濾波器(rotate)將處理過後的mpi結構傳到下一個影像濾波器(vo)的影像處理函式(put_image())。

8、影像濾波器(vo)將傳入的圖像資料，傳給影像輸出模組(x11)的影像轉換函式(draw_slice())。

9、影像輸出模組(x11)將傳入的圖像資料轉成XImage結構資料。

11、影像輸出模組繪製OSD(draw_osd())。

12、影像輸出模組檢查是否有鍵盤或滑鼠輸入訊息(check_event())。

13、影像輸出模組(x11)的影像輸出函式(flip_page())。此時x11影像輸出模組使用Xlib的XPutImage()函式，將XImage的圖像資料，透過X Window系統將畫面顯示到x11視窗上。

第四章　Client 端和 Server 端實作

這章我們分兩個部份，第一個部份為 client 端實作，包含 MPlayer rdp 影像輸出模組和 rdesktop 程式的 MPlayer_Control_Thread()和 MPlayer_Display_Thread()。當 MPlayer 播放影片時，將會透過 rdp 影像輸出模組，將影像資料寫到 fifo 檔案。然後透過 rdesktop 的 MPlayer_Display_Thread()，將影像資料從 fifo 讀取，再透過 X Lib 的 XPutImage()函式影像顯示到 rdesktop 的視窗畫面。MPlayer_Control_Thread()則是用來接收 Rdesktop Media Server 播放指令。

第二部份為 server 端實作，這是建立一個 Rdesktop Media Server。Rdesktop Media Server 是我們自行撰寫的程式，裡面包含了兩個伺服器，一個是 TCP Socket Server，主要是和 rdesktop 的 MPlayer_Control_Thread()建立連線和下達播放指令。另一個為 Web Server，提供影片傳送功能。這部份將說明 Rdesktop Media Server 如何被實現。

4.1 Rdesktop Plugin Output 模組

為了將 MPlayer 影像畫面顯示在 Rdesktop 的視窗，我們額外撰寫 Rdesktop Plugin Output 的影像輸出模組(簡稱 rdp 影像輸出模組)。在第 3.4 節我們已經詳細介紹影像輸出模組的運作，在此將說明如何實現 Rdesktop Plugin Output 模組。這個模組實現目的是要將 MPlayer 的影像嵌入 rdesktop 的畫面中，讓使用者可以直接在 rdesktop 的視窗畫面觀賞影片。

4.1.1 rdp 影像輸出模組實現

這邊將說明 rdp 影像輸出模組的程式碼，而將模組加入到 MPlayer 的步驟請參閱附錄一。rdp 影像輸出模組必須包含 vo_functions_t 結構所有的函式和變數，在此將說明這

些函式。首先是建立 rdp 影像輸出模組的 info 資訊，如圖 4-1。

```
static vo_info_t info =
{
    "rdesktop plugin output",          //name
    "rdp",                             //short_name
    "Ju-Hung Teng <kuwaktw@gmail.com>", //author
    ""                                 //comment
};

LIBVO_EXTERN(rdp)
```

圖 4-1. rdp 模組的 info 資訊

control()：

VOCTRL_GET_IMAGE

不使用直接渲染功能，圖像資料的記憶體空間，不和前面的影像濾波器共用，所以回傳 VO_FALSE。

VOCTRL_QUERY_FORMAT

我們使用的是 MPlayer 內建的 colorspace 轉換函式(swscale)，而這個函式支援 I420、IYUV 和 YV12。除了這些格式外，我們回傳 0，代表不支援。

```
static int control(uint32_t request, void *data, ...)
{
    switch (request) {

        case VOCTRL_GET_IMAGE:
            return VO_FALSE;

        case VOCTRL_QUERY_FORMAT:
            switch(*((uint32_t*)data)){
                case IMGFMT_I420:
                case IMGFMT_IYUV:
                case IMGFMT_YV12:
                    return VFCAP_CSP_SUPPORTED | VFCAP_SWSCALE | VFCAP_ACCEPT_STRIDE;
            }
            return 0;

    }
    return VO_NOTIMPL;
}
```

VOCTRL_GET_IMAGE 對應的程式碼

VOCTRL_QUERY_FORMAT 對應的程式碼

圖 4-2. rdp 模組的 control()函式

config()：

第 1，依影像的解析度和目前系統顯示環境建立 XImage 的結構。

```
static int config(uint32_t width, uint32_t height, uint32_t d_width, uint32_t d_height,
uint32_t flags, char *title, uint32_t format)
{
    XGetWindowAttributes(mDisplay, mRootWin, &attribs);
    depth = attribs.depth;

    if (!XMatchVisualInfo(mDisplay, mScreen, depth, DirectColor, &vinfo) ||
        vinfo.visualid != XVisualIDFromVisual(attribs.visual))
          XMatchVisualInfo(mDisplay, mScreen, depth, TrueColor, &vinfo);

    image_width = width;
    image_height = height;

    if (myximage)
    {
        Destroy_XImage();
        sws_freeContext(swsContext);
    }
    Create_XImage();
```

讀取顯示環境的參數

建立 XImage 結構的函式

圖 4-3. rdp 模組的 config()函式第 1 部分

```
static void Create_XImage(void)
{
    myximage = XCreateImage(mDisplay, vinfo.visual, depth, ZPixmap, 0,
                            NULL, image_width, image_height, 8, 0);
    myximage->data = malloc(myximage->bytes_per_line * image_height);
    memset(myximage->data, 0, myximage->bytes_per_line * image_height);
}
static void Destroy_XImage(void)
{
    XDestroyImage(myximage);
    myximage = NULL;
}
```

建立 XImage 結構

宣告 XImage 圖像資料的記憶體空間

刪除 XImage 結構

圖 4-4. 建立和刪除 XImage 函式

第 2，選定 colorspace 轉換函式，由來源和目標的 colorspace 利用查表方式選定轉換的函式。

第 3，開啟 fifo 檔案為寫入模式，將 XImage 的結構資料寫入 fifo 檔案中。

```
    in_format = format;
    out_format = fmte->mpfmt;
    bpp = myximage->bits_per_pixel;

    swsContext = sws_getContextFromCmdLine(width, height, in_format, width, height, out_format);

    if (!swsContext)
        return -1;

    if (fd==0) {
        fd = open(HALF_DUPLEX, O_WRONLY);
        printf("FIFO file open\n");
        first_sent = 0;
        write(fd, myximage, sizeof(*myximage));
    }

    return 0;
} ? end config ?
```

由來源 colorspace 和目標 colorspace 選定轉換函式

開啟 fifo 檔案，並且寫入 XImage 結構資料

圖 4-5. rdp 模組的 config()函式第 2、3 部分

draw_slice()：利用 config 選定的 colorspace 轉換函式，將解碼後的來源影像轉成 XImage 的圖像。例如將 planar I420 轉成 packed RGB32 格式。

```
static int draw_slice(uint8_t *src[], int stride[], int w,int h,int x,int y)
{
    uint8_t *dst[3];
    int dstStride[3];

    dstStride[1] = dstStride[2] = 0;
    dst[1] = dst[2] = NULL;

    dstStride[0] = image_width * ((bpp + 7) / 8);
    dst[0] = ImageData;
    sws_scale_ordered(swsContext, src, stride, y, h, dst, dstStride);

    return 0;
}
```

ImageData = myximage->data

colorspace 轉換函式

圖 4-6. rdp 模組的 draw_slice()函式

flip_page()：將 XImage 的結構資料和其所指向的圖像資料寫入 fifo 檔案。

```
static void flip_page(void)
{
    write(fd, myximage, sizeof(*myximage));
    write(fd, myximage->data, myximage->bytes_per_line * myximage->height);

    first_sent++;
}
```

寫入 XImage 的結構資料

寫入 XImage 的圖像資料

圖 4-7. rdp 模組的 flip_page()函式

uninit()：將 XImage 的圖片寬度設定為 0，並將所指向的圖像資料寫入 fifo。解除 XImage 的記憶體空間。

```
static void uninit(void)
{
    int temp;

    if (!myximage) {
        printf("myximage doesn't exist!\n");
        return;
    }

    if (fd!=0) {
        temp = myximage->width;

        myximage->width = 0;

        write(fd, myximage, sizeof(*myximage));
        write(fd, myximage->data, myximage->bytes_per_line * myximage->height);

        myximage->width = temp;
        close(fd);

        first_sent = 0;
        printf("FIFO file close\n");

    }
    Destroy_XImage();
    sws_freeContext(swsContext);

} ? end uninit ?
```

將 XImage 的圖像寬度改為 0

將 XImage 結構和圖像資料寫入 fifo

圖 4-8. rdp 模組的 uninit()函式

draw_osd()、draw_frame()、check_events()：這些函式我們沒有使用，但這是 vo_function_t 結構的必要函式。所以我們保留函式但是不撰寫任何程式碼在函式中。

```
static void draw_osd(void) {
}

static int draw_frame(uint8_t *src[]) {
    return 0;
}

static void check_events(void) {
}
```

圖 4-9. rdp 模組的 draw_osd()、draw_frame()、check_events()函式

4.1.2 rdp 影像輸出模組流程

圖 4-10 為 rdp 影像輸出模組函式執行的順序。圖 4-11 為 rdp 模組各個函式寫入 fifo 的資料。config()做了相關初始化後，建立 XImage 結構並寫入到 fifo 檔案。draw_slice()將 planar YUV 格式轉換成 packed RGB 格式。draw_osd()和 check_events()沒有任何操作。flip_page()將 XImage 結構和圖像資料一併寫入 fifo 檔。當影片結束，uninit()將 XImage 結構的圖像寬度改為 0，將 XImage 結構和圖像資料一併寫入 fifo 檔。

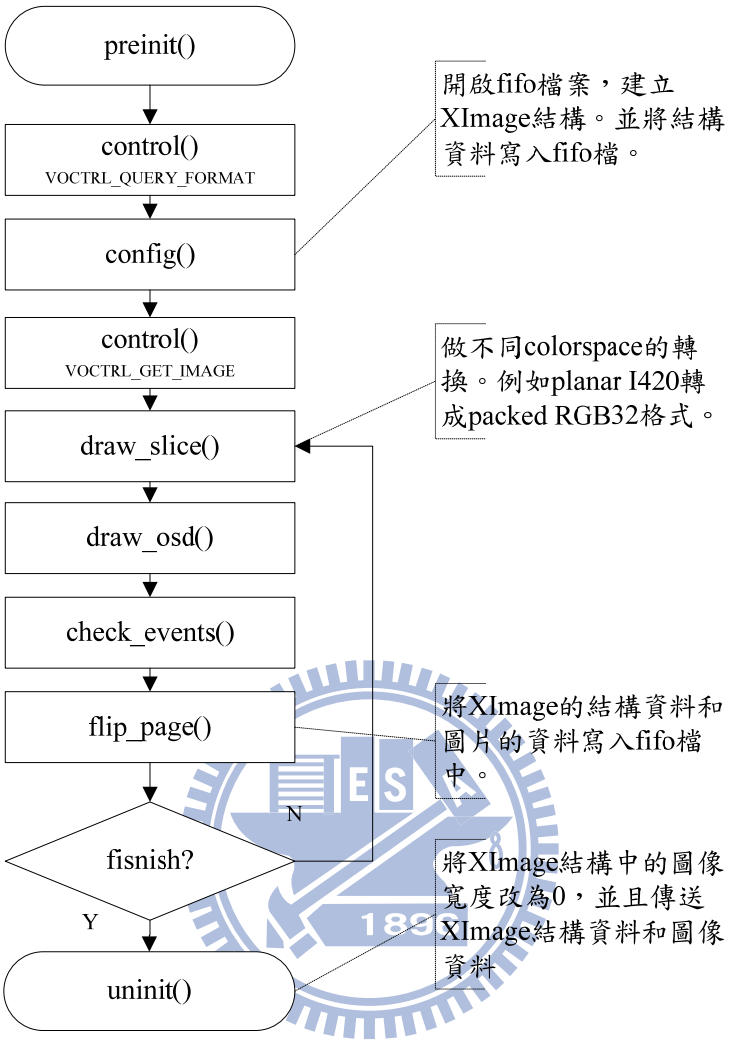

圖 4-10. rdp 影像輸出模組流程

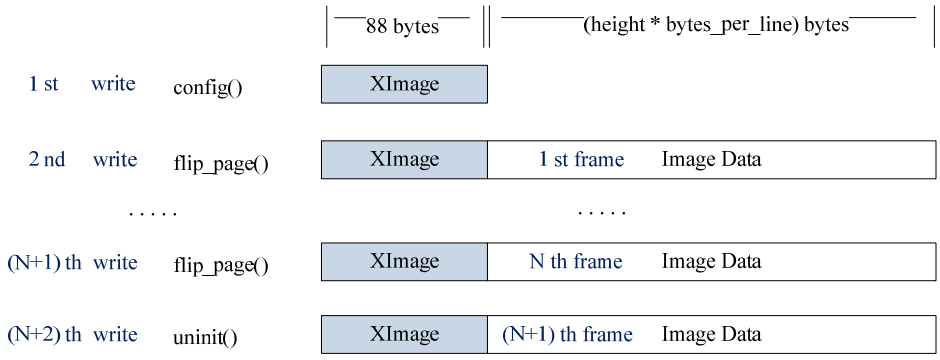

圖 4-11. 寫入資料到 fifo

4.2 Rdesktop 程式

rdesktop 是一個開放原始碼的 RDP Client，其操作平台在 unix、linux 作業系統下，主要是參照 RDP 協定經過不斷嘗試，進行反組譯完成的程式。由於是開放原始碼，我們可以了解程式的運作並且改進，這邊要更改的目的有兩個。第一是建立 MPlayer_Control_Thread()，用來接收 Rdesktop Media Server 所傳來的播放影片的指令。第二就是建立 MPlayer_Display_Thread()，是用於讀取 MPlayer 影像輸出模組寫入到 fifo 檔的影像，並且顯示在 rdesktop 的子視窗中。

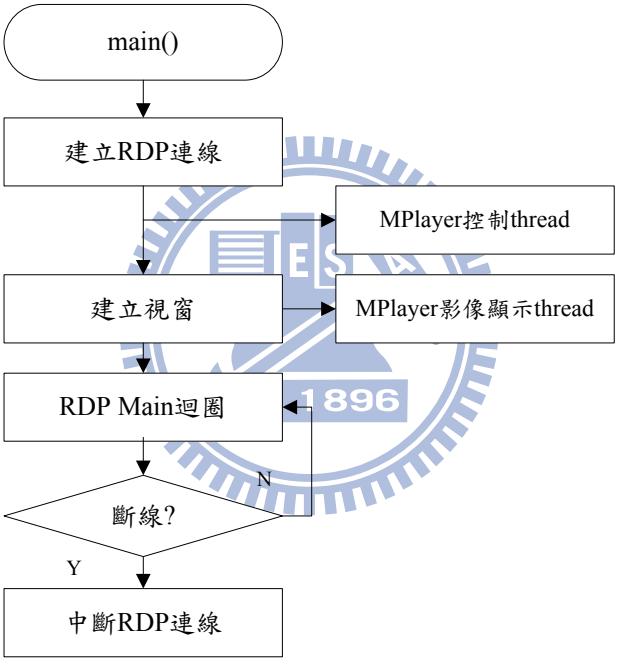

圖 4-12. rdesktop 的主要流程

main()函式前面會根據使用者的設定值和 RDP Server 進行協議，rdp_connect()連線完成後，就會呼叫 ui_create_window()建立視窗。rdp_main_loop()只有呼叫 rdp_loop()，並且是持續用迴圈呼叫。除非 rdp 連線中斷，才會跳出這個迴圈。這個 rdp_main_loop()的部分我們不做更動，而是在其他地方加入 thread 程式。MPlayer_Control_Thread()是加在 rdp_connect()之後，這是為了確保成功連到 RDP Server 時才做連線。而 MPlayer_Display_Thread()則是建立在 ui_create_window()函式返回之前，這是因為建立

子視窗需要 rdesktop 母視窗的 ID 參數。圖 4-12 中分叉點是同時執行，當 thread 建立後，主程式繼續進行。

4.2.1 MPlayer_Control_Thread()函式

這個 thread 建立一個 TCP 連線到 Rdesktop Media Server，並且等待播放指令。thread 執行時，main()會傳入伺服器的位址，而這個位址可能是網域名稱。所以利用 gethostbyname()函式將伺服器網域名稱轉換成 IP 位址。利用 IP 位址建立 TCP 連線，連線成功就會進入 read()讀取指令。因為是 blocking 模式，所以只要伺服器沒有送資料過來，thread 就會被 block 在 read()。當有收到伺服器傳來的資料，會判斷是否為播放指令，否則回到 read()等待下個資料。當收到播放指令後，呼叫 MPlayer 播放 Rdesktop Media Server 的影片。如果 TCP 連線中斷，read()會回傳 0。此時會關閉 socket，等待 3 秒後重新建立連線。

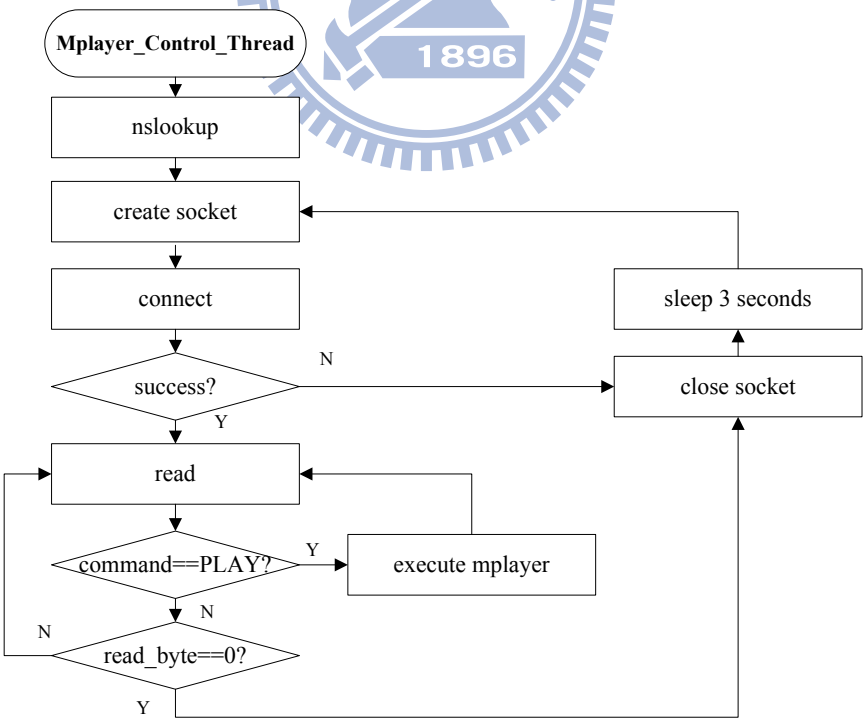

圖 4-13. MPlayer_Control_Thread()流程圖

4.2.2 MPlayer_Display_Thread()函式

Rdesktop 的視窗開啟是在 ui_create_window()，所以這邊會得到一個 rdesktop 主畫面的視窗 ID。而在開子視窗時需要主視窗 ID 這個參數。

thread 開啟時，先建立一個 fifo 檔案，並且開啟為讀取模式。然後讀取 fifo 檔案，我們使用的 fifo 是 blocking 模式，所以會等待 MPlayer 傳來的資料。當 MPlayer 播放影片前，vo_rdp.c::config()會寫入 XImage 的結構資料到 fifo。如此我們可以從 fifo 檔案讀取 XImage 的結構資料。這結構資料有圖像的訊息，將其中的 bytes_per_line*height 可以計算出圖像所佔用的記憶體空間。

再來從 fifo 檔讀取 XImage 結構資料和 XImage 的圖像資料。讀取完後，將 XImage 結構資料中的 data 位址改成接收 XImage 圖像資料的位址。這時候利用主視窗 ID 建立一個子視窗，再利用 XPutImage()將圖像資料貼到子視窗上。完成後回到讀取 fifo 檔，再接收一次資料，更改 XImage 結構資料 data 的位址，再貼圖到子視窗上，一直反復到影片結束為止。

當影片結束時，vo_rdp.c::unint()會將 XImage 結構的 width 資料更改為 0 並寫入到 fifo 檔。此時讀取到 width 為 0 的 XImage 資料，就會進行關閉子視窗的程序，並且釋放圖像的記憶體空間。結束後，程式回到第一次讀取 XImage 結構資料的地方，等待下一個影片播放。

圖 4-14 為從 fifo 讀取資料的順序，第一次是讀取 XImage 的結構資料為 88bytes，這時候從結構資料計算出一張圖像所需要的記憶體資訊，並且宣告一張圖像資料和一個 XImage 結構所要的記憶體空間。XImage 結構大小為 88 bytes，所以需要的記憶體空間為(88 + height*bytes_per_line) bytes。第二次開始，讀取的資料為 XImage 結構和圖像資料，如此讀取直到影片結束。如果影片有 N 張 frame，那最後一次讀取會是第(N+1)張 frame，這張將是無效的，此時我們僅要取得 XImage 結構資料的 width 為 0 的資訊。

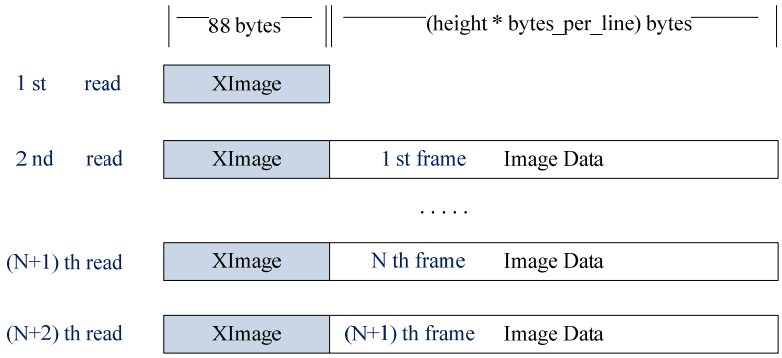

圖 4-14. 從 fifo 讀取資料

圖 4-15 為 MPlayer_Display_Thread()的流程圖，主要是由兩個迴圈構成。藍色大迴圈是影片開始播放與結束的流程，綠色小迴圈為播放影片中的貼圖流程。

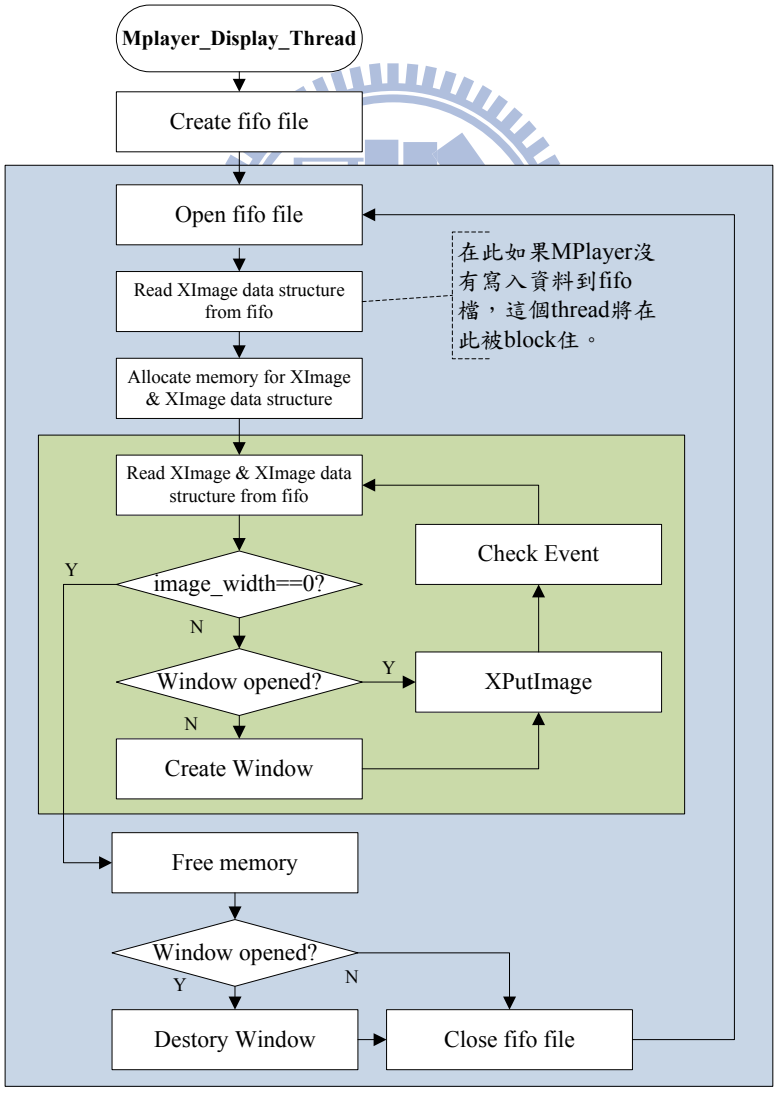

圖 4-15. MPlayer_Display_Thread()流程圖

4.3 Rdesktop Media Server

Rdesktop Media Server 是我們自行撰寫的程式，裡面包含了兩個伺服器，一個是 TCP Socket Server，主要是和 rdesktop 的 MPlayer_Control_Thread()建立連線和下達播放指令。另一個為 Web Server，提供影片傳送功能。這部份將說明 Rdesktop Media Server 如何被實現。

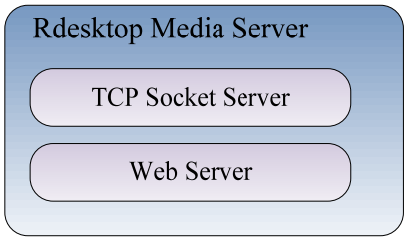

圖 4-16. Rdesktop Media Server

4.3.1 TCP Socket Server

我們利用 Borland C++ Builder 6 (BCB6)內建的 Internet 的 ServerSocket 元件完成 TCP Socket Server。圖 4-17 為 ServerSocket 元件的圖示。

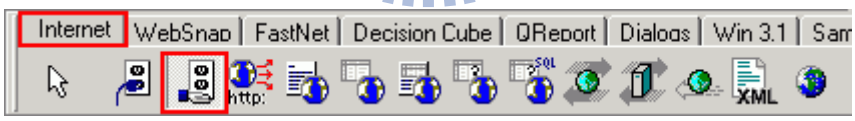

圖 4-17. ServerSocket 元件

將屬性(Properties)中的 Port 設定為 668，這個 Port 就是 TCP Socket Server 對外的處理窗口。利用事件(Events)中的 OnClientConnect 和 OnClientDisconnect 判斷 Server 是否有和 Client 建立連線。有建立連線才可以對 Client 傳送播放影片的指令。另外我們使用 Win32 中的 RichEdit 顯示 log 資訊，將程式的狀態和訊息顯示在上面。

4.3.2 Web Server

我們利用 BCB6 內建的 Indy 的 IdHTTPServer 元件完成 Web Server。圖 4-18 為 IdHTTPServer 元件的圖示。

圖 4-18. IdHTTPServer 元件

將屬性中的 Port 設定為 667，這個 Port 就是 Web Server 對外的處理窗口。利用事件中的 OnConnect 和 OnDisconnect 判斷是否有 Client 連線進來。最重要的事件是 OnCommandGet，當 Client 有個 GET 的請求訊息，會觸動這個事件連結的函式。這個函式處理請求檔案的資料傳送，包含中斷點下載。中斷點下載的功能，決定了 MPlayer 是否可以對檔案進行搜索，也就是快轉和倒轉的功能。

RFC2616 定義 HTTP 1.1 的請求訊息有八種方法：OPTIONS、GET、HEAD、POST、PUT、DELTETE、TRACE 和 CONNECT。常用的四種方法為 GET、POST、PUT 和 DELTETE。[13]

1. OPTIONS：和 Client 說明網頁資源的需求和選項。
2. GET：從 Server 取得資源。
3. HEAD：和 GET 相同，只是 Server 僅回傳標頭資訊而不回傳網頁資源。
4. POST：在 Server 上建立網頁資源。
5. PUT：更新 Server 上的網頁資源。
6. DELTETE：刪除 Server 上的網頁資源。
7. TRACE：將 Client 的請求從 Server 回傳給 Client。
8. CONNECT：保留給 proxy 使用。

IdHTTPServer 元件包含 OnCommandGet 和 OnCommandOther 的請求事件，處理 GET 方法為 OnCommandGet 事件，其他方法則用 OnCommandOther 事件。

1. OnCommandGet 事件

OnCommandGet 事件為 Client 發出 GET 的 HTTP 請求訊息時所觸發的事件。圖 4-19 為 OnCommandGet 事件流程，首先判斷請求檔案是否為“/Video”。如果否就回傳 404 代碼，表示檔案不存在。是的話，檢查是否有請求中斷點下載，有的話回傳 206 訊息，並僅傳送該範圍的檔案資料。沒有請求中斷點下載，則回傳 200 訊息，並傳送整個檔案。

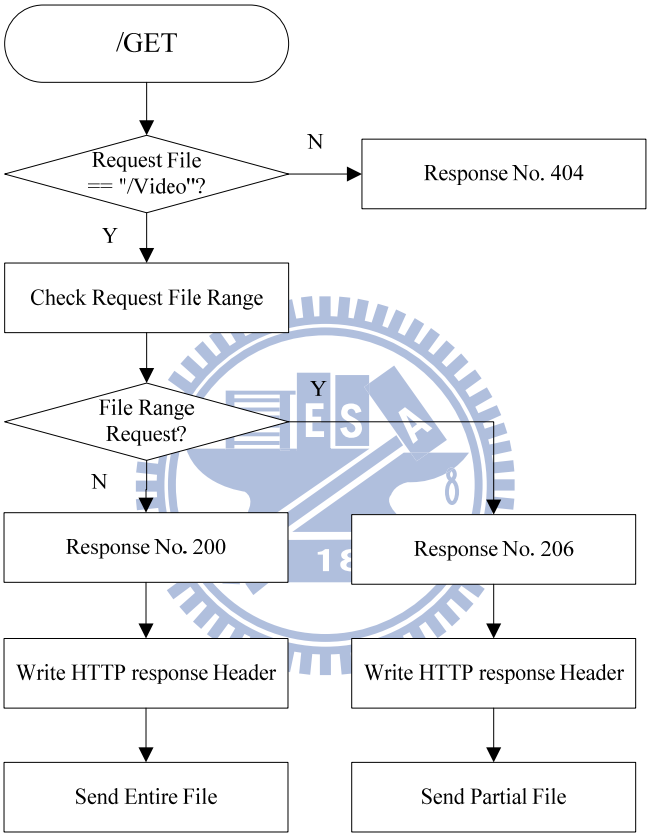

圖 4-19. OnCommandGet 事件流程

2. 請求訊息

圖 4-20 為 MPlayer 請求檔案的標頭訊息；圖 4-21 為請求部份檔案的標頭訊息，兩者差別在於部份檔案多了“Range: bytes=10240-”。 “Range: bytes=10240-”的意思為請求 Web Server 從檔案的第 10240 位元組開始傳送。注意的是檔案的最開頭位置是從第 0 個 bytes 開始算。

```
GET /Video HTTP/1.0
Host: 140.113.13.81:667
User-Agent: MPlayer/1.0rc2-4.1.2
Icy-MetaData: 1
Connection: close
```

圖 4-20. MPlayer 請求檔案

```
GET /Video HTTP/1.0
Host: 140.113.13.81:667
User-Agent: MPlayer/1.0rc2-4.1.2
Icy-MetaData: 1
Range: bytes=10240-
Connection: close
```

圖 4-21. MPlayer 請求部份檔案

BCB6 中內建的 Indy IdHTTPServer 元件，沒有解析檔案範圍的功能，所以要自行用字串比對的方式將檔案範圍的請求找出來。然後在傳送檔案的時候，要將檔案讀取位置跳到檔案請求的起始位置。

3. 回應訊息

回應的標頭訊息中，對於 MPlayer 來說最重要的參數為 Response No.、Accept-Ranges、Content-Type 傳送 Content-Length 和 Content-Range。圖 4-22 為回應 200 代碼的標頭訊息；圖 4-23 為回應 206 代碼的標頭訊息；圖 4-24 為回應 404 代碼的標頭訊息。

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Server: Rdesktop Media Server
Content-Type: application/octet-stream
Content-Length: 44176206
```

圖 4-22. Web Server 200 回覆訊息

圖 4-22 為回應 200 代碼的標頭訊息。Accept-Ranges 決定中斷點下載與否；Content-Type 為檔案的 MIME 型態，這邊都是回應“application/octet-stream”。Content-Length 為傳送的檔案大小。

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 12582912-44176205/44176206
Accept-Ranges: bytes
Server: Rdesktop Media Server
Content-Type: application/octet-stream
Content-Length: 31593294
```

圖 4-23. Web Server 206 回覆訊息

圖 4-23 為回應 206 代碼的標頭訊息。當接收中斷下載的請求，回應代碼要改為 206，並且利用 Content-Range 的欄位說明檔案的範圍。Content-Range “bytes 12582912-44176205/44176206”意思為目前要傳送的檔案為第 12582912 位元組到 44176205 位元組/檔案全部大小為 44176206 位元組。此時回應 Content-Length 的值為該部分檔案的大小。

```
HTTP/1.1 404
Server: Rdesktop Media Server
Content-Type: text/html
```

圖 4-24. Web Server 404 回覆訊息

圖 4-24 為回應 404 代碼的標頭訊息。這是代表請求的檔案不存在，所以也不會有任何檔案描述的訊息。

4. 檔案傳送

不論是傳送完整檔案或部份檔案，都是將檔案切成 BLOCKSIZE 的大小傳送。一個原因是 IdHTTPServer 元件只提供傳送整個檔案的函式，沒有傳送部份檔案的函式。另一個原因是可以將檔案切成 TCP 封包的最大值(MSS)傳送。

從硬碟開啟檔案，更改檔案讀取的位置。宣告一個記憶體空間，從檔案位置讀取 BLOCKSIZE 的資料到記憶體。透過 TCP 將記憶體資料傳送給 Client，傳送完成回到讀取檔案，再透過 TCP 傳送，直到檔案傳送完畢。 圖 4-25 為傳送檔案的流程。

BCB6 程式細節。首先利用 TFileStream 將檔案開啟成唯讀模式，更改檔案讀取的位置。建立一個 TMemoryStream 的資料，利用 CopyFrom()函式從 TFileStream 讀取資料到 TMemoryStream，讀取的大小為 BLOCKSIZE。將 ResponseInfo 結構中的 ContentStream 指定為 TMemoryStream 的資料，再利用 WriteContent()指令傳送。使用 ContentStream 傳送資料，將會把原本傳送的資料釋放掉，所以進行下一個區段傳送要再建立一個 TMemoryStream。如此反覆傳送資料，直到整個檔案傳送結束。

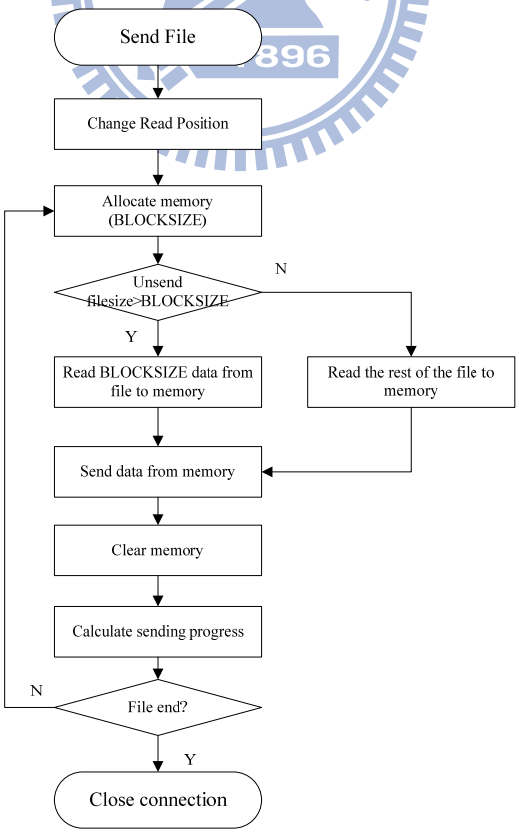

圖 4-25. 檔案傳送流程

目前可以被 Ethernet 接受的 MTU(Maxumum Transmission Unit)最大值為 1500bytes。扣除 IP 標頭 20bytes 和 TCP 標頭 20bytes，最大 TCP 可以傳送的資料(MSS)為 1460bytes。所以 BLOCKSIZE 的值選擇 1460bytes。

傳送速度是利用 1 個 Timer 元件，計算 1 秒內總共傳送幾個 BLOCKSIZE 的資料，再換算成每秒傳輸的位元組數。而傳送進度是用 Progress Bar 顯示目前檔案位置的百分比。

4.3.3 運作流程

MPlayer 支援*.avi *.rm *.rmvb *.mp3 *.mpg *.mpeg *.mov *.wav *.wmv 檔案格式，所以將 Rdesktop Media Server 和這些副檔名建立檔案關連。當使用者點選已經建立檔案關連的檔案，遠端的作業系統直接會呼叫 Rdesktop Media Server 程式，並且傳入檔案位址。此時 Rdesktop Media Server 會將檔案開啟，並等待 client 連線。圖 4-26. Rdesktop Media Server 運作流程。我們將這運作流程分三個部分。

第 1 部分：

使用者在遠端桌面點選這些檔案時，作業系統會呼叫 Rdesktop Media Server 並且傳入檔案路徑。同一時間我們只准許執行一個 Rdesktop Media Server，這是因為當使用者點選第二個影片的時候，第一個 RMS 已經將 port 佔用了，第二個開啟的會出現問題。

當第一次執行 Rdesktop Media Server 我們會建立一個 Mutex，當第二個 Rdesktop Media Server 開啟的時候先檢查 Mutex 是否存在。如果 Mutex 存在，代表 Rdesktop Media Server 已經在執行。當程式已經開啟，而使用者點選檔案，作業系統會開啟第二個 Rdesktop Media Server 並傳入檔案路徑。此時利用 FindWindow()去找前一個已經開啟 Rdesktop Media Server 的 Handle，透過 SendMessage()將檔案路徑傳給第一個 Rdesktop Media Server 的 Handle 去處理，而第二個 Rdesktop Media Server 中止程式。

第 2 部分：

第一個 Rdesktop Media Server 透過攔截消息，將第二個 Rdesktop Media Server 傳來的檔案路徑抓出來。並且判斷 TCP Socket Server 是否有和 Client 建立連線，已經建立連線才傳送播放指令給 Client (rdesktop::MPlayer_Control_Thread())。

第 3 部分：

Web Server 等待 Client (MPlayer::Open_stream())連線。當 Client 連線並且傳送請求訊息，Web Server 會回應 Client 的請求並且傳送檔案。當檔案傳送完成，Client 和 Web Server 中斷連線，此時 Rdesktop Media Server 等待使用者開啟下一個影片。

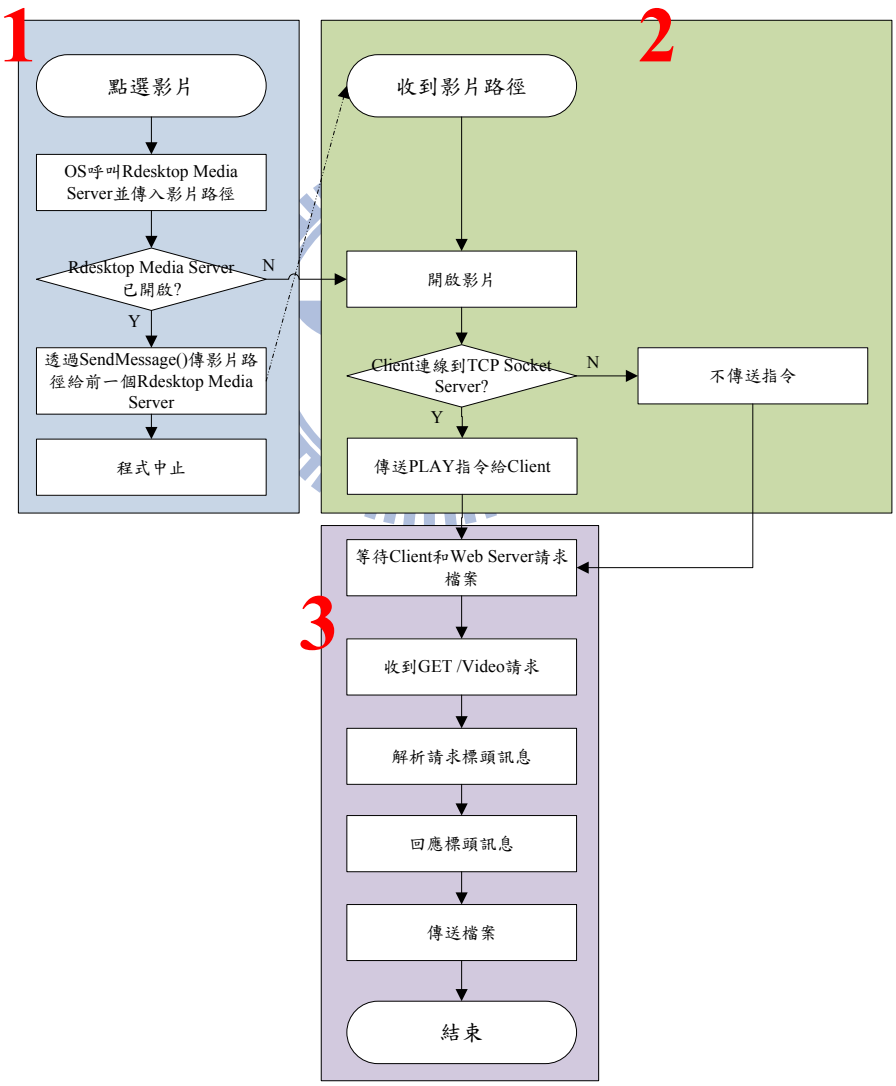

圖 4-26. Rdesktop Media Server 運作流程

4.3.4 程式畫面

圖 4-27 為 Rdesktop Media Server 執行的畫面，這是沒有開啟任何檔案的狀態。開啟檔案的按鈕只有在檔案沒有傳送的時候才可以按；播放檔案的按鈕則是有開啟檔案而且 TCP Socket Server 有連線才可以按。如圖還有傳送速度和傳送檔案進度顯示。log 資訊記錄：開啟檔案、Web Server & TCP Socket Server 啟動、TCP Socket Client 連線、Web Server Client 連線、Client HTTP 請求訊息和 PLAY 指令的傳送。

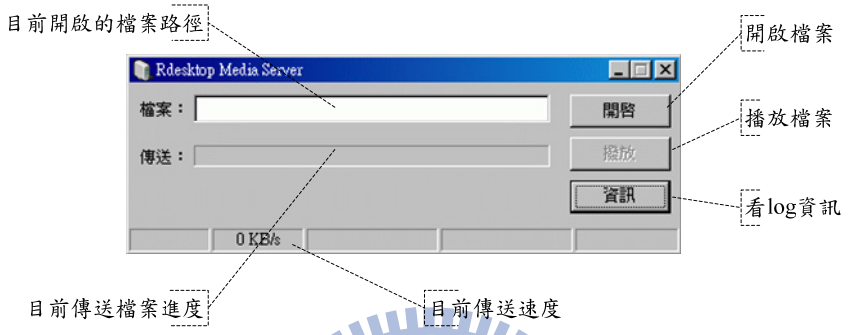

圖 4-27. Rdesktop Media Server 程式畫面

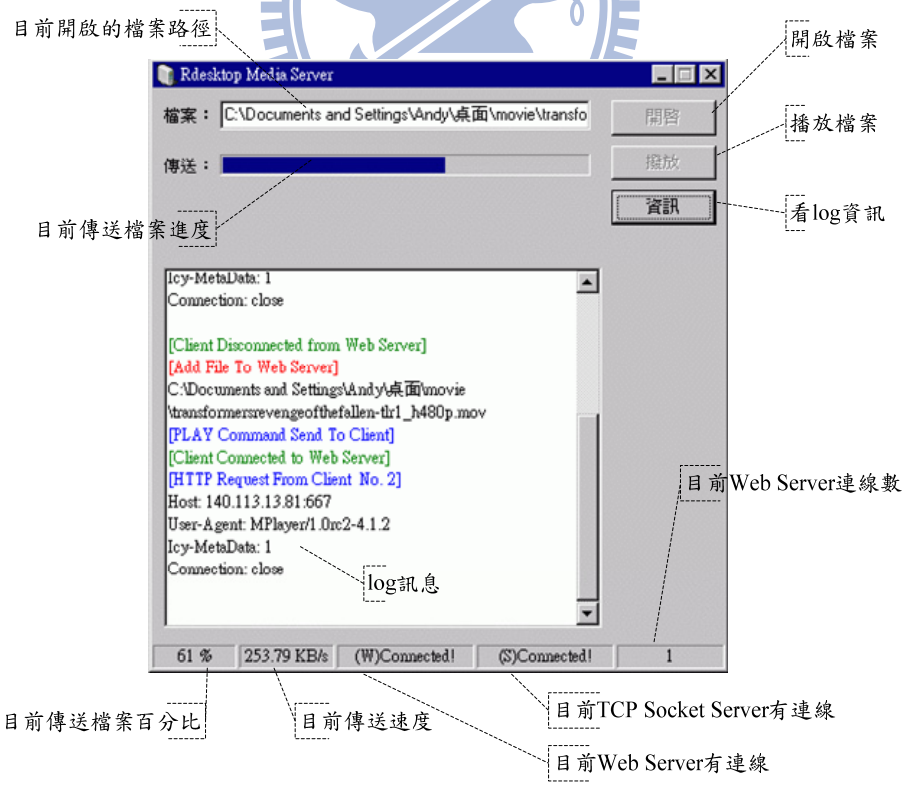

圖 4-28. Rdesktop Media Server 程式畫面(傳送檔案)

圖 4-28 為 Rdesktop Media Server 傳送檔案的畫面。當 Web Server 或 TCP Socket Server 有連線的時候，會顯示在下面的狀態列。而狀態列的最右下角是顯示目前和 Web Server 連線的數目。

4.4 系統流程

圖 4-29 為整個系統的連線流程。左邊為 Client 端，包含了 Rdesktop 和 MPlayer。右邊為 Server 端，包含了遠端桌面服務(Remote Desktop Services)和 Rdesktop Media Server。Client 和 Server 間是透過網際網路傳輸。

整個操作流程分七個步驟，第一步為使用者登入遠端桌面服務，第二步為 rdesktop 建立接收播放指令的 TCP 連線。這兩個步驟完成後，使用者將遠端桌面服務當成一般電腦操作，可以執行 Office 或是使用 IE 瀏覽網頁。當使用者點選影片的時候，第三步至第七步為影片播放的步驟。

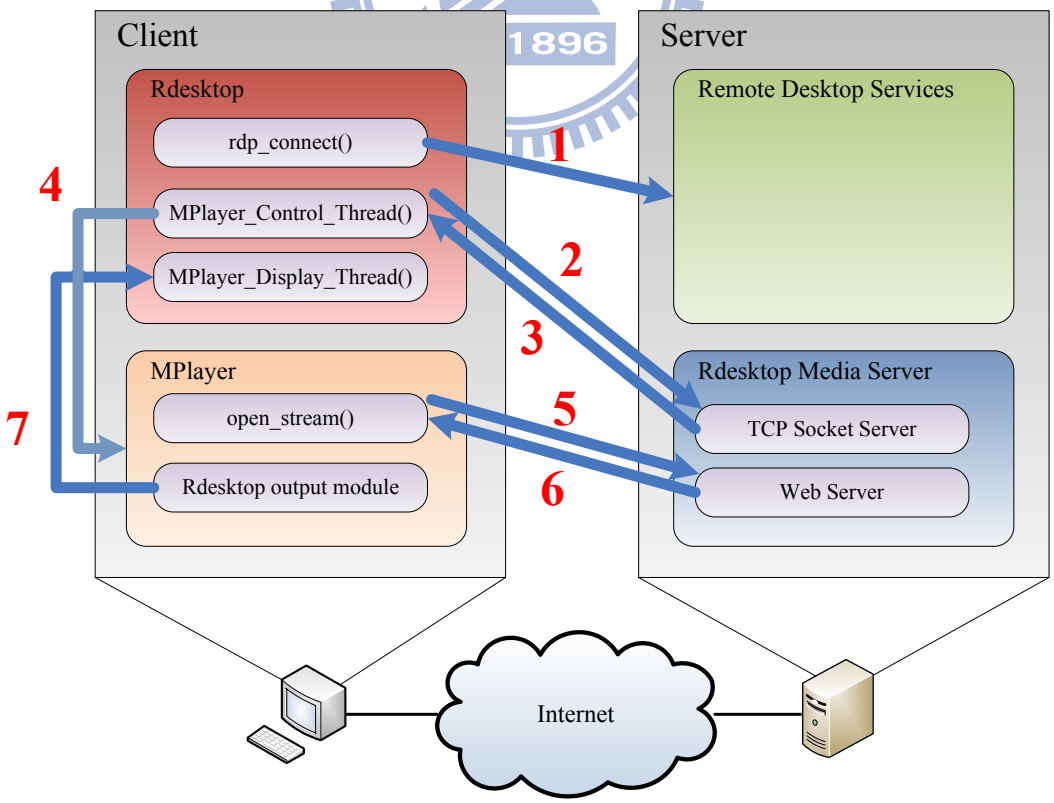

圖 4-29. 連線流程

a.使用者登入遠端桌面服務：

第 1 步：使用者在 Client 端透過 rdesktop 程式連線到 Server 的遠端桌面服務(Remote Desktop Services)。

第 2 步：MPlayer_Control_Thread()透過 TCP Socket 連線到遠端的 Rdesktop Media Server 裡的 TCP Socket Server。

b.使用者點選影片播放：

第 3 步：當使用者點選影片的時候，遠端的作業系統會呼叫 Rdesktop Media Server 程式，並帶有影片的檔案路徑。Rdesktop Media Server 會將影片放到 Web Server，並且傳送 PLAY 的指令到 Client 端。

第 4 步：MPlayer_Control_Thread()收到指令後，啟動 MPlayer 程式，並指定播放網址和影像輸出模組(Video Output Module)。

第 5 步：MPlayer 啟動後，透過 HTTP 開檔案模組，連線到 Rdesktop Media Server 的 Web Server，並送出 Request。

第 6 步：Web Server 收到 MPlayer 的 Request 後，解析 Request 的訊息，回傳 Response 並且開始傳送影音串流。

第 7 步：MPlayer 收到影音串流並解碼完成，將影像輸出到 Rdesktop Output Module。Rdesktop Output Module 再透過寫入 fifo 檔案將影像傳送到 rdesktop 的 MPlayer_Control_Thread()。MPlayer_Control_Thread()最後將影像畫面顯示到 rdesktop 的視窗上。

c.影片播放結束：

當影片快要結束時，MPlayer 和 Web Server 會先中斷連線，這是因為 MPlayer 有預先儲存資料在 Cache2。當 Cache2 資料播放完，影片結束，MPlayer 進入關閉程序。MPlayer 通知 MPlayer_Display_Thread()關閉 rdesktop 的子視窗。此時 rdesktop 恢復沒有播放影片的狀態。當使用者再次點選影片，才會再從第 3 步開始。

第五章　實驗結果

本章分三個部份，第一部份為擷取播放影片時 Client 和 Server 間傳送的封包，並且了解 TCP 的流量控制。第二部份為 VNC、RDP 遠端桌面程式和本研究改進的程式之間流量的比較。第三部分為將程式移植到 ARM 的嵌入式平台的執行結果。

5.1 TCP 封包分析

我們利用 Wireshark 軟體分析 Server 端的封包，並且擷取 Server 端的 Rdesktop Media Server 和 Client 端的 rdesktop、MPlayer 間溝通的訊息。最後剖析 MPlayer 和 Rdesktop Media Server Web Server 間的 TCP 流量控制。實驗環境如下：

封包擷取軟體：Wireshark 1.0.3
連線桌面大小：1024*768 pixels
bits per pixel：16 bpp
RDP Server：Microsoft Windows XP SP3，RDP 5.1
RDP Client：Linux Fedora 8，rdesktop-1.6.0
Media Server：Rdesktop Media Server
Media Player：MPlayer 1.0rc2-4.1.2

5.1.1 播放影片封包分析

1. Rdesktop Media Server 傳送 PLAY 指令

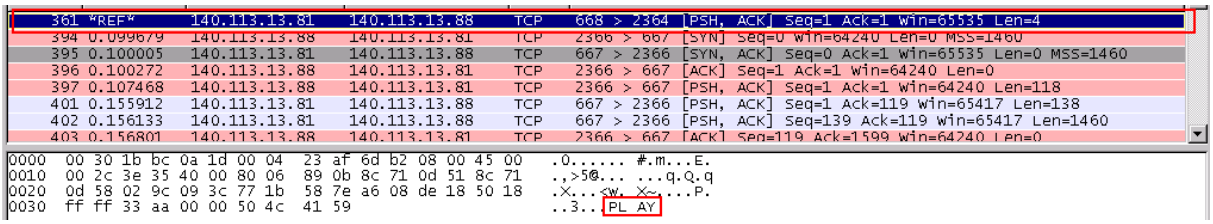

圖 5-1. Rdesktop Media Server 傳送 PLAY 指令

當使用者播放影片，Server 端 Rdesktop Media Server 的 TCP Socket Server 傳送播放的指令給 Client 端 rdesktop 的 MPlayer_Control_Thread()。MPlayer_Control_Thread()收到播放指令後，呼叫 MPlayer 並且指定檔案位址。

2. MPlayer 和 Rdesktop Media Server 建立 TCP 連線

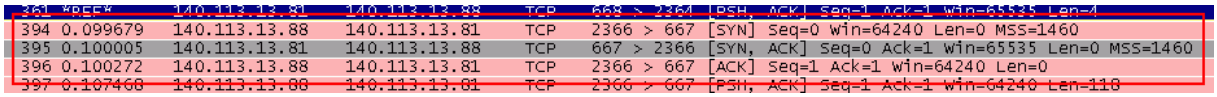

圖 5-2. MPlayer 和 Rdesktop Media Server 建立 TCP 連線

Client 端的 MPlayer 的 HTTP 開檔模組和 Server 端 Rdesktop Media Server 的 Web Server 利用三向交握建立 TCP 連線。這個 TCP 連線是用來傳送 HTTP 請求/回應/檔案資料。

3. MPlayer 送出 HTTP 請求訊息

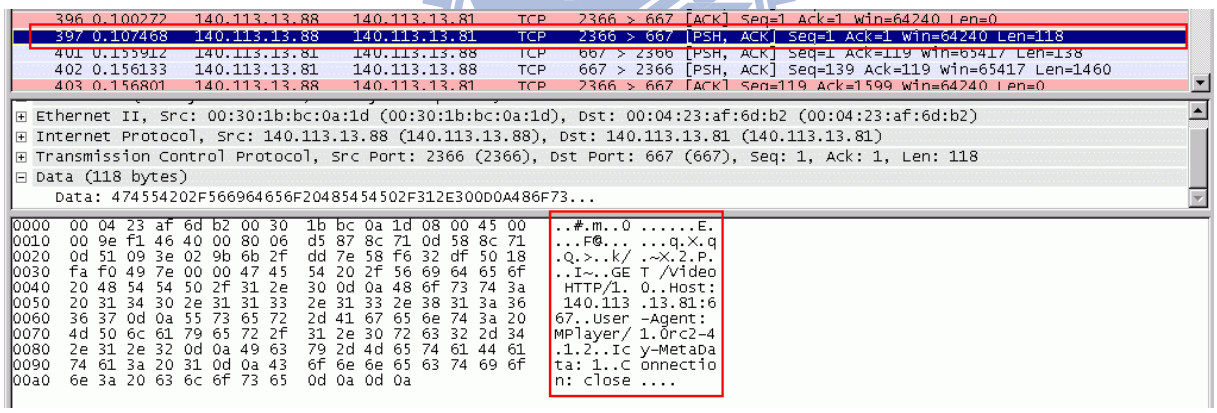

圖 5-3. MPlayer 送出 HTTP 請求訊息

當 TCP 連線建立完成，Client 端 MPlayer 的 HTTP 開檔模組發送 HTTP 請求訊息給 Server 端 Rdesktop Media Server 的 Web Server。

4. Rdesktop Media Server 回傳 HTTP 回應訊息

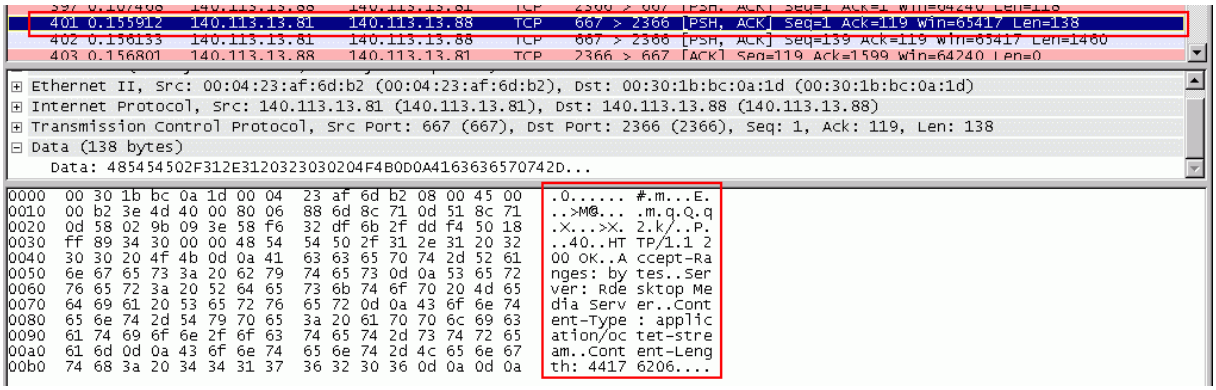

圖 5-4. Rdesktop Media Server 回傳 HTTP 回應訊息

Rdesktop Media Server 的 Web Server 收到請求訊息後，回應訊息給 Client 端 MPlayer 的 HTTP 開檔模組。

5. Rdesktop Media Server 開始傳送影片資料

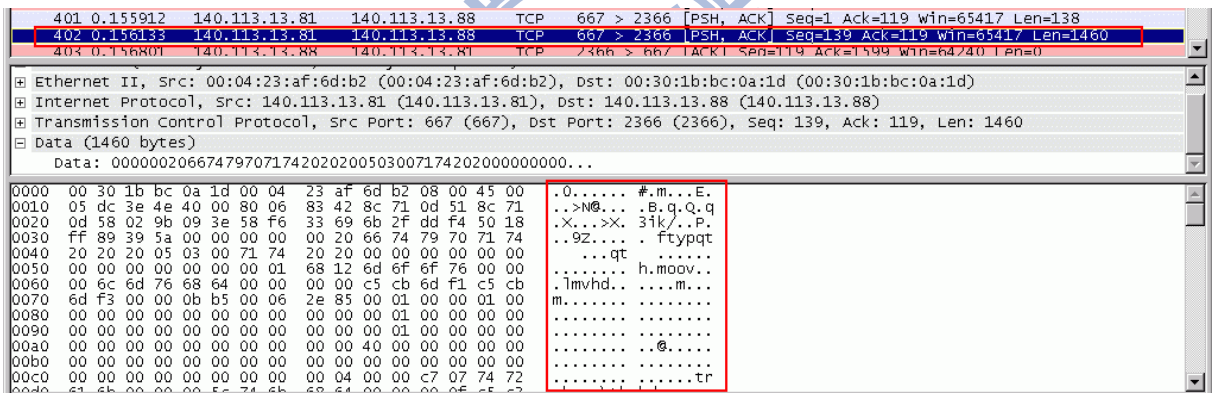

圖 5-5. Rdesktop Media Server 開始傳送影片資料

Server 端 Rdesktop Media Server 的 Web Server 開始傳送影片資料。

5.1.2 播放影片流量控制

MPlayer 播放影片，需要穩定的資料。而 Web Server 是利用 best effort 的方式傳送影片資料。MPlayer 的 Cache2 大小有限，這邊的重點是觀察 MPlayer 如何利用 TCP window size 做流量控制。

Web Server 傳送資料時，將待傳送的資料放到 send buffer。透過網路將 send buffer 的資料傳送到接收端。接收端收到資料後，將資料放到 receive buffer 並且回 ACK 訊息

到傳送端。MPlayer 的 Cache2 再到 receive buffer 取得影片資料。

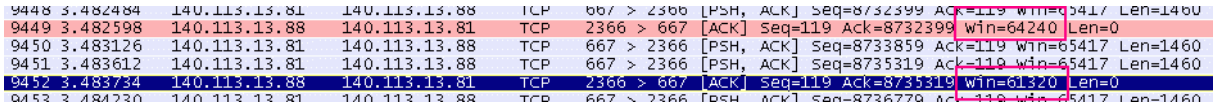

圖 5-6. Receiver window size 變小

如果 Cache2 停止填充資料，代表不去 receive buffer 收取資料。最後 receive buffer 將越來越小，此時作業系統持續從傳送端接收 TCP 封包，並在回 ACK 時將 window size 變小。傳送端收到接收端的 ACK 訊息，得知接收端的 window size 變小，如圖 5-6。當 receive buffer 滿了，會回 ZeroWindow 的訊息到傳送端，如圖 5-7。此時傳送端收到 ACK 訊息後，停止傳送資料。

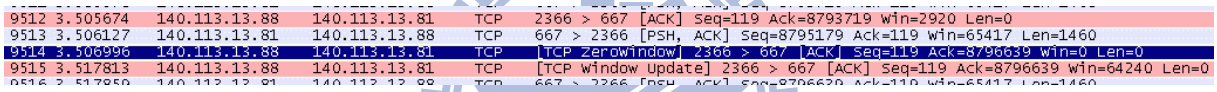

圖 5-7. TCP ZeroWindow

傳送端停止傳送資料，接收端不再收到資料，也不會有 ACK 訊息給傳送端。但是這樣 TCP 連線是否有斷線就不得而知。所以傳送端會傳送 ZeroWindowProbe 的訊息到接收端，接收端收到後回傳 ZeroWindowProbeAck 的訊息，這樣可以確保連線還存在，如圖 5-8。

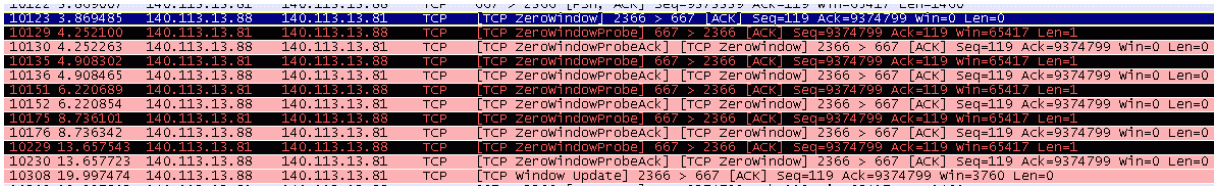

圖 5-8. TCP ZeroWindowProbe

當 Cache2 到 receive buffer 收取資料，receive buffer 空間會釋放出來，此時接收端

會回傳 TCP Window Update 的訊息，如圖 5-9。傳送端收到 window size 異動的訊息，就可以繼續傳送資料。

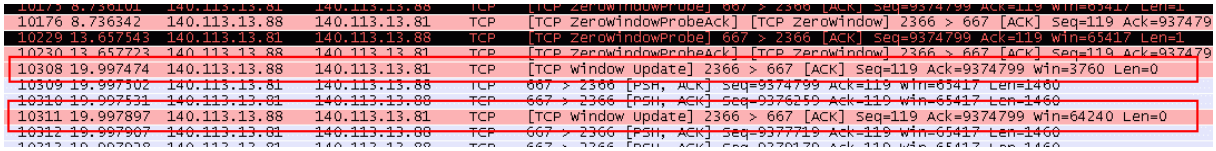

圖 5-9. TCP Window Update

如果 Cache2 到 receive buffer 收取資料的速度比較慢，而傳送端傳送資料比較快，最終 receive buffer 將會溢位。此時也會傳送 Zero Window 的訊息，接收端因此停止傳送，直到收到接收端 TCP Window Update 的訊息，才繼續傳送。

5.1.3 播放影片時流量

影片檔案大小 43,141KB、長度 135.18 秒、MPlayer Cache2 大小設定 8192KB。圖 5-10 為播放影片的影片流量，橫軸單位為秒，縱軸單位為位元組，資料為每 0.1 秒的傳輸量。5-11 為 Client 端 TCP 的 window size，橫軸單位為秒，縱軸單位為位元組。0 秒位置為 Rdesktop Media Server 傳送播放指令，將流量分成 3 個階段：

第 1 階段(~4 秒)：

當 Web Server 開始傳送影片，MPlayer 會先將資料填到 Cache2，填到將近 100%才停止，這階段是以 best effort 傳送，流量會比較大。Client 端 TCP window size 維持在 64KB。

第 2 階段(4~20 秒)：

Cache2 填滿後，資料停止填充，Web Server 停止傳送資料到 MPlayer，這階段的流量為 0。Client 端 TCP window size 為 0。

第 3 階段(20 秒~)：

當 Cache2 被消耗到 50%，開始填充資料，並且維持在 50%的位置。這時候的

流量，即為 MPlayer 消耗的資料量。最後播放快要結束時，影片最後的資料都存在 Cache2，此時 Web Server 已經將資料傳送完畢，TCP 連線將中斷。

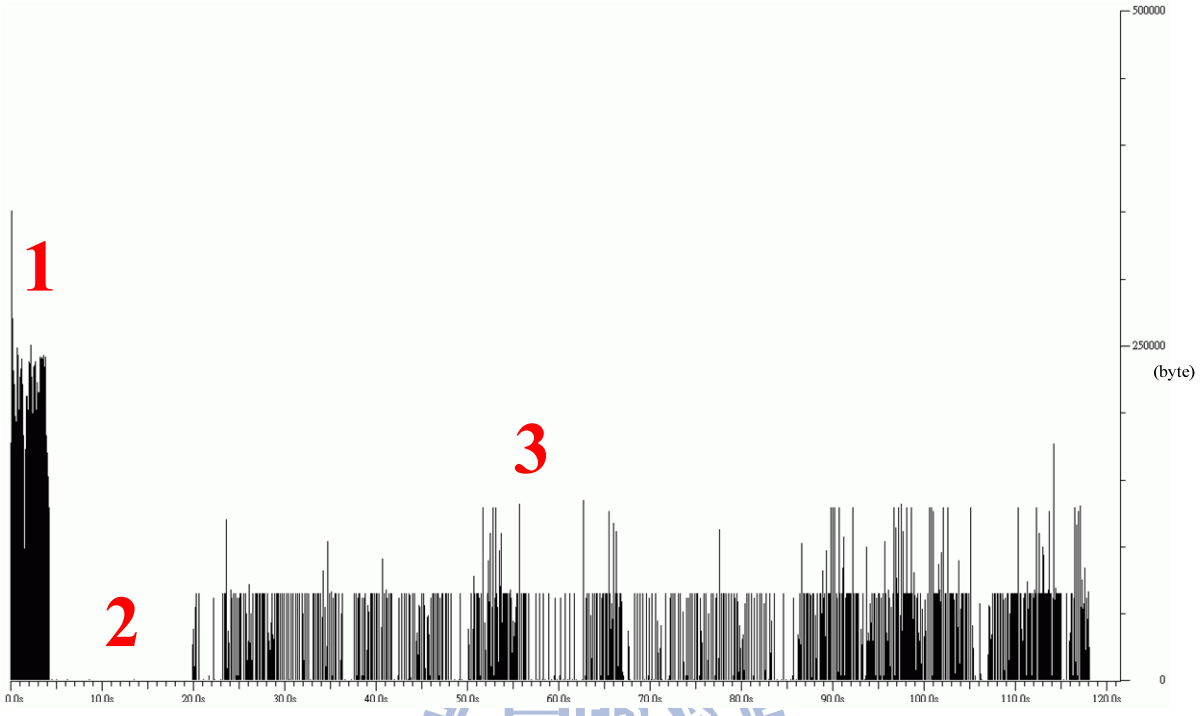

圖 5-10. 播放影片流量圖

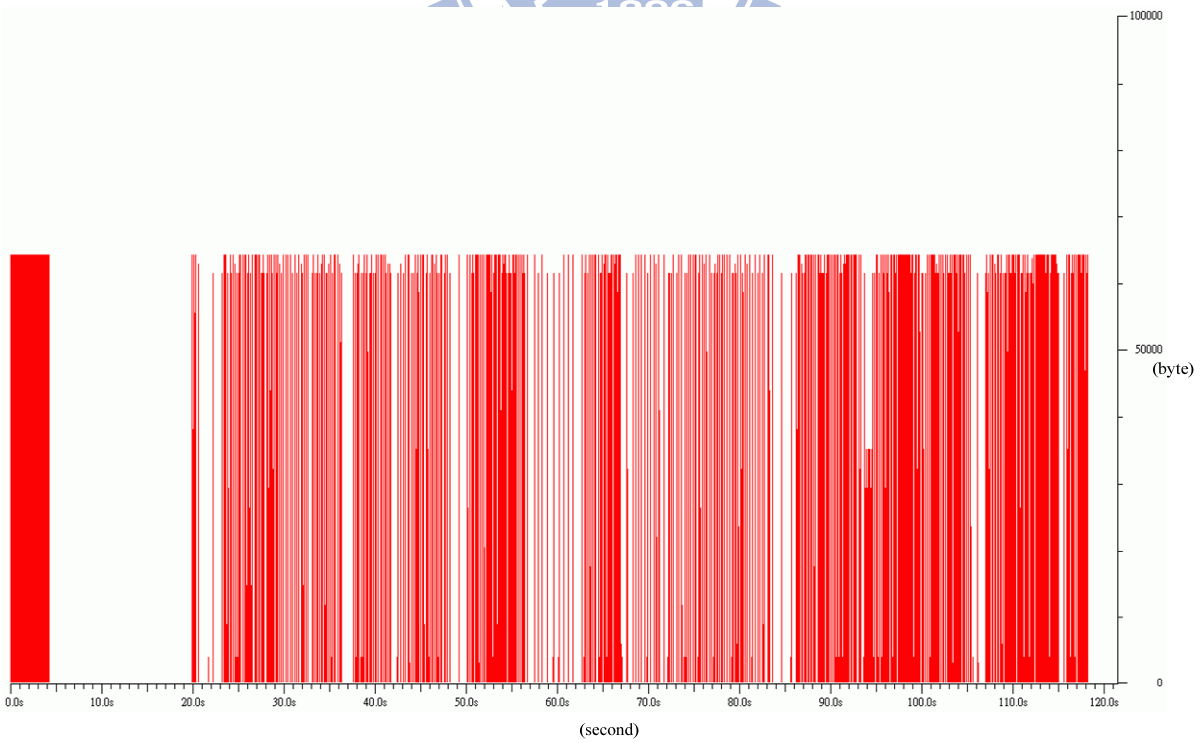

圖 5-11. Client 端 TCP window size

5.1.4 播放影片時 Cache2 狀態

圖 5-12 為播放影片時 Cache2 的狀態，Cache2 大小設定為 8192KB，圖片中橫軸單位為秒，縱軸單位為位元組。第 3.2 節說明了 Cache2 的機制，這邊將測量播放影片時 Cache2 的狀態。

新資料即未播放的影片資料，一開始從 0%填充到 100%即停止。當新資料剩餘 50%，即舊資料為 50%的大小，開始填充資料並且新資料維持在 50%。在播放即將結束時，剩下的影片資料存在 Cache2，此時新資料將從 50%被耗用到 0%。播放影片也隨即結束。

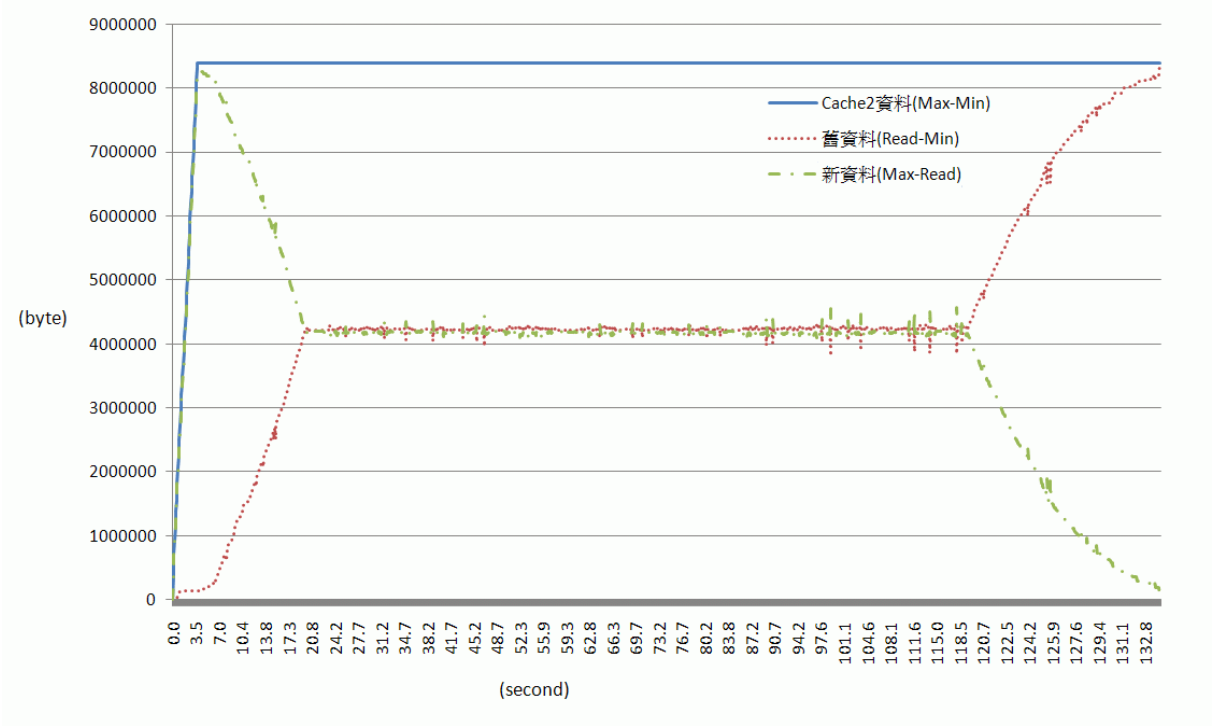

圖 5-12. 播放影片 Cache2 的狀態

5.2 實驗數據比較

表 5-1 為本研究使用的三個不同解析度的影片檔，除了解析度和檔案大小不同，影片長度為一致。分別利用 VNC、RDP 和 RMS 進行影片播放，並測量所消耗的資料量和頻寬。實驗環境說明：

流量測量軟體：DU Meter 3.03

VNC：
連線桌面大小：1024*768 pixels
bits per pixel：16 bpp
VNC Server：WinVNC 3.3.7
VNC Client：WinVNC 3.3.7
encode：Raw
Media Player：Media Player Classic 6.4.9.0

RDP：
連線桌面大小：1024*768 pixels
bits per pixel：16 bpp
RDP Server：Microsoft Windows XP SP3，RDP 5.1
RDP Client：Linux Fedora 8，rdektop-1.6.0
Media Player：Media Player Classic 6.4.9.0

RMS：
連線桌面大小：1024*768 pixels
bits per pixel：16 bpp
RDP Server：Microsoft Windows XP SP3，RDP 5.1
RDP Client：Linux Fedora 8，rdektop-1.6.0
Media Server：Rdesktop Media Server
Media Player：MPlayer 1.0rc2-4.1.2

實驗步驟：

一、分別用 VNC 和 RDP 遠端桌面程式，挑選一個影片播放，並測量播放影片所需要的流量。實驗十次計算平均值，測量完成換另一個解析度實驗，直到三個解析度實驗完成。

二、利用 RDP 遠端桌面程式，並且在 Server 端開啟 Rdesktop Media Server 程式，挑選一個影片播放，並測量播放影片所需要的流量。實驗十次計算平均值，測量完成換另一個解析度實驗，直到三個解析度實驗完成。

表 5-1. 三個解析度的影片資訊

解析度	320*136	480*204	848*352
檔案大小(Kbytes)	5,691	16,353	43,141
位元深度(bpp)	24	24	24
影片長度(s)	135.18	135.18	135.18
更新率(fps)	23.976	23.976	23.976
資料率(kbit/s)	339.77	986.17	2610

5.2.1 資料量比較

表 5-2 和圖 5-13 可以得知，我們的 Rdesktop Media Server 大量縮小了播放影片所需要的資料量。在 848*352 的解析度，Rdesktop Media Server 所需要的資料僅是 RDP 的 1/20；在 480*204 的解析度，Rdesktop Media Server 的資料量為 RDP 的 1/35；在 320*136 的解析度，RMS 的資料量僅是 RDP 的 1/40。

表 5-2. 資料量比較表

解析度	320*136	480*204	848*352
VNC (Mbytes)	275.75	495.09	582.99
RDP (Mbytes)	269.23	609.52	889.27
RMS (Mbytes)	6.62	17.23	43.99

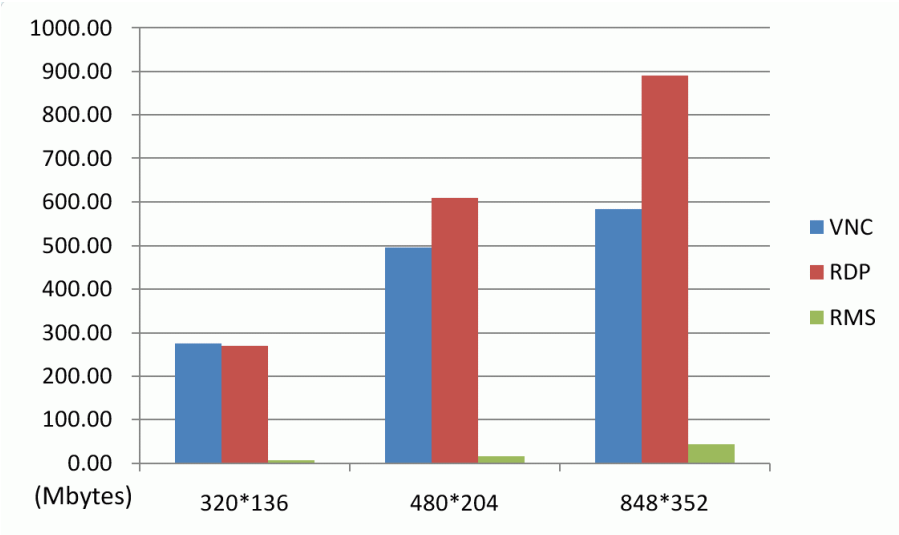

圖 5-13. 資料量比較圖

5.2.2 平均流量比較

表 5-3 和圖 5-14 為平均流量，若是以家用光纖網路下載 10Mbps/上傳 2Mbps 的頻寬。以 Rdesktop Media Server 方式播放影片，家裡的電腦當 Client，三種解析度的影片都可以順利播放。家裡的電腦當 Server，上傳頻寬只能提供 320*136(約 0.4Mbps)和 480*204(約 1Mbps)解析度播放，848*352(約 2.6Mbps)就顯得不足夠。

表 5-3. 平均流量比較表

解析度	320*136	480*204	848*352
VNC (Mbytes/s)	2.040	3.662	4.313
RDP (Mbytes/s)	1.992	4.509	6.578
RMS (Mbytes/s)	0.049	0.127	0.325

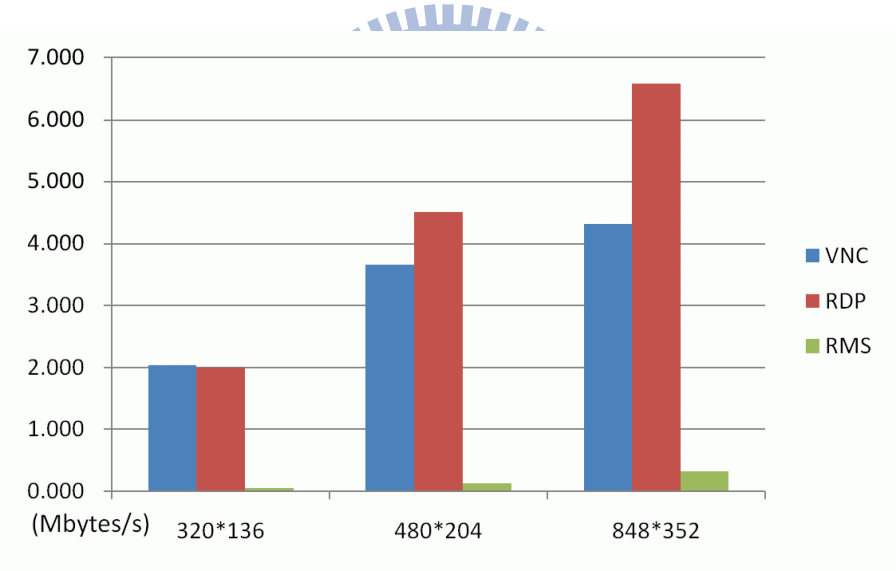

圖 5-14. 平均流量比較圖

5.3 嵌入式平台

當我們更改的 MPlayer 和 rdesktop 程式在 x86 的環境測試完成後。將程式移植到以 ARM 為核心的嵌入式平台。本研究採用的平台儲存空間有限，所以將移植好的程式存在 CF 卡，再透過超級終端機輸入指令將 CF 卡掛載到檔案系統。實驗環境如下：

連線桌面大小：640*480 pixels
bits per pixel：16 bpp
Server Platform：AMD64 X2 4800+(x86)
RDP Server：Microsoft Windows XP SP3，RDP 5.1
Media Server：Rdesktop Media Server
Client Platform：Intel Xscale PXA270 520MHz(ARM)
RDP Client：Linux (kernel 2.4.21) tinyX，rdesktop-1.6.0
Media Player：MPlayer 1.0rc2-4.1.2

在此嵌入式平台只是個終端機的角色，所有 Client 端的滑鼠鍵盤訊息都會透過 rdesktop 擷取，並且利用網際網路將訊息傳送到 Server 端。Server 會將訊息處理完，將畫面傳回 Client。主要的運算都是在 Server 端，所以就一般的操作 Client 端的處理器並不需要很強。圖 5-15 為系統架構圖，左邊為 Client 是以 ARM 為核心的嵌入式平台，並且在上面執行 rdesktop 和 MPlayer 程式；右邊為提供遠端桌面服務的 Server。中間的資料傳輸是透過網際網路。

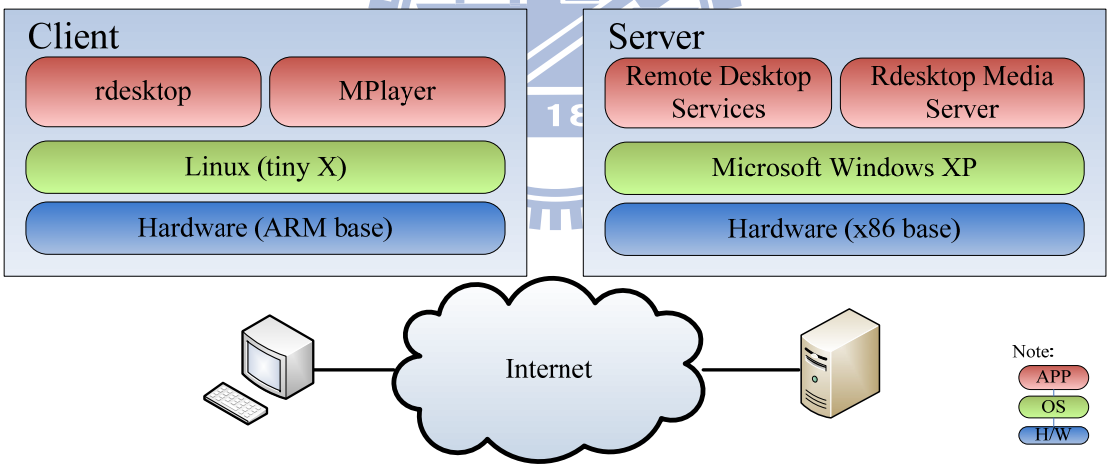

圖 5-15. 系統架構圖

嵌入式平台執行 rdesktop 的畫面，如圖 5-16。當使用者點選影片，透過我們的系統，可以順利開啟影片視窗，如圖 5-17。但是由於影片的解碼是在 Client 端執行，所以在這邊 Client 端的處理器能力是需要考慮。在嵌入式平台開啟影片時，畫面非常不順而且有影音不同步的問題。所以推斷本研究採用的 Intel XScale PXA270 核心，對於影片解碼能力並不足夠。

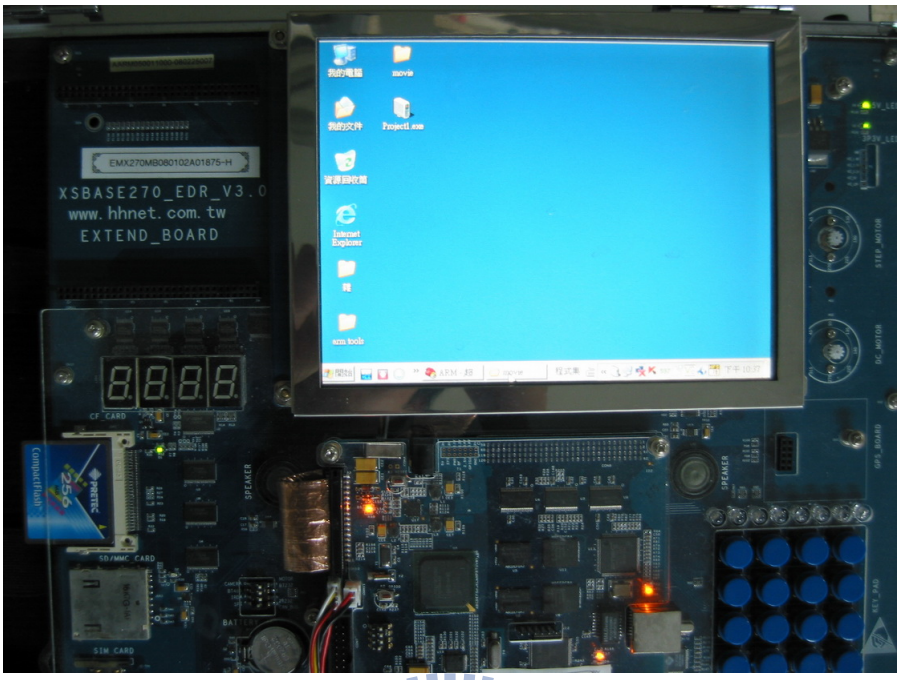

圖 5-16. 嵌入式平台執行 rdesktop 的畫面

圖 5-17. 嵌入式平台播放影片的畫面

第六章　結論

6.1 結論

遠端桌面程式共有的問題－播放影片造成網路流量暴增，本研究利用串流技術解決資料暴增的問題。但由於 RDP Server 為微軟所掌握，無法取得相關資訊。所以透過外加的串流伺服器，將 Server 端的影像資料傳送到 Client 端播放。本研究採用開放程式碼 RDP Client－rdesktop；多媒體播放器－MPlayer，完成 Client 端實作。Server 端實作則是自行撰寫的程式。

本論文第三章解析 MPlayer 多媒體播放程式，包含開檔模組、Cache2 機制、影像濾波器和影像輸出模組。解說開檔模組、影像濾波器和影像輸出模組使用的資料結構，而這結構成員有許多函式指標指到特定模組的操作函式。在播放影片前，將建立 video chain，這會決定影像解碼器、影像濾波器和影像輸出模組。最後說明影像在解碼後，圖像資料是如何在影像濾波器和影像輸出模組間傳遞。

第四章，在 Client 端撰寫 MPlayer 的 rdp 影像輸出模組，將影像畫面輸出到 fifo 檔。rdesktop 程式，加入 MPlayer 播放控制和影像接收並顯示。前者負責接收 Server 的播放命令並執行 MPlayer 程式；後者則是從 fifo 檔讀取影像資料並且將影像顯示到視窗上。在 Server 端，撰寫 Rdesktop Media Server，裡面包含 TCP Socket Server－負責傳送播放指令和 Web Server－傳送使用者點選的影片。

RDP 和 VNC 在播放影片時，影片的畫面並非單調而是複雜的，此時都是採用非失真壓縮的方式傳送，所以耗費的網路流量會非常大。由於我們是利用 Progressive Download 的串流方式，所以網路流量取決於影片的編碼率。編碼率對於非失真壓縮的圖像，即是影片的壓縮率。所以 Progressive Download 串流的方式網路流量會小很多！

6.2 未來發展

本研究利用串流技術解決遠端桌面程式播放影片造成的資料暴增和不順暢的問題。但還是有改進的空間。

- 使用遠端桌面時，點選影片可以很順暢地播放，但是控制只有播放而沒有停止或快倒轉功能。影像畫面嵌入到遠端桌面程式，子視窗內影像大小無法調整。未來可以增加控制介面和影像畫面縮放功能。
- 由於是使用一般的 Web Server 來進行 Progressive Download 的串流方式，所以影片的解析度、編碼率或是取樣率並沒辦法調整。
- 嵌入式平台使用的 Intel XScale PXA270 影片解碼能力不足，可以改用運算更快或是內建硬體解碼的 CPU。

參考文獻

[1] Tristan Richardson, “The RFB Protocol”, ORL/AT&T Labs Cambridge, June, 2008

[2] rdesktop: A Remote Desktop Protocol Client
http://www.rdesktop.org/

[3] Oliver Jones 著，Introduction to the X Window System，黃豐隆譯，松崗電腦圖書資料股份有限公司，民國 82 年，台北

[4] David Austerberry, The Technology of Video and Audio Streaming, 2nd edition ,Focal Press, 2004

[5] VLC media player - Open Source Multimedia Framework and Player
http://www.videolan.org/vlc/

[6] MPlayer - The Movie Player
http://www.mplayerhq.hu/

[7] Windows Media Player 11
http://www.microsoft.com/windows/windowsmedia/player/11/default.aspx

[8] Introduction to Interprocess Communication Using Named Pipes
http://developers.sun.com/solaris/articles/named_pipes.html

[9] 李宗學，“終端服務的桌面影像壓縮”，國立交通大學，碩士論文，2008

[10] Doug Abbott, Linux for Embedded and Real-time Applications, Newnes, 2003

[11] 江元新，Linux 核心開發與實務-INTEL XScale PXA270，長高科技圖書，2006

[12] 華亨科技股份有限公司，XSBase270(EELiod) ADS/Linux/WinCE 實驗開發與實務

[13] Hypertext Transfer Protocol -- HTTP/1.1
http://www.w3.org/Protocols/rfc2616/rfc2616.html

附錄一：加入 rdp 影像輸出模組

要在 MPlayer 加入自己的影像輸出模組有下列幾個步驟

第一步：建立檔案 libvo/vo_rdp.c。

第二步：按照輸出模組的函式撰寫程式。紅框部分是模組的簡稱，這是要讓 MPlayer 可以呼叫此模組。

```
static vo_info_t info =
{
    "rdesktop plugin output",
    "rdp",
    "Ju-Hung Teng <kuwaktw@gmail.com>",
    ""
};

LIBVO_EXTERN(rdp)
```

第三步：更改 libvo/video_out.c。這邊有兩個地方要加入。

```
//
// Externally visible list of all vo drivers
//
extern vo_functions_t video_out_mga;
extern vo_functions_t video_out_xmga;
extern vo_functions_t video_out_x11;
extern vo_functions_t video_out_xover;
extern vo_functions_t video_out_xvmc;
extern vo_functions_t video_out_xv;
extern vo_functions_t video_out_bl;
extern vo_functions_t video_out_fbdev;
extern vo_functions_t video_out_fbdev2;
extern vo_functions_t video_out_svga;
extern vo_functions_t video_out_png;
extern vo_functions_t video_out_ggi;
extern vo_functions_t video_out_aa;
extern vo_functions_t video_out_caca;
extern vo_functions_t video_out_mpegpes;
extern vo_functions_t video_out_yuv4mpeg;
/*---------add on code start----------*/
extern vo_functions_t video_out_rdp;
/*---------add on code end----------*/
```

```
vo_functions_t* video_out_drivers[] =
{
#ifdef HAVE_XVR100
        &video_out_xvr100,
#endif
#ifdef HAVE_TDFX_VID
        &video_out_tdfx_vid,
#endif
#ifdef HAVE_DIRECTX
        &video_out_directx,
#endif
#ifdef WIN32
    &video_out_winvidix,
#endif
    &video_out_cvidix,
#endif
        &video_out_null,

/*---------add on code start----------*/
    &video_out_rdp,
/*---------add on code end----------*/
```

第四步：更改 libvo/Makefile 這樣在 make 的時候才會將程式編譯。

```
include ../config.mak

LIBNAME_COMMON  = libosd.a
LIBNAME_MPLAYER = libvo.a

SRCS_MPLAYER = aspect.c \
               geometry.c \
               spuenc.c \
               video_out.c \
               vo_mpegpes.c \
               vo_null.c \
               vo_rdp.c \
               vo_yuv4mpeg.c \
               $(VO_SRCS) \
```

附錄二：Build rdesktop & MPlayer for ARM

rdesktop

第一步：網路下載 openssl-0.9.6m.tar.gz，解壓縮後，輸入下面指令

```
[root@localhost openssl-0.9.6m]# ./Configure linux-elf-arm no-shared
```

第二步：開啟 Makefile，將"CC=gcc"改成"CC=arm-elf-gcc"。CFLAG 的後面"-O3"改成"-O2"，並加上"-D__PIC__ -fpic -msingle-pic-base -Wl,-elf2flt=-z"，然後 make

```
[root@localhost openssl-0.9.6m]# make
```

第三步：編譯完成後，將 libcrypto.a 和 libssl.a 放到/opt/xscalev1/lib/下。

第四步：網路下載 rdesktop-1.6.0.tar.gz，解壓縮 rdesktop 後，輸入下面指令。

```
[root@localhost rdesktop-1.6.0]# ./configure --host=arm-linux CC=arm-linux-gcc --with-sound=oss
```

第五步：更改 rdesktop 下的 Makefile，將 LDFLAGS 後面改成-L/opt/xscalev1/lib -lcrypto –lpthread。如此重新 make，可以成功編譯 rdesktop for ARM。

第六步：編譯程式

```
[root@localhost rdesktop-1.6.0]# make
```

MPlayer

下載 MPlayer 並且解壓縮後，輸入下面指令

```
[root@localhost MPlayer-1.0rc2]# ./configure --prefix=/usr --target=arm-linux-gnu --cc=arm-linux-gcc
--host-cc=gcc --as=arm-linux-as --disable-md5sum --disable-pnm --disable-tga --disable-dvdnav
--disable-dvdread --disable-dvdread-internal --disable-ftp --disable-tv --disable-dvbhead
--disable-mmx --disable-mmxext --disable-3dnow --disable-3dnowext --disable-sse --disable-sse2
--disable-ssse3 --disable-shm --disable-altivec --disable-armv5te --disable-armv6 --disable-iwmmxt
--disable-fastmemcpy --disable-dynamic-plugins --enable-alsa --enable-ossaudio --enable-select
[root@localhost MPlayer-1.0rc2]# make
```