可熱換置嵌入式系統之模組化設計

Modules Design of the Hot-Swappable Embedded System

蔡亮宙[1,*]、戴源良[2]、陳冠宇[1]

[1]南台科技大學

電機工程學系

台南縣永康市

ljtsai@mail.stut.edu.tw

[2]國立成功大學

電機工程學系電腦與網路組

台南市

dai@crow.ee.stut.edu.tw

Lian-Jou Tsai[1]、Yuan-Liang Dai[2] and Guan-Yu Chen[1]

[1] Department of Electrical Engineering

Southern Taiwan University of Technology

Yungkung, Tainan

ljtsai@mail.stut.edu.tw

[2] Department of Electrical Engineering

National Cheng Kung University

Tainan City

dai@crow.ee.stut.edu.tw

Received 18 September 2006; Revised 13 November 2006 ; Accepted 13 November 2006

摘 要

嵌入式系統已經具有非常普及的產品，例如時常使用的行動電話、網路電話、汽車內衛星定位系統、錄放影機、電視遊樂器等，在如此眾多的嵌入式設備中，在軟體上往往更新不易，且由於嵌入式設備大多屬於消費型電子商品，軟體的開發時間相對較短，如果可以線上熱換置嵌入式設備內核心軟體模組，除了廠商樂見，一般使用者也可以藉由更新韌體版本達到最穩定的狀態，使其設備發揮最大功用；本文中，我們提出一個可以線上熱換置(Online Hot-Swap)的系統架構，並實現一個雛型系統HS[2] (Hot-Swap System)，它將所有的核心軟體與應用程式模組化，再藉由遠端一部組態工作站所提供的熱換置功能與使用者界面，對嵌入式系統上的任一模組實施動態的上傳、刪除或遷移，讓嵌入式系統擁有動態維護、軟體再用及效能最佳化的優點，而且在多個系統組成的分

* 通訊作者

散式應用中，也可以由組態工作站做最適當的工作動態分配或負載平衡，讓有限的處理器與記憶體資源，能夠被最佳化的使用。本系統軟體模組則包含系統模組與應用模組，在熱換置方面的考量基本上是不予區分的；本文將說明HS^{2}之設計理念，進而針對個別模組解釋其工作機制與其特有的優點。

關鍵詞：嵌入式系統、熱換置系統、核心程式

ABSTRACT

Nowadays there are prevailing products from embedded systems. They cover so many facilities in our living environment such as cellular phones which we carry all day long, the internet phone, the mobile GPS, video players and TV game players. Among those facilities, their function of software updating is not so easy and automatic. Plus, when people think a product as a commodity, they think it is consumable. This character of short life-cycle pushes vendors to shorten their development period. If an embedded system can be hot-swapped remotely, it will in some degree satisfy vendors as well as users, because vendors can update their products after delivery whenever potential bugs revealed or additional functions developed, and therefore users gain the benefits of stability and selectable of software modules. In this paper, we propose an on-line hot-swap system architecture. We modulize all the software including system software and applications in our prototype, called HS^2. The functions of on-line configuration are supported on a remote configuration PC running HS-Console. Users use the visual interface provided by the HS-Console to manipulate the operations such as updating, adding, deleting and migrating modules. With the features, embedded systems not only have the advantages of dynamic maintenance, software reuse and optimized efficiency, but also load balance and fault tolerance when we deploy a group of embedded systems to work together and dynamically dispatch or retreat software modules to the right system in the right time. The overall processing power and memory resource are therefore optimized. Software modules are kernel modules and application modules, while the two are treated no differences in HS^2. We will describe, in this paper, the design philosophy of HS^2, and then explain each module by focusing the working mechanism and the specialties it have.

Keywords: Embedded systems, Hot-swap systems, Kernel

一、動機與相關研究

1.1 動機

在早期嵌入式系統(Embedded Systems)核心程式屬於客制化(customized)的程式且大多是封閉型系統居多，例如 VxWork、pSOS、QNX 等，但近年來由於半導體的製程進步，促成整合式晶片 SoC（System on Chip）蓬勃發展，相對的嵌入式系統核心程式的需求也日漸地增加，促成許多運行在 PC 上的系統程式移植到嵌入式系統上，如 Microsoft 發行的 WinCE、Mobile 5.0、與 GNU/Linux[1]等系統，但隨著嵌入式系統越來越普及，許多軟體的問題接踵而來，例如行動電話爲了搶占市場，開發的時間往往都很短暫，但是軟體總是需要長時間的測試才可以確保穩定，然而市場的需求卻不容許廠商如此，導致有讓使用者或者通路商負責軟體更新的現象，這也讓使用者承擔了更新失敗的風險與操作的複雜度，如果系統能提供一個可以線上熱換置的機制，讓提供的廠商或使用者可以很方便的更新軟體而不需要關機或停止操作，或進一步的提供自動下載自動更新等功能，以下列出熱換置系統可以具備的優點：

A. 熱換置的嵌入式系統可以打破業者在硬體與軟體間開發上的相依性。

一如工業用 PC 市場一樣，硬體廠商獨立選用不同軟體廠商所開發的軟體模組，軟體廠商也可以爲一些不同硬體平台開發獨立的軟體模組；嵌入式軟體有時爲了配合硬體的特殊性，軟硬體間可能有較強的相依性，而獨立的模組因爲透過 IPC_port 機制與其他模組連絡，對硬體的相依性相對減輕許多。

B. 線上對軟體或硬體做必要的更換或升級。

搶先時機就是搶先商機，導致生產廠商擠壓 α 測試及 β 測試以利上市時程，這在嵌入式系統上時有所聞，即使是一套非常穩定的系統，也可能面臨時間因素產生的環境變化或是有心人士的破壞所產生的漏洞，導致軟體面臨更新或版本修正的事件幾乎是難以避免。此時將系統下架維修停止服務又可能發生程度不等的代價，視所應用場合而定；本計畫可用以解決此項困擾，甚至需要更換整個硬體時，維修人員只需攜帶新硬體到用戶端，由遠端下載原有特定程式模組，安裝上新品後再載入到新系統上即可。

C. 出廠最初狀態只需配置最少的必要模組。

省略大顆 ROM/Flash 硬體成本，其餘的模組只需使用遠端工作站載入即可，降低硬體成本與提升競爭力。

D. 軟體再用(Software Reuse)。

硬體可以升級，軟體也可升級或增加功能模組；被換掉的或不適用的軟體模組並不一定是一無是處，尤其是那些因爲系統擠進太多的工作，不得不移出部分工作到別的地方做另外的適度發揮，或是重新組合軟體模組到不同硬體等級平台上執行，以達資源使用最佳化的目標。

E. 硬體再用(Hardware Reuse)。

使用者除了可以使用前述的方法修正舊版軟體外，還可以使用相容且升級的硬體更換掉原有硬體系統，再載入原有軟體，終究硬體的升級速度是非常快的；此時硬體的容量與運算能力提昇，新增或添購新的軟體模組又變得非常適合了。不再適用或被更換掉

的嵌入式系統硬體可用到別的地方做其他方面的硬體再用，至少可以重新規劃爲線上容錯(fault tolerance)裝置亦可。

F. 嵌入式系統群組的合作(Collaborated)應用。

由數部嵌入式硬體分組而組成的系統可以由遠端組態工作站負責管理，依不同時段或不同環境狀況，對嵌入式成員上的軟體做個別的調校以達到群組的最佳化操作方式，這是因爲應用程式也是以模組型式存在，舉例而言，組態工作站亦可承擔群組負載平衡(load balance)或失誤容忍(fault tolerance)的工作，視需要在重負載的平台上遷移某些模組到輕負載的平台上執行，或是爲了容忍可能的失誤發生而放置備份(redundant)軟體到其他平台上。

G. 動態硬體可變架構的可行。

在最初的規劃中，我們將模組劃分爲軟體模組與硬體模組；軟體模組就如前面所敘，硬體模組則是一個用以燒錄到 Flash 的映像檔，這個映像檔可以是作爲 bootloader 的執行檔，用以變更系統開機行爲；也可以是將 VHDL 或 Verilog 等語言編譯後的.sof 檔(以 Altera 公司爲例)，用以載入到 FPGA/CPLD 旁的 Flash 零件中，再觸發重置(reset) 讓 FPGA/CPLD 自行載入並且變更硬體功能。我們已經完成前者的硬體模組上下載，可以動態更改 bootloader 功能，至於後者因爲需要硬體平台的配合，還在規劃中。

1.2 相關研究

在嵌入式系統的應用中，Linux 是很容易取得也容易開發的系統，主要是因爲開放原始碼並且有大量的應用程式可以搭配使用，使得開發的時效大幅縮短。在 Linux kernel 提供了一個可線上動態更新模組的機制，免除重新載入核心程式時須重新開機的困擾，但卻只限於驅動程式模組而且不包含核心程式中關鍵的部份，如需變更關鍵模組就須重新編譯核心且重新載入，變更或重建系統時等同於將系統重開機，這是我們不樂於見到的；因此如果系統能提供熱換置(Hot-Swap)機制，那以上的問題將能解決，當使用者更換與檔案系統相依的模組時，也可以做到使用者透通(transparent)，不需卸載不需重開機。目前 IBM 的 K42 計畫中[2][3][4]以 Linux 核心爲基礎，是一個專爲熱換置功能爲目標所發展的系統。它是一個開放性原始碼的專案，由 IBM 主導，K42 的特點有以下兩點:

A、物件模組化使用 C++撰寫程式碼，導入物件導向技術。

B、靜態偵測系統是否執行動態更新與熱換置。

當 K42 要更換模組時會在系統內部插入一個中介的監控程式，負責有關於新舊模組的系統呼叫攔截與轉接的工作，當舊模組執行完成後中介的監控程式會將系統呼叫轉向至新的模組，之後中介的監控程式退出監控。K42 對於安全點(safe point)方面花費不少心力，安全點是用來確保目前舊模組轉換到新模組的一致性(consistency)，狀態監控與轉換是透過中介的監控程式來監督此模組的狀態與資料轉換，舉例而言，在 Linux 檔案系統內要動態的更換模組，必須在新舊模組內插入中介的監控程式查看每個工作與舊模組通訊的狀態，並且將舊模組的資料作轉移，例如 I-node 等資訊，當一切工作完成確定在安全點時就可以將新模組替換上將舊模組移除，當然在動態更新這個機制上面也須記

錄模組版本資訊。

除了 K42 之外，尚有許多相關的研究，例如在 Linux 系統上提供檢查點(checkpoint)的機制，方便系統做程序遷移(process migration) [5][6]，與針對嵌入式 Linux 核心程式模組換置 [7]等相關研究，這些系統皆提供動態的載入系統驅動程式，唯尚未達到所有的系統模組皆可替換的地步。

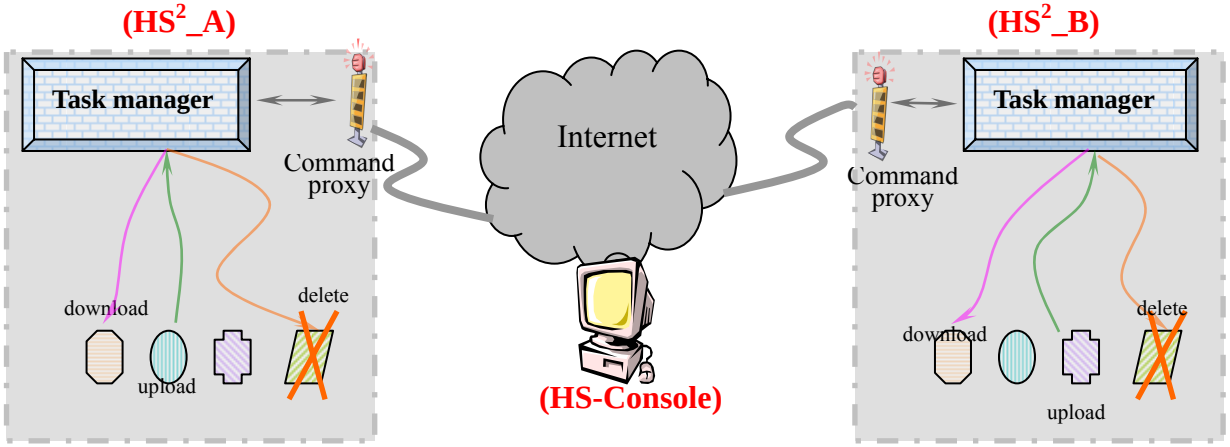
二、系統設計

2.1 簡介

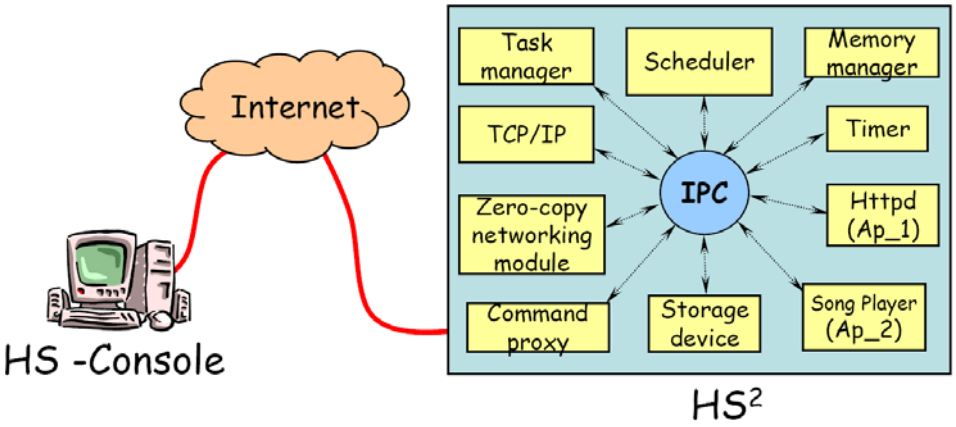
如何提供一個可以線上熱換置系統是本研究設計優先考量的主要目標之一，我們採用與L4[8]、uCosII[9]相同的微核心設計架構，並將雛型系統取名爲HS^2 (Hot-Swap System)，圖一中簡單的說明熱換置的功能流程，每一部HS^2目標平台內都有許多模組存在，其中的命令代理者(command proxy)負責執行遠方組態工作站所下達的命令，組態工作站中執行著HS-Console視窗界面，用以提供使用者方便的上傳、下載、遷移等動作。

2.2 系統架構

目前的HS^2是架構在ARM-Base平台上，實現在ARM7（Samsung s3c4510）、ARM9(Samsung s3c2410)上執行，圖二爲本系統架構圖，屬於只能使用埠(port)作訊息傳輸(message passing)的系統，每個獨立模組的通訊都是透過IPC (Inter-process communication)完成，對硬體操作模式而言，IPC是唯一的一個在系統核心空間(Kernel Space)執行的模組，其餘的模組皆放在使用者空間(User Space)，如此一來不論是排序器(Scheduler)或者乙太網驅動模組(Network Driver)與應用程式(applications)在角色上並無兩樣，特性上較適合嵌入式系統應用，也因爲核心較與傳統微核心(micro-kernel)小，較像nano-kernel架構。



圖一：線上熱換置簡述流程

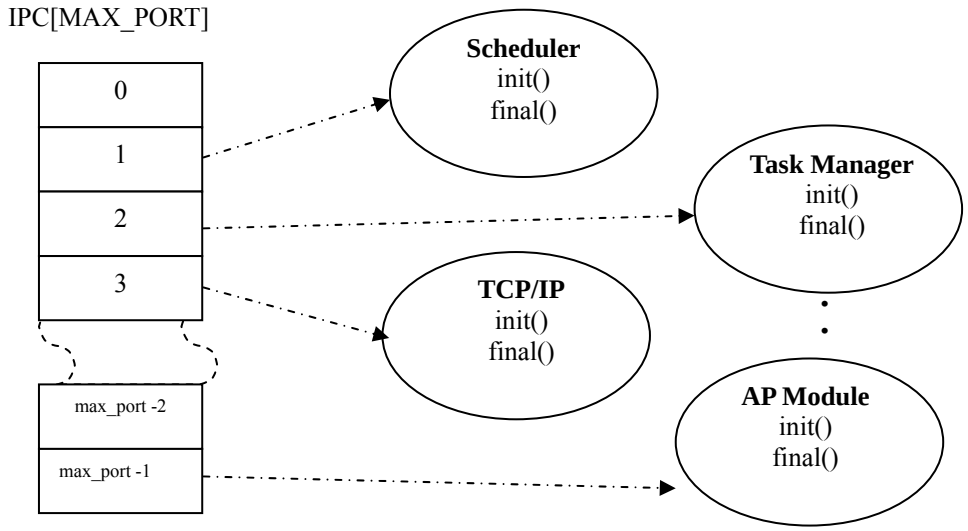


圖二：系統架構

每一模組皆包含init()及final()兩個函數，作爲模組的初始化與結束時的垃圾清理工作；執行中的工作通常是包含狀態的(stateful)，當換置執行中的模組時，有時因爲應用上的允許而可以直接更換，但在大部分的應用中卻是不可行的，而且，每一個模組保有自己的資料結構，很有可能這些資料結構又會與系統其他的模組有著強烈的關聯性，這種現象在系統基礎模組上普遍存在，例如系統中的記憶體管理者，它記錄了系統中所有模組所使用的記憶體狀態，若想動態更換它時必須將其記憶頁使用狀態詢問出，並讓新模組轉換成自己的資料格式，此時，若是舊模組不提供內部資料詢問功能時，換置模組即無法保持現有狀態而能接續執行。因此，換置或是遷移模組必須考量其安全點(Safe Point)以及資料的交接才能保障其過程的一致性(consistency)，在HS^{2}中，模組置換與遷移時所產生的一致性問題必須由模組本身管理，我們認爲，對於遷移或換置執行中的模組所需知道的安全點問題，只有模組本身最清楚，例如在移動模組之前，命令代理者(Command Proxy)必須先對被遷移模組發出mig_req要求遷移的訊息，被遷移模組則等到安全點時再發出mig_gnt訊息，告知自身已經可以被凍結及遷移了。這種模組自行管理一致性的問題也增加了模組開發時的額外負擔，以及多出程式記憶體的資源，只是，對於將模組加入一個運行中的系統，用以增加或是取代系統中某一工作，此時，了解工作運行狀態後再接續執行似乎是必要的，若考量只執行一次但卻浪費程式記憶體資源時，則可考量模組適當的釋放此記憶體空間；以下分別說明HS^{2}內的基本模組。

2.3 行程通訊模組(IPC: Inter-Process Communication)

我們取名 IPC_port 作爲 HS2 系統中的通訊埠(port)，以免與別種功能的埠混淆，IPC_port 爲模組間訊息傳遞(message passing)的機制，它是唯一執行於 ARM 架構下的系統模式(system mode)的模組，可接受中斷產生的訊息(message)，並提供所有其他模組基本通訊命令(primitives)，包含有非同步用的 ipc_send()、ipc_receive()以及同步用的 ipc_request()、ipc_reply()，這些基本通訊命令最後皆用軟體中斷指令(swi)交換到 IPC 模組傳遞訊息；如圖三所示，每一個在系統中的模組在執行期都必須對應到至少一個以上的 IPC_port，模組之間的通訊只能以 IPC_port 爲來源或目標識別，所有新模組皆必須向



圖三：IPC 對應的 IPC_port

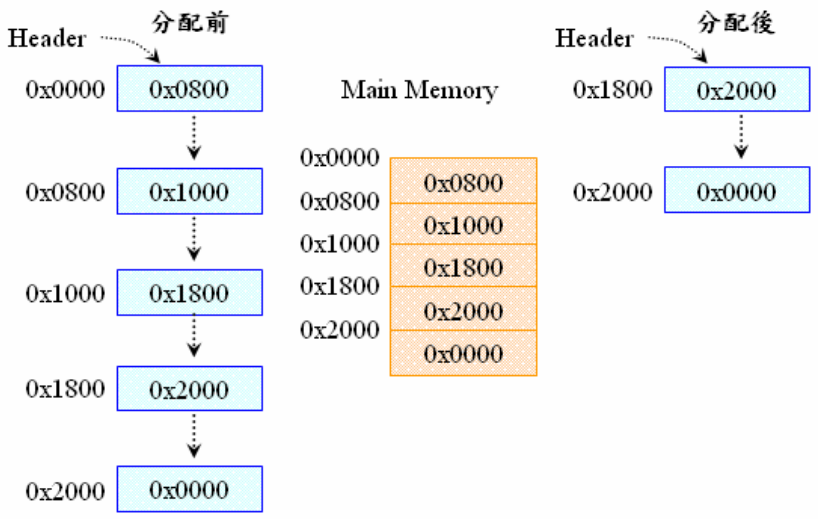
所有新模組皆必須向系統註冊取得 IPC_port，正常的作法是在 init()函數上執行，init()函數在每一個模組上皆有，用以執行必要的初始化動作，例如若要換置某一舊模組時，新模組可以先行註冊取得一個未用的 IPC_port，並用以與舊模組連絡取得舊模組的資料結構，轉換到自己的資料結構，再行搶用舊模組的 IPC_port，此時才可通知記憶體管理者回收舊模組的記憶體了，final()函數與 init()函數類似，用來做自我清除的動作。

2.4 記憶體管理者(Memory Manger)模組

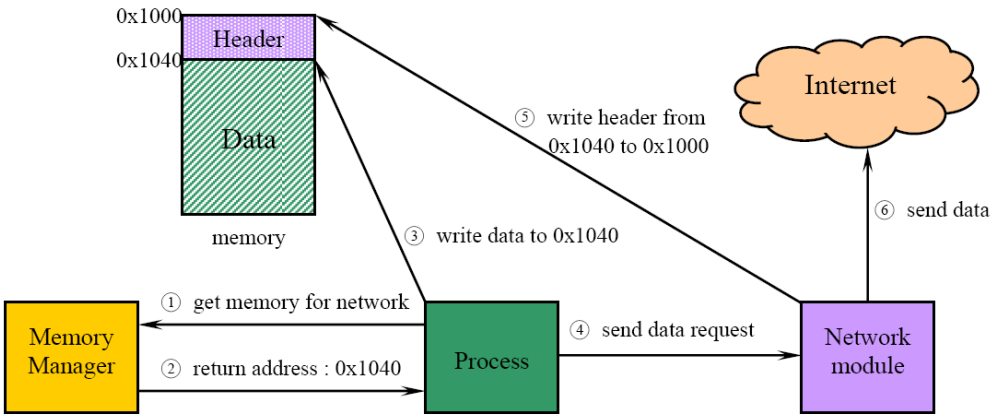
在系統內的每一個行程要執行之前，一定會跟記憶體管理行程要求一段足夠的記憶體空間，而當行程執行結束之後也必須將其歸還至記憶體管理行程。在嵌入式系統中記憶體是珍貴的資源，記憶體分配、回收與破碎問題對於整體系統效能有很大影響，目前我們的記憶體管理模組具備兩項特色，其一、除了一般的記憶體配置外，還額外提供給乙太網路控制器專用的記憶體框架(frame)配置，以配合乙太網路零拷貝(zero copy)機制；其二、配置記憶體時以速度爲考量，乘除法只使用移位指令，並未使用乘除指令。以下分別說明一般記憶體管理與網路零拷貝機制乙太網路記憶體管理。

A. 傳統記憶體管理

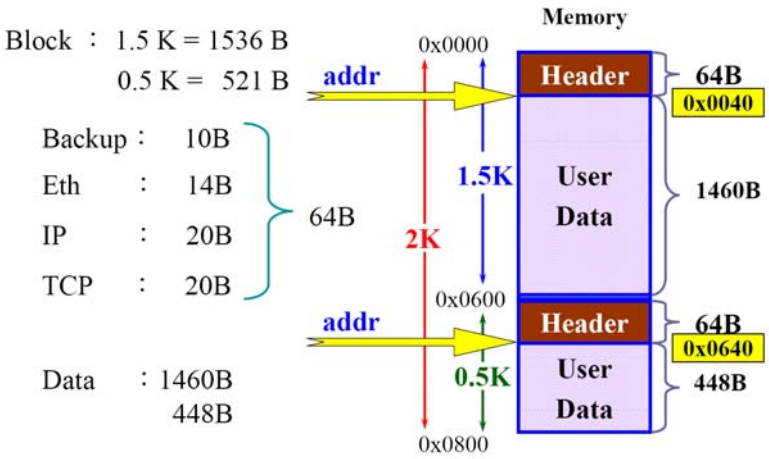
與一般傳統作業系統[10]的記憶體管理相同，任何有需要記憶體之行程可以向記憶體模組發出請求。記憶體分頁大小爲 2Kbyte，主要是針對嵌入式小系統應用並可配合乙太網路驅動程式零拷貝機制，所有記憶體以 2KB 爲單位使用鏈結串列紀錄，若行程對記憶體管理要求分配記憶體大小時，記憶體管理者會從鏈結串列中尋找最適合連續空間配置。圖四舉例說明，假設目前有一個行程請求 4.5K 記憶體時，記憶體模組會在鏈結串列中尋找連續 3 頁，找到之後回傳其起始位址給請求之行程使用。否則將傳回記憶分配請求失敗。



圖四：記憶體分配例



圖五：零拷貝工作流程



圖六：零拷貝記憶體使用的兩種訊框

B. 零拷貝機制乙太網路記憶體管理

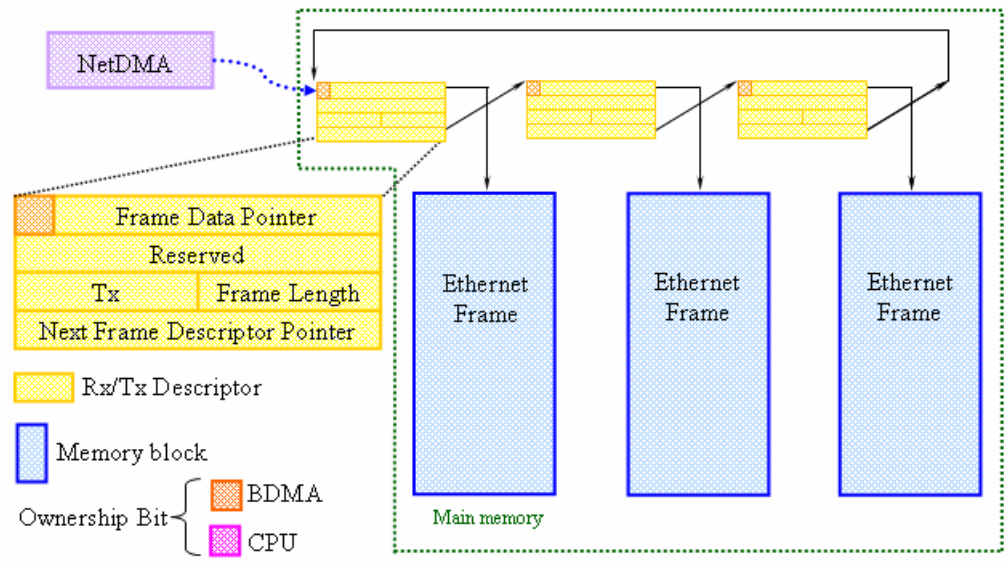
HS^{2}一開始架構於ARM7TDMI ，本身並沒有硬體記憶體管理MMU(Memory Management Unit)支援，在分配記憶體給行程時必須採用連續記憶體配置方式，這也導致封包由上層往下層傳送時，爲了標頭資料所要求的新記憶體空間有時無法佔用連續位址而必須有封包複製的現象發生，因爲連續的記憶體已經被要走了。HS^{2}中的記憶體管理者可以解決這個問題，無須複製進而提升效能；只是在另一個ARM920T的CPU架構上已包含了MMU，程式設計者可以直接設定記憶體映對暫存器，無須複製資料即可變更記憶體應對位址；唯目前在ARM9-base系統上並沒使用MMU虛擬記憶體特性，而是使用與ARM7 上相同的作法；我們認爲，此種設計不只設計較簡單應用較廣泛，也可鎖定一些較小而無MMU之嵌入式系統可以方便使用。

Zero Copy 技術關鍵以圖五表示，當記憶體分配時在記憶體上預留一段空間讓網路模組去填寫乙太網路標頭、IP 標頭、TCP/UDP 標頭與 CRC 檢查碼資訊。這樣網路模組傳送封包時，不再需要爲了將標頭資料與使用者資料放在連續位址而對資料作拷貝動作，進而提升網路模組的效能。圖五說明當應用程式要求配置網路記憶體時，記憶體模組在回傳記憶體給應用程式時將起始位址扣除標頭 64Bytes 用來預留乙太網路標頭資訊。

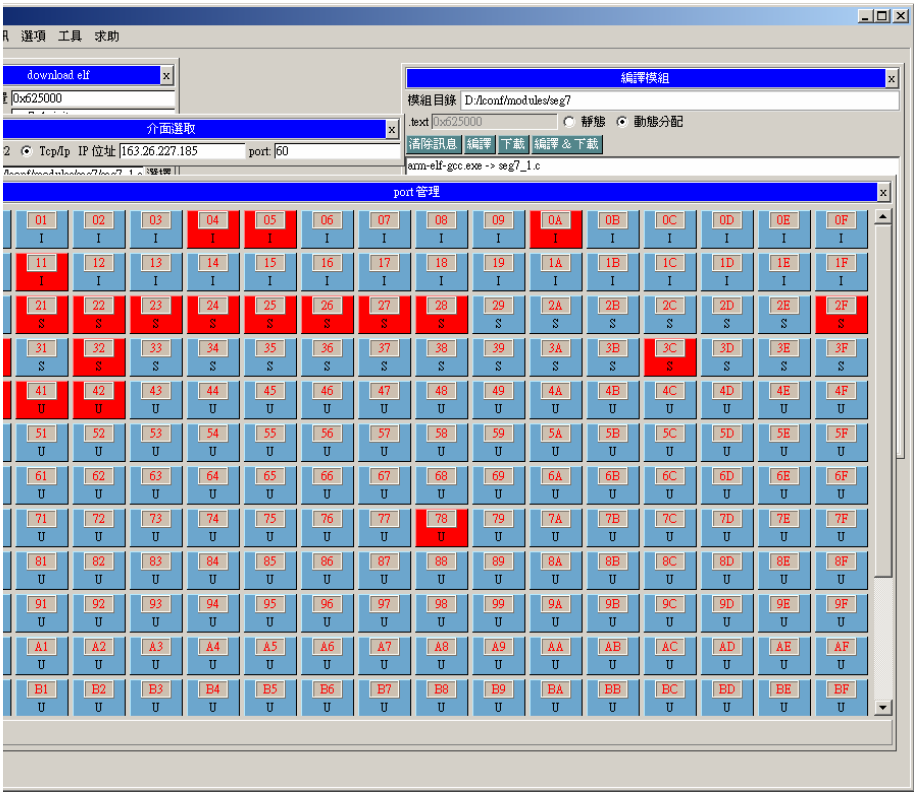
記憶體管理模組內提供了內建乙太網路控制器緩衝區之網路記憶體取得(network_get)與網路記憶體釋放(network_free)兩項命令，使網路模組達到零拷貝機制的功能。網路上最大訊框(frame)乘載量爲 1536 Byte，資料大於乘載量時必須先行切割。在系統記憶體管理每個分頁爲 2 Kbyte，爲了不使 512 Byte 無法使用必須審慎規劃；我們定義網路記憶體的使用只能有 1.5K 與 0.5K 兩種單位，0.5K 用來供上層較小的封包使用；任何應用程式要求配置網路記憶體時，使用上只能這兩種單位的倍數，圖六更進一步的說明記憶體管理者如何在 2K 內使用兩個單位的訊框，圖中也說明了預留的 64-Byte 空間的作用以及兩種訊框記憶體空間的最大資料使用空間分別爲 1460-Byte 及 448-Byte。

2.5 通訊模組

已完成的通訊模組有兩個，一個是使用速率較慢的 UART 模組，並不在這裡討論，另一個是乙太網路模組，任何與外界的通訊都需透過通訊模組再與系統內模組聯繫；因爲在開發板內的處理器內已經含有乙太網路控制晶片並且透過 MII（Media Independent Interface）介面與乙太網路解碼器連接，此種整合式乙太網路控制器本身並不提供資料緩衝空間，需要利用系統記憶體與其配合充當此網路控制器的資料緩衝空間，我們將它直接配合先前提到的網路記憶體管理配置所需資料緩衝空間的大小。一般傳統乙太網路模組在網路封包傳送處理的過程中需要一層層的拷貝，例如接收時系統預先分配一個記憶體空間，網路模組或 Socket 再將其搬入使用者預設的記憶體空間，花費 CPU 與記憶體資源，此種系統空間與應用程式空間壁壘分明的狀況並不一定全然適合嵌入式系統，尤其是我們強調的無 MMU 的系統。本文主要探討 TCP 以及下層的驅動程式，程式部份主要強調爲了充分發揮包含 MII 介面的整合式網路驅動晶片而設計，在驅動程式設計



圖七：網路晶片所使用的方法與資料結構



圖八：遠端工作站 HS-Console 程式界面

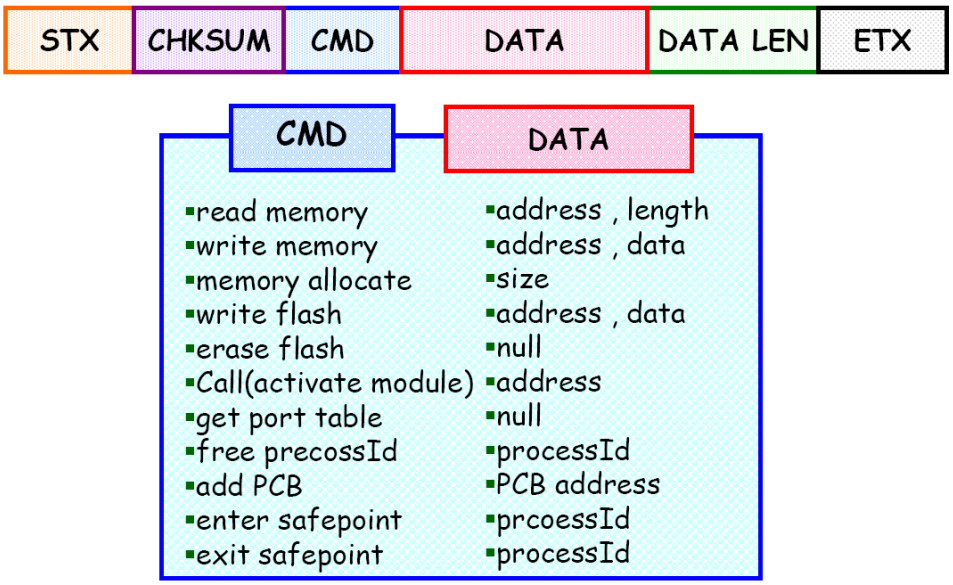
方面最重要的工作就是要填寫傳送以及接收的描述器(Rx/Tx Descriptor)，其所使用的方法與描述器的資料結構如圖七所示，在圖中，網路晶片上的 NetDMA 暫存器用以指向第一個描述器的位址，而數個描述器則以指標串接起來成一個環狀鏈結，網路晶片可以自動循序傳送或接收每一個描述器上的指標所指到的訊框(frame)資料，CPU 只需監督及控制描述器內的 ownership bit 即可。

2.6 命令代理模組(Command Proxy module)

命令代理模組可視爲本系統的 Shell，使用者可藉由此介面對系統內部通訊，例如要求某個模組進入安全點並停止執行、新增或更新模組，皆可透過命令代理模組對系統下達指令，本模組主要是聽命於遠端組態工作站上的 HS-Console 程式。HS-Console 負責下達命令，Command proxy 負責解析並依照所接收到的命令傳予各個對應該命令的模組。因爲之間的介面是乙太網路及使用 TCP/IP 協定，可以透過網際網路在遠端執行線上新增、換置或遷移等重新組態系統的動作。圖八爲 HS-Console 使用者界面，其中每一小格代表 IPC_port，紅色的 IPC_port 爲已經被佔用了的，小格內的字母，I 代表屬於硬體中斷 IPC_port，S 代表屬於核心 IPC_port，U 代表屬於使用者 IPC_port。HS-Console 是使用 CrowTDE [11]所開發，CrowTDE 是一個整合的 Tcl/TK 語言開發除錯環境，並且是由本文的第二作者所獨立開發，爲 GPL 下唯一的一個 Tcl/TK 軟體開發環境，目前除了中文與英文版外，還有德文、法文與義大利文版。

當電腦端的 HS-Console 程式透過 Ethernet 或是 UART 與 Command Proxy 模組連線時必需要有可溝通的機制，我們在 Ethernet 與 UART 中共用相同的通訊資料格式，如圖九所示。

由圖九的 CMD 命令上看來，許多功能並非爲了操作方便而設計，主要還是著重在雛型系統上的測試與驗證工作。使用目前的 CMD 項目，可以讓管理者對記憶體讀取與寫入資料、向記憶體管理者要求配置與釋放記憶體空間、寫入或抹除 Flash 硬體模組、執行一個任務、取得各個 IPC_Port 資訊、刪除任務、讀取記憶體資料寫入檔案、檔案資料寫到記憶體等命令。透過 HS-Console 程式界面可以取得本系統上運行的每個模組狀態與系統的負載量，更近一步的操作線上熱換置模組的動作。



圖九：Command Proxy 封包格式

2.7 任務管理者(Task Manager)

任務(Task)管理者主要用來管理系統上所有的任務，維護每一個任務的 PCB (Process Control Block)，系統上對於模組的新增、刪除或遷移動作皆必須藉助它的幫忙，相較而言，遷移模組比換置模組有著不同方面的考量，換置模組對任務管理者而言是新增與刪除的組合加上安全點的考量，其餘的新舊交接工作則大部分皆交由新模組自行轉移完成，至於遷移模組時，任務管理者還需紀錄模組記憶體頁、.text、.data、.bss、.rodata 段長度與模組初始化進入點等資訊到 PCB 內，必須深入了解 ELF[12]檔案格式，以便能夠重新定位。安全點的考量也是交由被換置者或是被遷移者自己負責，當模組收到 sofepoint_requst 訊息時，必須先確保自己身在安全一致性狀態時才能送出 safepoint_grant 訊息並且停止自身的正常工作等候進一步的要求訊息。

2.8 其他模組

其他還有一些系統必要的軟體模組，唯其特殊性不足，在此一併說明，包含一個排序器，一個計時器，一個儲存裝置，三個測試應用程式。

排序器模組預設方式是以優先權作爲選取法則，從任務串(Task List)中挑出一個最高的優先權來執行。由於一個任務在被建立時，就已經依據優先權高低插入至任務串內。因此排序器只需判斷此串列中最高的優先權任務(Task)狀態是否爲 Ready 以及沒進入安全點即可；若其狀態並非 Ready 或正在安全點上，則代表此任務尚未滿足執行的要求，可能正等待中斷發生、訊息回覆、離開安全點，或其他事件完成；此時排序器會依據串列中優先權次高的任務判斷其狀態，直到找出適合執行之任務爲止；假設整個串列一路搜尋下來沒有一個任務的狀態爲 Ready，那麼此時排序器就會找到在串列最末端，系統中最低優先權且狀態爲 Ready 的 Idle Task 來執行。目前我們還提供另一種 round-robin 排序器，可以在雛型系統上動態的換置掉預設的排序器，並感受得到系統上應用程式音樂播放的明顯不同。

計時器用以計時且是每一系統必備的，對於任務來說，當其需等待一段時間時，將會依據等待之時間寫入其 PCB 的 counter 欄位中，並將其狀態更改爲 Sleep。此時計時器模組的工作就是負責將 PCB 串列中狀態爲 Sleep 之任務內的 counter 值減 1，當該任務之 counter 值被計時器減至 0 時，代表此任務等待之時間已到，可將其狀態更改爲 Ready，讓此任務可獲得競爭 CPU 使用權的資格。

目前的儲存裝置提供了 SD、IDE、CF 三種儲存介面，SD 擁有體積小的優點，而 IDE 則提供高的儲存容量並且與 CF 介面控制操作相容，使用 LBA(Logical Block Addressing)方式的 IDE 介面最高可以定址到 13TB，可惜的是，目前三種都只提供到原生讀寫(raw read/write)功能，還欠缺與之配合的檔案系統(File system)。

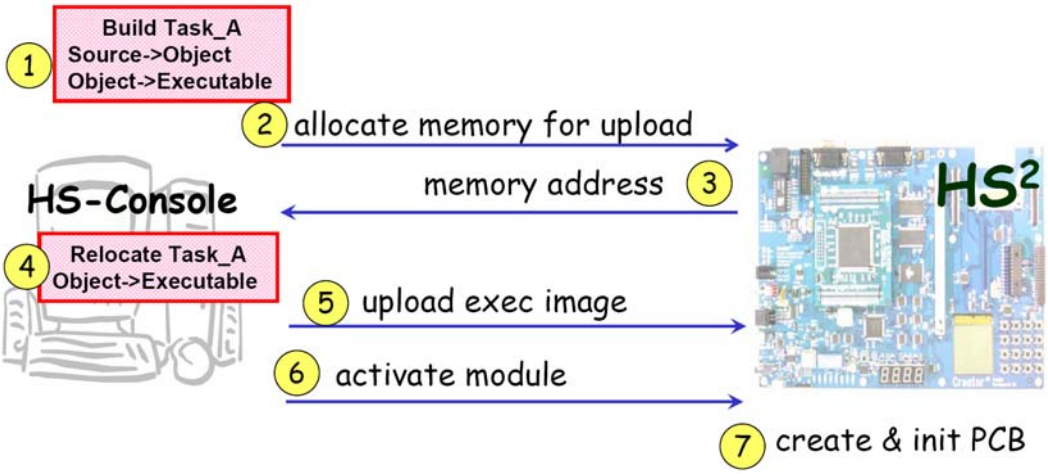
應用程式方面，對一個完全自行開發的系統，可用的應用程式相對薄弱；目前的應用程式主要是用來驗證與展示爲導向；計有一個 LED 跑馬燈，一個直接輸出小喇叭的音樂播放程式，以及一個網頁伺服程式，其中網頁伺服程式只具備最基本功能，可以直接回應‘GET’要求，網頁則顯示板子上的開關狀態並每隔幾秒更新畫面一次。以上應用程式皆用來做動態上載或遷移的驗證。

三、系統驗證

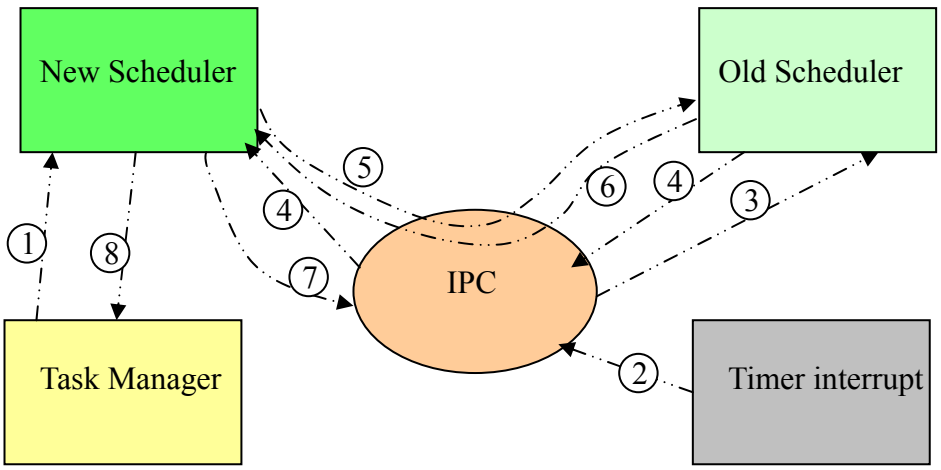
本章節將逐一說明新增、換置、遷移模組時所引發的系統操作與內部行爲；基本上，線上更換模組爲新增與刪除模組的組合，而線上遷移模組則除了完整保留執行中的映像資料外，還須考量此映像檔在新平台上的環境與重新定位，

3.1 線上新增模組

圖十說明要新增一個Task_A模組時的動作流程；首先HS-Console編譯Task_A原始程式碼，取得執行程式所需的記憶體大小(1)，再對HS^{2}發出訊息，要求在此系統上配置記憶體(2)，這個要求由命令代理者(Command proxy)執行後送回訊息，告知已配置的記憶體位址(3)，HS-Console再次連結定位成可執行檔(4)並送出訊息要求Command proxy 寫入記憶體(5)，HS-Console發出訊息啓動Task-A(6)，HS^{2}內透過任務管理者產生任務應有的資料結構並且併入排序(7)。



圖十：新任務模組 Task-A 上傳動作



圖十一：更排序器(Scheduler)模組的動作流程

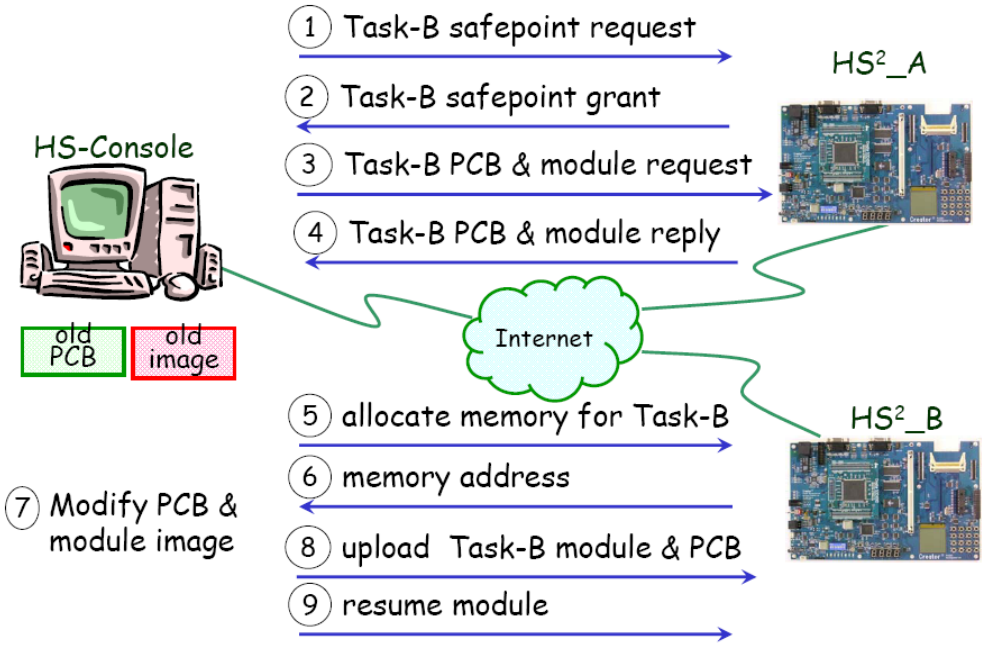
3.2 線上更換排序器模組

所有的模組換置動作應是一致的，本系統以更換排序器作爲分析說明與展示的方式，主要是因其較爲典型，因爲每當執行一次IPC後就需要重新排程一次，再加上所有的任務都須經過它的排程才能獲取CPU，所以該模組使用頻率極高，而且許多模組替換的動作爲新舊模組間自行交涉的結果，本例引發的動作可能較多。圖十一用以說明更換排序器模組所引發的動作過程。首先遠端組態工作站PC準備一個優先權與舊排序器相同的排序器，並使用HS-Console下載至HS^{2}系統上。此時在圖十一所引發的大致動作依序如下：

(1) Task Manager 接受遠端 HS-Console 的要求並且安插新的排序模組至 Task Queue 中。
(2) 當下一次的時間(Timer)中斷發生時，會將控制權交給行程通訊模組(IPC)。
(3) IPC 將控制權交給舊排序器選擇下一個可以執行的模組。

(4) 由於新排序器模組的優先權與舊排序器相同，排序程式將會挑選到此模組。
(5) 新的排序模組註冊暫時的 IPC_port，要求舊排序器進入安全點並詢問及取得所需資訊。
(6) 舊排序器回應其內部的資料結構，新排序將其轉換爲自己的排序任務串資料結構。
(7) 新的排序器取代舊有排序器 IPC_port 號碼，正式服務系統的排序工作。
(8) 通知 Task Manager 刪除舊有的排序器模組。

3.3 模組遷移



圖十二：遷移 Task-B 程序

遷移正在執行中的任務必須考量安全點以及映像檔在新環境的重定位，以求在新環境中執行的一致性，本系統將執行中的任務是否身在安全點的時機交由任務模組自行處理，因爲我們認爲最了解任務執行狀況的莫過於任務本身，是以任務在設計之初即應考量回復進入安全點的要求，直接強制性的遷移有可能導致不可預期的後果；圖十二說明Task-B從HS^2_A遷移到HS^2_B過程，上半部說明先要求Task-B進入安全點後再取得PCB及其模組映像檔，下半部的動作與圖十一類似，皆是新增模組到HS^2_B，只是對照圖十一的(4)連結定位須改爲修改PCB與映像檔位址參考，以及不似圖十一最後的從頭執行，而是依據PCB接續執行。

四、結論及未來工作

目前雛型系統可在兩種平台上執行，但是爲了測試與驗證，有些目的動作並不是一氣喝成；而且應用程式只有一個簡易的網頁伺服器與音樂播放程式，未來加入檔案系統模組後應可獲得某些程度上的助益。再來就是安全上的考量了，在目前階段我們並未實現任何保護措施，而這些措施在實用上是必要的，必須在未來加上一層安全機制。

未來的工作包含兩個方面，第一方面爲硬體模組的研究，硬體模組用以寫入Flash並改變FPGA零件功能與演算法；我們將探討傳統核心程式動態置入硬體的可行性，用以分擔傳統核心程式的部分工作，例如硬體排序器與硬體TCP的可行性研究。第二方面則是以HS^2爲平台，發展一個符合T10 規範[13]的物件儲存裝置OSD (Object-based Storage Device)；物件儲存系統可以將資料的儲存物件化，讓傳統的檔案系統不需要管理儲存的媒體細節；另外還有一個重要的概念就是將metadata與data分開儲存，Metadata Server處理存取頻率高但是資料量少的metadata，OSD負責處理存取頻率低但是需要大量I/O的資料部分，藉此分攤系統負擔，爲一種具有共享能力、高擴充性、可變尺度與包含智慧功能的儲存裝置。

參考文獻

[1] See web site http://www.linux.org/ .

[2] A. Baumann, J. Kerr, J. Appavoo, D. D. Silva, O. Krieger, and R. W. Wisniewski. “Module Hot-Swqpping for Dynamic Update and Reconfiguration in K42” *LCA 2005 linux.conf.au* .

[3] A. Baumann, J. Appavoo, D. D. Silva, J. Kerr, O. Krieger, and R. W. Wisniewski, “Providing Dynamic Update in an Operating System", *Proceedings of USENIX 2005,* Anaheim California, pp.279-291, *April 2005.*

[4] J. Appavoo, M. Auslander, M. Burtico, D. D. Silva, O. Krieger, M. Mergen, M. Ostrowski, B. Rosenburg, R. W. Wisniewski, J. Xenidis, “K42: an Open-Source Linux-Compatible Scalable Operating System Kernel”, *IBM Systems Journal,* Vol. 44, No. 2, pp. 427-440, 2005.

[5] See web site http://www.research.rutgers.edu/~edpin/epckpt/

[6] “Libckpt: Transparent Checkpointing under Unix”, *Proceedings of Usenix Winter 1995 Technical Conference,* New Orleans, LA, pp. 213--223, Jan. 1995.

[7] D.-W. Chang, R.-C. Chang, “OS Portal: an economic approach for making an embedded kernel extensible”, *Journal of Systems and Software*, Vol. 67, No. 1, pp.19-30, July 2003.

[8] See web site http://www.cse.unsw.edu.au/~disy/L4/

[9] J. J. Labrosse, *MicroC/OS-II :The Real Time Kernel,* CMP books, 2002

[10] A. S. Tanenbaum, *Modern operating systems 2ed,* Prentice Hall, Singapore, Nov. 2001.

[11] GPL software, *downloadable from sourceforge.net*, see web site http://sourceforge.net/projects/crowtde

[12] *Executable and Linkable Format (ELF)*, pdf file available on web site www.skyfree.org/linux/references/ELF_Format.pdf

[13] Working Draft, *Information technology – SCSI Object-Based Storage Device Commands*, also available on web site http://www.t10.org