視窗應用程式檢查點與錯誤回復機制之實作
Implementation of Checkpointing and Error Recovery Mechanisms for Windows Applications

蔡智強 教授
國立中興大學電機工程研究所
台中市國光路250 號
jichiangt@nchu.edu.tw

吳子靈
國立中興大學電機工程研究所
台中市國光路250 號
ericwu@ares.ee.nchu.edu.tw

陳衍堅
國立中興大學電機工程研究所
台中市國光路250 號
lawrence_1025@yahoo.com.tw

許玉霞
國立中興大學電機工程研究所
台中市國光路250 號
hsia@ares.ee.nchu.edu.tw

摘要

檢查點的容錯技術在許多的研究中都被廣泛的討論，尤其是應用在通訊與資料庫的設計中為最多。檢查點的基本觀念可以簡單的解釋為：將目標程式當時的狀態與資料記錄並且儲存起來，以作為將來錯誤發生時，能將其回復當時狀態的依據。

本論文選擇在Windows 作業系統平台上以軟體方式實作檢查點錯誤回復機制，整個系統實作可分為三部分，分別為檢查點運算、錯誤回復運算、記憶體區塊掃描，三個部分的實作程式碼都以C/C++程式語言建構起來。

本實作的目的在建立一個以軟體方式運作的檢查點錯誤回復系統，可以利用此檢查點程式對應用程式作處理，透過儲存執行緒的狀態資訊及掃描記憶體區塊的方式，將某執行程式的狀態與資料記錄並保存起來，此檢查點程式會將此資訊以檔案的方式存放在永久性的儲存體，以作為將來發生錯誤時回復應用程式到正常狀態的依據。

關鍵辭 ： 軟體容錯，檢查點，錯誤回復，Windows 作業系統架構

1. 序論

現今的個人電腦擁有驚人的潛力，且能提供無可限量的運算能力，但是擁有驚人的運算能力並不表示不會發生錯誤停擺的狀況。原因也許是軟體臭蟲，或者是硬體故障，也有可能是發生天災，最惱人的是人為的操作不當。

容錯，字面上的意思就是容許錯誤或容忍錯誤。也就是在錯誤發生時，此錯誤會被系統偵測到，而且系統會以特殊的機制來使得錯誤不會影響到整個系統的運作。而什麼是錯誤呢? 錯誤（Fault），可解釋成在系統的實體區域（Physical domain）中所發生的事件（Event）。這些事件的類型有：在硬體系統中的元件發生錯誤（Component Failure）、除以零、線路被持續施以固定電壓、行程被重新啟動、訊息（Message）在通訊通道（Communication Channel）中遺失、行程在與其他行程作溝通時，偶而的會失去訊息、行程任意反覆的啟動、輸入感應器腐化損毀、網路上負載的激增。[1]

容錯又有分成硬體容錯及軟體容錯。硬體錯誤通常都是元件發生錯誤，而硬體容錯的方式通常都是用一個新元件代替或是準備一個備用的元件。所謂的軟體容錯，所指的是利用一組軟體來偵測作業系統或硬體方面所無法處理的錯誤，以及從錯誤的情況加以還原。電腦發生錯誤的情形很多，根據應用程式的可用性（Availability）與資料的一致性（Data Consistency），來發展出偵測錯誤的方法以及還原的技術。[2]

錯誤發生會導致資料遺失，更嚴重者還會使得整個系統毀損，但是要避開種種的錯誤似乎是不太可能，所以我們必須使系統有容許錯誤發生的能力，也必須使系統有修復錯誤、再生的能力，也就是容錯最基本的概念。

論文章節安排如下：第一章，首先對本論文的研究動機與背景作說明。第二章，對實作的環境--Windows NT 作業系

統架構作一番的介紹。第三章，對軟體容錯技術--檢查點的實作技術作介紹。第四章，除了對本論文的實作方法作介紹之外，對檢查點實作、錯誤回復機制、及掃描所需要儲存記憶體空間的方法都作詳細的說明。第五章，對本論文的實作研究作結論，同時對未來可能的改良與研究方向提出一些建議。

2. Windows作業系統架構

Windows 經過許久的發展，已經從最初功能十分有限的16 位元作業系統演變成了現在的介面親切、管理方便、功能強大的作業系統家族，如Windows 2000、Windows XP。它之所以有這麼大的發展、除了微軟公司強大的商業推動力量之外，Windows NT 本身的技術因素也是重要的原因，尤其是它自身的系統架構所具有的可擴充性、可執行性、穩定性、相容性和效率提升，正是這些特性讓Windows 不斷得以進步。

本章將從作業系統設計的角度介紹Windows NT 所具有的基本系統架構和執行原理。

2.1 Windows NT 作業系統模型

Windows NT 透過硬體機制執行核心模式(Kernel Mode)以及使用者模式(User Mode)兩種權限分別。當作業系統為前者時，CPU 處於特權模式（Privileged Mode），可以執行任何指令，並且可以改變狀態；而在後者時，CPU 處於非特權模式，只能執行非特權指令。一般來說，作業系統中那些比較重要的程式碼都執行在核心模式，而使用者程式一般都在使用者模式下執行。當使用者程式使用了特權指令，作業系統就能借助硬體提供的保護機制限制使用者程式的控制權，必作出相對應的處理。[28]

Windows NT 的系統架構如圖2-1 所示。

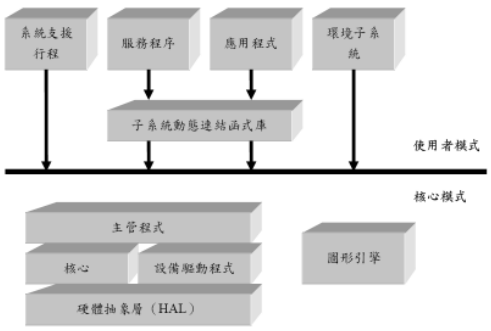

圖 2-1 Windows NT 系統架構圖。

2.1.1 Windows NT 的組成

Windows 作業系統大致上延襲著標準作業系統的架構，整個作業系統可分為內核層和外殼層，其實就是圖2-1 中的核心模式(Kernel Mode)和使用者模式(User Mode）。粗線以下是Windows NT 的核心模式元件，它們都在統一的核心位址空間中執行。

核心模式元件包括以下內容：

1. 核心（Kernel）：包括了最低階的作業系統功能，例如執行緒排程、中斷和異常排程、多處理器同步等，同時它也提供了主管程式（Executive）來執行高階結構的一組程式和基本物件。

2. 主管程式：包括了基本的作業系統服務，例如記憶體管理器、行程和執行緒管理、安全控制、I/O 以及行程間的通訊（IPC）。

3. 硬體抽象層（Hardware Abstraction Layer，HAL）：將核心、設備驅動程式以及主管程式跟硬體分隔開來，使得Windows作業系統得以實現跨越硬體平台的限制。

4. 設備驅動程式（Device Driver）：包括檔案系統和硬體設備驅動程式等，其中硬體設備驅動程式將使用者的I/O 函式呼叫轉換為對特定硬體設備的I/O 請求。

5. 圖形引擎：包含了圖形使用者介面（Graphical User Interface，GUI）的基本函式。

圖2-1粗線上部的方塊代表了使用者模式，它們是在私有位址空間中執行，使用者模式下有四種基本類型的行程：

1. 系統支援行程（System Support Process）：例如登入行程WINLOGON和會話管理器SMSS，它們不是Windows NT 的服務，不由服務控制器啟動。

2. 服務行程（Service Process）：它們是Windows NT 的服務，例如事件日誌服務。

3. 環境子系統（Environment Subsystem）：它們向應用程式提供執行環境（作業系統功能呼叫介面），

Windows NT 有四個環境子系統，Win32、POSIX、OS/2 1.2 和 Security。其中以Win32 子系統為中心，提供OS/2 1.2 及POSIX 兩個子系統作相容性對應的轉換。而Security 子系統則是為Win32 子系統提供身分認證的機制。

4. 應用程式（User Application）：它們是Win32、Windows 3.1、MS-DOS、POSIX 或OS/2 1.2 這五種類型之一。此層也就是一般使用者的應用程式執行時所在的位置，在這一層運作的應用程式必須依賴各個子系統提供的API 函式來使用系統所提供的各項資源。

2.1.2 Windows NT 的可攜性

Windows NT 的設計目標之一就是能夠在各種硬體系統架構上執行，它用兩種方法執行了對硬體結構和平台的可攜性。

第一個方法是分層的設計，依賴於處理器系統架構或平台的系統底層部分被隔離在單獨的模組之中，系統的高層可以被遮罩在各種不同的硬體平台之外。提供作業系統可攜性的兩個關鍵元件是HAL 和核心，依賴於系統架構的功能（如執行緒描述表切換）在核心中執行，在相同系統架構中，因電腦而異的功能則在HAL 中執行。

第二個方法是Windows NT 幾乎全部使用高階語言寫成--主管程式、應用程式和設備驅動程式都是用C 語言編寫的，圖形子系統部份和使用者介面是用C++編寫的。只有那些必須和系統硬體直接交換訊息的作業系統部分（如中斷陷阱處理程式），或效能極度敏感（如描述表切換）的部分是用組合語言編寫的，組合語言代碼主要分佈在核心及HAL 中。

3. 軟體容錯技術

所謂的軟體容錯技術，所指的是利用一組軟體來偵測作業系統或硬體方面所無法處理的錯誤以及從錯誤的情況加以還原。為什麼軟體容錯會比硬體容錯複雜呢？主要的原因在於軟體錯誤狀態種類繁多、集合眾多元件，使得考量的因素增多。軟體系統通常動輒牽扯數百萬的互動式計算元件，同時軟體系統所結合的一些性質，在軟體系統中要達到完美的設計，是很困難的；而要做到錯誤偵測更有其難度。本章將討論軟體容錯的技術，以及軟體容錯的優點，並對本論文實作所採用的方法，加以詳細的介紹。

3.1 軟體容錯機制

現今科技生活中，對於應用程式軟體容錯能力的需求越來越大，有容錯能力的應用程式，可以去偵測到錯誤，並且能從應用程式底層硬體及作業系統所無法解決的錯誤回復回正常狀態。

近年來，越來越多廣為人們使用的應用程式陸續出現在Windows NT 上，因此軟體容錯越趨重要，實作在Windows NT 上的容錯軟體一般稱之為NT-SwiFT（Software Implemented Fault Tolerance）。而這些軟體提供了完整錯誤偵測及回復、檢查點運算、事件紀錄與重現、通訊錯誤回復、遞增資料備份以及IP 封包的重送等。[2][3][4][11][12][20]

3.2 檢查點容錯機制

檢查點是容錯中的一種機制，它最主要的功能，在於能夠允許使用者去保存目前執行中行程在任何時間點上的狀態資訊，當有錯誤發生時，能透過上回最後一次作檢查點時的狀態資訊，回復到正常執行的狀態。[1][2][3][4][5]

檢查點的容錯方式除了可以不用針對個別的程式邏輯或使用者操作方式作錯誤的檢查與預防外，在對硬體故障所引發的錯誤回復需求上也有異地復原的解決方案。[6][12]

3.2.1 Windows 平台上檢查點資料收集

從檢查點的應用方式來看，大多數的相關研究有兩種不同的實作機制。第一種，先建立好檢查點函式庫（Checkpoint Library），供程式開發時在加入檢查點的功能；第二種，採用攔截API（Intercept API）的技術，直接對執行中的程式植入檢查點運算的功能。這兩種實作技術在應用層面上雖有不同，但核心部分卻都是一樣的，主要的目的都是在收集並且儲存目前正在執行中程式的檢查點資訊。而本文所採納的方法是屬於檢查點函式庫這個方法。

3.3 應用程式的行程

在作業系統中一個正在執行中的程式，我們一般稱之為行程（Process）[7][28]，而在Windows 作業系統中一個一般的使用者行程所會用到的系統資源大概有以下幾個大項：

◆ 記憶體：行程擁有自己獨立的32 位元

定址空間。

◆ 執行緒：一個行程中可以有多個執行緒共用行程資源。

◆ 檔案與目錄：行程中開啟讀或寫的檔案資源。

◆ 視窗物件：如視窗圖示、視窗功能表、視窗對話框等。

◆ 通訊物件：如Socket、Pipe 等。

◆ 輸出輸入裝置：如印表機裝置。

◆ 同步共用物件：號誌（Semaphore）、互斥（Mutex）等系統物件。以上的系統資源除了記憶體空間是程式載入為行程的時候就由系統配置之外，其餘資源的取得都要由行程透過系統函式的叫用才能取得。

3.4 行程的記憶體空間

行程在Windows 作業系統中獨占線性的32 位元定址空間，但更精確的說法是32 位元的虛擬記憶體空間。Windows 的記憶體管理採用的方式為：

1. 需求分頁（Demand Paging）：以4Kbytes 為分頁單位，行程記憶體分頁的實體配置位置交由虛擬記憶體管理模組管理，在有需要的時候由磁碟的記憶體交換區取出記憶體分頁，將其載入實體記憶體中供行程使用。
2. 虛擬記憶體（Virtual Memory）：藉由使用磁碟上的記憶體交換區，提供行程大於實體記憶體容量的虛擬定址空間作運算。
3. 32 位元行程定址空間（32-bit Addressing Space）：在行程中支援完整的4GB 定址空間，其中低位址空間中的2GB 供使用者行程使用，而高位址空間中的2GB 則保留給核心模組使用。

就Windows 作業系統中行程所使用的2GB 虛擬記憶體空間而言，檢查點運算要實作的部分就是要在這個2GB 的記憶體空間中找出需要儲存的記憶體區塊。由於Windows 中對於每一個記憶體區塊皆紀錄有該區塊的狀態與使用資訊，我們將在實作中還會作更詳細的說明。圖3.1 所示為Win32 記憶體管理階層圖。[8][28][29]

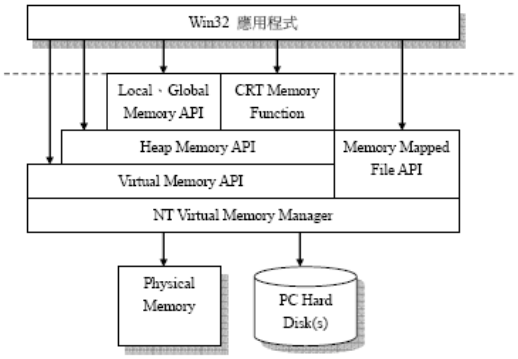

圖 3.1 Win32 記憶體管理階層圖。

3.5 Windows虛擬記憶體空間的配置與保護

虛擬記憶體的主要功能是在位址空間中保留一個區域、從分頁檔中為該區域安排一塊實體記憶體、並為該實體記憶體設定其保護屬性。要利用記憶體必須先將其配置一塊空的記憶體，首先呼叫VirtualAllocEx( )函式配置一塊區域（Region），一塊區域是以一個配置間隔邊界（Allocation granularity boundary）為起點，也就是64KB 為範圍。如圖3.2 所示。一個區域的大小是一個頁面（Page）大小的倍數，在Windows NT 中也就是4KB 的倍數。例如，若你想要保留一個10KB的位址空間區域，系統會自動幫你配置成12KB。[9][29]

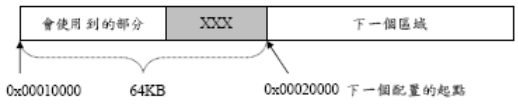

圖 3.2 記憶體區域的配置。

要使用一個已經保留的記憶體區域之前，必須要先配置一塊實體記憶體。要知道記憶體的配置情況，可以透過VirtualQueryEx( )函式來查詢，並設定記憶體屬性，之後再將實體記憶體映射至這已保留的區域，這部分的設定一樣可以透過VirtualAllocEx( )函式來完成。當不再使用記憶體時，必須使用VirtualFreeEx( )函式來將配置的記憶體釋放掉，指定MEM_RELEASE 則記憶體狀態會變為可用的（FREE）；指定MEM_DECOMMIT，則記憶體狀態會變為保留的（RESERVE）。

屬於某行程的記憶體會很明確地被自己私有位址空間所保護。除此之外，Windows 也藉由虛擬記憶體硬體來提供記憶體的保護，這些保護的做法會隨著處理器而改變，例如在一個行程位址空間中

的程式碼區段可能會被使用者模式下的執行緒改為唯讀及被修改成受保護的。[14][15][29]

Windows 所提供的記憶體保護有以下幾種，當我們配置或保護一塊記憶體中頁面時，可以選擇其中一項來指定存取權限。

- PAGE_EXECUTE，允許執行有提交實體記憶體的頁面區域；
- PAGE_EXECUTE_READ，允許執行、讀取有提交實體記憶體的頁面區域；
- PAGE_EXECUTE_READWRITE，允許執行、讀取、寫入有提交實體記憶體的頁面區域；
- PAGE_EXECUTE_WRITECOPY，允許執行、讀取、寫入有提交實體記憶體映像檔程式碼頁面的區域，此頁面可以共享為read-on-write及copy-on-write；
- PAGE_NOACCESS，不允對有提交實體記憶體的頁面區域作任何存取；
- PAGE_READONLY，只能允許讀取有提交實體記憶體的頁面區域，
- PAGE_READWRITE，允許讀取、寫入有提交實體記憶體的頁面區域；
- PAGE_WRITECOPY，對有提交實體記憶體的頁面區域給予copy-on-write 的保護。

4. 檢查點錯誤回復實作

由Windows NT 作業系統架構得知，要對於一個應用程式作檢查點的運算可以有兩種切入方式，一種從核心模式切入，另一種是從使用者模式切入。

這兩種方式的運算邏輯完全不同，所需要的條件也各不一樣，但是其中有一項考量十分關鍵，那就是安全。從核心模式下切入時就必須跳過主管程式中安全控制的權限，直接在系統服務製作一個裝置驅動模組來實施檢查點的運算。這個意思就是說檢查點的操作在特權模式（Privileged Mode）下，因此對於實作中各部分的安全性檢驗與查核必須由此實作負責處理。如此的做法除了很難評估會對Windows 系統的整體安全與一致性有什麼樣的影響之外，對於Windows 系統中權限與安全機制維護工作會造成沉重的負擔，也容易使得原本要作的檢查點運算目的失焦。因此採用使用者模式的檢查點運算就可以免去這些負擔，只要把檢查點運算的實作焦點放在執行檢查點運算時期相關資料的處理即可。[19][23]

4.1 實作環境

實作檢查點錯誤還原系統時，在程式語言方面使用了C 語言，編輯器使用MSVC（Microsoft® Visual C/C++），開發環境為MSVS（Microsoft® Visual Studio 6.0），作業平台為Microsoft® Windows XP Professional。[23][24][25][26[27][29][30]

4.2 實作方法

WinCkp 的實作方法主要透過使用Win32 API 來建構整個架構，WinCkp 也大量使用了Win32 函式，而這些函式可以讓一個行程去改變另一個行程的狀態。

4.2.1 檢查點（Checkpointing）與回復（Rollback）

在WinCkp 這個專案中，我們透過CreateProcess( )此函式來啟動應用程式。啟動應用程式之後，若要對此應用程式執行檢查點運算，必須先暫停所有執行緒，且必須將目前狀態的資訊存到一個檔案中，而這個檔案在實作中名為ckplog。

在作回復運算期間，也必須先暫停所有的執行緒，然後再透過之前對其作檢查點運算所存放的檔案（ckplog）恢復之前的狀態資訊，最後，WinCkp 才會重設（Reset）執行緒內文（Thread Context），再恢復執行緒的執行。

WinCkp 會在目標程式（指應用程式）注入檢查點運算的執行緒，而這個執行緒所負責的部分就是將系統函式相關的資訊保存到一個穩定的儲存體內，而在返回或錯誤回復時，也能將這些系統函式的功能重現（Replay）。[18][19]

4.2.2 執行緒(Threads)

執行WinCkp 的行程與執行應用程式的行程是獨立且不同的。在NT 系統系列中，允許一個行程去取得及設定不同行程內執行緒的執行緒內文。要取得一個執行緒內文，必須使用GetThreadContext( )來獲得；而要去改變及回存執行緒內文，必須透過SetThreadContext( )來設定。[21]

4.2.3 記憶體(Memory)

WinCkp 會儲存應用程式的記憶體資訊，包括資料、堆積區、堆疊區。要取得一個行程記憶體映像，必須使用 ReadProcessMemory( )來獲得；而要回存修改後的記憶體，則必須透過 WriteProcessMemory( )來設定。

之前提到一個NT 系統的行程會有大概2GB 的私有位址空間，而範圍是從 0x00010000 到0x7FFEFFFF，但在這個空間上並不是每個區域都需要被儲存的。必須透過VitualQueryEx( )這個系統呼叫來一個一個區域的檢查位址空間，只有在讀取以及寫入存取是啟動的（enabled）及實體儲存體被提交（commit）時，就必須儲存記憶體區域。而Winckp可儲存每一個記憶體區域以及 MEMORY_BASIC_INFORMATION 結構。[8][16][28][29]

4.2.4 檔案（Files）

為了作檢查點檔案及從檔案回復，WinCkp 使用檔案儲存的方法將執行緒內文（Thread Context）、應用程式視窗資訊、記憶體資訊、以及檢查點資料儲存起來，將透過C 函式中的fopen( )、fclose( )、fread( )、fwrite( )函式來達成。在實作中，將有三個檔案分別用來存放這些資訊（hwndlog、contextlog、ckplog）。

4.2.5 視窗物件（Window Objects）

WinCkp 會去儲存應用程式的視窗物件代碼，像是應用程式的視窗、視窗對話框等。視窗代碼與其他GDI 物件的回復會有一個較大的困難點，就是介於這些物件的相依性，且它們並不能從其他的行程或核心中隔離出來。

為了要回復一個視窗代碼，WinCkp 首先會啟動一個新的行程，讓這個行程產生所有相關的視窗。之後，WinCkp 會在舊的視窗代碼與新的視窗代碼間產生映射，在回存到記憶體前，用新的代碼值來取代檢查點檔案中舊的代碼值。它可適用Windows 上的應用程式，如踩地雷（winmine.exe）、接龍（sol.exe）、小畫家等。[5][19]

4.3 檢查點運算

在Windows NT 中，使用者所參考到的記憶體定址空間稱之為虛擬記憶體空間，在之前的章節已經有作過說明。在這個章節，我們要對應用程式執行緒與記憶體空間的內容作比較詳細的處理，以便能在作檢查點運算時，將需要的資料儲存起來。

4.3.1 行程執行緒的暫停與繼續

若要對執行中的應用程式作檢查點運算之前，必須要能使此行程執行緒暫停（Suspend），之後才能做後續的處理。

我們採用的函式是 SuspendThread( )，它會暫停一個執行緒的動作，也就是會停止執行應用程式的動作，而此函式必須傳入此行程執行緒的代碼，這個代碼是透過 PROCESS_INFORMATION 結構來指定的。

暫停執行緒，做完行程記憶體與資料的處理之後，必須恢復執行緒的執行，則透過ResumeThread( )函式來恢復應用程式執行緒的執行。

4.3.2 執行緒的狀態資訊

在行程執行緒被暫停之後，此時的目標行程已經完全停止動作，除了要儲存行程資料、記憶體頁面區域之外，還要將目前的這個行程中執行緒的執行資訊儲存起來。

我們將透過Windows 提供的兩個系統函式， 一個是GetThreadContext( )，在暫停執行緒後用來取得執行資訊；一個是SetThreadContext( )，在恢復執行緒前用來設定執行緒的執行資訊。

4.3.3 行程記憶體區塊的掃描

行程在記憶體至少有兩個資料區塊，分別為堆積區與堆疊區，這兩個區塊內所存放的內容是行程的變數資料。堆積區中為行程的全域與靜態變數區；堆疊區中則是行程的區域變數存放區。除了這兩區之外，執行時期由系統記憶體管理函式所配置出來的記憶體區塊則是零散的分佈在行程的記憶體定址空間中。

因此必須對整個記憶體空間作一次掃描，將其中屬於資料的記憶體區塊儲存起來。不過在掃描記憶體之前，要先暫停行程中的執行緒，否則儲存下來的記憶體資料區塊可能因為有其他執行緒正在作讀寫的運算，而造成資料的不一致（Inconsistency）。

4.3.4 篩選記憶體區塊的條件

對一個行程來說，Windows 作業系統下的行程雖然使用完整的32位元的定址空間（4GB 位元組），但事實上可供行程使用的定址空間是被限制在0x00010000 到0x7FFEFFFF（約2GB 位元組），因此我

們要在這約2GB 的空間中找出需要儲存的區塊，但並非真的對這2GB 中的每一個Byte 去作搜尋， 而是利用了Windows 系統函式VirtualQueryEx( )來達到我們的目的。

在應用程式行程2GB 記憶體空間，有哪些資訊是屬於檢查點在作錯誤還原的必要資訊呢？很明顯的答案是程式的全域、靜態以及區域變數。因為這些變數紀錄了程式在執行時期的行程狀態，能收集這些變數的的正確資訊是能夠正確還原檢查點的要素之一。

在Windows 中的行程所配置的記憶體被系統切割為許多大小不一的區塊，每個區塊都有相對應的屬性，這些區塊的資訊都紀錄在MEMORY_BASIC_INFORMATION 結構中。

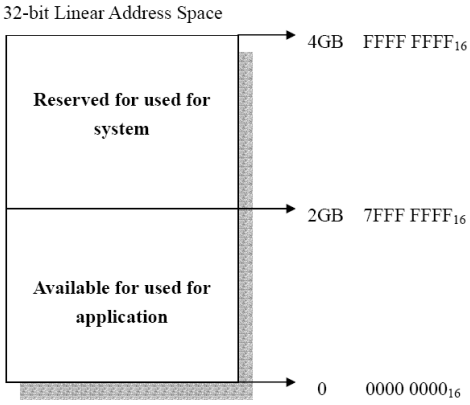

圖4.2 Windows 虛擬記憶體空間

Windows 在這2GB 的虛擬記憶體空間中採區塊的方式配置，各區塊分別以類別（Type）、狀態（State）、保護（Protect）來作區分。

- 類別（Type）：表示該虛擬記憶體區塊內資料的屬性，有IMAGE、MAPPED、以及PRIVATE 三種資料屬性。
- 狀態（State）：表示該虛擬記憶體區塊的實體記憶體配置狀況，有COMMIT、RESERVE、以及FREE 三種狀況。
- 保護（Protect）：表示該虛擬記憶體區塊被允許的操作方式，有READONLY、READWRITE、EXECUTE、…等。

只要這三種屬性中的任意一種不同就不會合併成為一個連續的區塊。更正確的說法是，即使某一個區塊的尾與另一區塊的頭是連續的，但是Windows 系統函式VirtualQueryEx( )並不會將其當作一個連續的區塊，除非這兩區塊的三種屬性一樣。

需要注意的一點是，在掃描記憶體時，記憶體區塊是允許可讀寫（Protect 為PAGE_READWRITE），以及有相對應的實體記憶體（State為MEM_COMMIT)時，就必須將此記憶體區塊連同MEMORY_BASIC_INFORMATION 結構儲存起來， 同時利用ReadProcessMemory( )來讀取行程中記憶體區塊裡的資料，並將之儲存到所開啟的檔案中，這就能將檢查點的資料儲存起來。

4.4 錯誤還原計算

將執行緒內文、記憶體區塊及寫入記憶體的資料通通存入到所開啟的檔案中，這就是檢查點運算負責的功能。而當檢查點資料儲存成功之後，要如何在開啟的新行程中將存放的資料複製進來，讓新行程擁有舊行程作檢查點時的狀態，這就是此節所要討論的重點。

首先是執行緒的狀態，新執行緒啟動時，必須先將其動作暫停，再將之前所存的內容取出，並利用SetThreadContext( )函式來設定新執行緒內文的狀態。其次是記憶體區塊狀態的復原， 先讀出所儲存的記憶體資訊， 判斷記憶體的保護必須為可讀寫的（ Protect 為PAGE_READWRITE），讀取紀錄檔中記憶體的內容，並設定其基底位址，修改記憶體狀態之後，再將此記憶體內容寫入行程的記憶體中。

若無法寫入，先透過VirtualQueryEx( )函式取得另外一塊記憶體的資訊，再透過VirtualAllocEx( )函式來設定配置記憶體區塊類別為有相對的實體記憶體（MEM_COMMIT），最後再透過WriteProcessMemory( )函式將其寫入行程的記憶體。

若虛擬記憶體並無實體記憶體對應時（MEM_RESERVE、MEM_FREE ）， 就必須將其設定成有實體記憶體對應（MEM_COMMIT）。如果狀態是為保留時，可直接透過VirtualAllocEx( )函式將其配置的類別改為有實體記憶體對應；如果狀態是為釋放時，就必須將之前所另外配置的記憶體區塊資訊傳入VirtualAllocEx( )，並將其類別設定為保留，之後再將其類別改為有實體記憶體對應。修改記憶體保護後，也順利寫入行程的記憶體，最後再將行程執行緒繼續動作，這就是錯誤回復最主要的部分。

5. 結論與未來展望

5.1 結論

本論文在Windows 作業系統平台建立起一個完整的檢查點運作的基礎模型，將檢查點運算所需要的功能一一給於明確的定義，並完成了在Windows 平台上實施檢查點運算技術的核心功能。經過多個應用程式的測試，幾乎所有Windows 平台上的應用程式都能透過此實作，達到檢查點錯誤回復的功能。此實作方法雖與大多數檢查點技術相去不遠，但卻不必透過複雜的機制，也能達到一樣的功能。

以本實作的模組架構為基礎，對於要進一步研究或是其他的實作應用，應該都有不少的幫助。

5.2 未來展望

本論文所作的實作，在檢查點資料的收集方面只針對行程中單一行程執行緒與記憶體作處理，而在多執行緒、視窗元件等其他方面的資訊，可以作更進一步實作研究[18][21]；可結合網路部分，以檔案存放方式，放置在不同主機上，以達到遠端回復的效果；可以移植到其他較小的作業系統上，例如WinCE，透過無線網路來達到行動式檢查點的功能；也可針對檔案存取作相關的控制應用，例如檔案的加密保護、檔案的壓縮/解壓縮。

參考文獻

[1] Ali Ebnenasir, "Software Fault-Tolerance", Computer Science andEngineering Department Michigan State University U.S.A http://www.cse.msu.edu/~cse870/Lectures/SS2005/ft1.pdf

[2] P. Emerald Chung, Yennun Huang, Chandra Kintala, Woei-Jyh Lee,Deron Liang, Timothy K. Tsai, and Chung-Yih Wang. "*NT-SwiFT: Software Implemented Fault Tolerance on Windows NT*", 2nd USENIX Windows NT Symposium, August 1998.

[3] Youhui Zhang, Dongsheng Wang, and Weimin Zheng. "*Transparent Checkpointing and Rollback Recovery Mechanism for Windows NT Applications*", April 2001, pp. 78-85.

[4] Johny Srouji, Paul Schuster, Maury Bach, and Yulik Kuzmin. "*A Transparent Checkpoint Facility on NT*", 2nd USENIX Windows NT Symposium, August 3-4, 1998.

[5] P. Emerald Chung, Woei-Jyh Lee, Yennun Huang, Deron Liang, and Chung-Yih Wang. "*Winckp: a Transparent Checkpointing and Rollback Recovery Tool for Windows NT Applications*", Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium, June 1999, pp. 220-223.

[6] Tom Boyd and Partha Dasgupta, "*Process Migration: A Generalized Approach Using a Virtualizing Operating System*", Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on, 2-5 July 2002, pp. 385-392.

[7] Jeffrey Richer, "*Process*", Chapter 4, in *Programming Applications for Microsoft Windows*, Fourth Edition, Microsoft Press, 1999.

[8] Jeffrey Richer, "*Win32 Memory Architecture*", Chapter 13, in *Programming Applications for Microsoft Windows*, Fourth Edition, Microsoft Press, 1999.

[9] Jeffrey Richer, "*Using Virtual Memory in Your Own Applications*", Chapter 15, in *Programming Applications for Microsoft Windows*, Fourth Edition, Microsoft Press, 1999.

[10] David A. Solomon and Mark E. Russinovich., "*Inside Microsoft Windows 2000*", 3rd Edition, Microsoft Press, 2000.

[11] J. Gray and A. Reuter, "*Transaction Processing: Concepts and Techniques*", Morgan Kaufmann Publishers, 1993.

[12] H. Abdel-shafi et al, "*Efficient User-Level Thread Migration and Checkpointing on Windows NT Clusters*", 3rd USENIX Windows NT Symposium, 1999.

[13] J. Plank, M. Beck and G. Kingsley, "*Libckpt: Transparent Checkpointing Under UNIX*", 1995 Usenix Conference.

[14] MSDN Library-January 2001, Microsoft Corporation, 2001.

[15] MSDN Library, "*Memory Protection*", Platform SDK: Memory Management. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/memory/base/memory_protection.asp

[16] MSDN Library, "*MEMORY_BASIC_INFORMATION*", Platform SDK: Memory Management. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/memory/base/memory_basic_information_str.asp

[17] Microsoft Foundation Class Library, NSDN Home, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcmfc98/html/mfchm.asp

[18] Jin-Min Yang, Da-Fang Zhang and Xue-Dong Yang, "*User-level implementation of checkpointing for*

multithreaded applications on Windows NT", Test Symposium, 2003.ATS 2003. 12th Asian, 16-19 Nov. 2003, pp. 496-499.
[19] Y. Wang, Y. Hung, K. Vo, P. Chung and C. Kintala, "*Checkpoint and its applications*", Proceedings of the 25th IEEE Fault Tolerant Computing Symposium, Pasadena, California, pp. 22-31, 1995.
[20] Diamantino Costa, Joao Carreira and Joao Gabriel Silva, "*WinFT: Using Off-the-shelf Computers on Industrial Environments*", Emerging technologies and Factory Automation Proceedings, 1997. ETFA '97., 1997 6th International Conference on, 9-12 Sept. 1997, pp. 39-44.
[21] Thuan Q. Pham, Pankaj K. Garg, "*Multithreaded programming with Win32*", Prentice Hall PTR, 1999.
[22] Seung-Woo Kim, "*Intercepting System API Calls*", http://www.devx.com/Intel/Article/21023
[23] Johnson M. Hart 著，黃昕暐 譯，"*Win32 應用程式設計聖經*"，和碩科技文化有限公司，1998
[24] H. M. Deitel, P. J. Deitel 著，吳國樑譯，"*C 程式設計藝術-第三版*"，全華科技圖書股份有限公司，2001
[25] Richard C. Leinecker, Tom Archer, Kevin Smith, Lars Klander, Derrel Blain, Garrett Pease 著，李奇 譯，"*Visual C++ 6 Bible-基礎與程式架構篇*"，文魁資訊股份有限公司，1999
[26] Richard C. Leinecker, Tom Archer 著，黃怡 譯，"*Visual C++ 6 Bible-進階與程式應用篇*"，文魁資訊股份有限公司，1999
[27] 蔡明志 著，"*Win32 程式設計實務-使用Visual C++*"，松崗電腦圖書資料股份有限公司，1999
[28] 尤普元，史美林等著，吳亞秀 譯，"*作業系統原理-Windows 核心剖析*"，全華科技圖書股份有限公司，2004
[29] 井民全，"*Advanced Windows Programming*"，http://debut.cis.nctu.edu.tw/~ching/Course/AdvancedC++Course/__Page/Windows_Programming.htm
[30] 李海，"*問專家-C/C++ - Windows API*"，http://www.china-askpro.com/cpp11.shtml