

真實模式定址

- Intel CPU 硬體支援的記憶體定址模式至少有真實模式與保護模式兩種，後來加入的AMD也不得不遵循。
- 真實模式 Real Mode
 - 16位元CPU,有20條的位址線(a0-a19)，所能處理的最大容量為2的20次方，亦即 1048576 Bytes
 - $1048576 / 1024 = 1024 \text{ Kbytes} = 1\text{Mbytes}$
 - 定址方式：由兩個16位元暫存器的內容經處理後，變成20位元的絕對位址
 - 表達方式：0000:0000 => 0x00000 ~ 0xFFFFF

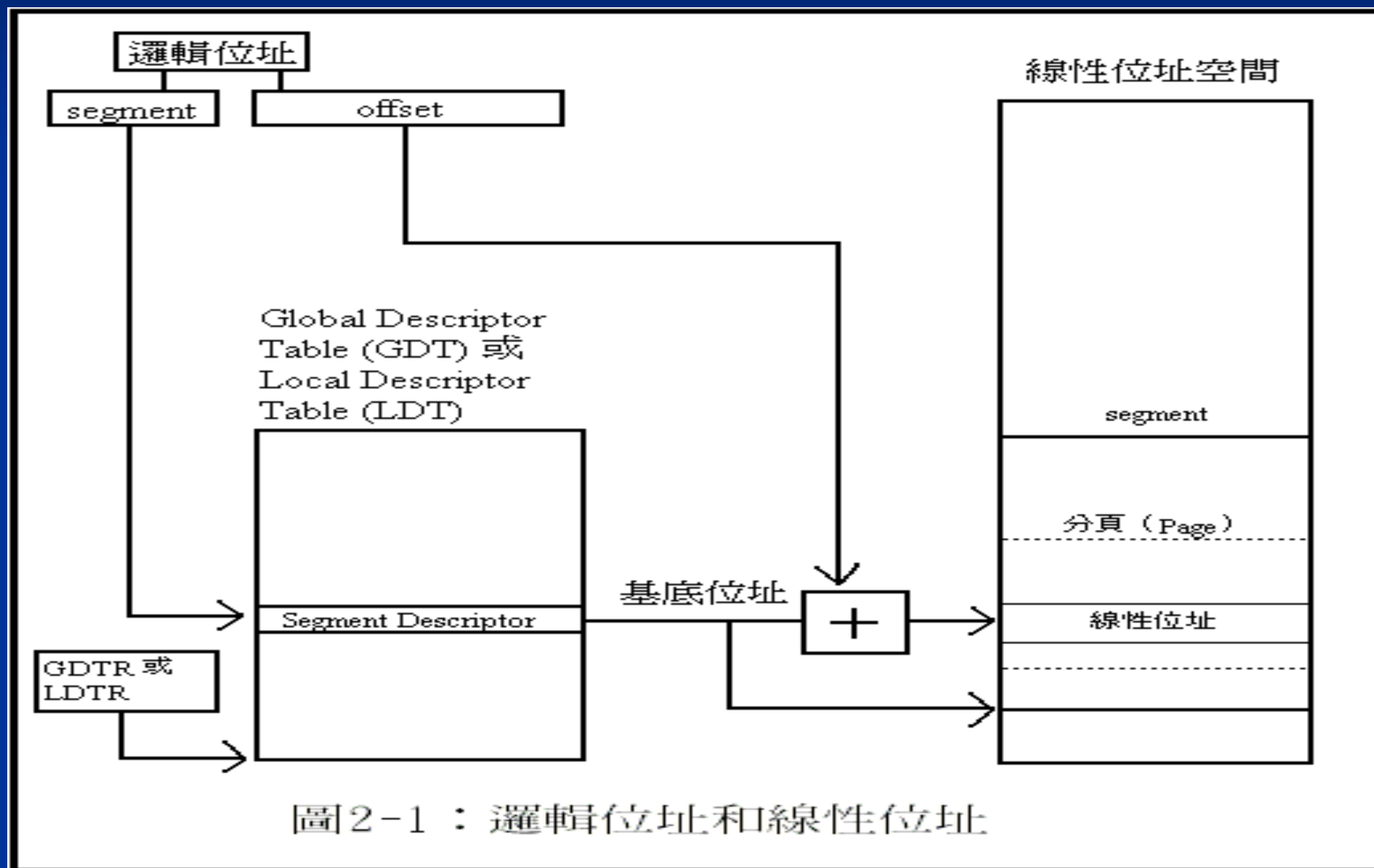
真實模式記憶體配置

00000H
中斷向量表
00400H
系統使用區
DOS
使用者程式區
A0000H
螢幕圖形資料區
B0000H
螢幕文字資料區
C0000H
其它介面卡控制區
F0000H
BIOS ROM
FFFFFFH

保護模式定址(1)

- 保護模式 Protect Mode：由CPU硬體進行記憶體의存取定址與保護機制，有三種定址方式
 - 實體位址（Physical Address）
 - 32位元CPU可實際定址4GB（2的32次方）
 - 0x00000000～0xFFFFFFFF
 - 線性位址（Linear Address）
 - Non-Paging 時，線性位址內容=實體位址內容
 - 有Paging時，某線性位址的內容可能在Swap中，不在實體位址中
 - Swap 由 OS 負責處理
 - 邏輯位址（Logical Address）：一般AP使用的定址法
 - 定址表示 Segment：offset
 - Segment => Segment Selector，16位元暫存器，指定Segment Descriptor號碼
 - Offset => 32位元暫存器，表示在Segment中的偏位移值
 - Segment：Offset等於Segment的基底位址加上Offset 就可以得到線性位址。
 - Segment Descriptor：GDT/LDT中的Record，描述一個 segment 的位置(Base Address)、大小、型態、存取權限等等資料
 - GDT/LDT：陣列的基底位址由GDTR/LDTR暫存器指定，藉由Segment Selector 指示的Index Number即可存取Segment Descriptor Record

保護模式定址(2)



保護模式定址(3)

■ 保護模式分頁

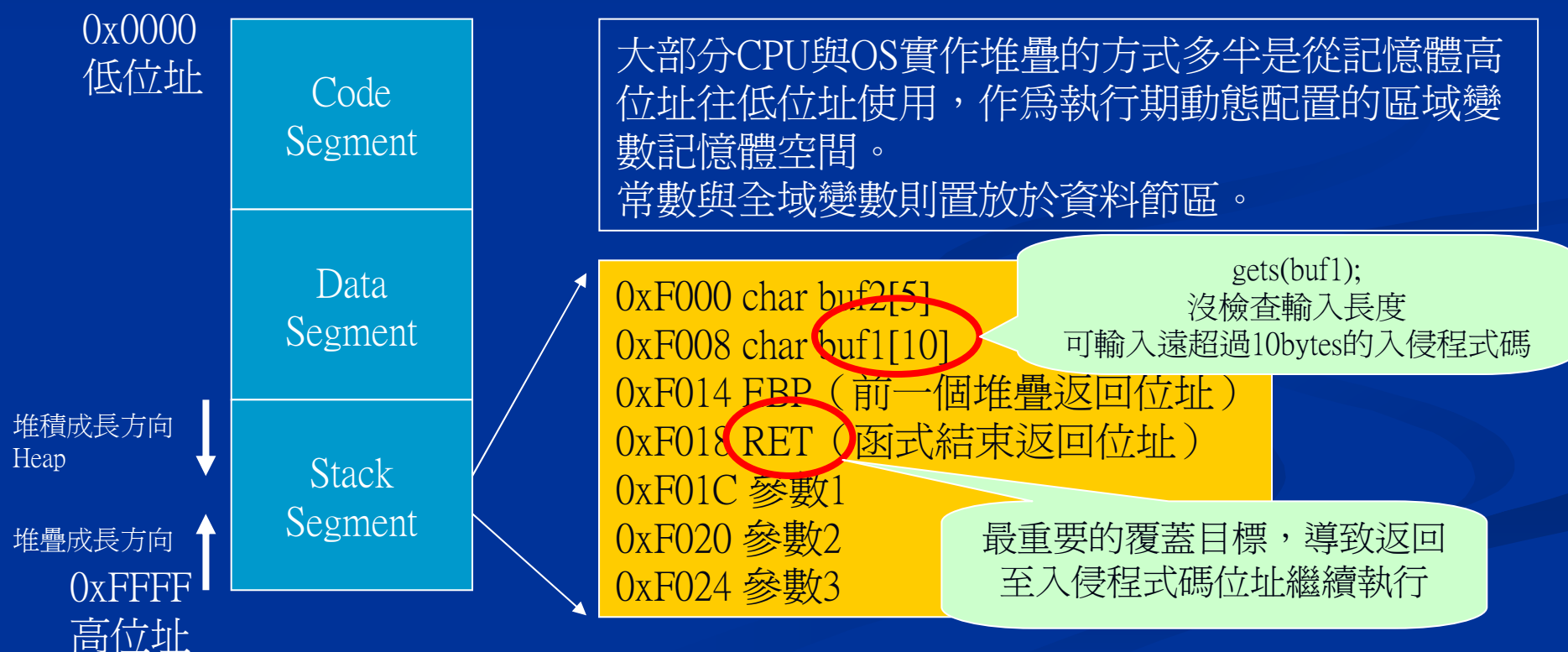
- 將線性位址分割成固定大小的Page，有4KB與4MB兩種Page Size
- CPU將Segment：Offset換算成線性位址，查對出此位址對應的Page，對應至實體位址中，如此頁並不在實體位址中，則發出page-fault的例外（Exception）中斷
- OS必須提供一個Exception handler來處理這個例外中斷，從Swap取出（或交換）所需要的頁至實體位址中
- 然後AP從中斷處繼續執行下去

■ 分頁結構

- 4KB Page：分Page Directory、Page Table、Page Offset 三層
- 4MB Page：分Page Directory、Page Offset 兩層

緩衝區溢出攻擊技術(1)

■ Runtime Process 結構分區與堆疊溢出原理



緩衝區溢出攻擊技術(2)

- 緩衝區溢出攻擊技術
 - 植入法
 - 利用Process現有的Function Code
 - 修改Function Point
 - 修改longjmp指標

UNIX 特點

- **Everything (including hardware) is a file**
 - 所有的事物（甚至硬體本身）都是一個個的檔案
- **Configuration data stored in text**
 - 以文字形式儲存組態資訊
- **Small, single-purpose program**
 - 程序盡量朝向小而單一的目標設計
- **Avoid captive user interfaces**
 - 盡量避免令人困惑的用戶介面
- **Ability to chain program together to perform complex tasks**
 - 能將多個程序連結起來，處理大而複雜的工作

UNIX 之分裂

■ UNIX System V Release v.s. BSD UNIX

- 1978年，為對抗AT&T壟斷UNIX的企圖，加大柏克萊分校從UNIX V6改出BSD版本，從此UNIX分成企業壟斷與學術開源兩支系長期對抗

■ BSD被AT&T終結，但免費開源作業系統的觀念已經開花結果

- BSD終於在1992被AT&T控告侵權，陷入長期官司，最後兩敗俱傷，AT&T最終將UNIX賣給Novell，而Novell放了BSD一馬，但386BSD的開發計畫已經被延誤將近十年，錯過PC起飛的黃金時代，其後陸續出現各家BSD分支，如FreeBSD、OpenBSD、NetBSD等等，始終堅守免費開源的理想。其後Novell又將UNIX賣給SCO。

■ 微軟視窗系統與Linux趁此空窗時期崛起

- 386BSD被AT&T阻礙期間，PC逐漸普及，微軟視窗在無對手情況下順風擴展獨大，Hackers們無力可施，將熱情轉注1991全新出現的Linux幼苗，1993就正式問世，並於短時間內成長茁壯

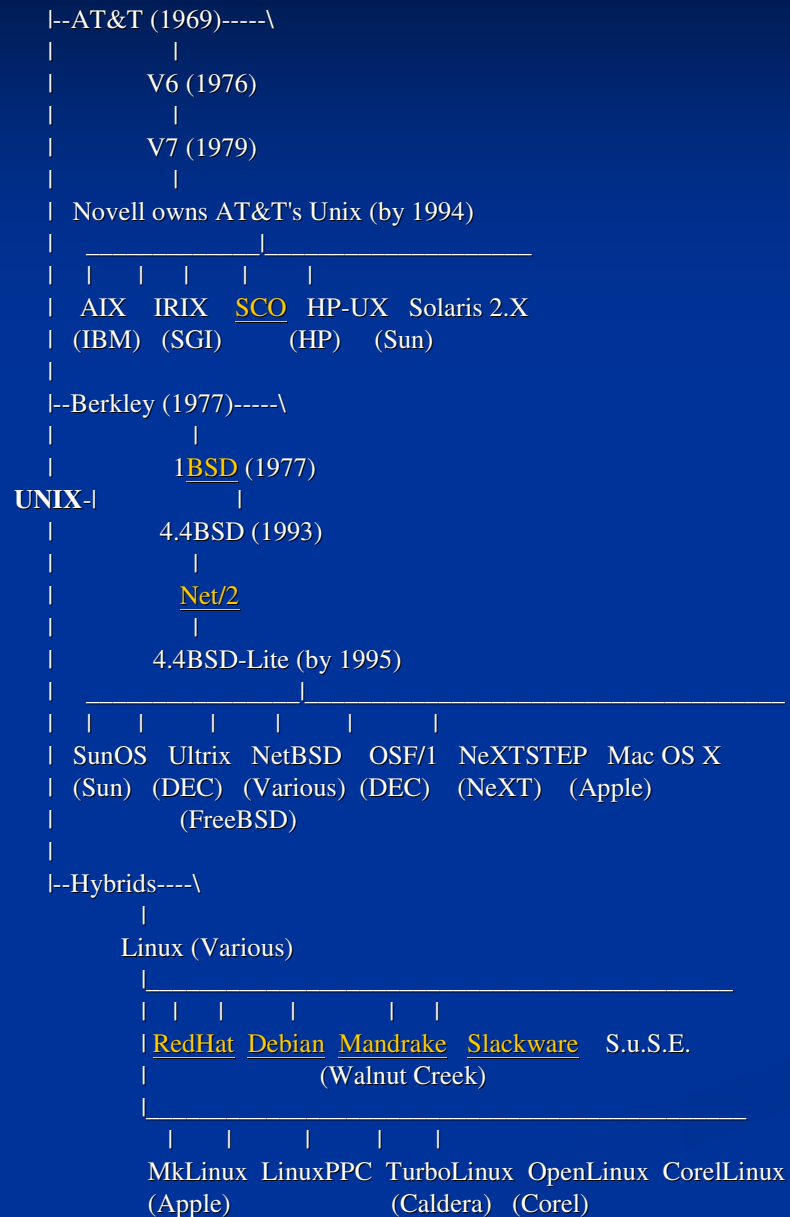
■ GNU與Linux之結合

- GNU打造了開源作業系統的所有套件，獨缺Kernel
- Linux具備核心卻欠缺周邊套件，雙方合作，迅速席捲世界

■ 企業與Linux之結合

- 為對抗巨大的微軟，SUN、IBM、SGI等企業都將多年累積的UNIX精華技術源碼轉移給Linux核心使用，使得Linux減少奮鬥三十年，使Kernel-2.4躍進成可與企業等級OS並駕齊驅的成熟系統。
- 微軟仿效AT&T重施故技唆使SCO出面控告Linux侵權（因IBM/SUN/SGI等公司無權將UNIX源碼轉送給Linux使用），促使Kernel-2.6又一次大躍進
- Linux在企業UNIX與開源BSD兩大極端陣營之間找到了中庸立足點，既保有免費開源的Kernel，又允許企業包裝套件營利，理想與現實兼顧調和的特色，同時獲得Hackers與企業的支持而快速成長

UNIX 簡易族譜



UNIX System V Release4 (SVR4)

- AT&T 於1989年推出的UNIX定義標準
 - 集成SVR3、4BSD、SunOS、XENIX的一些特性，還添加了一些新功能，如實時調度，Korn shell,以及對STREAMS子系統的改進
- 目前市面上主流的商用UNIX系統皆遵循此標準來實現
 - Sun Solaris、IBM AIX、SCO Unixware
- 定義了UFS（Unix File System）
 - Sun (SVR4)、BSD都使用此定義與名稱，但實作內容不盡相同
- Linux 也參照許多 SVR4 的概念，同時吸收BSD的概念組合而成

檔案系統

- UFS (Solaris 、 FreeBSD 、 Linux)
- Ext2 (Linux)
- Journal File System
 - NTFS (Windows NT 、 2000 、 XP)
 - Ext3 (Linux)
 - Reiserfs (Linux)
 - XFS (SGI IRIX 、 Linux)
 - JFS (IBM AIX 、 Linux)
 - UFS with Logging after Solaris7
 - Vxfs (Veritas)

UFS (Unix File System)

■ UFS

- SuperBlock：紀錄分區內重要資料位置，如i-node table等
- i-Node：等長度的Record組成的Table，舉例欄位如下
 - i-node的種類(file, directory, device等)
 - 這個節點參考的次數(目錄數)
 - 參考，產生，變化的時間
 - 權限、所有者的user id / group id
 - 本體的長度
 - 收集本體的data block的block號碼的固定長度的對應表的一些記錄。
- Data Block Table：將固定數量的Blocks組成連續塊的表

■ UFS2

- 調高分區最大限制
- 增強ACL功能

■ FFS (Berkeley Fast File System)

- FreeBSD針對UFS和UFS2改善效能的實作版本
- 具備更小分區能力，縮小Table size與領域
- 考量目錄及其下子目錄與檔案應在同領域，優化i-node與data之配置方法
- SuperBlock在分區內多藏幾處，避免無法復原的損毀
- 加入 Soft Update 機制，作為 Journal 的替代方案

File System 加速探討

- Buffer / Cache
 - 使用空閒的RAM作為儲存媒體的高速緩衝區及快取區
- Write-through (sync)
 - 資料立即寫入媒體，同時放一份在Cache
 - 因寫入慢會影響程式效率，但crash後復原度高
 - UFS的傳統作法
- Write-Back (async)
 - 僅放在Buffer，再伺機寫入媒體，Buffer可兼Cache
 - 程式效率高，但crash將損失慘重
 - Async UFS、Linux Ext2
- Update daemon
 - 30秒一次Auto Flush buffer
- bdflush daemon :
 - Linux 特有，由update啟動，一有空閒就flush部分buffer

Journal & Soft Update

■ Journal file system

- 使用特別區塊作為日誌存放區
- 所有對檔案系統 Metadata (i-node, 屬性等等) 的修改都先寫在日誌區後才進行真正的檔案內容操作
- 日誌空間為循環式覆蓋使用，若處理不及，則暫停新的metadata寫入，直至被清空一定空間。
- 因日誌實體位置集中且量少又循序，故採用Write-Through的效能亦相當高
- Crash後，fsck僅需檢核日誌內有記載的操作即可，不需檢核全系統所有資料，故服務恢復速度快
- 對服務不中斷要求甚嚴的企業來說，能快速回復服務的檔案系統是極為重要的

■ Soft Update

- 累積零碎的小IO成一塊，一併寫入，並在Data寫入前先確保Metadata已經被確實寫入
- 合併方法極為複雜，以致於雖增加安全性，但效能增加無預期之多

Journal File System (1)

■ Ext3

- 僅僅是在Ext2上添加了日誌功能，具有完全兼容Ext2的關鍵特性
- 最大的缺點是沒有現代文件系統所具有的能提高文件數據處理速度和解壓的高性能

■ XFS

- 原為SGI公司用在IRIX作業系統上的檔案系統，SGI為表示對Linux的支持，採用GPL方式開放原始碼，並協助移轉至Linux平台
- 目前世界上最擅長操作超大檔案的系統

■ JFS

- 原為IBM公司在AIX作業系統上的檔案系統，IBM為表示對Linux的支持，採用GPL方式開放原始碼，並協助移轉至Linux平台
- IBM超級電腦亦使用AIX，故JFS已歷經數十年的考驗
- 具有彈性擴充分區大小而不需重新格式化的特異功能

Journal File System (2)

■ Reiserfs

- Namesys在SuSE上開發的檔案系統
- 因SuSE原本就是Linux一支，是最早出現在Linux上的JFS，故其穩定性是最成熟的。
- 整個文件系統完全是從頭設計的
- 可管理16TB Partition、千百萬個檔案
- 支援>4GB的超大檔案
- 日誌(Journaling/logging)功能機制
- 高效的硬碟空間利用 (Tail Packing)
- 使用平衡樹(balanced tree)搜索，是一種非常高效的算法，在檔案定位上速度非常快

Journal File System (3)

■ NTFS

- Auto remap bad sector and marked
- 可加密檔案系統
- 權限與帳號相結合，安全控制可依使用者區分
- 可管理16TB Partition
- 可變動 default cluster size (default 4KB)
- 日誌(Journaling/logging)功能機制
- Quotas support

測試比較@Linux (1)

■ Partition capacity

- Initial (after filesystem creation) and residual (after removal of all files) partition capacity was computed as the ratio of number of available blocks by number of blocks on the partition
- XFS = 99.95%
- ReiserFS = 99.83%
- JFS = 99.82%
- Ext3 = 92.77

測試比較@Linux (2)

- File system creation, mounting and unmounting
 - Creation
 - Speed : XFS (0.7s) 、 JFS (1.3s) 、 Reiserfs (2.2s) 、 Ext3 (14.7s)
 - CPU : between 59% - ReiserFS and 74% - JFS
 - Mount
 - Speed : Ext3 (0.2s) 、 JFS (0.2s) 、 XFS (0.5s) 、 Reiserfs (2.3s)
 - CPU : between 6 and 9%
 - UnMount
 - Speed : Ext3 (0.2s) 、 JFS (0.2s) 、 XFS (0.2s) 、 Reiserfs (0.4s)
 - CPU : Reiserfs (14%) 、 JFS (27%) 、 Ext3 (37%) 、 XFS (45%)

測試比較@Linux (3)

- Operations on a large file (ISO image, 700MB)
 - The initial copy of the large file
 - Speed : XFS (34.8s) 、 JFS (35.1s) 、 Ext3 (38.2s) 、 ReiserFS (41.8s)
 - CPU : between 46%(JFS) and 51%
 - The recopy on the same disk
 - Speed : XFS (33.1s) 、 Ext3 (37.3s) 、 JFS (39.4s) 、 ReiserFS (43.9s)
 - CPU : between 38%(JFS) to 50%
 - The ISO removal
 - Speed : XFS 、 JFS (0.02s) 、 ReiserFS (1.5s) 、 Ext3 (2.5s)
 - CPU : XFS 、 JFS 、 Ext3 (10%) 、 ReiserFS (49%)
 - The number of minor page faults
 - ranging from 600 - XFS to 661 - ReiserFS

測試比較@Linux (4)

- Operations on a file tree (7500 files, 900 directories, 1.9GB)
 - The initial copy of the tree
 - Speed : Ext3 (158.3s) 、 XFS (166.1s) 、 ReiserFS (172.1s) 、 JFS (180.1s)
 - CPU : between 27 and 36%
 - The recopy on the same disk
 - Speed : Ext3 (120s) 、 XFS (135.2s) 、 ReiserFS (136.9s) 、 JFS (151s)
 - CPU : between 29% - JFS and 45% - ReiserFS
 - The Tree removal
 - Speed : ReiserFS (8.2s) 、 XFS (10.5s) 、 JFS (12.5s) 、 Ext3 (22s)
 - CPU : JFS 、 Ext3 (15%) 、 XFS (65%) 、 ReiserFS (86%)
 - The number of minor page faults
 - XFS 、 JFS 、 Ext3 (1400 to 1490) 、 Reiserfs (5843)

測試比較@Linux (5)

- Directory listing and file search into the previous file tree
 - The complete (recursive) directory listing of the tree
 - Speed : ReiserFS (1.4s) 、 XFS (1.8s) 、 Ext3 (2.5s) 、 JFS (3.1s)
 - CPU : Ext3 、 JFS (35%) 、 XFS (70%) 、 Reiserfs (71%)
 - The file search
 - Speed : ReiserFS (0.8s) 、 XFS (2.8s) 、 Ext3 (4.6s) 、 JFS (5s).
 - CPU : Ext3 、 JFS (6%) 、 XFS (10%) 、 Reiserfs (36%)
 - The number of minor page faults
 - XFS 、 JFS 、 Ext3 (704 to 712) 、 Reiserfs (1991)

檔案系統的選擇

- 檔案系統各有其優缺點、擅長與不擅長之處
- 考量項目：
 - 需求重點排序
 - 分區大小、檔案大小、檔案數量、速度、可靠度、成本.....等等
 - 作業系統穩定性與效能
 - 後續維護成本
 - Crash修復能力與恢復時間