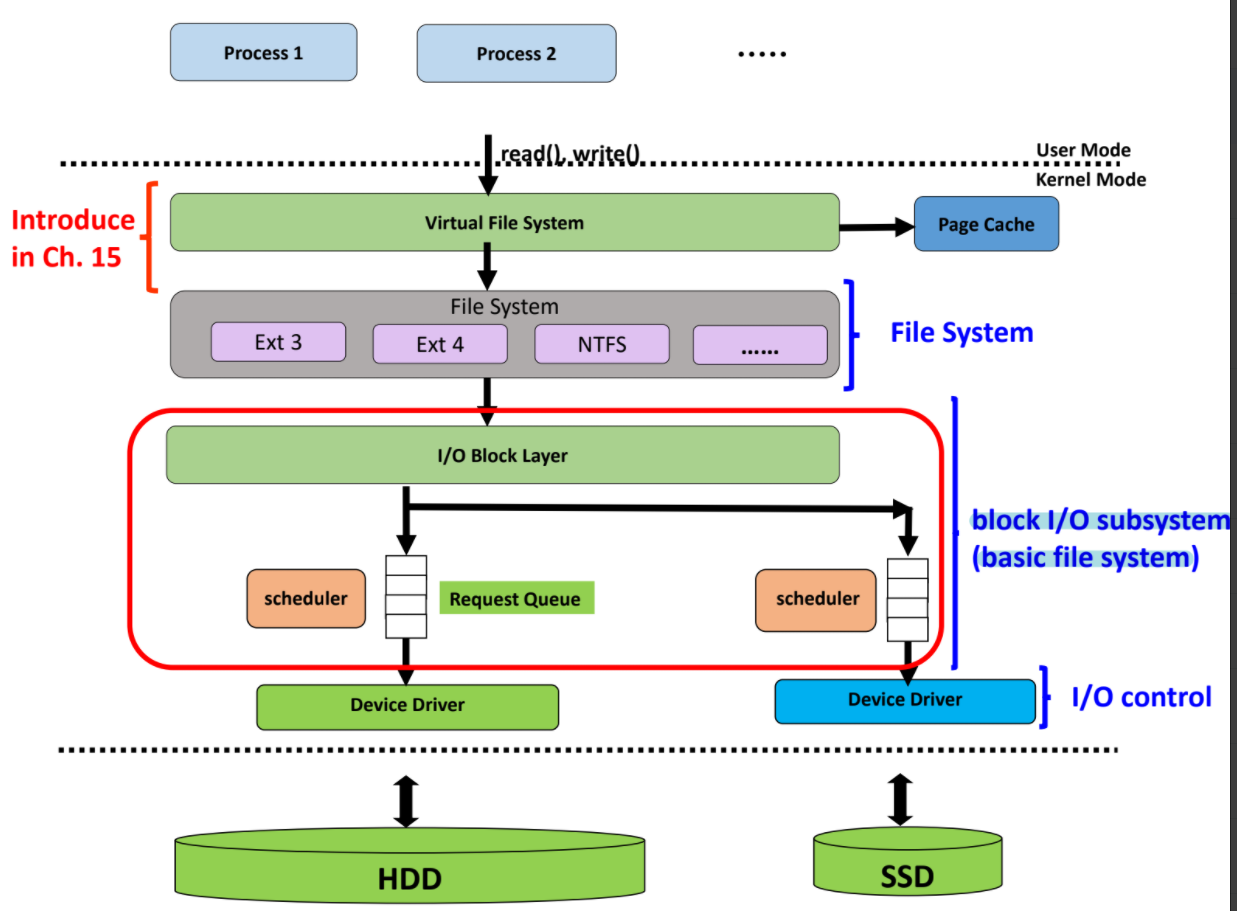
Chapter 14-3 File System Implementation

資工三穆冠蓁 4107056007

* File-System Structure



1. File System

(1)將檔案轉換成logical blocks(透過inode)

(2) Free space manager : 透過inode bitmap

(3) Blocks allocation

(4) Input : (file,offset) , Output : LBN

2. Basic file system : block I/O subsystem

(1) 管理buffer and caches

(2) 針對I/O requests 排程

(3) 將request送到正確的device driver

(4) Input,Output : LBN (有可能不同，因為可能有些block在cache中)

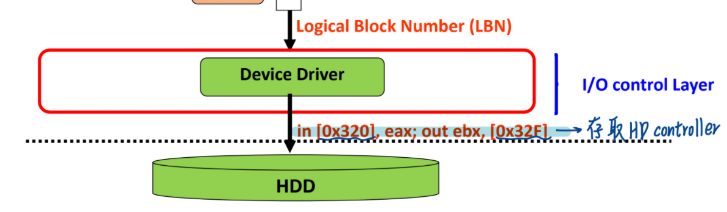
3. I/O control level :

(1) 由device drivers and interrupt handlers 組成

(2) device drivers像是一個translator

- Input : logical block numbers

- Output : 硬體指令，用來存取disk controller

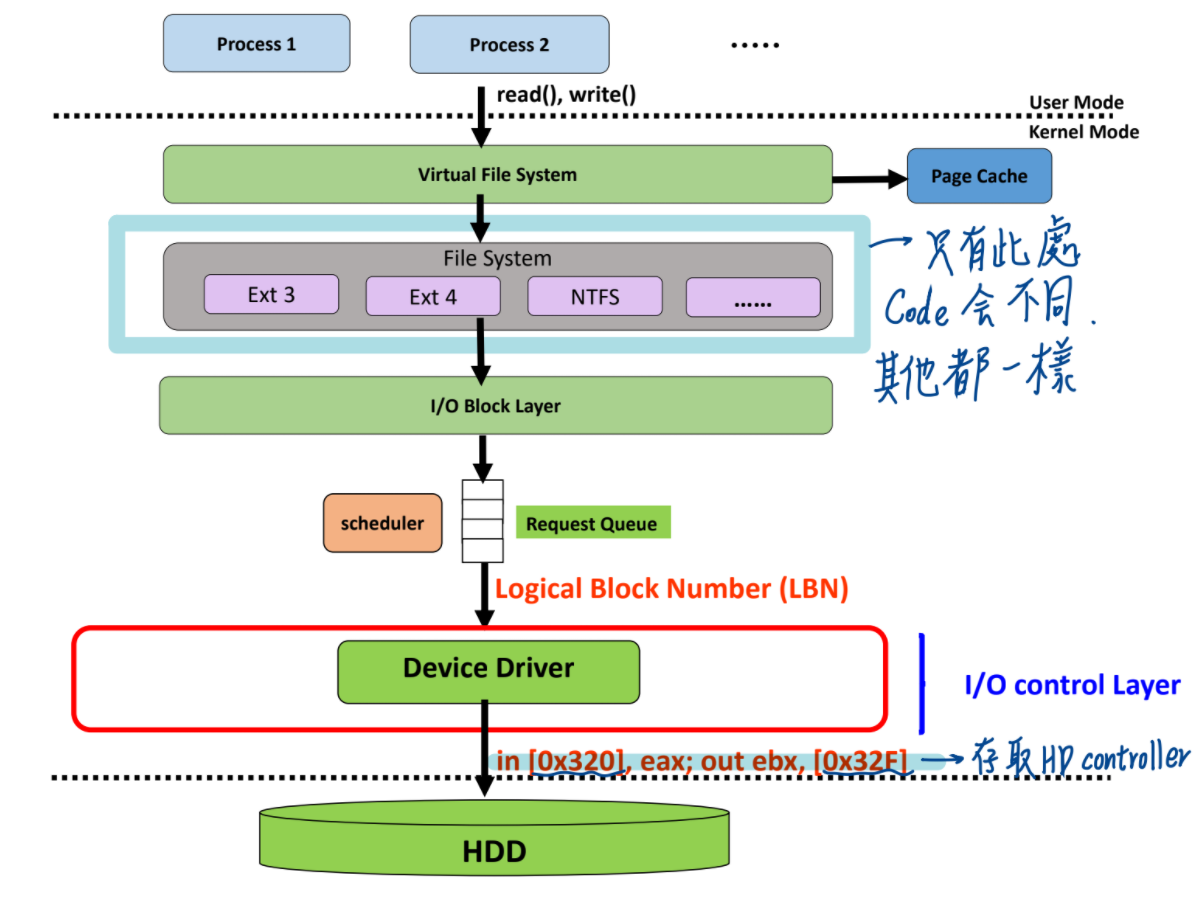


4. Layered File System

(1) 可以減少程式碼的重複

- 一個電腦系統中，可以有多個不同的檔案系統

- 除了檔案系統，I/O control 和 basic file system的code可以被不同的檔案系統共用。



* File-System Implementation

1. Implement file systems needs both :

(1) Data Structure

- On-disk data structures

- In-memory data structures

(2) Operations : Implement open(), read(), write(), close()

2. On-Disk Structures

(1) Boot control block (per volume): 包含初始化的bootstrap program 用來載入OS

- first block in that volume

- 若沒有os-> empty

- UFS : boot block

(2) Volume control block(per volume) : 記錄這個volume的相關資訊

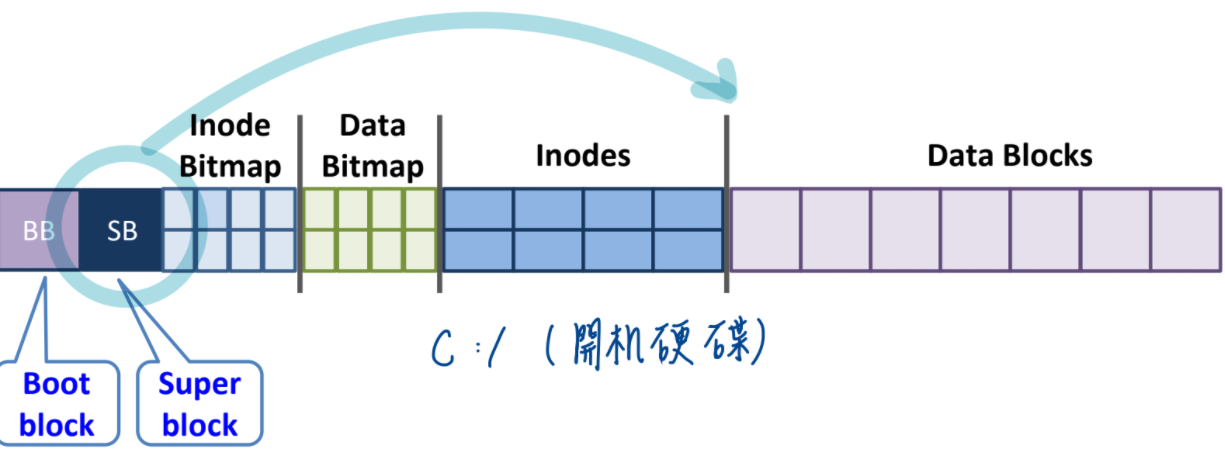
- UFS: called superblock

(3) File Control Block(per file):紀錄關於檔案的資訊

- UFS : inode

(4) Directory structure(per FS): used to organized files

- UFS : 一個目錄包含list of (filename,inode number)，一個目錄在UNIX中是一種特殊的檔案類型，並存放在data region



3. In-Memory Information

(1) 用來進行檔案處理系統的管理並增加效能(透過cache)

(2) in-memory directory-structure cache:

- 將最近使用到的目錄資訊cache在記憶體中

- 內容為 (file/subdirectory name, inode number)

(3) system-wide open-file table : 將inode cache 到記憶體(metadata可共享)

(4) per-process open-file table : metadata 不可共享

(5) Buffer/caches : 將data blockcache到記憶體

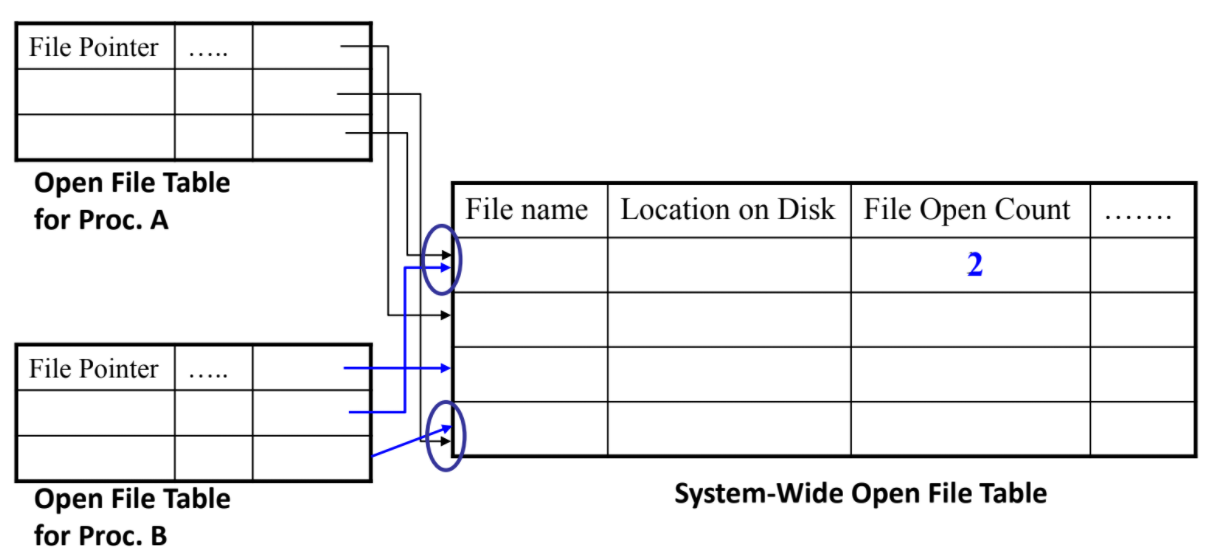
補: The Linux Kernel's I/O Caches

* Inode Cache (inodes)
* Directory Cache ( directory entries)
* Buffer Cache(data blocks)

4. Operations : Implement open(), read(), write(), close()

(1) Open :

- 先搜尋system-wide open-file table，若是已經正在被其他process使用，則per-process file table entry會被建立(指向system wide table)，system-wide entry 的 open count 也會加一



* 若是這個檔案沒有被其他process使用，搜尋目錄，並將FCB複製到system-wide open-file table，並在per-process open-file table建立一個entry，

(2) Close :

- 將**per-process table entry**移除，並將**system-wide table** 的count 減一

- 若是count == 0，就將metadata寫回硬碟，system-wide open-file table 的entry 也移除

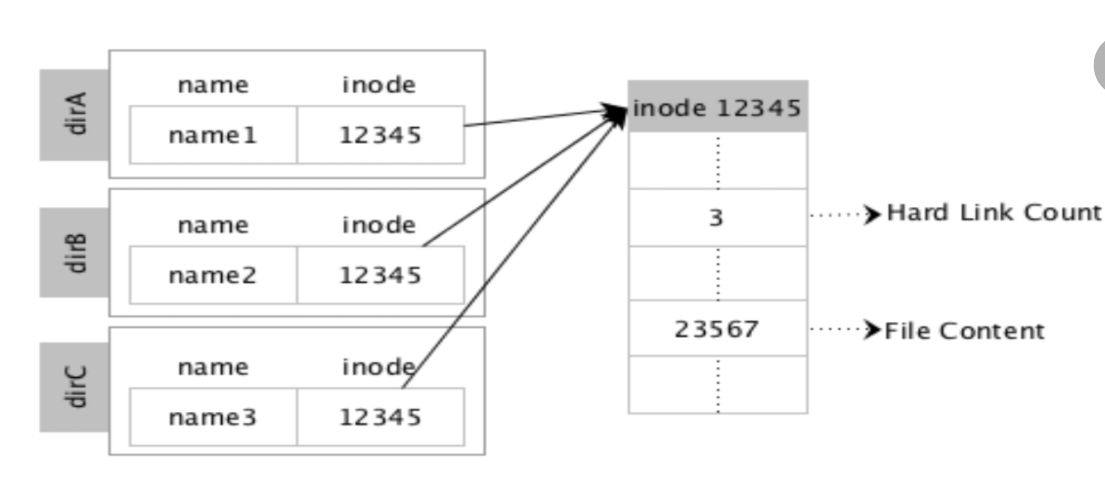
* Hard Links and Symbolic Links

1. Acyclic-Graph Directories : shared subdirectories and files

2. Hard links :

(1) ln (link()system call)產生一個hard links來指到舊的檔案

(2) link 在目錄裡增加一個目錄項目，指到舊檔案的inode



(3) Implement : ls -I 指出每個檔案的inode number

- 若是將其中一個檔案移除，另一個檔案仍可以access到檔案，因為unlink()會讓**reference count --**，但只有reference count =0時，才會刪除

3. Symbolic (Soft)Links :一個包含檔案名稱(檔案路徑)的text file

(1) Dangling reference : 若是刪除原始檔案，會讓指向他的檔案，找不到路徑

4. Summary :

(1) hard Link

- 無法跨FS(不同FS，hard link格式會不同)

- 不可link目錄(Cyclic)

- 跟原始檔案有相同的inode number、存取權限

(2) soft link :

- 可以跨FS(因為是存path name)

- 允許link目錄(因為作業系統會自動忽略)

- 跟原始檔案有不同的inode number、存取權限

* Directory Implementation

1. Directory Structure : 每個項目將檔案名稱轉換成FCB(inode)

2. 紀錄的方式 :

(1) Linear list :

(2) Hash

3. Linear list :

(1) 缺點 : 太浪費時間 O(n)

(2) 解決方法 :

- Cache : 快取常用的目錄資訊到記憶體

-允許binary search 的Sorted List : hard to insert

4. Hash Table : linear list with hash

(1) (File name) => hash function => a pointer to the linear list

(2) 減少search time

(3) 問題 : Collisions (hash 到同樣的位置)