

# Linux 系統程式研究

系統程式-期中報告

姓名：張靜云

學號：110710519 | 班級：資工二

1. 何謂 Linux?	3
Linux 簡介	3
Linux 的誕生	3
Linux 系統的基本構成	4
1. 內核	4
2. shell	4
3. 文件系統	5
4. 應用程式	5
啟動流程	5
記憶體管理	7
系統調用	9
Linux 程序的運行狀態	10
2. 文件 I/O	11
標準 C 的 I/O	11
FILE 結構體	11
標準 C 的 I/O 緩存類型	12
文件 I/O 的系統調用	13
文件操作方式	14
文件描述符	14
文件描述符與文件指針的轉換	15

內核數據結構.....	15
I/O 處理方式.....	17

# 1. 何謂 *Linux*?

## Linux 簡介

Linux 就是所謂的作業系統之一，1991 年由赫爾辛基大學的 Linus Torvalds 建立的作業系統。“Linux” 這個名字來自 Linux 內核。它是電腦上的軟體，使應用程式和用戶能夠訪問電腦上的設備以執行某些特定功能。

## Linux 的誕生

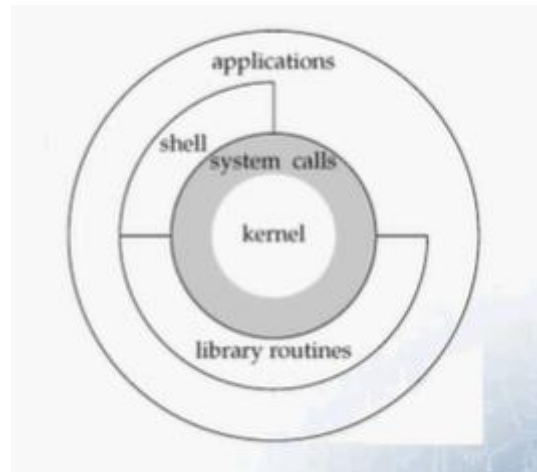
早期的電腦並非像現代如此普及，不僅只有軍方和一些重要單位才有之外，運作速度也是一個很大的問題，輸出也只能有像是列表機的方式做輸出，沒有螢幕，也沒有鍵盤。在鍵盤跟螢幕誕生之後，由於主機當相昂貴，所以大家都只能各自排隊使用電腦，而且電腦一次只能做一件事情，效率和現在也是有非常大的落差。爲了讓大型主機可以提供更多的資源，來達到同步多工的效果，許多單位和人材都投入了核心的開發，於是在 70 年代，利用 C 語言為基底的 Unix 誕生了，也歸功在 C 語言上，讓 Unix 具備了高移植性，只要你有程式碼，你就可以依照自己的主機硬體，把 Unix 移植到自己的主機上。但發行 Unix 在幾年後就讓 Unix 不再開源原始碼，所以在 80 年代，GNU 組織誕生了，最主要的目的是建立一個自由開放的 Unix 作業系統。

Linus Torvalds 就讀赫爾辛基大學電腦科學系期間，因為學校只有一台最新的 Unix 系統，所以等待使用 Unix 的時間相當耗時，於是讓 Torvalds 興起想要自己做一套 Unix 的想法，開始了他的開發之旅。

一開始只是想要嘗試做硬體效能以及 CPU 多工上的嘗試，但最後卻越改越多，也因此完成了 Linux 第一版。在開發過程中，他參考了許多 GNU 計劃裡提供的像是 bash 工作環境軟體以及 gcc 編譯器等自由軟體，但這些畢竟都是他一個人的產物，也深怕會有甚麼 Bug 以及可以優化的建議，於是，他選擇把程式放在一個名為 Linux 的 FTP 上公開，並且歡迎大家下載，修改這個核新程式，因此 Linux 就此誕生。

# Linux 系統的基本構成

- 內核
- shell
- 文件系統
- 應用程式



## 1. 內核

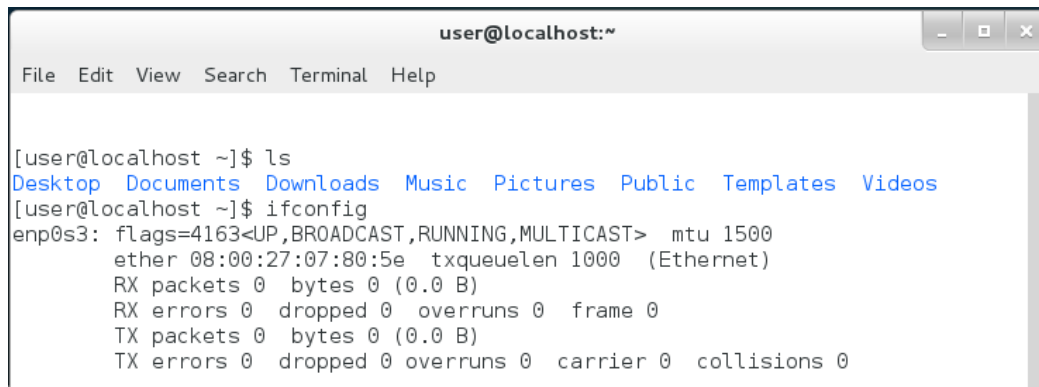
內核是作業系統的核心，它幫我們管理程序、設備的驅動程式(ex:鍵盤、滑鼠、LCD、網卡)、文件系統、網路系統等等，負責進行和硬體之間的通訊，比如：USB、掃描器。內核本身也是一段程式，用 C 語言編寫，它決定了系統的穩定性和可靠性。

```
user@localhost:~  
File Edit View Search Terminal Help  
[user@localhost ~]$ ls /usr/src  
debug kernels  
[user@localhost ~]$ ls /boot/vmlinuz*  
/boot/vmlinuz-0-rescue-330857d4454ad34b8fd0d77756849592  
/boot/vmlinuz-3.10.0-123.el7.x86_64  
[user@localhost ~]$
```

我的虛擬機畫面-內核放置目錄位置及壓縮文件

## 2. shell

提供用戶介面的軟體，用戶可以在 shell 上輸入命令，執行的命令可以在 shell 上顯示出來。輸入的命令會交給內核去執行，執行之後將結果交給 shell 顯示出來。

A terminal window titled 'user@localhost:~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and outputs:

```
[user@localhost ~]$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
[user@localhost ~]$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 08:00:27:07:80:5e  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

提供用戶可輸入的介面並顯示結果

### 3. 文件系統

一個操作系統當中必須有一個文件系統，在 linux 當中，文件系統根從「/」開始，下面有很多子目錄。實際上在 linux 當中，可以在安裝的時候對它進行分區，只不過不像 windows 當中有分區的盤符號。文件系統有七種類型：ext2、ext3、ext4、XFS、vfat、ReiserFS、btrfs。

### 4. 應用程式

在 linux 可以安裝很多應用程式，在 centOS 中，安裝應用程式的指令為 yum install 套件名稱；在 ubuntu 中的指令為 apt-get install 套件名稱。

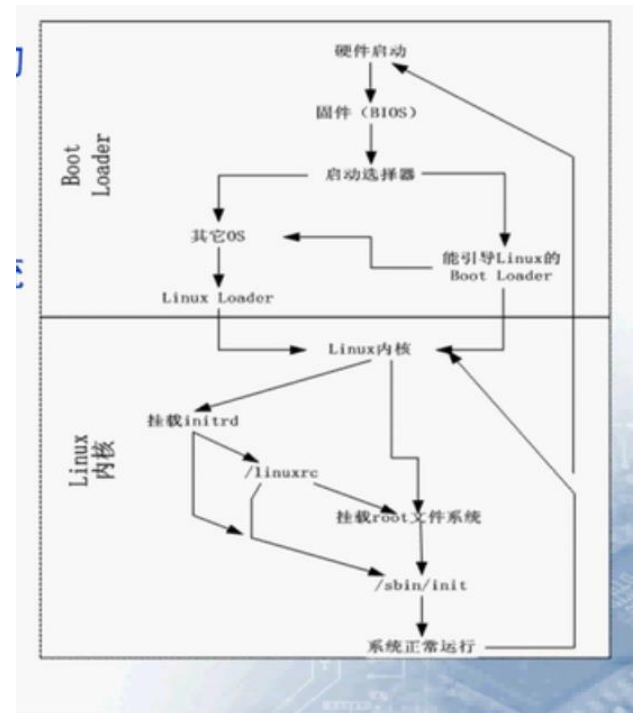
## 啟動流程

**Boot-loader(grub):**根的裝載器，負責引導整個整個作業系統的啟動，在內核運行之前的一段小程序，用彙編語言編寫。初始化硬體設備、建立記憶體空間的映射圖，將系統的軟硬體環境帶到一個合適的狀態，為作業系統內核準備好正確的環境。

## BIOS:

本身是燒錄在 BIOS 芯片裡，本身用彙編語言編寫，主要負責作業系統相關的啟動。包括：

1. 外圍電路的初始化。
2. 加載作業系統的內核。系統才能真正啟動。
3. 加載最小文件系統(initrd)。它是一個臨時性的文件系統，主要是在啟動之前被 linux 的內核調用，在根文件系統裝載之前，做初始化的準備工作。
4. 加載硬碟上的根文件系統。從根目錄開始。
5. 啟動程序。內核啟動最重要的一號程序 init，也就是系統啟動的第一個程序，執行 /etc/init.d 目錄中所有的腳本文件，初始化



啟動流程圖

0、1、2，啟動用戶登錄程序，然後系統就會正常運行。

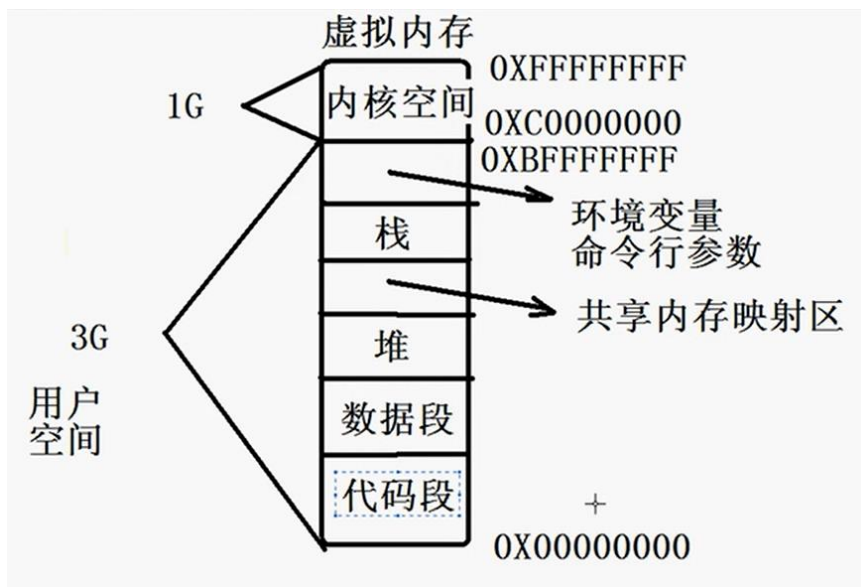
```
user@localhost:~  
File Edit View Search Terminal Help  
[user@localhost ~]$ ls -l /boot/vmlinuz*  
-rwxr-xr-x. 1 root root 4902416 Apr  8 21:16 /boot/vmlinuz-0-rescue-330857d4454a  
d34b8fd0d77756849592  
-rwxr-xr-x. 1 root root 4902416 Jun 13 2014 /boot/vmlinuz-3.10.0-123.el7.x86_64  
[user@localhost ~]$ clear  
[user@localhost ~]$ ls /  
bin  dev  home  lib64  media  opt  root  sbin  sys  usr  
boot  etc  lib  lost+found  mnt  proc  run  srv  tmp  var  
[user@localhost ~]$ ls -l /boot/initrd*  
-rw-r--r--. 1 root root 867240 Jun 19 2014 /boot/initrd-plymouth.img  
[user@localhost ~]$ ls /etc/init.d  
functions  iprinit  livesys  netconsole  README  
iprdump  iprupdate  livesys-late  network  rhnsd  
[user@localhost ~]$ ls /sbin/getty  
ls: cannot access /sbin/getty: No such file or directory  
[user@localhost ~]$ ls /sbin  
accessdb  mkfs.cramfs  
accton  mkfs.ext2  
addgnupghome  mkfs.ext3  
addpart  mkfs.ext4  
adduser  mkfs.fat  
agetty  mkfs.minix  
alternatives  mkfs.msdos
```

最小文件系統、根文件系統、init 目錄

# 記憶體管理

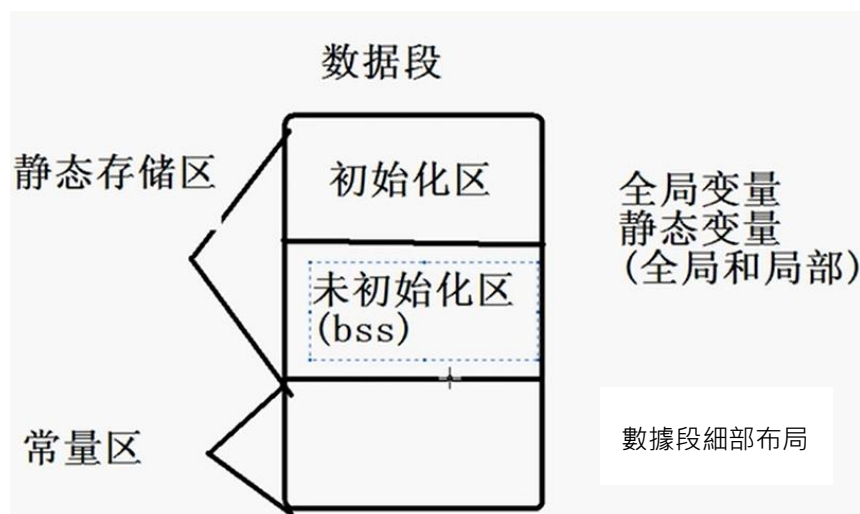
Linux 作業系統啟動之後，後面會啟動很多程序，會佔據記憶體，這個記憶體實際上是塊虛擬記憶體，我們在這塊記憶體上操作。它的大小跟當前電腦 CPU 處理器的位數有關係，現在的電腦基本上都是 32 位的 CPU，它決定了整個虛擬記憶體的空間，也就是 2 的 32 次方，大概就是 4 個 G 的空間。

## 1. 記憶體隔離：防止互相干涉數據和存儲空間。



記憶體空間佈局

棧=堆疊  
堆=堆積  
內存=記憶體

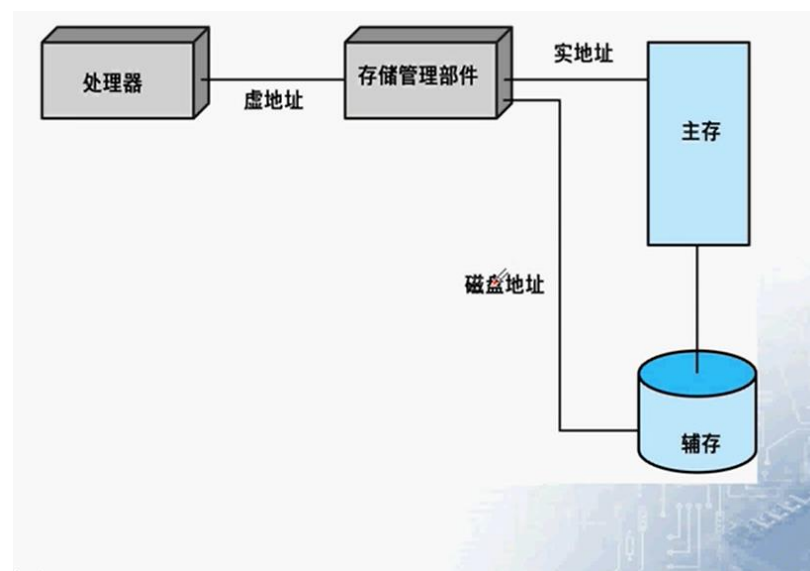
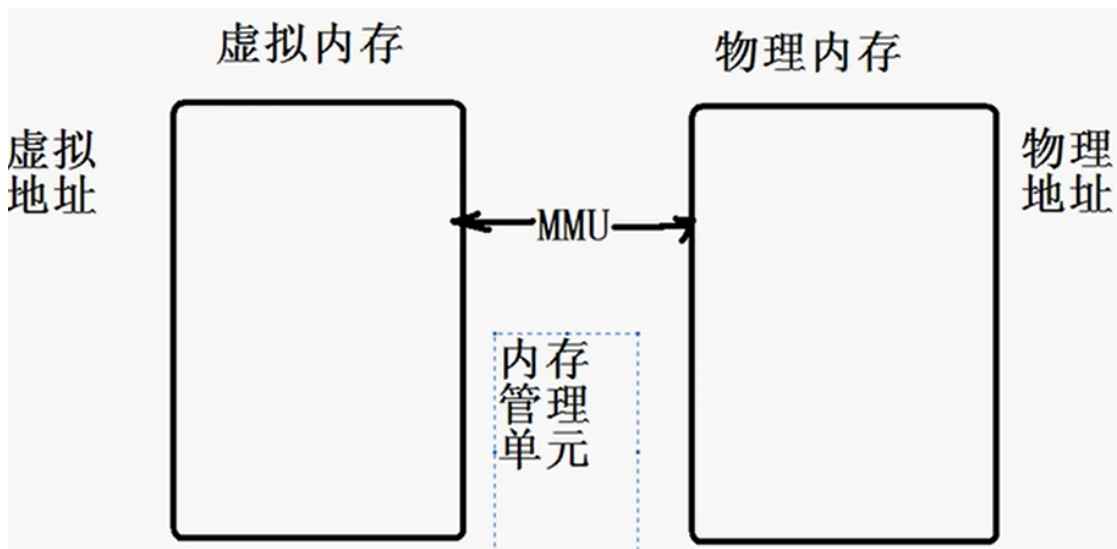


數據段細部布局



最終的數據交換還是存放到物理記憶體，在虛擬記憶體中所用到的地址都是虛擬地址，真正在存放的數據最終會存放在物理記憶體。

記憶體管理單元(MMU)：負責將虛擬地址轉換成對應的物理地址，然後根據地址將虛擬記憶體的數據存放到物理記憶體，同時也可以將物理記憶體的數據讀取到虛擬記憶體。

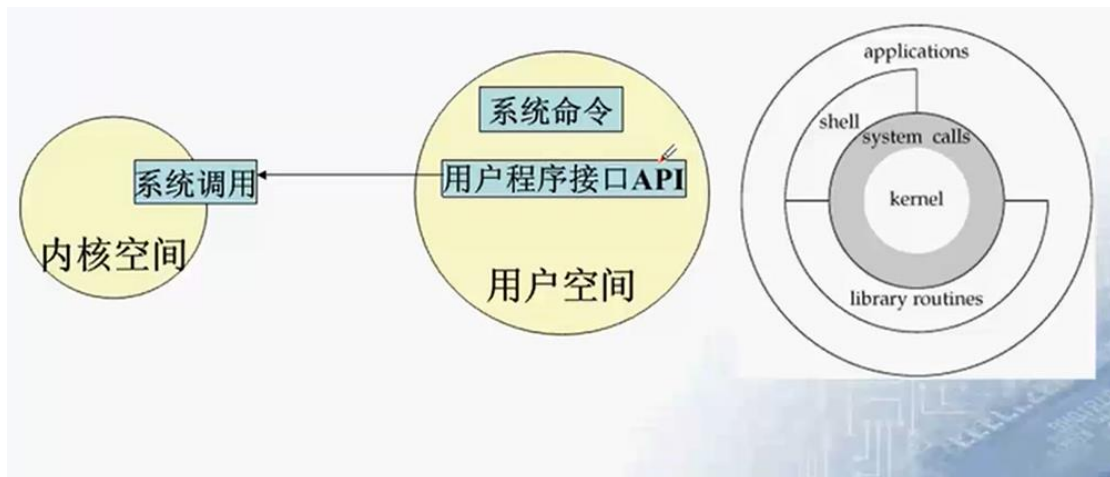


虛擬記憶體與物理記憶體的相互交付

2. **自動分配和管理**：程序人員可以通過調入相應的函數項系統申請一快在虛擬空間當中這塊記憶體可以動態分配，用完了以後可以釋放掉。
3. **支援模塊化的程式設計**：能夠定義程式模組，動態創建模組，銷毀、改變模組大小。
4. **保護和訪問控制**：如果程式能夠直接訪問物理記憶體，那記憶體就可能會出問題，所以通過架了虛擬記憶體，用戶直接和虛擬記憶體進行交付，然後用 MMU 將虛擬記憶體數據交換到物理記憶體，起到保護和防治的作用。
5. **長期存儲**：關機後長時間保存信息。

## 系統調用

所謂系統調用是指，作業系統提供給用戶程序的一組特殊接口，用戶程序可以透過者組特殊接口來獲得作業系統內核提供的特殊服務。這些接口實際上就是一些內核所提供的函數，這些函數就稱為系統調用。



可以使用系統調用的用戶：

1. **應用程式**：要獲得內核提供的服務，必須通過系統調用的接口。

2. **Shell**: 比如說，在系統上面輸入一條命令 `ls`，`ls` 就會列出當前目錄和指定目錄下面的內容，`ls` 本身就是用 C 語言寫的程式，實際上它調用內部一系列的跟文件操作相關的系統調用。
3. **標準 C 函數庫**：這些函數庫底層都是去用了系統調用，來獲得內核提供的服務。

*當用戶程序通過系統調用，當前的程序狀態會發生變化。*

## Linux 程序的運行狀態

**內核態**：程序運行在內核空間。

**運行態**：程序運行在用戶空間。

如果當前用戶的啟動程序在用戶空間當中，它所處的狀態就是運行態，它是開機時的默認狀態。當用戶空間的程序去調用了系統調用，來獲得內核服務，它的狀態就會切換成內核態，當調用完畢，會返回到用戶空間，變回運行態。

如果每個作業系統都有自己的系統調用，那麼工程師就必須花很多時間學習不同系統的系統調用，這是一件困難的事情，因此 IEEE 對這些系統調用制定了一個標準，這些系統調用接口遵循了 POSIX 標準，只要學會一套，幾乎就可以在每個作業系統上使用。

## 2. 文件 I/O

### 標準 C 的 I/O

一些 I/O 函數

```
char *fgets(char *s, int size, FILE *stream);
int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
...
```

fgets: 針對整行的輸入輸出

printf: 標準的輸入輸出

fprintf: 針對文件的操作

fread/fwrite: 針對二進制文件的讀寫操作

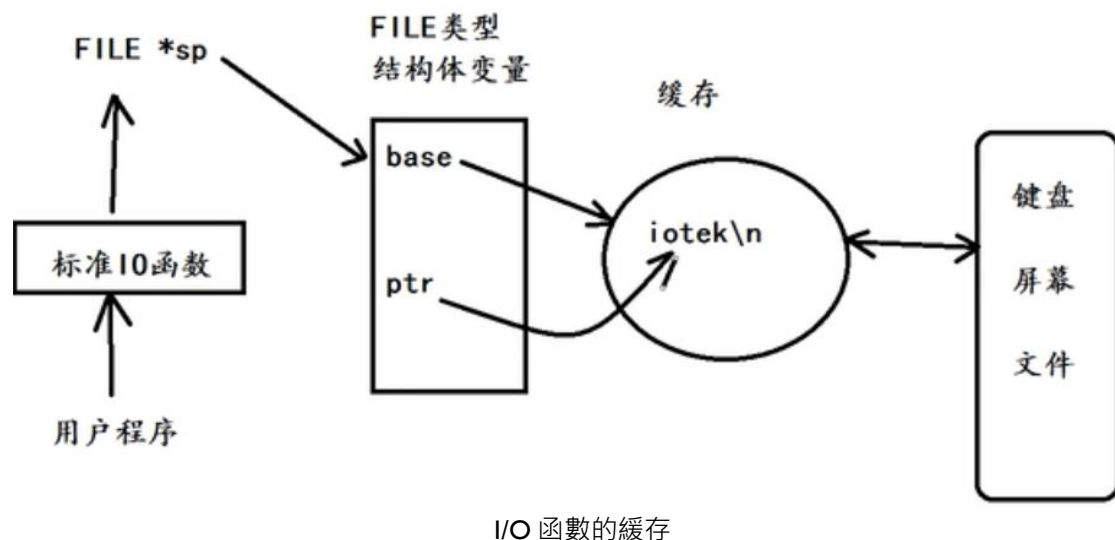
### FILE 結構體

```
typedef struct iobuf{
    int cnt;          /* 剩余的字节数 */
    char *ptr;        /* 下一个字符的位置 */
    char *base;       /* 缓冲区的位置 */
    int flag;         /* 文件访问模式 */
    int fd;           /* 文件描述符 */
} FILE;
```

cnt: 在 I/O 操作中，剩餘要讀寫的字節數

ptr: 字符形指針，存在結構體變量當中，指向下一個字符的位置

base: 字符形指針，存在結構體變量當中，指向緩衝區位置。在 C 當中，所有的函數，不管是針對輸入，鍵盤操作等等，這些所有的函數都是有緩存功能的。比如說：從標準輸入去讀取一行數據，實際上在內部不會把這些數據直接輸出到螢幕上，中間會經過一個緩存，通過緩存存放內容，再把數據寫入文件。



flag: 文件訪問模式，比如說：文件是讀寫的或者只讀等等。

fd: 文件描述符，所有通過 I/O 系統調用去操作文件都是基於文件描述符進行操作。

## 標準 C 的 I/O 緩存類型

**全緩存：**要求填滿整個緩存區之後才把內容一次性寫到文件或 I/O 設備。

若是緩存區沒有滿，可以：

1. 強制調用一個函數 `fflush`，強制清緩存。
2. 在結束之前程式會自動清緩存。

這種全緩存主要是針對磁碟文件進行操作。

**行緩存：**涉及到一個終端機操作(例如：標準輸入 scanf 和標準輸出 printf)，使用行緩存。當行緩存滿的時候會自動輸出，每當碰到換行符也會自動輸出。

**無緩存：**主要針對標準錯誤流 stderr，出錯訊息在螢幕上不通過緩存，直接通過程式內部輸出，使得錯誤信息能夠盡快顯示出來。

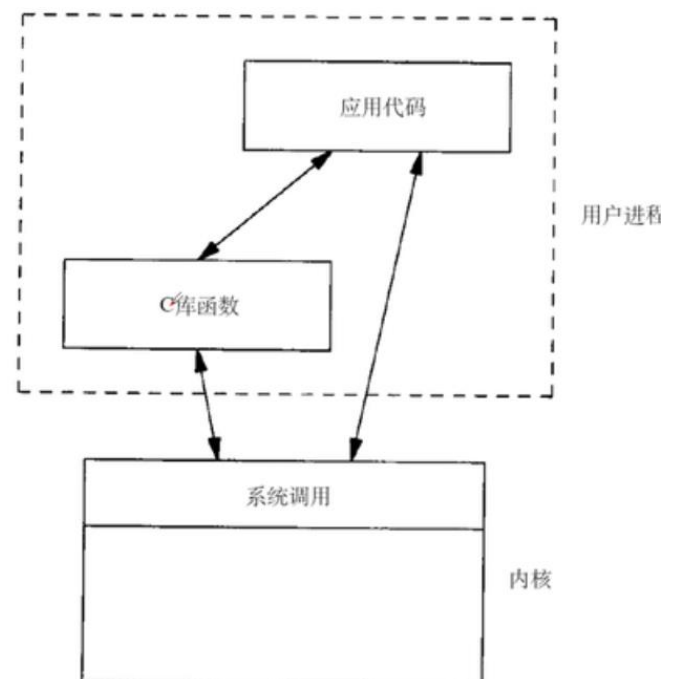
## 文件 I/O 的系統調用

open()	creat()	close()	read()	write()	lseek()
打開文件	創建文件	關閉文件	讀取文件	寫入文件	文件定位

※這些不帶緩存的函數都是內核提供的系統調用

標準 C 的 I/O 函數，它內部底層實際上還是去調用內核所提供的系統調用。

例如：標準 C 當中 I/O 函數有一個叫 fopen()，在內核所提供的 I/O 系統調用當中有一個 open()，實際上在 fopen() 內部本身又去調用 open() 這個函數。



系統調用與 C 庫

## 文件操作方式

**標準庫函數：**遵循 ISO 標準，基於流的 I/O，對文件指針(FILE 結構體)進行操作。

**系統調用：**兼容 POSIX 標準，基於文件描述符的 I/O，對文件描述符進行操作。

## 文件描述符

對於內核而言，所有打開文件都由文件描述符引用，文件描述符是一個非負整數，當打開一個現存文件或創建一個新文件時，內核會向用戶程序返回一個文件描述圖。當讀寫一個文件時，用 open 或 creat 返回的文件描述符標示該文件，將其作為參數傳遞給 read 或 write。

在 POSIX 應用程式中，整數 0、1、2，被替換成符號常數 STDIN\_FILENO、STDOUT\_FILENO、STDERR\_FILENO，這些常數都定義在<unistd.h>中。

文件描述符的範圍是 0~OPEN\_MAX。早期的 unix 版本上限為 19，最多每個程式可以同時操作 20 個文件。現在很多系統增加至 63 個，LINUX 則是 1024。

## 文件描述符與文件指針的轉換

### 標準文件指針

stdin	0
stdout	1
stderr	2

### fdopen()

FILE\*fdopen(int fd, const\_char\*mode);

文件描述符→文件指針(fd→FILE\*)

### fileno()

int feline(FILE\*stream);

文件指針→文件描述符(FILE\*→fd)

## 內核數據結構

※一個打開的文件在內核中使用三種數據結構表示

### 文件描述符表

文件描述符標誌

文件表項指針

### 文件表項

文件狀態標誌

當前文件偏移量



i 節點表項指針

引用計數器

## i 節點

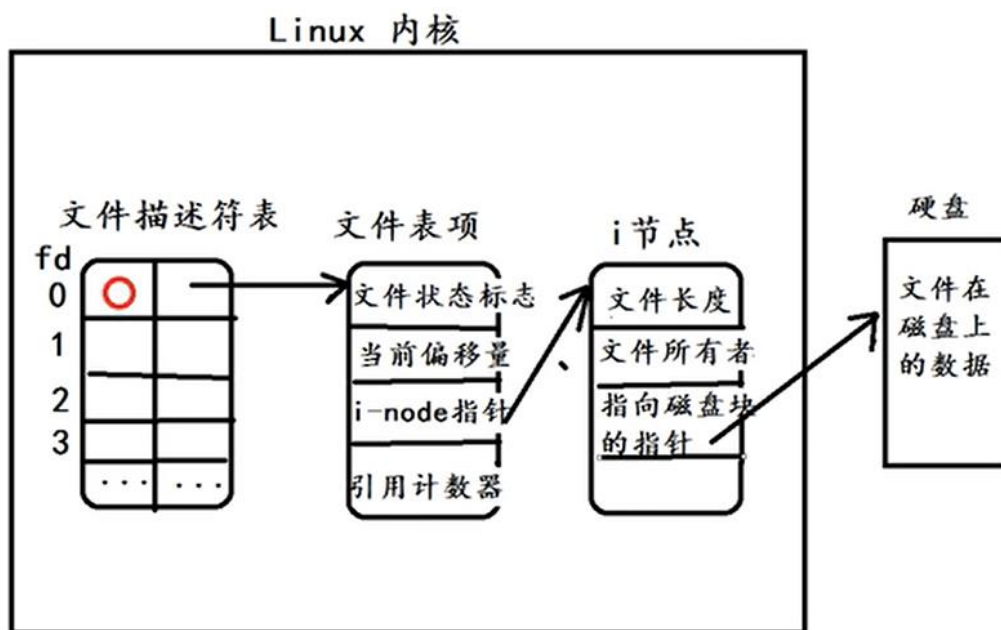
文件類型和對該文件的操作函數指針

當前文件長度

文件所有者

文件所在設備、文件訪問權限

指向文件數據在磁碟上所在位置的指針



三種數據結構

## I/O 處理方式

### I/O 處理的五種模型：

**阻塞 I/O 模型：**若所適用的 I/O 函數沒有完成相關的功能就會使程序掛起，直到相關數據到達才會返回。如：終端、網絡設備的訪問。

**非阻塞模型：**當請求 I/O 操作不能完成時，則不讓程式休眠，而且返回一個錯誤。如：open, read, write 訪問。

**I/O 多路轉接模型：**如果請求 I/O 操作阻塞，且它不是真正阻塞 I/O，而且讓其中一個函數等待，在這期間，I/O 還能進行其他操作。如：select 函數。

**信號驅動 I/O 模型：**在這種模型下，通過安裝一個信號處理程式，系統可以自動捕獲特定信號的到來，從而啟動 I/O。

**異步 I/O 模型：**在這種模型下，當一個描述符已經準備好，可以啟動 I/O 時，程式會通知內核，由內核進行後續處理，這種用法現在較少。

參考資料：Linux 系統程序設計-<https://space.bilibili.com/337344256>

Linux 是什麼？-<https://progressbar.tw/posts/113>

謝謝觀看