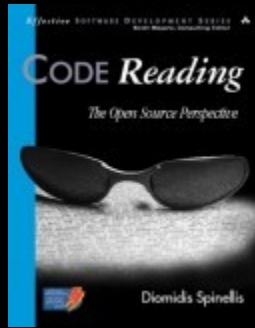




以工程觀點重新檢視 UNIX 與 C 語言

Jim Huang(黃敬群) “jserv”
blog: <http://blog.linux.org.tw/jserv/>
Oct 22, 2012 / 中正大學

爲什麼要重新檢視？



- "I wonder how many great novelists have never read someone else's work, ... painters ... another's brush strokes, ... skilled surgeons ... learned by looking over a colleague's shoulder, 767 captains ... in the copilot's seat ..."

(沒有研讀過其他作家作品的偉大作家，沒有研究過其他畫家筆法的偉大畫家，沒有盜取過並肩作戰的同事的技術的高明外科醫生，沒有在副駕駛的位置積累實際經驗的波音 767 的機長，在現實生活中真的會存在這樣的人嗎？)

Dave Thomas 在 《 Code Reading- The Open Source Perspective 》 的序文
<http://www.spinellis.gr/codereading/foreword.html>

```
#!/usr/bin/perl
```

Perl

這真的能執行，
Perl 太神奇了

```
+ $I=sub{+s+^+
$ "x$_[1]+gem;$ /x$_#
[0].$_.$ /};$W=sub{$~!q~
~.pop();system($^O=~Win?Cls:#
'clear'),print,select$Z,$Z,$Z,!
"||$~for@_};$H=sub{+join$ /,map($_#
x$_[0],pop=~m-.+~g),!_};$_=!Mima,s--
"@{['=9+)w'^RINGS]}\%;local@{[Saturn^#
wNXIBP]}~-see;s-^#!...?$/ (?="$"+;)--is
y-;-'-;s-\w--gi;$S=$_;#--Beautiful]
@S=m-.+~g;$N=1+.6-!th_, $--=-82-$---
$_="$x-(y---c-$-)for@S;$R=sub{$i#
=0;join$ /,map{$j=$%;join!_,grep#
!($j++%$_[$%]),m-.~g}grep!($i#
++%$_[0]),@S);$L=join!_,map#
~~reverse.$ /,@S;@R=(&$I(q-
$_=^q-q-),&$I(20,41-!q-
I->(15,31,$_=&$R(4-!q-
;;",28,$_=&$R(3)),&${
;;;;",$_=&$R->(2)),q-
;;;; " &&$H)($_,&${
++++ " +2..2*~~2
++++ &${m--
++++ b-
++++ -],!1!~~1);&$W($S.!q-
++++ -,$L,0.16)for$%..5+!q-
Cassini-;&{$W||q-
-})(@Y,1.6)
```

C

Perl 是神人在用的語言，凡人還是乖乖寫 C 程式吧

This is C!

```
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("Hello, world!\n");
    return 0;
}
```

爲什麼談 C 語言？

- C 不只是程式語言
 - 貼近硬體的軟體設計方法
 - 徹底解決系統問題的思維
 - *Beautiful Code*
- 不只學語法，更要深入欣賞
 - 《文心雕龍》

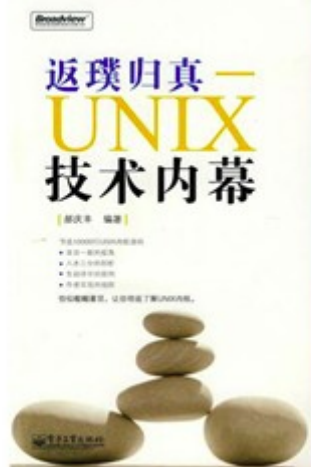
UX

「起初他們忽視你，而後嘲笑
你，接著打壓你，最後就是你的
勝利之日」

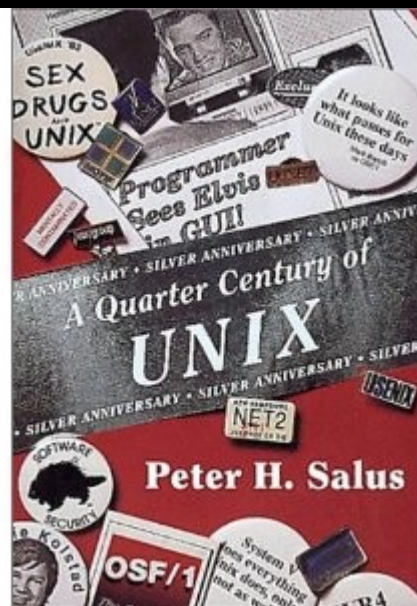
—— 甘地

背景知識

- 《返璞歸真 -UNIX 技術內幕》
- 《黑客列傳：電腦革命俠客誌》
- 《A Quarter Century of UNIX》
- 《電的旅程》
- 《浪潮之巔》



这不是Salus的'AIX'、'HP-UNIX'，但这是它们的基石



承先啟後的四個世代

- 1st system (CTSS)

- terrified of failure
- simplified to bare bones
- successful beyond its intended life-span

- 2nd system (Multics)

- hugely ambitious
- usually conceived by academics
- many good ideas
- a little ahead of its time
- doomed to fail

- 3rd system (Unix)

- pick and choose essence
- usually made by good hackers
- emphasize elegance and utility over performance and generality
- become widely adopted

- 4th systems (BSD)

- maturation

CTSS (Compatible Time-Sharing System; 1961-1973)

- MIT 推動 MAC 計畫 (Mathematics and Computation) , 在 IBM 7090 主機上開發
- 可支援 32 使用者同時使用
- 實現了互動使令操作與除錯功能
- 硬體 :
 - 0.35 MIPS; USD \$3.5 million (IBM 7094)
 - 32 KB Memory made up of 36-bit words
 - Monitor uses 5KB words, leaving 27KB for users
 - User memory swapped between memory and fast drum
- Multi-level feedback queue scheduling

Source: <http://www.multicians.org/thvv/7094.html>

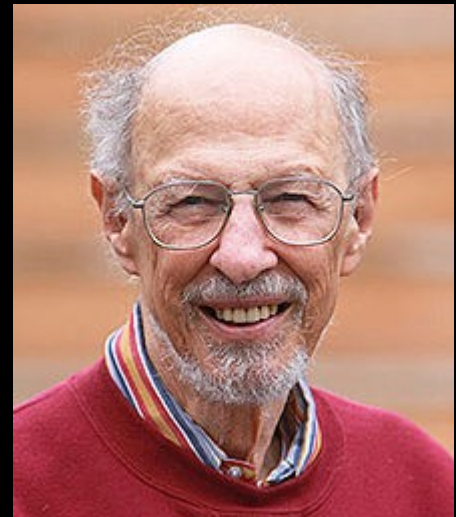


CTSS 的軟體特色

- 自批次系統移轉到多人分時
- 線上工作環境並提供線上儲存機制 (雲端概念？)
- 互動的 debugging, shell, editor
- 多個使用者間沒有保護機制

1990 年 Turing Award 得主： Fernando Corbato

- Fernando Jose Corbato (July 1, 1926–)
- 1990 Turing Award Citation:
“For his pioneering work organizing the concepts and leading the development of the general-purpose, large-scale, time-sharing and resource-sharing computer systems, CTSS and Multics.”



Multics (1964-1969-1985)

- 最初作為 CTSS 系統的延伸
- 由 MIT, GE (正在發展大型主機), Bell Labs (不能販售電腦卻人才濟濟的單位, 負責承包軟體) 等單位合作開發
- 願景:

“As larger systems become available, they will be connected by telephone wires to terminals in offices and homes throughout the city. The operating system would be a time-sharing system running continuously with a vast file system, just like electricity and water services”

Multics

- 最初在 GE645 工作站開發
- 提出一套通用的 framework，採用 PL/1 高階程式語言開發
- 重新設計每項系統（《人月神話》的第二系統效應）
 - 客製化的硬體 + 新的程式語言 + 新的作業系統
- 影響
 - 雖然從未被廣泛採納，但探索了扣除電腦網路以外，幾乎所有作業系統設計的議題

PL/1 程式語言

- <http://en.wikipedia.org/wiki/PL/I>
- first specified in detail in the manual “PL/I Language Specifications. C28-6571” written in New York from 1965

- Sample program:

```
Hello2: proc options(main);  
        put list ('Hello, world!');  
end Hello2;
```


Multics 四大準則

- Naming and addressing
 - Combine virtual memory and file system
 - $\text{Filename} + \text{offset} == \text{Segment ID} + \text{offset}$
- Fine-grained sharing
 - Sharing procedures, not complete programs
- Dynamic Linking
 - Recompile without relinking
- Autonomy
 - Independent addresses for each application

UNIX



UNIX 公諸於世

“Perhaps the most important achievement of Unix is to demonstrate that a powerful operating system for interactive use need not be expensive...it can run on hardware costing as little as \$40,000.” and less than two man-years were spent on the main system software.”

- **The UNIX Time-Sharing System**, D. M. Ritchie and K. Thompson (1974) , 對 Multics 的反思
- <http://cm.bell-labs.com/who/dmr/cacm.html>

UNIX 兩大巨人

Kenneth Thompson

- Born in New Orleans in 1943
- Navy Brat
- Graduates with B.S. and M.S. (no Doctorate!) from UC Berkley in 1966
- Joins Bell Labs to work on Multics
- A mainframe timesharing operating system

Dennis Ritchie

- Born in Bronxville N.Y. in 1941
- Graduates with B.S. in Physics, M.S. and Doctorate in Applied Math from Harvard in 1968
- Doctoral Thesis was on “Subrecursive Hierarchies of Functions.”
- Followed his father to work at Bell Labs



“My undergraduate experience convinced me that I was not smart enough to be a physicist, and that computers were quite neat”

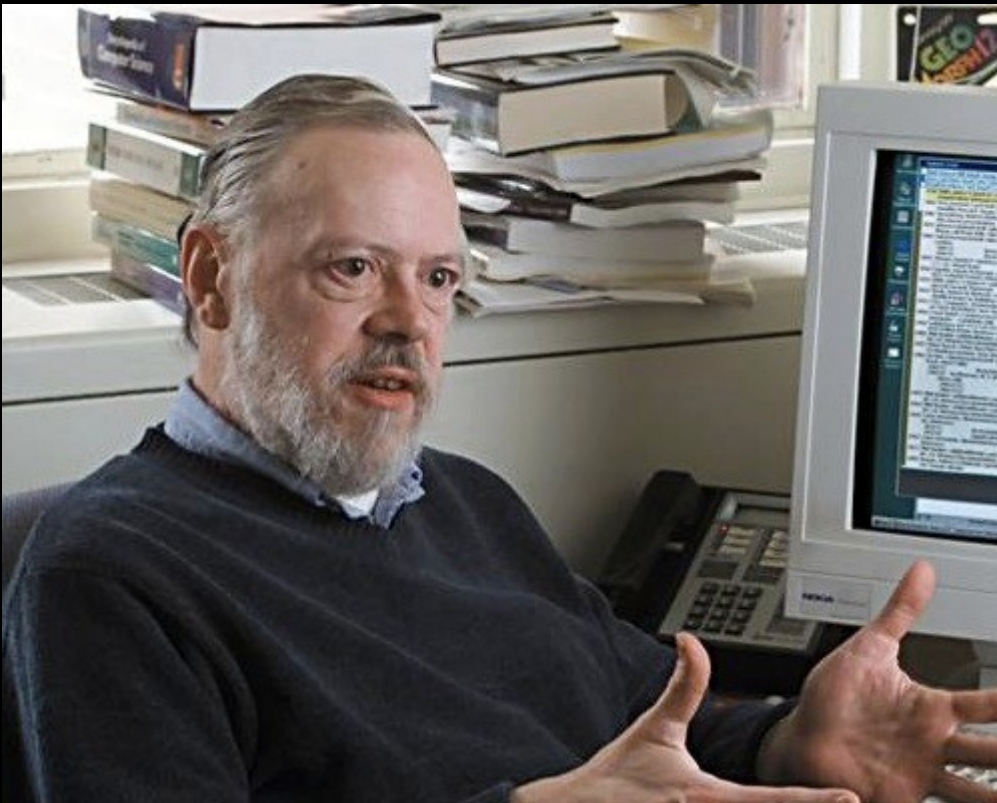
~Dennis Ritchie

除了 Steve Jobs , 2011 年逝世的科技先鋒

<http://www.ifanr.com/66495>

Dennis Ritchie

- 如果電腦沒有作業系統，沒有軟體，那麼它不過是一個大鐵盒子加電視加打字機而已
- 讓電腦擁有軟體運行的環境，同時也讓「程序員」成為一種職業
- 2011 年 10 月 12 日在位於美國新澤西州的家中被發現病逝



2011 年逝世的科技先鋒：Ken Olsen

Kenneth Olsen

- 世界上第一台運行著 UNIX 的電腦是小型機 PDP-7，由 Digital Equipment Corp. (DEC) 製造
- Ken Olsen 是 DEC 公司的創始人之一。DEC 一直致力於計算機的小型化，在計算機發展史的早期擔當著急先鋒的角色，推出過 PDP 與 VAX 兩大系列電腦產品
- 1992 年 Ken Olsen 從董事長退位
- 1998 年 DEC 公司被 Compaq 收購



2011 年逝世的科技先鋒：Paul Baran

Paul Baran

- 現代 Internet 誕生於美國空軍的一個項目 ARPANET，Paul Baran 是項目主要的參與者之一，他在為這個項目的工作期間發明了 packet switching 技術，一同提出此概念的還有 Leonard Kleinrock, Donald Davies
- 還是一個發明家，手上持有超過一打專利，包括印表機、衛星數據傳輸、互動電視甚至包括飛機上的金屬探測器



2011 年逝世的科技先鋒： Jacob Goldman

Jacob Goldman

- 如果沒有 Xerox PARC，現在的電腦會變成什麼樣？Jacob Goldman 是那個令施樂明白計算機將會是一場革命的人，在他的推動下施樂成立了 Xerox PARC，成就計算機歷史上的一段傳奇
- 20 世紀 70 年代到 80 年代，PARC 的成果豐碩，發明 OOP, Xerox Alto, GUI, Ethernet, WYSIWYG, VLSI
- 儘管 Goldman 只是創辦了 PARC，並沒有在裡面工作過，但沒有他，也就沒有那麼多重要的發明誕生



2011 年逝世的科技先鋒： John McCarthy

John McCarthy

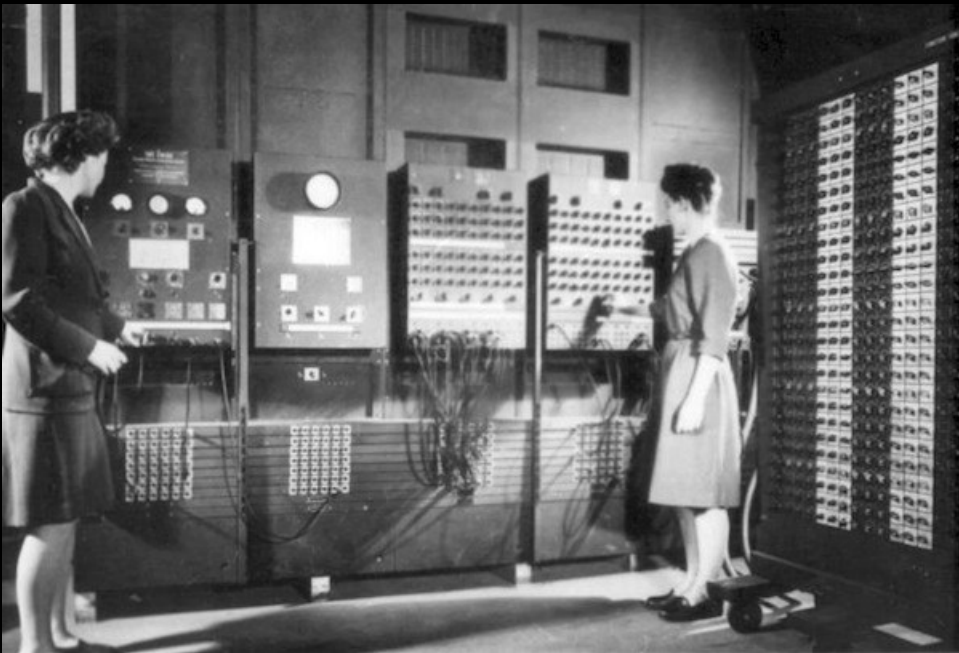
- 因提出 Artificial Intelligence 被成為「人工智慧之父」，同時還是 Lisp 語言的發明者——這個語言是電腦最古老的語言之一，而且直到今天還擁有相當的生命力
- 舉辦了第一屆電腦國際象棋大賽，由電腦與人對弈，1997 年 IBM 的「深藍」打敗了當時的世界棋王 Garry Kasparov



2011 年逝世的科技先鋒： Jean Bartik

Jean Bartik

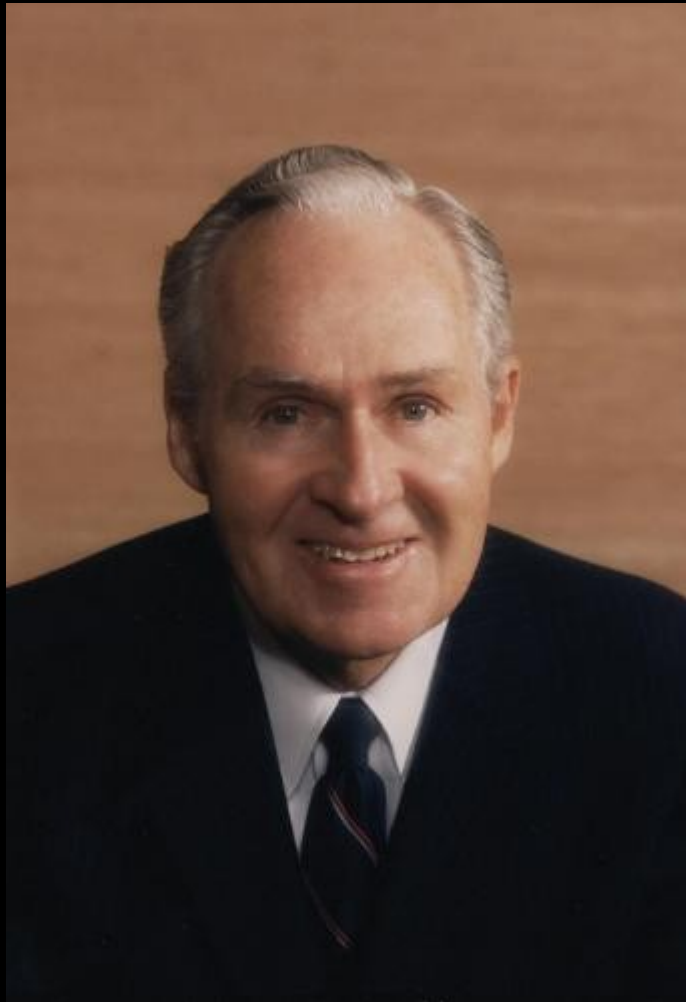
- ENIAC 是世界上第一台電腦，然而沒有鮮少有人知道 ENIAC 上面的程式開發由六名女性完成，2011 年這六名女性中的最後一位，Jean Bartik，去世了
- 除了 Jean Bartik，其餘五位傑出的女性分別是：Kathy Kleiman, Marly Meltzer, Kay Antonelli, Betty Holberton



2011 年逝世的科技先鋒： Bob Galvin

Bob Galvin

- 摩托羅拉創始人 Paul Galvin 之子，摩托羅拉 1959-1986 年間富有遠見的 CEO
- 在 Galvin 的領導下，摩托羅拉引領了全球手機產業的誕生。
- 摩托羅拉在他的帶領下還成為了全球半導體、尋呼、對講機、太空與軍事通訊以及汽車嵌入控制技術產業的領袖



資訊技術已發展到現在幾乎無所不能的地步，但這些理論居然都主要建立於 1970 年代

想想那時有如怪物般的機器，時代變化得肯定都超出這些締造者的想像了吧？

奠定編譯器技術的先鋒：

Grace Murray Hopper (1906 –1992 Jan 1)

- 終身在美國海軍任職，首位女性將官
- Computer bug!
- A-0 編譯器
- 影響 FORTRAN, COBOL
- Murray: 承先啟後



9/9

0800 Antan started
 1000 " stopped - antan ✓
 1300 (032) MP-MC ~~1.582647000~~
 (033) PRO 2 2.130476415
 conch 2.130676415

{ 1.2700 9.037847025
 9.037846995 conch
 4.615925059(-2)

Relays 6-2 in 033 failed special speed test
 in relay " 11.000 test.

Relay
 2145
 Relay 3371

1100 Relays changed
 Started Cosine Tape (Sine check)
 1525 Started Multi Adder Test.

1545



Relay #70 Panel F
 (moth) in relay.

First actual case of bug being found.
 1630 Antan started.
 1700 closed down.

寧靜致遠的傳奇人物： Dennis M. Ritchie (dmr)

- 1983 年與 Kenneth Lane Thompson 一同獲得 Turing Award，以表彰他們對「研究發展了通用的作業系統理論，尤其是實現了 Unix 作業系統」的貢獻。1999 年兩人又因創造發展 C 語言和 Unix 作業系統獲得了美國國家技術獎章
- 父親 Alistair E. Ritchie 是他的第一位人生導師。Alistair Ritchie 長期擔任 Bell Labs 科學家一職，在電路晶體管理論方面頗有造詣。在父親的影響下，dmr 在大學開始對計算機著迷，那時的計算機還是古老的打孔卡片設備。dmr 更加著迷於計算機處理的理論和實際問題，1968 年 dmr 獲得數學博士學位的論文，正是計算機理論相關的《遞迴函數的層次》
- 加入 Bell Labs 室不久，dmr 就參與了 Multics 項目，負責多道處理機的 BCPL 語言和 GE650 的編譯器，它們都屬於 GECOS 系統。同樣的，他也寫了 ALTRAN 語言的代數編譯器，那是用於符號計算機的一種語言和系統

GCOS: General Comprehensive Operating System

http://en.wikipedia.org/wiki/General_Comprehensive_Operating_System

寧靜致遠的傳奇人物： Dennis M. Ritchie (dmr)

- dmr 在 Bell Labs 到了對他職業生涯影響最大的人，Ken Thompson。 dmr 曾表示 Unix 大部分是 Ken 的工作，不同於 DMR 對理論的偏好，Ken 是位電子發燒友，與 dmr 形成互補
- 完成 Unix 和 C 語言之後， dmr 並未停止創新工作
 - 1995 年，發佈 Plan 9 作業系統
 - 1996 年，發佈 Inferno 作業系統與 Limbo 程式語言
- 1978 年， dmr 和 Brian Kernighan 合作出版的《The C Programming Language》，成為 C 語言方面最權威的教材也是後來所有語言著作的範本

KISS = Keep It Simple Stupid

- Dennis Ritchie : "Unix is simple. It just takes a genius to understand its simplicity."
- 為何原本屬於簡單的事物，要經歷不簡單的路途，才能領悟它的簡單？
- dmr 的人生信條是保持簡單，終生保持單身，只在貝爾實驗室工作 (26 歲開始)，最後獨處在老家，最後退休獨居在家，共事 20 年的同事 Rob Pike (在 google 服務) 從加州到新澤西去拜訪他，才發現 dmr 已過世
- dmr 的一生完全符合他設計 UNIX 時的思想 : KISS

紀念已故的 dmr，在 Bell Labs 的追思會 (新澤西梅山); 2012 年 9 月 7 日

- Alfred Aho (前 bell Labs 成員、哥倫比亞大學電腦講座教授，《編譯原理》龍書第一作者，awk 共同作者) 發表演說
- <http://www.youtube.com/watch?v=GfoSbffSIQ4>
- <http://www.ituring.com.cn/article/14315>
- 僅僅微軟 Windows XP 作業系統的程式碼就超過 4500 萬行；SAP 公司開發的商業應用環境程式碼超過了 2 億 5000 萬行
- 剛登上火星的好奇號漫遊車 (Curiosity Rover)，它的空中吊車降落桿非常厲害，其控制程序就是由 380 萬行 ISO C 代碼實做 dmr 的發明，花了 40 年時間，從貝爾實驗室的 Unix 研發室走到了火星的蓋爾隕坑
- 1989 年，Richard Gabriel 寫了一篇文章，題目用了矛盾修辭，《Worse is Better》，闡釋為何著重簡單靈活的 Unix/C 方法在市場上壓倒了著重一致完整的 MIT 方法。此描述也稱為「新澤西風格」：「短小即美好」

有次有人問 Dennis Ritchie :
「一位程式設計師從新手到精通 C
語言，並能寫出卓爾不凡的成品代
碼，據您的經驗要多久？」

Dmr 答道：

「我不知道，我又不需要學 C！」

UNIX 背景

- 網路農夫的〈 UNIX 系統簡介〉：

<http://www.farmer.idv.tw/?viewDoc=479>

- Micro implementations of Unix:

<http://www.robotwisdom.com/linux/nonnix.html>

- The Strange Birth and Long Life of Unix

<http://spectrum.ieee.org/computing/software/the-strange-birth-and-long-life-of-unix/0>

UNIX v6 廣泛的學術研究

"After 20 years, this is still the best exposition of the workings of a 'real' operating system."

Ken Thompson

Lions' Commentary on UNIX[®] 6th Edition

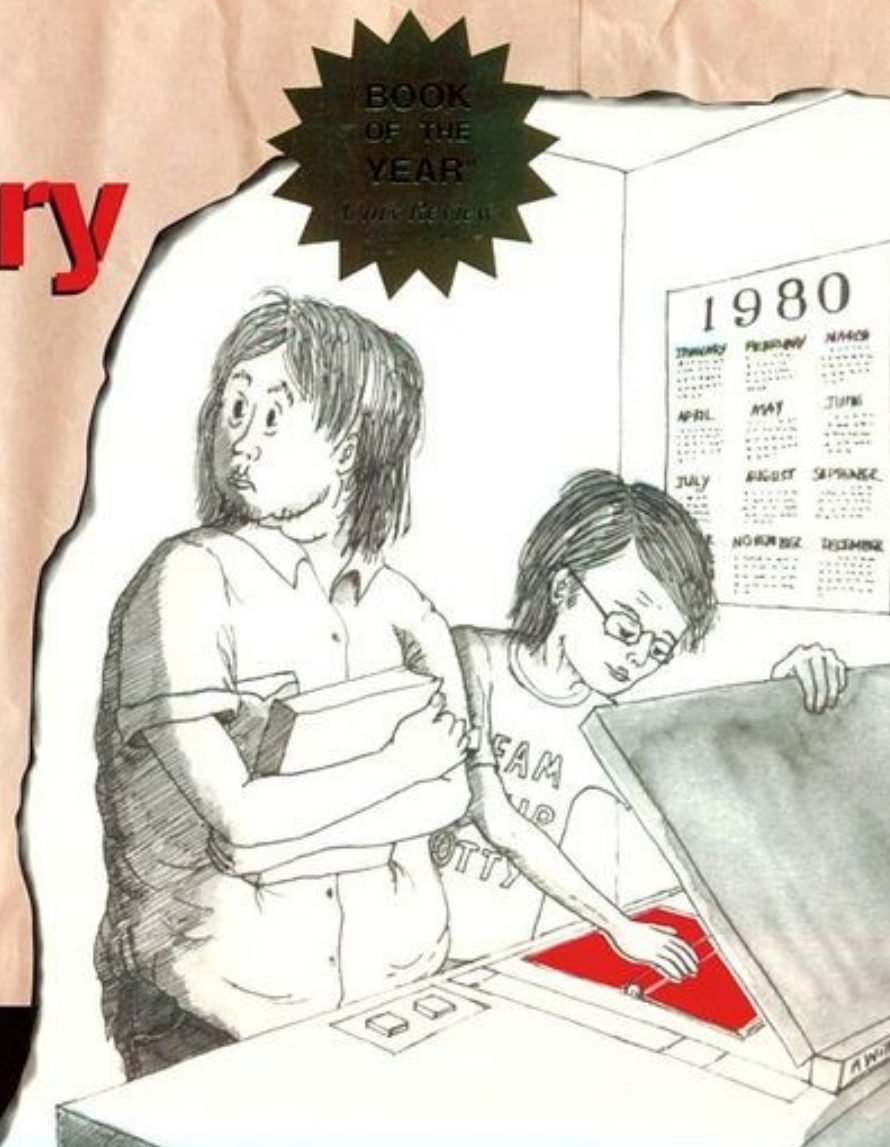
with Source Code

John Lions

Foreword by Dennis Ritchie

BOOK
OF THE
YEAR

Library Journal



UNIX v6 廣泛的學術研究

- 1976 年新南威爾斯大學 (也是催生 OKL4 與一系列重要作業系統研究的學術殿堂) John Lions 教授以註解的 UNIX v6 原始程式碼作為課堂教材，揭露 UNIX 簡潔又強大的設計
- AT&T 對 UNIX v7(含) 之後的版本採取嚴苛的授權條件，而 v6 則是允許課堂使用的最後一個版本
- 在 1996 年再次印刷，至今仍販售
- 2002 年一月份，Caldera International (目前為 SCO) 對以下 UNIX 版本重新授權，變更為 BSD License，自此研究不再受限
 - UNIX/32V, UNIX Version 1-7
 - http://en.wikipedia.org/wiki/Ancient_UNIX

Mutlics 與 UNIX 簡短比較

- 研發資源投入量
- ? Work years vs. 2 man-years
- 關鍵概念

Feature	Multix	Unix
Key abstraction	Unify file = memory	Unify I/O = file
Protection	rings (jump to memory)	suid (execute file)
Sharing	N segments; arbitrary sharing	3 segments (text, heap, stack) text is shared (RO); Communication between domains via files, pipes

UNIX 作為 Multics 的反思

- dmr 始終意識到使用最少資源的好處，當年所做的很多事情都是對當時使用的同類系統的反思
- 發明 Unix 就是對 Multics 操作系統的反思
- C 語言是對 Multics 用的 PL/1 的反思
- 而 EPL, Bliss 尤其是 BCPL 則對 C 語言有著更多積極的影響。這些促使 C 語言取得成功的因素不僅啟發了後來的其他語言和系統，而且也使 C 語言能繼續和後來的新語言和系統並駕齊驅

UNIX 早期的限制

- 20 世紀 70 年代的計算受到當時條件的嚴重制約，簡直不堪想像
- 設計 Unix 初期，只能讓它運行 24000 bytes 類似的約束無所不在：磁盤空間、帶寬（網絡才剛發明），甚至 I/O 設備
- Unix 的命令和輸出都十分簡短，這是根據所用的印表機做的設計，爲了節省紙張

「UNIX 室」的文化

- 因為第一部 Unix 是運行在一個糟糕的小型機上，是在一間有很多終端的機房裡
- 從事此系統工作的人都聚在這個房間——要用計算機，你就不得不在它那兒
- 當時這想法一點兒也不怪；它自然演化自古老的 IBM7090 那種每次預定一個小時的方式
- 這夥人就喜歡這樣幹活，因此當機器移到遠離終端的另一間房間，甚至後來可以從自己的辦公室連入了，人們還是一樣聚集在那個有好多終端的『Unix 室』，設計、討論程式，而咖啡機也在那。
- 這可能是 Unix 作為一項技術，得以成功的最重要的文化因素

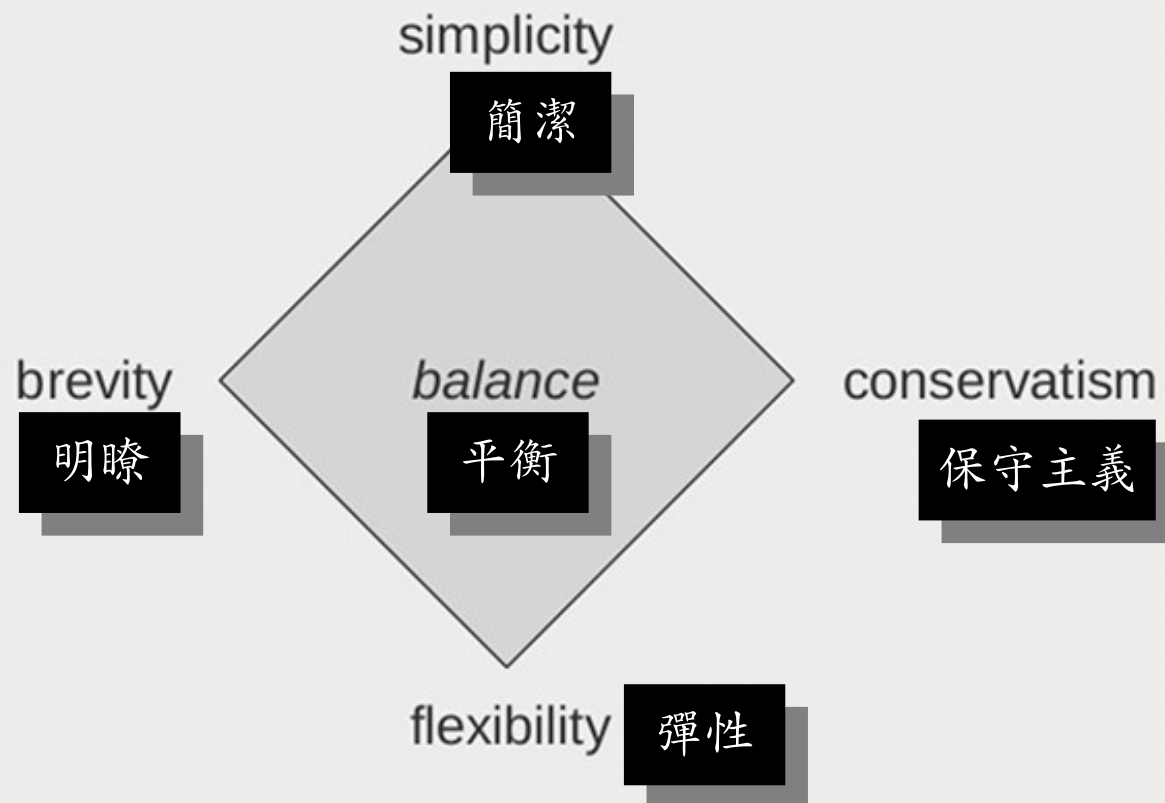
Unix 哲學 [9 條主要準則]

- 小即是美
- 讓每一個程序只做好一件事情
- 盡快建立原型
- 舍高效率而取可移植性
- 使用純文本文件來存儲數據
- 充分利用軟件的槓桿作用
- 使用 shell 腳本來提高槓桿作用和可移植性
- 避免強制性的用戶界面
- 讓每一個程序都成為過濾器

Unix 哲學 [10 條次要準則]

- 允許用戶定製環境
- 儘量使操作系統內核小而輕巧
- 使用小寫字母，並儘量保持簡短
- 保護樹木
- 沉默是金
- 並行思考
- 各部分之和要大於整體
- 尋找 90% 的解決方案
- 更糟的反而效果更好
- 層次思考

So, what is “Beautiful Code”?



And if you also make sure to have fun writing and reading code, you will experience happiness as a programmer.

Happy Hacking!

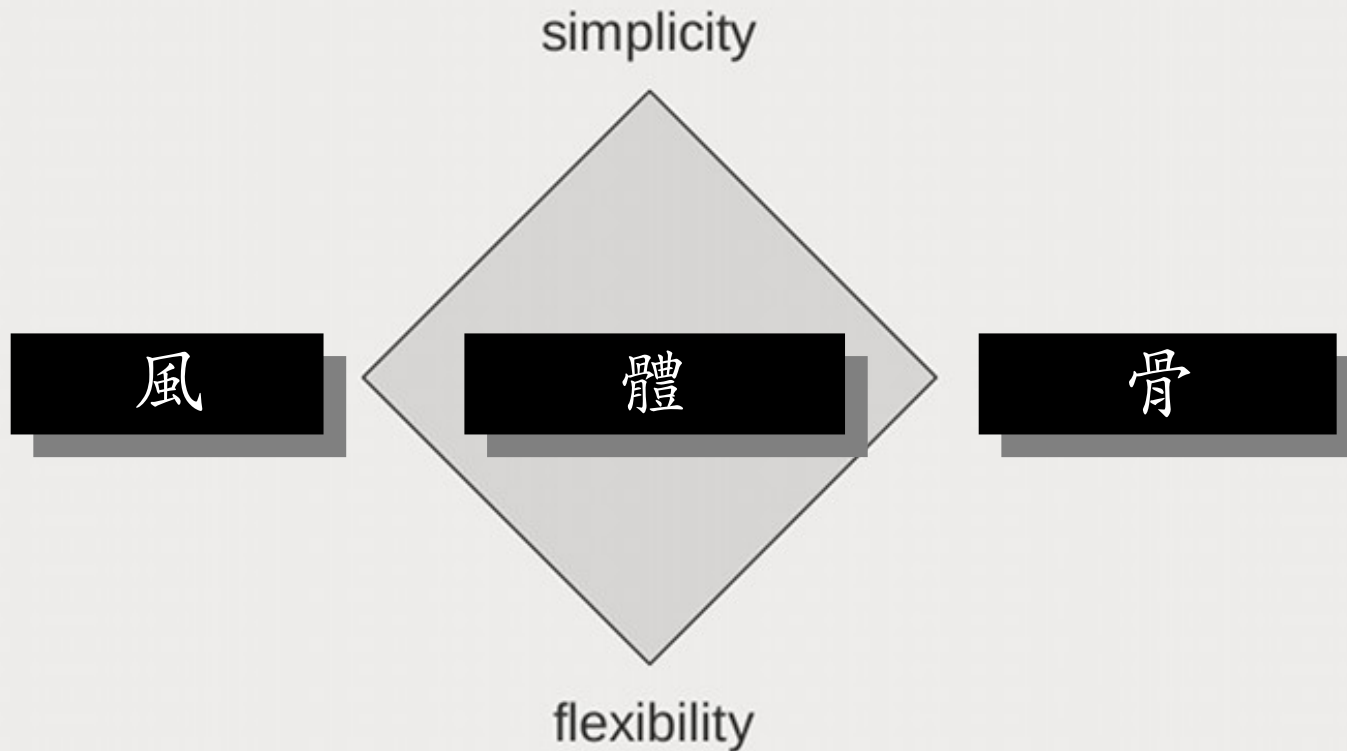
文化資產：《文心雕龍》

- 如何從閱讀前代佳作，歸納整理寫作法則，以爲創作法門？
- 摯虞〈文章流別論〉，李充〈翰林論〉，曹丕《典論·論文》，陸機〈文賦〉
- 劉勰集前代之大成，《文心雕龍》順勢而生
- 中國首部精於文學評論的專書，貫串全書的兩大重點
 - 反對不切實用的浮靡文風
 - 講求實用的落實文風

《文心雕龍》的「體」「風」「骨」

- 「體」源於最基本的含意
「體」對文人而言，不是虛縹的形式，而是跟身體一樣是現實有形的
體必須伴隨「風」和「骨」才活得下去
- 「風」象徵著氣和節奏
- 「骨」則是與骨骼和結構相聯繫

So, what is “Beautiful Code”?



And if you also make sure to have fun writing and reading code, you will experience happiness as a programmer.

Happy Hacking!

「文心之作也，本乎道，
師乎聖，體乎經，酌乎緯，
變乎騷，文之樞紐，亦云極矣」

- 劉勰掌握文學的關鍵，是從形而上之道，透過聖人表現為經、緯之文，又轉變為騷體（以《離騷》為代表）的文章；可說是從虛渺演進為現實，同時亦是從極廣泛的問題進入極專門的問題上
- 這樣一系列表現過程，實際也是文之演變過程

劉勰試圖將所有的書籍視為文學書來分析，
程式創作何嘗不得如此？

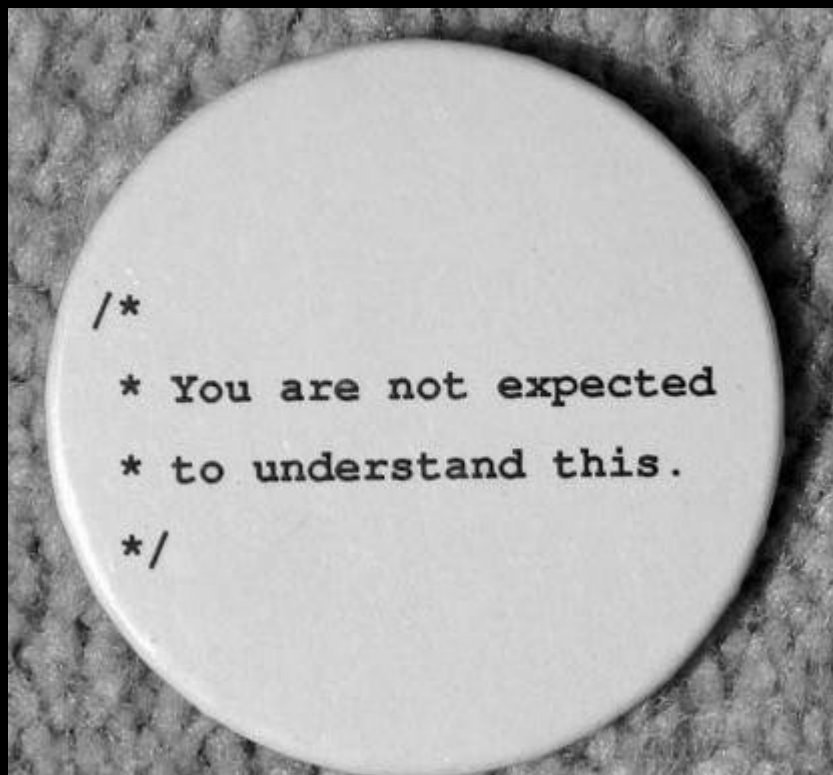
「文 C 雕龍」系列演講之提綱

- 總論 -- 「文之樞紐」，回顧理論基礎
- 文體論 -- 探究「文體」，也就是程式設計的思維與表現
- 創作論 -- 創作過程、作家風格、文質關係、寫作技巧
- 批評論 -- 從不同角度對過去提出批評，並對批評方法作探討
- 總序 -- 說明了自己的創作目的和部署意圖

總論：文之樞紐

- C 語言的根源與強烈的 UNIX 設計哲學
- 以 UNIX 探討對象，奠定理論基礎
- C 語言的手法
 - 與系統緊密的關聯
 - 模組化、清晰的介面
 - 物件導向設計

UNIX 精妙的註解



❖ Sixth Edition Unix (1975). Line 2240-2243

```
/* You are not expected to understand this. */  
if(rp->p_flag&SSWAP) {  
    rp->p_flag =& ~SSWAP;  
    aretu(u.u_ssav);  
}
```

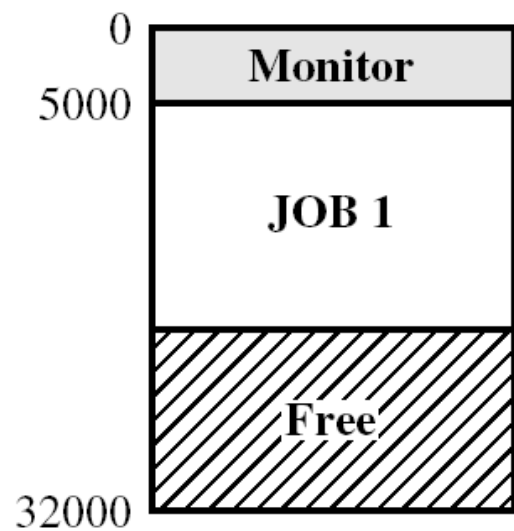
From: Dennis Ritchie <dmr@alice.att.com>
Date: 2 Apr 92 09:34:24 GMT
Organization: AT&T Bell Laboratories, Murray Hill NJ

People might be interested in more of the context of the famous 'you are not expected to understand this' comment. (Tim Smith is wrong on the details.) It was made somewhat in the spirit of '**this won't be on the exam,**' not as a contemptuous challenge. Nevertheless, people did find it necessary to understand it, and the comment was too flippant.

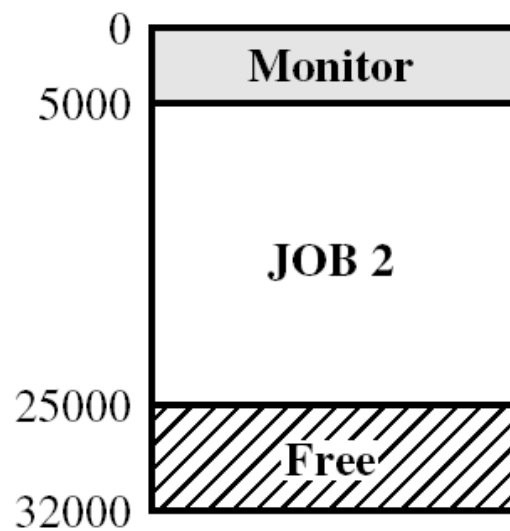
And of course, the real joke was that we did not understand what what was really happening either: the setu/retu mechanism of pre-Seventh Edition Unix was basically unworkable, because it depended inextricably on subroutine calling conventions of the PDP-11 implementation, and more fundamentally because it was not the right way to do things. Specifically, as the comment says, 'savu' arranges that a routine that subsequently calls 'retu' jumps out not to a location near the 'savu' (as with setjmp/longjmp), but to the routine that called the routine with the 'savu.'

過往：*MUTICS: Time-Sharing Systems*

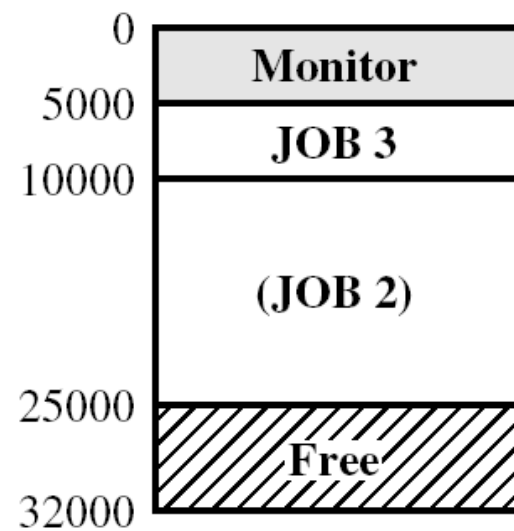
- 因為多重程式的使用，批次處理可以相當有效率。然，許多工作需讓使用者與電腦互動
- 多重程式也可用來處理多個交談式工作，稱為「分時」——處理器的處理時間被許多使用者所分享
- 1961 年 CTSS 系統：系統時鐘每 0.2 秒發出一次中斷請求，作業系統重新取得控制權
- 今日的個人電腦週邊，滑鼠每秒鐘產生 300 次中斷，這些中斷佔用處理器所有工作週期不到 1%



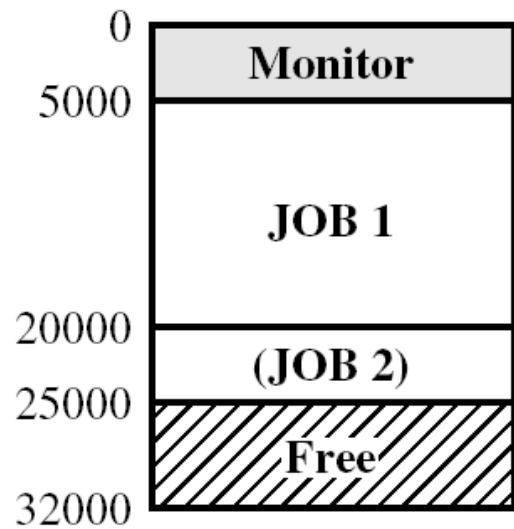
(a)



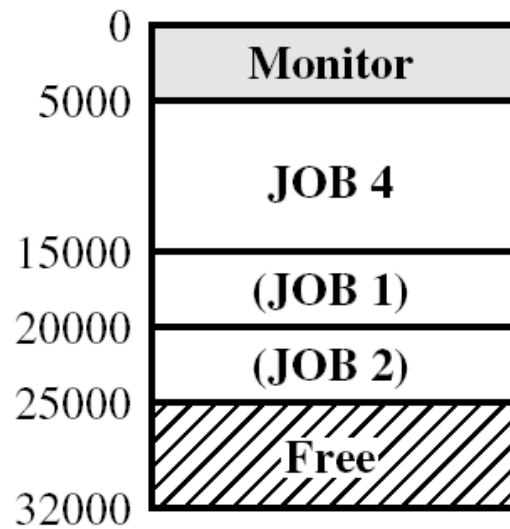
(b)



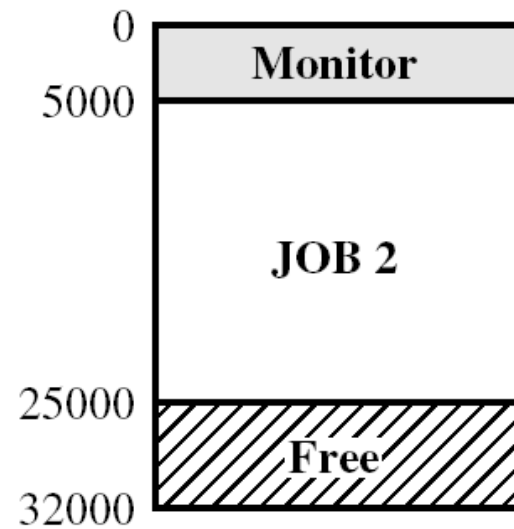
(c)



(d)

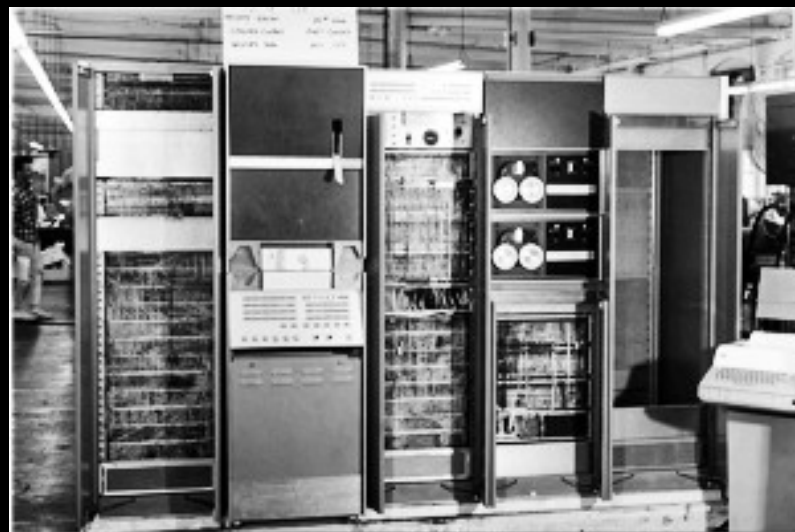
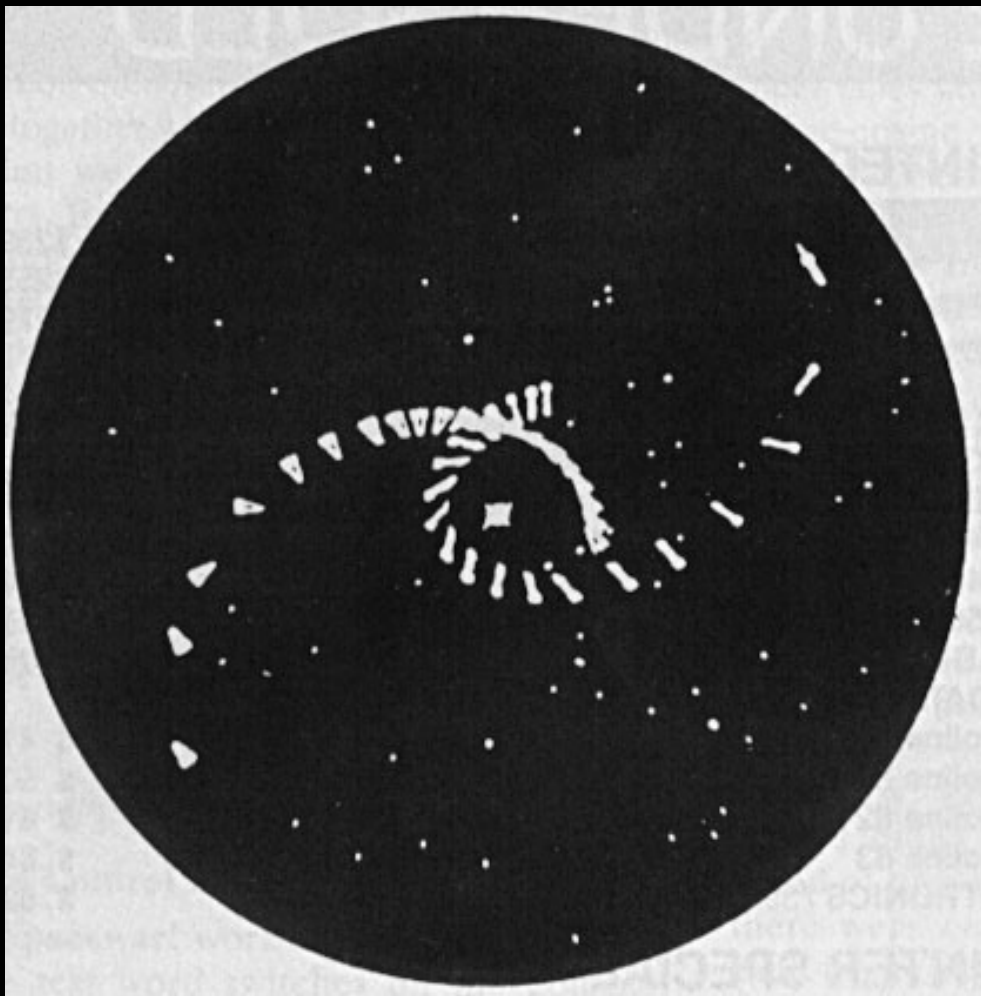


(e)



(f)

大師也愛玩電動！



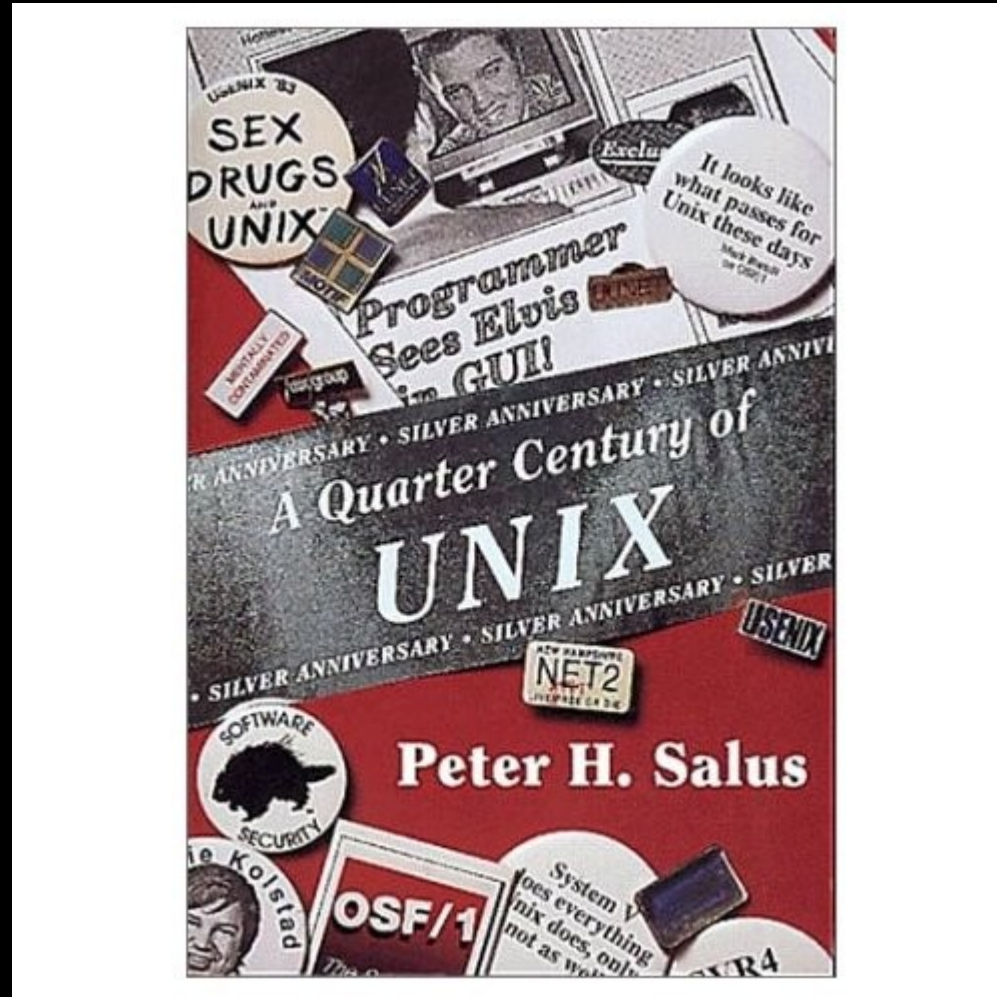
MULTICS: Multi User file system from MIT/GE/Bell Labs, on expensive GE-645. Introduced hierarchical file system

1965 Fall → MIT/GE/Bell Labs → 1969

MULTICS died, but Thompson/Ritchie still wanted to play Space Pilot. In order to do so, they decided to create a simplified version that would work on a PDP-7 that was laying around.

「體」

《 A Quarter Century of UNIX 》



第一代的 UNIX 有什麼？

- 60 個使用者指令

```
as = assembler; cat = concatenate (串連)  
file; db = symbolic debugger; dtf =  
format DECTape; sdate = adjust date and  
time; su = become superuser; ...
```

- File

- Interface

```
creat (create a new file)
```

- 多人多工

UNIX 原始硬體組態

- 最早在 DEC PDP-7 主機上開發，稍後移到 PDP-11/20
- 在 PDP-11 之前的硬體並非以 byte (8 bits) 為基礎
 - PDP-7: 18-bit
 - PDP-6: 36-bit
 - PDP-5: 12-bit
- 為什麼當初會有 18-bit 與 36-bit 的系統呢？
 - “Many early computers aimed at the scientific market had a 36-bit word length. ...just long enough to represent positive and negative integers to an accuracy of ten decimal digits... allowed the storage of six alphanumeric characters encoded in a six-bit character encoding.”



- <http://cm.bell-labs.com/cm/cs/who/dmr/picture.html>
- 這張宣傳照大概拍攝於1972年，展示的是Ken和dmr在PDP-11前工作的場景
- 自右向左，設備主要組成如下：
 - 最右邊的桌子上，有人認出是一台VT01A存儲管顯示器（基於Tek 611），配一個小鍵盤。不太容易辨認
 - 主CPU機櫃，部分被桌子擋住。處理器為PDP-11/20；這應該是我們的第二台，配有Digital專屬系統KS-11存儲管理單元。我們的第一台只是PDP11，而非「11/20」。它和其他機櫃上那些扭曲的方形是DECtape盒上的標籤
 - 第二個機櫃。Steve Westin仔細檢查這張相片，發現了11/45 CPU面板的頂部，只是在Ken使用的電傳打字機右邊那台上方露出一點點。它上面是一台紙帶閱讀機
 - 第三個機櫃頂部是台雙DECtape驅動器
 - 第四個機櫃裝的也是DECtape驅動器，裡面可能還包含BA-11擴展盒
 - 第五個機櫃內裝RK03磁碟機，由Diablo（後被Xerox收購）製造並OEM給Digital。Digital後來開始生產自有版本(RK05)
 - 第六個機櫃裝有RF11/RS11控制器和固定頭磁盤。當時/（根分區）和交換分區就放在這些磁盤上，/usr則位於RK03
 - 機器上面放的應該是磁帶（magtape）。就在Ken下巴下面的位置可能是台TU10倒帶器，幾乎看不到，要是樂意調一下監視器的亮度和對比度，你會看到一點
- 設備前面的是：
 - 兩台Teletype 33終端

範例 C 程式

/usr/include/stdio.h

```
/* comments */
#ifndef _STDIO_H
#define _STDIO_H

... definitions and
prototypes

#endif
```

/usr/include/stdlib.h

```
/* prevents including file
 * contents multiple
 * times */
#ifndef _STDLIB_H
#define _STDLIB_H

... definitions and
prototypes

#endif
```

example.c

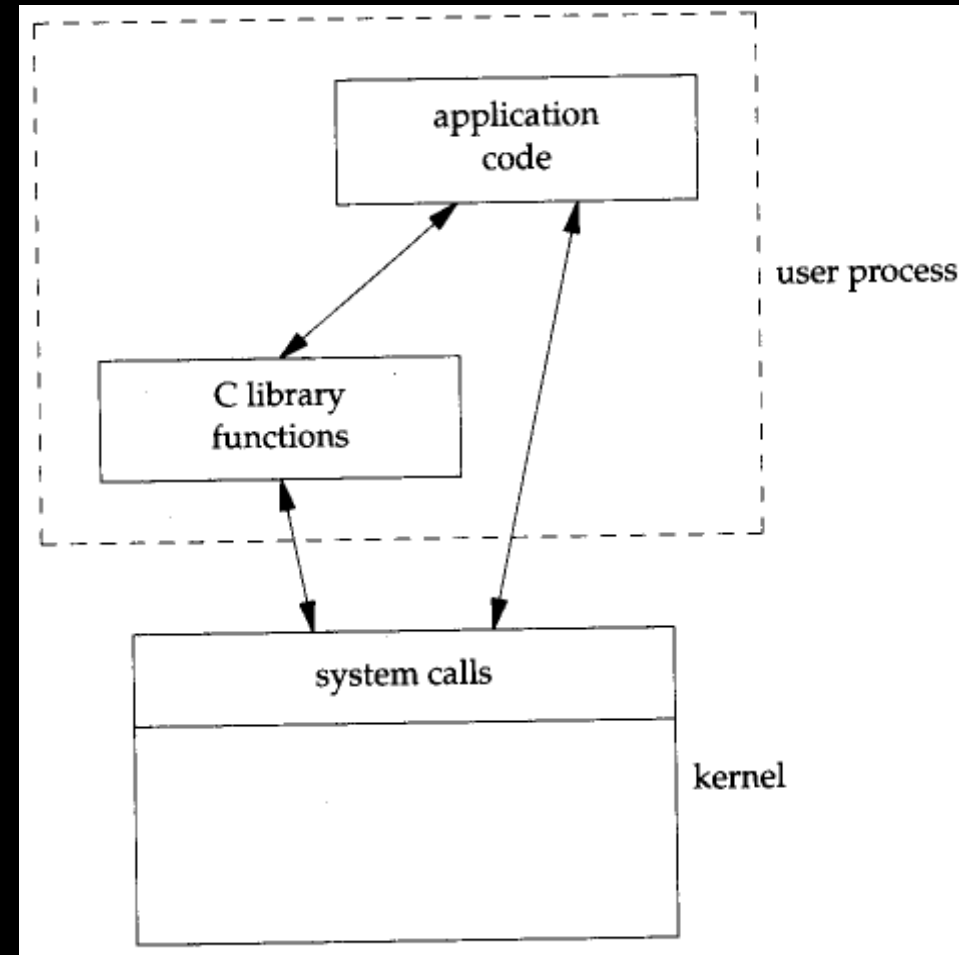
```
/* this is a C-style comment
 * You generally want to place
 * all file includes at start of file
 * */
#include <stdio.h>
#include <stdlib.h>

int
main (int argc, char **argv)
{
    // this is a C++-style comment
    // printf prototype in stdio.h
    printf("Hello, Prog name = %s\n",
           argv[0]);

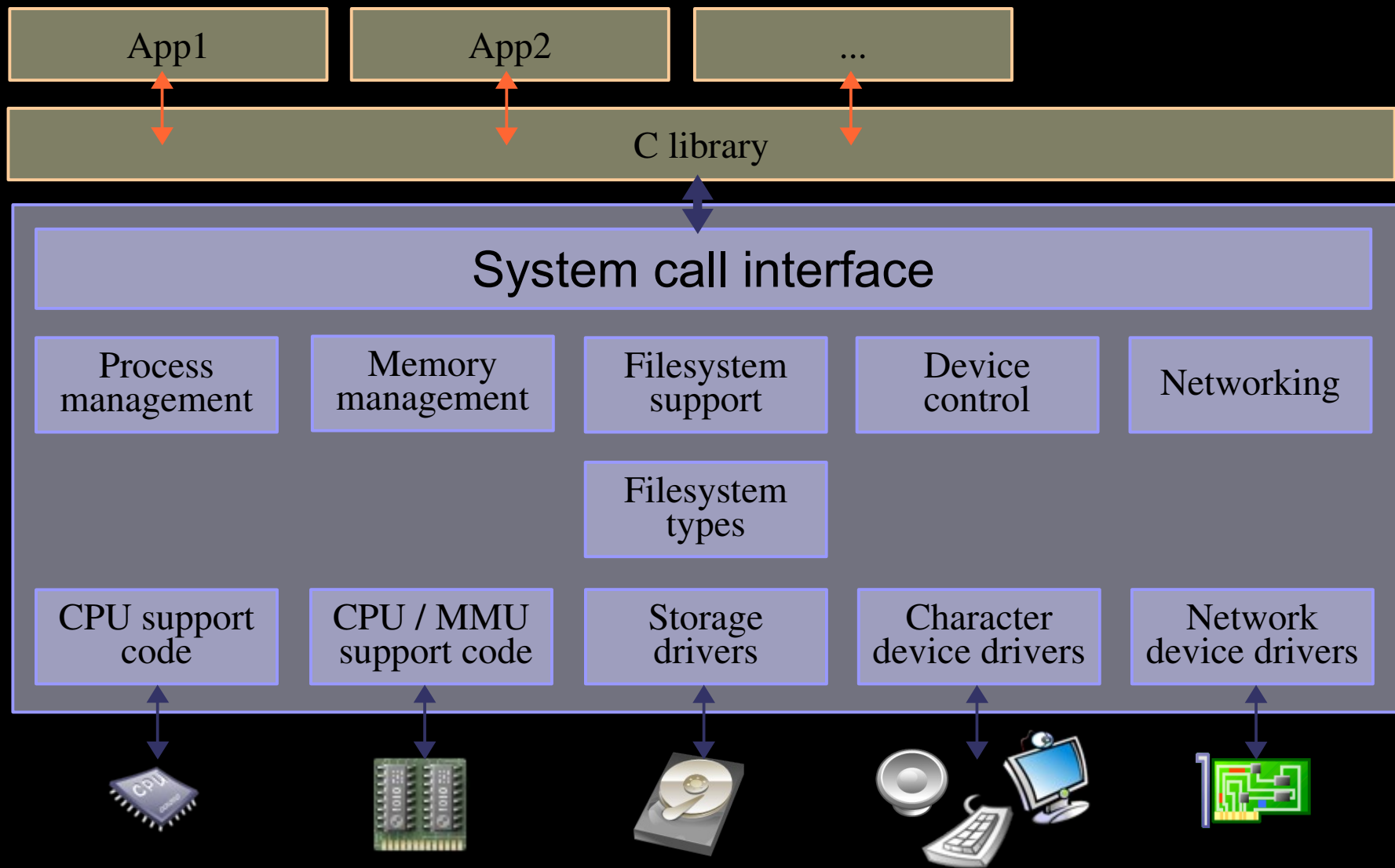
    exit(0);
}
```

典型的系統需要 System Call

- Kernel 實做一系列特別的系統服務
- 使用者的程式透過觸發硬體 TRAP 而呼叫 Kernel 的服務
- TRAP 使 CPU 切入保護 / 特權模式，致使 Kernel 執行 system call。直到做完，CPU 切回使用者模式
- A C language API exists for all system calls



Kernel : 架構觀點



「骨」

那麼， Kernel 是 ...

- 在開機程序中，載入至主記憶體，並常駐於實體記憶體的程式
- 負責管理 CPU 狀態的切換、個別程序、檔案系統，以及與硬體裝置的通訊互動

Unix 系統架構

圖形介面

Web browser, office, multimedia...



命令列指令

ls, mkdir, wget, ssh, gcc, busybox, shells (scripts)...



共享程式庫

libstdc++, libxml, libgtk2...

C 程式庫

GNU C library, uClibc...

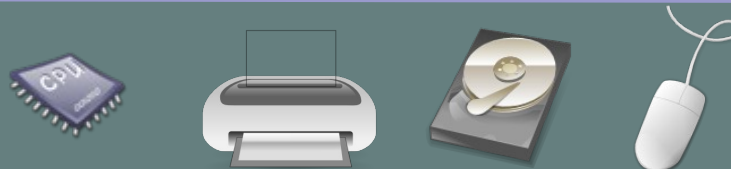


作業系統核心

Linux, Hurd...



硬體與週邊



典型核心架構

execution
environment

application

libraries

user

kernel

System call interface

System
Services

File subsystem

IPC

Buffer cache

scheduler

char

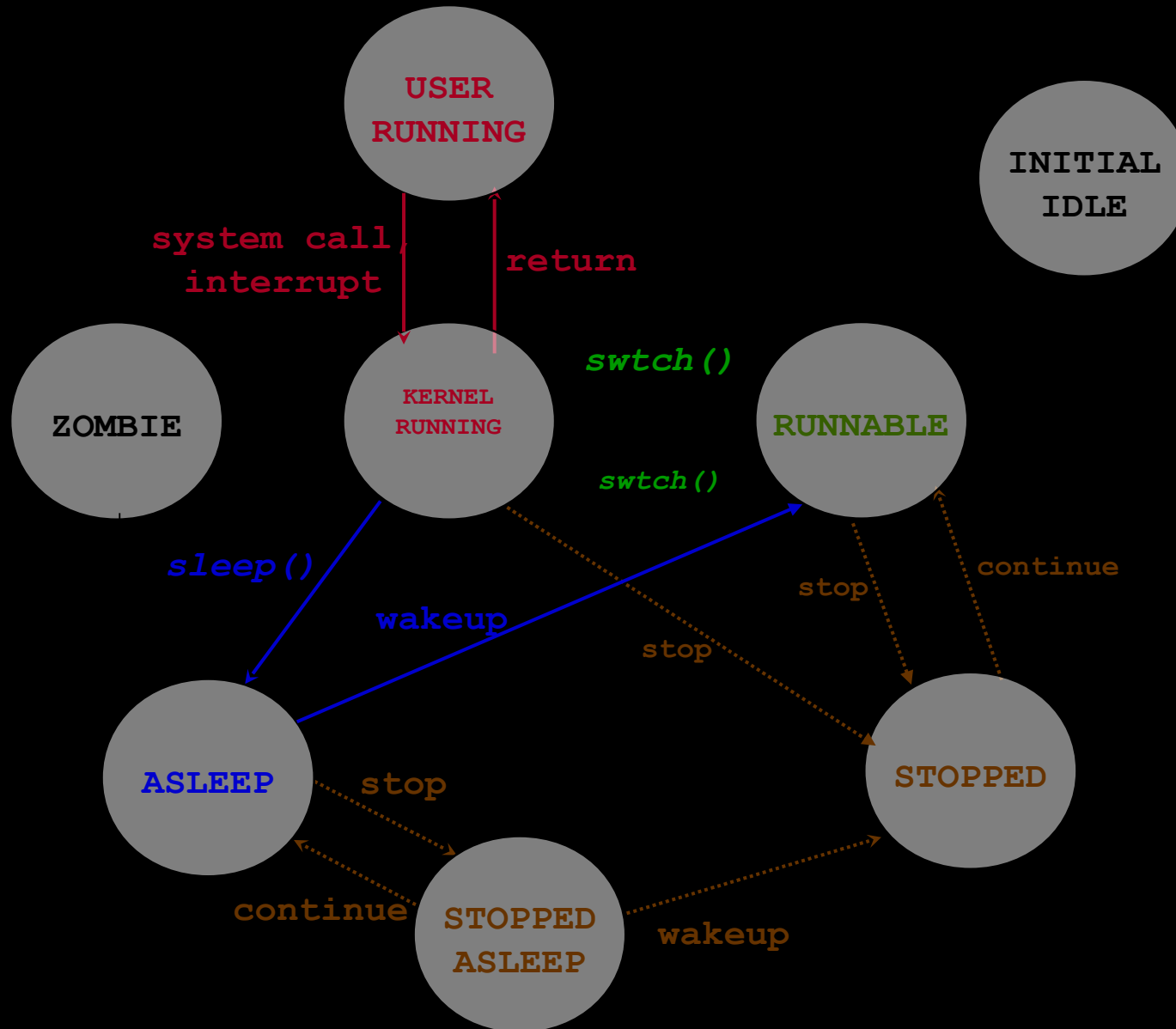
block

memory

Device drivers

hardware

Process states



Process Address Space

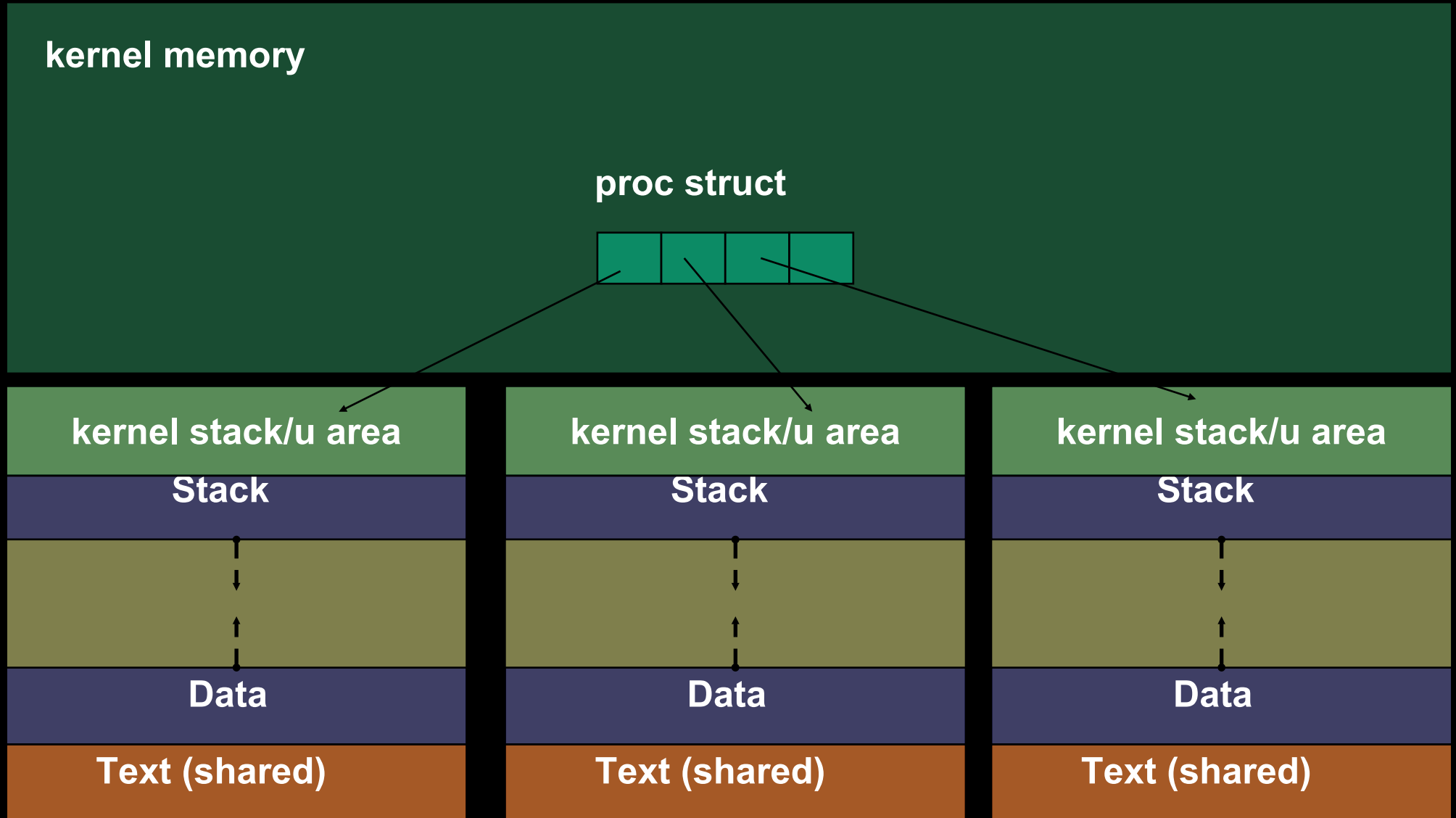
Kernel stack

stack

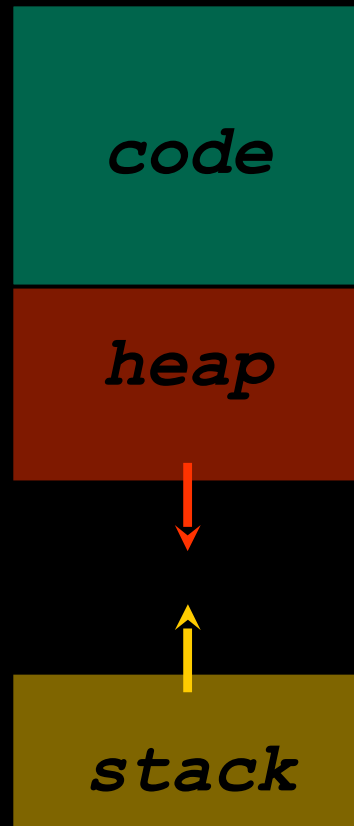
Data

Text (shared)

Big Picture of Process



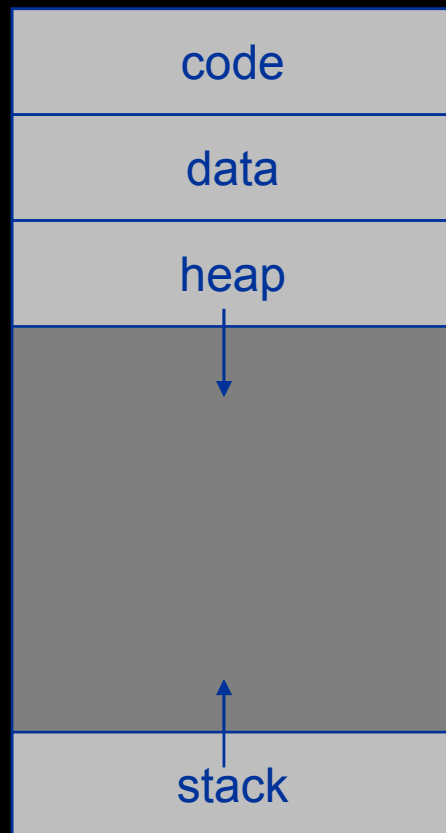
Example memory layout



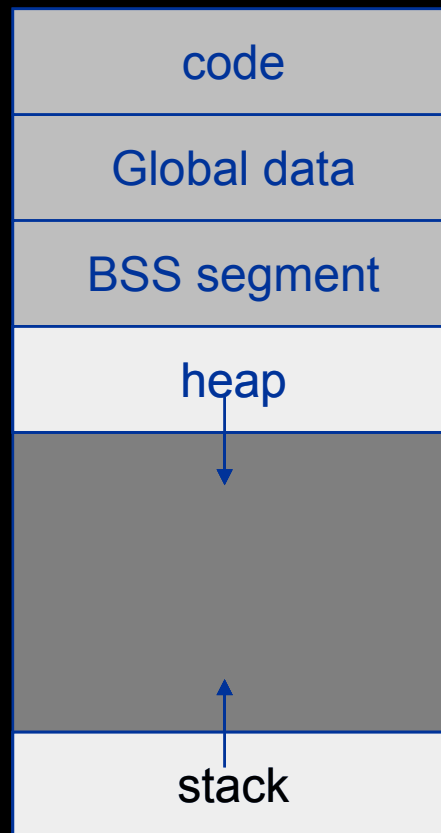
Dynamic allocation
`malloc()`, `free()`

Memory Layout

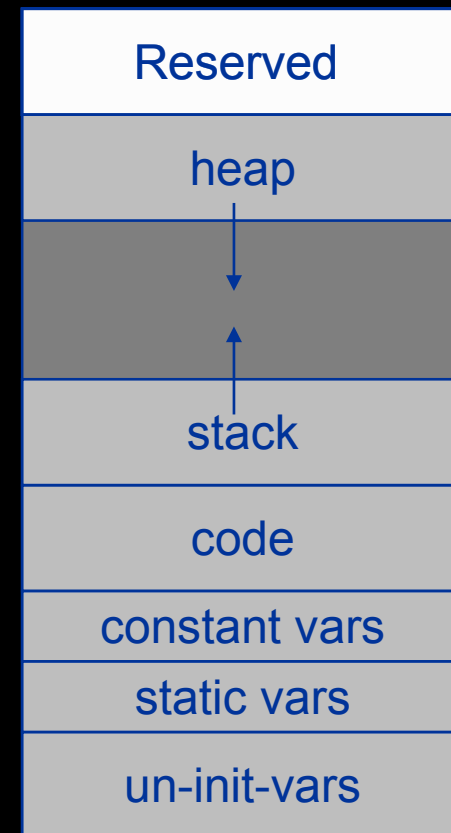
Generic



Win32



Unix



Unix 與物件導向設計

- 現代 UNIX(-like) 系統，充斥物件導向設計

”We observed above that the Unix tradition of modularity is one of thin glue, a minimalist approach with few layers of ***abstraction between the hardware and the top-level objects of a program.***

Part of this is the influence of C. It takes serious effort to simulate true objects in C. Because that's so, piling up abstraction layers is an exhausting thing to do. Thus, object hierarchies in C tend to be relatively flat and transparent.

Even when Unix programmers use other languages, they tend to want to carry over the thin-glue/shallow-layering style that Unix models have taught them.”

http://catb.org/esr/writings/taoup/html/unix_and_oo.html

「文心之作也，本乎道，
師乎聖，體乎經，酌乎緯，
變乎騷，文之樞紐，亦云極矣」

從形而上之道 [MUTICS 的動機 + UNIX 設計哲學]，透過聖人 [Thompson/Ritchie] 表現為經 [分層 + 模組化]、緯 [系統呼叫 + 介面 + 工具程式] 之文，又轉變為騷體 [雋永的泛 UNIX 系統與 C 語言設計的觀點] 的文章；可說是從虛渺演進為現實，同時亦是從極廣泛的問題進入極專門的問題上

插曲：C is subset of C++?

- C 「可以是」 C++ 的 subset，大部分的情況
- 但概念上仍有落差

```
$ gcc -o sizeof sizeof.c
$ ./sizeof
4, 1
```

```
$ g++ -o sizeof sizeof.c
$ ./sizeof
1, 1
```

`sizeof.c`

```
#include <stdio.h>

int
main (int argc, char **argv)
{
    printf"%d, %d\n",
        sizeof('J'),
        sizeof(char));
    return 0;
}
```

在 ANSI C 規格中，`sizeof(char)` 被嚴格定義為 1 個 `size_t`

依據 Standard C++ language definition 的說法：

A class with an empty sequence of members and base class objects is an empty class.

Complete objects and member subobjects of an empty class type shall have nonzero size.

這也是說，沒有任何一個 complete object 可有 zero size，任何空的 structure 空間至少為 "1"

- C 語言的根源與強烈的 UNIX 設計哲學
- 以 Linux Kernel 為探討對象，
奠定理論基礎
- C 語言的手法
 - 與系統緊密的關聯
 - 模組化、清晰的介面
 - 物件導向設計



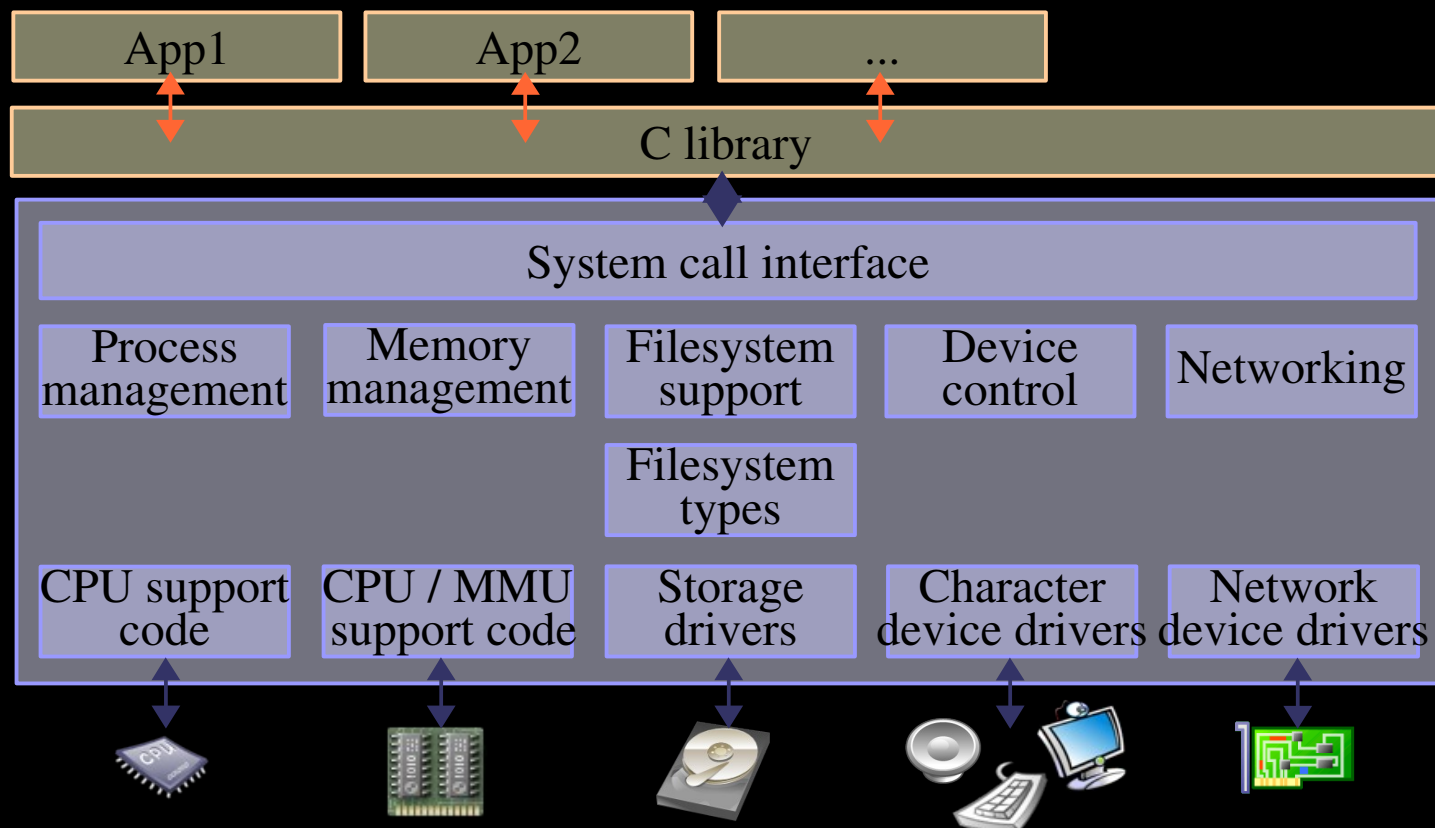
高度物件導向的 C

賞析：
Linux Kernel

**Everything is
file.**

Everything is file.
::Device File::

Kernel architecture

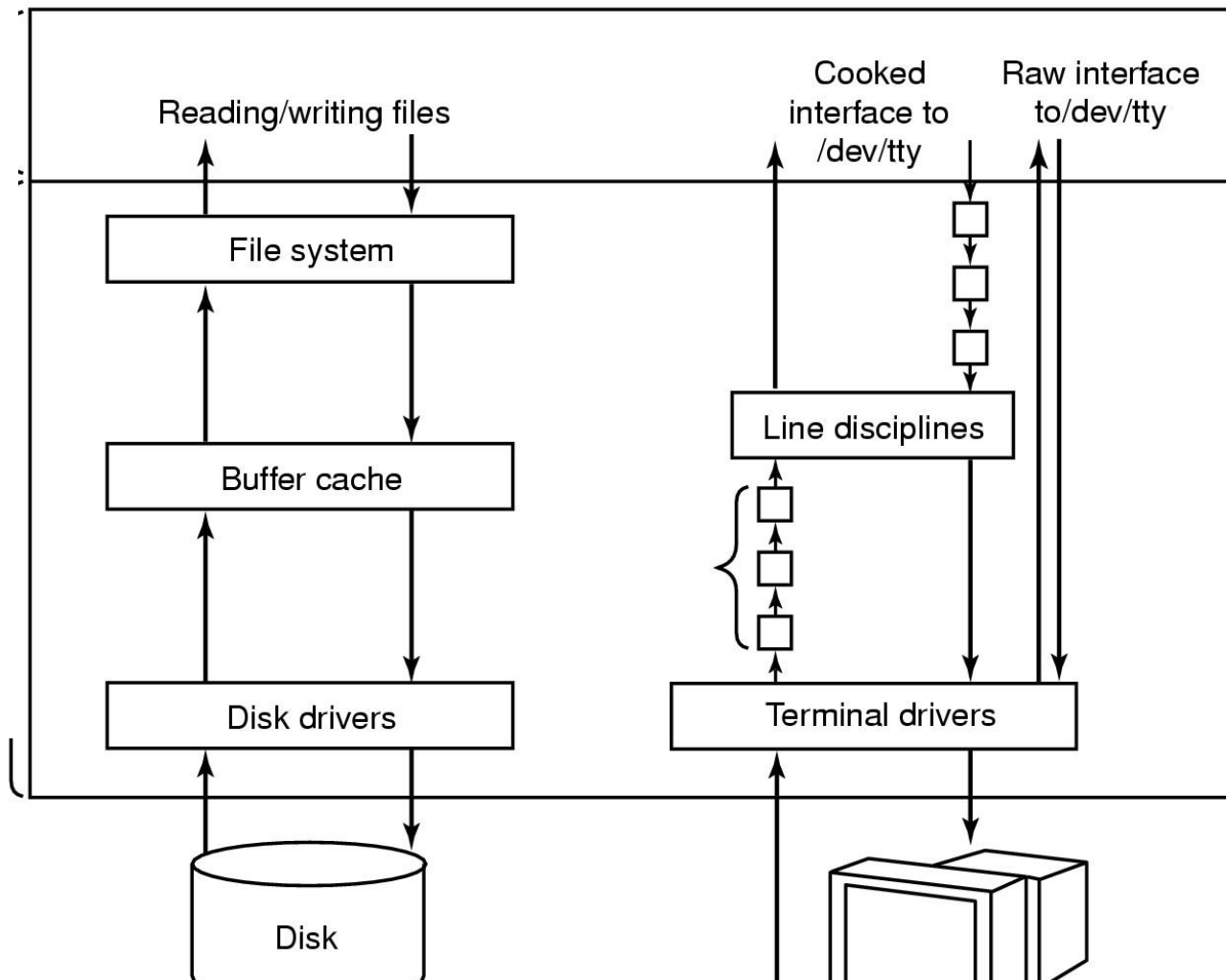
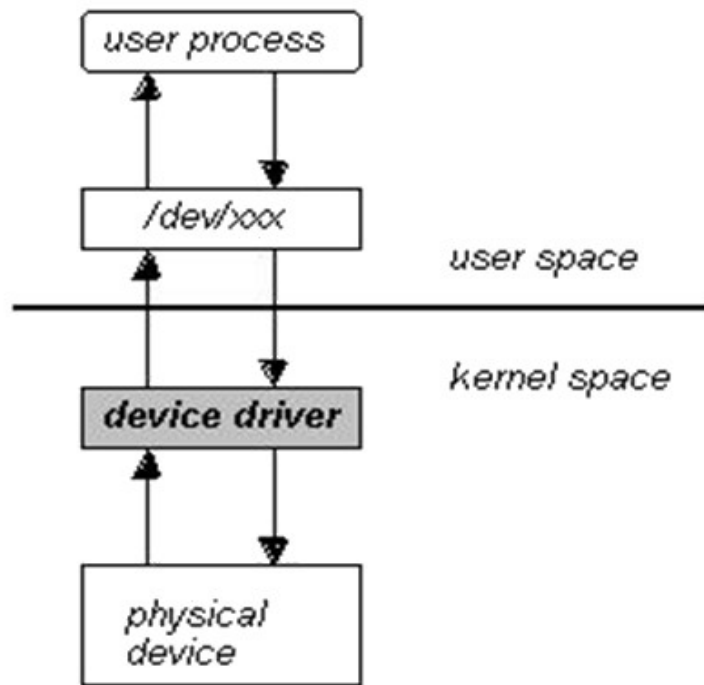


UNIX 法則：“Everything is file”

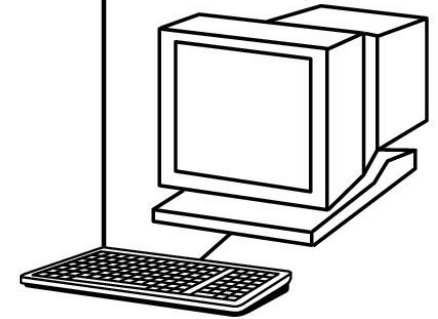
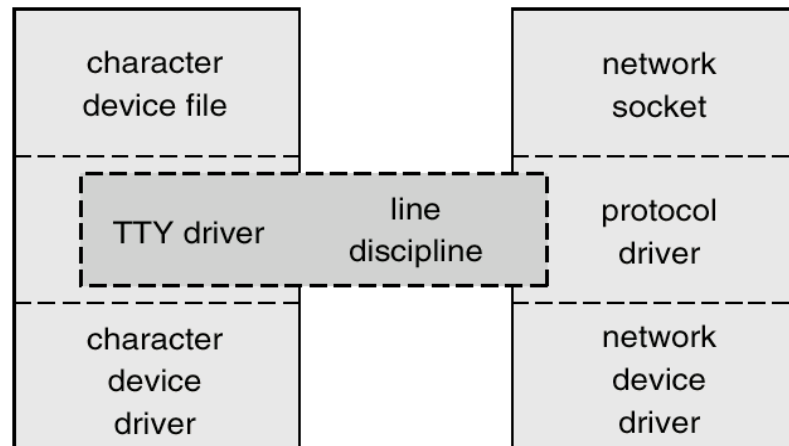
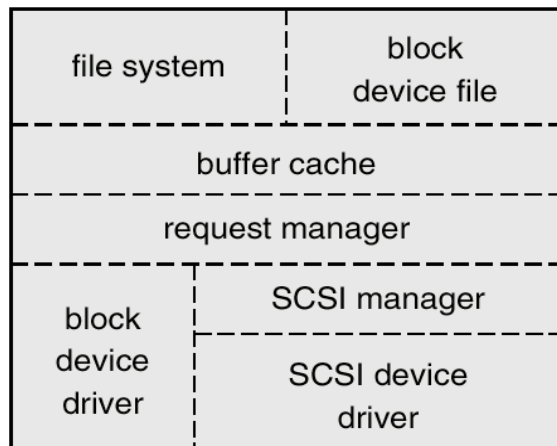
- 實體記憶體 (/dev/mem)、網路 (socket)、實體 / 虛擬裝置、系統資訊、驅動程式的控制、週邊組態、...

Linux Device Driver 的角色即是將 file operation 映射到 Device

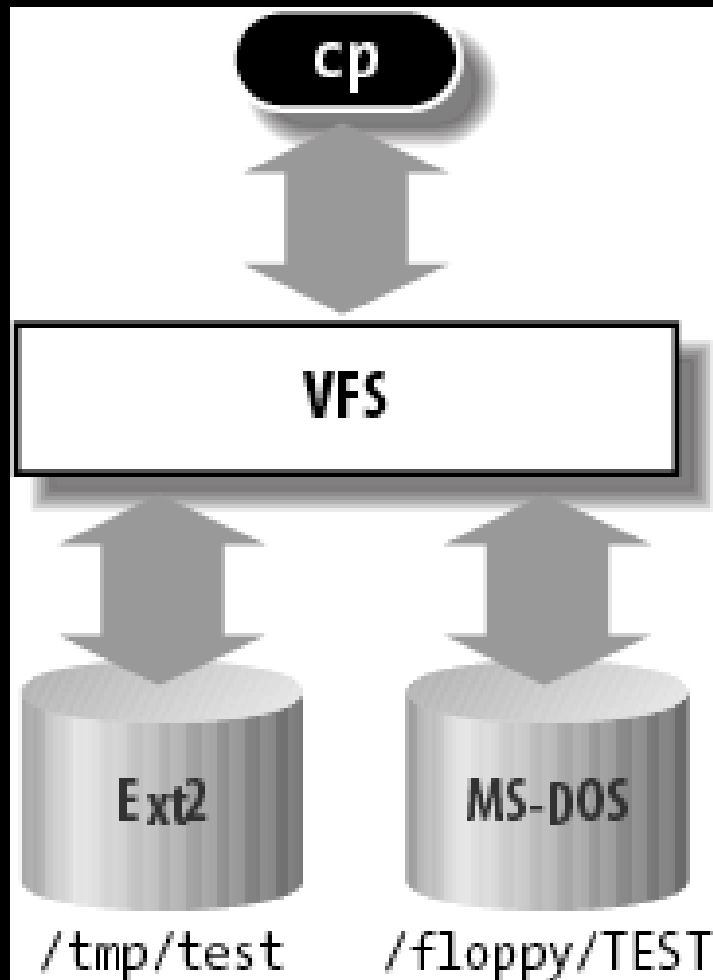
- 有明確階層概念
- 不限於 kernel-space driver
- 經典的 user-space driver 即 X11 video driver



user application



VFS role in a simple file copy operation

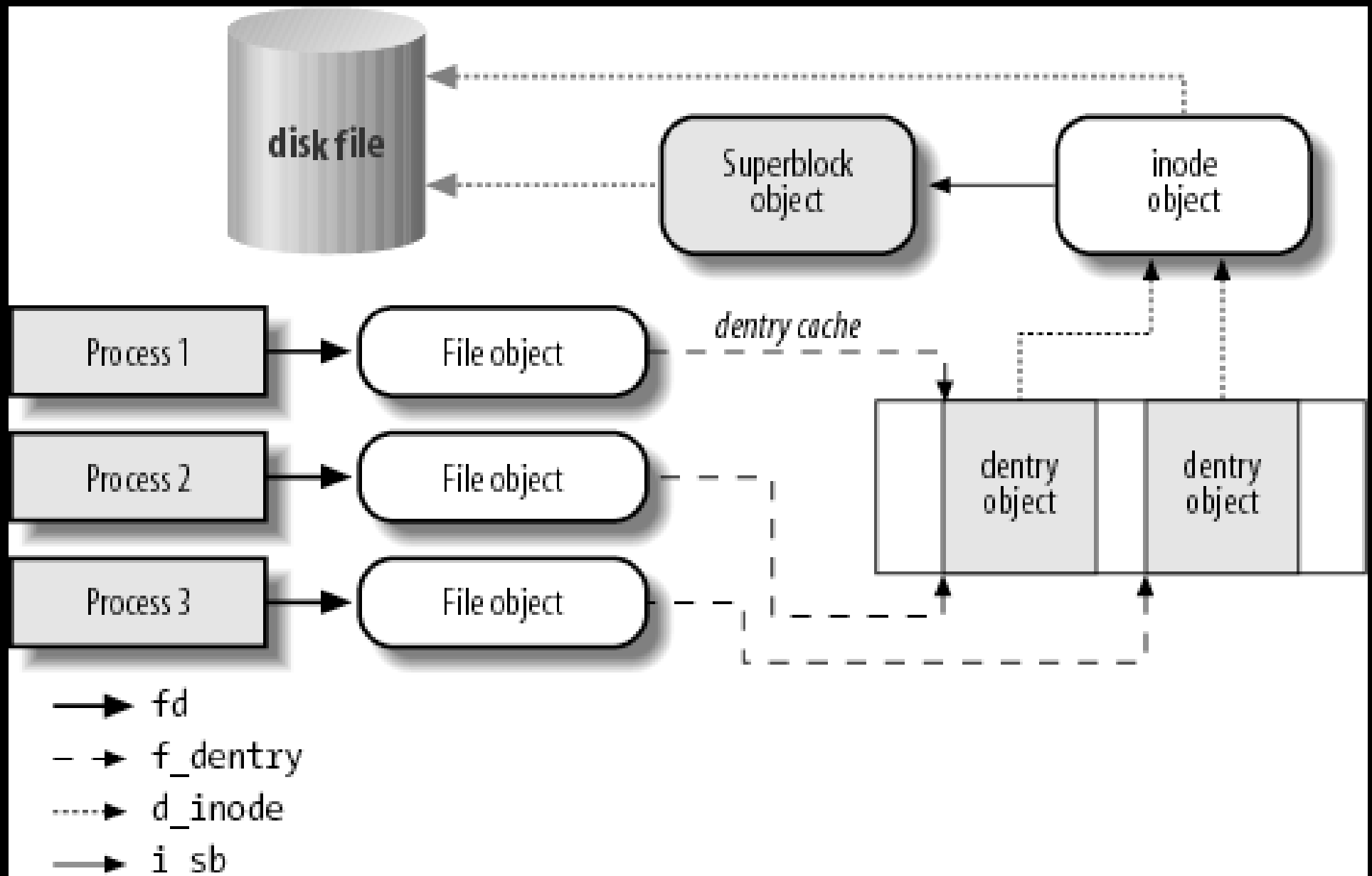


(a)

```
inf = open("/floppy/TEST", O_RDONLY, 0);  
outf = open("/tmp/test",  
            O_WRONLY|O_CREAT|O_TRUNC, 0600);  
do {  
    i = read(inf, buf, 4096);  
    write(outf, buf, i);  
} while (i);  
close(outf);  
close(inf);
```

(b)

Interaction between processes and VFS objects



Types of Device Files

Character Device Driver	Block Device Driver	Network Device Driver	USB, Firewire, SCSI,...
----------------------------	------------------------	--------------------------	----------------------------

crw--w--w-	0	root	root	5,	1	Oct	1	1998	console
crw-rw-rw-	1	root	root	1,	3	May	6	1998	null
crw-----	1	root	root	4,	0	May	6	1998	tty
crw-rw----	1	root	disk	96,	0	Dec	10	1998	pt0
crw-----	1	root	root	5,	64	May	6	1998	cua0

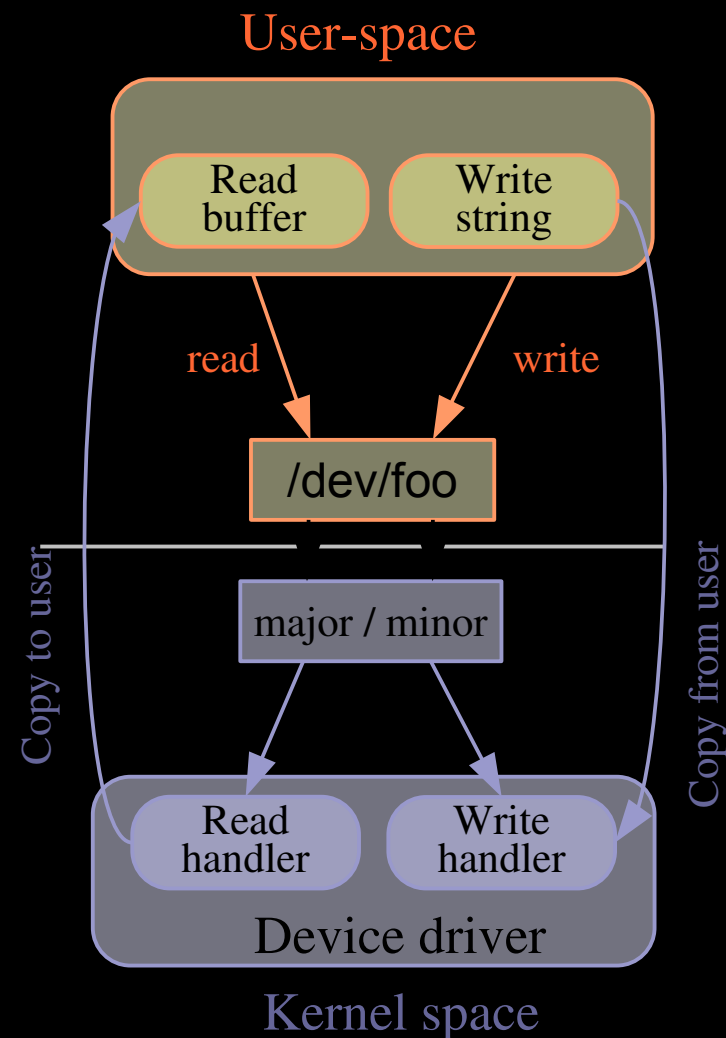
brw-----	1	root	floppy	2,	0	May	6	1998	fd0
brw-rw----	1	root	disk	3,	0	May	6	1998	hda
brw-rw----	1	root	disk	3,	1	May	6	1998	hda1
brw-rw----	1	root	disk	8,	0	May	6	1998	sda
brw-rw----	1	root	disk	8,	1	May	6	1998	sda1

Major Number and Minor Number

/dev 目錄下的檔案稱為 device files，用以辨識 / 對應於裝置

Kernel 透過 major number 來指定正確的驅動程式給 device files

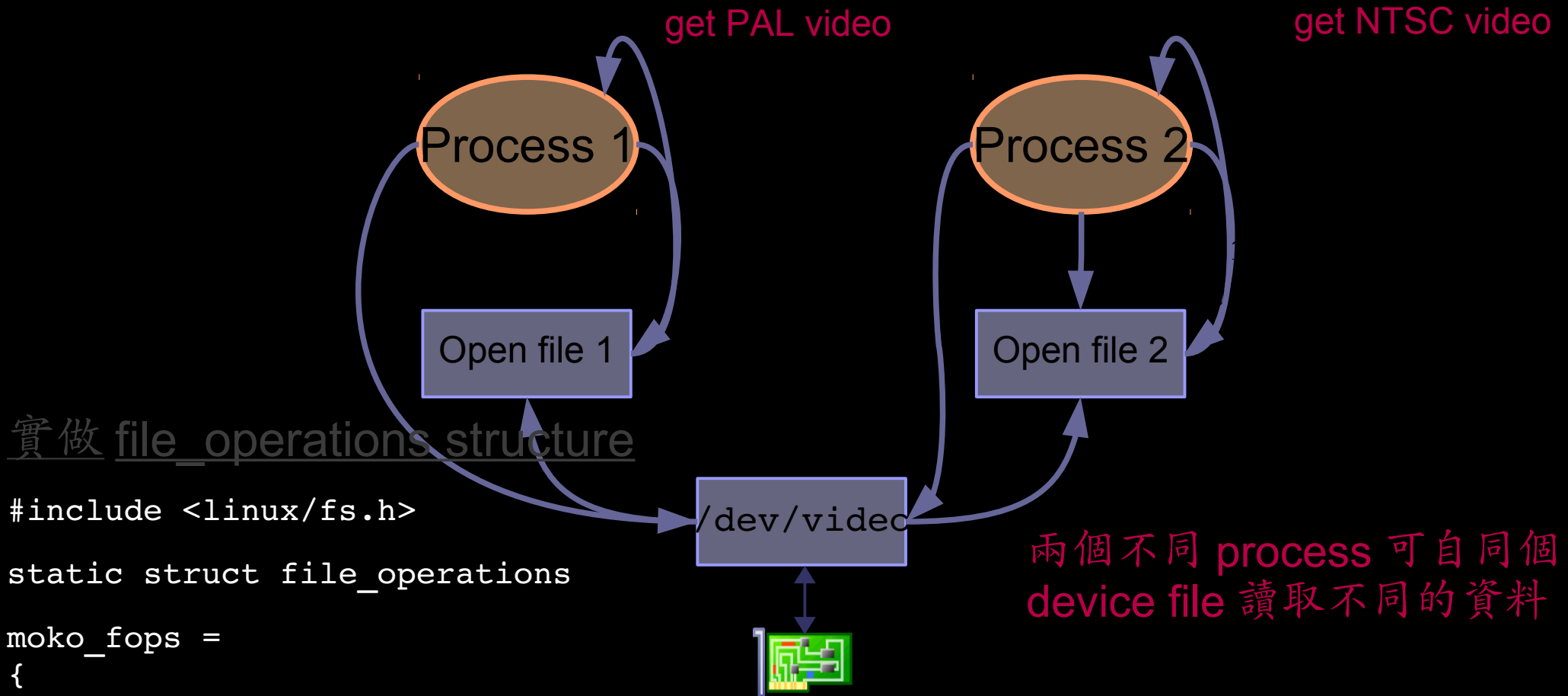
Minor number 只有驅動程式本身會使用到



```
struct    file_operations    {           /* character device drivers */
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char *,size_t, loff_t *);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    int (*ioctl) (struct inode *, struct file *,
                  unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    ...
};
```

<linux/fs.h>

File operations to each open file



```
#include <linux/fs.h>

static struct file_operations
moko_fops =
{
    .owner = THIS_MODULE,
    .read = moko_read,
    .write = moko_write,
};
```

兩個不同 process 可自同個 device file 讀取不同的資料

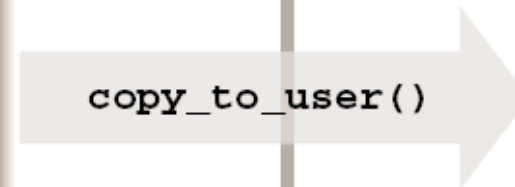
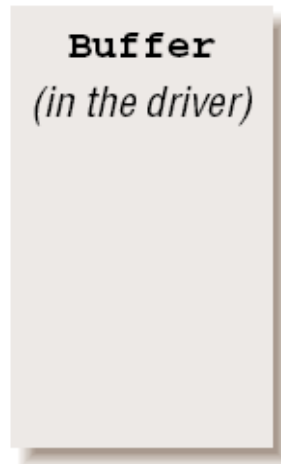
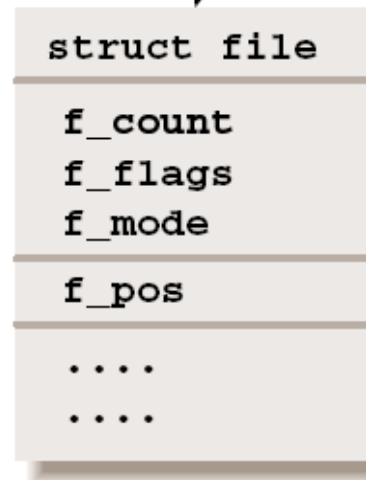
將上述 `moko_read/moko_write` 指向為具體實做，否則採用預設 handler function(如 `open, release...`)

Read method

- The *read* methods copies data to application code.

```
ssize_t read(struct file *filp, char *buff, size_t count, loff_t *offp);
```

```
ssize_t dev_read(struct file *file, char *buf, size_t count, loff_t *ppos);
```



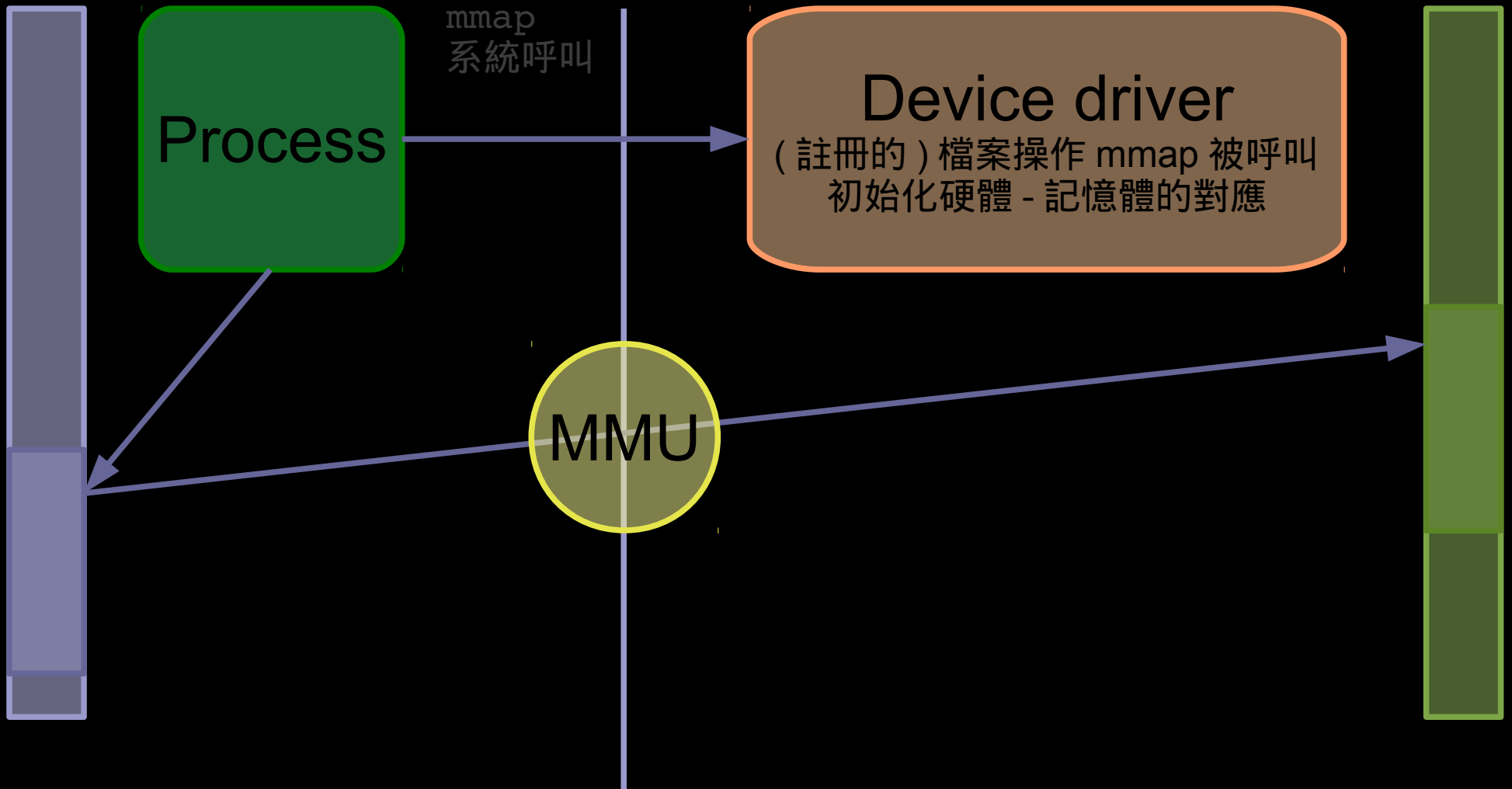
Kernel Space
(nonswappable)

User Space
(swappable)

Everything is file.

* 需將裝置對應到
Virtual Memory 才
能使用

mmap



以 X server 為例

```
# pmap `pidof X` | grep dev | head
00590000      40K r-x-- /lib/libudev.so.0.6.1
0059a000       4K r---- /lib/libudev.so.0.6.1
0059b000       4K rw--- /lib/libudev.so.0.6.1
00b02000     36K r-x-- /usr/lib/xorg/modules/input/evdev_drv.so
00b0b000       4K r---- /usr/lib/xorg/modules/input/evdev_drv.so
00b0c000       4K rw--- /usr/lib/xorg/modules/input/evdev_drv.so
a75e8000    128K rw-s- /dev/dri/card0
a7608000    128K rw-s- /dev/dri/card0
a7628000    128K rw-s- /dev/dri/card0
a7648000    128K rw-s- /dev/dri/card0
```

start address

size

Mapped file name



可對照 `/proc/`pidof X`/maps` 的內容

mmap :: user-space

取得裝置的 fd (file descriptor) 後 ...

• mmap 系統呼叫

```
void * mmap(  
    void *start,      /* Often 0, preferred starting address */  
    size_t length,    /* Length of the mapped area */  
    int prot,         /* Permissions: read, write, execute */  
    int flags,        /* Options: shared mapping, private copy */  
    int fd,           /* Open file descriptor */  
    off_t offset      /* Offset in the file */  
);
```

mmap :: kernel-space

```
int (*mmap) (  
    struct file *,          /* Open file structure */  
    struct vm_area_struct * /* Kernel VMA structure */  
);
```

實例： *devmem*

- 直接 peek (read) 或 poke (write) 某一段已對應的實體位址，預想情境 (b: byte, h: half, w: word)

```
devmem 0x000c0004 h (reading)
```

```
devmem 0x000c0008 w 0xffffffff (writing)
```

```
if((fd = open("/dev/mem", O_RDWR | O_SYNC)) == -1)
```

```
FATAL;
```

```
# devmem 0x000c0004 h
```

```
Memory mapped at address 0xb7fb5000.
```

```
Value at address 0xc0004 (0xb7fb5004): 0xE3A9
```

```
/* Map one page */
```

```
map_base = mmap(0, MAP_SIZE, PROT_READ |  
PROT_WRITE, MAP_SHARED, fd, target & ~MAP_MASK);
```

```
if(map_base == (void *) -1)
```

```
FATAL;
```

```
virt_addr = map_base + (target & MAP_MASK);
```

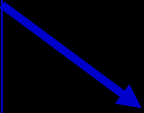
探訪 C 程式

尋訪 C 程式的資料表示
奇妙的 `pointer/macro`

Function vs. Macro

Macro
Macro
Function

```
#include <stdio.h>
void f()
{ puts("Function"); }
#define f() puts("Macro")
int main()
{
    f();
    f ();
    (f) ()
}
```



C 語言精髓：Pointer



- Pointer 也就是記憶體 (Memory) 存取的替身

*(0xF00AA00) &myVariable

- 重心：OS → Pointer → Memory

offsetof

- 依據 C99 規格

"The `offsetof()` macro returns the offset of the element name within the struct or union composite. This provides a portable method to determine the offset."

- 範例

```
typedef struct {  
    int i;  
    float f;  
    char c;  
} EEPROM;  
  
ee_rd(offsetof(EEPROM, f),  
      sizeof(float) /* f in struct EEPROM */, dest);
```

offsetof 實做

`((s *)0)->m` dereferences that pointer to point to structure member m

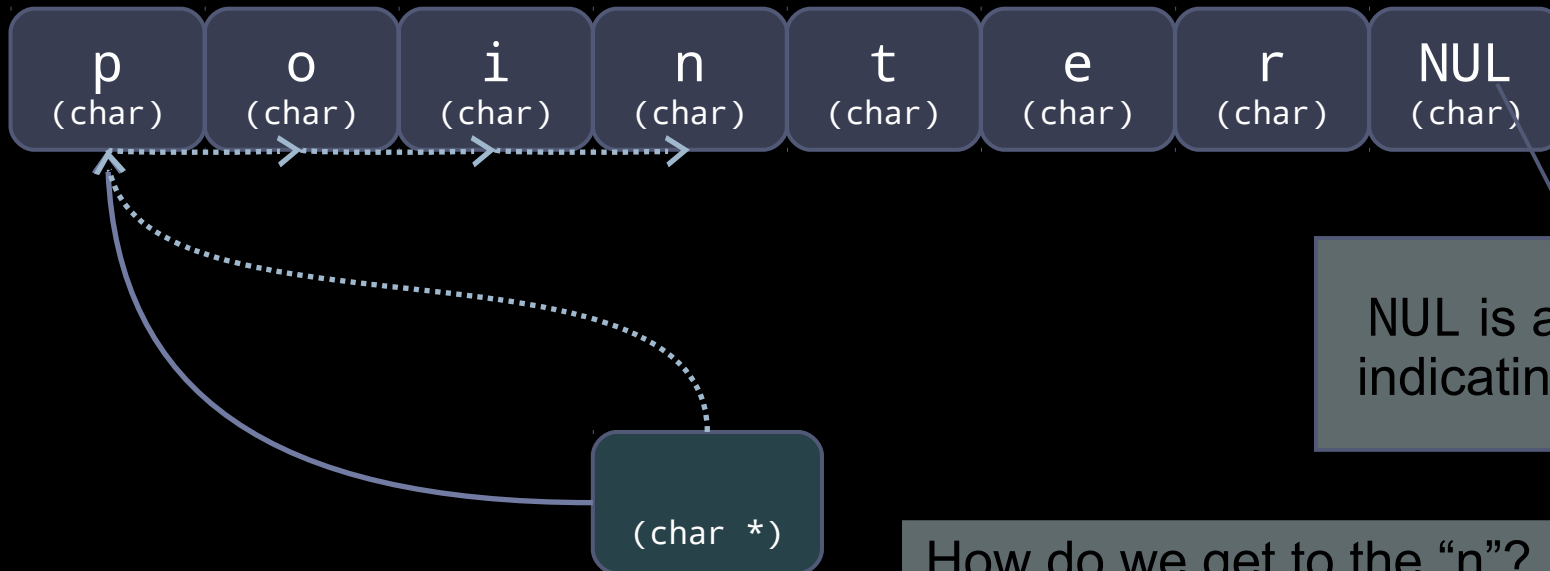
`((s *)0)` takes the integer zero and casts it as a pointer to s.

- `// Keil 8051 compiler`
`#define offsetof(s,m) \`
`(size_t)&(((s *)0)->m)`

`&(((s *)0)->m)` computes the address of m

- `// Microsoft x86 compiler (version 7)`
`#define offsetof(s,m) \`
`(size_t)(unsigned long)&(((s *)0)->m)`
- `// Diab Coldfire compiler`
`#define offsetof(s,memb) \`
`((size_t)((char *)&((s *)0)->memb-(char`
`*)0))`

String layout and access



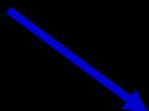
NUL is a special value indicating end-of-string

What is input?
It's a string!
It's a pointer to char!
It's an array of char!

How do we get to the "n"?
Follow the input pointer,
then hop 3 to the right
`*(input + 3)`
- or -
`input[3]`

char* vs. char[]

```
gcc -O3 -S const_char.c
```



```
static const char *str1 = "azerty";  
static const char str2[] = "azerty";  
void f(const char *x);  
void try(void) {  
    f(str1);  
    f(str2);  
}
```

char* vs. char[]

```
gcc -O3 -S const_char.c
```

```
#include <stdio.h>
```

```
static const char *str1 = "azerty";
```

```
static const char str2() = "azerty";
```

```
void f(const char *x);
```

```
void try(void)
```

```
{
```

```
    f(str1);
```

```
    f(str2);
```

```
}
```

```
movl $str2, (%esp)
call f
```

```
movl str1, %eax
pushl %eax
call f
```

```
.LC0:
```

```
    .string "azerty"
```

```
    .data
```

```
    .align 4
```

```
    .type str1, @object
```

```
    .size str1, 4
```

```
str1:
```

```
    .long .LC0
```

```
str2:
```

```
    .string "azerty"
```

- C 語言的根源與強烈的 UNIX 設計哲學
- 以 Linux Kernel 為探討對象，奠定理論基礎
- C 語言的手法
 - 與系統緊密的關聯
 - 模組化、清晰的介面
 - 物件導向設計



Linux 核心充斥大量的物件 導向設計

kobject: ADT (Abstract Data Typing)
device driver: Inheritance/polymorphism
hotplug(), probe(): dynamic binding

- Buses : 處理器與一個或多個裝置間的通道
- Devices 與對應的 device drivers

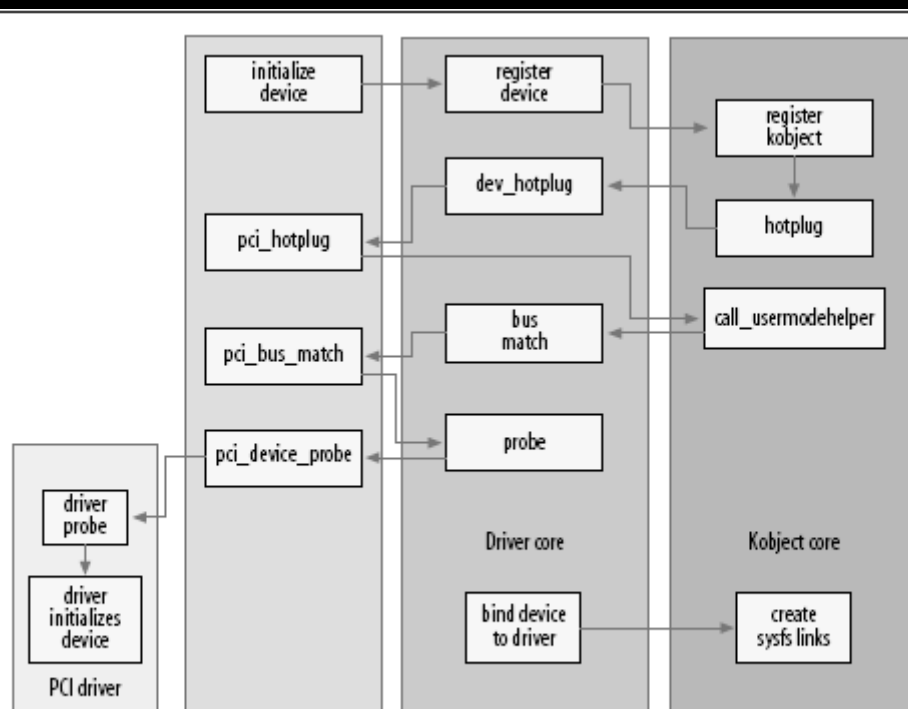
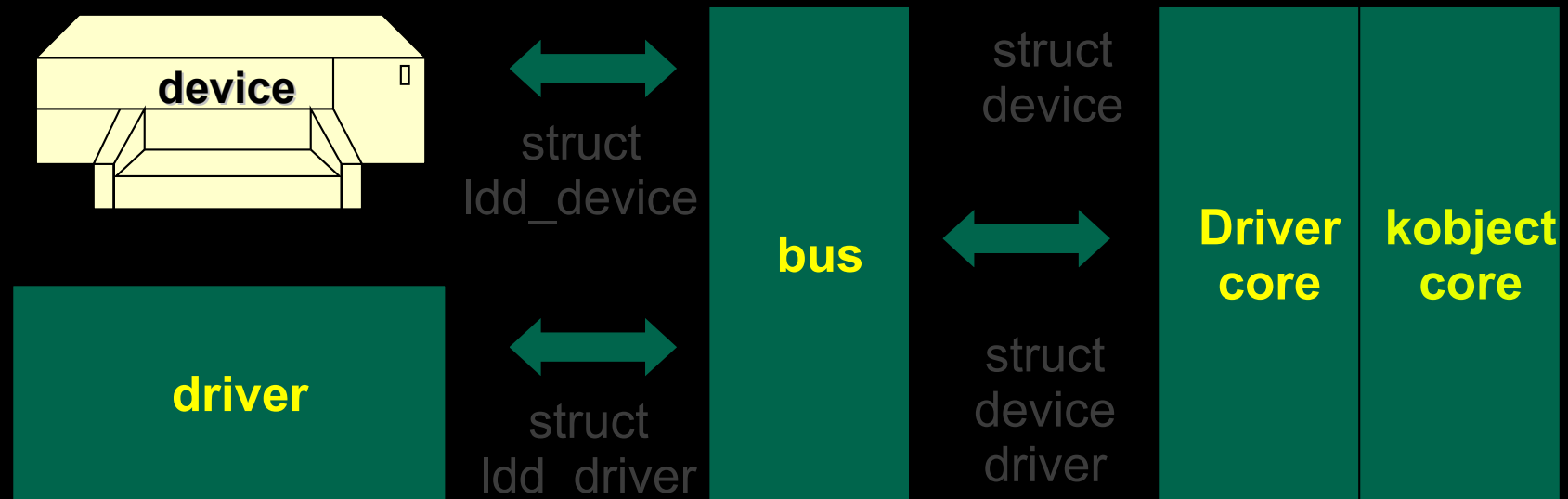


Figure 14-3. Device-creation process

Linux Device Model Tree

- /proc, /dev, /sysfs

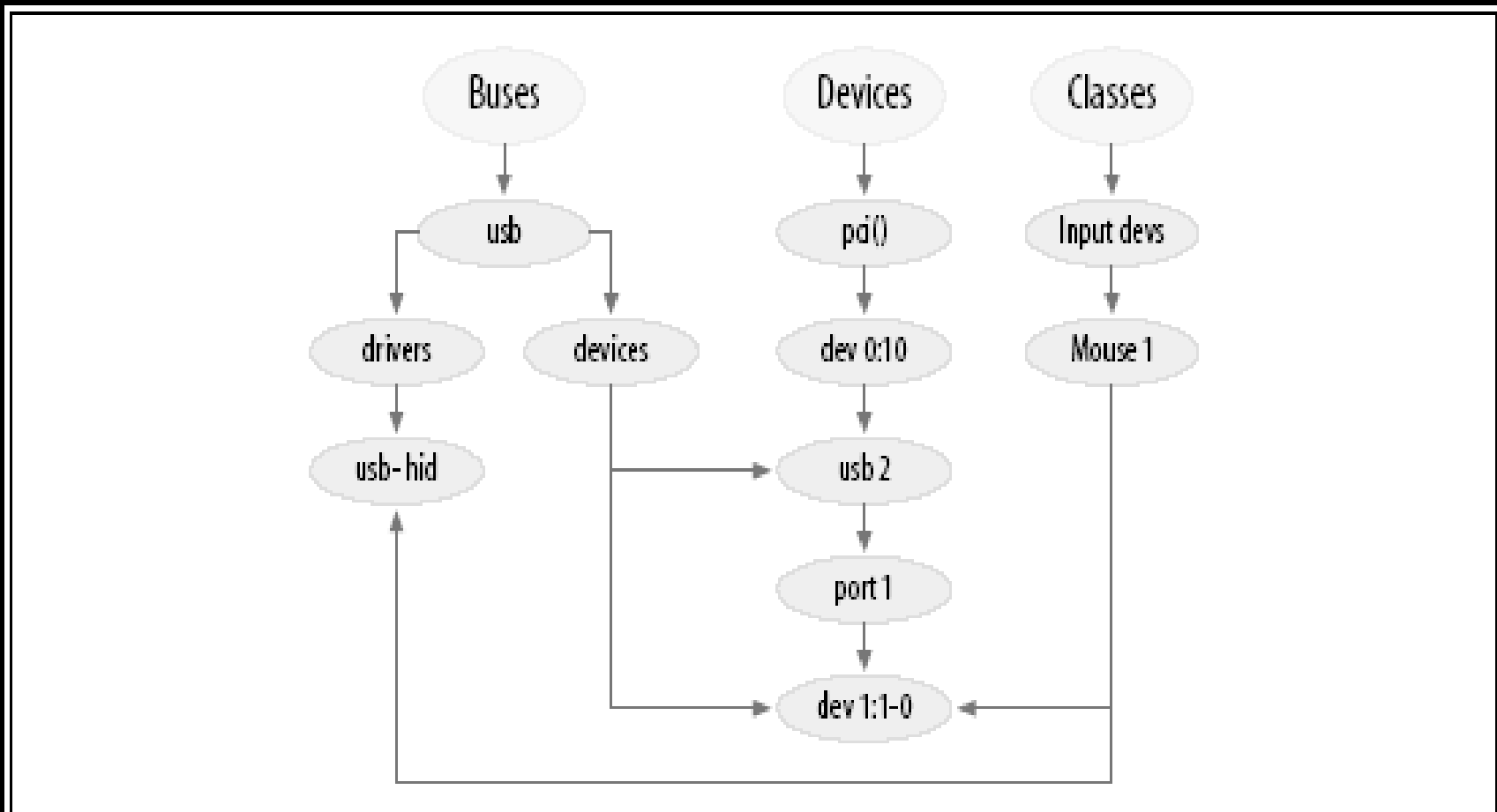


Figure 14-1. A small piece of the device model

Linux 核心的物件導向設計

- Abstract Data typing

- Information hiding

- Encapsulation

- Inheritance

- Derive more specialized classes from a common class

- Polymorphism

- Refers to the object's ability to respond in an individual manner to the same message

- Dynamic binding

- Refers to the mechanism that resolves a virtual function call at runtime

- You can derive modified action that override the old one even after the code is compiled .

Kobject, Kset

Bus, driver, device,
partition...

hotplug(), match(),
probe(), kobj_type

Kobject 的繼承關係

- parent pointer 與 ksets
- “parent” points to another kobject, representing the next level up
- “kset” is a collection of kobjects
- kset are always represented in sysfs
- Every kobject that is a member of a kset is represented in sysfs

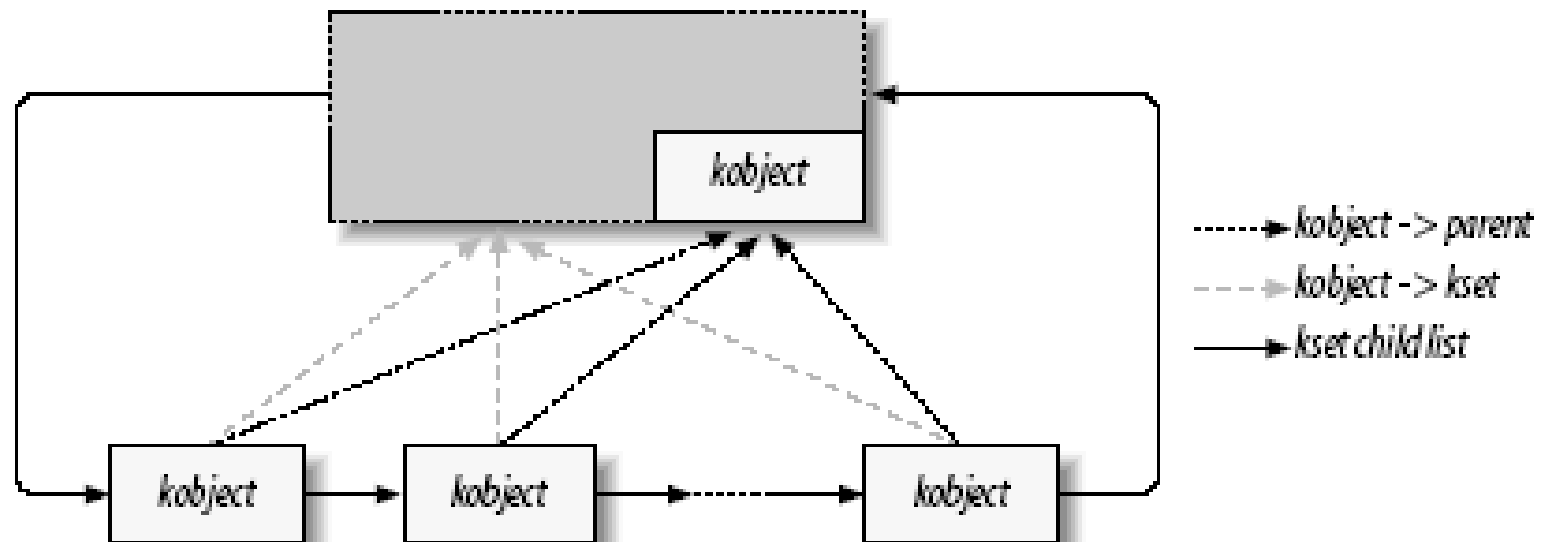
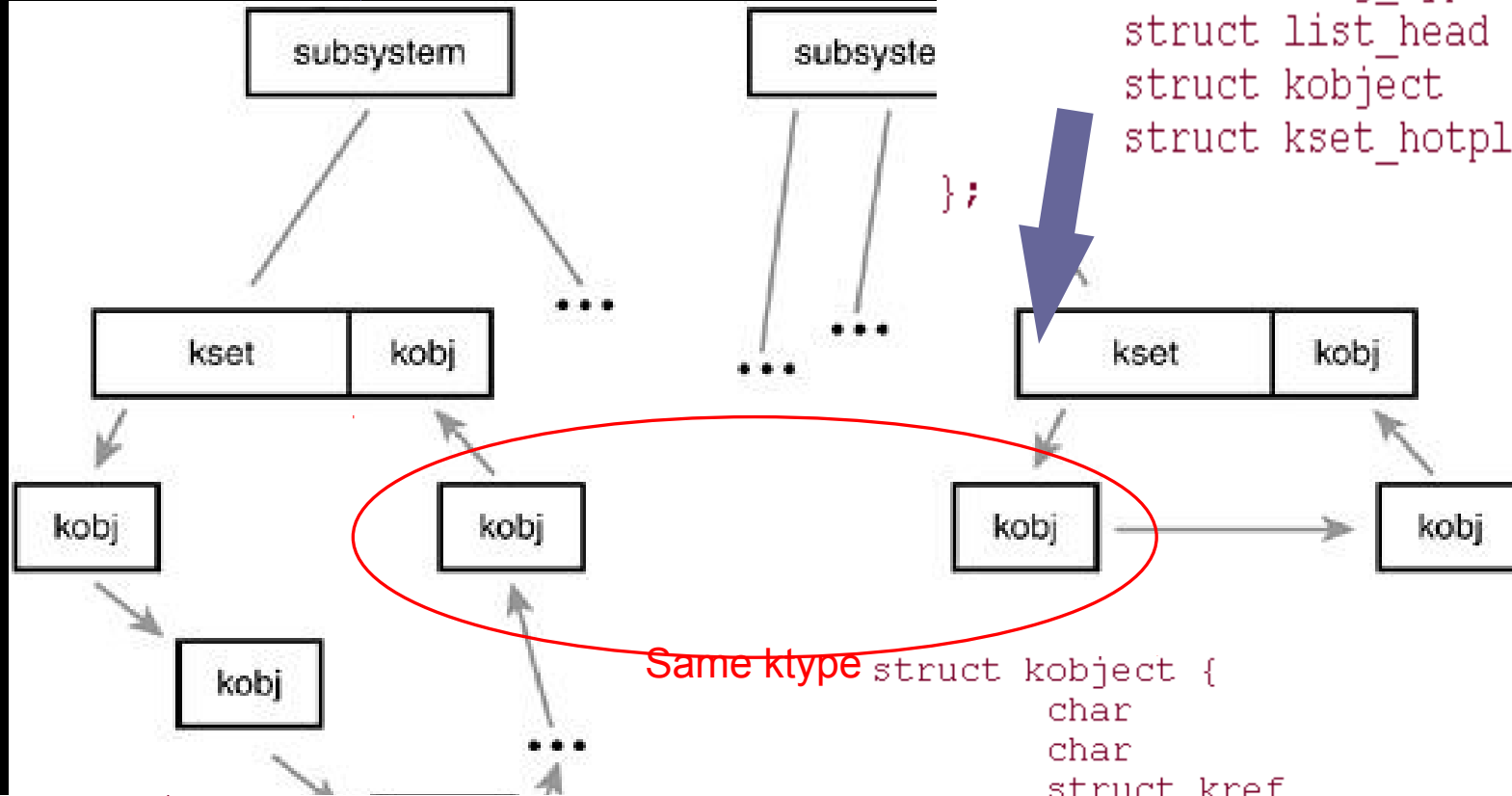


Figure 14-2. A simple kset hierarchy

```
struct subsystem {
    struct kset      kset;
    struct rw_semaphore rwsem;
};
```



```
struct kset {
    struct subsystem      *subsys;
    struct kobj_type      *ktype;
    struct list_head      list;
    struct kobject        kobj;
    struct kset_hotplug_ops *hotplug_ops;
};
```



Same ktype

```
struct kobject {
    char          *k_name;
    char          name[KOBJ_NAME_LEN];
    struct kref    kref;
    struct list_head entry;
    struct kobject *parent;
    struct kset    *kset;
    struct kobj_type *ktype;
    struct dentry  *dentry;
};
```

```
struct kobj_type {
    void (*release)(struct kobject *);
    struct sysfs_ops *sysfs_ops;
    struct attribute **default_attrs;
};
```