





Linux 核心設計/實作 (Linux Kernel Internals)

- Instructor: **Jim Huang** (黃敬群) [<jserv.tw@gmail.com>](mailto:jserv.tw@gmail.com)
 - **Facebook 粉絲專頁** (不要擔心提了笨問題，這專門用來和學生互動，可預約一對一討論)
- 訂閱 **Google Calendar**
- 討論區: <https://www.facebook.com/groups/system.software2022/>
- 課程信箱: [<embedded.master2015@gmail.com>](mailto:embedded.master2015@gmail.com) (授課教師和助教會以該信箱發公告), [往年課程進度](#)
- **Linux 核心設計 (線上講座)**
- 注意: 下方課程進度表標註有  的項目，表示內附錄影的教材
- 注意: 新開的「Linux 核心實作」課程內容幾乎與「Linux 核心設計」一致，採「全線上」方式進行

Linux 核心設計/實作 (Spring 2022) 課程進度表暨線上資源

- 第 1 週 (Feb 14, 15, 17): 誠實面對自己
 - **課程簡介和注意須知 / 課程簡介解說錄影** 
 - 每週均安排隨堂測驗，採計其中最高分的 8 次
 - 學期評分方式: 隨堂測驗 (20%) + 個人作業+報告及專題 (30%) + 自我評分 (50%)
 - 歷屆修課學生心得: **張家榮**, **陳品睿**, **蕭奕凱**, **方鈺學**
 - 分組報告示範: **ARM-Linux**, **Xvisor**
 - **GNU/Linux 開發工具共筆** : 務必 自主 學習 Linux 操作, Git, HackMD, LaTeX 語法 (特別是數學式), GNU make, perf, gnuplot
 - 確認 Ubuntu Linux 20.04-LTS (或更新的版本) 已順利安裝到你的電腦中
 - 透過 **Computer Systems: A Programmer's Perspective 學習系統軟體** : 本課程指定的教科書 (請及早購買: [天瓏書店](#))
 - **軟體缺失導致的危害**
 - 1970 年代推出的首款廣體民航客機波音 747 軟體由大約 40 萬行程式碼構成，而 2011 年引進的波音 787 的軟體規模則是波音 747 的 16 倍，約 650 萬行程式碼。換言之，你我的性命緊緊於一系列極為複

雜的軟體系統之中，能不花點時間了解嗎？

- 軟體開發的安全性設計和測試驗證應獲得更高的重視

◦ 解讀計算機編碼

- 人們對數學的加減運算可輕易在腦中辨識符號並理解其結果，但電腦做任何事都受限於實體資料儲存及操作方式，換言之，電腦硬體實際只認得 0 和 1，卻不知道符號 + 和 - 在數學及應用場域的意義，於是工程人員引入「補數」以表達人們認知上的正負數
- 您有沒有想過，為何「二補數」(2's complement) 被電腦廣泛採用呢？背後的设计考量是什麼？本文嘗試從數學觀點去解讀編碼背後的原理

◦ 你所不知道的 C 語言：指標篇 *

◦ linked list 和非連續記憶體操作 *

- 安排 linked list 作為第一份作業及隨堂測驗的考量點：
 - 檢驗學員對於 C 語言指標操作的熟悉程度 (附帶思考：對於 Java 程式語言來說，該如何實作 linked list 呢？)
 - linked list 本質上就是對非連續記憶體的操作，乍看僅是一種單純的資料結構，但對應的演算法變化多端，像是「如何偵測 linked list 是否存在環狀結構？」和「如何對 linked list 排序並確保空間複雜度為 $O(1)$ 呢？」
 - linked list 的操作，例如走訪 (traverse) 所有節點，反映出 Locality of reference (cache 用語) 的表現和記憶體階層架構 (memory hierarchy) 高度相關，學員很容易從實驗得知系統的行為，從而思考其衝擊和效能改進方案
 - 無論是作業系統核心、C 語言函式庫內部、應用程式框架，到應用程式，都不難見到 linked list 的身影，包含多種針對效能和安全議題所做的 linked list 變形，又還要考慮到應用程式的泛用性 (generic programming)，是很好的進階題材

■ 題目 1 + 分析 *

■ 題目 2 / 參考題解1, 參考題解2

■ 題目 3 / 參考題解

■ 題目 4 / 參考題解

◦ 作業: 截止繳交日: Mar 1, 2022

■ lab0 *

■ quiz1

◦ 第 1 週隨堂測驗: 題目 (內含作答表單)

◦ 課堂問答簡記

• 第 2 週 (Feb 21, 22, 24): C 語言程式設計

- 課程基本資料表單: 務必填寫，以接收課程資訊

- **Linux: 作業系統術語及概念** *
- **系統軟體開發思維**
- **C 語言: 數值系統** *
 - 儘管數值系統並非 C 語言所特有，但在 Linux 核心大量存在 u8/u16/u32/u64 這樣透過 typedef 所定義的型態，伴隨著各式 alignment 存取，若學員對數值系統的認知不夠充分，可能立即就被阻擋在探索 Linux 核心之外——畢竟你完全搞不清楚，為何在 Linux 核心存取特定資料需要繞一大圈。
- **C 語言: Bitwise 操作** *
 - Linux 核心原始程式碼存在大量 bit(-wise) operations (簡稱 bitops)，頗多乍看像是魔法的 C 程式碼就是 bitops 的組合
 - **類神經網路的 ReLU 及其常數時間複雜度實作**
 - **題目 / 參考題解**
- **為什麼要深入學習 C 語言？** *
 - C 語言發明者 Dennis M. Ritchie 說：「C 很彆扭又缺陷重重，卻異常成功。固然有歷史的巧合推波助瀾，可也的確是因為它能滿足於系統軟體實作的程式語言期待：既有相當的效率來取代組合語言，又可充分達到抽象且流暢，能用於描述在多樣環境的演算法。」
 - Linux 核心作為世界上最成功的開放原始碼計畫，也是 C 語言在工程領域的瑰寶，裡頭充斥各式「藝術」，往往會嚇到初次接觸的人們，但總是能夠用 C 語言標準和開發工具提供的擴展 (主要來自 gcc 的 GNU extensions) 來解釋。
- **基於 C 語言標準研究與系統程式安全議題**
 - 藉由研讀漏洞程式碼及 C 語言標準，討論系統程式的安全議題
 - 透過除錯器追蹤程式碼實際運行的狀況，了解其運作原理;
 - 取材自 dangling pointer, CWE-416 Use After Free, CVE-2017-16943 以及 integer overflow 的議題;
- **C 語言：記憶體管理、對齊及硬體特性** *
 - 搭配閱讀: **The Lost Art of Structure Packing**
 - 從虛擬記憶體談起，歸納出現代銀行和虛擬記憶體兩者高度相似: malloc 給出 valid pointer 不要太高興，等你要開始用的時候搞不好作業系統給個 OOM——簡單來說就是一張支票，能不能拿來開等到兌現才知道。
 - 探討 heap (動態配置產生，系統會存放在另外一塊空間)、data alignment，和 malloc 實作機制等議題。這些都是理解 Linux 核心運作的關鍵概念。
- **C 語言: bit-field**
 - bit field 是 C 語言一個很被忽略的特徵，但在 Linux 和 gcc 這類系統軟體很常出現，不僅是精準規範每個 bit 的作用，甚至用來「擴充」C 語言
- **參考題目** * / **參考題解 1, 參考題解 2, 參考題解 3**

- 作業: 截止繳交日 Mar 13, 2022
 - [quiz2](#)
- 第 2 週隨堂測驗: [題目](#) (內含作答表單)
- [課堂問答簡記](#)
- 第 3 週 (Feb 28, Mar 1, 3): 並行和 C 語言程式設計
 - [Linux: 發展動態回顧](#) *
 - [並行和多執行緒程式設計](#) *: 應涵蓋 Part 1 到 Part 4
 - [C 語言: 函式呼叫](#) *
 - 著重在計算機架構對應的支援和行為分析
 - [C 語言: 遞迴呼叫](#) *
 - 或許跟你想像中不同, Linux 核心的原始程式碼裡頭也用到遞迴函式呼叫, 特別在較複雜的實作, 例如檔案系統, 善用遞迴可大幅縮減程式碼, 但這也導致追蹤程式運作的難度大增
 - [C 語言: 前置處理器應用](#) *
 - C 語言之所以不需要時常發佈新的語言特徵又可以保持活力, 前置處理器 (preprocessor) 是很重要的因素, 有心者可逕行「擴充」C 語言
 - [C 語言: goto 和流程控制](#) *
 - goto 在 C 語言被某些人看做是妖魔般的存在, 不過實在不用這樣看待, 至少在 Linux 核心原始程式碼中, goto 是大量存在 (跟你想像中不同吧)。有時不用 goto 會寫出更可怕的程式碼
 - [C 語言程式設計技巧](#) *
 - 作業: 截止繳交日: Mar 21, 2022
 - [fibdrv, quiz3](#)
 - Week3 隨堂測驗: [題目](#) (內含作答表單)
- 第 4 週 (Mar 7, 8, 10): 浮點數 + 編譯器和連結器
 - 公告: 請填寫 [Google 表單](#), 以利後續追蹤
 - [追求神乎其技的程式設計之道](#)
 - [CS:APP 第 2 章重點提示和練習](#) *
 - [浮點數運算](#) *: 工程領域往往是一系列的取捨結果, 浮點數更是如此, 在軟體開發有太多失誤案例源自工程人員對浮點數運算的掌握不足, 本議程希望藉由探討真實世界的血淋淋案例, 帶著學員思考 IEEE 754 規格和相關軟硬體考量點, 最後也會探討在深度學習領域為了改善資料處理效率, 而引入的 [BFloat16](#) 這樣的新標準
 - 核心開發者當然要熟悉編譯器行為
 - [Linus Torvalds 教你分析 gcc 行為](#)
 - [Pointers are more abstract than you might expect in C](#)

```
int main(void) {
    int a, b;
    int *p = &a;
    int *q = &b + 1;
    return p == q;
}
```

這段程式碼在 gcc 開啟/關閉編譯器最佳化，會有截然不同的返回值 (`-o0` 編譯會得到 `1`; `-O1` 或更高等級編譯會得到 `0`)，為何？C 語言標準並未定義像 `a`, `b` 在記憶體中的配置，因此比較其記憶體位置屬於未定義行為，當最佳化未開啟時，GCC 不檢查程式行為是否符合 C 規範，才會去比較其內容，而在最佳化啟動後，推論這樣的未定義行為而略去記憶體操作。

■ 藝術與核心

- C 編譯器原理和案例分析 *
- C 語言: 未定義行為 *: C 語言最初為了開發 UNIX 和系統軟體而生，本質是低階的程式語言，在語言規範層級存在 undefined behavior，可允許編譯器引入更多最佳化
- C 語言: 編譯器和最佳化原理 *
- C 語言: 動態連結器 *
- C 語言: 連結器和執行檔資訊 *
- C 語言: 執行階段程式庫 (CRT) *
- 作業: 截止繳交日: Mar 28, 2022
 - review, quiz4
- Week4 隨堂測驗: 題目 (內含作答表單)
- 課堂問答簡記
- 第 5 週 (Mar 14, 15, 17): Linux Process
 - 從 $\sqrt{2}$ 的運算談浮點數
 - Linux 核心模組運作原理
 - Linux: 不僅是個執行單元的 Process *: Linux 核心對於 UNIX Process 的實作相當複雜，不僅蘊含歷史意義 (幾乎每個欄位都值得講古)，更是反映出資訊科技產業的變遷，核心程式碼的 `task_struct` 結構體更是一絕，廣泛涵蓋 process 狀態、處理器、檔案系統、signal 處理、底層追蹤機制等等資訊，更甚者，還很曖昧地保存著 thread 的必要欄位，好似這兩者天生就脫不了干係
 - 探討 Linux 核心設計的特有思維，像是如何透過 LWP 和 NPTL 實作執行緒，又如何透過行程建立記憶體管理的一種抽象層，再者回顧行程間的 context switch 及排程機制，搭配 signal 處理
 - UNIX 作業系統 fork/exec 系統呼叫的前世今生

- **Linux: 不只挑選任務的排程器 ***: 排程器 (scheduler) 是任何一個多工作業系統核心都具備的機制，但彼此落差極大，考量點不僅是演算法，還有當應用規模提昇時 (所謂的 scalability) 和涉及即時處理之際，會招致不可預知的狀況 (non-determinism)，不僅即時系統在意，任何建構在 Linux 核心之上的大型服務都會深受衝擊。是此，Linux 核心的排程器經歷多次變革，需要留意的是，排程的難度不在於挑選下一個可執行的行程 (process)，而是讓執行完的行程得以安插到合適的位置，使得 runqueue 依然依據符合預期的順序。
- **作業**: 截止繳交 Apr 25
- **Week5 隨堂測驗: 題目** (內含作答表單)
- **課堂問答簡記**
- **第 6 週 (Mar 21, 22, 24): Linux Process + Scheduler**
 - **公告**
 - 本週開放讓學員 (選課的學生 + 完成前二次作業過半要求的旁聽者) 跟授課教師預約一對一線上討論，請參照**課程行事曆**裡頭標注 “Office hour” 的時段，發訊息到 **Facebook 粉絲專頁**，簡述你的學習狀況並選定偏好的時段 (建議是 30 分鐘)
 - 選課修課程的學員在本學期至少要安排一次一對一討論，否則授課教師難以評估學習狀況，從而會影響評分，請重視自己的權益。
 - **建構 User-Mode Linux 的實驗環境 ***
 - **Linux: 不只挑選任務的排程器 ***: 排程器 (scheduler) 是任何一個多工作業系統核心都具備的機制，但彼此落差極大，考量點不僅是演算法，還有當應用規模提昇時 (所謂的 scalability) 和涉及即時處理之際，會招致不可預知的狀況 (non-determinism)，不僅即時系統在意，任何建構在 Linux 核心之上的大型服務都會深受衝擊。是此，Linux 核心的排程器經歷多次變革，需要留意的是，排程的難度不在於挑選下一個可執行的行程 (process)，而是讓執行完的行程得以安插到合適的位置，使得 runqueue 依然依據符合預期的順序。
 - **作業**: 截止繳交 Apr 25
 - **Week6 隨堂測驗: 題目** (內含作答表單)
 - **課堂問答簡記**
- **第 7 週 (Mar 28, 29, 31): 並行和多執行緒**
 - **公告**
 - 學員應及早跟授課教師預約一對一線上討論，請參照**課程行事曆**裡頭標注 “Office hour” 的時段，發訊息到 **Facebook 粉絲專頁**，簡述你的學習狀況並選定偏好的時段 (建議是 30 分鐘)
 - 除了進行師生互動，一對一線上討論時段將重新調整學員的作業內容及規劃，請選課學員務必留意
 - 依據**成功大學行事曆**，4 月 4 日和 5 日放假，但本課程仍安排學員在家進行線上測驗，請留意信件通知
 - 本週不安排隨堂測驗
 - **CPU Schedulers for Linux Kernel**
 - **並行和多執行緒程式設計 ***: 應涵蓋 Part 5 到 Part 7

- CS:APP 第 12 章
 - [Concurrency](#) / 錄影 *
 - [Synchronization: Basic](#) / 錄影 *
 - [Synchronization: Advanced](#) / 錄影 *
 - [Thread-Level Parallelism](#) / 錄影 *
- [Linux: 淺談同步機制](#) *
- [課堂問答簡記](#)
- 第 8 週 (Apr 4, 5, 7): 測驗 + 作業檢討
 - 公告:
 - 新的作業已指派: [Homework5](#), 作業要求是重作第 5, 6, 8 週測驗題及其延伸問題, 截止繳交日為 4 月 25 日
 - [Week8 隨堂測驗: 題目](#) (內含作答表單)
 - [作業](#): 截止繳交 Apr 25
 - [課堂問答簡記](#)
- 第 9 週 (Apr 11, 12, 14): 伺服器開發與 Linux 核心對應的系統呼叫
 - [Linux 核心設計: 針對事件驅動的 I/O 模型演化](#) *
 - [精通數位邏輯對 coding 有什麼幫助?](#)
 - [Linux: 透過 eBPF 觀察作業系統行為](#) *: 動態追蹤技術 (dynamic tracing) 是現代軟體的進階除錯和追蹤機制, 讓工程師以非常低的成本, 在非常短的時間內, 克服一些不是顯而易見的問題。它興起和繁榮的一個大背景是, 我們正處在一個快速增長的網路互連異質運算環境, 工程人員面臨著兩大方面的挑戰:
 - 規模: 無論是使用者規模還是機房的規模、機器的數量都處於快速增長的時代;
 - 複雜度: 業務邏輯越來越複雜, 運作的軟體也變得越來越複雜, 我們知道它會分成很多很多層次, 包括作業系統核心和其上各種系統軟體, 像資料庫和網頁伺服器, 再往上有腳本語言或者其他高階語言的虛擬機器或執行環境, 更上面是應用層面的各種業務邏輯的抽象層次和很多複雜的程式邏輯。
 - [Week9 隨堂測驗: 題目](#) (內含作答表單)
 - [課堂問答簡記](#)
- 第 10 週 (Apr 18, 19, 21): 現代微處理器
 - Twitter 上面的笑話: index 的複數寫作 indices, complex 的複數寫作 complices, 那 mutex 的複數是什麼? 答 "deadlock" – [出處](#)
 - [現代處理器設計: 原理和關鍵特徵](#) *
 - [Linux: 中斷處理和現代架構考量](#) *
 - [Linux: 多核處理器和 spinlock](#) *
 - [CPU caches](#) by Ulrich Drepper

- 進行中的繁體中文翻譯: 《每位程式開發者都該知道的記憶體知識》
 - 本文解釋用於現代電腦硬體的記憶體子系統的結構、闡述 CPU 快取發展的考量、它們如何運作, 以及程式該如何針對記憶體操作調整, 從而達到最佳的效能。
- CS:APP 第 6 章重點提示 *
- CS:APP 第 9 章重點提示 *
- Week10 隨堂測驗: 題目 (內含作答表單)
- 作業: 截止繳交 May 10
 - ktcp, sehttpd
- 課堂問答簡記
- 第 11 週 (Apr 25, 26, 28): 現代微處理器 + 記憶體管理
 - 公告
 - 第 6 次作業 已指派
 - 學員應及早跟授課教師預約一對一線上討論, 請參照課程行事曆裡頭標注 “Office hour” 的時段, 發訊息到 Facebook 粉絲專頁, 簡述你的學習狀況並選定偏好的時段 (建議是 30 分鐘)
 - CPU caches by Ulrich Drepper
 - 進行中的繁體中文翻譯: 《每位程式開發者都該知道的記憶體知識》
 - 本文解釋用於現代電腦硬體的記憶體子系統的結構、闡述 CPU 快取發展的考量、它們如何運作, 以及程式該如何針對記憶體操作調整, 從而達到最佳的效能。
 - 解析 Linux 共享記憶體機制
 - Linux: 賦予應用程式生命的系統呼叫 *
 - Linux: 記憶體管理 *: 記憶體管理是 Linux 核心裡頭最複雜的部分, 涉及到對計算機結構、slob/slab/slub 記憶體配置器、行程和執行檔樣貌、虛擬記憶體對應的例外處理、記憶體映射, UMA vs. NUMA 等等議題。
 - Linux 核心的 /dev/mem 裝置
 - Week11 隨堂測驗: 題目 (內含作答表單)
 - 課堂問答簡記
- 第 12 週 (May 2, 3, 5): 共享記憶體 + 裝置驅動程式
 - 公告
 - 徵求 logo 製作
 - 5 月 3 日下午恢復實體課程
 - 以下材料的講解錄影
 - POSIX Shared Memory: 在 Linux 中要實作出共享記憶體 (shared memory) 的機制很多, 例如: 1) SysV shared memory; POSIX shared memory; 3) 以 mmap 對檔案進行記憶體映射; 4) 以 memfd_create() 實作跨

越行程存取; 本文章探討 POSIX shared memory 的使用, 並提供完整應用案例, 最後探討相關的同步議題。

- [C 語言: 物件導向程式設計 *](#)
- [Object-oriented design patterns in the kernel, part 1 / Object-oriented design patterns in the kernel, part 2](#)
- [C 語言: Stream I/O, EOF 和例外處理 *](#)
- [CS:APP 第 10 章重點提示 *](#)
- [Linux: 裝置驅動程式介面和模型 / 錄影 *](#) by Greg Kroah-Hartman
 - 針對 Linux v5.x 的素材請見 [《The Linux Kernel Module Programming Guide》](#)
- [How to avoid writing device drivers for embedded Linux / 錄影 *](#)
- [Linux: Device Tree / 錄影 *](#)
- [vcam](#) 是個針對 Linux 核心開發的虛擬攝影機裝置, 全部程式碼僅 1 千 5 百行, 從而可理解 V4L2 (video for Linux APIs, version 2) 的使用和 Linux 多媒體架構。開發一個虛擬的攝影機裝置除了理解 Linux 核心設計外, 也有資訊安全的幫助, 例如你可以安插相關程式碼, 紀錄有哪些應用程式偷偷啟動攝影機, 但過程中又不會揭露真正的隱私。
 - [2020 年技術報告 / 2021 年技術報告](#)
- [vwifi](#) 是個程式碼約 500 行, 真的可運作的 WiFi 裝置驅動程式, 採用 cfg80211 框架, 在 Linux v5.4 和 Linux v5.14 都測試過, 支援 scan, connect, disconnect 等 [cfg80211](#) 的介面操作, 且允許封包轉向 Linux 核心網路堆疊
- [Week12 隨堂測驗: 題目](#) (內含作答表單)
- [課堂問答簡記](#)
- 第 13 週 (May 9, 10, 12): Scalability + 同步機制
 - [以下教材的解說錄影 *](#)
 - [Linux: Timer 及其管理機制 *](#)
 - [Linux: Scalability 議題 *](#)
 - [An Introduction to Cache-Oblivious Data Structures](#)
 - 「自動快取資料結構」, 特性是無視硬體特定的快取大小, 可能達到接近最優化快取的效能;
 - 在現代 CPU 多層多種大小的快取架構下, 它的理論宣稱其能自動優化在所有層的快取的存取效率。傳統上電腦科學做偏理論的人不重視實作的效能表現, 而實作或硬體優化的從業人員往往不重視理論分析。這個學門卻是橫跨相當理論的演算法分析 (需要相當多的進階數學工具), 及相當低階的硬體效能理解;
 - 影片: [Memory Hierarchy Models](#)

- Google Research 強者的心得: [關於變強這檔事 \(九\)](#) , [設計高效 Hash Table \(一\)](#), [設計高效 Hash Table \(二\)](#)
 - **Skip List**: 置放大量數字並進行排序的資料結構。不用樹狀結構, 而改用高度不同的 List 來連接資料。資料結構在概念上可以表示成 Left Child-Right Sibling Binary Tree 的模式。是 Cache-oblivious Algorithm 的經典範例, 時間複雜度與空間複雜度與 Binary Search Tree 皆相同, 但精心調整的實作可超越 Binary Search Tree。
 - Linux 核心: [A kernel skiplist implementation \(Part 1\)](#), [Skiplists II: API and benchmarks](#)
- Linux: linked list, Queues, Maps, Binary Trees: [錄影 *](#) / [共筆](#)
- **Linux: RCU 同步機制 ***
- [C11 atomic variables and the kernel / Linux Documentation: Circular Buffers](#)
- Week13 隨堂測驗: [題目](#) (內含作答表單)
- [課堂問答簡記](#)
- 第 14 週 (May 16, 17, 19): 網路封包處理
 - 公告:
 - [2022 年 Linux 核心設計/實作課程期末專題](#)列表受理學員經由授課教師討論後, 予以更新和進行
 - [以下教材的解說錄影 *](#)
 - [討論區回顧](#)
 - [Linux 核心網路](#)
 - **cserv** is an event-driven and non-blocking web server.
 - 展現 [event-driven, non-blocking I/O Multiplexing](#) (主要是 [epoll](#)), shared memory, processor affinity, coroutine, context switch, UNIX signal, [dynamic linking](#), circular buffer, hash table, red-black tree, atomic operations 等議題的實際應用
 - 可視為 [seHTTPd](#) 的後繼改進實作
 - [Memory Externalization With userfaultfd / 錄影 *](#) / [kernel documentation: userfaults](#)
 - [How Linux Processes Your Network Packet / 錄影 *](#)
 - [Kernel packet capture technologies / 錄影 *](#)
 - **PACKET_MMAP**
 - `PACKET_MMAP` 在核心空間內配置一塊核心緩衝區, 一旦使用者層級的應用程式呼叫 `mmap` 將前述緩衝區映射到使用者層級時, 接收到的 `skb` 會直接在該核心緩衝區, 從而讓應用程式得以直接捕捉封包
 - 若沒有啟用 `PACKET_MMAP`, 就只能使用低效率的 `AF_PACKET`, 不但有緩衝區空間的限制, 而且每次捕捉封包就要一次系統呼叫。反之, `PACKET_MMAP` 多數時候不需要呼叫系統呼叫, 也能實作出 zero-copy
 - [圖解 Linux tcpdump](#)

- Week14 隨堂測驗: [題目](#) (內含作答表單)
- [課堂問答簡記](#)
- 第 15 週 (May 23, 24, 26): 多核處理器架構 + Linux 同步處理機制 + 程式碼最佳化概念
 - 公告:
 - 請學員及早進行 [課程期末專題](#) 並預約一對一討論
 - [以下教材的解說錄影](#) *
 - [Multi-Core in Linux](#) *
 - [Multicore Caches / Cache Coherence](#) * / video
 - [並行程式設計: Lock-Free Programming](#) *
 - [A Deep dive into \(implicit\) Thread Local Storage](#)
 - 允許執行緒擁有私有的資料。對於每個執行緒來說, TLS 是獨一無二, 不會相互影響。案例: 全域變數 `errno` 可能在多執行緒並行執行時錯誤, 透過 TLS 處理 `errno` 是個解決方案
 - `__thread`, 在 POSIX Thread 稱為 thread-specific data, 可見 `pthread_key_create`, `pthread_setspecific`
 - 在 x86/x86_64 Linux, `fs segment` 用以表示 TLS 的起始位置, 讓執行緒知道該用的空間位於何處
 - [RCU 同步機制](#) *
 - [CS:APP 第 5 章重點提示和練習](#) *
 - [CS:APP Assign 5.18](#) *
 - 專案賞析
 - [threaded-logger](#): 向 Linux 核心的 `lockless printk` 致敬
 - 在 glibc 的實作中, 就算是 `puts` 這樣貌似單純的實作, 也要有 lock
 - [123elf / 解釋](#)
- Week15 隨堂測驗: [題目](#) (內含作答表單)
- [課堂問答簡記](#)
- 第 16 週 (May 30, 21, Jun 3): 多核處理器架構
 - [以下教材的解說錄影 1](#) *, [錄影 2](#) *
 - [A tour of the ARM architecture and its Linux support](#) *
 - [Multiprocessor OS](#)
 - [Linux-Kernel Memory Ordering: Help Arrives At Last!](#) / video *
 - `atomic` 和 `memory order`, `memory barrier` 的實作和效果
 - [Memory barriers in C](#)
 - Week16 隨堂測驗: [題目](#) (內含作答表單)

- 第 17 週 (Jun 6, 7, 9): 即時 Linux 的基礎建設
 - 公告
 - 佔學期總成績 50% 的自我評量, 請在 6 月 30 日前完成。範例: [User/OscarShiang](#)
 - 自我評量的網址**必須**符合 `/User/你的GitHub帳號名稱` 格式 (區分大小寫), 請不要打錯字
 - 自我評分項目 (都要有對應的超連結和延伸資訊)
 - 成果發表: 與 Linux 核心相關的公開演講
 - Linux 核心和相關專案貢獻: 貢獻到 Linux 核心和相關專案
 - 作業/隨堂測驗: 你的開發紀錄, 人在做, Google 在看
 - 期末專題: 開發紀錄, 標注與授課教師「一對一討論」的時間, 並列出你的啟發及成果
 - 所見所聞所感, 包含授課教師編撰/翻譯的書籍 (《Demystifying the Linux CPU Scheduler》, 《Concurrency Primer》, 《Linux Kernel Module Programming Guide》, 〈每位程式開發者都該有的記憶體知識〉) 的讀後
 - 自我評量: 介於 1 到 10 之間的整數 (不要自作主張寫 `8.7` 這樣的數值)
 - [以下教材的解說錄影](#) *
 - [Re: \[問卦\] 精通作業系統對Coding有什麼幫助?](#)
 - [Linux: Timer 及其管理機制](#)
 - [PREEMPT_RT 作為邁向硬即時作業系統的機制](#)
 - [Linux 核心搶佔](#)
 - Week17 隨堂測驗: [題目](#) (內含作答表單)
- 第 18 週 (Jun 13, 14, 16): 多核處理器, 時鐘管理, Real-time
 - Rust 程式語言
 - 現況: 已被 Google Android 團隊選為開發系統軟體的另一個程式語言, 與 C 和 C++ 並列; 自 2017 年 Facebook 內部採納 Rust 程式語言的專案增加, 像是加密貨幣 Diem (前身為 Libra) 就將 Rust 作為主要程式語言並對外發布,
 - Steve Klabnik 與 Carol Nichols, 及 Rust 社群的協同撰寫的《The Rust Programming Language》線上書籍, 由台灣的 Rust 社群提供繁體中文翻譯
 - [Rust by Example](#)
 - [C vs. Rust](#)
 - [Linux 核心採納 Rust 的狀況](#)
 - Memory Barrier
 - [Memory Barriers in the Linux Kernel: Semantics and Practices](#)
 - [From Weak to Weedy: Effective Use of Memory Barriers in the ARM Linux Kernel](#) / video *
 - [Uh-oh, It's I/O Ordering!](#) / video * / 筆記

◦ Week18 隨堂測驗: [題目](#) (內含作答表單)

分類: [linux](#) [performance](#) [security](#) [compiler](#) [kernel](#) [rtos](#) [unix](#)

ALSO ON NCKU CSIE WIKI

0	0	0	No Access	No Access	All accesses generate Permission faults
0	0	1	Read/Write	No Access	Privileged access only
0	1	0	Read/Write	Read Only	Writes in User mode generate Permission faults
0	1	1	Read/Write	Read/Write	Full access
1	0	0			Reserved
1	0	1			Privileged read-only
1	1	0	Read Only	Read Only	Privileged and User read-only, deprecated in Linux 2.6.31

FreeRTOS (MMU)

7 years ago • 2 comments

協作者 2015 年春季 陳冠任, 許士杰, 黃睦林, 孫瑋, 涂逸祥 共筆 2015 年春季 ...

Linux 效能分析工具: Perf

6 years ago • 4 comments

簡介 Perf 全名是 Performance Event, 是在 Linux 2.6.31 ...

Arch Linux 介紹 與 安裝

8 years ago • 3 comments

如果懶得按no no yes yes...
:: Please enable JavaScript to view the comments ...

期末專題:

9 years ago •

Please enable JavaScript to view the comments by Disqus.

3 Comments

NCKU CSIE Wiki

 Disqus' Privacy Policy

 Login ▾

 Favorite 13

 Tweet

 Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 



Name



slbttt • 21 days ago

Thanks for publish those content.

^ | ▾ • Reply • Share ›



Anjan • 3 years ago

can some one please translate this guide to English? Or is there one already available?

^ | ▾ 1 • Reply • Share ›



Webber Han → **Anjan** • 2 years ago

I attached some English translation for week #1. Hope you enjoy this great class.

^ | ▾ • Reply • Share ›

 Subscribe

 Add Disqus to your site

 Do Not Sell My Data

DISQUS



本站所有內容，除另有標註外，採用創用 CC 姓名標示-相同方式分享 3.0 台灣 授權條款授權

| [說明](#) |

Powered by [gitit](#) |

Customized by CrBoy |