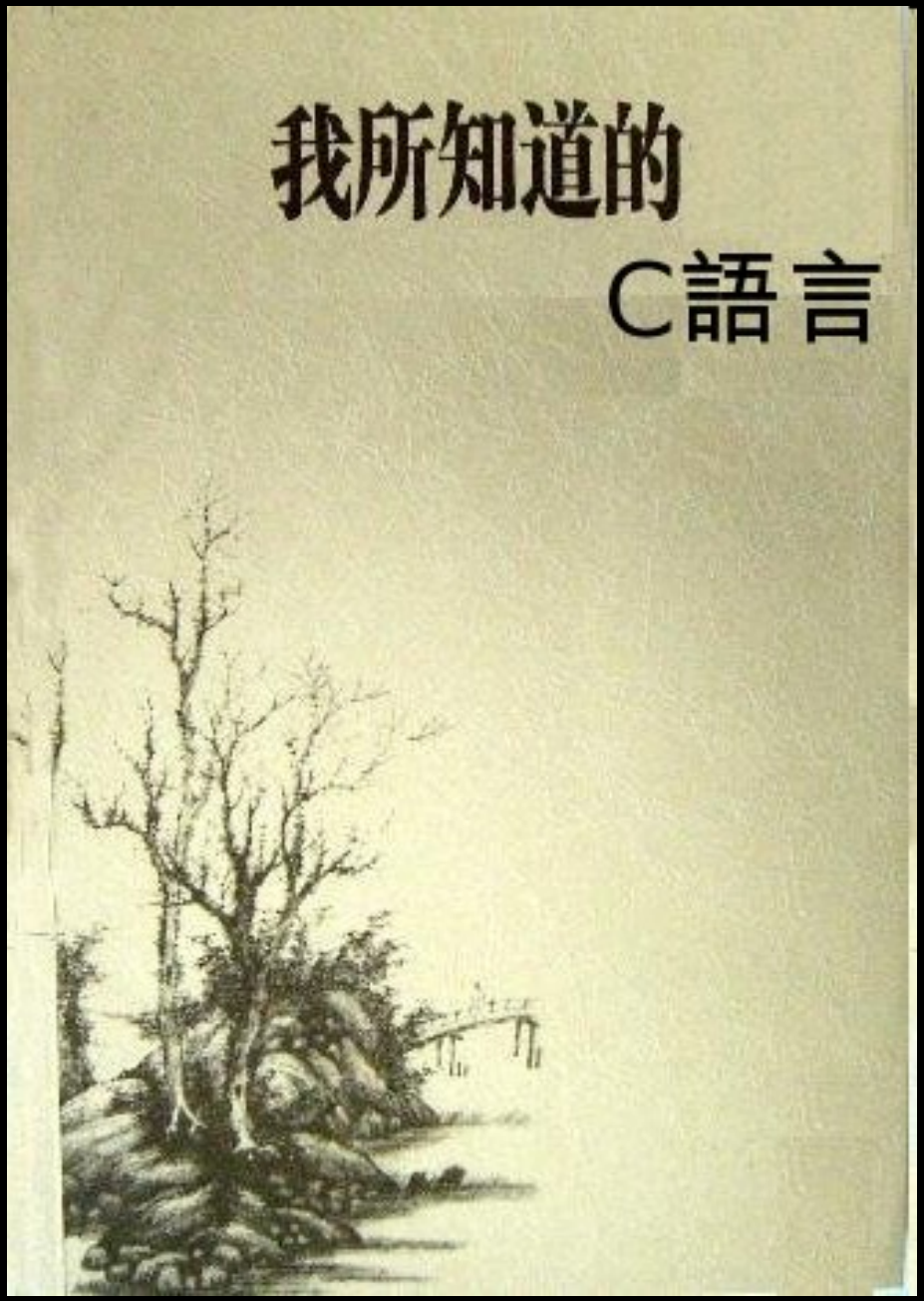


我所知道的 C 語言

Jim Huang(黃敬群) “jserv”
website: http://jserv.sayya.org/
blog: http://blog.linux.org.tw/jserv/
March 28, 2009

C

程式語言

C

程式語言

科技**出櫃**：「我就是愛 C 語言！」

This is C!

```
#include <stdio.h>

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

I Speak BASIC

```
10 PRINT "Hello World!"
20 GOTO 10
```

I Speak C

```
main()
{
  for(;;) {
        printf ("Hello World!\n")}
}
```

I Speak BINARY

```
00001101000010100000011010000
10100000110100001010011011 01
01100001011010010110111000 10
10000010100100000110100001010
01111011000011010000101000 10
```

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
- 理解C = 設計作業系統的需求+操控硬體的機制
- 不只學語法，更要深入欣賞
  - 《文心雕龍》
- 由軟體反思硬體組成
  - 近代的硬體受C語言影響頗深

提綱

- 海角 C 語言 -- 被遺忘的淒美故事
- 高度物件導向的 C
- 窺探 C 程式
  - 尋訪 C 程式的資料表示
  - 奇妙的 pointer
- C 語言與硬體擦出的火花，呈現 C 設計的彈性

海角 C 語言

被遺忘的淒美故事

C語言設計來開發作業系統

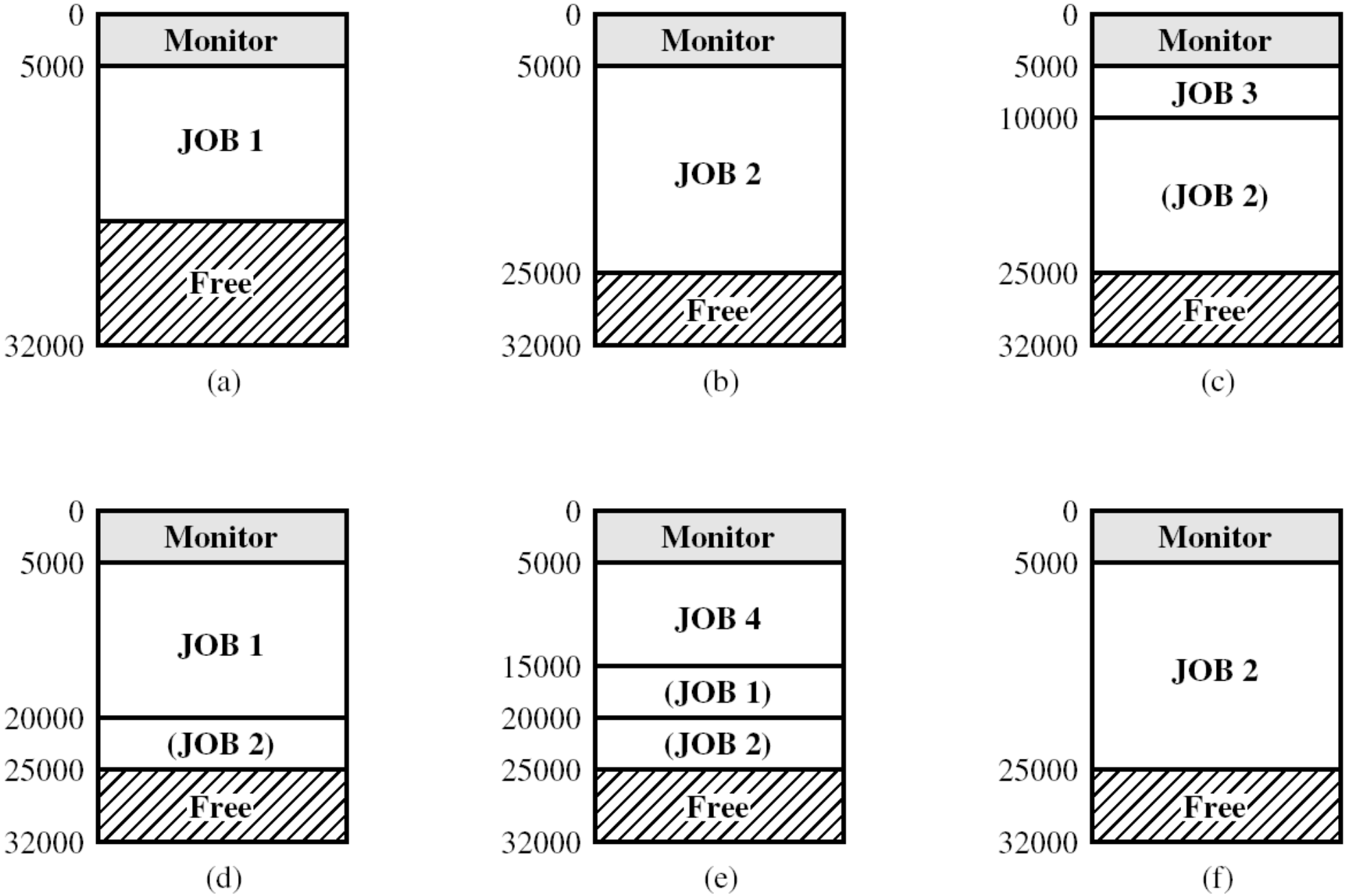
Ken Thompson　　Dennis Ritchie

MUTICS: Time-Sharing Systems

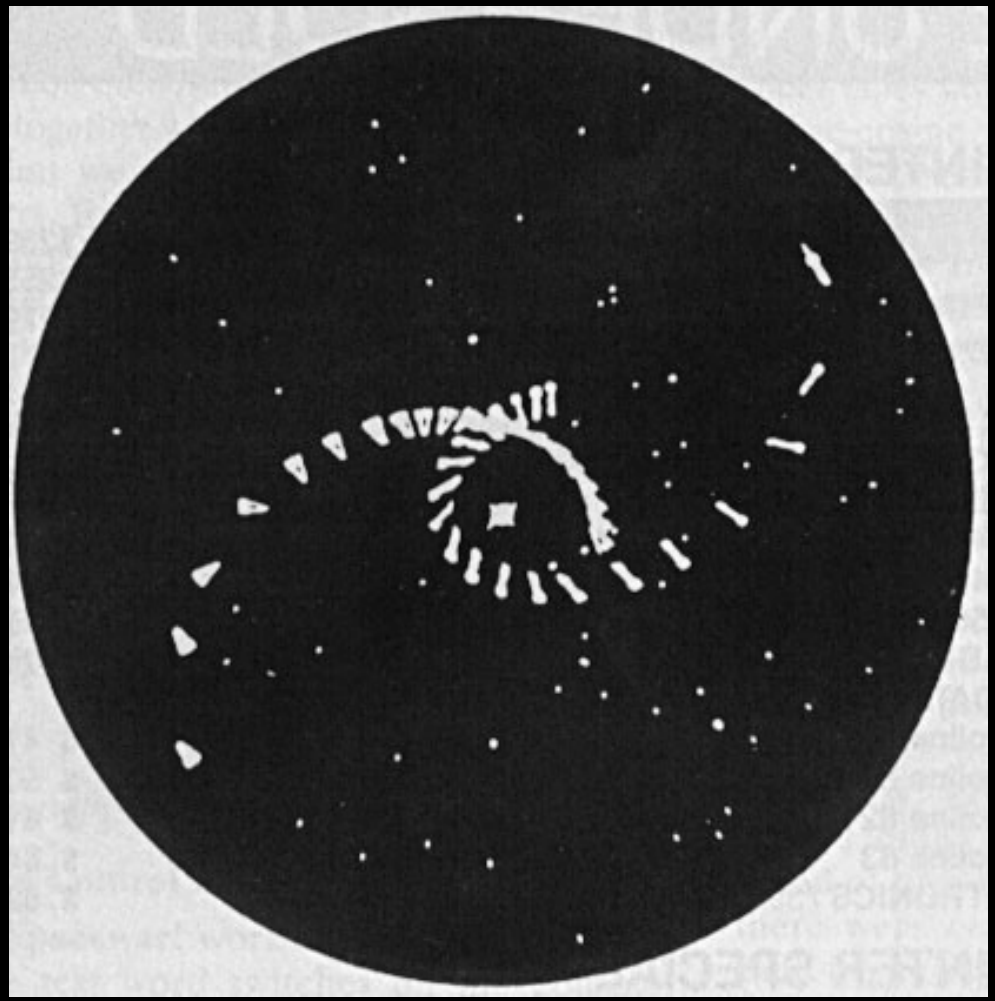
- 因爲多重程式的使用，批次處理可以相當有效率。然，許多工作需讓使用者與電腦互動。
- 多重程式也可用來處理多個交談式工作，稱爲「分時」—處理器的處理時間被許多使用者所分享

	批次多重程式處理	分時處理
主要目標	處理器使用率最大化	回應時間最小化
作業系統的命令來源	由工作所提供的工作控制語言 (JCL) 命令	在終端機輸入的指令

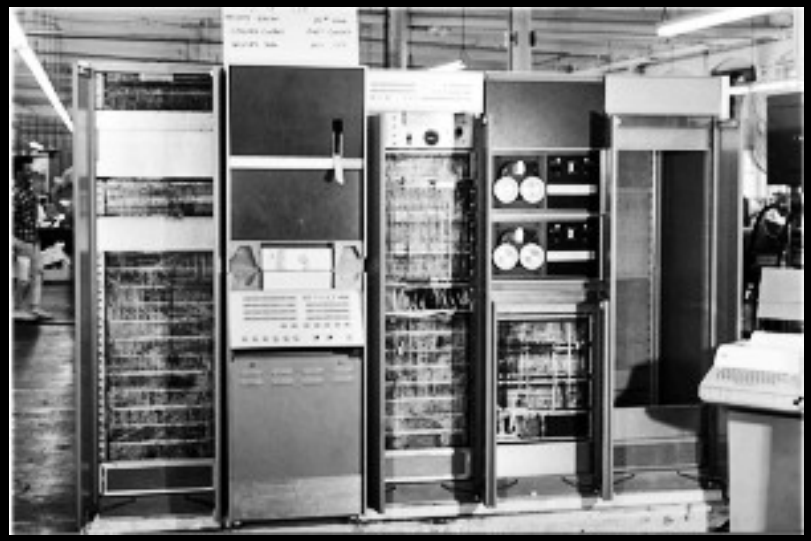
- 1961 年 CTSS 系統：系統時鐘每 0.2 秒發出一次中斷請求，作業系統重新取得控制權


0
5000
32000
Monitor
JOB 1
Free
(a)
0
5000
25000
32000
Monitor
JOB 2
Free
(b)
0
5000
10000
25000
32000
Monitor
JOB 3
(JOB 2)
Free
(c)
0
5000
20000
25000
32000
Monitor
JOB 1
(JOB 2)
Free
(d)
0
5000
15000
20000
25000
32000
Monitor
JOB 4
(JOB 1)
(JOB 2)
Free
(e)
0
5000
25000
32000
Monitor
JOB 2
Free
(f)

大師也愛玩電動！



Space Pilot



PDP-7

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

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.

淒美的故事

PDP-11

UNICS: 1969 – PDP-7 minicomputer
($72K, 18 bit)

因專利議題，從 PDP-7 移轉到 PDP-11
($10K, 16 bit)

V1: 1971

V3: 1973 (pipes, C language)

V6: 1976 (rewritten in C, base for BSD)

V7: 1979 (Licensed, portable)

C 語言由 Dennis Ritchie 設計

- 前身

  ALGOL 60 (1960)
  CPL (Cambridge, 1963)
  BCPL (Martin Richard, 1967)
  B (Ken Thompson, 1970)

範例 C 程式

/usr/include/stdio.h

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

... definitions and
protoypes

#endif
```

/usr/include/stdlib.h

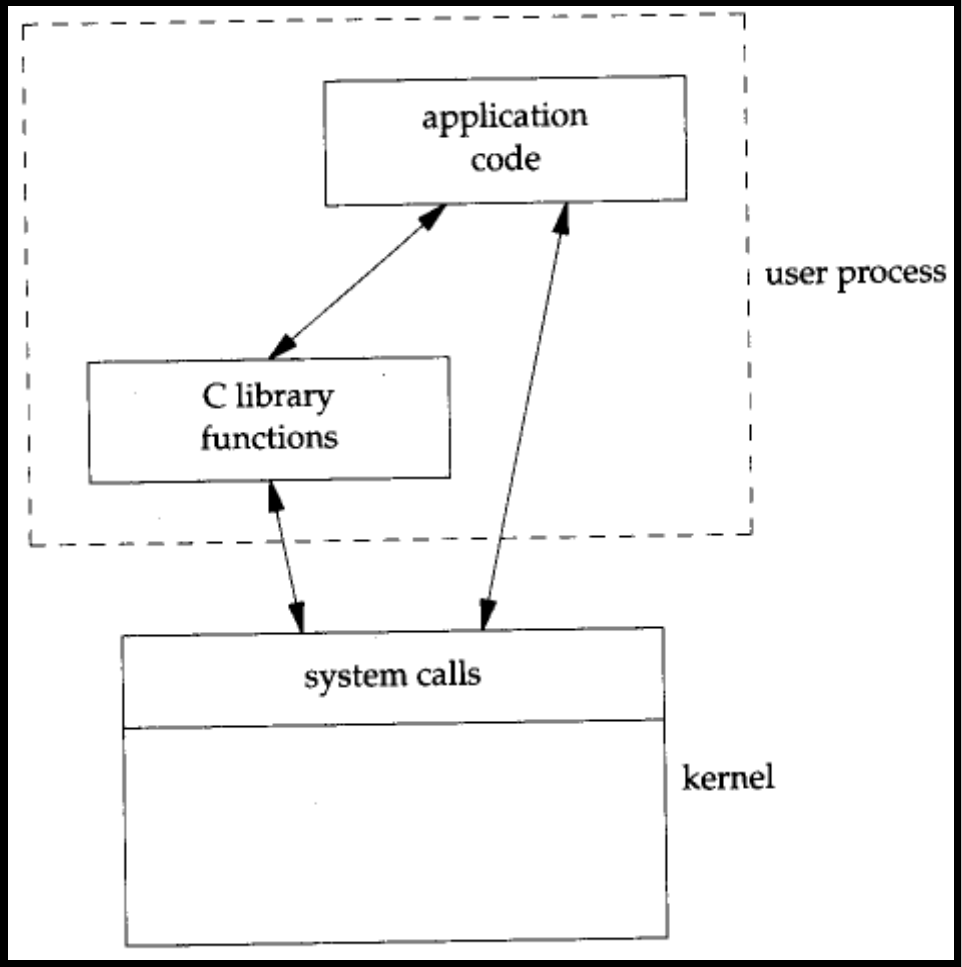
```
/* prevents including file
 * contents multiple
 * times */
#ifndef _STDLIB_H
#define _STDLIB_H

... definitions and
 protoypes

#endif
```

example.c

```
/* this is a C-style comment
 * You generally want to palce
 * 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

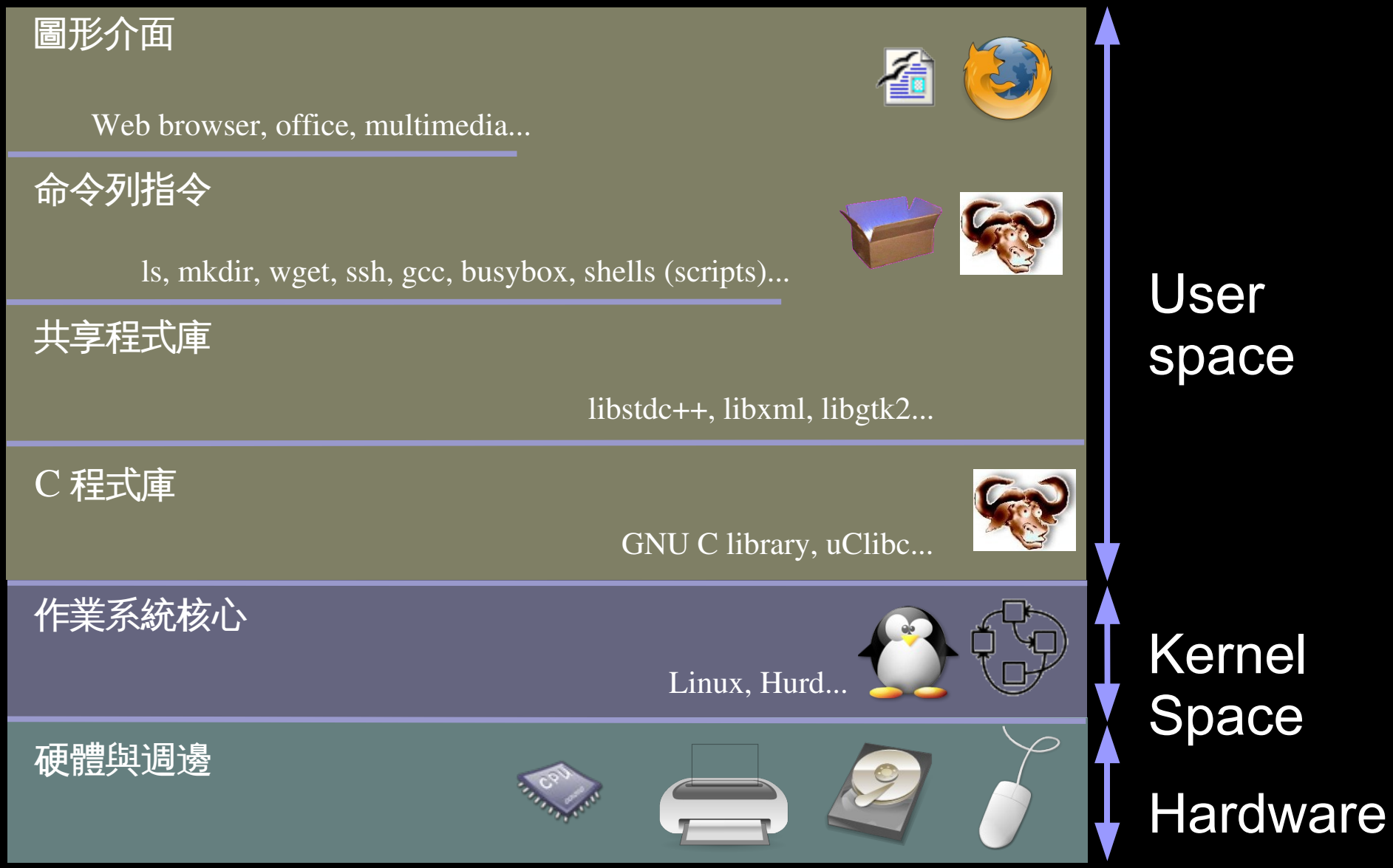
- Kernel 實做一系列特別的系統服務
- 使用者的程式透過觸發硬體 TRAP 而呼叫 Kernel 的服務
- TRAP 使 CPU 切入保護 / 特權模式，致使 Kernel 執行 system call 。直到做完， CPU 切回使用者模式
- A C language API exists for all system calls



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

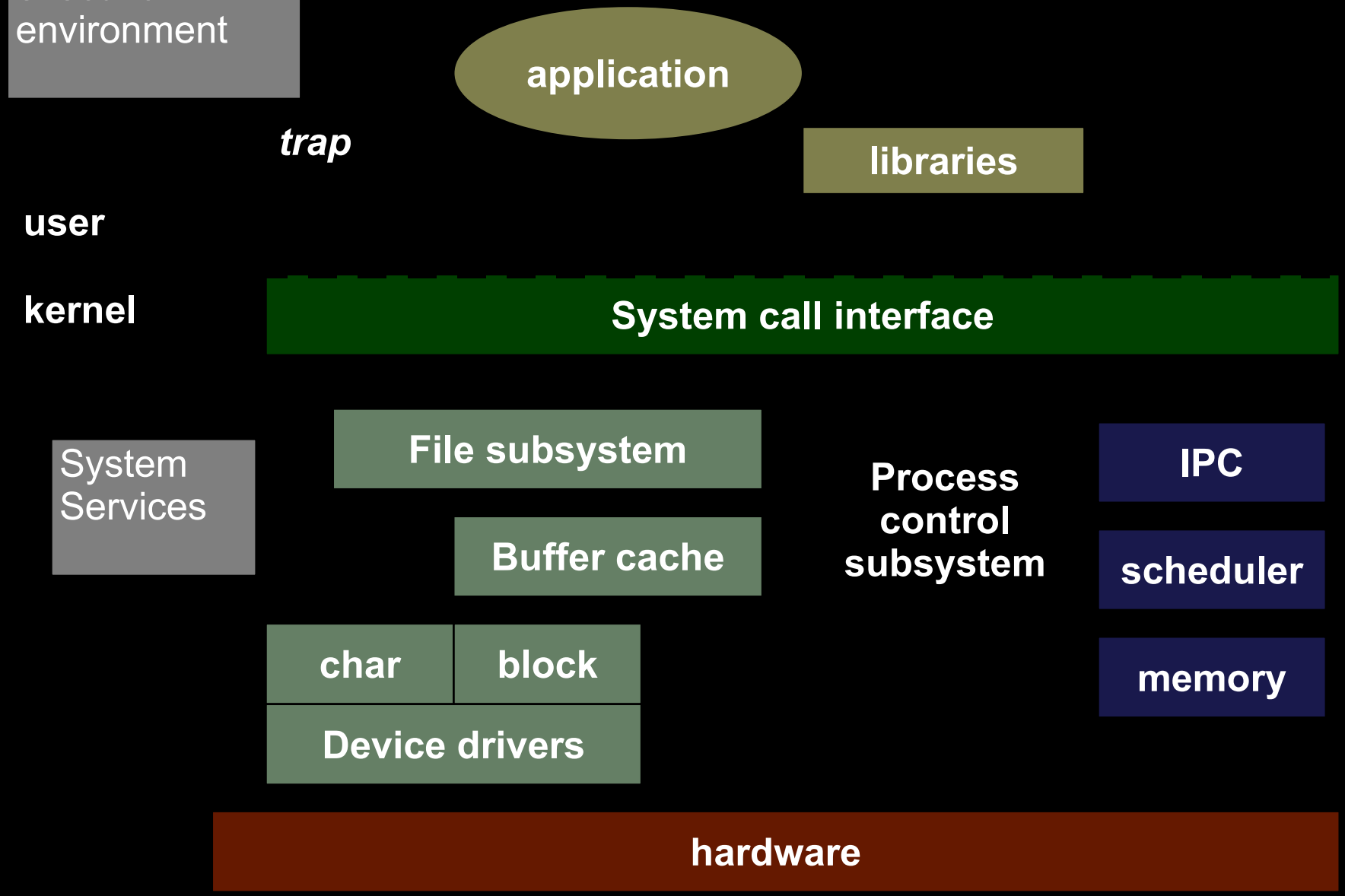
Unix 系統架構


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
CPU
User space
Kernel Space
Hardware

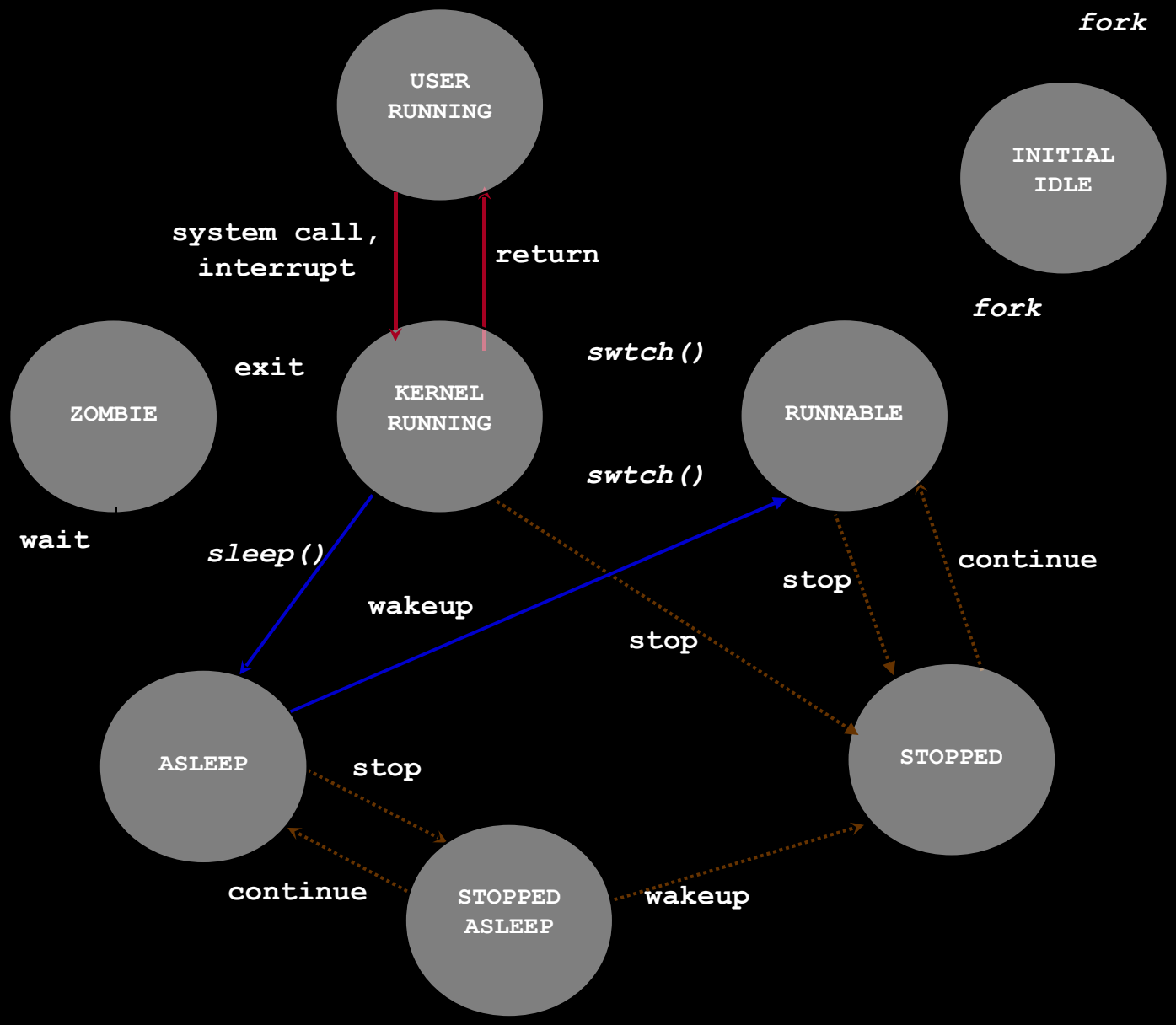
UNIX 結構

shell scripts system calls	User Space	utilities C programs compilers
signal handler device drivers	Kernel	scheduler swapper
terminal disk	Devices	printer RAM

典型核心架構


execution environment
application
trap
libraries
user
kernel
System call interface
System Services
File subsystem
Buffer cache
char
block
Device drivers
Process control subsystem
IPC
scheduler
memory
hardware

Process states



Process Address Space

0xffffffff

Kernel stack

Kernel address space

0x7fffffff

stack

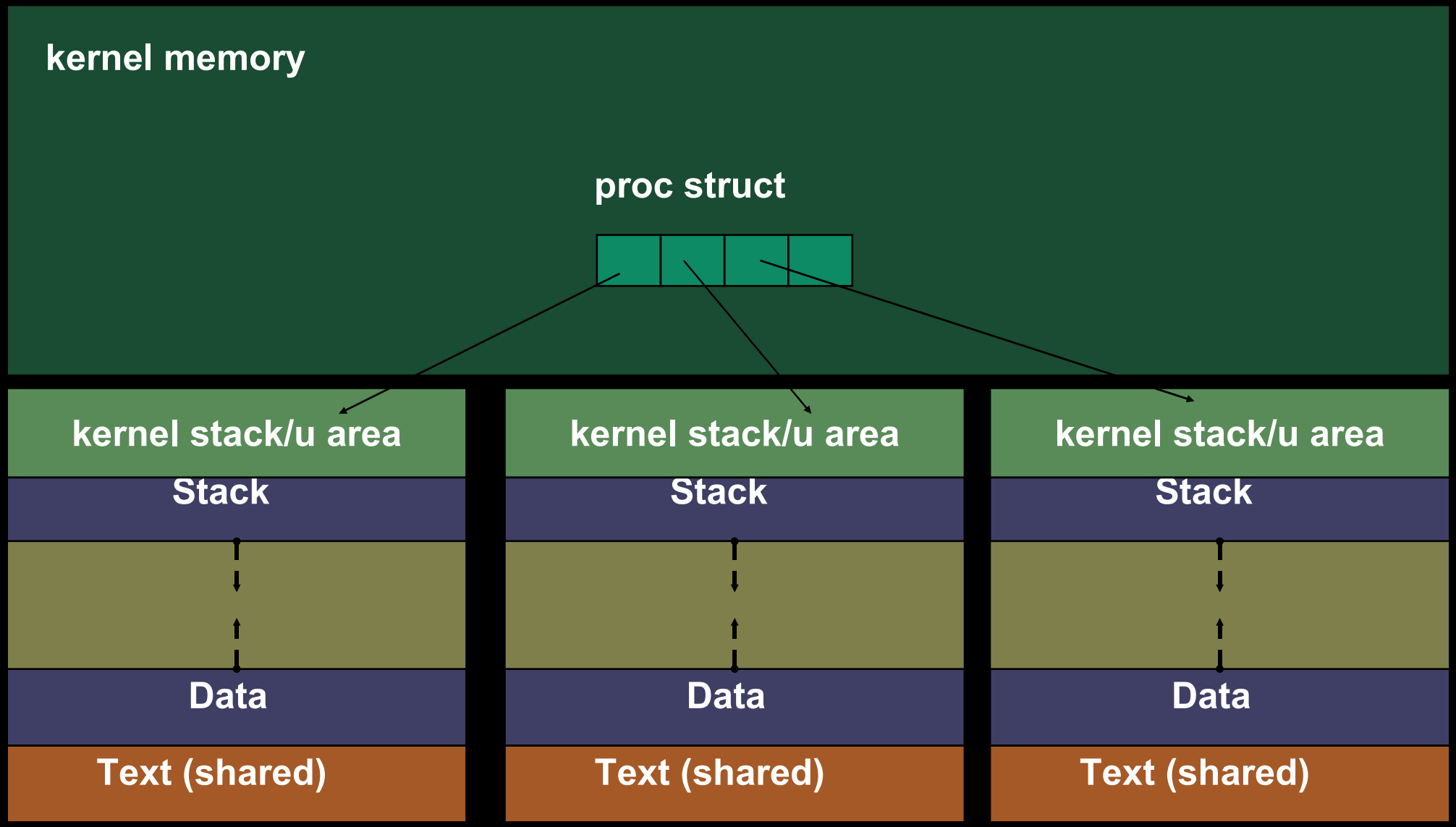
Process address space

Data

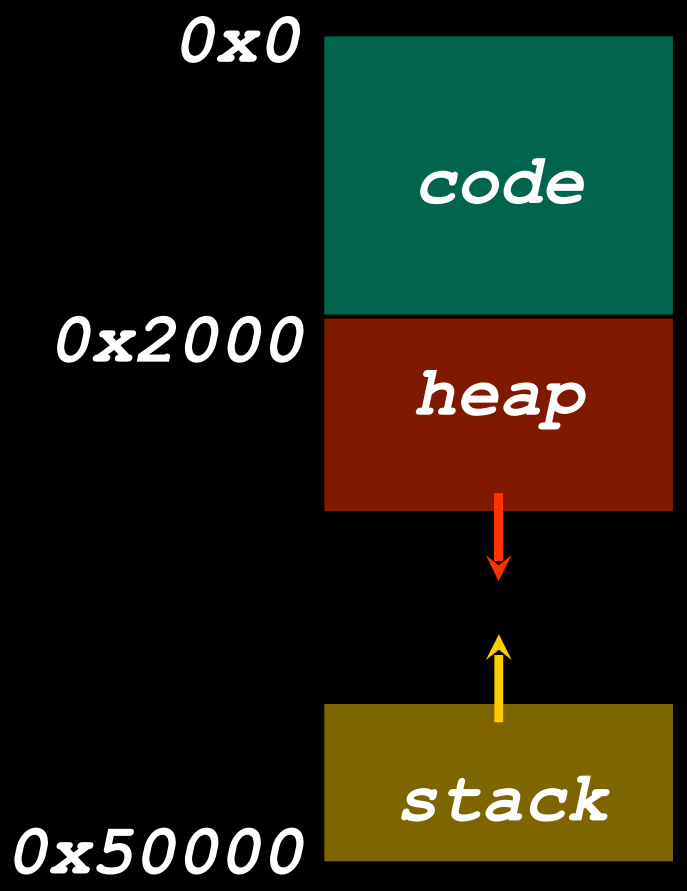
0x00000000

Text (shared)

Big Picture of Process



Example memory layout



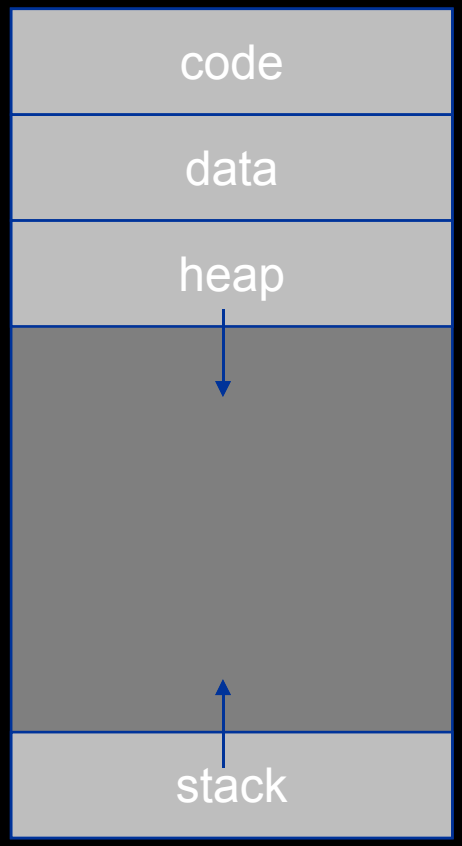
Immutable

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

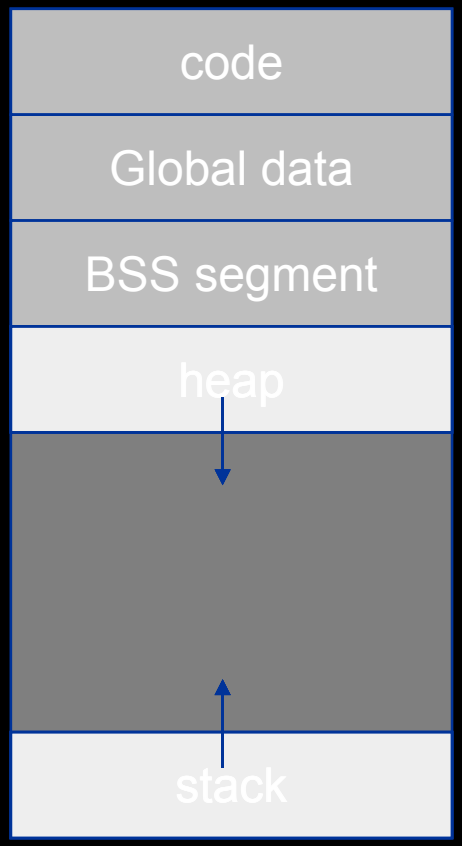
Function calls, local variables

Memory Layout

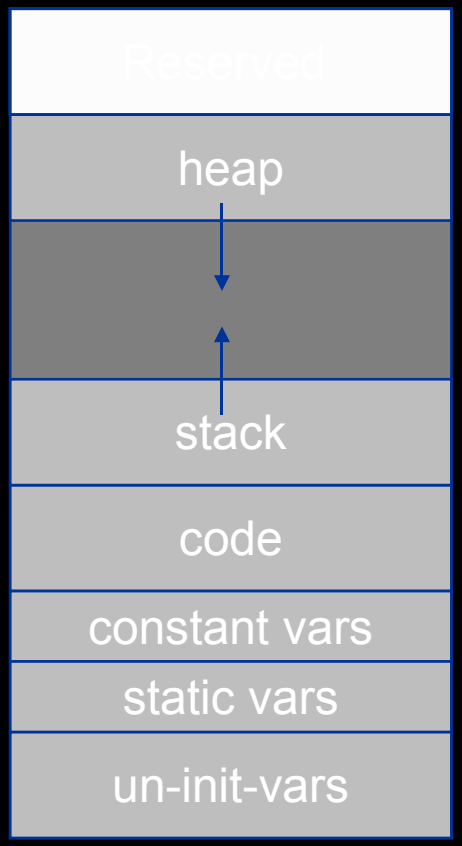
Generic



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

Nada Productions COLLECTION

EDWARD D. WOOD, JR.'S

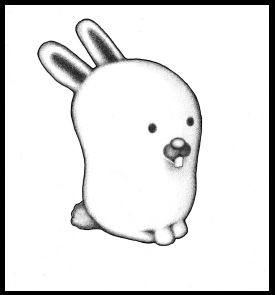
PLAN 9 FROM OUTER SPACE

almost starring **BELA LUGOSI**

with Gregory Walcott • Vampira • Tor Johnson

Conrad Brooks • Paul Marco • Norma McCarty Wood

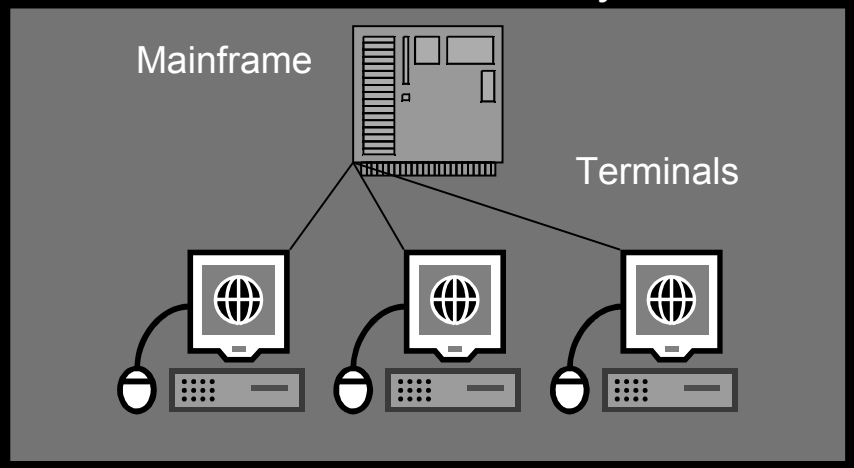
Plan 9



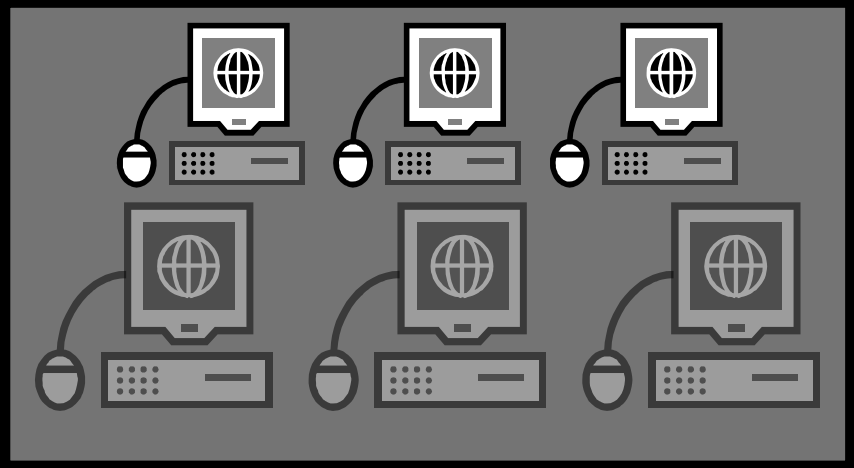
- 評價爲史上最爛的科幻片，由 Ed Wood 自立拍攝
- 由 UNIX/C 的開發人員於 1980 年代重新開發
- 設計哲學：
  - Resources are represented as file trees
  - Resources are privately assembled by processes
  - Resources are accessed by a standard protocol
- 簡化版本爲 Inferno

Plan 9 目標

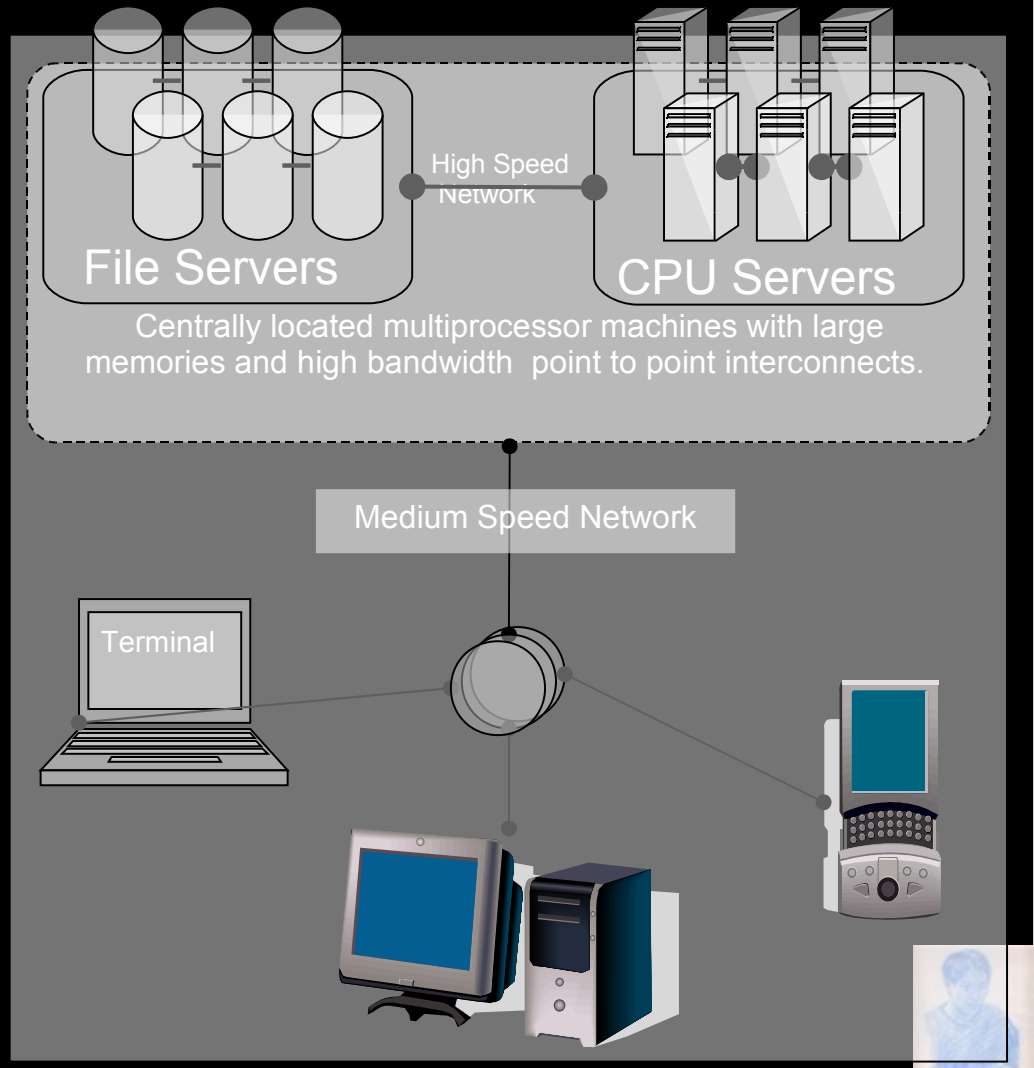
Centralized Time shared Systems



Personal Workstation Systems



Plan 9 Distributed Operating System



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) 被嚴格定義爲 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"

高度物件導向的 C

賞析：

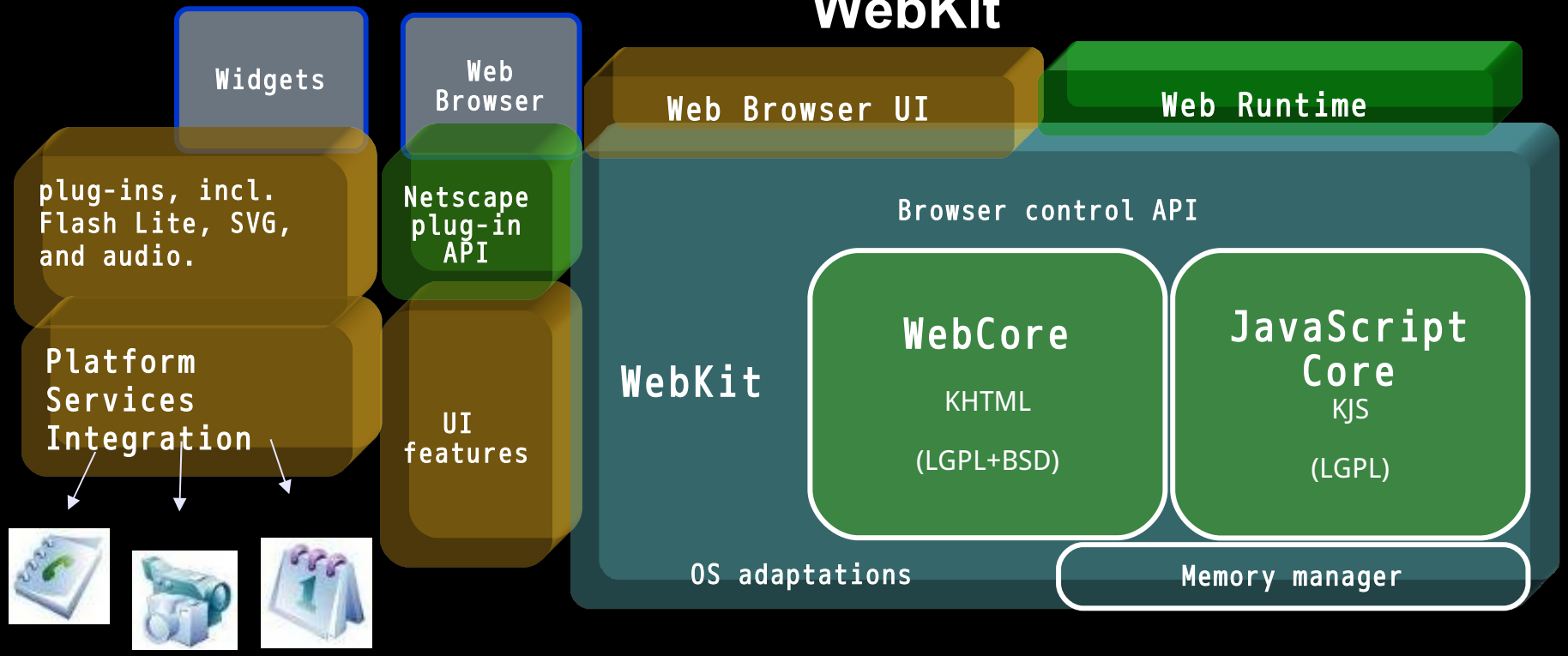
Gtk+/WebKit

JavaScriptCore

Linux Kernel

Runtime for Web Developers

Web Browser and Web Runtime are based on WebKit



WebKit: http://www.webkit.org

Gtk+/WebKit：打造網路瀏覽器

- 目標
  - 在 Web 2.0 的時代，將豐富的服務「嵌入」到 Gtk+ 應用程式中
  - 思考 Gtk+ 程式的運作要素

```
#include <stdlib.h>
#include <gtk/gtk.h>
#include <webkit/webkit.h>

static void hello( GtkWidget *widget, gpointer  data ) { ... }

int main (int argc, char *argv[])
{
   char *uri = "http://www.google.co.uk";
   gtk_init (&argc, &argv);
   GtkWidget *web_view = webkit_web_view_new ();

   GtkWidget *window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
   g_signal_connect (G_OBJECT (window), "destroy", G_CALLBACK (hello), NULL);
   gtk_window_set_default_size (GTK_WINDOW (window), 680, 480);

   gtk_container_add (GTK_CONTAINER (window), web_view);

   gtk_widget_show_all (window);

   webkit_web_view_open (WEBKIT_WEB_VIEW (web_view), uri);

   gtk_main ();
   return 0;
}
```

引入 <gtk/gtk.h> 與 <webkit/webkit.h> 標頭檔

自訂的 callback function

留意 g_signal_connect 的呼叫

Gtk+ 的初始化動作

建立 WebKit 的 WebKit::WebView 的實體

進入 Gtk+ 的 main loop

main.c

第一次寫瀏覽器就上手！

觀察應用程式的外框大小：
680x480

當按下 [X] 按鈕結束執行時，
自訂的 callback function 會執行

JavaScriptCore
portable C API

WebCore
content engine

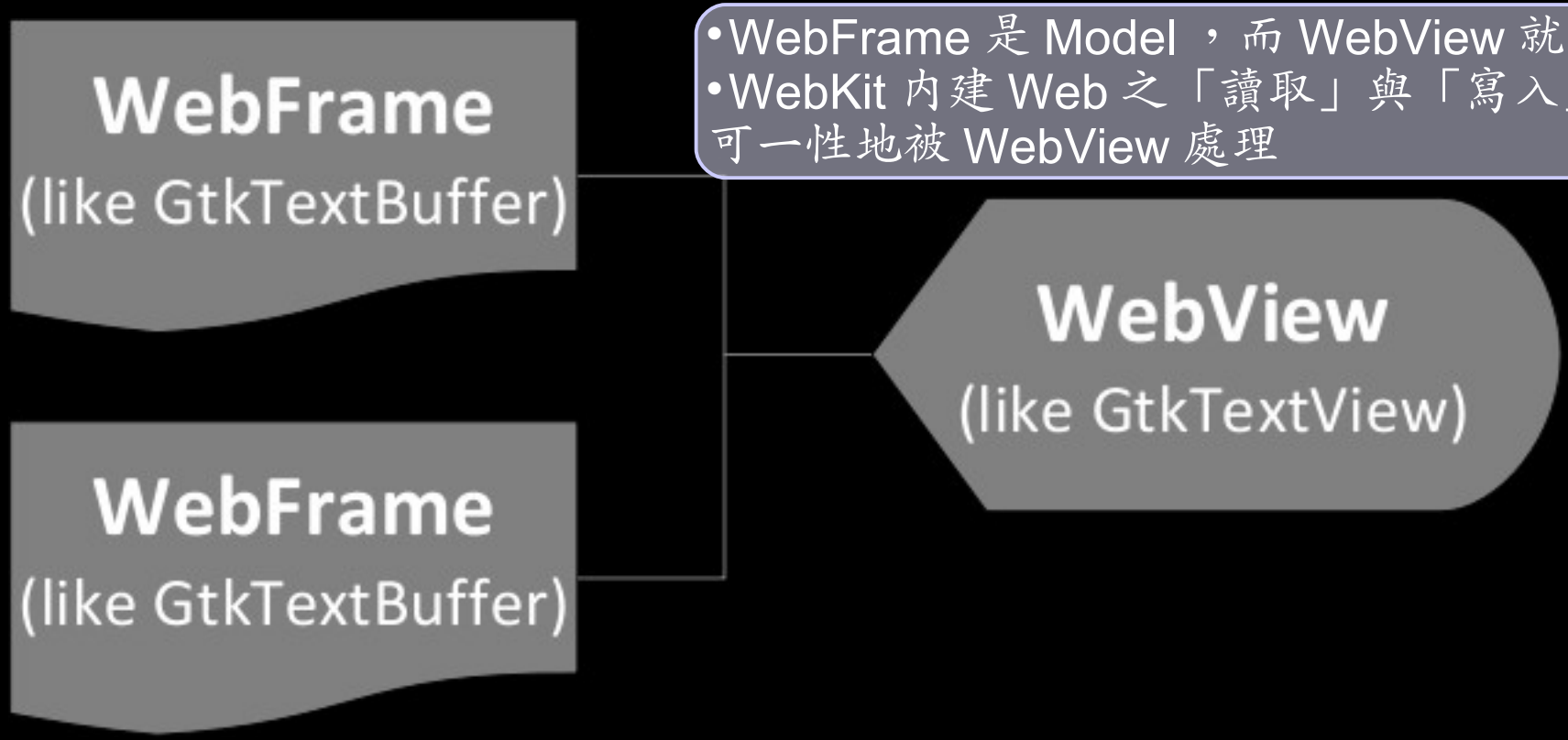
WebKit
GObject API

GTK+ applications
C, C#, C++, Vala, Python

整合 Gtk+/WebKit 相當便利

Gtk+/WebKit 依循 Model-View 設計

- 思維：自資料來源 (model) 中，將 widget 視覺的部份 (view) 抽離
  - GtkTextView : GtkTextBuffer
  - WebKitWebView : WebKitWebFrame



- WebFrame 是 Model ，而 WebView 就是 View
- WebKit 內建 Web 之「讀取」與「寫入」能力，可一性地被 WebView 處理

提醒

- 善用 devhelp 查閱 Gtk+ Reference Manual
- 觀察程式碼的要點：版面配置、 GObject signal 的連結（connect）與觸發條件

深入 JavaScriptCore

- JSGlobalContext
- JSObject
- JSString

```
function Login(string, anotherString) {
  this.username = string;
  var password = anotherString;
  this.check = function(pwrd) {
    return password == pwrd;
  };
  this.reset = function(oldPwrd,newPwrd) {
    if (this.check(oldPwrd)) {
      password = newPwrd;
    };
  };
};
login = new Login('Doe','xyz');
login.username; //=>'Doe'
login.password; //=>undefined
login.reset('xyz', 17);
login.check(17); //=>true
```

整合 JavaScriptCore API 與 C 物件

- 擴充 JavaScript，加入自訂 callback
- 建立 JS 執行環境

```
$ ./js 'print("Hello World")'
Hello World
```

```
$ ./js "var data = {2:'二',4:'四',6:'六',8:'八',10:'十'};
  for (var i = 1; i <= 10; i++) print(i%2?i:data[i])"
1
二
3
四
5
六
7
八
9
十
```

print() 爲自訂函式，
其實做對應於 C 程式

```
#include <JavaScriptCore/JavaScript.h>

static JSValueRef jsGlobalPrint( … );

int main(int argc, char** argv)
{
  if (argc == 1) exit(0);

  JSGlobalContextRef ctx = JSGlobalContextCreate(NULL);
  JSObjectRef jobjGlobal = JSContextGetGlobalObject(ctx);

  JSStringRef jstrPrint = JSStringCreateWithUTF8CString("print");
  JSObjectRef jfuncPrint = JSObjectMakeFunctionWithCallback(
  ctx, jstrPrint,
        (JSObjectCallAsFunctionCallback) jsGlobalPrint);
  JSObjectSetProperty(
        ctx, jobjGlobal, jstrPrint, jfuncPrint,
        kJSPropertyAttributeNone, NULL);
  JSStringRelease(jstrPrint);

  JSStringRef jstrSource = JSStringCreateWithUTF8CString(argv[1]);
  JSEvaluateScript(ctx, jstrSource, NULL, NULL, 0, NULL);
  JSStringRelease(jstrSource);

  JSGlobalContextRelease(ctx);
  JSGarbageCollect(ctx);
  return 0;
}
```

js.c

```
static JSValueRef jsGlobalPrint(
      JSContextRef        ctx,
      JSObjectRef         jobj,
      JSObjectRef         jobjThis,
      size_t              argLen,
      const JSObjectRef   args[],
      JSValueRef*         jobjExp);

int main(int argc, char** argv)
{
   if (argc == 1) exit(0);

   JSGlobalContextRef ctx = JSGlobalContextCreate(NULL);
   JSObjectRef jobjGlobal = JSContextGetGlobalObject(ctx);

   JSStringRef jstrPrint = JSStringCreateWithUTF8CString("print");
   JSObjectRef jfuncPrint = JSObjectMakeFunctionWithCallback(
         ctx, jstrPrint,
         (JSObjectCallAsFunctionCallback) jsGlobalPrint);
....
```

js.c

```
...
static JSValueRef jsGlobalPrint(
      JSContextRef        ctx,
      JSObjectRef         jobj,
      JSObjectRef         jobjThis,
      size_t              argLen,
      const JSObjectRef   args[],
      JSValueRef*         jobjExp)
{
   if (argLen) {
      JSStringRef jstrArg =
         JSValueToStringCopy(ctx, args[0], jobjExp);
      size_t len =
         JSStringGetMaximumUTF8CStringSize(jstrArg);
      char *szArg   =  (char *) malloc(len);
      JSStringGetUTF8CString(jstrArg, szArg, len);
      puts(szArg);
      JSStringRelease(jstrArg);
      free(szArg);
   }

   return JSValueMakeUndefined(ctx);
}
```

js.c

Everything is file.

::Device File::

Types of Device Files

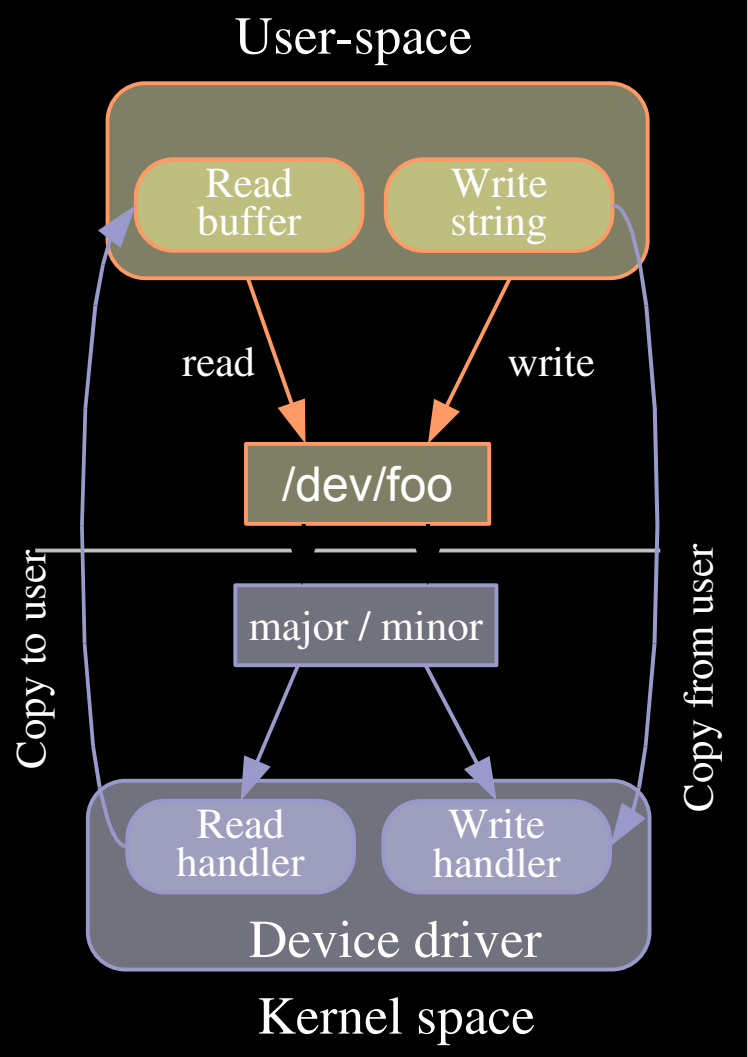
Character Device Driver	Block Device Driver	Network Device Driver	USB, Firewire, SCSI,...

```
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

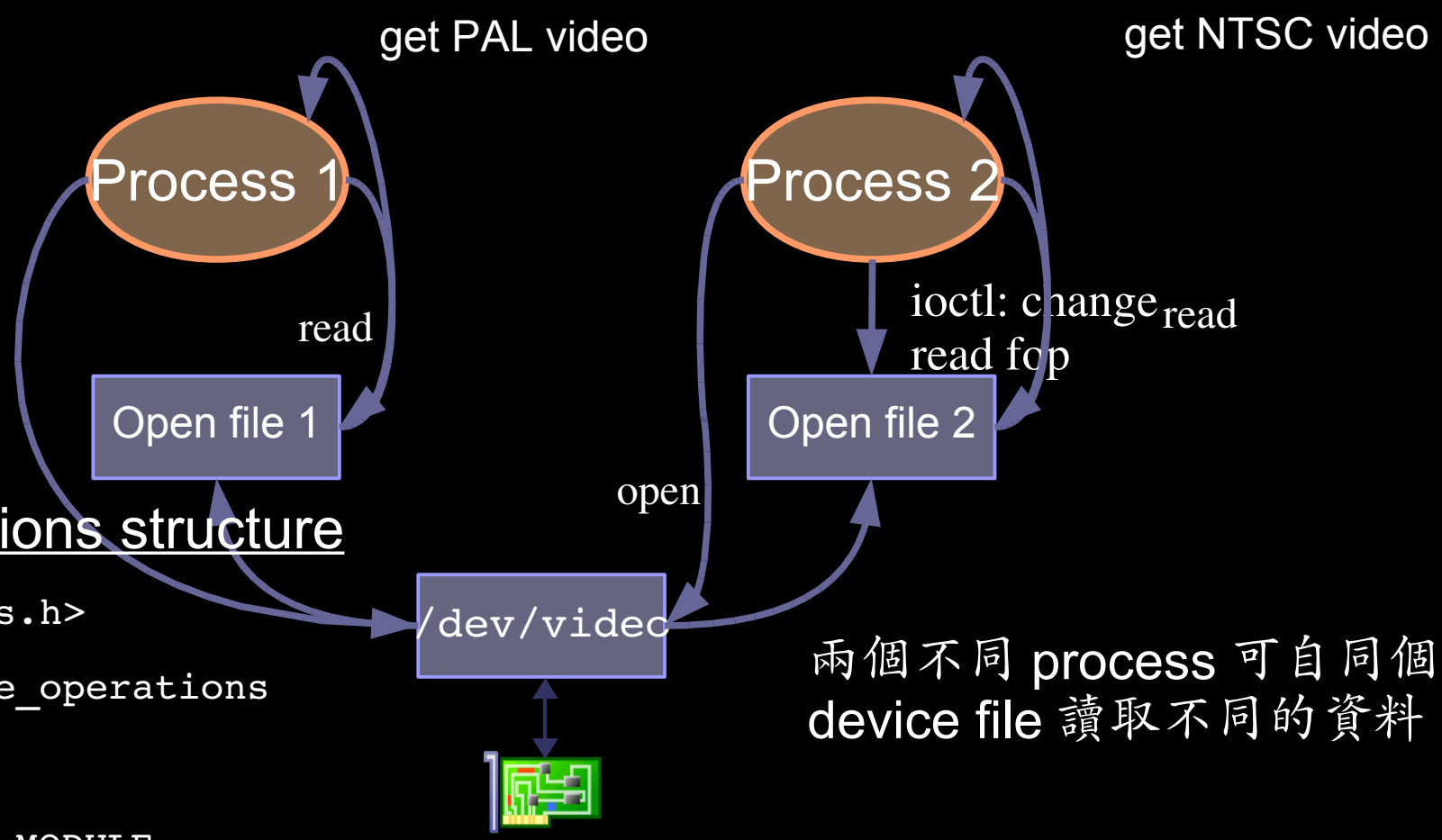
- /dev 目錄下的檔案稱爲 device files，用以辨識 / **對應於**裝置
- Kernel 透過 major number 來指定正確的驅動程式給 device files
- Minor number 只有驅動程式本身會使用到



```
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



兩個不同 process 可自同個 device file 讀取不同的資料

實做 file_operations structure

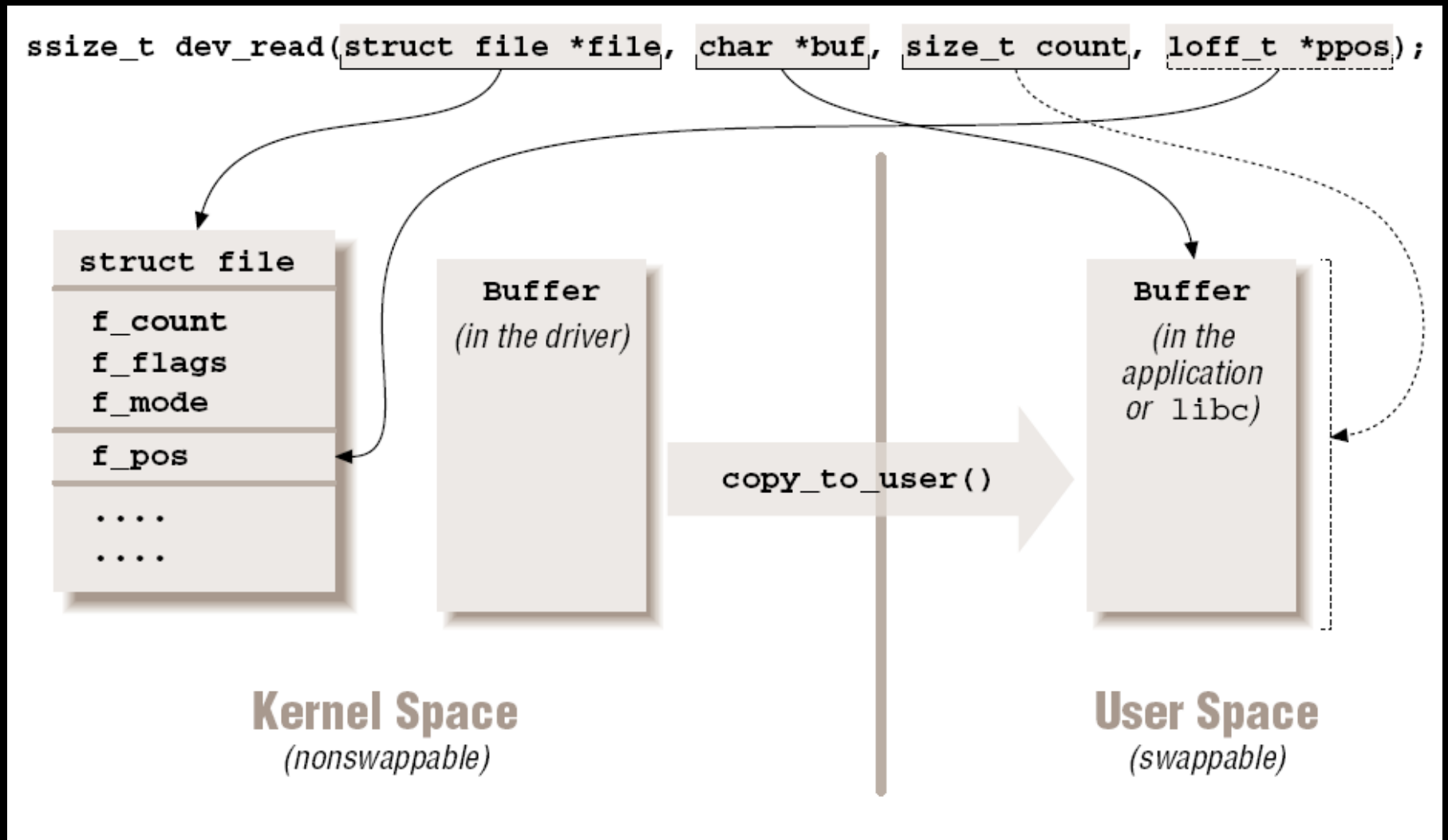
```
#include <linux/fs.h>

static struct file_operations

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

將上述 moko_read/moko_write 指向爲具體實做，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);
```



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
```

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

用 C 語言模擬 LISP/Scheme 語法

```
(define (fact n)
  (if (= n 0)
      1
      (* n (fact (- n 1)))))
```

in C ?!

- 整個 scheme 可說是 read-eval-print loop 的運作方式
  - 即「讀取、計算，印出」的過程

- 由函數組合所構成，可以巢狀組合，以小括號將運算式括起來，函數名稱或運算元在左括號的右邊，運算子彼此以空白爲間隔，如“3+4*5”這個運算式，以 Scheme 語法撰寫如：

  (+ 3 (* 4 5))

  類似資料結構中的前序運算式

- 基本的資料型態爲原子 (atom) 及字串 (list)：
  - 原子 (atom) 包含符號 (symbol) 及數值 (number)
  - 串列 (list) 則是以小括號括起來的一串資料

```
(define (fact n)
    (if (= n 0)
        1
        (* n (fact (- n 1)))))
```

階層運算 /Scheme

```
define(int, factorial, (int n),
       if(eq(n, 0),
          1,
          mul(n, factorial(sub(n, 1)))))

define(int, main, (void),
       (printf("10! = %d\n", factorial(10)),
              EXIT_SUCCESS))
```

階層運算 /C

用 C 語言模擬 LISP/Scheme 語法

```
(define (fact n)
  (if (= n 0)
      1
      (* n (fact (- n 1)))))
```

in C ?!

- Makefile

```
CFLAGS = -Wall \
  -D'define(ret, name, args, block) = ret name
args { return block; }' \
  -D'if(expr, block1, block2) = expr ? block1 :
block2' \
  -D'eq(a, b) = a == b' \
  -D'sub(a, b) = a - b' \
  -D'mul(a, b) = a * b' \
  -include "stdio.h" -include "stdlib.h"

TARGET=factorial
all:
    gcc -o $(TARGET) $(TARGET).c $(CFLAGS)
Clean:
    rm -f $(TARGET)
```

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.

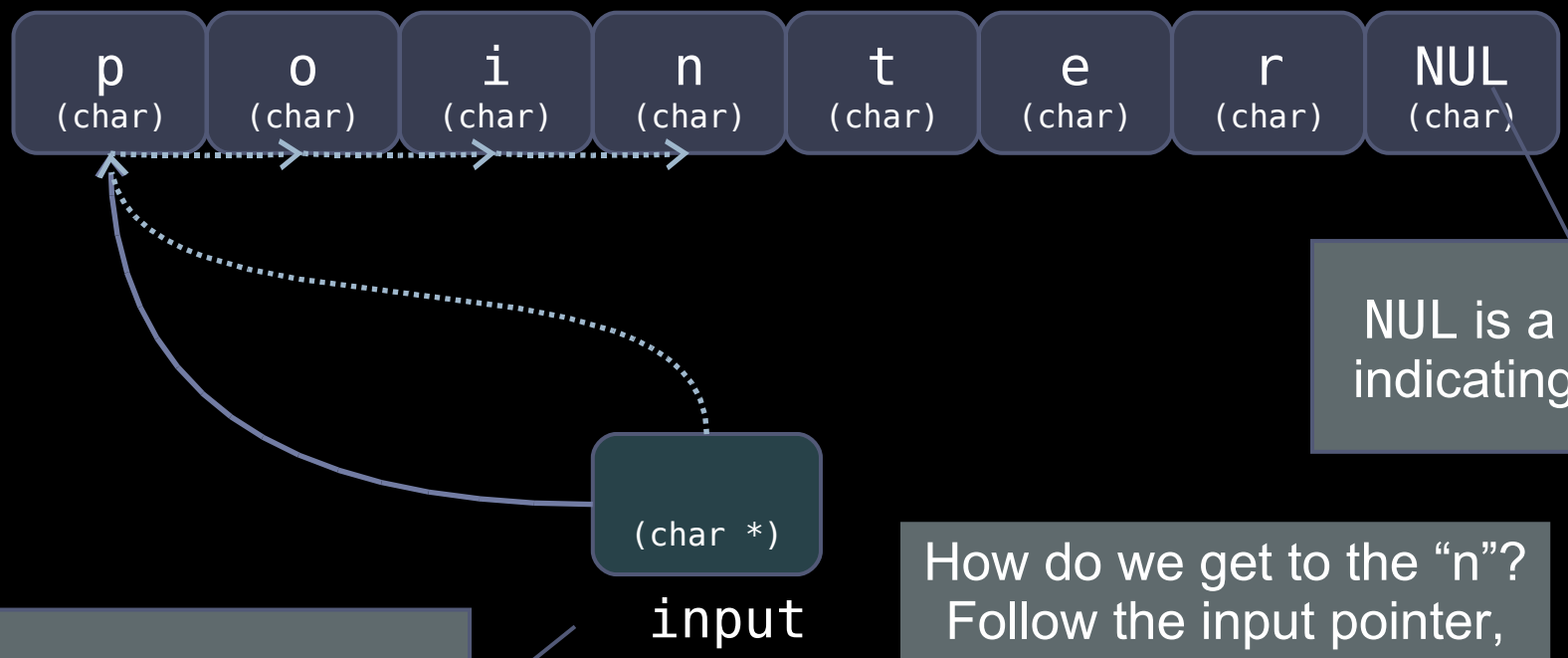
&(((s *)0)->m) computes the address of m

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

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

```
#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);
}
```

```
.LC0:
        .string "azerty"
        .data
        .align 4
        .type   str1, @object
        .size   str1, 4
str1:
        .long   .LC0
```

```
str2:
        .string "azerty"
```

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

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

參考資料

- 《美麗程式》*(Beautiful Code), O'Reilly*
- *JavaScript: The Definitive Guide*, D. Flanagan, O'Reilly, 5th edition
- *JavaScript Guide & Reference*, Mozilla Developer Center, http://developer.mozilla.org/en/docs/JavaScrip
- *ECMAScript Language Specification* — 3rd edition, http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf