政大資科系

作業系統

Operating System

廖峻鋒

cfliao@nccu.edu.tw

Operating System

Operating System Structure

Chun-Feng Liao

廖峻鋒

Department of Computer Science

National Chengchi University

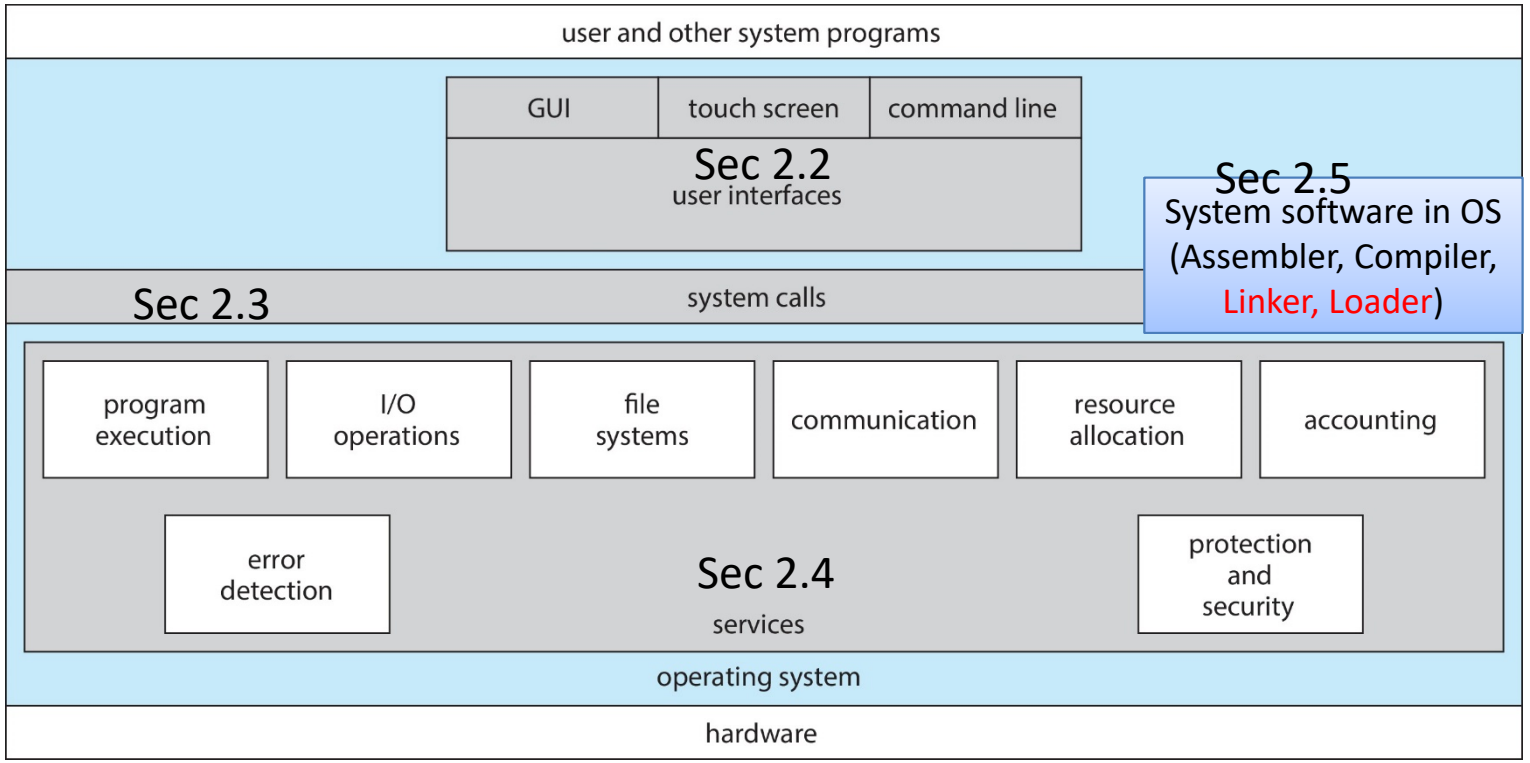
Outline

- OS Structure
  - User interface
  - System call
  - System services
  - System software: Linker and loaders
- Designing and implementing OS
  - Architectural styles
  - Build and debug tools

這一章說些什麼?

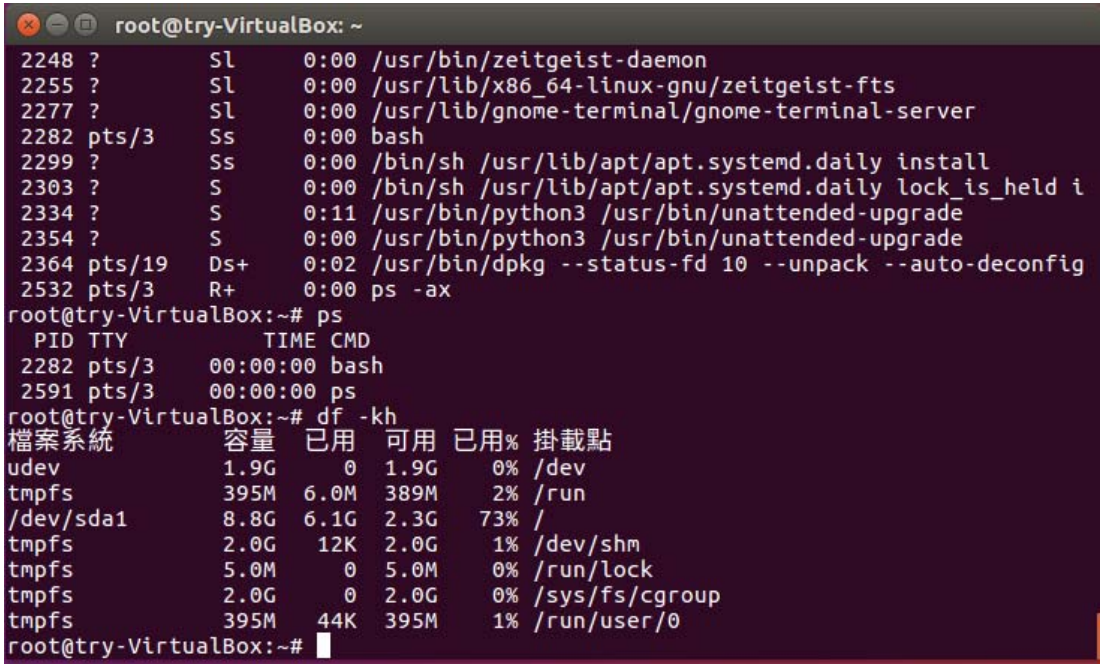
- 由軟體設計角度看OS結構
  - 認識System Call (函式呼叫方法)
  - 認識系統重要服務
  - 了解實作OS時要考量的設計議題

OS Structure



User Interface

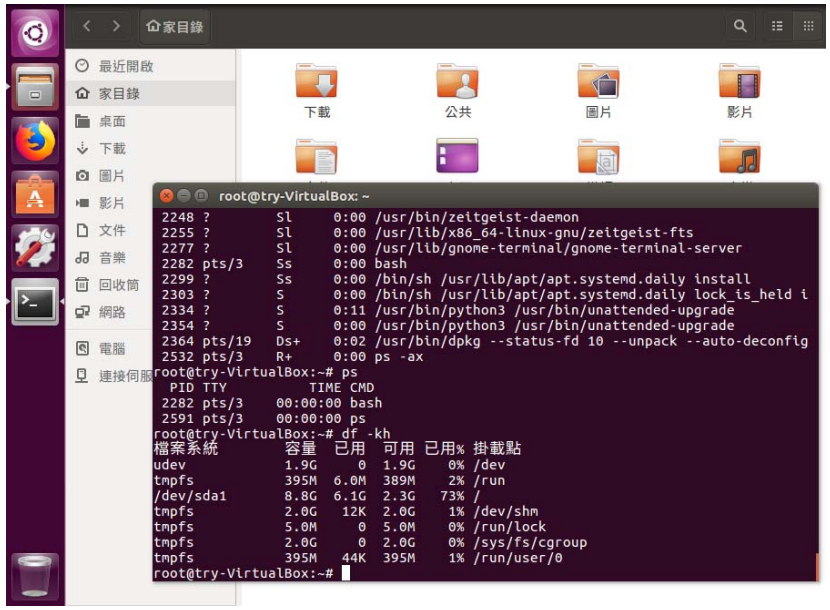
- CLI (Command Line Interface)
  - Shell: Command-line interpreter (CSH, BASH, KSH)
    - Used to parse, search and execute the input command
    - 內建指令: shell已了解的指令 (已預載入記憶體)
    - 外部指令: shell根據設定好的path，尋找第一個符合的指令



User Interface

- GUI (Graphic User Interface)
  - Invented in the 1970's at Xerox PARC
  - Icons represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions
- Most systems have both

  CLI and GUI



User Interface

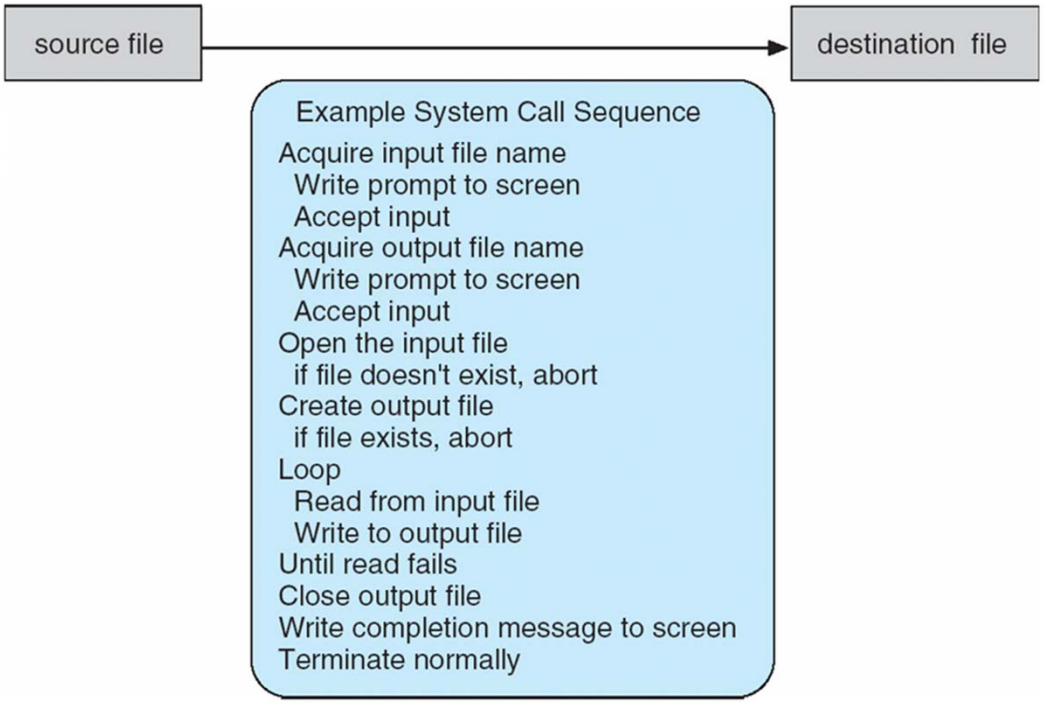
- TUI (Touch user interface)
  - GUI relies on sense of sight; TUI relies on sense of touch
  - Users interact by making gestures on the screen
  - Standard UI in mobile devices
- VUI (Voice user interface)
  - user interacts with a system through voice/speech

System Calls

- System calls
  - The interface between program and OS
  - An explicit request to the kernel made via a software interrupt
  - Available as assembly-language instructions
- Request OS services
  - Process control—abort, create, terminate process allocate/free memory
  - File management—create, delete, open, close file
  - Device management—read, write, reposition device
  - Information maintenance—get time or date
  - Communications—send receive message

System Calls

- Example: the cp command
  - cp in.txt out.txt



一個簡單動作，需要多個syscall完成

Demo: the “strace” command

- 寫作
  - nano hello.cc
- 編譯、產生可執行檔
  - g++ -o hello hello.cc
- 解析可執行檔
  - strace –o hello.log ./hello
- 結果
  - 約57個system calls
  - 另一個方法: strace –c ./hello

```
#include <iostream>

using namespace std;

int main(void) {

  cout<<"Hello!“<<endl;
  return 0;

}
```

Strace時間量測參數
-T; -tt

Demo: the “strace” command

- 寫作
  - nano hello.py
- 解析
  - strace –c python3 ./hello.py
- 結果
  - 約740個system calls

```
print(‘Hello’)
```

Demo: the “strace” command

- 寫作
  - nano hello.js
- 解析
  - strace –c node /hello.js
- 結果
  - 約547個system calls

```
console.log(‘Hello’)
```

API: Application Program Interface

- System calls are exposed via API
  - Wrappers of system calls
  - Ex:
    - Windows: Win32 API
    - Linux and MacOS: POSIX API
    - Java API、Node.js API
      - JVM and node bridge APIs to the underlying Win32 or POSIX APIs

Types of system call APIs

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

	Windows	Unix

Process control	`CreateProcess()`
	`ExitProcess()`
	`WaitForSingleObject()`
		`fork()`
		`exit()`
		`wait()`

File management	`CreateFile()`
	`ReadFile()`
	`WriteFile()`
	`CloseHandle()`
		`open()`
		`read()`
		`write()`
		`close()`

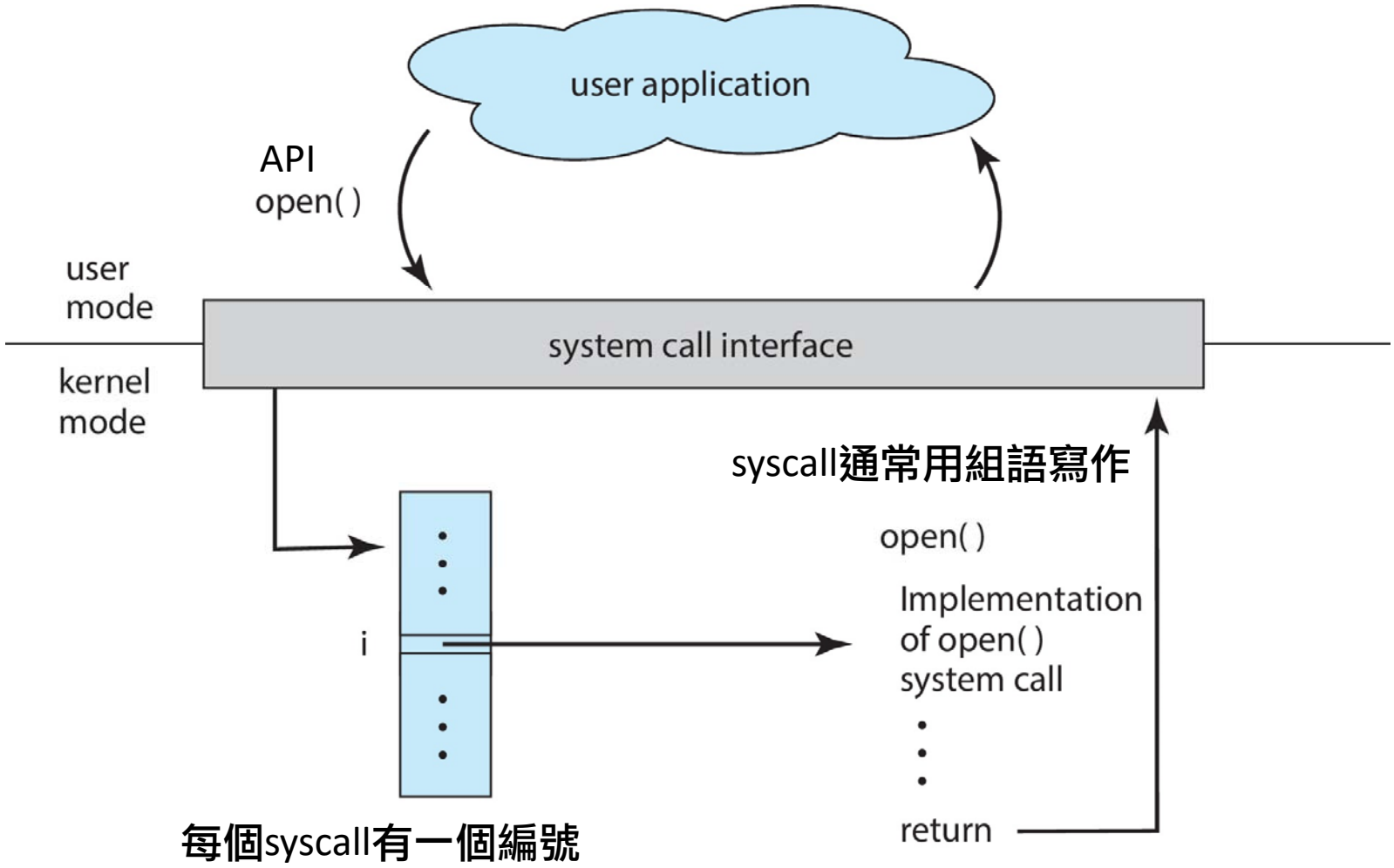
Device management	`SetConsoleMode()`
	`ReadConsole()`
	`WriteConsole()`
		`ioctl()`
		`read()`
		`write()`

Information maintenance	`GetCurrentProcessID()`
	`SetTimer()`
	`Sleep()`
		`getpid()`
		`alarm()`
		`sleep()`

Communications	`CreatePipe()`
	`CreateFileMapping()`
	`MapViewOfFile()`
		`pipe()`
		`shm_open()`
		`mmap()`

Protection	`SetFileSecurity()`
	`InitlializeSecurityDescriptor()`
	`SetSecurityDescriptorGroup()`
		`chmod()`
		`umask()`
		`chown()`

API – System Call – OS Relationship



System Service Table

Set $vo as

```
addi $v0,$zero, 1     # $v0 =1
syscall
addi $v0,$zero,10     # $v0 = 10
syscall
```

Service	System Call Code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		

Example of APIs

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

return value — function name — parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer into which the data will be read
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns –1.

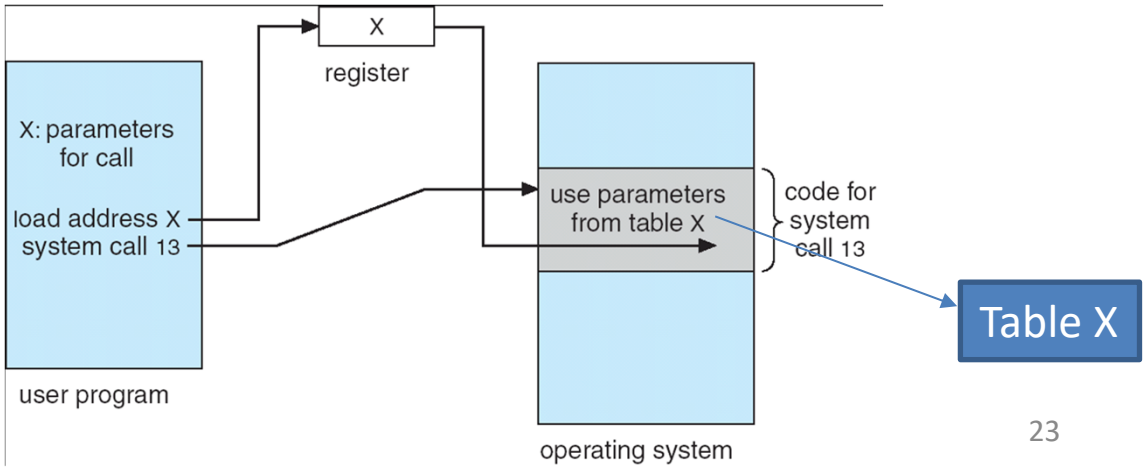
ssize_t = signed size_t

P.65下半

傳參數的方法

- Three general methods
  - Pass parameters in registers
    - In MIPS: $a0-$a3; 大於4個就要用其它方法
  - Push (store) the parameters onto the stack
    - See next slides
  - Store the parameters in a table in memory, and the table address is passed as a parameter in a register

TBC: 呼叫時，參數是傳到caller的stack，還是callee的stack?



Procedure相關暫存器

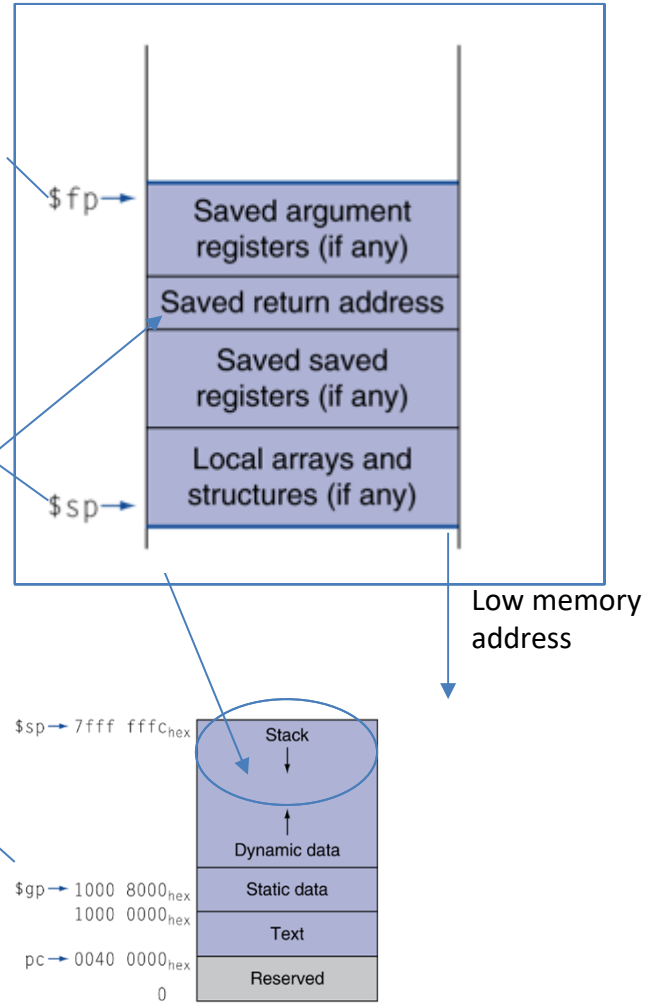
呼叫函式使用

- $a0 – $a3: arguments (reg's 4 – 7)
- $v0, $v1: result values (reg's 2 and 3)
- $t0 – $t9: temporaries 主要用來存區域變數
- $s0 – $s7: saved
- $gp: global pointer for static data (reg 28)
- $sp: stack pointer (reg 29)
- $fp: frame pointer (reg 30)
- $ra: return address (reg 31)

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

Procedure相關暫存器

$fp (30): points to the first word of a procedure frame
$sp (29): points to the top of stack

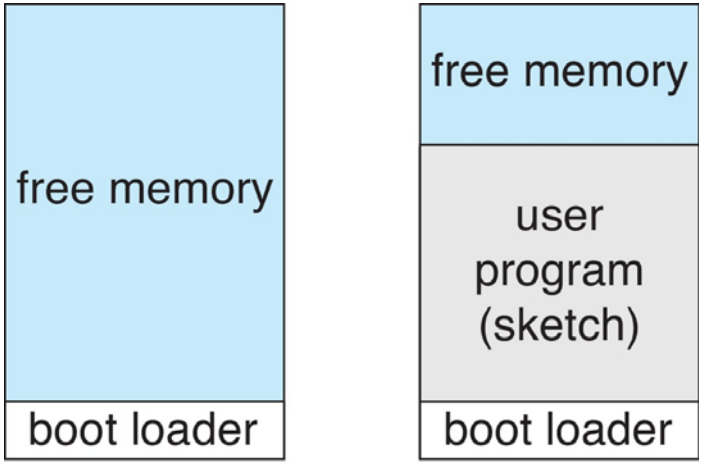


- $ra: return address of executing procedure (reg 31)
- $gp: global pointer for static data (reg 28)

  static variables

(single task example) Arduino process control

- Single-tasking and memory space
- Only one sketch in the memory
- Boot loader loads program; no OS

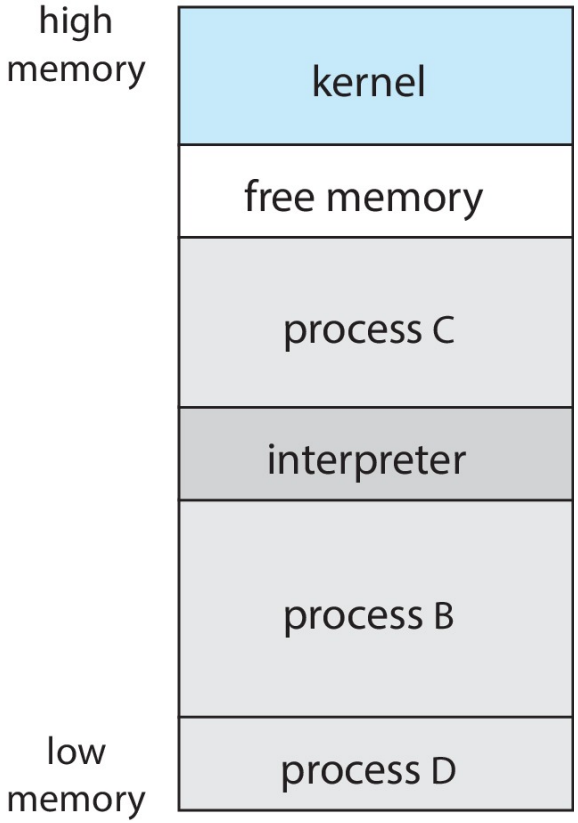


(a) At system startup

(b) running a sketch

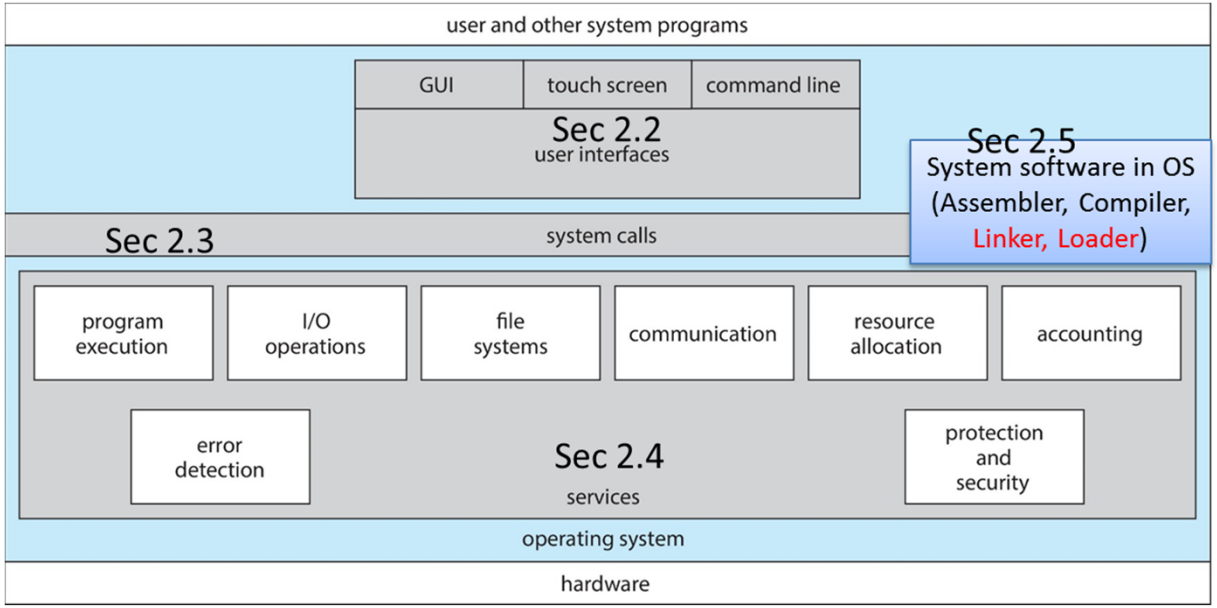
(multi-task example)
FreeBSD process control

- Use system call to create process
  - fork() to create process
  - exec() to load program into process
- Process exits with:
  - code = 0 : no error
  - code > 0 : error code
- More on chapter 3



System Services

- 目的
  - provide a convenient environment for program development and execution

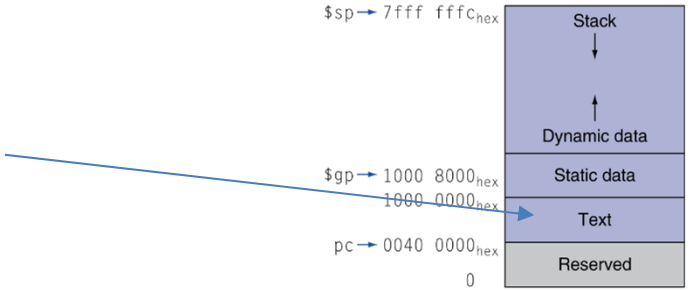


System Services

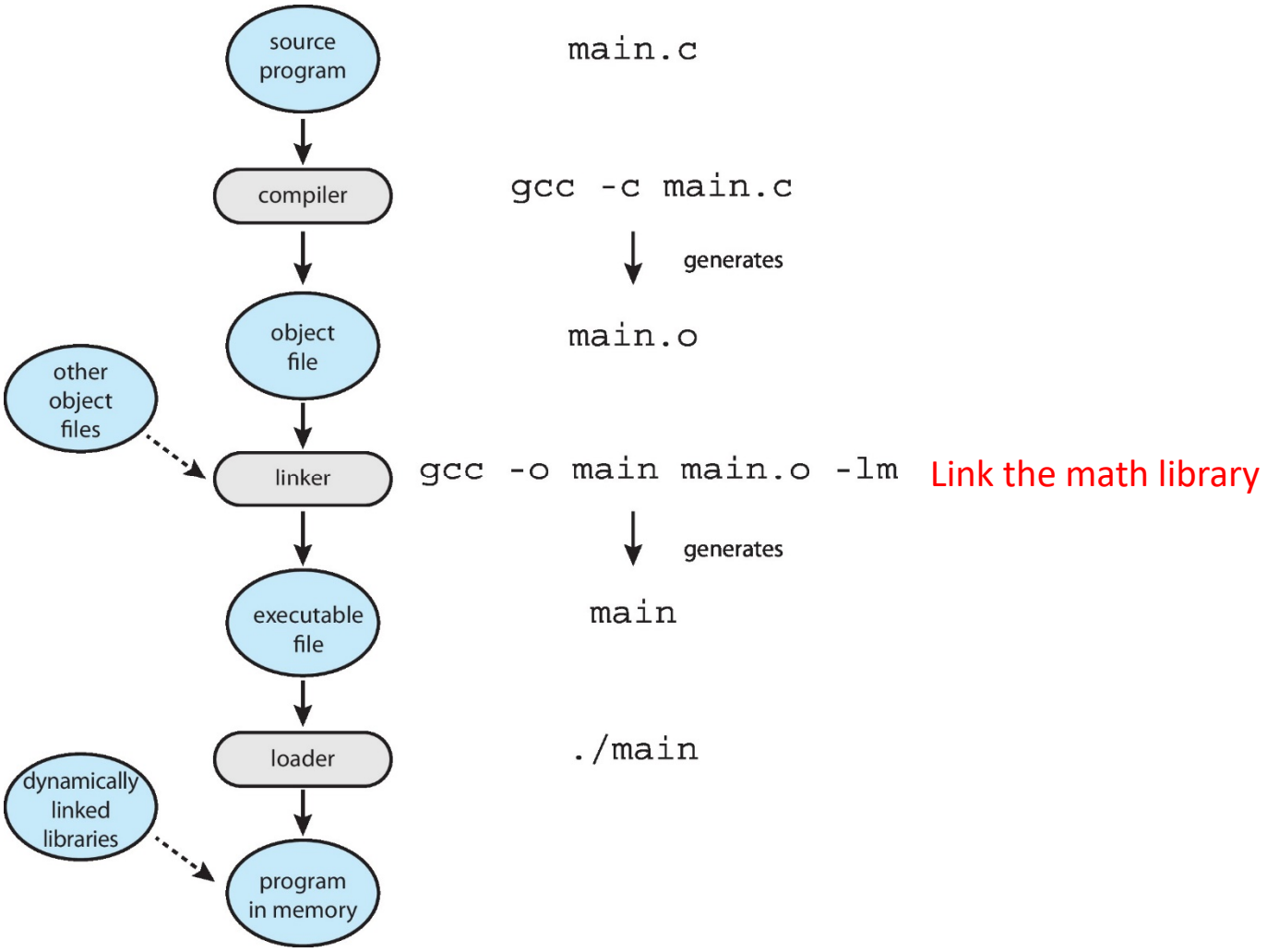
- Include utility-tools and daemons, such as:
  - File manipulation
  - Status information (ex: windows registry-regedit)
  - Programming language support
    - Compiling and linking
    - Loading and execution
  - Communications
    - IPC

Linkers and Loaders

- Linker
  - Combines relocatable object file
  - **編譯時期連結**: Produces single executable file
  - **執行時期連結**: DLL (Dynamically linked library)
    - Object file**透過**proxy (relocation info) **共享常用函式庫，省空間**
- Loader
  - Load the executable file into memory
  - Relocation
    - **由於**swapping、virtual memory**等機制，程式位址可能會搬動**
    - **程式中的記憶體位址都要用相對的，不能寫成絕對位址**



Linkers and Loaders



ELF Format

- ELF (Executable and Linkable Format)
  - The object file standard for UNIX and Linux
  - Include the compiled machine code and a symbol table containing metadata about functions and variables
- Demo
  - g++ hello.cc
  - file a.out (查詢a.out的屬性)
  - readelf –h -all a.out |more (顯示ELF資訊)

Designing OS

- Contemporary OSs are typically large and complex
  - Many software engineering practices have been extensively used in designing OS
  - E.g.: Architectural styles, software design principles
- Determining design goals
  - User goals –easy to use and learn, reliable, safe, and fast
  - System goals –easy to design, implement, and maintain, as well as reliable, error-free, and efficient
  - Tradeoffs are made based on goals

Key Techniques for Designing OS

- Abstraction
  - The fundamental principles humans use to cope with complexity
  - 中文語意較接近: 摘要、歸納、泛化
  - Ex: File, socket, process
- Encapsulation and information hiding
  - 儘量不被外部存取 → 控制元件間的相依性→ 降低改變成本
  - Reflection或多或少違反此法則
    - 要小心使用 (較適合在變動不大的底層服務/函式庫使用)

Key Techniques for Designing OS

- Modularization
  - meaningful decomposition of a software system and with its grouping into subsystems and components
  - Ex: LKM (Loadable kernel modules)
- Separation of Concerns
  - 一個單元一個責任
  - Do one thing and do it well
- Coupling and cohesion
  - 模組間low coupling
  - 模組內high cohesion

Key Techniques for Designing OS

- Separation of policy and mechanism
  - Policy: what to do
    - What to do
    - Context-sensitive decisions
    - 透過參數、設定檔或其它方式實現
  - Mechanism: how to do
  - 實作軟體時應該將未來易變與不易變的部份分開

Key Techniques for Designing OS

- Separation of Interface and Implementation
  - Interface
    - defines the functionality and specifies how to use it
    - Accessible by the clients
    - Ex: POSIX
  - Implementation
    - includes the actual code for the functionality
    - may also comprise additional functions and data structures that are only used internally
    - not accessible by the clients

Implementing OS

- Early OS
  - Using assembly
  - C
- Current OS uses a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++ (Android and Tiny OS: Java)
  - Scripting languages in Python and shell scripts

Debugging OS

- Process
  - Core dump
    - A snapshot of process in memory
  - strace
    - Trace system calls
    - strace a.out
- System
  - perf stat <command>
  - tcpdump/ wireshark
    - Dump network packets

P96 為什麼OS debug比起一般應用程式開發難

Performance Monitoring

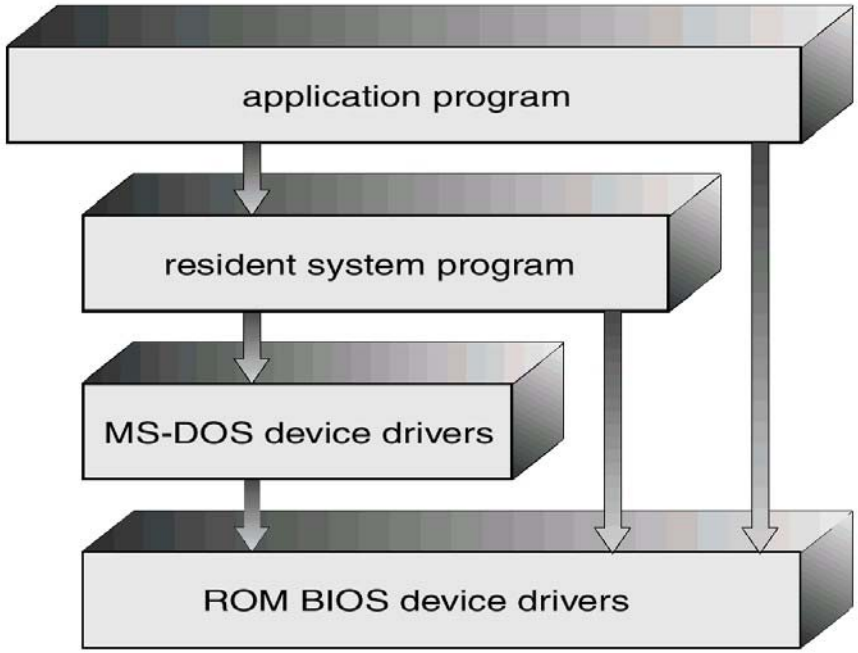
- Process information
  - ps/top
  - /proc
- Memory usage
  - vmstat
- Network and I/O
  - netstat/ iostat

Architectural Styles

- Monolithic
- Layered
- Microkernel
- Modules

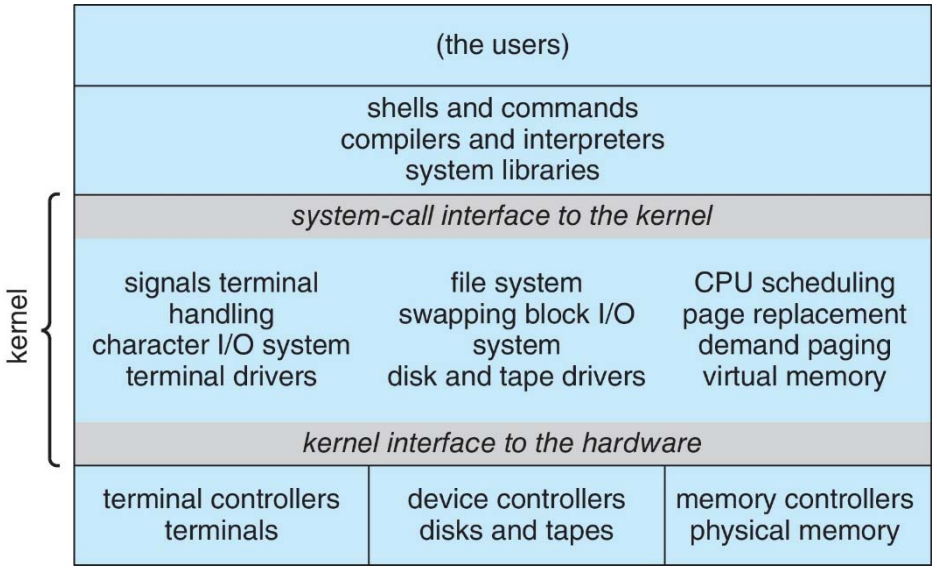
Monolithic Style

- Performance +; Flexibility: -; maintainability: -



MS-DOS

https://github.com/microsoft/MS-DOS

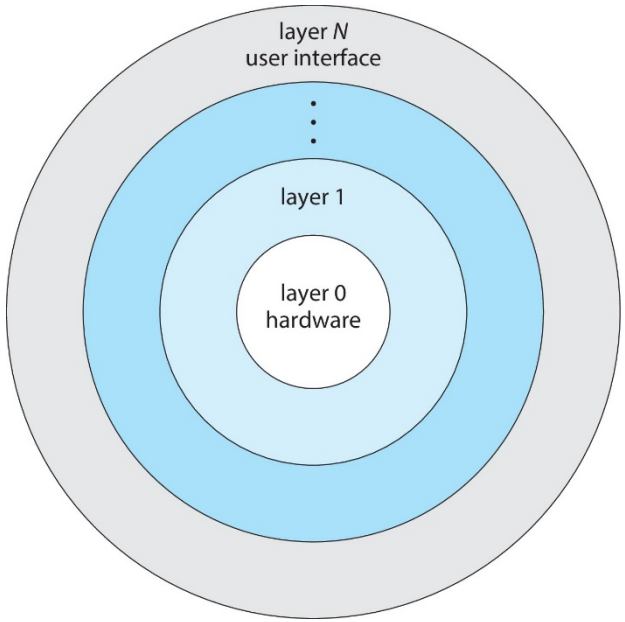


UNIX

https://github.com/Wangzhike/HIT-Linux-0.11

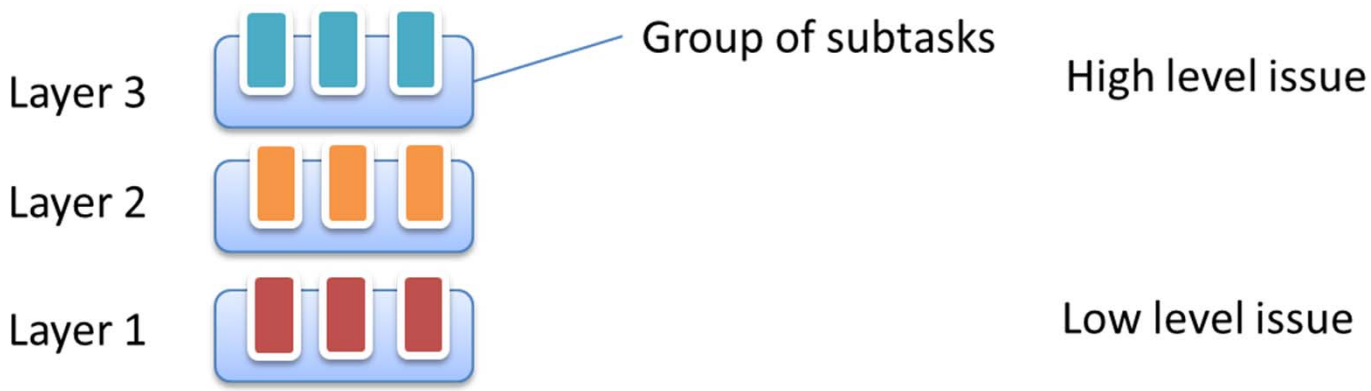
Layered OS Architecture

- Lower levels independent of upper levels
  - N^{th} layer can only access services provided by $0\sim(N-1)^{th}$ layer
- Pros: Easier debugging/maintenance
- Cons: less efficient, difficult to define layers



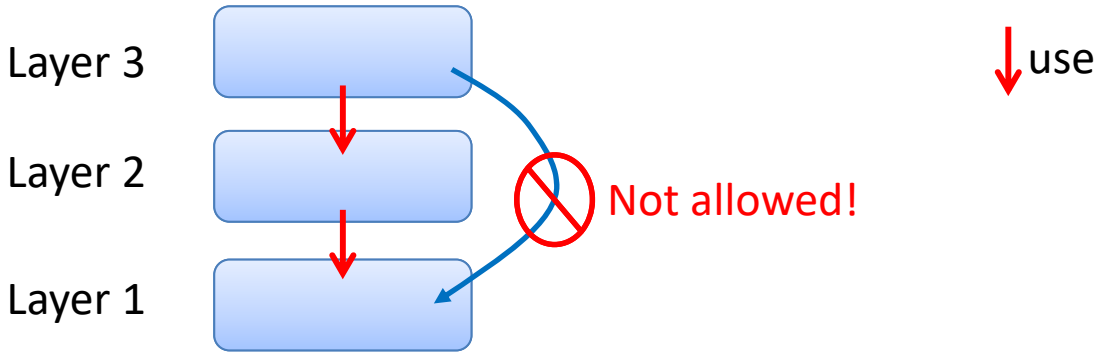
Layered Style

- Decompose functionalities into groups of subtasks
  - A group of subtask = A layer
  - Each layer is at a particular level of abstraction
- Lower levels independent of upper levels
  - Pros: Easier debugging/maintenance
  - Cons: less efficient; difficult to define layers



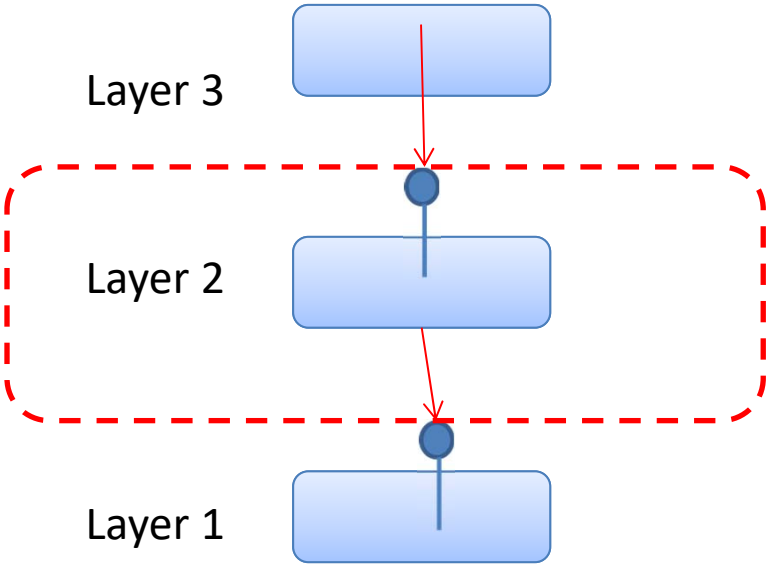
Layered Style

- Basic principle
  - Services of Layer i is only used by Layer i+1



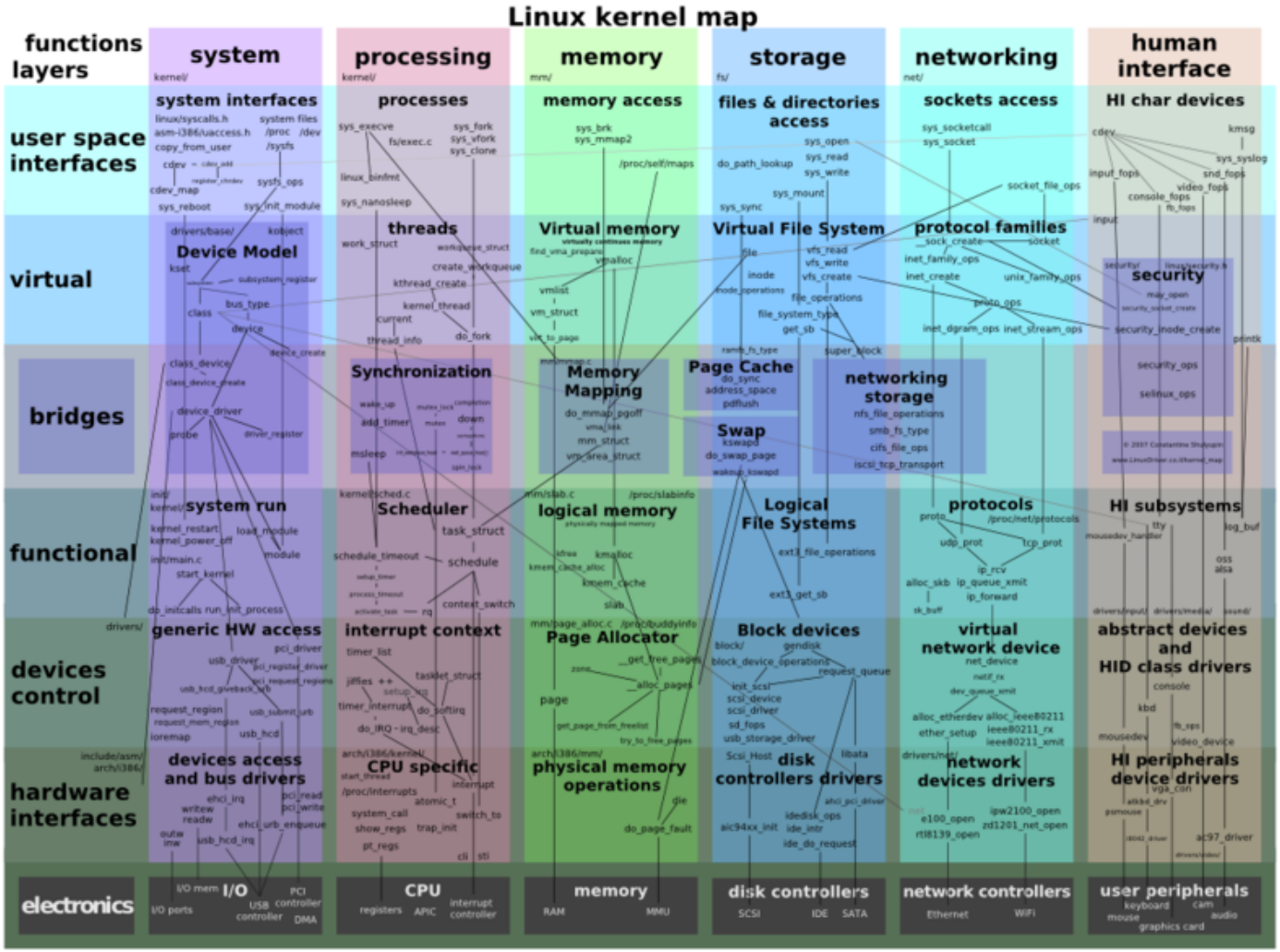
Layered Style

- Basic principle
  - Each layer
    - Provides services to the upper layer
    - Use services of the bottom layer



Known Uses

- Linux kernel architecture (a relaxed layer pattern)



Modules

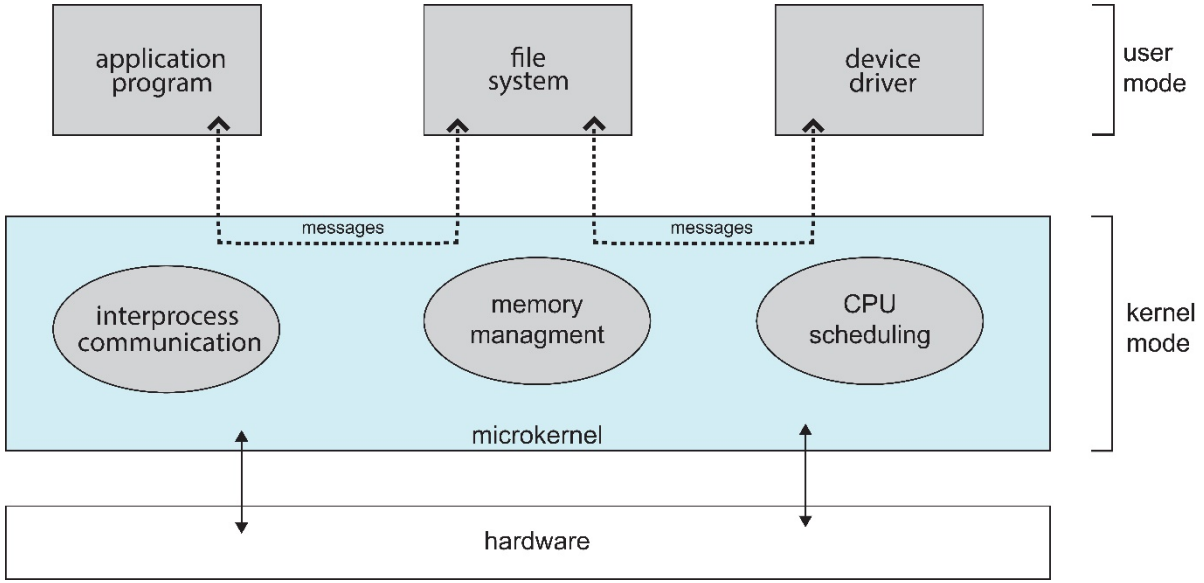
- Many modern operating systems implement **loadable kernel modules** (**LKMs**)
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Comments
  - Overall, similar to layers but with more flexible
  - No message passing
  - Ex: Linux, Solaris, etc

Microkernel Style

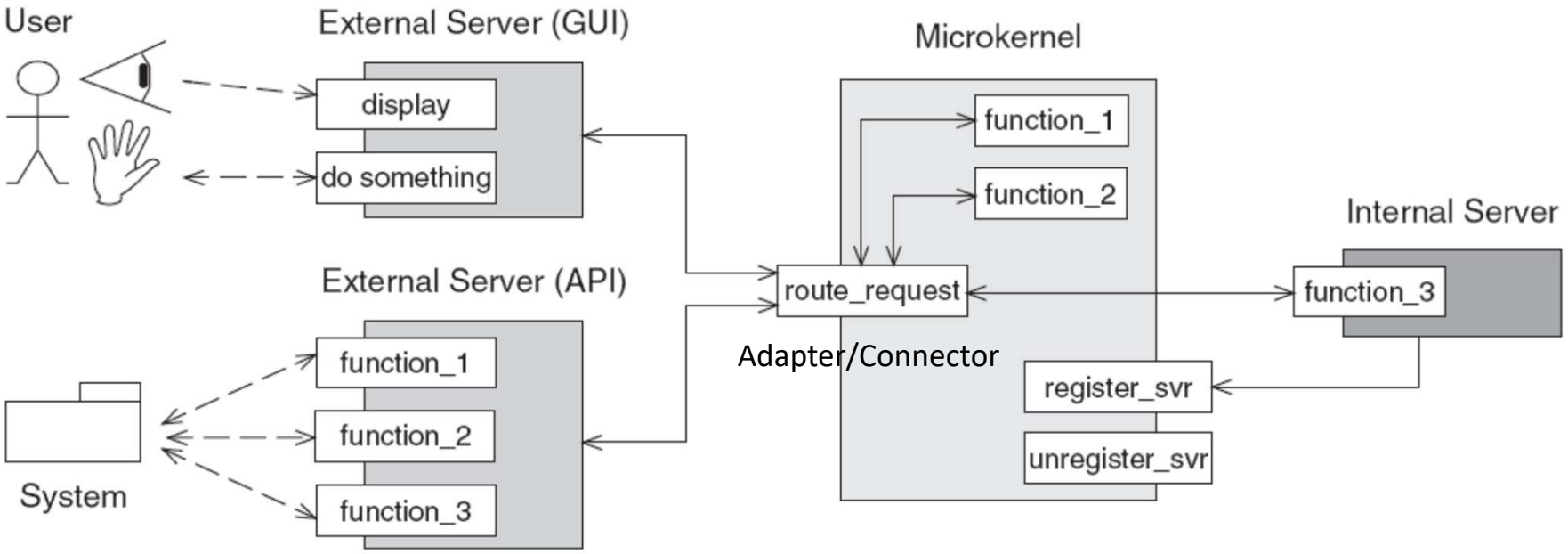
- Purpose
  - OS that adapts to changing requirements
    - More extensible, easier to be ported to new machines
    - More reliable, the failure of a module does not impact the kernel
  - Mid-1980s: Mach in CMU
- Main idea
  - Separate a minimal functional core from customizable components
  - The Microkernel serves as a hub for plugging in these extensions and coordinating their collaboration

Microkernel Style

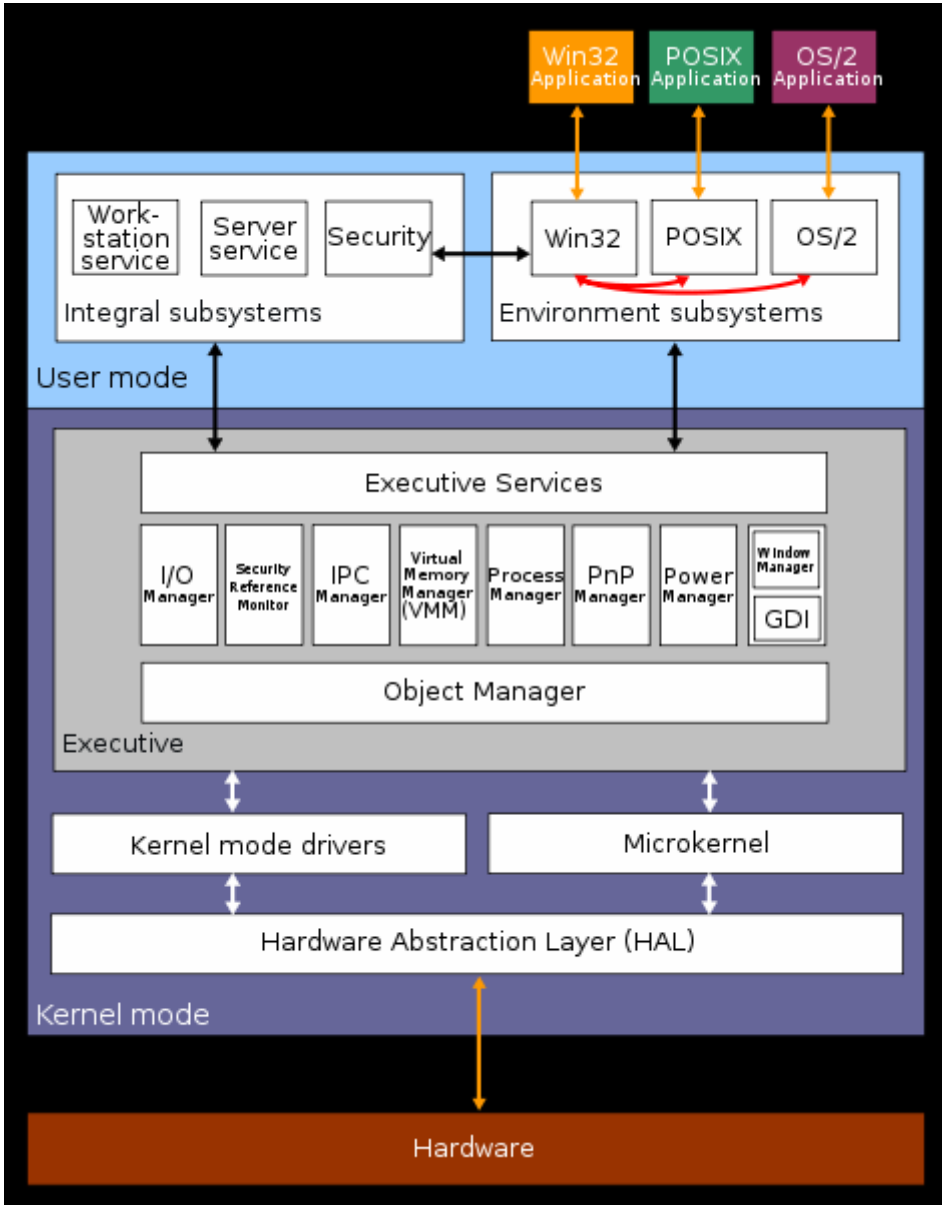
- Moves as much from the kernel into “user” space
- Communication is provided by message passing



Structure


User
External Server (GUI)
display
do something
Microkernel
function_1
function_2
Internal Server
External Server (API)
route_request
function_3
Adapter/Connector
function_1
function_2
function_3
register_svr
unregister_svr
System

Windows NT Architecture



External Servers

Microkernel

Internal Servers

Hybrid Systems

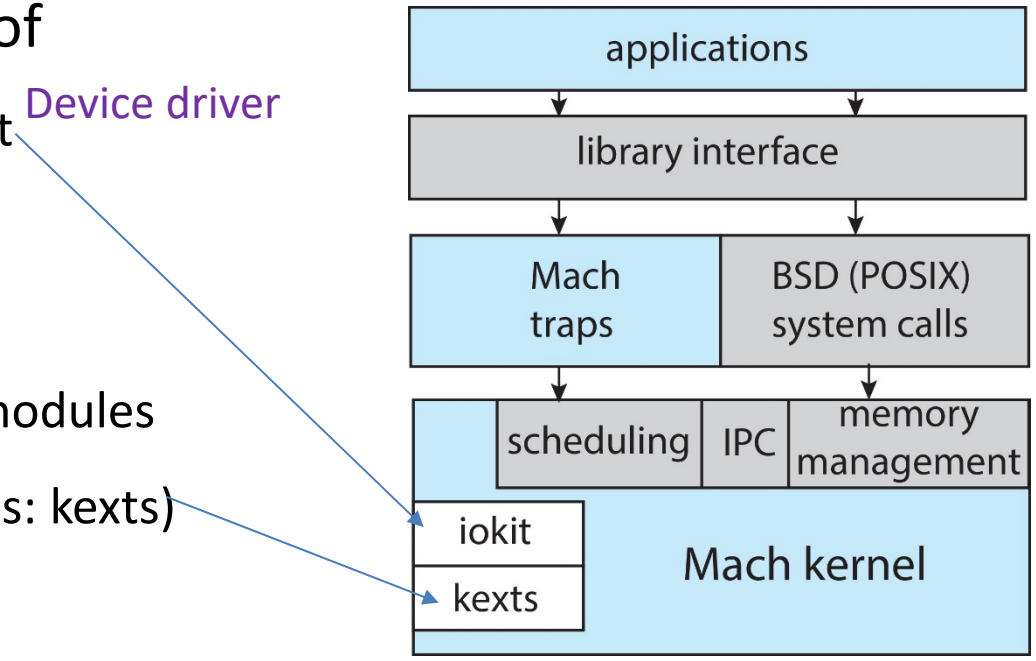
- Most modern OSs are actually not one pure model
  - Combines multiple approaches to address performance, security, usability needs
- Examples
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystems (***personalities)***
  - Apple Mac OS X: hybrid, layered, **Aqua** UI plus **Cocoa** programming environment
    - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

P89 Mac和iOS採用的Darwin核心，如何改善microkernel message passing效能低落的缺點 (message passed by ref)

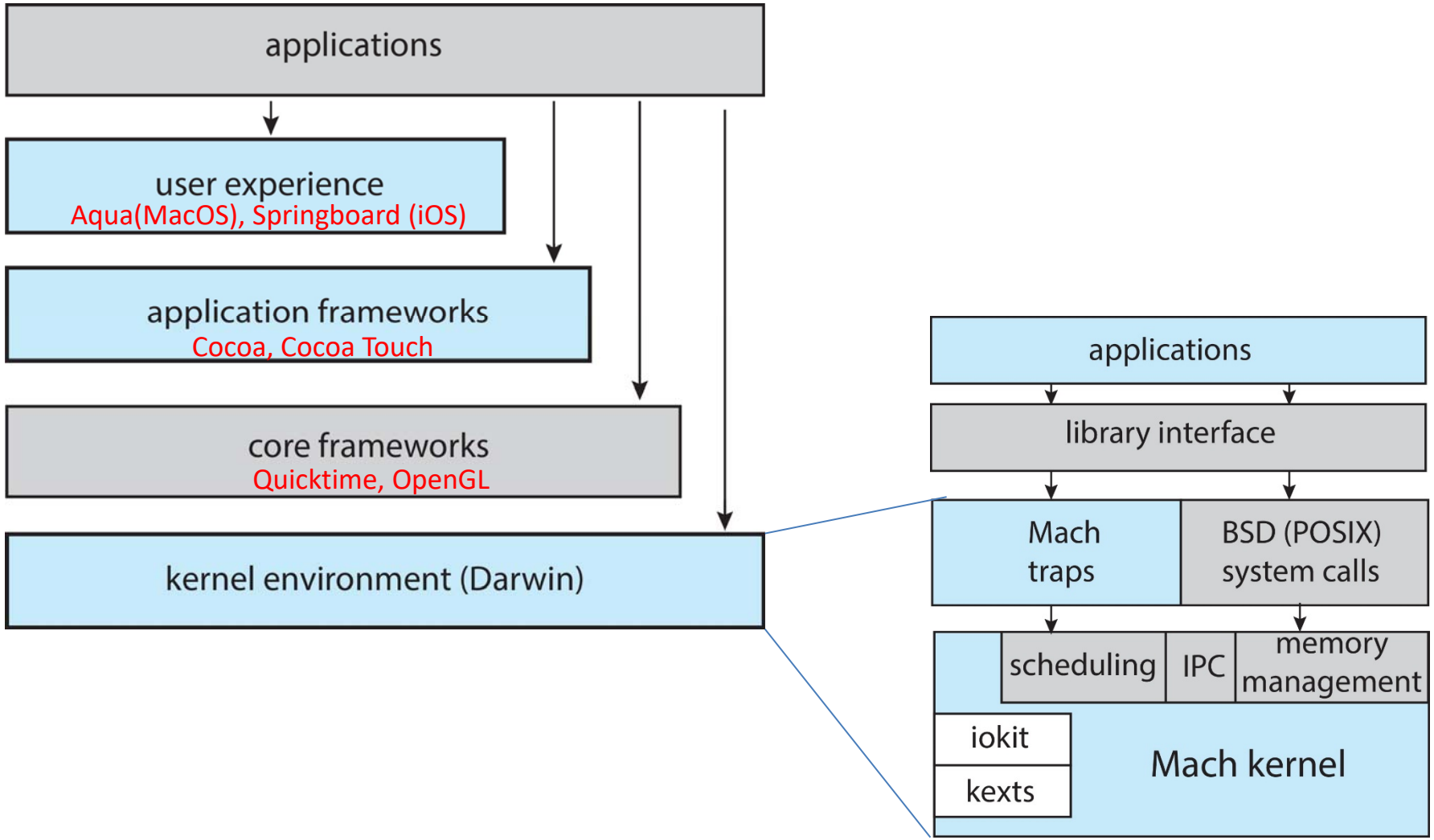
Case: macOS and iOS

- Apple Mac OS X: hybrid, layered
  - Kernel env. consisting of
    - Mach microkernel, iokit
    - BSD Kernel
      - Scheduling/IPC/MM
    - dynamically loadable modules

      (called kernel extensions: kexts)
    - System call
      - Mach traps
      - BSD POSIX



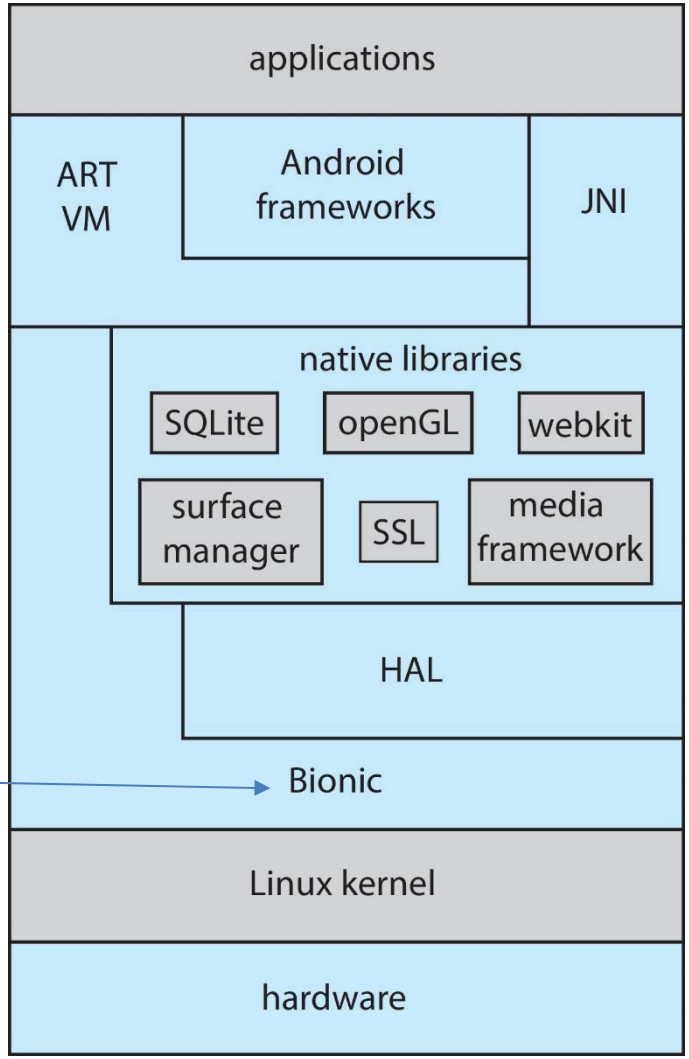
Case: macOS and iOS



Android

- Developed by Open Handset Alliance (mostly Google)
  - Open Source
- Similar stack to iOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - Apps developed in Java plus Android API
    - Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Android Architecture

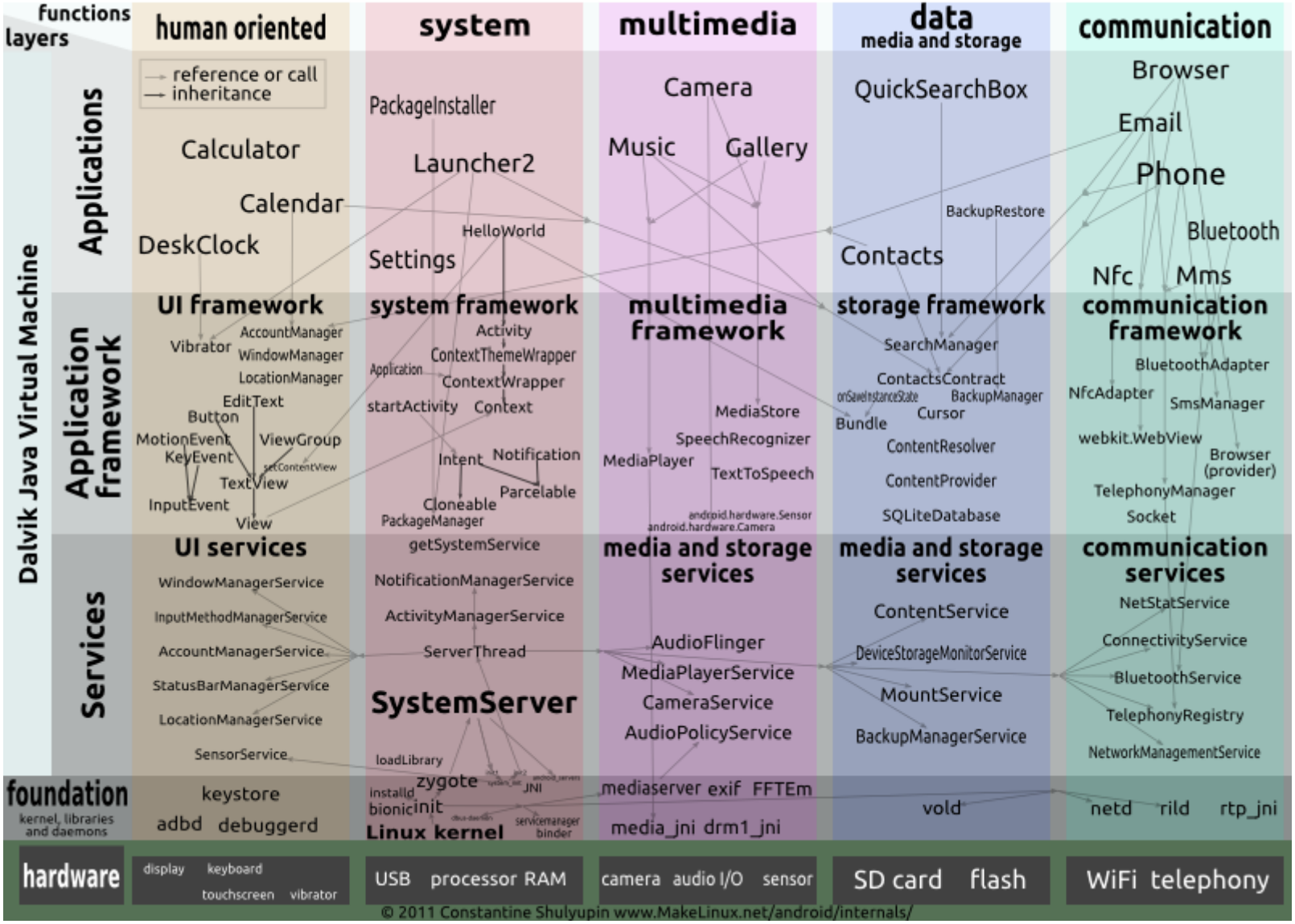


A compact C library

DRAFT version for discussion

Android Internals

API Level 9



課後閱讀

- P78 ABI是什麼?
- P89 Mac和iOS採用的Darwin核心，如何改善microkernel message passing效能低落的缺點
- P97 /proc 的說明和其作用
- P91 windows subsystem for linux
- P96 為什麼OS debug比起一般應用程式開發難
- P94 作業系統啟動程序
- P465 Windows啟動程序 (MBR)

Q & A