CH3、OS

OS 之 Develop

目錄：

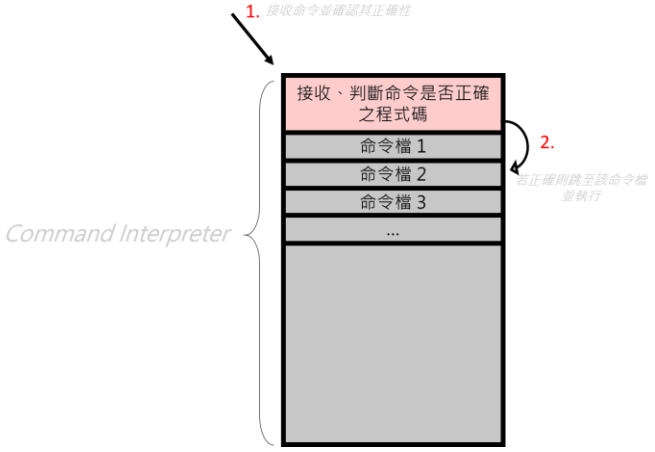
Command Interpreter

(一) Def：作為 User 與 OS 之溝通介面，主要工作有：

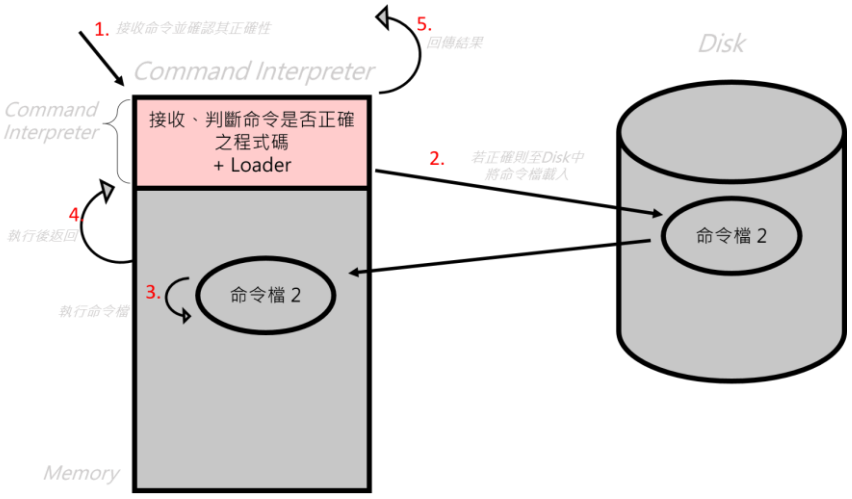
1. 接收 User 之 input commands
2. 判斷命令格式、參數正確與否
3. 若正確，則執行對應的命令副程式(Command Routines)

(二) 設計問題

1. 問題一：命令副程式是否應該包含在 Command Interpreter 中？
   (1) 包含(ex：DOS 內建命令)
   優點：命令執行啟動迅速(因為命令檔都已存在 Memory 中、執行時直接 Jump 即可)
   缺點：不常使用的命令檔佔用 Memory 空間、命令的增減困難(因為 Command Interpreter 要隨之變更)



(2) 不包含(ex：DOS 外建命令)：優、缺點與上述(1)相反



2. 問題二：Command Interpreter 與 kernel 之間的關係為何？

Tightly-Coupled：例：Windows、Apple…

優點： User 操作介面、命令一致

缺點： Command Interpreter 不易變更*(∵kernel 也要隨之變更)*

User 較難定義自己的操作介面

Loosely-Coupled：例：UNIX

UNIX 的 Shell 與 kernel 是獨立的*(同樣指令可因機器不同而有不同功能、定義)*

API

一般程式大都傾向使用 High-level Application Program Interface(API)，而不是直接使用 System Call，因為 API 較簡單也較易移植*(可以任意參數對應，無需克意查表)*。但在需要特殊服務*(ex：特權指令)*時，依然需要 System Call

API	function	使用語言/機器	程式碼
C standard library API	Fopen("w+"…)	C language	int printf(const char* format, ...);
WIN32 API	CreateFile()	Windows >= win4.0, >= 95	BOOL WINAPI WriteFile(_In_ HANDLE hFile, _In_ LPCVOID lpBuffer, _In_ DWORD nNumberOfBytesToWritten, _Out_opt_ LPDWORD lpNumberOfBytesWritten, _Inout_opt_ LPOVERLAPPED lpOverlapped);
Kernel API	NTCreateFile()	WinNT, 2k, XP, vista	NTSTATUS NtWriteFile(HANDLE hFile, HANDLE hEvent, PIO_APC_ROUTINE apc, void* apc_user, PIO_STATUS_BLOCK io_status, const void* buffer, ULONG length, PLARGE_INTEGER offset, PULONG key)
System Call	Int 2e	X86 machine instruction	Err… Not disclosed by Microsoft… May looks like: mov eax, <service #> lea edx, <addrof 1st arg> int 2e

而常用的 API 有三種：

1. Win32 API：Windows
2. POSIX API：POSIX-based system(UNIX、Linux、Mac OS X…)
3. JAVA API：Java Virtual Machine(JVM)

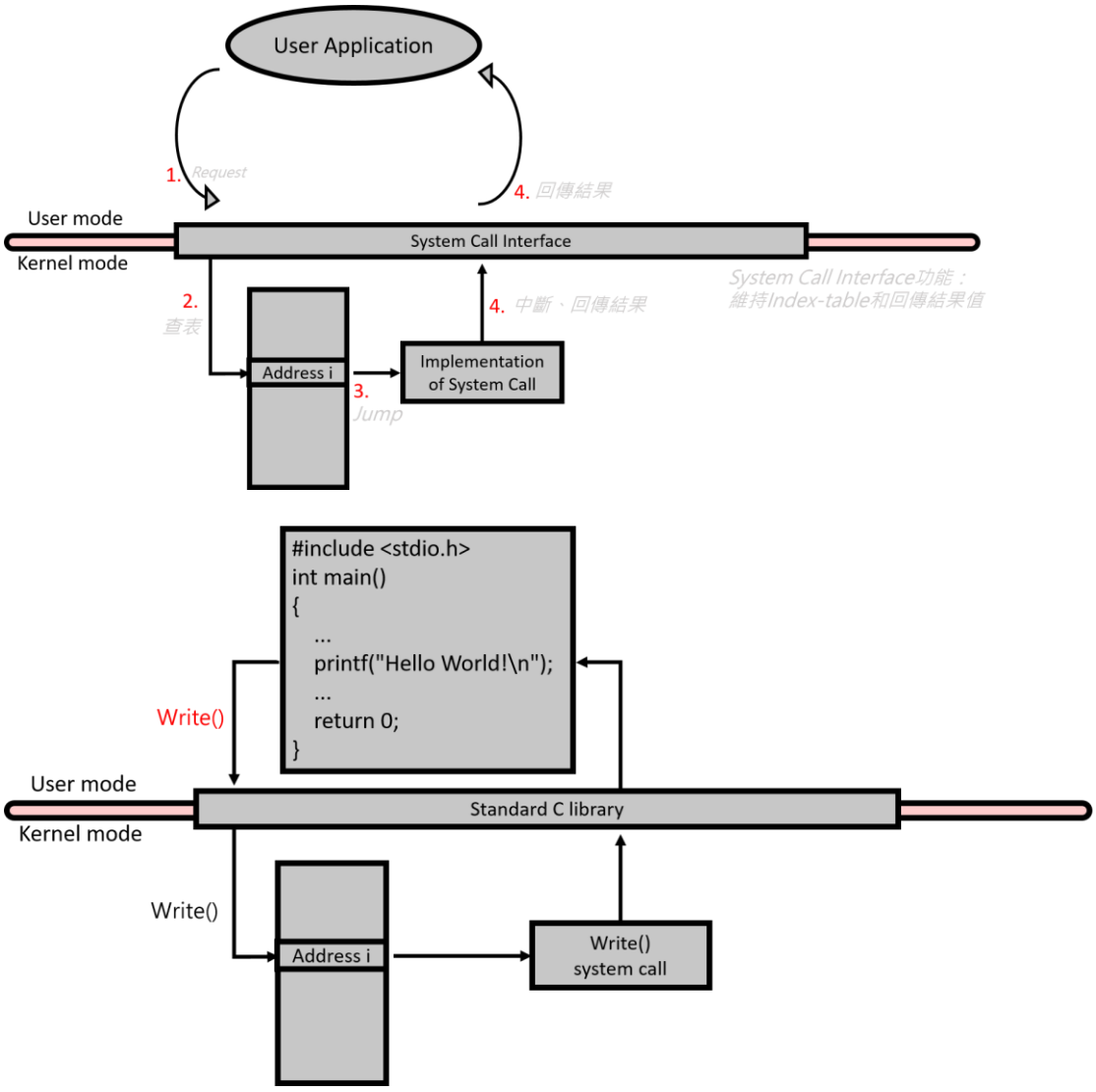
System Call

(一)Def：作為執行中 user process 與 kernel 之間的溝通界面，當 user process 需要 OS 提供某種服務時，會先 Trap 通知 OS，並代入 System Call ID (No.)及所需參數，Then OS 執行對應的 System Call，完成後，將服務結果 return to user process.

It's a programming "Interface" to the services provided by the OS.

比喻：銀行活動

Trap=服務鈴 or 號碼牌；System Call=服務項目(ex：1 號領錢、2 號存錢…)
*當客戶需要銀行職員提供領錢服務，會**先按鈴**(or 拿號碼牌)通知銀行職員，並**告知服務項目**(領錢為 1 號代碼)及**帳號**、**提領數目**…等資料，而後銀行職員**根據代碼**提供對應服務，至其**金庫**帳戶領取該金額數目給客戶*



(二)System Call 種類：

	Windows	UNIX	簡述
Process Control	CreateProcess() ExitProcess() WaitForSingleObjcet()	fork() exit() wait()	建立、刪除、暫停、恢復 process、set/read attribute
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()	建立、讀寫、開關、命名、改名、複製
Device Manipulation	SetConsolMade() ReadConsole() WriteConsole()	ioctle() read() write()	
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()	取得系統時間、日期、取得 process 屬性
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()	Process 之間的通訊，而且只針對 Message Passing 方式提供服務
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()	HW Resource Protection、File access control

List by System Call Number

00 sys_setup [sys_ni_syscall]	41 sys_dup	81 sys_setgroups	121 sys_setdomainname	161 sys_sched_rr_get_interval
01 sys_exit	42 sys_pipe	82 sys_select [old_select]	122 sys_uname [sys_newuname]	162 sys_nanosleep
02 sys_fork	43 sys_times	83 sys_symlink	123 sys_modify_ldt	163 sys_mremap
03 sys_read	44 sys_prof [sys_ni_syscall]	84 sys_oldlstat [sys_lstat]	124 sys_adjtimex	164 sys_setresuid (2.2+)
04 sys_write	45 sys_brk	85 sys_readlink	125 sys_mprotect	165 sys_getresuid (2.2+)
05 sys_open	46 sys_setgid	86 sys_uselib	126 sys_sigprocmask	166 sys_vm86
06 sys_close	47 sys_getgid	87 sys_swapon	127 sys_create_module	167 sys_query_module (2.2+)
07 sys_waitpid	48 sys_signal	88 sys_reboot	128 sys_init_module	168 sys_poll (2.2+)
08 sys_creat	49 sys_geteuid	89 sys_readdir [old_readdir]	129 sys_delete_module	169 sys_nfsservctl (2.2+)
09 sys_link	50 sys_getegid	90 sys_mmap [old_mmap]	130 sys_get_kernel_syms	170 sys_setresgid (2.2+)
10 sys_unlink	51 sys_acct	91 sys_munmap	131 sys_quotactl	171 sys_getresgid (2.2+)
11 sys_execve	52 sys_umount2 [sys_umount] (2.2+)	92 sys_truncate	132 sys_getpgid	172 sys_prctl (2.2+)
12 sys_chdir	53 sys_lock [sys_ni_syscall]	93 sys_ftruncate	133 sys_fchdir	173 sys_rt_sigreturn (2.2+)
13 sys_time	54 sys_ioctl	94 sys_fchmod	134 sys_bdflush	174 sys_rt_sigaction (2.2+)
14 sys_mknod	55 sys_fcntl	95 sys_fchown	135 sys_sysfs	175 sys_rt_sigprocmask (2.2+)
15 sys_chmod	56 sys_mpx [sys_ni_syscall]	96 sys_getpriority	136 sys_personality	176 sys_rt_sigpending (2.2+)
16 sys_lchown	57 sys_setpgid	97 sys_setpriority	137 sys_afs_syscall [sys_ni_syscall]	177 sys_rt_sigtimedwait (2.2+)
17 sys_break [sys_ni_syscall]	58 sys_ulimit [sys_ni_syscall]	98 sys_profil [sys_ni_syscall]	138 sys_setfsuid	178 sys_rt_sigqueueinfo (2.2+)
18 sys_oldstat [sys_stat]	59 sys_oldolduname	99 sys_statfs	139 sys_setfsgid	179 sys_rt_sigsuspend (2.2+)
19 sys_lseek	60 sys_umask	100 sys_fstatfs	140 sys__llseek [sys_lseek]	180 sys_pread (2.2+)
20 sys_getpid	61 sys_chroot	101 sys_ioperm	141 sys_getdents	181 sys_pwrite (2.2+)
21 sys_mount	62 sys_ustat	102 sys_socketcall	142 sys__newselect [sys_select]	182 sys_chown (2.2+)
22 sys_umount [sys_oldumount]	63 sys_dup2	103 sys_syslog	143 sys_flock	183 sys_getcwd (2.2+)
23 sys_setuid	64 sys_getppid	104 sys_setitimer	144sys_msync	184 sys_capget (2.2+)
24 sys_getuid	65 sys_getpgrp	105 sys_getitimer	145 sys_readv	185 sys_capset (2.2+)
25 sys_stime	66 sys_setsid	106 sys_stat [sys_newstat]	146 sys_writev	186 sys_sigaltstack (2.2+)
26 sys_ptrace	67 sys_sigaction	107 sys_lstat [sys_newlstat]	147 sys_getsid	187 sys_sendfile (2.2+)
27 sys_alarm	68 sys_sgetmask	108 sys_fstat [sys_newfstat]	148 sys_fdatasync	188 sys_getpmsg[sys_ni_syscall]
28 sys_oldfstat [sys_fstat]	69 sys_ssetmask	109 sys_olduname [sys_uname]	149 sys__sysctl [sys_sysctl]	189 sys_putpmsg [sys_ni_syscall]
29 sys_pause	70 sys_setreuid	110 sys_iopl	150 sys_mlock	190 sys_vfork (2.2+)
30 sys_utime	71 sys_setregid	111 sys_vhangup	151 sys_munlock	
31 sys_stty [sys_ni_syscall]	72 sys_sigsuspend	112 sys_idle	152 sys_mlockall	
32 sys_gtty [sys_ni_syscall]	73 sys_sigpending	113 sys_vm86old	153 sys_munlockall	
33 sys_access	74 sys_sethostname	114 sys_wait4	154 sys_sched_setparam	
34 sys_nice	75 sys_setrlimit	115 sys_swapoff	155 sys_sched_getparam	
35 sys_ftime [sys_ni_syscall]	76 sys_getrlimit	116 sys_sysinfo	156 sys_sched_setscheduler	
36 sys_sync	77 sys_getrusage	117 sys_ipc	157 sys_sched_getscheduler	
37 sys_kill	78 sys_gettimeofday	118 sys_fsync	158 sys_sched_yield	
38 sys_rename	79 sys_settimeofday	119 sys_sigreturn	159 sys_sched_get_priority_max	
39 sys_mkdir	80 sys_getgroups	120 sys_clone	160 sys_sched_get_priority_min	
40 sys_rmdir				

例：

```
Void Test (int a){
    int *p, k;
    p=new(int);
    printf("input the number:");
    scanf("%d",&k);
    *p=a+k;
    fopen(xxx);
    fwrite(xxx);
    fclose(xxx);
}
```

(跟硬體有關的，即 process、file…等，就會有 System Call： 黑粗體)

(三)System Call 參數(parameters)的 3 種傳送方式：

[法一]利用 Register 保存參數

優點： 1.Simple

2.存取速度最快(∵without memory access)

缺點： 不適用於大量參數之情況

[法二]利用 Memory 以一個 block/table 儲存這些參數，且將這些參數的起始位址，置於 1 個 Register 中，pass 給 OS

優點：適用大量參數

缺點：存取速度較慢且操作較為麻煩

[法三]利用 Stack 將參數 push 入此 Stack，OS 再 pop from Stack 以取得參數

優點：也適用大量參數、操作也 Simple

缺點：Stack 空間可能要大些

比喻：用餐

1. *單點*
2. *套餐*
3. *吃到飽*

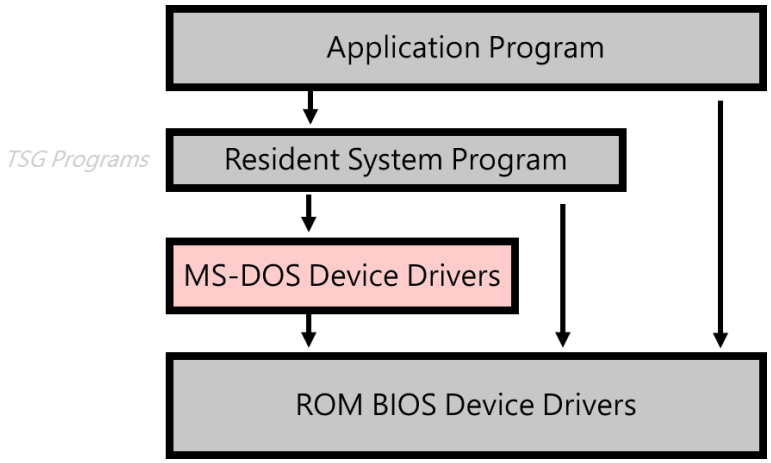
OS 之系統架構(Structure)分類

1. Simple：無 kernel 概念，常見於嵌入式系統，故如果單就數量而言，是最多機器使用的架構*(包括如電鍋…等的一般家電)*

   優點：使用者可以任意存取硬體、彈性大

   缺點：無保護機制，相當容易 crash 而要重新開機 reset*(ex：非法記憶體存取)*

   例：MS-DOS



2. More Complex than Simple (monolithic)

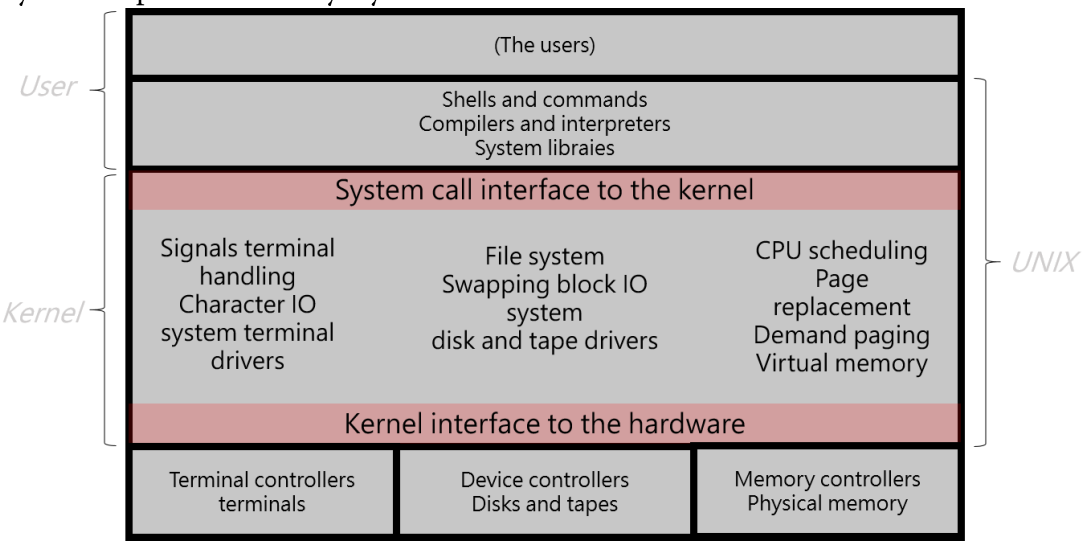
   首先有 kernel 的概念，雖然概念古老，但還是現今絕大多數電腦使用的架構*(如果不分出 Hybrid 這類的話)*，故曾經有不少論戰

   UNIX(limit by HW functionality)：The original UNIX had limit structuring.

   The UNIX 包含 2 個 separate parts：

   1. System Programming
   2. The kernel

Beyond simple but not fully layer

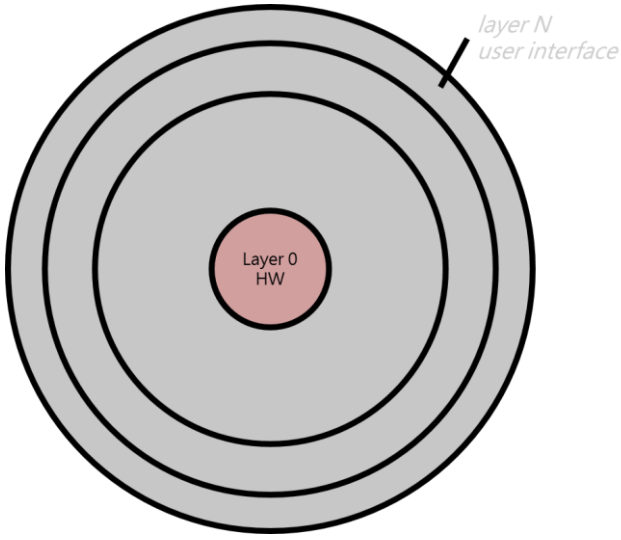


3. Layer approach(純理論，無實作出機器)

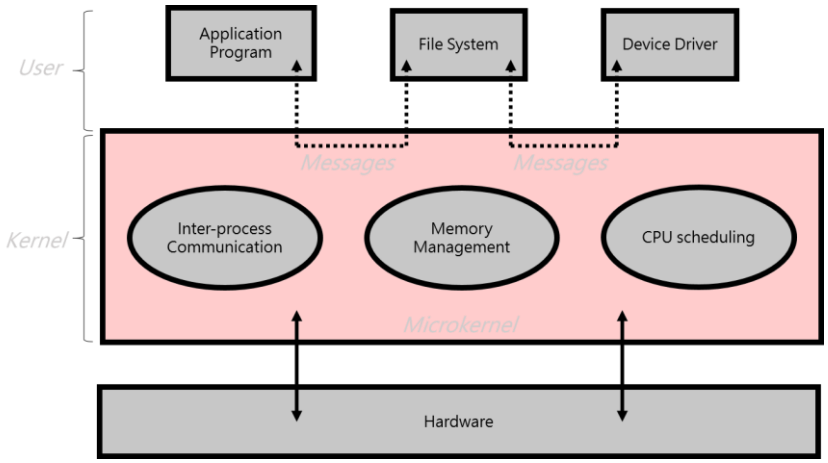
   Def：

   1. 採取 Top-Down 方式切割系統功能/元件，以降低複雜度
   2. 元件/模組之間依呼叫關系分層，即上層可使用下層，但下層不可以使用上層的功能
   3. 使用 Bottom-Up 的方式進行除錯 Debugging
   4. Layer 的畫分並沒有明確的規定；
      Layer 0 層=HW、最外(n)層=User Interface
   5. 優點：降低設計的複雜度有助於分工，測試、除錯、維持、擴展容易
      缺點：　1.很難作到很精准層次畫分
      2.若 Layer 層次數過多，中斷會大量增加，則效能差
      3.分層困難

      一般情況下，Disk 應該於 Memory 外層，但當需要進行 Virtual Memory 時，又應該是 Memory 在外層、Disk 在內層，為矛盾，故難以正確分層



4. Microkernel：
   (一)由 CMU(卡內基美隆大學)率先提出。代表產品：Mach OS
   (二)Def：盡可能地將 kernel 中一些 Non-essential Services(比較不是那麼重要、必要)自 kernel 中移除，改成在 user site 提供服務以 System Program 方式存在，如此一來，可以得到一個較小的 kernel
   一般而言，microkernel 提供下列 3 個 minimum services：
      1. process control
      2. memory management(不包括 Virtual Memory)
      3. process communication(提供 Message Passing 服務而已，Share Memory 是 programmer 的責任)



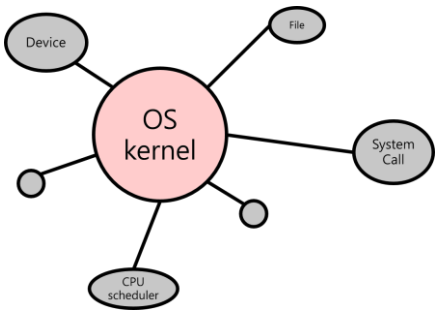
Messages 通道要用到時建立，但也有可能有兩種元件無法溝通

   (三)Benefit(好處)：
      1. Easier to extend a microkernel
      2. Easier to port the OS to new architecture
      3. More Reliable
      4. More Secure
         說明如下：
         1. 服務的增加/刪除是容易的，因為是在 user site 執行，∴服務的增刪不需要牽扯到 kernel 的變更，即使要也是小幅修改
         2. ∵kernel 很小，所以移到新的硬體平台之更動幅度不大
         3. 萬一其中某一個服務在執行中掛了，充其量相當於一個 user process 掛掉而已，對 kernel、user process 並無不良影響，∴更加安全可靠
         4. 同 3.
   (四)缺點：Performance overhead of user space to kernel space communication
   效能較差*(建立過多的 Message Passing 通道)*

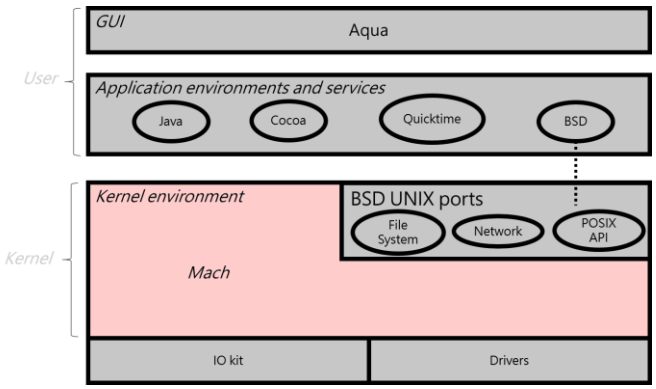
(註)microkernel 的相反詞為"Monolithic" kernel：所有的 Services 皆需 Run in kernel mode，優、缺與 microkernel 相反

比喻：鐵金庫 Iron Bank
《冰與火之歌》中的鐵銀行，規模小但精良：1.易擴充、2.易於移動、3.可靠且安全

5. Module：
   Many OS implement loadable kernel modules
   1. Use object-oriented approach.
   2. Each core component is separate
   3. Each talks to the others over know interface
   4. Each is loadable as needed within the kernel
      比 layered 還 flexible*(想 call 誰就 call 誰)*
      比 microkernel 還有效率*(沒有繁雜的 System Call)*
      比 monolithic 還有架構



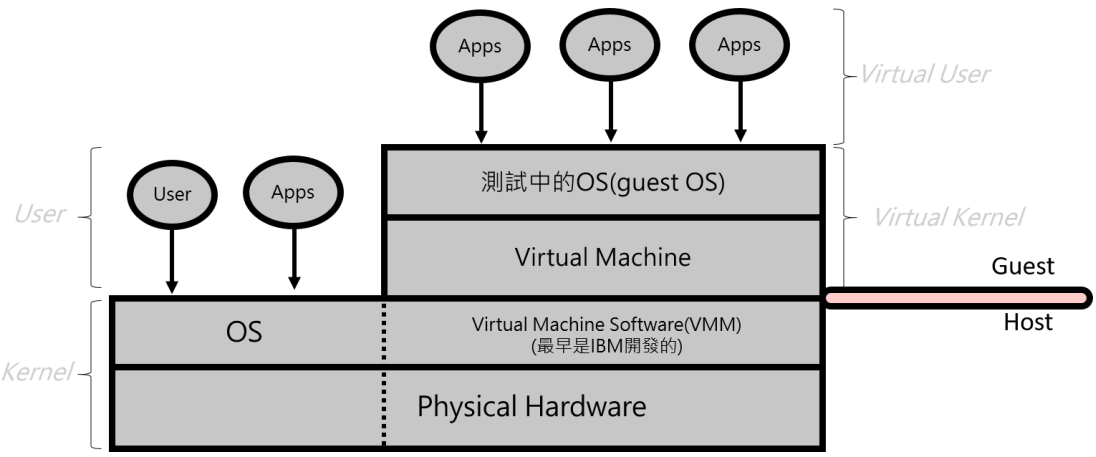
6. Hybrid(混合型)
   1. 現代的 OS 很難純粹歸到某一種
   2. 例：Linux and Solaris 是 monolithic 且也是 module for dynamic loading
   3. 例：Windows mostly monolithic，有時針對不同客戶之需求，加上 microkernel for subsystem
   4. 例：Apple Mac OS 也是混合型，kernel 包含了：Mach microkernel、部分的 BSD UNIX、I/O kit、dynamic loadable module(叫作 kernel extension)



Mac OS X Structure

Virtual Machine

(一)Def：利用 Software Simulation 技術模擬出一份與底層 HW 一模一樣的功能界面之抽象化機器(abstract machine)



[名詞]：

1. Host：underlying HW System、OS
2. VMM(Virtual machine manager)或 Hypervisor：creates and managing/runs virtual machine
3. Guest

[恐]其他之英文

1. Abstract HW of a single computer into several different extension environment
2. Similar to Layered approach, but layer creates VM

(二)優點：

1. 利於測試：作為測試開發中的 OS 一個良好的負載平台，好處有：
   甲、其他 user process 工作可持續運作，不需暫停
   乙、萬一測試中的 OS 不穩定、掛掉、失敗了，也不會影響 host HW、OS…等其他 User process 之工作，因為只是相當於一個 User process fails 而已，不會對 System 有重大危害
2. 經濟度：同一部 host HW 上可執行多個 VM，可節省成本
3. 安全性：如果 VM 被病毒入侵，不致擴散，因為 VMs 各自獨立
4. 擴展性：可以 Freeze、Suspend、Running VM 及 clone VM
5. 合併性(Consolidation)：在 cloud computing 的時候，我們會用有限的機器建立為數很多的 VM，我們可以依 VM 上的 application 之執行負擔輕重，調動 host machines 資源作對應的支援
6. 模板化：可依需求提供不同服務
7. 移植性：可輕鬆移植

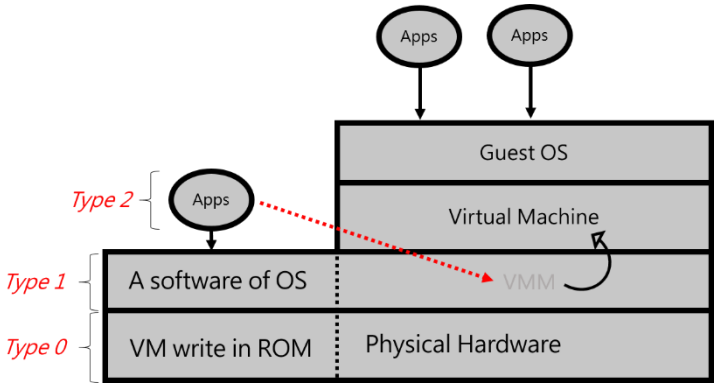
5、6、7 點皆與雲端運算有關

(三)缺點：

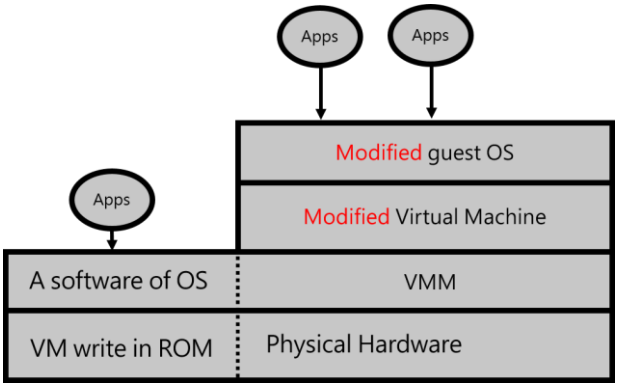
1. 不易製作
2. 效能較實體機來得差
3. 需要硬體支援

Virtual Machine Management 的 implementation

1. Type 0：HW-based solutions via firmware，把 VM 功能的軟體放到 ROM 內。
   例：IBM LPARS and Oracle LDOMs
2. Type 1：
   1. OS-like SW，就是一個 OS 的軟體。
      例：VMware ESX、Joyent SmartOS、Citrix XenServer
   2. general purpose OS that provide VMM functions (services)，多一個 kernel service 是 VMM。
      例：Microsoft Window Server with Hyper V、Red Hat Linux with KVM
   3. Type 2：App that run on standard OS but provides VMM features to guest OS。
      例：VMware Workstation、Parallel Desktop、Oracle Virtual Box



4. Para-virtualization：The guest OS need modify to work in cooperation with VMM to optimizing performance, presents the guest with similar, but not identical to host HW. guest must be modified to run on para-virtualization HW.

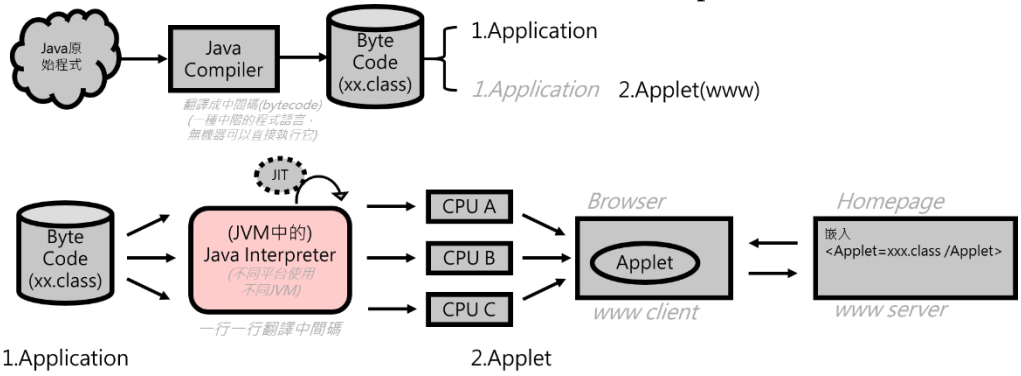


5. Programming-environment Virtualization：VMM do not virtualization real HW but instead create an optimized virtual system. ex：JAVA Virtual Machine(JVM)、Microsoft Network

JVM is a specification, not a implementation.

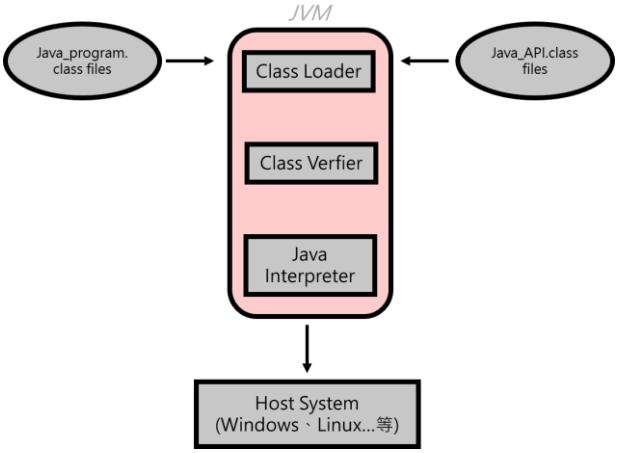
Java 是由 Sun Microsystems 於 1995 年發行，其特色為『獨立』、適用於多種不同平台，其流程如下：

將寫好的 Java 程式，透過 Java Compiler 編譯成 Java Bytecode*(是一種中階語言，無法直接在機器上執行)*，接著使用透過不同種 JVM*(需下載/使用對應平台之 JVM，ex：Linux 有 Linux 自己的 JVM、Windows 有 Windows 自己的 JVM…)*，將 Bytecode 一行行翻譯成該平台可用之機器語言，在不同平台上執行。但因為一行行翻譯速度較慢，故翻譯過程(可能)會使用 Just In Time Compiler(JIT)協助



JVM 基本上都會包含以下三種：

1. 類別載入器 class loader
2. 類別驗證器 class verifier
3. JAVA 直譯器 JAVA interpreter



6. Emulator(模擬器)： Allow application written for one HW on a very different type of CPU
7. Application containment：ex：Oracle Solaris Zones、BSD Jails、IBM AIX WPARs

補充：

Virtual machine << physical machine
Virtual Memory << physical memory
Virtual Disk >> physical Disk *(因為是由 RAM 來虛擬 Disk)*

Policy(政策、策略)與 Mechanism(機制)

(一)Policy： 1.定義 “What” to be provide
2.經常改變(參數化)

(二)Mechanism： 1.定義 “How” to do that
2.The underlying mechanism、甚少改變或不改變

(三)設計原則：“Policy 與 Mechanism 宜 separate，以增進 System flexibility.”

(四)例： 1. 運用 Timer 作為 CPU protection：Mechanism
Max Time quantum 大小制定：Policy
2. CPU 排班採用 Priority 排班：Mechanism
Priority 大小定義：Policy