一、 緒論

1. 何謂 OS
2. OS 的別名
3. OS 的組成元件
4. CPU 的工作型態
5. OS 的演進、類型
6. SYSGEN
7. booting、bootstrap
8. 開機程序

二、 I/O

1. I/O 的運作方式
   i. polling 又稱 PIO(Programmed I/O)
   ii. interrupt
   iii. DMA
2. Busy waiting
3. Daisy Chain
   多個 device 共用同一條 interrupt line 與 CPU 連線
4. Buffering vs Spooling(??到底有什麼不同??)
5. I/O Protection
6. Memory Protection
7. CPU Protection
8. process 與 OS 之間的參數傳遞
9. Layered approach(層級型方式)

三、 程序(Processes)

1. 定義
2. Process State Diagram(處理程序的狀態轉換圖)
3. PCB（Process Control Block）處理程序控制區段
4. Thread(LWP)

四、 CPU Scheduling

1. 程序排程佇列種類
2. scheduler 的種類
3. preemptive、non-preemptive 排程
4. context switching(內文交換)
5. 如何降低 context switching
6. Dispatcher(分配器)
7. CPU scheduling 的 criteria(衡量準則)
8. 各種 CPU scheduling 的探討(考試重點，各種排程的 waiting time 計算、比較)
   i. convoy effect

五、 DeadLock

1. DeadLock 的定義
2. DeadLock 的必要條件
3. Resource Allocation Graph
4. DeadLock Prevent(死結預防)
5. DeadLock Avoidance(死結避免)
6. DeadLock Detection(死結偵測)
7. DeadLock Recovery(死結復原)
8. UNIX 採用 Ostrich algorithm 來避免死結

六、 Memory

1. Address Binding(OS 課本 p6-3)
2. Dynamic Loading
3. Overlay
4. Dynamic Linking
   i. Stub
5. Swapping
6. 記憶體配置的策略
7. External Fragmentation
8. Paging(分頁記憶體管理)
   Reentrant、Pure code
9. Internal Fragmentation
10. Segmentation(分段記憶體管理)
11. Paged Segment(分頁式分段)

七、 Virtual Memory

1. 意義、優點
2. Demand Paging
   i. Page Fault
   ii. Pure Demand Page
   iii. Prepaging
3. Page Replacement
   i. Belady's anomaly
   ii. Global replacement
   iii. Local replacement
4. Thrashing
   i. Thrashing 的現象
   ii. Thrashing 如何處理
      1. working set
5. Page size
6. 程式結構

八、 Secondary storage

1. Disk Free Space Management
2. Allocation Method
3. Disk scheduling
4. Storage hierarchy

九、 Concurrent Processes

1. 生產者、消費者問題模式
2. Critical Section
3. Two Processes 的解決方案
4. N Processes 的解決方案
    i. Eisenberg and Mc Guire's algorithm
    ii. Bakery algorithm
    iii. Hardware Instruction Support Solutions
        1. test and set
        2. swap
    iv. Swap 指令
5. Semaphore
    i. Spinlock(自旋鎖) !!!???
    ii. Counting semaphore
    iii. Bounded Buffer Producer-Consumer Problem
    iv. Reader/Writer Problem
    v. 哲學家用餐問題
    vi. 使用 semaphore 的一些問題
6. Monitor
7. Message Passing
8. Rendezvous

十、 File、Directory

1. open file、close file
2. 檔案存取的方式
3. 目錄結構(OS 課本 p9-11)(汪八回 p145~149)
4. File Protection

十一、 Assembler、Linker、Loader、Macro Processor、Compiler

十二、 有的沒的名詞解釋

1. NFS：Network File System
2. Process Migration
3. Race Condition

一：何謂 OS：

4. user 與 H/W 的溝通介面
5. 提供 user 方便、有效的環境使用電腦系統
6. 資源分配
7. 監督程式的執行

note：Bare Machine 是指硬體，Extended Machine 是指系統軟體、應用軟體

一：OS 的別名：

Resource allocator	Control program	kernel

一：OS 的組成元件：

打	P	Process Management	程序的建立、刪除、暫停、恢復、同步(critical section)、通訊(message passing、shared memory)、死結處理
不	M	Memory Management	記憶體的監督、載入、分配、回收
死	S	Secondary Storage Management	可用空間管理、儲存體配置、磁碟排程
	I/O	I/O system Management	buffer、driver
讓他打	F	File Management	檔案、目錄的建立、刪除、管理 檔案的 mapping、backup
	P	Protection	CPU Protection Memory Protection I/O Protection File Protection Other resources Protection
穩	N	Network	
死	C	Command Interpreter System	

一：CPU 的工作型態：

I/O Bound Jobs	CPU 等 I/O
CPU Bound Jobs	I/O 等 CPU

一：OS 的演進、類型：

1940	無 OS，人工撰寫 Machine Language
1950	Batch Processing
1960	Multi-Programming：mem 中有多個程序，當正在執行的程序正在 I/O，CPU 才執行另一個程序
	Multi-Processing = Multi-Processor：多 CPU 又分為 symmetric(同型 CPU)、asymmetric(不同型 CPU)
	Time-sharing 由 Multi-Programming 衍生而來，條件上多加了每個程序只能佔用一定的執行時間
	Real-time：即時系統，若是要求很高的即時性系統，通常沒有 HD，以 RAM 做 HD，所以也沒有 Virtual Memory

Multi-Tasking	一個 user 可以同時處理多個工作 windows、Linux、Unix 都是
Multi-Thread Thread 又稱為 Light weight process	一個程序同時可以有多個 thread 在執行 windows、Linux、JVM 支援，傳統的 Unix 沒有，IBM 的 AIX，HP 的 HP-ux，Sun 的 Solaris 支援

Foreground(前景處理)	CPU 即時處理
Background(背景處理)	CPU 有空時才處理

一：SYSGEN：

在任何一類機器上都可以執行的作業系統，程式會測試硬體的組態，自動產生可以使用的 OS

一：booting、bootstrap：

booting：載入核心程式來使電腦開始運作

bootstrap：存在 rom 中的程式，可以載入 kernel 至記憶體中，並執行此 kernel

▲booting 是一種動作，bootstrap 是一個程式

一：開機程序：

	P(打)	Test processor
	B(不會)	Verify BIOS integrity
	I(唉)	Initialize chipset
	RI(你)	Test RAM Initialize video device
	P(打)	Initialize Plug&Play devices
	R(啊)	ROM scan
		Load from boot device
		Run bootstrap loader
		Find and load OS loader
		Run OS loader
		Load and run OS

▲電腦開機時，存於 ROM 內的啟動常式會發出命令，將作業系統的哪三部分從硬碟讀入主記憶體的系統記憶區(system memory)？

1. 監督程式(supervisor/monitor)：負責與使用者互傳訊息、執行輸出入作業、接受命令、解釋並執行命令以及控制電腦系統的全盤作業。
2. 常駐命令(resident command)：存於常駐命令(程式)區，供存放經常使用的公用程式，如 Copy。
3. 暫駐命令(transient command)：存於暫駐命令(程式)區，供存放不常用的程式，如 Format 等。需要此命令時，所需的指令會從磁碟讀入暫駐區並執行，若有其他命令要做時，對應的軟體會讀入該區，並蓋上原先該區的資料。所以暫駐區的資料可能經常變更。

二：I/O 的運作方式：

1. Polling：又稱 Programmed I/O(PIO)或狀態核對(Status Checking)，其利用程式主動控制，OS 定時主動向各週邊設備輪詢(polling)。週邊設備如有需求就把 CPU 使用權交給該週邊設備，若無需求，則跳到下個週邊設備輪詢。適合低速資料的 I/O。缺點：1.CPU 浪費很多時間在測試 I/O 設備的狀態。2.資料傳輸經由 CPU，而不是直接由主記憶體與 I/O 進行傳輸
2. Interrupt I/O

1	2	3	4	5
CPU 收到 Interrupt 訊號	中斷目前工作，進行 context switch	透過 Interrupt Vector 尋找對應的 ISR(Interrupt Service Routine)，並執行之	完成 ISR	回到之前執行程序的狀態

中斷的種類

External interrupt	Machine Check Interrupt
	I/O interrupt
	Device(H/W Error)Interrupt
Internal interrupt 又稱為 Trap	Overflow
	Divide by zero
Software interrupt	System call(OS p2-9，它也被視為 Trap 的一種)

所以 Trap 又可以稱之為「由軟體產生的中斷」

3.DMA 如下所述

二：DMA：

1. DMA 利用 DMA controller 代替 CPU 監督資料輸過程，此時 CPU 可執行其它工作。
2. cycle stealing：DMA controller 有資料傳送時，disable CPU 的 bus access，然後進行 memory access，資料傳送完後再 enable CPU 的 bus access。中斷的時間以一個 block 的傳輸為單位。
3. DMA 在 I/O 時，CPU 不需執行 ISR 動作。

汪八回 p11 的圖

二：Buffering vs Spooling：

目的：為了讓 CPU 的運作與 I/O 的運作能夠重疊

Buffering	將一項工作的 I/O 與本身的計算重疊
Spooling	將一項工作的 I/O 與另一工作的運算重疊

Spool：Simultaneous Peripheral Operation On-Line

二：I/O Protection：

目的：避免硬體被不當使用，例如兩個人同時列印

1. Dual Mode：在硬體中加入一個 Mode bit，0：monitor mode，1：user mode；使用者只能在 user mode 時使用硬體
2. 特權指令(Privileged Instruction)：

   口訣：(CS)W(HIT)

   僅可在 monitor mode 下執行的指令

   甲、 Change to monitor mode

   乙、 Set timer

   丙、 Write into monitor memory

   丁、 I/O instructions

   戊、 Turn on/off interrupts

   己、 Halt 指令
3. System call

   用途：執行程式與 OS 之間的溝通介面

   Q：既然 I/O instructions 是特權指令，那麼 user program 如何進行 I/O?

   Ans：呼叫 system call，再由 OS 執行 I/O

二：Memory Protection：

1. OS 跟 user program 的記憶體保護：防止 OS 的記憶體內容被不當修改

Monitor(OS)
Fence Register
User Program

2. Job 之間的記憶體保護：防止 Job 間的記憶體內容被不當修改

Job1	Limit register
	Base register
Job2	Limit register
	Base register
Job3	Limit register
	Base register

二：CPU Protection：

目的：避免 CPU 被無限期佔用或進入無窮迴圈

★Timer：根據特定的設定時間發出 timer interrupt，取得 CPU 控制權

二：process 與 OS 之間的參數傳遞：

1. 送入到暫存器
2. 把參數儲存在記憶體中的某個位址，再將該位址當作參數傳到暫存器中
3. 存放在 Stack 中

二：Layered approach：

▲將 OS 分成數個層級，每一個層級都建構在較低層級之上。層級的優點是模組化，每一層級只能准許利用低層次中的功能與服務，因此系統的除錯與驗證特別容易。

▲

Layer 5	使用者程式
Layer 4	輸入與輸出之緩衝
Layer 3	作業員-控制台裝置驅動程式
Layer 2	記憶體管理
Layer 1	CPU 排程
Layer 0	硬體

三：定義：

1. 程序是指執行中的程式
2. 程序包含：stack、memory、program counter、register
3. 程式是被動的實體，程序是主動的實體

三：Process State Diagram：

汪八回 p22

new	→ Admitted	ready	← interrupt → Scheduler dispatch	running	→ Exit	terminate
		↑ ← event or completion	Waiting	← ↓ I/O or event wait		

三：PCB：process control block

作業系統利用 PCB 來記錄每個處理程序的所有資訊

Process id	State 上表的 ready、running、wait 或 suspend(暫停)	PC(program counter)	CPU schedule
CPU registers	Memory information(ex:base、limit register、page)	I/O state	Account 資訊,使用 CPU 時間、時間限制

三：Thread(LWP)：

汪八回 p28

thread 又稱為 light weight process 是 CPU 執行時的基本單元。它包含

Thread id	Program counter	Register set	Stack space

Single thread	Multi-thread		
Code、Data、Files	Code、Data、Files		
Registers、Stack	registers	registers	Registers
thread	Stack	Stack	Stack
	thread	thread	Thread

四：程序排程佇列種類：

汪八回 p24

Job queue	系統中的所有程序
Ready queue	Main memory 中的 ready 程序
Device queue	等待某一特殊 I/O 裝置的程序集合

四：scheduler 的種類：

Long Term 又稱為 job scheduler	它決定那一個程序應該被選入至待命佇列中。此 scheduler 決定 multiprogramming 的 degree
Short Term 又稱為 CPU scheduler	它決定哪一個程序將在下次擁有 CPU

四：preemptive、non-preemptive 排程：

new	→ Admitted	ready	← interrupt → Scheduler dispatch	running	→ Exit	terminate
		↑ ← event or completion	Waiting	← ↓ I/O or event wait		

當 CPU 只在下列情況進行排程時，稱之為 non-preemptive 排程，否則稱之為 preemptive 排程

running→waiting
running→ready
waiting→ready
running→terminates

四：context switching：

當 CPU 從一個 process 切換到另一個 process 時，OS 要保存原有 process 的執行狀態，然後載入新的 process 之前保存的執行狀態的過程稱為 context switching。

四：如何降低 context switching：

(OS 課本 p4-15)

1. 提供專用指令(加速 context switching 的處理)
2. 運用 register set 的觀念，context switching 時，只轉換指向的 register set，而不用做大量的資料異動
3. process 專屬的 register set
4. 運用 thread 來處理，因為 thread 的 context switch 較少

四：Dispatcher(分配器)：

將 CPU 的控制權交給由短期排程所選定的程序，要進行的工作有：

1. 執行本文交換

2. 將執行模式變成使用者模式
3. 跳至適當的程式，開始執行

note：dispatch latency：分配程式終止一個程序並開始執行另一個程序所花費的時間

四：CPU scheduling 的 criteria：

T	Turnaround time	1. 從程序提交到程序完成過程所花的時間 2. 程序等待進入記憶體，等待進入預備佇列、CPU 執行、以及 I/O 等運作過程的時間
W	Waiting time	在預備佇列中等待的時間
C	CPU utilization	CPU 的使用率
R	Response time	從程序提交到得到第一個回應的時間間隔
T	Throughput(生產量)	每單位時間完成的 process 量

四：各種 CPU scheduling 的探討：

FCFS 或 FIFO	▲先來先做，效率最差，平均 turnaround time、waiting time 都比較長，會造成 convoy effect(護航效應) △convoy effect：其它 process 都在等一個要執行很久的 process
SJF (shortest job first)	▲可插隊排程，不中斷目前正在執行的程序 △CPU 每次執行都挑選工作時間最短的時間一樣長的，就用 FCFS 決定 △困難是不容易預知 process 的執行時間
SRT、SRJF (shortest remaining time first)	▲可插隊排程，會中斷目前正在執行的程序 △一有新的程序到達，就檢查新程序是否剩餘時間最短，是就先執行
Priority	▲根據 process 的優先權值決定執行順序，因此 SJF 可視為 Priority 排班法的特例，只是 SJF 是以執行時間或剩餘時間做為 priority △優先權一樣高的，就用 FCFS 決定 ▲priority 排程會有 starvation 的問題，可用 aging technique(老化技術)逐步提升久等的 process 的 priority
RR (Round Robin)	運用 time quantum(時間配額)或 time slice(時間分割)的方式，限定每個 process 的執行時間。 turnaround time 比 SRTF 長，但 response time 比 SRTF 好
MLQ (Multi Level Queue)	將 queue 分為多個等級，每個 queue 各自有各自的排班演算法，process 不能在 queue 間移動，因此會有 starvation 的問題
MLFQ (Multi Level Feedback Queue)	同上，只是應用 aging technology，把用太久的 process 降低 priority，等太久的提高 priority

五：DeadLock 的定義：

一組被停滯的程序中，因互相等待而形成的現象

五：DeadLock 的必要條件：

Mutual exclusion	同一時間一個資源只能被一個 process 佔用
Hold and wait	Process 已持有資源，並在等待被其它 process 佔用的資源
No preemption	資源要由 process 自願釋放，不能被其它 process 強取
Circular wait	持有與等待的 processes 形成一個循環

五：DeadLock Prevent：

只要使死結的四個必要條件不完全成立，即可預防死結

破 Mutual exclusion	非共享資源的互斥條件無法打破，如印表機，天生就不可共用
破 Hold and wait	1. 一開始就全配：Process 要取得所需的全部資源才能開始執行，否則要空手等待 (此方法是一種預先資源分配的方式) 2. 吃完了再夾菜：process 要在沒有佔用資源的情形下才能請求資源 缺點： 1. 會發生 starvation，因為某些資源總是被別人取得 2. 資源利用率低，因為 process 會用到該資源就要先分配，但 process 可能要過一段時間才會用到它
破 No preemption	也就是允許插隊的意思，做法如下： 請求資源時： 1. 沒人在用→給 2. 有人佔用，但目前是等待狀態→搶 3. 不能給→我等，而別人此時可以來搶我 適用時機： 狀態容易保存、被搶奪後容易恢復的資源，如 register、memory，但不適用於 printer、磁帶機
破 Circular wait	1. 將資源遞增編號 2. 規定：process Pi，resource Ri ▲如果 Pi 目前未佔用資源→可以請求任何編號的資源 ▲如果 Pi 目前佔用資源 Ri→則它只能申請比 Ri 編號大的資源 ▲如果 Pi 目前佔用資源 Ri→而申請比 Ri 編號小的 Rj 資源，則要先釋放目前佔用的資源編號大於 Rj 的所有資源 3. 證明： 反證法：假設使用上述原則，還是會發生 circular waiting，那麼必存在著 $R_i \rightarrow P_i \rightarrow R_{i+k}$ 其中 i,k>0 $R_{i+k} \rightarrow P_j \rightarrow R_i$ 其中 j>i (note：上述的表示法是 Pi 佔用 Ri 資源並且等待 R_{i+k} 資源) 但前面已規定 i 必需小於 i+k 狀況一才能存在，而 i+k 必需小於 i 狀況二才能存在，因為不存在這樣的 i,k 值，所以此法能避免 circular wait

五：DeadLock Avoidance：

1. 何謂安全狀態

Safe state 是指可以找到一組執行順序完成執行所有的程序；而 unsafe state 則是找不到這樣的執行順序，但不一定會發生死結，所以死結只是 unsafe state 的一部分

為什麼 unsafe 不一定 deadlock 呢?

2. ★Banker's Algorithm★

資料結構：5 個 process，3 種 resource

	Allocation 已分配資源			Max 最大需求資源			Need=Max-Allocation 尚需多少資源			Available 可用資源		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0												
P_1												
P_2												
P_3												
P_4												

時間複雜度：$m*(n(n+1)/2)=O(mn^2)$，其中 n 表 process，m 表 resource

證明：

步驟 1：Need=Max-Allocation，初始化一個 m*n 陣列→O(mn)

步驟 2：

檢查是否 need<=available→最多檢查 n 個

檢查是否 need<=available→最多檢查 n-1 個

…

檢查是否 need<=available→最多檢查 2 個

檢查是否 need<=available→檢查最後 1 個
1+2+3+……+(n-1)+n

3. OS 課本 p5-15
   如果每個資源的 available 都是 1 時，可用圖形的方式(claim edge)，檢查圖形有沒有發生循環，有的話表示會發生循環

五：DeadLock Detection：

1. 偵測方式：Banker's algorithm
2. resource allocation graph 轉成 wait for graph(OS 課本 p5-19)
3. 執行偵測的時間點
   甲、 當有程序請求資源卻沒有立即得到允許時
   乙、 固定時間間隔執行
   丙、 CPU 的使用率低於 40%時

五：DeadLock Recovery：

1. 終止所有程序
2. 逐一終止程序直到狀況解除

六：Address Binding：

Compiling time	如果此時知道程式要擺在那裡，就產生 absolute code，否則就產生 relocatable code 例如.com 檔是 absolute 的，而.exe 可以是 relocatable
Loading time	
Linking time	
Execution time	

六：Dynamic Loading：

當程式被叫用到時，程式才載入到記憶體中；運作方式：一開始只有主程式被載入，當副程式被呼叫時，系統檢查程式是否已在記憶體中，若不在，則利用可重新定位 linker 來將此副程式載入，然後將控制權交給這個副程式。

六：Overlay：

Dynamic loading 的一種技巧，程式執行時只保留當時所需的指令及資料，當需要用到其它指令時，就把這些指令載入到已經不再使用的指令所佔用的空間中。

六：Dynamic Linking：

Dynamic loading 是執行時才「載入」，Dynamic Linking 是執行時才「連結」。

載入跟連結的差異是，載入是指該段程式還不在記憶體中，而連結則是已在記憶體中

1. 執行程式中會有一段程式碼(stub)，它用來表示如何找到記憶體中的函式庫(library routine)
2. 當執行到這段 stub 時，會把自己代換成那個函式庫所在的起始位置，並跳到那個位址開始執行
3. 優點是數個程式可以共用同一個函式庫

ex：MS 的.dll 檔，Linux、Unix 的.so 檔

六：Swapping：

swap in：將程式由記憶體移至備用儲存體上

swap out：將程式由備用儲存體移至記憶體中

▲有一種以優先權為基礎的 swapping，若有一個優先權較高的程序要執行，而記憶體空間不足，此時為了載入此優先權較高的程序，可以將優先權較低的程序先 swap out，當優先權高的程序執行完後，再將優先權低的程序 swap in 繼續執行。這種就是 swapping，又稱之為 roll in/roll out。

六：記憶體配置的策略：

假設有一 process 執行所需的 memory size 是 n

1. First-Fit
   從可以使用的位置開頭開始找，找到 size>=n 的區塊分配給 process 為止
2. Next-Fit
   從上一次配置後的位置開始找起，找到 size>=n 的區塊分配給 process 為止
3. Best-Fit
   搜尋整個可用的空間，取 size>=n 且 size-n 最小的區塊
4. Worst-Fit
   搜尋整個可用的空間，取 size>=n 且 size-n 最大的區塊

比較：

	搜尋時間	空間利用
First-Fit	快	佳
Best-Fit	慢	優
Worst-Fit	慢	差

缺點：

1. External Fragmentation
2. Internal Fragmentation

六：External Fragmentation：

▲定義：隨著程序在記憶體移進、移出，記憶體會被分割成許多碎片。雖然這些零碎的空間加起來可以滿足一個程序的需要，但由於空間不連續，所以仍無法配置。

▲三種解決方式

1. 記憶體壓縮(memory compaction)
   移動執行中的 processes，讓可用區塊能夠連續。
   限制：必需是 dynamic binding 的 process 才能用此方法，因為壓縮會重新定位 process 的記憶體位置
2. 多套基址暫存器

運用多套 Base-Limit registers，將 process 所需的記憶體分成幾個小部分，這樣在記憶體裡比較容易裝填。

3. paging

六：Paging：

▲相關名詞：

Frames	實際記憶體被分成許多固定大小的區塊
Pages	邏輯記憶也被分成許多「同樣大小」的區塊
Shared Pages (共用頁面)	Reentrant：執行中的程式碼可跟別人共用的稱為可重入式程式碼，也叫做純碼(Pure code)

Logical address to Physical address：OS 課本 p6-16

▲Paging 的優點：

1.解決 External fragmentation
2.頁面可以共用
3.可設定頁面為 Read only，頁面保護較容易

▲Paging 的缺點：

會有內部碎裂	為了減少內部碎裂，可將 page size 縮小，但會造成 page table 變大，反而佔據空間
Logical address to Physical address 的轉換時間	
需要 HW 的支援，右列是三種實做法式，以第三種為最佳	1.用一組暫存器來儲存，可以快速完成 Logical address to Physical address 的轉換時間，但限制是分頁表(亦即分頁數量)要很小
	2.把分頁表放在主記憶體中，使用 PTBR：Page Table Base Register 分頁表基址暫存器，來指向分頁表在記憶體中的位址。這樣內文轉換時只需改變這個暫存器，可縮短處理轉換的時間 做法：CPU 去 PTBR 中讀取 page 所在的記憶體位置，讀出 frame number 後，再+page offset→所要的位置 因此一共要讀取主記憶體 2 次
	3.運用 TLBS(Transaction lookaside buffers 或稱 associative register)，根據電腦的 locality 特性，TLBS 做為 Page Table 的 cache，可大幅提升記憶體存取的效率

汪八回 p92

六：Internal Fragmentation：

1. 定義：分配給程序的記憶體空間比程序所需還大，多出來的部分就稱之為內部碎裂。

六：Segmentation：

Logical address to Physical address：OS 課本 p6-22

▲Segmentation 的優點：

1.沒有 internal fragmentation
2.區段可以共用
3.可設定區段為 Read only，區段保護較容易

▲Segmentation 的缺點：

會有外部碎裂	
Logical address to Physical address 的轉換時間	
需要 HW 的支援，右列是三種實做法式，以第三種為最佳	1.用一組暫存器來儲存，可以快速完成 Logical address to Physical address 的轉換時間，限制是分段不能太多
	2.類似 paging 的方法 2，要用到 STBR(Segment Table Base Register)、STLR(Segment Table Length Register) 因此一共要讀取主記憶體 2 次
	3.跟 paging 的做法類似，運用 TLBS

六：Paged Segment：

Logical address to Physical address：OS 課本 p6-24

七：Virtual Memory 的意義、優點：

▲作法：CPU 送出虛擬位址，透過 memory map 對應到實體記憶體的位址。

▲意義：允許執行的 process 不必全部 load 到記憶體中，亦即實際執行中的 process 大小可以大於實際記憶體的大小。

▲優點：

1.程式可以不受實體記憶的可用空間限制
2.每個 process 只佔有少許的記憶體空間，提升 CPU 的使用率以及輸出量
3.process 載入記憶體的 I/O 時間縮短(一開始不必全部載入)
4.程式撰寫更容易，可以不用 overlay 的技術

七：Demand Paging：

實作 Virtual Memory 的技術，透過 Lazy swapper 或 pager(分頁器)的方式，只調入執行時所須的頁面，其餘的一概不予載入。

★Page Fault

1. 當執行中的 process 要存取某個沒有載入到記憶體中的頁面時，就會發生 page fault 中斷。
2. Page Fault 的處理：

OS 課本 p7-6

1. 檢查 process 的內部表(通常在 PCB 中)，以確定記憶

體存取是否合法

2. 若 interrupt 是因為不合法的存取造成，則終止 process，否則表示是由 page fault 造成
3. 從主記憶體中找尋一個 free frame(若沒空間則執行 page replacement)
4. 將所需的頁面讀入新分配的 frame 裡
5. 載入頁面後，修改 process 的內部表，以及修改分頁表，將 invalid bit 改為 valid bit

★Pure Demand Page

process 執行之初，並不事先載入任何頁面，當該處理程序在執行第一個指令時即會發生 Page Fault，此時才載入需求之頁面的做法。

★Prepaging

process 執行之初會用到的 page 都先行載入

七：Page Replacement：

▲當記憶體內沒有可用的頁框時，尋找一個犧牲頁面，將頁框裡的內容寫到磁碟上空出這個頁框，然後更正分頁表

Page Replacement Algorithms：(常考計算題)

1.FIFO	
2.Optimal	當需要從記憶體中找出一個 frame 來替換時，選擇未來最久才會被參考存取的那個 frame 此法保證不會有 Belady's anomaly，page fault 次數最少，但有實作上的困難，因為無法預知未來(跟 SJF 遇到的困難類似)
LRU(Least Recently Used Algorithm)	置換最長時間沒被用到的 frame 實作方式： 1. 計數器 2. 堆疊
LRU 近似演算法	1. Additional Reference Bits Algorithm(附加參考位元演算法) 2. Second Chance Algorithm(二次機會演算法) 以 FIFO 為基礎，選擇一個 FIFO 頁面後，檢查參考位元，如果為 0，就替換這個頁面，若為 1，則修改為 0，但這次不做替換。如果某個頁面經常被使用到，參考位元就會一直保持在 1。
Enhanced Second Chance	(0,0)沒用，沒改 (0,1)沒用，有改 (1,0)有用，沒改 (1,1)有用，有改 替換的優先順序為上列四項由上而下
LFU(Least Frequently Used)	每個 frame 一個 counter，剛載入時設為 0，參考到一次就加 1，替換時選擇 counter 最小的頁面替換
MFU(Most Frequently Used)	每個 frame 一個 counter，剛載入時設為 0，參考到一次就加 1，替換時選擇 counter 最大的頁面替換

★Belady's anomaly：一般而言，page frames 越大，page fault 的機率就會降低，但對於某些頁面替換演算法，page fault 的發生次數會隨著 page frames 變大而增加，此現象稱之為 Belady's anomaly。

例：1,2,3,4,1,2,5,1,2,3,4,5，frames 從 3 個→4 個

▲Global replacement：在執行 page replacement 時，所有記憶體中的 frame 都可做置換

▲Local replacement：在執行 page replacement 時，只能從自己所擁有的 frame 中做置換

七：Thrashing：

▲一個程序如果花在 page replacement 的時間比執行時間還多，就表示它正處於猛移現象之中

▲形成的原因：

key word：page fault、page replacement、multiprogramming 的 degree

1.在 multiprogramming 的環境下，採用 global page replacement algorithm(即不管替換的頁面屬於那個 process，都一律可以替換)
2.執行中的 process 需要載入未載入的頁面，因此發生 page fault，但頁框都滿了，所以要執行 page replacement
3.輪到下一個 process 執行，恰好它要用的頁面被換掉了，於是發生 page fault，然後進行 page replacement
4.由於 process 都花時間在 page fault、page replacement 上，所以 CPU 的使用率會降低，於是 CPU 提高 multiprogramming 的 degree，執行更多 process
5.page fault、page replacement 更嚴重→Thrashing 現象

▲如何處理：

1.局部替換演算法、優先權演算法	如果 process 正發生 thrashing，則此 process 不能再從別的 process 取得 frame，在做 page replacement 時，只能對自己使用到的 page 做置換。這樣可以把猛移現象限制在局部的範圍內。
2.Working Set Model	Working set window 代表 process 最近△次存取的頁面 頁框總需要量 $D=\sum WSS_i$ WSSi：每個 process 的 working set 的 frame 數 若 D>M(實際可用頁框量)，就會出現 thrashing 現象，因此 OS 可以先行防範
3.Page Fault Frequency	OS 設定 page fault ratio 的上、下限，若 process 的 page fault ratio 過高，OS 就多分一點頁框給 process，若太低，則搶走一些

★working set

working set window 用來記錄程序最近△時間內的存取頁面

1	2	3	4	5	6	7	2	3	1	2	1	2	1
						t1 ↑							t2 ↑

以上表為例，時間點 t1 的 working set window 記錄 {1,2,3,4,5,6,7}window size=7，而時間點 t2 則記錄 {1,2,3}window size=3；由此可以推估在某一時間店的 frame 總需求量是多少，若系統的可供給量小於總需求量，表示可能發生 thrashing，此時系統可先暫停某些程序，將其佔用的 frame 釋放出來給其它程序使用。

七：Page size：

Page size 大	優點	1. Page fault ratio 減少
	缺點	1. I/O transfer 時間長 2. internal fragmentation 較嚴重
Page size 小	優點	1. 記憶體使用率高 2. I/O transfer 時間短 3. internal fragmentation 較輕微
	缺點	1. Page Table 大，佔空間 2. Page fault ratio 增加

由於 CPU 越來越快，Memory 容量越來越大，所以 page size 的設計也朝大 size 發展。

七：程式結構：

row-major 的記憶體配置、column-major 的記憶體配置跟程式的寫法會影響到 page fault。

八：Disk Free Space Management：

1. Bit Vector：將 Disk 的可用空間串列以 Bit Map 或 Bit Vector 來表示。每個 bit 代表一個區段，如果該區段 free，則對應的位元為 0，如果該區段被佔用，則對應的位元為 1。

優點	缺點
1. 簡單 2. 易於找到連續的空間	必需把全部的位元向量存放於主記憶中，所以只適用小型磁碟。

2. Linked List：用 linked list 的方式，將可用的區段用指標連結起來，每個可用區段皆用指標指向下一個可用區段。

優點	缺點
加入、刪除方便	只能線性搜尋，效率不佳

3. Combination

   將可用區段的位址都集中在同一個區段中，若區段大小為 n，則此區段可以放 n-1 個可用區段的記錄，第 n 筆記錄則指向下一個集中存放可用區段位址的區段。

4. Counting

可用區段位址	counting	意義
可用區段 001	2	意思是可用區段 001 及其後連續兩個區段都是可用的
可用區段 012	0	意思是沒有連在可用區段 012 之後的可用區段

八：Allocation Method：

1. Contiguous Allocation

   Directory

File	Start	Length
xx1	19	6
xx2	28	4

xx1 檔案從區塊 19 分佈到區塊 24

xx2 檔案從區塊 28 分佈到區塊 31

優點	缺點
1. 支援循序、直接存取 2. seek time 最短	1. 會造成 External Fragmentation 2. 不容易事先預知檔案的大小

2. Linked Allocation

   Directory

File	Start	End
xx1	9	25

Directory 記錄檔案的起始區塊，結束區塊，而區塊之間則是以指標相連，並非連續分佈。

優點	缺點
1. 檔案建立容易，不需事先知道檔案的大小 2. 不會有 External fragmentation 的現象	1. 支援循序存取，不支援直接存取 2. 指標佔用空間 3. 可靠度不佳，萬一指標被破壞，則 block lost

▲FAT(File Allocation Table)

在磁碟上設置一個磁區段，用來存放該表，表中的每一項對應磁碟上的一個區段。

3. Index Allocation

   建立索引區(Index Block)記錄檔案的每個區段所在位置。

優點	缺點
1. 檔案建立容易，不需事先知道檔案的大小 2. 不會有 External fragmentation 的現象 3. 支援直接存取 4. 檔案擴充容易	1. Index 較 Linked 方式多佔用空間，對於小檔案的儲存不划算

對於大檔案的處理，可運用「多層索引」，當一個索引區滿了，再指向下一個索引區塊。

八：Disk scheduling：

1. 磁碟運作

Seek Time	磁頭移動到要讀寫的磁軌上的時間
Latency Time(rotation time)	磁頭已位於正確磁軌，等待要讀取的區段轉它的下方
Transfer Time	Disk 跟 memory 間傳送資料的時間

上述時間中以 seek time 最耗時

2. 磁碟排程種類

FCFS		優點：1.公平；2.簡單 缺點：不能提供最佳服務
SSTF(Shortest Seek-Time First Service)		選擇 seek time 最短的優先 缺點：1.會發生 starvation 2.不公平，也不是 optimal
SCAN	SCAN Scheduling	讀寫頭從磁碟的一端開始，向另一端移動，直到達磁碟的另一端再回頭反向服務
	C-SCAN(Circular SCAN)	與上述方法類似，只是讀寫頭只有從 A→B 的過程中會讀寫，到另一端回來的過程中不進行任何讀寫
Look	Look Scheduling	類似 SCAN 的做法，只是不用移動到底，只要移動到該方向上的最後一個工作做完，就可以立刻回頭
	C-Look	類似 Look 的做法，只是回頭的路上不工作

八：Storage hierarchy：

類型	容量	速度	價格
Register	小	快	便宜 貴
Cache			
Memory			
Backing store	大	慢	

Disk			
Magnetic			

九：生產者、消費者問題模式：

▲生產者程序產生訊資給消費者程序
▲欲使生產者與消費者兩個 process 並行運作，系統必須建立一個 Buffers，由生產者處理程序來輸入資料，並由消費者 Process 使用
▲Unbounded Buffer：producer 可以無限制產生資料，但 consumer 可能因為 buffer 空了而需等待。
▲Bounded Buffer：producer 可能 buffer 滿了而需等待，而 consumer 可能因為 buffer 空了而需等待。
▲此問題模式是在講，為了達成上述的控制，producer 與 consumer 間必需有一些共享的變數：
counter、in、out，其中 counter 變數是共用的

```
生產者：
repeat
…
produce an item in nextp
…
while counter=n do no-op;
buffer[in]:=nextp;
in:=in+1 mod n;
counter:=counter + 1;
until false;
```

```
消費者：
repeat
while counter=0 do no-op;
nextc:=buffer[out];
out:=out+1 mod n;
counter:=counter-1;
…
consume the item in nextc;
…
until false;
```

生產者、消費者問題即是指 counter 共用變數的保護，避免生產者、消費者同時修改 counter 變數而發生錯誤。

詳情請參閱汪八回 p43

bounded buffer 版本一是不會有這個問題的，但 buffer 的空間利用只能用到 n-1，上述的 counter 程式碼的寫法則全部可以利用，但會有共用變數同時修改的問題，由這裡牽扯到程式同步的討論。

九：Critical Section：

Critical section	當一個 process 在此區間運作時，其它 process 就不能進入此區間
Entry section	用來提出要求要進入 critical section 的程式段
Exit section	緊接在 critical section 之後的程式區
Remainder section	其餘的程式區段

★Critical section 需有下列三點特性：

Mutual exclusion	如果 process Pi 正在 critical section 中執行時，其它 process 均不允許在它們的 critical section 中執行
Progress	如果沒有任何程序在臨界區間，且有某個程序想進入自己的臨界區間，它必須可以進去，且不能被無限期的延遲
Bounded waiting	一個程序提出進入臨界區的請求後，等待允許的時間必需是有限的。Ex:若有 n 個 process 要進入臨界區，則最多等到(n-1)次之後必可進入臨界區

九：Two Processes 的解決方案：

1. 演算法 1

```
repeat
while turn <> i do no-op
critical section
turn=j
remainder section
until false;
請指出這段程式有什麼問題
```

2. 演算法 2

```
repeat
flag[i]=true;
while flag[j] do no-op;
critical section
flag[i]=false;
remainder section
until false
請指出這段程式有什麼問題
```

3. 演算法 3

```
repeat
flag[i]=true;
turn=j;
while (flag[j] and turn=j) do no-op;
critical section
flag[i]=false;
remainder section
until false
請指出這段程式有什麼問題
```

九：N Processes 的解決方案：

Eisenberg and Mc Guire's algorithm:

```
   var turn : 0..n-1;
   flag : array [0...n-1] of (idle, want-in, in-cs);
   initialize all flag[i] as idle,
   initial value of turn is immaterial.
   { for pi of n processes }
1  repeat
2    repeat
3      flag[i] := want-in;
4      j := turn;
5      while j <> i do
6        if flag[j] <> idle then j := turn
7        else j := j + 1 mod n;
8      flag[i] := in-CS;
9      j:=0;
10     while ( (j < n) and ((j=i) or (flag[j] <> in-CS)) ) do
11       j := j + 1;
12   until ( (j >= n) and ((turn = i) or (flag[turn] = idle)) );
13   turn := i;
14   CS
15   j := turn + 1 mod n;
16   while ( (j <> turn) and (flag[j] = idle) ) do
17   j := j + 1 mod n;
18   turn := j;
19   flag[i] := idle;
20   non-CS;
21 until false;
```

證明

滿足 mutual exclusion	情況一：某程序已在 CS 中，此時新進的程序會在 5~7 列轉圈圈 情況二：假設程序 1、5 同時執行，此時 turn=0，若兩個程序同時通過了 5~7 列的檢查，則在 10~11 列一定會有一個被攔下來
滿足 progress	沒有程序在 CS 裡時，任意程序都能進得了 CS
滿足 bounded waiting	假設 n=5，而提出申請 CS 的順序是 0→4→1→2→3，則觀察一下程序的特性，雖然程序 4 是第 2 個提出的，但當 0 在 CS 中，而 1,2,3 也陸續提出申請 CS 時，0 一離開 CS，1,2,3 因為程序編號比 4 小，所以很可能(但不一定)先獲得執行權，但由於列 17 的作法，因此程序 4 一定會在 n-1 次的等待時間內獲得執行權

Bakery algorithm: (Lamport,1974)

	var choosing: array [0..n-1] of boolean;
	number : array [0..n-1] of integer;
	initially: all entries of choosing are false
	number = 0
	notation :
	(i) (a,b) < (c,d) if a < c,or if a = c and b < d
	(ii) max (a0, a1, a2, ... , an-1) is k such that k >= ai ; for i=0,1,...,n-1
	{ for pi of n processes }
1	repeat
2	choosing[i]:=true;
3	number[i]:=max (number[0],number [1],...number[n-1])+1;
4	choosing[i]:=false;
5	for j=0 to n-1 do
6	begin
7	while choosing[j] do no-op;
8	while number[j] <> 0 and (number[j],j) < (number[i],i) do no-op;
9	end;
10	CS
11	number[i] := 0 ;
12	non-CS
13	until false;

證明

滿足 mutual exclusion	由列 7、8 負責；其中列 7 到底有什麼用途呢?詳述如下： 假設程序 3 正在執行 max(……)+1 的運算，且因故做得特別久，而此時程序 4 已完成選號進到列 6~9。假設程序 4 拿到了 10 號，且因為沒有列 7 的檢查，程序 4 通過了列 6~9(因為此時 number[3]還是 0)；程序 4 進到了 CS 此時程序 3 獲得一連串的執行權，且當時因為跟程序 4 同時在選號，意即程序 3 在執行 max(……)+1 的動作時，已讀到了 number[4]，而當時 number[4]還是 0 因為程序 4 也正在取號，所以程序 3 獲得一連串的執行權時，也取到了 10 號。於是程序 3 也會順利進到 CS，所以如果少了列 7 會違反 mutual exclusion
滿足 progress	Ok
滿足 bounded waiting	Ok

Bakery 演算法並不保證兩個 process 不會收到同樣的號碼，如果 Pi、Pj 取到同號，則若 i<j，Pi 會先取得服務。

Hardware instruction Support Solutions

```
function Test-and-Set(var target:Boolean):Boolean;
begin
        Test-and-Set:=target;
        Target:=true;
end;

//注意。var target 是傳址呼叫
```

```
Pi 之程式：Lock 初值為 false
repeat
while Test-and-Set(lock) do no-op;
        CS
Lock:=false;
        Remainder section
Until false;
```

此程式滿足 Mutual Exclusion，但並不滿足 Bounded Waiting

	Var waiting:array[0……n-1] of boolean;
	lock:Boolean;
	var j:0….n-1;
	key:Boolean;
	repeat
1	waiting[i]:=true;
2	key:=true;
3	while waiting[i] and key
4	do key:=Test-and-set(Lock);
5	waiting[i]:=false;
6	CS
7	j:=i+1 mod n;
8	While (j<>i) and (not waiting[j]) do j:=j+1 mod n;
9	If j=i then lock=false
10	Else waiting[j]:=false;
11	Remainder section
12	Until false;

列 3~4	若 Lock=false，表示沒有程序在用 則一執行列 4 之後，就會順利進入 CS，而此時 Lock 變成 true 而由於 Test-and-set 函數只會 set Lock=true，所以其它程序會在 Lock 等於 true 時，一直在列 4 繞圈圈
列 9	如果都沒有別的程序，而且下一個程序是自己
列 10	由於其它所有在等待的程序都在列 3、4 繞圈圈，所以不能去動 Lock 變數，因為 Lock 變數一動，可能會有一狗票程序衝進來，因此從 waiting[j]變數下手，因為每個程序各自擁有一個，只要讓 waiting[j]=false，它就會結束列 3 進到 CS；也因此目前在 CS 中的程序可以指定下一個要執行的程序，如此可以達成 Bounded waiting

Swap 指令

```
Procedure Swap(var a,b:Boolean)
Var temp:Boolean;
begin
    temp:=a;
    a:=b;
    b:=temp;
end;
```

```
Pi 之程式：lock:公用變數，key:局部變數，初值皆為 false
Repeat
    key:=true;
    repeat
        swap(lock,key);
    until key:=false;
        CS
    Lock:=false;
        Remainder section
until false;
```

此程式滿足 Mutual Exclusion，但並不滿足 Bounded Waiting，只要沒有做 j=i+1，然後由 j 往後檢查的都不會符合 bounded waiting

	第一個執行的程序，lock=false，key=true，所以會通過 swap 進到 CS，而當第一個程序一進入 swap，公用變數 lock 就變成 true，所以其它程序都會在 swap 那列繞圈圈，一直到 CS 完成把 lock 改成 false，才會讓下一個 process 進入 CS，但這個做法並不保證 Bounded Waiting

九：Semaphore：

▲semaphore 是一種不需使用 busy waiting 方式的同步方法

▲semaphore 是一個整數變數，提供兩種 atomic 運作：wait 與 signal，或稱 P(wait)與 V(signal)。wait、signal 的定義如下：

標準 semaphore
wait(S)：while s<=0 do no-op; S:=S-1; Signal(S)：S:=S+1

```
Pi 程式：
Semaphore:mutex=1;
Repeat
Wait(mutex);
        Critical section
Signal(mutex);
        Remainder section
Until false;
```

▲semaphore 可用來解決各種同步問題。例如 P1 程序中有 S1 敘述，P2 程式中有 S2 敘述。假設要求 S2 只能在 S1 執行完後再執行。要完成這要求，我們讓 P1 與 P2 共享一個 synch 號誌(synch 初值為 0)，做法如下：

```
P1：
        S1;
        Signal(synch);
P2：
        Wait(synch);
        S2;
```

這樣即可確保 P2 在 P1 執行過 S1 後才執行 S2；

★Spinlock(自旋鎖)

??汪八回 p50 是說 busy waiting 的方式即稱做 spinlock!!??

之前所提的多種方法，都是用 busy waiting 的方式，亦即在等待進入 CS 的 processes 都要過迴圈循環的方式進行等待，執行這些迴圈會浪費 CPU 的時間。因此應用非 busy waiting 方式的 semaphore 就稱之為 spinlock。

Spinlock 的優點是當 process 在等待一個鎖時，不需進行任何 context switching。

九：Counting semaphore：

★Suspend、wakeup 的方式

為了克服 busy waiting 的方式，將要等待進入 CS 的 process 予以停滯(block)，停滯可將一個處理程序送入號誌對應的等待佇列之中，且將該 process 的 state 改為 waiting state。之後系統的控制便交由 CPU 的排程程式，由它來選擇另一個處理程序運作。

```
Type semaphore=record
        Value:integer;
        L:list of process;
End;
Var S:semaphore;
```

每個號誌均有一個整數值及一佇列。當一個處理程序必須等待一個號誌時，它就加入處理程序的佇列。Signal 運作則可以移走等待佇列中的一個處理程序，並喚醒該處理程序。

運作方式如下：

Counting semaphore

Wait(S):
S.value:=S.value-1
if S.value<0 then
begin
add this process to S.L;
block(p);
end
Signal(S):
S.value:=S.value+1;
if S.value<=0 then
begin
remove a process P from S.L;
wakeup(P);
end;

Value 的絕對值就是在等待號誌的 process 數目

這個做法是用串列把等著要進入 CS 的 process 串起來，再一個一個叫醒

▲Bounded Buffer Producer-Consumer Problem

利用號誌來完成生產者-消費者問題，共用變數：

mutex：CS 互斥用，初值為 1

empty：判斷 buffer 是否為空，初值為 n

full：判斷 buffer 是否滿了，初值為 0

```
生產者：
repeat
…
produce an item in nextp
…
wait(empty);
wait(mutex);       //lock CS
…
add nextp to buffer
…
signal(mutex);       //release CS
signal(full);
until false;
```

```
消費者：
repeat
wait(full);
wait(mutex);       //lock CS
…
remove an item from buffer to nextc
…
signal(mutex);       //release CS
signal(empty);
…
consume the item in nextc
…
until false;
```

▲Reader/Writer Problem

為了避免同時有 2 個以上程序對相同物件進行寫入，必需保護共享物件，當有寫入者要更新共享物件時採取互斥。

Var mutex=1

wrt:semaphore=1

readcount:integer=0

```
寫入者：
wait(wrt);
…
writing is performed;
…
signal(wrt)
```

```
讀取者：
wait(mutex);
readcount:=readcount+1;
if readcount=1 then wait(wrt);
signal(mutex);
…
reading is performed
…
wait(mutex);
readcount:=readcount-1;
if readcount=0 then signal(wrt);
signal(mutex)
```

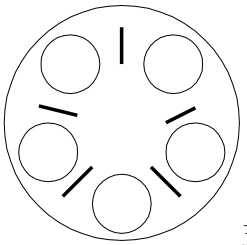
if readcount=1 then wait(wrt);

第一個讀取者要負責上鎖，避免寫入者做寫入動作

if readcount=0 then signal(wrt);

最後一個讀取者離開，要開鎖，才能讓寫入者寫入

▲哲學家用餐問題



五個人、五根筷子

哲學家問題也是一種程序同步的問題，在上述的情況中如果每人都拿起一根筷子就死結了

```
Repeat
   wait(chopstick[i]);
   wait(chopstick[i+1 mod 5]);
      ……
      eat
      ……
   signal(chopstick[i]);
   signal(chopstick[i+1 mod 5]);
      ……
      think
      ……
until false
```

上述用 semaphore 解決了 CS 的問題，不過還是存在著 deadlock、starvation 的問題

▲deadlock 的避免方式：

1. 最多脫許 4 個人同時提出用餐的 request
2. 只有當身邊的兩根筷子都能用時，才允許拿筷子
3. 奇數哲學家拿左邊，再拿右邊，偶數先拿右再拿左

△但還是會有 starvation 的問題，該如何避免?

▲使用 semaphore 的一些問題

死結現象

P0	P1
Wait(S); Wait(Q); Signal(S); Signal(Q);	Wait(Q); Wait(S); Signal(Q); Signal(S);

九：Monitor：

Monitor 是一個高階同步結構。Monitor 的主要特徵是由一組變數的宣告以及在這些共享變數上運行的程序和函數所組成。

監督程式型態內的資料並不能直接被各個處理程序所使用。監督程式結構可以保證在某時刻只有一個處理程序可以在監督程式內部活動。

▲運用 semaphore 完成 monitor 的製作

九：Message Passing：

1. 直接通訊

   send(P, message)　　//發送一段訊息給 process P

   receive(Q, message)　　//從 process Q 接收一段訊息

2. 間接通訊

   透過共用的 mailbox 或稱 port 來發送及接收

   send(A, message)　　//對郵箱 A 發送一段訊息

   receive(A, message)　　//從郵箱 A 中接收一段訊息

九：Rendezvous：

訊息的傳送者與接收者之間並沒有緩衝區，發送者發送訊息時必須等待，直到接收者接收了這段訊息時為止，為了使訊息傳輸得以進行，兩個 process 必須保持同步。這種同步稱之為「一次相遇」(rendezvous)。

十：open file、close file：

Open file	1. 避免一直去目錄查詢 2. OS 會有一張表格，記錄已開啟的檔案資訊，減少每次都重新查詢目錄的時間
Close file	當檔案不再使用時，就將它 close，亦即從表格中刪除

十：檔案存取的方式：

1. Sequential Access
2. Direct Access
3. Index

十：目錄結構：

Single level	
Two level	
Tree structured	
Acyclic graph	
General graph	

十：File Protection：

1. Name Protection：不知道檔名就不能用
2. Password Protection：檔案加上密碼
3. Access List：根據使用者的身份
4. Access Group：同上，再加上群組功能

十二：有的沒的名詞解釋：

NFS：Network File System	將各種不同的系統利用一種協定串連在一起，不管任何的作業系統，從大型主機到個人電腦，NFS 對使用者都是都是透明的。透過 NFS，你可以隨時存取別台電腦，彷彿是自己的機器一樣。
Process Migration	就是把某種作業平台上的程序移植到另外一種相異的作業平台上
Race Condition	兩個或更多的獨立任務同時訪問或修改同一狀態信息時出現的狀態。這種狀態可能導致系統行為的不一致，因此，這是幷行系統設計中的根本問題