Process concept:

\*process VS program

Program:被動，儲存在disk的binary file

Process: 主動，啟動於memory的program (放入memory才能給CPU執行)

\*process的組成

Memory會開幾塊給process

程式碼區:

1. Code segment (text section)

變數區:

2. Data section --- global variable (一開始就存在，且全域)

3.Stack --- temporary local variable 、 function

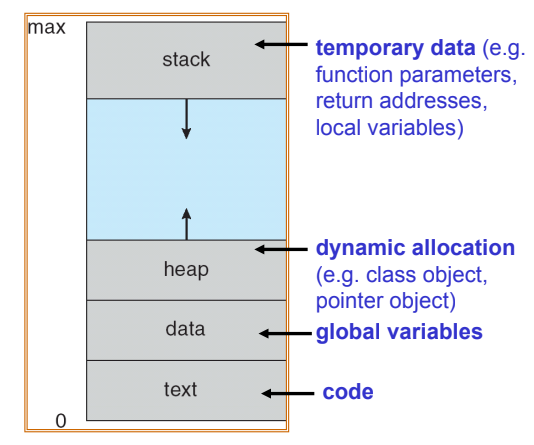
4.Heap --- dynamic variable

(Stack 與 heap 是放兩邊，往中間長的)

Meta data 區: (Time sharing 會切換，所以要記住目前資訊)

5. 目前資訊 (program counter, register content(暫存在memory的register資料))

6. resource 資訊 (open file handlers 、 file token …) (用以紀錄使用資源)



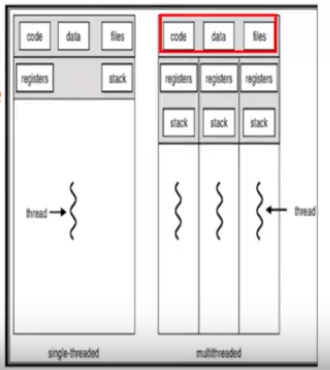
\*threads

輕量級的 process

同一個process 的thread是一些共用空間的 (code、data section、OS resource)

但process 必須有自己的register、pc、thread ID 與 stack (想想function例子)

是CPU執行的最基本單位 (process可以切成多個process)

 (multi Vs single thread)

Process與 thread管理方式是類似的，故有些OS不會區分它們

\*Process的狀態

1.New (初始化、load到memory)

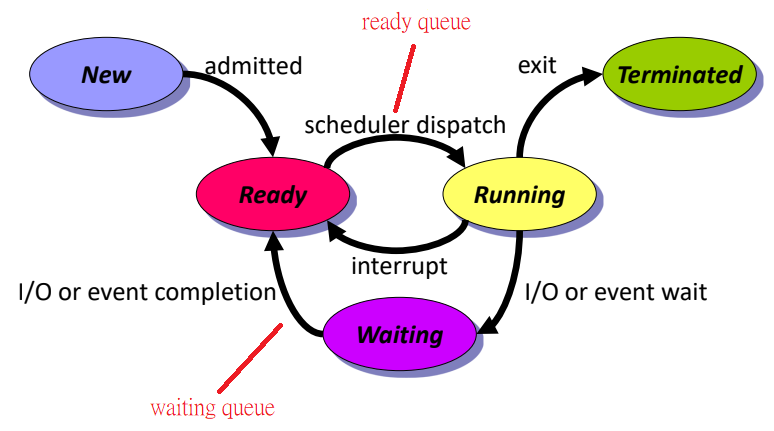
2.Ready (放到ready Queue裡，準備被放到CPU中)

3.Running (放在CPU中執行)

4.Waiting (等待事件(如IO)，放在waiting queue)

5.Terminate (process結束執行)

\*\* process cycle



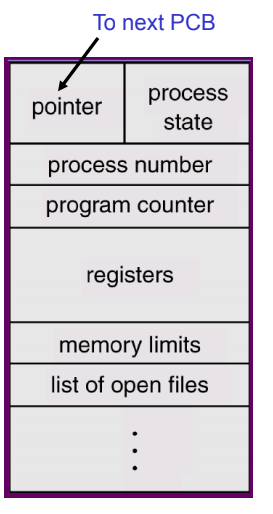
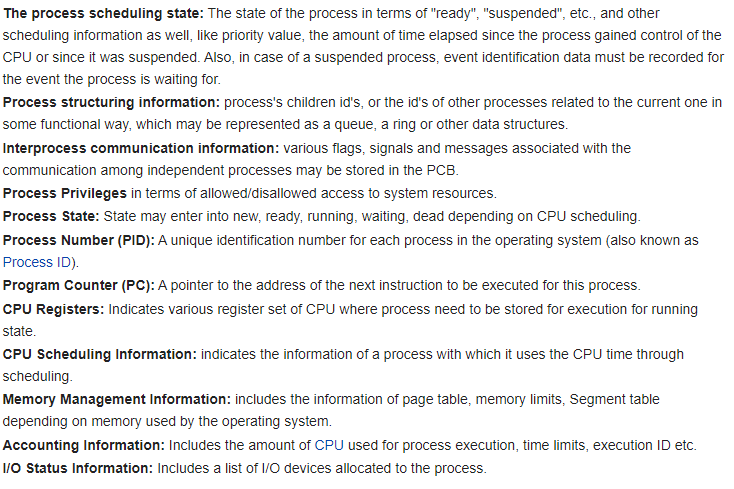
\*Process Control Block (PCB) 實作上述管理process的概念

OS會在(OS的)memory創建PCB來記錄各個process資訊

例如process state、PC、register、IO status(哪個IO的waiting queue)、priorty

CPU scheduling 、 其他資訊

Queue相當於放PCB位置的queue，所以PCB會有pointer指向下一個PCB(也可以用以實作ring等其他struct)

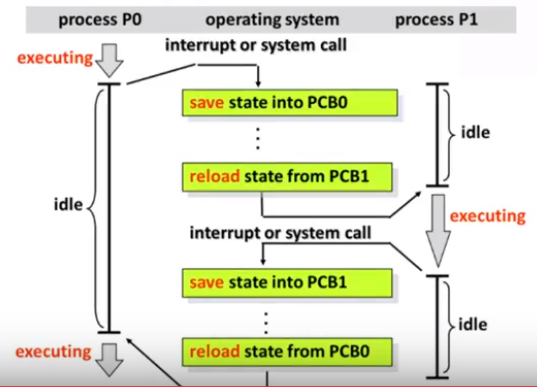
 

Context switch (Time sharing 的基礎)

一定是因為interrupt 或 自己叫system call 產生 content switch

要記錄原process的資訊於PCB

然後要reload 另一process於PCB (注意content switch期間，兩個程式都是idle的) ((某個角度是浪費時間，但又不得不需要))



Swich 速度因素

1. memory 速度 (load)
2. register數量
3. 特殊instruction (讓save 、reload一次完成)
4. Hardware support (如:mutiple sets of registers，及設計一組register記程context swich就不用access memory了) 相當於用register替代memory