A.Bootstrap program

\*開機後執行的第一支程式

\*存在ROM中

\*會載入boot loader

\*舊版:BIOS: 沒有filesystem

新版 UEFI: 有自己的filesystem

Boot loader工作

1. 找到kernal image (舊版要先在block level中找/新版可以用自己的filesystem)
2. 取得kernal argument
3. 啟動並傳入kernal argument

Kernal的啟動工作

1. 初始化硬體 (CPU、device controller 、memeory)
2. 啟動OS { mount root filesystem,

執行init (systemd), \*init可以說是所有process的根結點

開始其他daemon,}

\*daemon系統服務 (狹義上就是執行服務的程式)

完成之後會給 軟體的interrupt (透過system call 給)

B. multiprogram VS Multitask

前者是batch: 單純吃入所有程式，然後依次執行(queue)

後者是time sharing : 快速切換製造多工假象

C.system call

一個API能讓OS發出trap(軟體發出的interrupt)

可以讓使用者使用OS的服務(讀檔、目錄 …)

由組語寫成

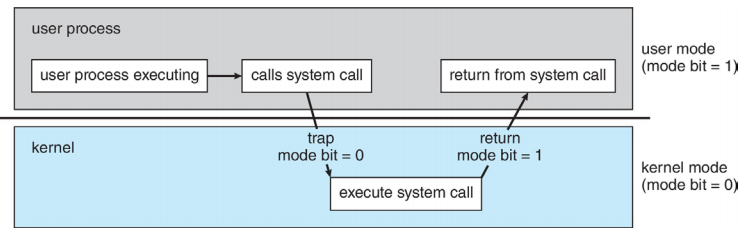
D.Duel mode

如果沒有duel mode，CPU並不知道如何區分OS與使用者

代表user mode(1) 與 kernal mod(0)

開機(bootstrap)以及使用到資源(危險行為)都須由kernal mode執行

目的是保護其他程式不受影響



Mutimode

Duel mode的延伸，給予了VM空間

例如:intel有四個空間，1、3給kernal與user，其他就給VM

(只有kernal mode)可以使用 Privileged instructions (OS才能執行)

E.security VS protection

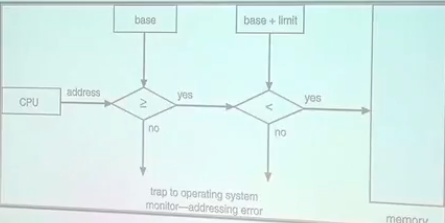
前者是防範外部惡意程式

後者是防止內部資源衝突

(保護IO:所有IO都是Privileged instruction)

(保護memory : 使用Base register(起點)與Limit register(長度)，會記錄程式的區間，不能使用超出的空間(segmentation fault)，malloc之類的也就是去調整區間長度，當然malloc這些最後還是要privileged instruction)

(保護CPU:要可以從使用者那搶走使用權(infinite loop) => 使用time sharing的好處=>Hardware有Timer(也是只有OS能調)會定時切換)



F.Virtualization

用軟體模擬硬體的行為

(欺上瞞下: 讓使用者以為有兩台電腦，讓下層以為有多個裝置)

其VMM(Virtual machine monitor負責管理VM)角色類似於OS

\*虛擬化的用途

1.區隔(隔離程式影響)

2.合併(共同作業)

3.遷移(搬資料)

