特權等級

概論

在 segment 的層次中，有四種特權等級，由 0 到 3。第 0 級（又稱 ring 0）是權力最大的，而第 3 級（又稱 ring 3）則是權力最小的。一般的情形中，應用程式是在 ring 3 中執行，而系統核心則在 ring 0 中執行。大部分的作業系統只用到兩個等級，即 ring 0 和 ring 3（只用到兩個等級的作業系統，應該要使用 ring 0 和 ring 3）。不過，如果有特別需要，也可以使用權力介於 ring 0 和 ring 3 之間的 ring 1 和 ring 2。例如，在微核心（micro kernel）系統中，微核心在 ring 0 中執行，而系統服務則可以在 ring 1（或 ring 2）中執行。
在一般的情形下，權力較低的程式不能存取權力較高的程式或資料。事實上，即使是權力較高的程式，也不能直接執行權力較低的程式，而必須經由 call gate 來呼叫。當然，也是有例外的情形。
特權等級有三種：

- 目前特權等級（Current Privilege Level，CPL）：目前執行的程式或工作的特權等級，存放在 CS 和 SS 的第 0 和第 1 位元（參考「記憶體管理」的「分段架構」）。一般情形下，CPL 會和目前程式所在的 segment 的特權等級（即 DPL）相同。而在轉移控制權時（例如，用 JMP 或 CALL 命令跳到另一 segment 中），CPL 會改變為新的 segment 的 DPL。不過，有一個例外：在轉移控制權到一個 conforming 的 segment 中的時候，CPL 並不會變為新的 segment 的 DPL，而還是維持呼叫者的 CPL。
- 分段特權等級（Descriptor Privilege Level，DPL）：一個 segment（或 gate）的特權等級，存放在 segment descriptor 中的 DPL 位元（參考「記憶體管理」的「分段架構」）。一段程式在存取一個 segment（或 gate）時，處理器會比較 CPL 和 segment 的 DPL，及 segment selector 的 RPL。對不同型態的 segment，處理器的處理方式也不大相同：
  - 資料 segment 的 DPL 指出存取該 segment 所需要的最低特權等級。例如，若一個資料 segment 的 DPL 為 2，則程式的 CPL 必需是 0、1、或 2 才能存取這個 segment。
  - Non-conforming 的程式 segment 的 DPL 指出「直接呼叫」（不經由 call gate）時，呼叫者所需要的特權等級。例如，若一 non-conforming 的程式 segment 的 DPL 為 1，則只有 CPL 為 1 的程式才能直接呼叫它。
  - Call gate 和 TSS 的 DPL 指出存取該 gate 所需要的最低特權等級。它們的規則和資料 segment 是一樣的。
  - 經由 call gate 呼叫程式 segment 時，其 DPL 指出呼叫者所容許的最高的特權等級。例如，若一程式 segment 的 DPL 為 2，則只有 CPL 為 2 和 3 的程式才能透過 call gate 呼叫它。
- 要求特權等級（Requested Privilege Level，RPL）：在存取一個 segment 時，可以在 segment selector 中指定 RPL（參考「記憶體管理」的「分段架構」），來要求以某個特定的特權等級來存取該 segment。處理器在檢查特權等級是否相符時，會以 CPL 和 RPL 中較低者來決定。所以，即使 CPL 的等級可以存取某個 segment，若 RPL 的等級不足，還是不能存取該 segment。RPL 可以用來避免系統程式以不正確的特權等級存取某個 segment。例如，某個作業系統中的 API 會把一些資料寫到使用者程式提供的 segment 中。假設系統 API 在 ring 0 中執行，而程式在 ring 3 中執行。若沒有特別檢查，則使用者可以把一個 DPL 為 0 的 segment（使用者程式不能存取它）傳到該 API 中，因為 API 有寫入該 segment 的權力，因此使用者程式就可以破壞該 segment 中的資料。為了避免這個問題，系統 API 在存取使用者傳入的 segment 時，可以先把 segment selector 的 RPL 設定成和使用者程式的 CPL 相同，就不會意外寫入原先使用者無權存取的 segment 了。

特權等級的檢查，是在把 segment selector 載入分段暫存器的時候進行的。對堆疊 segment 而言，雖然在性質上類似資料 segment，但是堆疊 segment 的 segment selector 在載入 SS 暫存器時，其 CPL 和 segment 的 DPL 必需相同（如同 non-conforming 的程式 segment 一般），否則會導致 general-protection fault（#GP）。
（註：在本章中，有關各個特權等級，以「較小」或「較大」來表示數值上的大小，而以「較高」或「較低」來表示權力的高低，以避免混淆。）

控制權轉移

程式可以經由執行 JMP、CALL、RET、INT n、和 IRET 指令來轉移控制權。在處理器發生例外（exception）、或是中斷（interrupt）、及 IRET 指令是比較特別的（參考「中斷 / 例外處理」）。
除了中斷和例外之外，只有 JMP、CALL 和 RET 可以進行跨 segment 的跳躍。在同一個 segment 中的跳躍（如 near 的 JMP、CALL、RET 和各個條件分支指令）並不會有任何限制。但是在跨 segment 的跳躍時，就會檢查相關的權限是否允許進行跳躍。跨 segment 的跳躍有以下幾種情形：

直接跳躍到另一個程式 segment：

當使用 CALL 或 JMP 直接跳躍到另一個程式 segment 時，會檢查目前的 CPL、目標 segment descriptor 的 DPL、目標 segment selector 的 RPL、和目標 segment descriptor 的 C 旗標。如果 C 旗標是 0，表示目標 segment 是一個 nonconforming 的程式 segment。在跳躍到 nonconforming 的 segment 時，CPL 一定要和目標的 DPL 相同，而且 RPL 一定要小於或等於 CPL，否則會導致 general-protection fault（#GP）。而且，在跳躍到 nonconforming 的 segment 時，CPL 並不會改變，即使 RPL 比較小也是一

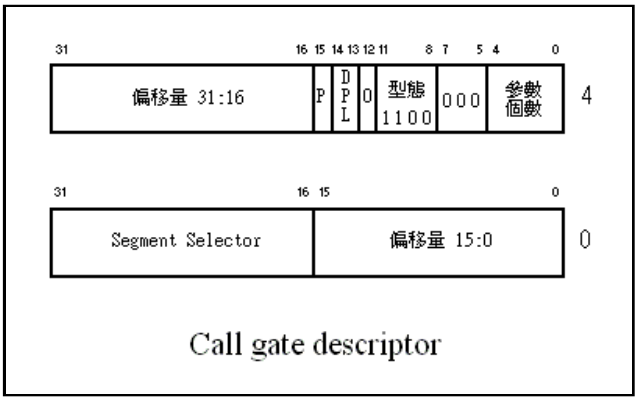
樣。
如果 C 旗標是 1，表示目標 segment 是一個 conforming 的程式 segment。在跳躍到 conforming 的程式 segment 中時，CPL 可以大於（權力較低）或等於 DPL；只有在 CPL 小於 DPL 時，才會導致 general-protection fault（#GP），而 RPL 則沒有任何影響。即使是在跳躍到 DPL 比較高的 conforming 的 segment 時，CPL 也不會改變，且因為 CPL 沒有改變，也不會進行堆疊切換（stack-switch）。
在作業系統中，可以把一些不會使用系統保護的部分的 API（例如，大部分的數學函式、和例外處理程式），設成 conforming 的 segment。這樣，一般的應用程式就可以直接使用這些 API。在呼叫這些 segment 時，CPL 並不會改變，可以避免在一個應用程式呼叫一個 DPL 權力較高的 segment 時，以較高的權力改變了某些應用程式不能改變或存取的地方。
大部分其它的程式 segment 都應該設定成 nonconforming。

經由 gate：

如果一個應用程式無法呼叫權限（DPL）較高的程式 segment，那麼要怎麼讓程式呼叫系統的 API（通常權限較高）呢？答案是：經由 gate 呼叫。Gate 是一種系統 segment，而它的 segment descriptor 稱為 gate descriptor。Gate 有四種：call gates、trap gates、interrupt gates、和 task gates。在這裡，只討論 call gates；trap gates 和 interrupt gates 在「中斷 / 例外處理」中會說明，而在「多工處理」中會說明 task gates。利用 call gate，才可以在不同權限的程式之間來回。

Call gates

Call gates 的 descriptor 格式如下（參考「記憶體管理」的「分段架構」）：



Call gate descriptor 可以在 GDT 或 LDT 中，不過不能在 IDT（Interrupt descriptor table）中。它指出了程式所在的 segment（由 Segment Selector 欄位指定），並指出了程式在該 segment 中的偏移量（程式的進入點），及呼叫者所需要的特權等級（由 DPL 指定）。如果需要堆疊切換，它還指出需要參數的個數。而上面的 P 則是表示 call gate 是否有效。
P 通常是設為 1，表示這是一個有效的 call gate。有時候，在某些特殊的情形下，會想要把 P 設為 0。例如，想要知道這個 call gate 被呼叫了幾次，可以把 P 設為 0，則在呼叫這個 call gate 時，會發出 not-present 例外。在例外處理程式中，把計數加一，再把 P 設為 1，讓程式繼續執行，這樣就可以追蹤這個 call gate 被呼叫的次數了。
用 CALL 命令或 JMP 命令都可以呼叫 call gate。呼叫時，把 segment selector 指向要呼叫的 call gate descriptor，而偏移量還是要指定，但是會被忽略（在 call gate 中有偏移量）。經由 call gate 呼叫時，CPL 和 RPL 都要小於 call gate 的 DPL；也就是說，call gate 的 DPL 指定「可容許的最高權限」。如果呼叫者的 CPL（或 RPL）比 call gate 的 DPL 還小（權限更高），則不能呼叫。
在使用 CALL 命令呼叫 call gate 時，無論目標 segment 是否為 conforming，該 segment（非 call gate）的 DPL 都必須小於或等於呼叫者的 CPL。但若用 JMP 命令呼叫 call gate 時，在目標 segment 是 conforming 時，和使用 CALL 命令呼叫時相同。但是若目標 segment 是 nonconforming，則目標 segment 的 DPL 必須和呼叫者的 CPL 相同，才能呼叫。也就是說，只能用 CALL 命令呼叫權限較高的程式，而不能用 JMP 跳到權限較高的程式碼中。
利用 call gate，可以在同一個程式 segment 中的各個程序設定不同的特權等級。例如，作業系統核心的程式 segment 中，有些程序是可供應用程式呼叫的 API；有些則是內部使用的程序，不希望由應用程式呼叫。這時，就可以把 API 的 call gate 的 DPL 設為 3，而將內部程式的 call gate 的 DPL 設為 1 或 0。
在經由 call gate 呼叫，將控制權轉移到新的程式 segment 之後，CPL 會變成和程式 segment 的 DPL 相同。

堆疊切換

為了避免權限較高的程式被權限較低的呼叫者影響，在 CPL 改變時，處理器會進行堆疊切換。每一個 task 對所有用到的特權等級，都要維護分開的堆疊。在 ring 3 中的堆疊指標，是直接存放在 SS 和 ESP 中；而 ring 2、ring 1、ring 0 的堆疊指標（由一個 segment selector 和一個 offset 組成）則存放在 TSS 中。當 CPL 變小時，處理器會載入 TSS 中相對映的堆疊指標，切換到新的堆疊。這三個堆

疊指標在程式執行途中是不能改變的。當然，如果作業系統只用到兩個特權等級（例如，只用到 ring 0 和 ring 3），則可以只維護兩個堆疊。在特權等級較高的程序返回到 ring 3 的程序時，會回復原來的 SS 和 ESP。因為在切換特權等級時，會自動進行堆疊切換，所以即使作業系統不使用多工處理的能力，也必須定義一個 TSS，來指定這些堆疊。

作業系統必須負責維護適當的堆疊空間給各個權限使用，並在 TSS 中指定適當的堆疊指標。堆疊的大小必須夠大，至少要能放得下呼叫者的 SS、ESP、CS、EIP，和程序的參數，及程序在執行時所需的區域變數。如果程序是中斷或例外處理程式，則還要能放下 EFLAGS 和錯誤碼。

由於在呼叫權限較高的程序時，會有堆疊切換，因此還必須把呼叫者所推到堆疊中的參數複製到新的堆疊中。在 call gate 中的「參數個數」欄位，就是用來指定參數的個數（最多可達 31 個）。處理器在堆疊切換時，會依序把呼叫者的 SS、ESP、參數、CS、和 EIP 推到新的堆疊中。如果程序所需要的參數超過 31 個，則可以傳入一個指標，指向一塊資料結構，或是利用呼叫者的 SS 和 ESP 來取得呼叫者的堆疊中存放的參數資料。

由程序中返回

在程序中，遇到 RET 指令時，會回到呼叫者程序中。近程（在同一 segment 中）的 RET 指令，因為沒有牽涉到權限的改變；因此，處理器只會進行邊界檢查，確保堆疊中存放的返回位址是在 segment 的範圍之內。但遠程的 RET 指令可能會需要在返回時，同時改變 CPL；因此，除了邊界檢查之外，處理器還會檢查返回的目標的 segment 的權限（即該 segment 的 DPL）是否大於或等於目前的 CPL。因為，只有權限較低的程序可以呼叫權限較高的程序，因此在返回時，目標 segment 的權限必然是比較低的。如果不是的話，就表示堆疊中返回位址是不正確的。

在返回時，處理器利用堆疊中存放的 CS 中的 RPL 值來判斷是否需要改變權限。如果 RPL 的值比目前的 CPL 還大（權限較低），則表示在返回時，需要改變權限。如果需要改變權限，則同時也需要切換堆疊。因此，處理器會在載入 CS、EIP 之後，再載入堆疊中存放的 SS、ESP（在呼叫程序時推入堆疊中，參考上節說明）。當然，處理器還是會檢查 SS 和 ESP 是否合法。最後，處理器會檢查各個資料分段暫存器（DS、ES、FS、GS）是否指向 DPL 比新的 CPL 更小（權限更高）的 segment。處理器會把指向 DPL 較小的分段暫存器設為 null selector；不過，指向一個權限較高的 conforming 的程式碼 segment 原本就是合法的，所以如果某個資料分段暫存器是指向 conforming 的程式碼 segment，則不會被設為 null selector（參考「記憶體管理」的「分段架構」）。