System call

是 application program 與 OS 的溝通

System call 就是Service的實作function

System call 有以下特性

1. 是OS的一部分
2. 是kernal mode 下使用的 software interrupt
3. 是使用組合語言寫的

所以我們會使用API(application program interface)，去使用 system call

API是為了方便programmer (如 C-Libarary)

API只有使用面 如int exp(int n,int e);

實作則在libarary裡

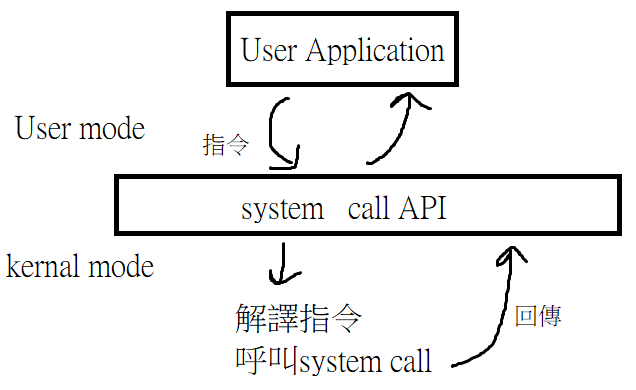
一個API有可能是0個或多個system call組成

Ex: printf() 可能用數十道systemcall，abs()並不需要system call

\***API不一定要使用OS system call**

\*system call需要越簡單越好 (interrupt不希望太久)

常用API 如 win32API、POSIX API、JAVA API (unix..)(不同hard ware會導致不同compile結果)



API 優點

簡化

Portable

效率 (幫使用者用最有效率的方式implement)

System call如何傳參數?

1.使用register (a0~a3 register)

2.table in memory (register傳pointer)

3.使用process中的data memory中的stack (program push，OS pop)

System call 種類

Process control (exec、load)

註解

fork == 創造procee，因為從systmd clone出process

wait(): 等待前一個process完成

File management (open、read)

Device management (request device , release device)

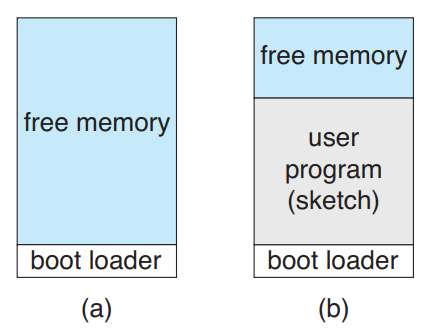
Information maintain (get time get date)

Communication( send message)

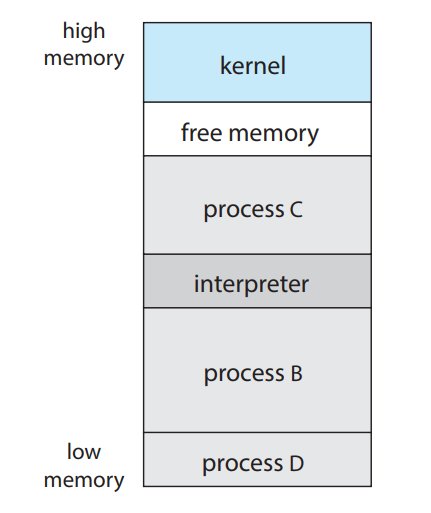
Protection (get permission)

如果只需要single task,不需要OS

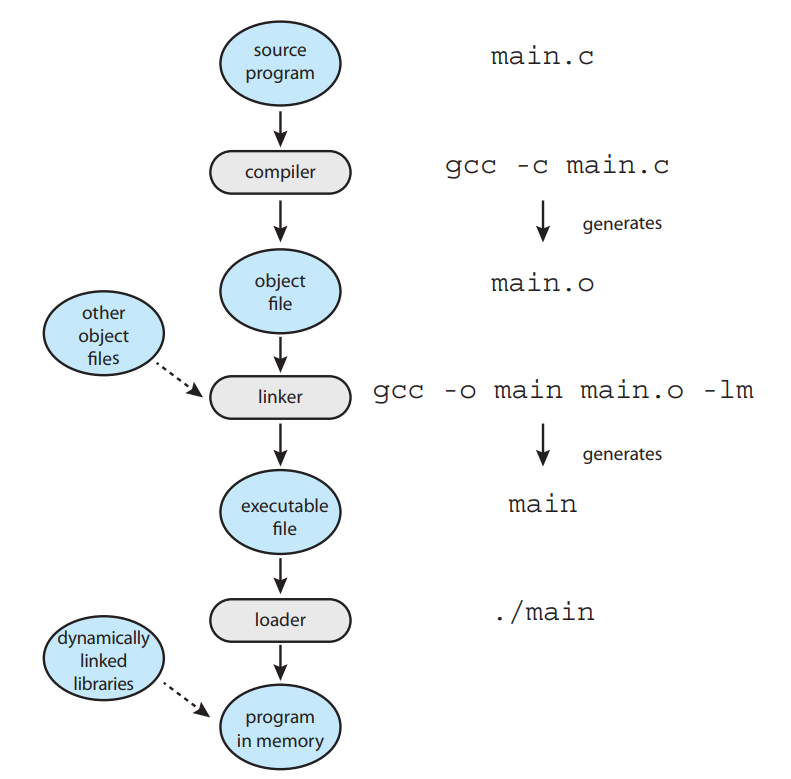
只要一塊free memory與 boot loader就好



如果是多工就需要定義error code、優先序、Kernal ….



注意DLL是在load前才加入



通常OS會針對編譯後的檔案給予格式，含有編譯的mechine code與 metadata

如:

ELF(Executable & Linkable format) linux 的標準可執行檔格式

可以使用file 指令去看檔案

如 file main.o 會得到ELF relocatable file，file main 會的到ELF executable file

此外windows 是 portal executable (PE) format

Mac 是 Mach-O

作業系統沒有可攜性原因

System call實作方式不同

解決辦法

1. 直譯語言
2. Virtual machine (JAVA)
3. 訂定標準 (POSIX)

然而仍舊可能有一些格式、定義上的不同導致搬移失敗