|  |  |
| --- | --- |
| 描述Linux運行級別0-6的各自意思. | 0: 關機模式  1: 單用戶模式 (破解root密碼)  2: 無網路支持的多用戶模式  3: 有網路支持的多用戶模式 (工作最常使用到的)  4: 保留,未使用  5: 有網路支持的X-windows支持多用戶模式(桌面)  6: 重新引導系統,就是重新啟動電腦的意思 |
| 描述Linux系統從開機到登陸介面的啟動過程. | 1. 開機BIOS自己檢查，加載硬碟。  2. 讀取MBR→MBR引導。  3. grub引導Menu ( Boot loader )。  4. 加載kernel.  5. 加載init process,根據initab文件設定運行級別。  6. init process,執行rc.sysinit文件。  7. 啟動kernel model,執行不同級別的腳本程式。  8. 執行/etc/rc.d/rc.local  9. 啟動mingetty,進入系統登陸介面 |
| 給出正確的關機和重新啟動服務器的命令. | 1. shutdown  [-t]指定在多長時間之後關閉系統  [-r]重啟系統  [-k]並不真正關機，只是給每個登入用戶發送警告信號  [-h]關閉系統(halt)  2. halt  halt 是最簡單的關機命令，原理就是調用sutdown-h命令，halt執行的時候，殺死應用process,完成後就會停止kernel.  halt命令的部分參數如下:  [-f]沒有調用shutdown而強制關機或重新啟動  [-i]關機或重新啟動之前，關掉所有網路的接口  [-p]關機時調用poweroff    3. reboot  reboot工作過程與halt類似，作用是重新啟動，而halt是關機。  4. init  init 是所有process的祖先，其Process號始終為1. init用於切換系統的運型級別，切換的工作是立即完成的。  而如果用init()命令將系統運行級別切為0，則為關機的意思，而如果是6則為重新啟動。 |
| 請描述下列路徑的內容是做些甚麼事情的.  1) /var/log/messages :\_\_\_\_\_\_\_\_\_\_\_\_\_  2) /var/log/secure : \_\_\_\_\_\_\_\_\_\_\_\_\_\_  3) /var/spool/clientmqueue: \_\_\_\_\_\_\_\_\_  4) /proc/inierupts: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  5) /etc/fstab: \_\_\_\_\_\_\_\_\_\_\_\_  6) /etc/profil: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | 1) /var/log/messages : 系統日誌文件  2) /var/log/secure : 系統安全文件(顯示登陸訊息的文件)  3) /var/spool/clientmqueue: 例行性任務回執郵件存放文件  4) /proc/inierupts : 當前系統中斷報告文件  5) /etc/fstab : 關機自動掛載磁碟的配置文件  6) /etc/profile : 環境變量存放的文件 |
| LINUX 命令及作用  1. cp :  2. mv :  3. mkdir :  4. cd :  5. cat :  6. ls :  7. rm :  8. find :  9. rmdir :  10. echo :  11. hostname:  12. whoami :  13. useradd :  14. passwd :  15. su :  16. alias :  17. head :  18. tail :  19. touch :  20. pwd : | 1. cp : 複製  2. mv : 移動文件或目錄  3. mkdir : 創建目錄  4. cd : 切換目錄  5. cat : 查看文件內容  6. ls : 查看目錄下文件  7. rm : 刪除文件或目錄  8. find : 查找文件或目錄  9. rmdir : 刪除空目錄  10. echo : 顯示輸出  11. hostname: 主機名  12. whoami : 查看當前用戶  13. useradd : 添加用戶  14. passwd : 改密碼  15. su : 切換用戶角色  16. alias : 查看及設置別名  17. head : 查看文件前N行 (default :10行)  18. tail : 查看文件後N行 (default :10行)  19. touch : 修改文件時間或建置新檔  20. pwd : 顯示目前所在的目錄 |
| 關鍵字static的作用是什麼？ | 1. 在函數體，一個被聲明為靜態的變量在這一函數被調用過程中維持其值不變。  2. 在模塊內（但在函數體外），一個被聲明為靜態的變量可以被模塊內所用函數訪問，但不能被模塊外其它函數訪問。它是一個本地的全局變量。  3. 在模塊內，一個被聲明為靜態的函數只可被這一模塊內的其它函數調用。那就是，這個函數被限制在聲明它的模塊的本地範圍內使用。 |
| 協處理器主要控制：\_\_\_\_\_\_、\_\_\_\_\_\_、\_\_\_\_\_\_。 | 片內的MMU、指令和數據緩存（IDC）、寫緩沖（Write Buffer） |
| 關鍵字volatile有什麼含意?並給出三個不同的例子。 | 1. 並行設備的硬件寄存器（如：狀態寄存器）  2. 一個中斷服務子程序中會訪問到的非自動變量(Non-automatic variables)  3. 多線程應用中被幾個任務共享的變量 |
| 中斷是嵌入式系統中重要的組成部分，這導致了很多編譯開發商提供一種擴展—讓標準C支持中斷。具代表事實是，產生了一個新的關鍵字 \_\_interrupt。下面的代碼就使用了\_\_interrupt關鍵字去定義了一個中斷服務子程序(ISR)，請評論一下這段代碼的。  \_\_interrupt double compute\_area (double radius)  {  　　 double area = PI \* radius \* radius;  　　 printf("\nArea = %f", area);  　　 return area;  } | 1. ISR不能retrun一個值。  2. ISR不能傳遞參數。  3. 在許多的處理器/編譯器中，浮點數一般都是不可重入的，有些處理器/編譯器不允許在ISR中做浮點數的運算，此外，ISR應該是短而且有效率的，在裡面做運算，是不明智的。  4. printf()經常有重入和性能上的問題，所以在這裡面printf東西，是很沒效率的。 |
| 當一個異常出現以後，ARM微處理器會執行哪幾步操作？ | 當一個異常出現以後，ARM微處理器會執行以下幾步操作。  Step1. 將下一條指令的地址存入相應連接寄存器LR，以便程序在處理異常返回時能從正確的位置重新開始執行。若異常是從ARM狀態進入，則LR寄存器中保存 的是下一條指令的地址（當前PC＋4或PC＋8，與異常的類型有關）；若異常是從Thumb狀態進入，則在LR寄存器中保存當前PC的偏移量，這樣，異常 處理程序就不需要確定異常是從何種狀態進入的。例如：在軟件中斷異常SWI，指令 MOV PC，R14\_svc總是返回到下一條指令，不管SWI是在ARM狀態執行，還是在Thumb狀態執行。  Step2. 將CPSR複製到相對應的SPSR中。  Step3. 根據異常類型、強制設置CPSR的運作模式。  Step4. 強制PC從相關的Exception向量位址取下一條指令執行，從而跳轉到相對應的exception處理程序處。 |
| 1. ARM微處理器在較新的體系結構中支持兩種指令集：\_\_\_\_\_\_、\_\_\_\_\_\_ | ARM指令集、Thumb指令集 |
| 多線程如何同步? | Linux系統中的mutiple thread最常用的是 1.互斥鎖、條件變量、信號量。 |
| 我們常常會需要和TCP和UDP打交到，可否敘述一下，TCP與UDP的區別是什麼? | 1.TCP: 是面向連接流的傳輸控制協議，具有高可靠性，確保傳輸數據的正確性，有驗證重發的機制，不會出現丟失或亂序的錯誤。  2.UDP:  是無連接的數據服務器，不對數據進行檢查與修改，無需等待對方的應答，會出現分組丟失、重複、亂序，但具有較好的實時性，UDP段結構比TCP的段結構簡單，因此，網路的開銷也小。 |
| 請解釋大小端 (Big-Endian/Little-Endian) | Little-Endian就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。  Big-Endian就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。 |
| 請簡單描述一下，Process/thread的區別是什麼。 | process:  已經被load到記憶體中，任何一行code隨時會被cpu執行，且其宣告的在記憶體的變數的值會隨需求而不斷變動，我們稱為[ 程序 ] 也就是活的Program，一個多工作業系統(Multitasking Operating System)可以同時運行多個Process然而一個CPU一次只能做一件事情，但CPU的數量永遠少於運行中的Process數，因此每個Process使用的時間需要被排程(Scheduling),又每個Process間在記憶體中，如果擺放的方式不當，就會在記憶體中產生很多沒辦法用到的碎片，因此Memory Mnagement是一個問題，每個Process所需要的記憶體總合，也可能大於實體記憶體，因此需要另外用二次儲存裝置充當虛擬記憶體(VirtualMemory)，但是二次儲存裝置的速度肯定很慢，因此如何做到對虛擬記憶體最小的依賴，盡量避免Page Fault(電腦在主記憶體中找不到資料，而要去二次記憶體找，就稱為Page Fault)防止Thrashing的發生(因為VirtualMemory演算法不當，造成幾乎每次存取都要依賴二次記憶體，就是Thrashing)，以達到效能最佳化。  Thread ：在同一個Process底下，有許多自己的分身，就是Thread，中文又翻成執行緒。以往一個Process一次只能做一件事情，因此要一面輸入文字，一面計算字數，這種事情是不可能的。但是有了Thread之後，可以在同一個Process底下，讓輸入文字是一個Thread，計算文字又是另外一個Thread，對CPU來說兩個都是類似一個Process，因此兩個可以同時做。又一個Process底下有數個Thread，而一個Process的Global Variable可以讓它的所有Thread共享，也就是所有Thread都可以存取同一個Process的Global Variable。而每個Thread自己也有自己的專屬Variable。  但是，如果有兩個Thread要存取同一個Global Variable，有可能發生問題，也就是說可能會存取到錯的值(例如兩個Thread同時要對一個Variable做加減，最後那個答案可能會是錯的)，這就是Synchronization問題。又，每一個Thread之間可能會互搶資源，而造成死結(Deadlock)，只要以下四個條件都滿足就有死結。  (1)這個資源不能同時給兩個人用  (2)有一個人拿了一個資源，又想拿別人的資源  (3)如果一個人占了茅坑不拉屎，占用資源很久，仍不能趕他走  (4)A等B，B等C，C等D，D又等A 等成一圈。 要解決這種狀況有Avoid(預防) 或 避免(Prevent)兩種方式，破除以上四種其中一種即可。 |