Linux操作系统的X86汇编程序设计（2） PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/144/2021_2022_Linux_E6_93_8D_E4_BD_c103_144864.htm 纲要: "global main" 必须声明为全局的(global) -- 并且既然我们用 GCC 来链接,进入点必须以 "main" 来命名 -- 从而装入系统. "extern printf" 只是一个声明,为以后在程序中调用. 注意这是必须的. 参数的大小不需要声明. 我已经把这个例子用标准的 .data, .text 分节, 但这不是严格必须的 -- 可能只需要一个 .text段, 就像在 DOS 下一样。 在代码的主体部分, 你必须把参数压栈来传递给调用. 在 Nasm 里, 你必须声明所有不明确数据的大小. 因此就有 "dword" 这个限定词. 注意和其他汇编器一样,Nasm 假设所有的内存/标号的引用都指的是内存地址或者标号, 而不是它的内容。 因而, 指明字符串 msg 的地址, 你应该使用 push dword msg, 指明字符串 msg 的内容, 应该用push dword [msg] (这只能包含 msg 的前四个字节). 因为 printf需要一个指向字符串的指针, 我们应该指明 msg 的地址。 调用 printf 非常的直接. 注意每一次调用后你必须把栈清除(见下). 所以 PUSH 了一个dword 后, 我从栈里把一个 dword POP 进一个无用的寄存器. Linux 程序只简单的用一个 RET 来返回系统, 由于每个进程都是 shell(或者是 PID)的产物, 所以程序结束后把 控制权还给它. 注意到在 Linux 下, 你是在 "API" 或中断服务的场所里使用系统带来的标准共享库.所有的外部引用由 GCC 管理, 它给 asm 程序员节省了大部分的工作. 一旦你习惯了基本的技巧, Linux 下的汇编编程实际上要比 DOS 简单的多。 C 调用的语法 Linux 使用 C 的调用模式 -- 意味着参数以相反的顺序进栈(最后一个最先), 调用者必须清除

栈. 你可以从栈里把值 pop 出来: push dword szText call puts pop ecx或者直接修改 ESP: push dword szText call puts add esp, 4调用的返回值在 eax 或 edx:eax 如果值大于 32 位的话. EBP, ESI, EDI, EBX 由调用者保存和恢复. 你必须保存你要使用的寄存器, 像下面这样: . loop.asm global main extern printf section .text msg db "HoodooVoodoo WeedooVoodoo",0Dh,0Ah,0 main: mov ecx, 0Ah push dword msg looper: call printf loop looper pop eax ret . EOF粗一看, 非常简单: 因为你在 10 个 printf() 调用用的是同一个字符串, 你不需要清除栈. 但当你编译以后, 循环不会停止. 为什么? 因为 printf() 里什么地方用了 ECX 但没有保存. 使你的循环正确的工作, 你必须在调用之前保存 ECX 的值, 调用之后恢复它, 像这样: . loop.asm global main extern printf section .text msg db "HoodooVoodoo WeedooVoodoo",0Dh,0Ah,0 main: mov ecx, 0Ah looper: push ecx .save Count push dword msg call printf pop eax .cleanup stack pop ecx .restore Count loop looper ret . EOF 