解析Linux内核获取当前进程指针的方法（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__E8_A7_A3_E6_9E_90Linu_c103_144931.htm 每个进程都有一个task_struct任务结构，和一片用于系统空间堆栈的存储空间，他们在物理内存空间中也是联系在一起的，当给进程申请task_struct任务结构空间时，系统将连同系统的堆栈空间一起分配，如下图为某个进程切换时刻的内存图： 下面针对代码实现来分析一下系统如何通过一系列操作获取进程在内核中的任务结构指针的： 由于linux内核分配进程任务结构空间时，是以8KB(2个页面空间，即2^1*4KB，linux对物理内存空间和虚拟内存空间管理时，均规定其页面单位的尺寸为4KB)为单位来分配的，所以内存应用地址是8KB(2^13)的整数倍，即指针地址的低13位全为0，所以根据小端字节序，分配内存返回地址应该是指向struct task_struct结构，如图中的0xc2342000地址所指，至于为何采用代码中的做法而不是直接将此指针保存在全局变量中以供应用，内核是从其自身的效率方面来考虑的，我们在此只针对代码解释：根据上图，此刻内存esp内容必定在0xc2342000和0xc2344000之间的一个数值，我们假设取0xc2343ffe(即堆栈压栈EIP、返回地址、内部数据等相关数据了，地址值要减小；只要符合0xc2342xxx 、0xc2343xxx的地址指针都是正确的)，来通过代码运算看是否current的指针是0xc2342000。 __asm__("andl %%esp,%0. ":"=r" (current) : "0" (~8191UL)).语句的意思是将ESP的内容与8191UL的反码按位进行与操作，之后再把结果赋值给current，其中8191UL=8192-1=2^13-1,计算过程如下： 8192UL=2^13 0000

0000 0000 0000 0010 0000 0000 00008191UL 0000 0000 0000 0000 0001 1111 1111 1111~8191UL(反码) 1111 1111 1111 1111 1110 0000 0000 00000xc2343ffe 1100 0010 0011 0100 0011 1111 1111 1110andl结果： 1100 0010 0011 0100 0010 0000 0000 0000 || (对照着看)0x c 2 3 4 2 0 0 0所以按位与操作之后的结果位0xc2342000，正好是struct task_struct结构的地址指针.通过观察可知，只要符合0xc2342xxx 、0xc2343xxx的地址指针经过相同的计算，都可以得到内核进程任务结构的指针。 另外，在进入中断或系统调用时所引用的宏操作(include\asm-i386\ hw_irq.h): #define GET_CURRENT \"movl %esp, \n\t" \"andl $-8192, \n\t"其原理与上述描述也是一致的。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com