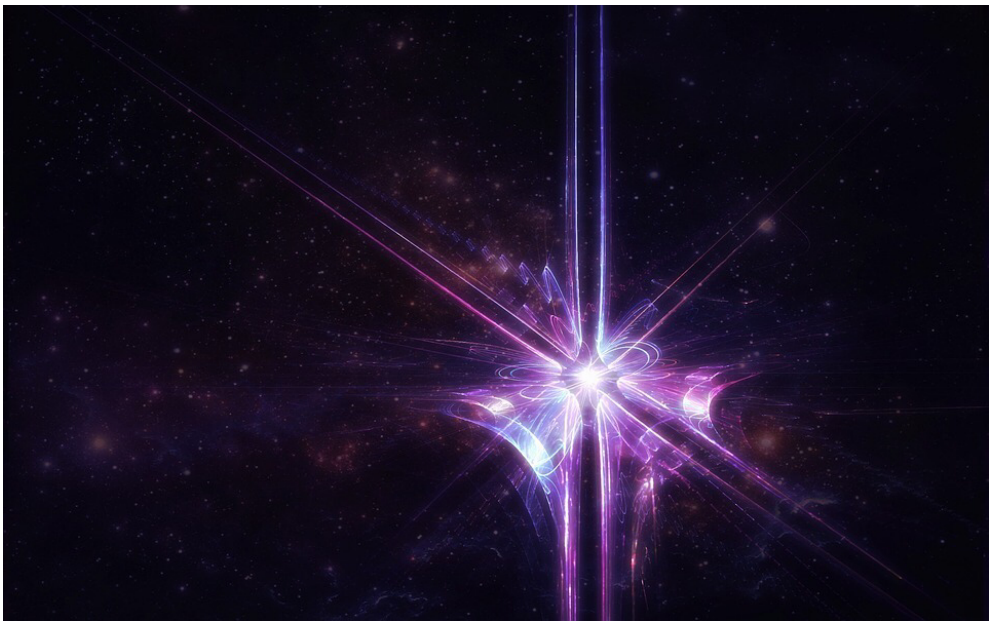


函数调用之堆栈原理（一）

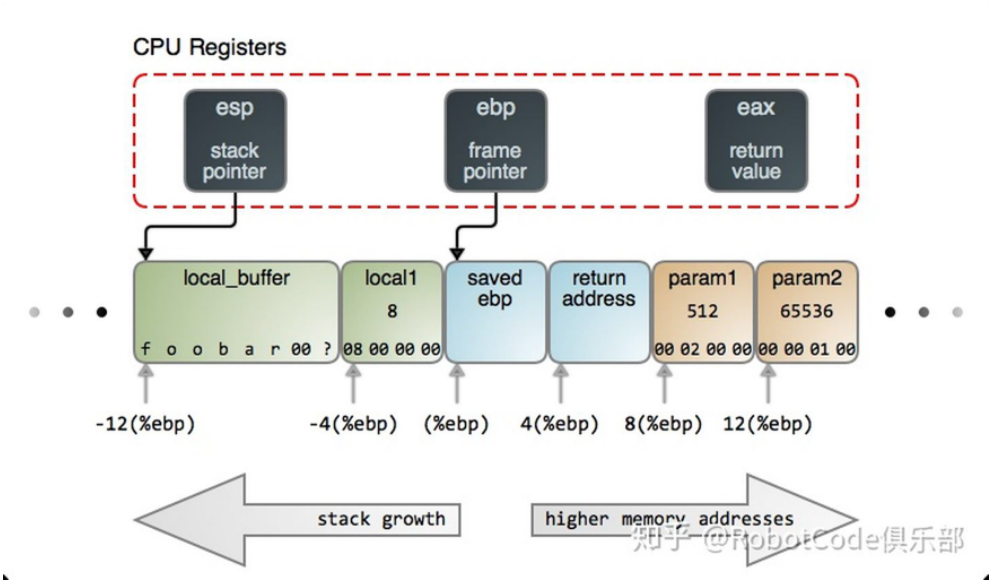
RobotCode俱乐部
知乎 | 公号: RobotCode俱乐部『认真记录，用心生活』

> 原文：manybutfinite.com/post/...
> 翻译：RobotCode俱乐部

在之前的文章中，我们探讨了程序的内存布局以及内核是如何管理内存的。（可参考我前面翻译的文章）现在我们来看看**调用堆栈**，它是大多数编程语言和虚拟机中的主要工作部件。在此过程中，我们将遇到一些奇妙的现象，如闭包、递归和缓冲区溢出。但是第一步是要精确地描述堆栈是如何工作的。

堆栈非常重要，因为它跟踪程序中运行的函数，而函数又是软件的基本组成模块。事实上，程序的内部操作通常非常简单。它主要由函数相互调用时**将数据压入堆栈和从堆栈弹出数据组成**，同时在堆上为 必须跨函数调用时需要存活的数据分配内存。对于调试、性能调优以及了解函数调用的低层原理，对堆栈这一原理的牢固掌握是非常宝贵的。

当调用一个函数时，将创建一个栈桢来支持该函数的执行。栈桢包含函数的局部变量和调用者传递给函数的参数。栈桢还包含管理信息，**允许被调用的函数(被调用方)安全地返回给调用方。**堆栈的确切内容和布局因处理器体系结构和函数调用约定的不同而不同。在本文中，我们将研究使用c风格函数调用(cdecl)的Intel x86堆栈。这里有一个栈帧位于栈的顶部:



现在来看看最重要的三个CPU寄存器：

堆栈指针(esp)指向堆栈的顶部。顶部总是被最后一个被压入到堆栈上但尚未弹出的项目所占据，就像在现实世界中的一堆盘子。存储在esp中的地址随着栈的内容的压入和弹出而不断变化，因此它总是指向最后一项。许多CPU提供了指令自动更新esp，没有这个寄存器使用堆栈是不切实际的。

在Intel架构中，就像在大多数架构中一样，堆栈的内存地址越来越低，即向地址小的地方增长。因此，“top”是包含实时数据的堆栈中最低的内存地址：在本例中是local_buffer。注意，从esp到local_buffer的箭头没有任何含糊之处，它专门指向local_buffer占用的第一个字节，因为这是存储在esp中的确切地址。

跟踪堆栈的第二个寄存器是ebp，即基指针或帧指针。它指向当前运行的函数的栈桢中的一个固定位置，并为访问函数参数和本地变量提供一个稳定的参考点(基)。**ebp仅在函数调用开始或结束时更改。**因此，我们可以很容易地将堆栈中的每个项作为ebp的偏移量来处理，如图所示。

与esp不同，ebp主要由程序代码维护，很少受到CPU干扰。有时候，完全抛弃ebp会带来性能上的好处，这可以通过编译器标志来实现。Linux内核就是这样做的一个例子。

最后，按照约定，**eax寄存器用于将大多数C数据类型的返回值传输回调用方**。

现在让我们检查栈桢中的数据。这些图显示了在调试器中可以看到的每个字节的精确内容，内存从左到右，从上到下不断增长。这里是：

本地变量local_buffer是一个字符数组，其中包含一个以null结尾的ascii字符串。字符串可能是从某处读取的，例如键盘输入或文件，它有7字节长。因为local_buffer可以容纳8个字节，所以还剩下1个空闲字节。这个字节的内容是未知的，因为在堆栈的无限的压入和弹出之中，你永远不知道内存中保存的是什么，除非你对它进行写入。由于C编译器不初始化栈帧的内存，所以内容是不确定的。

继续，local1是一个4字节的整数，您可以看到每个字节的内容。它看起来是个大数，所有的0都跟在8后面，但是这里你的直觉让你误入歧途。

英特尔处理器是小端计算机，这意味着内存中的数字从小端开始。因此，多字节数的最小

有效字节位于最低内存地址中。因为这通常显示在最左边，这与我们通常的数字表示方式不同。大小端的典故是从格列佛游记中借来的：就像小人国的人吃鸡蛋是从小端开始的一样，**英特尔处理器“吃数字”也是从小字节开始的。**

local1实际上是8，就像章鱼腿的数目一样。然而，param1在第二个字节位置的值是2，因此它的数学值是2 * 256 = 512(我们要乘以256，因为每个位置值的范围是从0到255)。同时param2为1 * 256 * 256 = 65536。

这个栈桢中的管理数据由两个关键部分组成：（上图中中间的浅蓝色部分）

1. **前一个栈桢的地址(保存的ebp)**
2. **函数退出时要执行的指令的地址(返回地址)。**

它们一起使函数能够正常返回，使程序能够继续运行。

--未完待续

> 由于本人水平有限，翻译必然有很多不妥的地方，欢迎指正。
> 同时，欢迎关注下方微信公众号，一起交流学习：）

编辑于 2019-02-02

Linux 内核　函数

文章被以下专栏收录

《Linux内核漫游记》
Linux内核探索记录　进入专栏

Linux I/O
包括不限于File System，Block　进入专栏

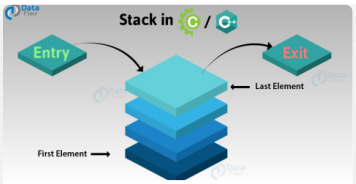
推荐阅读

函数调用之堆栈原理（二）

RobotCode俱乐部

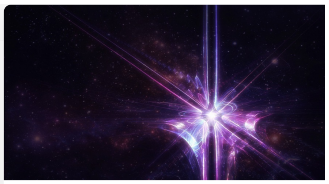
函数调用之堆栈原理（三）

RobotCode俱乐部



c语言进阶：堆栈原理揭秘

Hao J... 发表于山阴路的夏...



函数调用之堆栈原理（终章）-缓冲区溢出

RobotCode俱乐部

还没有评论

写下你的评论...