第5篇:C/C++ 内存布局与程序栈

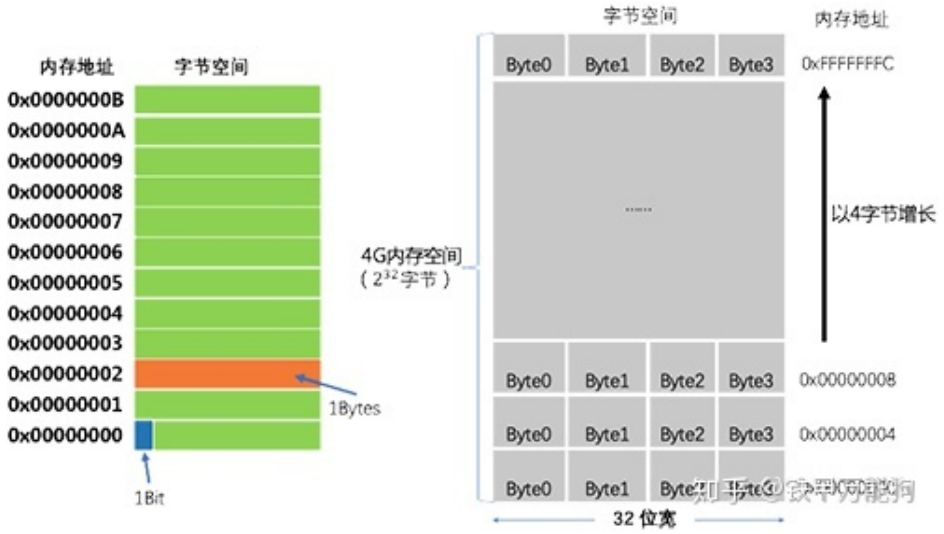

铁甲万能狗
自由开发者，专攻C++/Python后端开发(简书平台同号)
+ 关注

如果你对C/C++基本数据类型的内存模型没概念的话，可以先查看该传送门《开篇1:C/C++ 内存中的数据表示》，反正我觉得

- 先掌握了基本数据的内存模型
- 再理解计算机的寻址模型和程序的内存布局
  这样能够对C/C++内存管理方面的认知起到以小见大的效果。

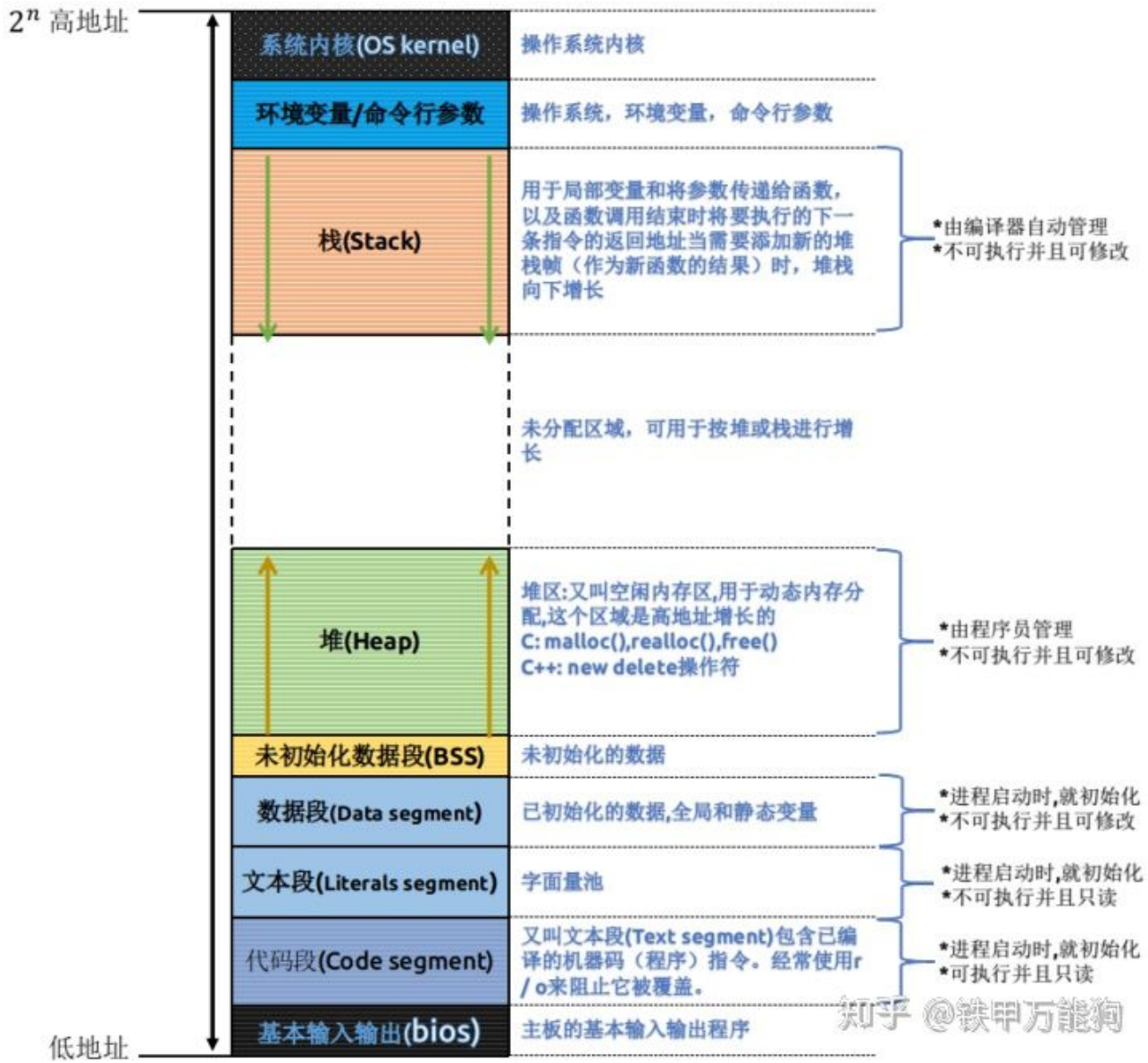
可寻址模型和内存布局

我们知道，内存是由操作系统统一管理的，内存里面一个字节就等于8个二进制位,然后操作系统就为内存空间进行编号,这就是我们所说寻址模型。那么我们经常说的32位指的是什么呢？其实操作系统给内存编号最大只编号到2的32次方(即只能编42,9496,7296个地址编号)，而每个编号逻辑上喜欢使用十六进制来表示,并且用于表示内存的具体位置。形式通俗点说就是4GB的内存大小。为什么32位x86的操作系统无法使用大于4GB的内存条的额外空间？原因就在这里。



C/C++程序（进程）的内存寻址模型

另外,像其他任何程序一样，BIOS和OS都需要内存(废话)，此处为了表示计算机内存模型的完整性。我们都把所有内存相关的内存区域都一一列出了,对于程序员感兴趣的主要内存区域是代码段，数据段和字面量池和bss，堆栈和堆。



从程序的组织的方式来查看程序的内存布局

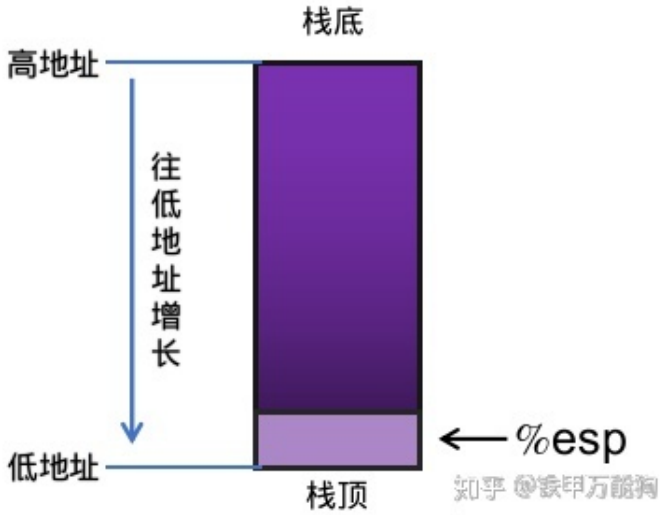
- **代码段**:程序的所有指令会存放在这个区域，这是已经编译后的机器码
- **字面量池**是程序初始化时的一些字符串字面量，在程序中

- **全局数据段**：程序初始化时的常量和全局/静态的变量。C/C++ 用global/static声明的变量都存放在这个区域，对所有函数公开可见。
- **堆**:这里保存的数据只是为了临时存储一些值而创建的，而我们可能在程序运行过程中可能会回收此内存。因为我们在程序执行期间不需要很长时间，所以使用C中的new或malloc这类内存分配程序来为我们所需的特定数据类型提供新的空间，并且随着我们要求越来越多的动态数据空间而该区域不断扩大，并且在内存中逐渐增长到更高的地址。
- **栈**:当我们执行这些过程调用时,堆的基本特性是LIFO,存储着该程序“上下文”，它将从内存的高层地址开始，然后向另一个方向向下扩展。**上下文**其实就是程序中各个函数之间调用的**先后顺序**。

这种典型的内存布局有一个比较有趣的地方,实际上栈向低层地址不断增长，动态数据会向高层地址增长，只要你的程序足够糟糕,例如用无止境的递归和不断抢占堆可用的空间，这两个货始终会碰面，这将是一件非常糟糕的事情。这是一种严重的错误，这种情况操作系统说它内存不足时,例如Windows臭名招嘱的蓝屏提示...!!

IA32平台的程序栈

让我们看一下ia-32体系结构的调用堆栈，我们将堆栈的底部放在内存的顶部，并将堆栈的顶部放在内存的底部。 这只是我们使用的约定，因为我就喜欢使用倒置的形式，也有人喜欢将栈顶定于为上方且栈底定义在下方,但如果没有显式标注高地址和低地址，那就“误人子弟”了。争论这些毫无意义



唯一要记住的是栈是朝着**内存低地址**方向增长，iA32栈中有一个特殊的寄存器，称为esp。该寄存器始终指向堆栈的顶部元素，即放置在堆栈上的最后一个元素。

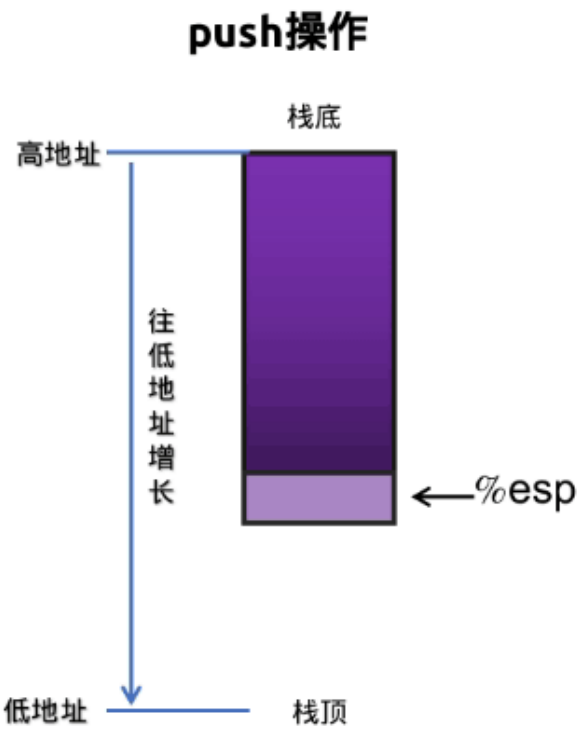
push操作

好的，所以我们要看的第一个堆栈操作是push指令，这里我们展示的是

```
pushl 寄存器名称 或  push 某个类型的指针
```

表示一个32位的值，并为其指定了要入栈的源寄存器或内存位置，基本上它是它会从该源获取值，无论它是寄存器还是内存位置都会推入到栈顶。它还会将栈指针递减4，为什么要减4，因为pushl刚好是4个字节，并且是超低地址方向增长的,因此栈指针递减，

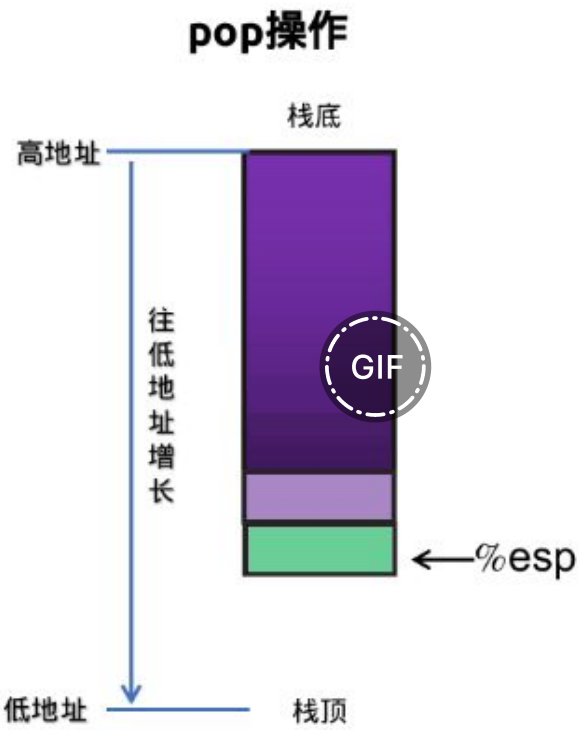
如下图所示,现在栈指针指向内存中已将该值添加或复制到内存中的新位置。



pop操作

```
popl 寄存器名称 或  popl  某个类型的指针
```

popl指令将数据从堆栈中移出。在这种情况下，我们还为它提供了一个dst参数,以获取从栈中弹出的值，然后将该值放入某个内存地址指向的位置或CPU中的寄存器。



我们从堆栈顶删除某个值，并再次为该32位字的esp向上调整堆栈指针。

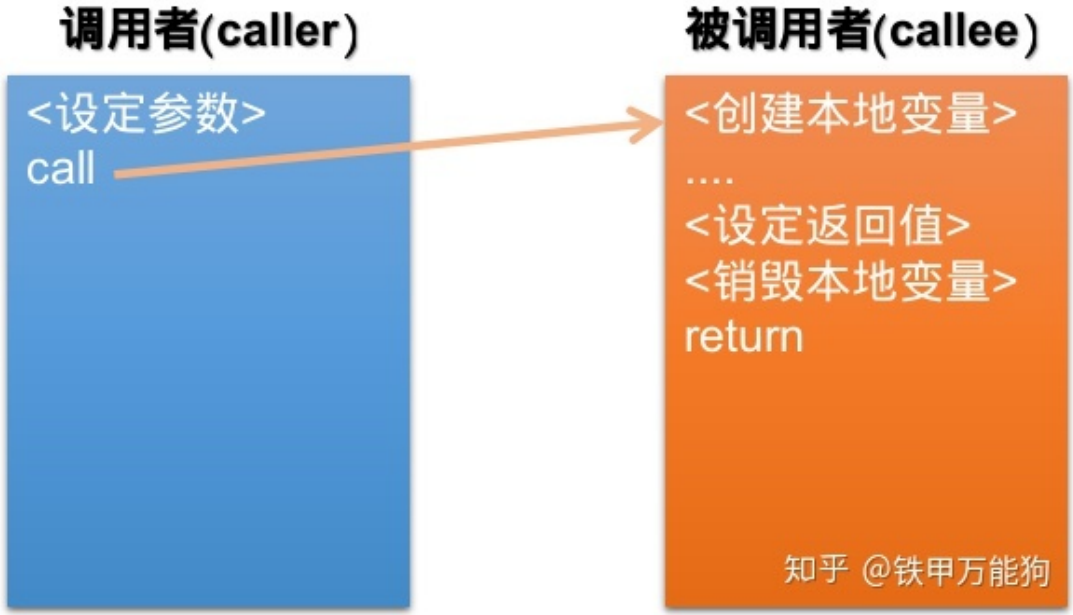
我们pop操作的时候是真的“删除”原先的值吗?
这个值并未删除，它仍然存在于内存中，只是我们不再引用它了。因为我们已经调整了堆栈指针，使其指向栈中的下一个值。 但是原先这些位的数据仍然驻留在原先的内存位置,只是程序不再解释解析这些位中的二进制码。 已经在某种意义上有效地删除了它们，因为我们可以回收该空间并将新数据压入栈并覆盖这些位。因此需要保留被弹出的数据，只需将它们拷贝到指定的位置即可。

让我们看看如何使用堆栈来跟踪过程调用，以及如何记住过程调用结束时需要返回的返回地址以及需要从该过程获取的返回值。

程序的过程调用概述

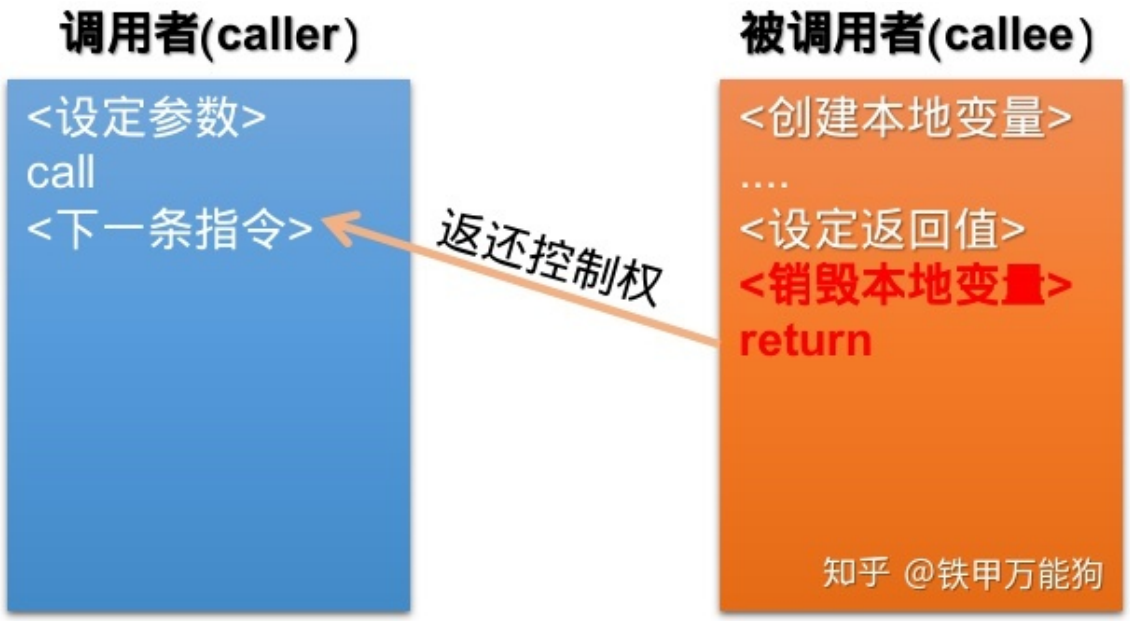
下面是一个过程调用的概述，而且是一个很简陋的例子，有经验的程序员可能已经看出很多漏洞了-_-b!!我说明在先这个例子仅仅起到抛转引玉的作用并且在最后通过该例子提出几个问题，而这些问题会在以后的文章里会详细得到解答，那么我们将从调用者和被调用者这两个程序开始。

1. 调用者将设置一些参数,并在执行call指令后,该指令将控制流跳转到被调用被调用者的函数，之前在被调用者初始化的参数也一同传递给被调用者。



2.此时控制权在被调用者的函数中，被调用者会创建一些局部变量，在执行一些运算的操作，并且运算的结果设为一个返回值，该返回值是被调用函数返回给调用者函数的。

3.在被调用者函数执行return之前,要清理创建的局部变量，并回收空间，最后执行return指令以告诉CPU要把控制权交还给调用者函数。



4.并转到调用者函数原先执行点之后的下一条指令，由于调用者的下一条指令后没有其他指令了就开始清理空间，该空间最初用于设置参数所占用的空间都会被回收。此处，我们应该要清楚原先被调用者函数所占用的空间已被回收，并且调用者函数再执行后也会销毁自己，这就是调用过程设置。

以上的例子很简单，基本稍微有一些代码基础的读者不用看都知道，但我的目的是导出如下几个问题点。

- 被调用者函数必须知道从哪里获取参数?
- 被调用者必须知道从哪里获取**返回地址**?
- 调用者必须知道从哪里获取返回值?

由于调用和被调用方在同一个CPU上运行，因此它们当然使用该CPU中的同一寄存器，因此要有一种机制确保两者之间不会同时争夺CPU的资源。这种机制就是：

- 如果调用者要使用某个寄存器，而刚好被调用者也需要使用该寄存器，调用者在交出该寄存器的控制权之前，它会先保存该寄存器(通常是一个地址)，当这一步完成后，就将寄存器让给被调用者。
- 同理，被调用者也可能会保存当前使用的寄存器的地址后，才让出寄存器的控制权。

这里也引出一个问题：究竟要赋予所有职责给调用者还是被调用者？或所有职责由两者共同承担?这就跟调用机制扯不上关系了，而是考验程序员如何合理设计函数的功能，明确函数之间的分工主次的问题了!!

发布于 2020-08-15 02:13

内存管理　堆栈（内存）　C / C++

文章被以下专栏收录

C/C++内存管理
侧重C/C++内存模型的分析

推荐阅读

C++的内存管理-堆和栈

涉及到C++的内存分区管理方式。 程序通常需要牵涉到三个可能的内存管理器的操作： 让内存管理器分配一个某个大小的内存块让内存管理器释放一个之前分配的内存块让内存管理器进行...

LayH

万字长文|深入 C++ 内存管理

https://blog.csdn.net/zju_fish1
引言 说到 C++ 的内存管理，我们可能会想到栈空间的本地变量、堆上通过 new 动态分配的变量以及全局命名空间的变量等，这些...

程序员编程指南

2 条评论

切换为时间排序

写下你的评论...

满船清梦压星河 2021-06-09

你好,我有一个疑问,请问这里的堆栈,数据段,代码段等是只有C/C++程序执行的时候从内存中获取的自己那一部分内存的划分,还是就算没有C/C++程序运行但是内存中依旧存在着这样,属于操作系统的内存,我学习过Java的JVM,知道Java是在运行时向操作系
自己根据一些设定将内存划分不同的区域,请问这里的划分和Java一样

赞同 11

2 条评论

分享

赞

铁甲万能狗 (作者) 回复 满船清梦压星河 2021-06-09

建议你了解一下操作系统的内存分配原理，C/C++的栈内存直接属于操作系统分配，而栈内存中的指针变量又会指向系统内存管理的堆内存。栈内存和堆内存，以及其它段构成了一个C/C++进城的实例。

赞