ALGORITHM

Call Stack 是什么?

LaiOffer

2017年6月21日

我们在上课讲到递归函数调用的空间复杂度的时候曾多次提到过call stack的概念，然而很多同学表示不太清楚。今天我们就来讲一下call stack是什么。相信有了上一篇文章对virtual memory的介绍之后，同学们理解起Call stack来会相对容易一些。

Call Stack 是什么?

Call stack（通常译作'调用栈'）也是计算机系统中的一个重要概念。在介绍 call stack 之前，我们首先来回顾一下 procedure 是什么。在计算机程序当中，一个Procedure（通常译作'过程'）吃进来一些参数，干一些事情，再吐出去一个返回值（或者什么也不吐）。我们熟悉的Function、method、handler 等等其实都是Procedure。当一个Procedure A 调用另一个Procedure B 的时候，计算机其实需要干好几件事。

一. 是转移控制——计算机要暂停 A 并开始执行 B，并让 B 在执行完之后还能回到 A 继续执行。

二. 是转移数据——A 要能够传递参数给 B，并且 B 也能返回值给 A。

三. 分配和释放内存——在 B 开始执行时为它的局部变量分配内存，并在 B 返回时释放这部分内存。

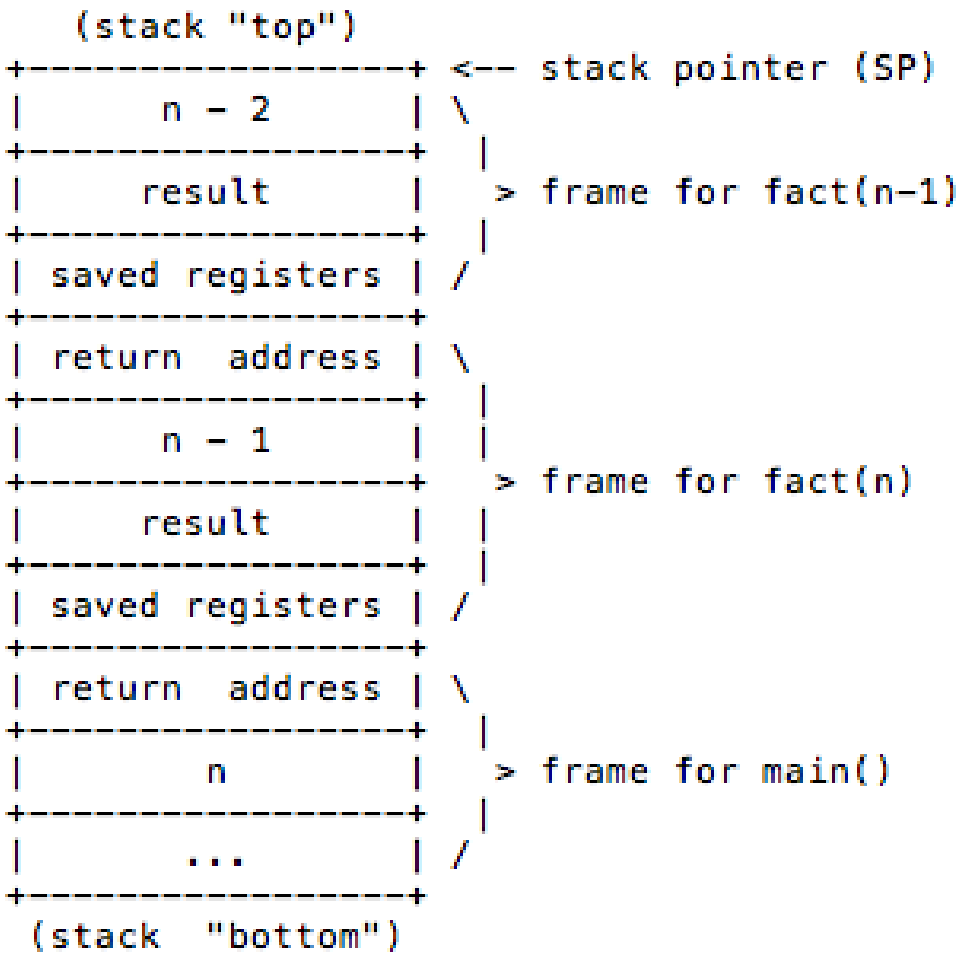
同学们想一下，假设 A 调用 B，B 再调用 C，C 执行完返回给 B，B 再执行完返回给 A，哪种数据结构最适合管理它们所使用的内存？ 没错，是stack，因为过程调用具有 last-in first-out 的特点。当 A 调用 B 的时候，A 只要将它需要传递给 B 的参数 push 进这个 stack，再把将来 B 返回之后 A 应当继续执行的指令的地址（学名叫 return address）也 push 进这个 stack，就万事大吉了。 之后 B 可以继续在这个 stack 上面保存一些寄存器的值，分配局部变量，进而继续构造调用 C 时需要传递的参数等等。 这个 stack 其实就是我们所说的Call Stack。（这里的描述有些简化，实际当中计算机会做一些优化，如果参数和局部变量不太多的话就懒得放在 call stack 里，而是直接使用寄存器了。） Call stack 在 virtual memory 里其实就是一段连续的地址空间，靠一个叫做 SP 的寄存器（32-bit 叫 ESP，64-bit 叫 RSP）来指向栈顶。 既然是连续的，于是它在使用上比我们理论课上讲的抽象的 stack 要更灵活一些，更接近 array 而不是 linked list，可以访问任意元素，而不仅仅是栈顶元素。（当然进栈出栈还是只能在栈顶进行。）这也就是为什么尽管它叫做 call stack，我们依然可以同时有不止一个参数和不止一个局部变量的原因。

Example

举个例子吧。假设我们有这样一段求阶乘的代码：

```
00  int fact(int n) {
01      int result;
02      if (n <= 1)
03          result = 1;
04      else
05          result = n * fact(n - 1);
06      return result;
07  }
```

当 main() 调用了 fact(n)，fact(n) 又调用了 fact(n-1)，fact(n-1) 即将调用 fact(n-2) 的时候，它的 call stack 差不多是这样：（具体情况大同小异，和编译器优化有关。）



其中每个 procedure 分配的内存区域叫做它的 stack frame（通常译作'栈帧'，吕老师译作'梦境'）。这也就解释了为什么当我们分析递归函数调用的空间复杂度时，既需要考虑 recursion tree 的深度，也需要考虑每层所分配的局部变量的大小。 对于上述 fact() 函数，它的 recursion tree 的深度是 n，这就意味着总共有 n 个 stack frame。每个 stack frame 里面除了保存 return address 和一些寄存器的值之外，还需要保存参数 n 和局部变量 result，它们都是 O(1) 的。所以 fact() 总的空间复杂度是 O(n) 的。

希望同学们能够通过了解 call stack 进一步理解空间复杂度的计算，在面试的时候一通百通。

（本文在写作过程中参考了 Randal E. Bryant 和 David R. O'Hallaron 所著的 Computer Systems: A Programmer's Perspective 第二版和第三版。）

汤老师

来offer王牌讲师之一。清华大学计算机系信息学竞赛保送生，美国哥伦比亚大学计算机系软件系统实验室博士生，曾在 OSDI、CACM 等操作系统领域最顶级学术会议和期刊上发表多篇论文。曾参与 Chrome 和 Google Cloud Platform 的研发工作。

图片来自网络，版权属于原作者

核心课程

软件工程师旗舰核心课程

全栈开发项目实践课程

人工智能与数据科学强化课程

无人车系统课程

UX工业实战课程

产品

LaiCode

Learning Mangement System

资源，资讯 & 案例

新闻 & 活动

精品视频

技术博客

学员心得

每周Offer榜

证书验证

消费者信息

Catalog

BPPE

Fact Sheets

BPPE Annual Report

关于我们

团队介绍

加入我们

政策 & 条款