操作系统 编程 堆栈（内存）

内存划分为什么要分为堆和栈，当初设计这两个的时候分别是要解决什么问题?

最近在看《深入理解操作系统》（a programmer's perspective）这本书的某专栏解析文章，看到内存这块的时候老有个疑问，当初是为了解决什么问题设计出了堆和栈这两个概念？作为一个程序员，应该在大脑中如何去理解这两个概念?

尼布甲尼撒
事实派

这个问题提的很好。追溯历史确实是理解概念的最好方法。但看了这个问题下面的回答，基本没有正面回答的，都在掉书袋，估计初学者看了不是更明白了，而是更糊涂了。

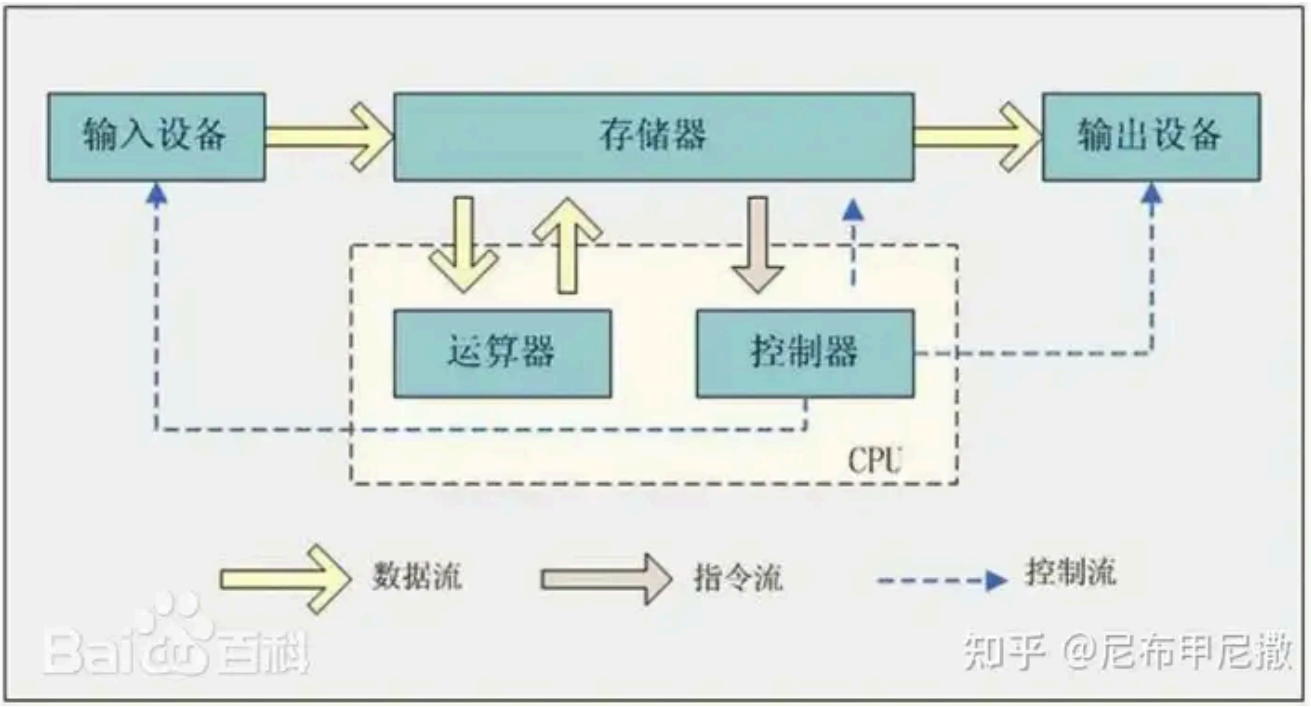
不仅学计算机是这样，学其他学科也是这样。比如，很多人学物理，就是三板斧：背概念、记公式、做题目。但我一直认为，仅仅这样做是无法深刻理解物理概念的。学习一点物理史，了解概念、公式提出的背景，所要解决的问题，对于更深刻的理解物理图像，会有很大帮助。

计算机科学也不例外，人们提出这些概念，不是为了提出概念而提出概念，而是要解决当时面临的问题。

言归正传。

最开始的时候，既没有栈，也没有堆。

冯·诺依曼提出冯·诺依曼架构的时候，只是说要计算机要有“存储器”，并没有进一步划分为“堆”和“栈”。



冯·诺依曼架构

所以，早期（1940s 到 1950s）实现的计算机，既没有栈的概念，有没有堆的概念。整个内存空间就是一张白纸，任你书写，任你驰骋。事实上，今天的一些设备，依然使用这种开发模式，比如，一些单片机的开发。

既然"All our memory are belong to you"了，那按照今天的话说，就是整个地址空间，每个bit，都是全局变量。

我们现在知道这样是不行的。计算机系的一年级新生都知道这样不行。

当时人也不傻，时间长了也发现，这样真不行。这样写出的代码都是“面条代码”，难以理解，不可维护。

终于，到了1958年，一群欧洲和美国的计算机科学家提出了ALGOL 58语言，1960年又发展为ALGOL 60语言。ALGOL语言是里程碑式的，对后世语言影响很大。可以说目前所有主流语言都是这门语言的后代。

ALGOL -60代码示例：

```
procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
    value n, m; array a; integer n, m, i, k; real y;
comment The absolute greatest element of the matrix a, of size n by m,
    is copied to y, and the subscripts of this element to i and k;
begin
    integer p, q;
    y := 0; i := k := 1;
    for p := 1 step 1 until n do
        for q := 1 step 1 until m do
            if abs(a[p, q]) > y then
                begin y := abs(a[p, q]);
                    i := p; k := q
                end
end Absmax
```

ALGOL 革命性的提出了结构化编程的理念，用于解决面条代码的问题。结构化编程的意思就是代码要想不变成面条，一定要结构化，划分成块(if块、循环块、函数块)，每块用begin...end包起来。begin...end中间，局部使用的变量也要本地化，不要再引用全局变量，避免变量耦合，代码变成面条。

那么，问题来了。如何实现变量的局部化，一个块只使用本快的变量呢？自然的想法就是进入每个块的时候，为这个块新开辟一块内存，由这个块独占。这里要注意，块是可以嵌套的(nested)，A函数会调用B函数，B函数 又可能调用C函数... 。这个嵌套结构有个特点：程序执行时，最后执行的块（比如函数），一定会最先执行完。所以，后申请的一块内存，一定会先释放。所以这一定是一种后进先出的数据结构。

读到这里，你就会发现，你跟当时的计算机科学家一样聪明。当时的科学家提出的也是这种Last-In-First-Out (LIFO)数据结构：进入一个块后，遇到一个变量就push到栈里，退出块时，按照LIFO逐个pop出来，块里再有块时，嵌套执行。非常自然的想法，不是吗？

这就是“栈”。是结构化编程的一个自然结果。

1960s~1970s，ALGOL 的结构化编程思想逐渐为大家所接受。当然，这种模式会带来大量的push和pop操作。为了提高程序效率，DEC在1963年推出的PDP-6计算中，率先提供了PUSH, POP, PUSHJ, POPJ四个指令。

DEC PDP-6

后继的几乎所有开发语言，以及CPU指令集，全部都遵循了这个范式，直到今天。

下面说“**堆**”。注意这个“堆”指题主说的堆，不是其他回答里辩经派说的堆。

大家知道最早的高级语言叫FORTRAN（这门语言至今仍然活跃，在最新的TIOBE榜上排名第10，高于Swift、Rust）。早期版本的FORTRAN除了基本类型外，使用DIMENSION定义一块内存，但这个DIMENSION必须在程序里写死维度和长度，按现在话说，内存必须在编译时静态分配好，不能在运行时分配。大家体会一下早期FORTRAN语言：

```
C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
C INPUT - TAPE READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 1 IN JOB CONTROL LISTING
      READ INPUT TAPE 5, 501, IA, IB, IC
  501 FORMAT (3I5)
C IA, IB, AND IC MAY NOT BE NEGATIVE OR ZERO
C FURTHERMORE, THE SUM OF TWO SIDES OF A TRIANGLE
C MUST BE GREATER THAN THE THIRD SIDE, SO WE CHECK FOR THAT, TOO
      IF (IA) 777, 777, 701
  701 IF (IB) 777, 777, 702
  702 IF (IC) 777, 777, 703
  703 IF (IA+IB-IC) 777, 777, 704
  704 IF (IA+IC-IB) 777, 777, 705
  705 IF (IB+IC-IA) 777, 777, 799
  777 STOP 1
C USING HERON'S FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE
  799 S = FLOATF (IA + IB + IC) / 2.0
      AREA = SQRTF( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
     +     (S - FLOATF(IC)))
      WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
```

```
  601 FORMAT (4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,
     +        13H SQUARE UNITS)
      STOP
      END
```

今天大家使用这样的语言编程，一定会疯掉。我在开发时候，就必须确定一个http返回包的大小并预先静态分配好内存？WTF！

当时人当然也不爽。所以1960s初开发的BCPL语言首次引入了“堆”的概念，提供 `GETBLK` (n)和 `FREEBLK` (p)两个函数。这两个函数可以在运行时动态申请一块内存。终于不用在代码里写死数据体大小了！

BCPL语言发展到C语言，就是著名的malloc和free函数。

这就是“堆”，为了能让你推迟到运行时再决定一块数据的大小，而不是在写代码时就必须决定。

注意这一点和“栈”是不同的，栈上数据的大小必须在编译期决定。

这就是“堆”和“栈”。这些概念的提出，都是为了解决一些问题的自然发展结果，没有任何高深之处。当你在函数中定义个一个变量的时候，你就在使用“栈”，当你malloc、new的时候，你就在使用堆。第一个提出这些概念的，当然很牛，但后面的人在了解了背景之后，理解作者的想法也并不难。

当然，在具体实现堆和栈的时候，会遇到很多细节问题，比如：栈的长度能不能无限大？堆里面要频繁申请释放，如何减少内存的窟窿，提高效率？进一步可能想到，每次malloc后一定要free，又麻烦又不安全，能不能由运行时自动free（GC，垃圾回收）？理解基本概念后，再理解这些实现细节也不会太难。

编辑于 2024-06-30 19:05 · IP 属地湖北