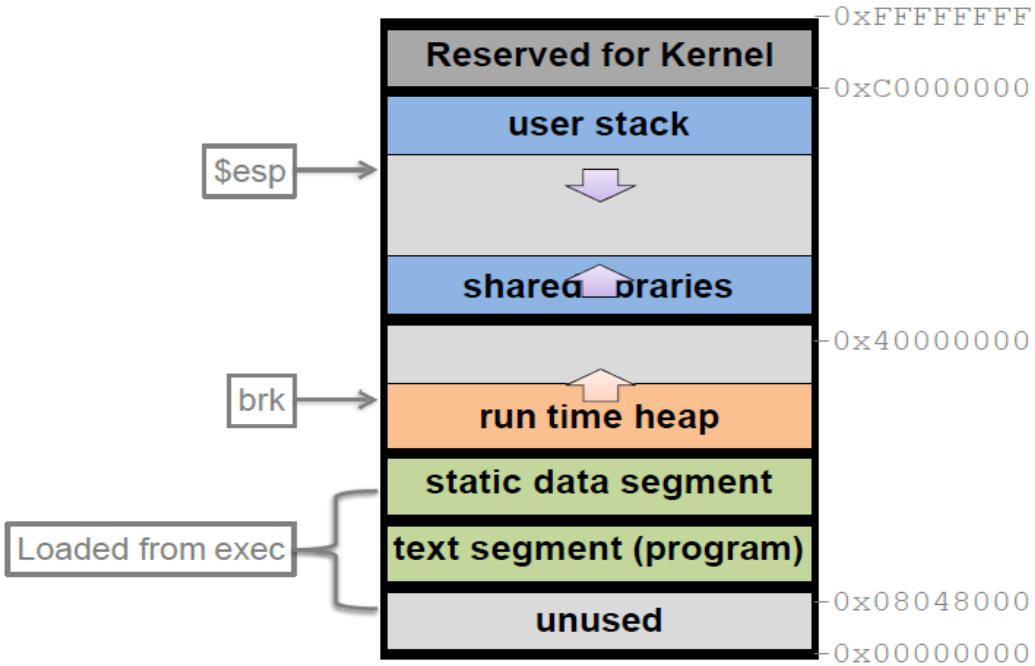
程序运行时的内存分配

一、应用程序是如何被运行起来的?

1. 每个人的电脑中都装载了许许多多个应用程序，在一个应用程序被运行之前，它被保存在计算机的硬盘中。
2. 当我们通过电脑的鼠标双击某个应用程序的图标，电脑的操作系统就会知道我们希望运行这个应用程序，它会为该应用程序分配一定的内存空间，并在硬盘中找到该应用程序的相关文件将它们复制到内存中。
3. 随后电脑的 CPU 会从内存中读取该应用程序的指令以执行，并将执行过程中产生的一些数据写入到应用程序的内存空间中。

二、应用程序运行时的内存层次结构

通过上一节我们了解到每个正在运行的应用程序会有一个自己专属的内存空间，尽管各个应用程序的功能不一，但每个应用程序运行时使用的内存空间结构却是一样的。下图所示是一个 Linux32 位程序的内存层次结构示意图。



Text segment：存储汇编后的程序指令。是只读的，该段不用来存储变量，有固定的的大小。

Static data segment：用来存储全局变量和静态程序变量。该部分的数据是可以改写的。**程序代码中相邻定义的全局变量，它们在该区域的存储地址也相邻。**

以上两个片段中的数据都是从硬盘复制到内存的，以下的堆和栈中的数据则是程序在运行过程中才产生的。

Heap：用来存储程序的其它变量，可变，由内存的低地址向高地址增长。程序中动态分配的内存都存储在堆区。

Stack：用作中间结果暂存来存储断点信息，或者用来存储函数调用间的传递变量。先进后出，大小可变，由内存的高地址向低地址增长。**大部分函数的形参和局部变量都存储在栈区，同一个函数中相邻定义的两个局部变量，它们在该区域的存储地址也相邻。**

三、程序运行过程中栈如何变化？

以下面的这个简单小程序为例

```
1   int func_A(int arg_A1,int arg_A2)
2   {
3       int Var_A;
4       return 0;
5   }
6   void main()
7   {
8       int var_main;
9       func_A(2,1)
10  }
```

栈的变化过程如下（每行开头的数字表示代码所在行数）：

6　程序运行后，执行的第一个函数为 main 函数，因此首先会将 main 函数压栈。

8　在栈中为局部变量 var_main 分配存储空间。

9　发生函数调用，先将传入函数 func_A 的实参 1 与 2 压入栈中，随后将执行完第 9 行代码后应该执行的指令地址压入栈中（这便是返回地址）。

1　开始执行函数 func_A，将 func_A 函数压栈

3　在栈中为局部变量 Var_A 分配内存空间

4　执行完该行代码后，func_A 函数出栈，ESP 指针回退指向之前保存的返回地址

下图为该程序运行过程的堆栈示意图，黄色部分为 main 的栈帧，蓝色部分为 func_A 的栈帧。

在执行过程中，CPU 内的 ESP 寄存器存放的指针指向栈顶，EBP 寄存器内存放的指针指向栈底。在执行完 func_A 函数后，ESP 寄存器由原先蓝色区域的顶部，回退至黄色区域的顶部，这便是出栈。