# C语言中内存分布及程序运行中(BSS段、数据段、代码段、堆栈）

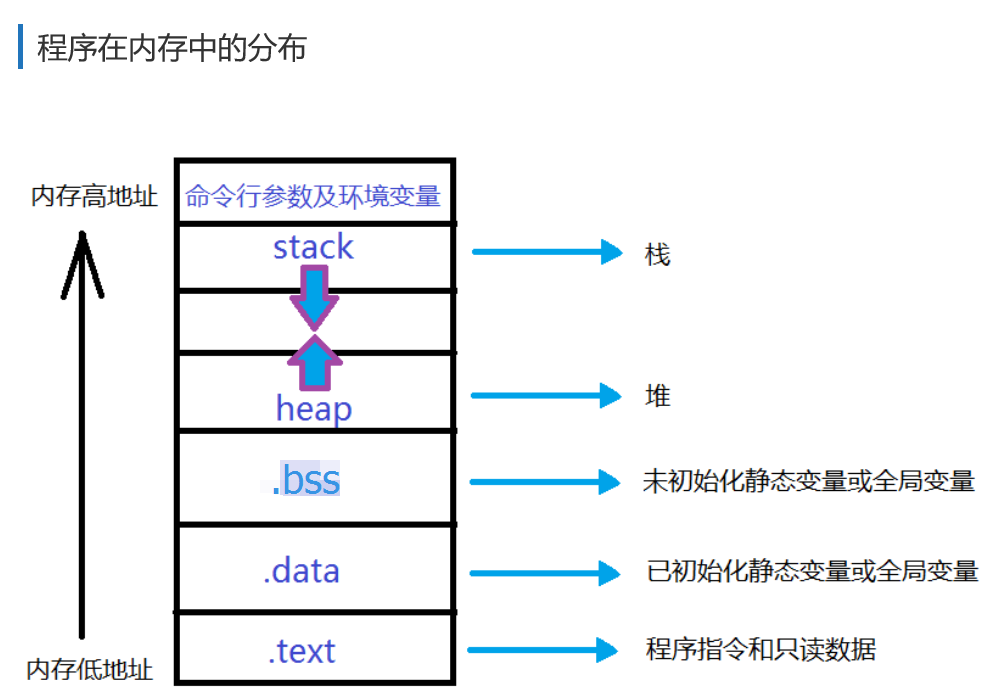
**BSS段:**（bss segment）通常是指用来存放程序中未初始化的全局变量的一块内存区域。BSS是英文Block Started by Symbol的简称。BSS段属于静态内存分配。

**数据段** ：数据段（data segment）通常是指用来存放程序中 已初始化 的 全局变量 的一块内存区域。数据段属于静态内存分配。

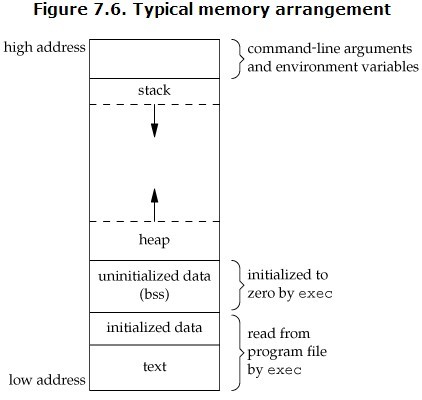
**代码段：** 代码段（code segment/text segment）通常是指用来存放 程序执行代码 的一块内存区域。这部分区域的大小在程序运行前就已经确定，并且内存区域通常属于 只读 , 某些架构也允许代码段为可写，即允许修改程序。在代码段中，也有可能包含一些 只读的常数变量 ，例如字符串常量等。程序段为程序代码在内存中的映射.一个程序可以在内存中有多个副本.

**堆（heap）** ：堆是用于存放进程运行中被动态分配的内存段，它的大小并不固定，可动态扩张或缩减。当进程调用malloc/free等函数分配内存时，新分配的内存就被动态添加到堆上（堆被扩张）/释放的内存从堆中被剔除（堆被缩减）

**栈(stack)** ：栈又称堆栈， 存放程序的 局部变量 （但不包括static声明的变量， static 意味着 在数据段中 存放变量）。除此以外，在函数被调用时，栈用来传递参数和返回值。由于栈的先进先出特点，所以栈特别方便用来保存/恢复调用现场。储动态内存分配,需要程序员手工分配,手工释放。



下图是APUE中的一个典型C内存空间分布图

[](http://static.oschina.net/uploads/img/201306/25114105_Xc9r.jpg)

 例如：

#include

int g1=0, g2=0, g3=0;

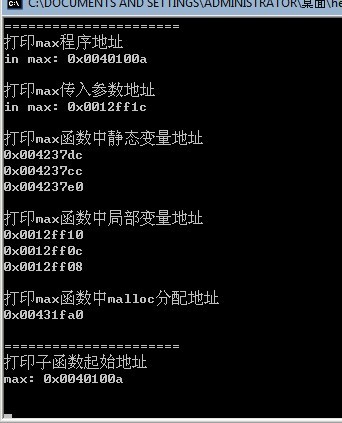
int max(int i)  
{  
    int m1=0,m2,m3=0,\*p\_max;  
    static n1\_max=0,n2\_max,n3\_max=0;  
     p\_max = (int\*)malloc(10);  
    printf("打印max程序地址\n");  
    printf("in max: 0xx\n\n",max);  
    printf("打印max传入参数地址\n");  
    printf("in max: 0xx\n\n",&i);  
    printf("打印max函数中静态变量地址\n");  
    printf("0xx\n",&n1\_max); //打印各本地变量的内存地址  
    printf("0xx\n",&n2\_max);  
    printf("0xx\n\n",&n3\_max);  
    printf("打印max函数中局部变量地址\n");  
    printf("0xx\n",&m1); //打印各本地变量的内存地址  
    printf("0xx\n",&m2);  
    printf("0xx\n\n",&m3);  
    printf("打印max函数中malloc分配地址\n");  
    printf("0xx\n\n",p\_max); //打印各本地变量的内存地址

    if(i) return 1;  
    else return 0;  
}

int main(int argc, char \*\*argv)  
{  
    static int s1=0, s2, s3=0;  
    int v1=0, v2, v3=0;  
    int \*p;     
    p = (int\*)malloc(10);

    printf("打印各全局变量(已初始化)的内存地址\n");  
    printf("0xx\n",&g1); //打印各全局变量的内存地址  
    printf("0xx\n",&g2);  
    printf("0xx\n\n",&g3);  
    printf("======================\n");  
    printf("打印程序初始程序main地址\n");  
    printf("main: 0xx\n\n", main);  
    printf("打印主参地址\n");  
    printf("argv: 0xx\n\n",argv);  
    printf("打印各静态变量的内存地址\n");  
    printf("0xx\n",&s1); //打印各静态变量的内存地址  
    printf("0xx\n",&s2);  
    printf("0xx\n\n",&s3);  
    printf("打印各局部变量的内存地址\n");  
    printf("0xx\n",&v1); //打印各本地变量的内存地址  
    printf("0xx\n",&v2);  
    printf("0xx\n\n",&v3);  
    printf("打印malloc分配的堆地址\n");  
    printf("malloc: 0xx\n\n",p);  
    printf("======================\n");  
    max(v1);  
    printf("======================\n");  
    printf("打印子函数起始地址\n");  
    printf("max: 0xx\n\n",max);  
    return 0;  
}

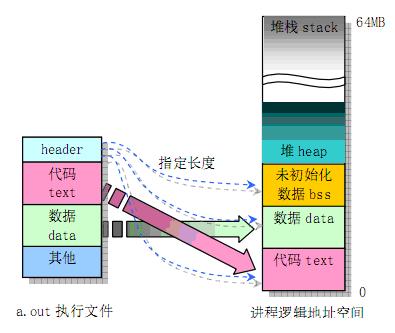
 打印结果：

[](http://static.oschina.net/uploads/space/2013/0625/113936_wfCb_1165866.jpg)

可以大致查看整个程序在内存中的分配情况:  
可以看出,传入的参数,局部变量,都是在栈顶分布,随着子函数的增多而向下增长.  
函数的调用地址(函数运行代码),全局变量,静态变量都是在分配内存的低部存在,而malloc分配的堆则存在于这些内存之上,并向上生长.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

在操作系统中，一个进程就是处于执行期的程序（当然包括系统资源），实际上正在执行的程序代码的活标本。那么进程的逻辑地址空间是如何划分的呢？

图1做了简单的说明(Linux系统下的)  
[](http://static.oschina.net/uploads/img/201306/25114106_BIx4.jpg)

左边的是UNIX/LINUX系统的执行文件，右边是对应进程逻辑地址空间的划分情况。  
首先是堆栈区(stack)，堆栈是由编译器自动分配释放，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。栈的申请是由系统自动分配，如在函数内部申请一个局部变量 int h，同时判别所申请空间是否小于栈的剩余空间，如若小于的话，在堆栈中为其开辟空间，为程序提供内存，否则将报异常提示栈溢出。      
其次是堆(heap)，堆一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表。堆的申请是由程序员自己来操作的，在C中使用malloc函数，而C++中使用new运算符，但是堆的申请过程比较复杂：当系统收到程序的申请时，会遍历记录空闲内存地址的链表，以求寻找第一个空间大于所申请空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序，此处应该注意的是有些情况下，新申请的内存块的首地址记录本次分配的内存块大小，这样在delete尤其是 delete[]时就能正确的释放内存空间。  
接着是全局数据区(静态区) (static)，全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。另外文字常量区，常量字符串就是放在这里，程序结束后有系统释放。  
最后是程序代码区，放着函数体的二进制代码。  
  
举例说明一下:  
int a = 0;                   //全局初始化区  
char \*p1;                //全局未初始化区

int main()  
{  
        int b;                // 栈  
        char s[] = "abc";       //栈  
        char \*p2;             //栈  
        char \*p3 = "123456";    //123456\0在常量区，而p3在栈上

        static int c =0；    //全局（静态）初始化区

        p1 = (char \*)malloc(10);

        p2 = (char \*)malloc(20); //分配得来得10和20字节的区域就在堆区

        strcpy(p1, "123456");    //123456\0放在常量区，编译器可能会将它与p3所指向的"123456"优化成一个地方。

        return 0;  
}

原文：my.oschina.net/pollybl1255/blog/140323‎