**http://www.cnblogs.com/CodeWorkerLiMing/archive/2012/06/13/2547514.html**

[**c程序内存分布[转载]**](http://www.cnblogs.com/CodeWorkerLiMing/archive/2012/06/13/2547514.html)

**原文地址：**[c程序内存分布](http://blog.sina.com.cn/s/blog_452425e70100hu1g.html)**作者：**[binbin](http://blog.sina.com.cn/u/1159996903)

C程序一般分为

1.程序段: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多有个副本.

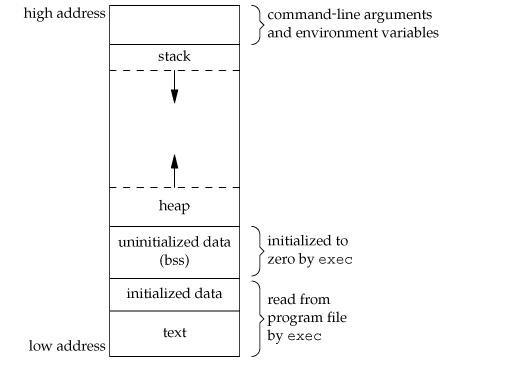
2.初始化过的数据:在程序运行值初已经对变量进行初始化的

3.未初始化过的数据:在程序运行初未对变量进行初始化的数据

4.堆(stack):存储局部,临时变量,在程序块开始时自动分配内存,结束时自动释放内存.存储函数的返回指针.

5.栈(heap):存储动态内存分配,需要程序员手工分配,手工释放.

附程序分布图:



#include <stdio.h>  
#include <malloc.h>  
  
void print(char \*,int);  
main()  
{  
    char \*s1 = "abcde";  
    char \*s2 = "abcde";  
    char s3[] = "abcd";  
    long int \*s4[100];  
    char \*s5 = "abcde";  
    int a = 5;  
    int b =6;//a,b在栈上，&a>&b 地址反向增长  
  
    printf("variables address in main function:n  
            s1=%p s2=%p s3=%p s4=%p s5=%p a=%p b=%pnn",   
            s1,s2,s3,s4,s5,&a,&b);  
    printf("variables address in process call:n");  
    print("ddddddddd",5);//参数入栈从右至左进行,p先进栈,str后进 &p>&str  
    printf("nmain=%p print=%pn",main,print);  
    //打印代码段中主函数和子函数的地址，编译时先编译的地址低，后编译的地址高main<print  
}

void print(char \*str,int p)  
{  
    char \*s1 = "abcde";//abcde在常量区，s1在栈上  
    char \*s2 = "abcde";//abcde在常量区，s2在栈上 s2-s1=6可能等于0，编译器优化了相同的常量，只在内存保存一份  
    //而&s1>&s2  
    char s3[] = "abcdeee";//abcdeee在常量区，s3在栈上，数组保存的内容为abcdeee的一份拷贝  
    long int \*s4[100];  
    char \*s5 = "abcde";  
    int a = 5;  
    int b =6;  
    int c;  
    int d;//a,b,c,d均在栈上，&a>&b>&c>&d 地址反向增长  
    char \*q=str;//  
    int m=p;//  
    char \*r=(char \*)malloc(1);  
    char \*w=(char \*)malloc(1);// r<w 堆正向增长  
  
    printf("s1=%p s2=%p s3=%p s4=%p s5=%p na=%p b=%p c=%p d=%pn   
            str=%p q=%p p=%p m=%p r=%p w=%pn",  
            s1,s2,s3,s4,s5,&a,&b,&c,&d,&str,q,&p,&m,r,w);  
}

|  |
| --- |
| 栈和堆是在程序运行时候动态分配的，局部变量均在栈上分配。栈是反向增长的，地址递减； malloc等分配的内存空间在堆空间。堆是正向增长的，地址递增。 r,w变量在栈上(则&r>&w)，r,w所指内容在堆中(即r<w)。 |

**=======================以下内容为网络转载=======================**

**一般认为在c中分为这几个存储区：  
    1. 栈 －－有编译器自动分配释放   
    2. 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   
    3. 全局区（静态区） －－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。程序结束释放。   
    4. 另外还有一个专门放常量的地方。程序结束释放。**

**在函数体中定义的变量通常是在栈上，用malloc, calloc, realloc等分配内存的函数分配得到的就是在堆上。在所有函数体外定义的是全局量，加了static修饰符后不管在哪里都存放在全局区（静态区）,在所有函数体外定义的static变量表示在该文件中有效，不能extern到别的文件用，在函数体内定义的static表示只在该函数体内有效。另外，函数中的"adgfdf"这样的字符串存放在常量区。   
比如：**

**代码:  
int a = 0; //全局初始化区  
char \*p1; //全局未初始化区  
main()  
{  
    int b; //栈  
    char s[] = "abc"; //栈  
    char \*p2; //栈  
    char \*p3 = "123456"; //123456在常量区，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放在常量区，编译器可能会将它与p3所指向的"123456"优化成一块。  
}   
  
    还有就是函数调用时会在栈上有一系列的保留现场及传递参数的操作。   
    栈的空间大小有限定，vc的缺省是2M。栈不够用的情况一般是程序中分配了大量数组和递归函数层次太深。有一点必须知道，当一个函数调用完返回后它会释放该函数中所有的栈空间。栈是由编译器自动管理的，不用你操心。   
    堆是动态分配内存的，并且你可以分配使用很大的内存。但是用不好会产生内存泄漏。并且频繁地malloc和free会产生内存碎片（有点类似磁盘碎片），因为c分配动态内存时是寻找匹配的内存的。而用栈则不会产生碎片。   
    在栈上存取数据比通过指针在堆上存取数据快些。   
    一般大家说的堆栈和栈是一样的，就是栈(stack)，而说堆时才是堆heap. 栈是先入后出的，一般是由高地址向低地址生长。**

堆(heap)和堆栈(stack)的区别

一、预备知识—程序的内存分配  
一个由c/C++编译的程序占用的内存分为以下几个部分  
1、栈区（stack）— 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。  
2、堆区（heap）— 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，呵呵。  
3、全局区（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。 - 程序结束后有系统释放  
4、文字常量区 —常量字符串就是放在这里的。 程序结束后由系统释放  
5、程序代码区(text)—存放函数体的二进制代码。  
  
二、例子程序  
这是一个前辈写的，非常详细  
//main.cpp  
int a = 0; //全局初始化区  
char \*p1; //全局未初始化区  
main()  
{  
    int b; //栈  
    char s[] = "abc"; //栈  
    char \*p2; //栈  
    char \*p3 = "123456"; //123456在常量区，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放在常量区，编译器可能会将它与p3所指向的"123456"优化成一块。  
}

二、堆和栈的理论知识

2.1申请方式  
stack:  
由系统自动分配。 例如，声明在函数中一个局部变量 int b; 系统自动在栈中为b开辟空间  
heap:  
需要程序员自己申请，并指明大小，在c中malloc函数  
如p1 = (char \*)malloc(10);  
在C++中用new运算符  
如p2 = (char \*)malloc(10);  
但是注意p1、p2本身是在栈中的。

2.2申请后系统的响应  
栈：只要栈的剩余空间大于所申请空间，系统将为程序提供内存，否则将报异常提示栈溢出。  
堆：首先应该知道操作系统有一个记录空闲内存地址的链表，当系统收到程序的申请时，  
会遍历该链表，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，另外，对于大多数系统，会在这块内存空间中的首地址处记录本次分配的大小，这样，代码中的delete语句才能正确的释放本内存空间。另外，由于找到的堆结点的大小不一定正好等于申请的大小，系统会自动的将多余的那部分重新放入空闲链表中。

2.3申请大小的限制  
栈：在Windows下,栈是向低地址扩展的数据结构，是一块连续的内存的区域。这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在 WINDOWS下，栈的大小是2M（也有的说是1M，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示overflow。因此，能从栈获得的空间较小。  
堆：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

2.4申请效率的比较：  
栈由系统自动分配，速度较快。但程序员是无法控制的。  
堆是由new分配的内存，一般速度比较慢，而且容易产生内存碎片,不过用起来最方便.  
另外，在WINDOWS下，最好的方式是用VirtualAlloc分配内存，他不是在堆，也不是在栈是直接在进程的地址空间中保留一快内存，虽然用起来最不方便。但是速度快，也最灵活。

2.5堆和栈中的存储内容  
栈： 在函数调用时，第一个进栈的是主函数中后的下一条指令（函数调用语句的下一条可执行语句）的地址，然后是函数的各个参数，在大多数的C编译器中，参数是由右往左入栈的，然后是函数中的局部变量。注意静态变量是不入栈的。当本次函数调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开始存的地址，也就是主函数中的下一条指令，程序由该点继续运行。  
堆：一般是在堆的头部用一个字节存放堆的大小。堆中的具体内容有程序员安排。

2.6存取效率的比较  
  
char s1[] = "aaaaaaaaaaaaaaa";  
char \*s2 = "bbbbbbbbbbbbbbbbb";  
aaaaaaaaaaa是在运行时刻赋值的；  
而bbbbbbbbbbb是在编译时就确定的；  
但是，在以后的存取中，在栈上的数组比指针所指向的字符串(例如堆)快。  
比如：  
#i nclude <...>  
void main()  
{  
char a = 1;  
char c[] = "1234567890";  
char \*p ="1234567890";  
a = c[1];  
a = p[1];  
return;  
}  
对应的汇编代码  
10: a = c[1];  
00401067 8A 4D F1 mov cl,byte ptr [ebp-0Fh]  
0040106A 88 4D FC mov byte ptr [ebp-4],cl  
11: a = p[1];  
0040106D 8B 55 EC mov edx,dword ptr [ebp-14h]  
00401070 8A 42 01 mov al,byte ptr [edx+1]  
00401073 88 45 FC mov byte ptr [ebp-4],al  
第一种在读取时直接就把字符串中的元素读到寄存器cl中，而第二种则要先把指针值读到edx中，在根据edx读取字符，显然慢了。   
  
2.7小结：  
堆和栈的区别可以用如下的比喻来看出：  
使用栈就象我们去饭馆里吃饭，只管点菜（发出申请）、付钱、和吃（使用），吃饱了就走，不必理会切菜、洗菜等准备工作和洗碗、刷锅等扫尾工作，他的好处是快捷，但是自由度小。  
使用堆就象是自己动手做喜欢吃的菜肴，比较麻烦，但是比较符合自己的口味，而且自由度大。  
  
堆和栈的区别主要分：  
操作系统方面的堆和栈，如上面说的那些，不多说了。  
还有就是数据结构方面的堆和栈，这些都是不同的概念。这里的堆实际上指的就是（满足堆性质的）优先队列的一种数据结构，第1个元素有最高的优先权；栈实际上就是满足先进后出的性质的数学或数据结构。  
虽然堆栈，堆栈的说法是连起来叫，但是他们还是有很大区别的，连着叫只是由于历史的原因。