**[new与malloc的区别，以及内存分配浅析](https://www.cnblogs.com/huhuuu/p/3432371.html)**

Posted on 2013-11-19 22:03 [huhuuu](https://www.cnblogs.com/huhuuu/) 阅读(15943) 评论(1) [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=3432371) [收藏](https://www.cnblogs.com/huhuuu/p/3432371.html)

从函数声明上可以看出。malloc 和 new 至少有两个不同: new 返回指定类型的[指针](http://baike.baidu.com/view/159417.htm)，并且可以自动计算所需要大小。比如：

|  |  |
| --- | --- |
| 1  2  3 | int \*p;  p = new int;  //返回类型为int\* 类型(整数型指针)，分配大小为 sizeof(int); |

或：

|  |  |
| --- | --- |
| 1  2  3 | int\* parr;  parr = new int [100];  //返回类型为 int\* 类型(整数型指针)，分配大小为 sizeof(int) \* 100; |

而 malloc 则必须要由我们计算字节数，并且在返回后强行转换为实际类型的[指针](http://baike.baidu.com/view/159417.htm)。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | int\* p;  p = (int \*) malloc (sizeof(int)\*128);  //分配128个（可根据实际需要替换该数值）整型存储单元，  //并将这128个连续的整型存储单元的首地址存储到指针变量p中  double \*pd=(double \*) malloc (sizeof(double)\*12);  //分配12个double型存储单元，  //并将首地址存储到指针变量pd中 |

第一、malloc 函数返回的是 void \* 类型。对于C++，如果你写成：p = malloc (sizeof(int)); 则程序无法通过编译，报错：“不能将 void\* 赋值给 int \* 类型[变量](http://baike.baidu.com/view/296689.htm)”。所以必须通过 (int \*) 来将[强制转换](http://baike.baidu.com/view/965170.htm)。而对于C，没有这个要求，但为了使C程序更方便的移植到C++中来，建议养成[强制转换](http://baike.baidu.com/view/965170.htm)的习惯。

第二、函数的[实参](http://baike.baidu.com/view/816501.htm)为 sizeof(int) ，用于指明一个[整型数据](http://baike.baidu.com/view/2317434.htm)需要的大小。如果你写成：

|  |  |
| --- | --- |
| 1 | int\* p = (int \*) malloc (1); |

代码也能通过编译，但事实上只分配了1个字节大小的内存空间，当你往里头存入一个整数，就会有3个字节无家可归，而直接“住进邻居家”！造成的结果是后面的内存中原有数据内容被改写。

**下面一段话的原理讲的比较清晰**

malloc函数的实质体现在，它有一个将可用的内存块连接为一个长长的列表的所谓空闲[链表](http://baike.baidu.com/view/549479.htm)。调用malloc函数时，它沿[连接表](http://baike.baidu.com/view/694082.htm)寻找一个大到足以满足用户请求所需要的内存块。然后，将该内存块一分为二（一块的大小与用户请求的大小相等，另一块的大小就是剩下的字节）。接下来，将分配给用户的那块内存传给用户，并将剩下的那块（如果有的话）返回到连接表上。调用[free](http://baike.baidu.com/view/512783.htm)函数时，它将用户释放的内存块连接到空闲链上。到最后，空闲链会被切成很多的小内存片段，如果这时用户申请一个大的内存片段，那么空闲链上可能没有可以满足用户要求的片段了。于是，malloc函数请求延时，并开始在空闲链上翻箱倒柜地检查各内存片段，对它们进行整理，将相邻的小空闲块合并成较大的内存块。如果无法获得符合要求的内存块，malloc函数会返回NULL[指针](http://baike.baidu.com/view/159417.htm)，因此在调用malloc动态申请内存块时，一定要进行返回值的判断。

二、malloc()到底从哪里得来了内存空间：

1、malloc()到底从哪里得到了内存空间？答案是从堆里面获得空间。也就是说函数返回的指针是指向堆里面的一块内存。操作系统中有一个记录空闲内存地址的链表。当操作系统收到程序的申请时，就会遍历该链表，然后就寻找第一个空间大于所申请空间的堆结点，然后就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。就是这样！

   说到这里，不得不另外插入一个小话题，相信大家也知道是什么话题了。什么是堆？说到堆，又忍不住说到了栈！什么是栈？下面就另外开个小部分专门而又简单地说一下这个题外话：

2、什么是堆：堆是大家共有的空间，分全局堆和局部堆。全局堆就是所有没有分配的空间，局部堆就是用户分配的空间。堆在操作系统对进程 初始化的时候分配，运行过程中也可以向系统要额外的堆，但是记得用完了要还给操作系统，要不然就是内存泄漏。

   什么是栈：栈是线程独有的，保存其运行状态和局部自动变量的。栈在线程开始的时候初始化，每个线程的栈互相独立。每个函数都有自己的栈，栈被用来在函数之间传递参数。操作系统在切换线程的时候会自动的切换栈，就是切换SS/ESP寄存器。栈空间不需要在高级语言里面显式的分配和释放。

   以上的概念描述是标准的描述，不过有个别语句被我删除，不知道因为这样而变得不标准了^\_^.

   通过上面对概念的描述，可以知道：

   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。操作方式类似于数据结构中的栈。

   堆一般由程序员分配释放，若不释放，程序结束时可能由OS回收。注意这里说是可能，并非一定。所以我想再强调一次，记得要释放！

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。（这点我上面稍微提过）

 所以，举个例子，如果你在函数上面定义了一个指针变量，然后在这个函数里申请了一块内存让指针指向它。实际上，这个指针的地址是在栈上，但是它所指向的内容却是在堆上面的！这一点要注意！所以，再想想，在一个函数里申请了空间后，比如说下面这个函数：

程序代码：  
   // code...   
       void Function(void)   
       {   
        char \*p = (char \*)malloc(100 \* sizeof(char));   
    }

    
   就这个例子，千万不要认为函数返回，函数所在的栈被销毁指针也跟着销毁，申请的内存也就一样跟着销毁了！这绝对是错误的！因为申请的内存在堆上，而函数所在的栈被销毁跟堆完全没有啥关系。所以，还是那句话：记得释放！

3、free()到底释放了什么

   这个问题比较简单，其实我是想和第二大部分的题目相呼应而已！哈哈！free()释放的是指针指向的内存！注意！释放的是内存，不是指针！这点非常非常重要！指针是一个变量，只有程序结束时才被销毁。释放了内存空间后，原来指向这块空间的指针还是存在！只不过现在指针指向的内容的垃圾，是未定义的，所以说是垃圾。因此，前面我已经说过了，释放内存后把指针指向NULL，防止指针在后面不小心又被解引用了。非常重要啊这一点！

   好了！这个“题外话”终于说完了。就这么简单说一次，知道个大概就可以了！下面就进入第三个部分：

三、malloc()以及free()的机制：

   这个部分我今天才有了新的认识！而且是转折性的认识！所以，这部分可能会有更多一些认识上的错误！不对的地方请大家帮忙指出！

   事实上，仔细看一下free()的函数原型，也许也会发现似乎很神奇，free()函数非常简单，只有一个参数，只要把指向申请空间的指针传递

给free()中的参数就可以完成释放工作！这里要追踪到malloc()的申请问题了。申请的时候实际上占用的内存要比申请的大。因为超出的空间是用来记录对这块内存的管理信息。先看一下在《UNIX环境高级编程》中第七章的一段话：

   大多数实现所分配的存储空间比所要求的要稍大一些，额外的空间用来记录管理信息——分配块的长度，指向下一个分配块的指针等等。这就意味着如果写过一个已分配区的尾端，则会改写后一块的管理信息。这种类型的错误是灾难性的，但是因为这种错误不会很快就暴露出来，所以也就很难发现。将指向分配块的指针向后移动也可能会改写本块的管理信息。

   以上这段话已经给了我们一些信息了。malloc()申请的空间实际我觉得就是分了两个不同性质的空间。一个就是用来记录管理信息的空间，另外一个就是可用空间了。而用来记录管理信息的实际上是一个结构体。在C语言中，用结构体来记录同一个对象的不同信息是

天经地义的事！下面看看这个结构体的原型：

程序代码：  
   struct mem\_control\_block {   
    int is\_available;    //这是一个标记？   
    int size;            //这是实际空间的大小   
    };

    
   对于size,这个是实际空间大小。这里其实我有个疑问，is\_available是否是一个标记？因为我看了free()的源代码之后对这个变量感觉有点纳闷（源代码在下面分析）。这里还请大家指出！

   所以，free()就是根据这个结构体的信息来释放malloc()申请的空间！而结构体的两个成员的大小我想应该是操作系统的事了。但是这里有一个问题，malloc()申请空间后返回一个指针应该是指向第二种空间，也就是可用空间！不然，如果指向管理信息空间的话，写入的内容和结构体的类型有可能不一致，或者会把管理信息屏蔽掉，那就没法释放内存空间了，所以会发生错误！（感觉自己这里说的是废话）

   好了！下面看看free()的源代码，我自己分析了一下，觉得比起malloc()的源代码倒是容易简单很多。只是有个疑问，下面指出！

程序代码：  
   // code...   
      
       void free(void \*ptr)    
    {   
            struct mem\_control\_block \*free;   
            free = ptr - sizeof(struct mem\_control\_block);   
            free->is\_available = 1;   
            return;   
    }

   看一下函数第二句，这句非常重要和关键。其实这句就是把指向可用空间的指针倒回去，让它指向管理信息的那块空间，因为这里是在值上减去了一个结构体的大小！后面那一句free->is\_available = 1;我有点纳闷！我的想法是：这里is\_available应该只是一个标记而已！因为从这个变量的名称上来看，is\_available 翻译过来就是“是可以用”。不要说我土！我觉得变量名字可以反映一个变量的作用，特别是严谨的代码。这是源代码，所以我觉得绝对是严谨的！！这个变量的值是1，表明是可以用的空间！只是这里我想了想，如果把它改为0或者是其他值不知道会发生什么事？！但是有一点我可以肯定，就是释放绝对不会那么顺利进行！因为这是一个标记！

   当然，这里可能还是有人会有疑问，为什么这样就可以释放呢？？我刚才也有这个疑问。后来我想到，释放是操作系统的事，那么就free()这个源代码来看，什么也没有释放，对吧？但是它确实是确定了管理信息的那块内存的内容。所以，free()只是记录了一些信息，然后告诉操作系统那块内存可以去释放，具体怎么告诉操作系统的我不清楚，但我觉得这个已经超出了我这篇文章的讨论范围了。

   那么，我之前有个错误的认识，就是认为指向那块内存的指针不管移到那块内存中的哪个位置都可以释放那块内存！但是，这是大错特错！释放是不可以释放一部分的！首先这点应该要明白。而且，从free()的源代码看，ptr只能指向可用空间的首地址，不然，减去结构体大小之后一定不是指向管理信息空间的首地址。所以，要确保指针指向可用空间的首地址！不信吗？自己可以写一个程序然后移动指向可用空间的指针，看程序会有会崩！

   最后可能想到malloc()的源代码看看malloc()到底是怎么分配空间的，这里面涉及到很多其他方面的知识！有兴趣的朋友可以自己去下载源  
代码去看看。

//部分转载自<http://www.bccn.net/Article/kfyy/cyy/jszl/200608/4238_2.html>

分类: [基础知识](https://www.cnblogs.com/huhuuu/category/522202.html)