**malloc，存储桶原理**

2015年11月06日 09:51:35 [YLM\_007](https://me.csdn.net/u011433762) 阅读数：990

 版权声明：本文为博主原创文章，未经博主允许不得转载~~~~ https://blog.csdn.net/u011433762/article/details/49678491

1--内存管理简述

2--malloc在内存管理中的地位

3--malloc的存储桶实现原理

1、linux0.99采用分段+分页两种内存分配管理方式，但是对于进程来讲这些都是透明不可见的。线性地址空间被分成64M为一个单元的很多段，每个进程占用一个段，在段内的进程并不知道分段机制的存在，都会认为自己占用从0开始到64M的所有地址空间。分段机制是为了进程间的隔离。线性地址空间仍然是“逻辑上的”，并非物理上的。

分页机制把物理地址空间分成4k为一个单位的很多页，用来进行线性地址空间到真实物理地址空间的映射过程。对于进程来讲更是不可见的。

分段和分页机制都是内核透明下完成的，这句话很重要！

2、在C语言中malloc函数用来完成内存的申请工作。需要明确两个点：1、malloc函数并非内核函数；2、malloc向谁申请？

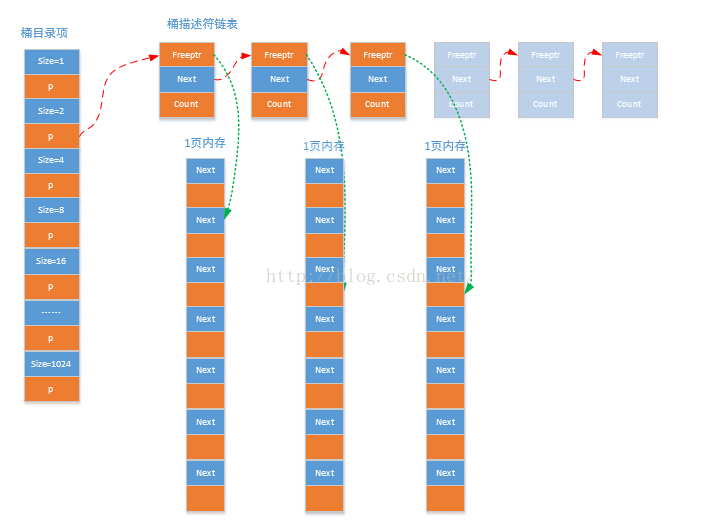
malloc并非内核函数，而是gcc提供的库函数，因此不同版本的gcc或者不同编译器厂商提供的其他编译器中的malloc函数的实现原理可能不同。程序眼中malloc函数和max()函数是没有区别的，是gcc免费赠送的一个模块。

**malloc是负责内存地址空间分配的，这个地址空间就是段内偏移地址空间0-64M范围，不是线性地址空间，也不是真实的物理地址空间。线性地址空间的分配有分段机制完成，真实物理地址空间的分配由分页机制完成。后两者空间的分配是由内核在初始化时就完成分配的。当进程创建完毕后，内核为进程分配了64M长度的段，其他事情进程不必关心。但是这64M长度的空间使用需要程序自己负责分配，这就是malloc原理。**

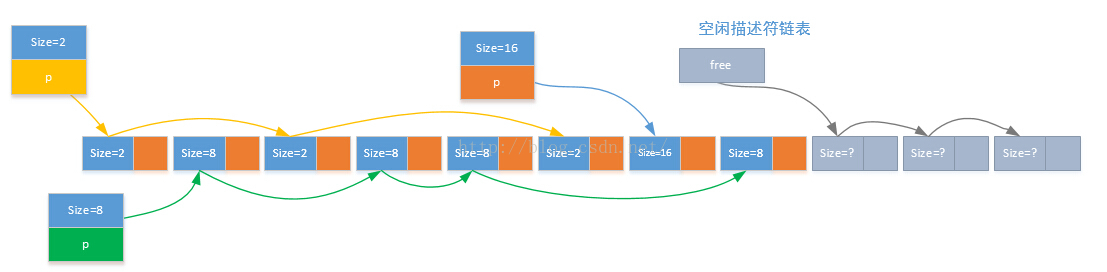
3、malloc对内存的管理是基于“存储桶原理”

malloc函数在第一次被调用的时候会创建一个存储桶数组，每一项都是一个桶，数组项结构如下：

1. *// 存储桶描述符目录结构。*
2. struct \_bucket\_dir { */\* 8 bytes \*/*
3. int size; *// 该存储桶的大小(字节数)。*
4. struct bucket\_desc \*chain; *// 该存储桶目录项的桶描述符链表指针。*
5. };

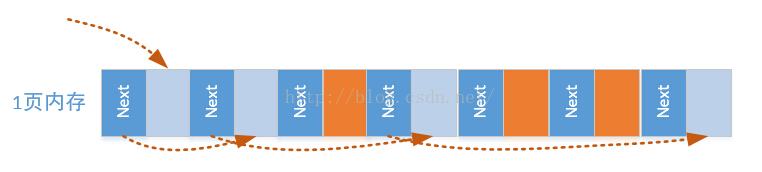


下面介绍桶描述符链表，这是一个单向链表，结构如图所示，链表中的每个节点都附加这1页4k的内存空间，这个1页空间和分页机制中的一页空间是不同的！这一页空间分布在程序的段内偏移地址上，也就是64M空间中的一页。至于这1页内存被具体映射到物理内存中的什么地方，malloc也不知道。当malloc函数中的某个size的桶第一次被需要时，创建该描述符链表，每添加一个节点，就附加一个页，并初始化页。



1. struct bucket\_desc { */\* 16 bytes \*/*
2. void \*page; *// 该桶描述符对应的内存页面指针。*
3. struct bucket\_desc \*next; *// 下一个描述符指针。*
4. void \*freeptr; *// 指向本桶中空闲内存位置的指针。*
5. unsigned short refcnt; *// 引用计数。*
6. unsigned short bucket\_size; *// 本描述符对应存储桶的大小。*
7. };

下面介绍这1页内存，这一页内存是连续的大小为4k的一块区域，当malloc第一次被调用时页初始化，页会被根据当前桶项的大小初始化为特殊的链表结构。比如对于size=8的桶项



内存申请过程：假如申请5个字节的内存  
1、桶的大小都是2的n次方，所以会使用size=8的桶项，进入到同描述符链表；  
2、检查指向链表的freeptr是否为NULL，如果不是NULL则返回freeptr指针，即为所找，如果为null，查找链表的下一个节点；  
3、查找到指针并返回后，freeptr指向下一个可用的节点，并且计数器++；  
注\*当查找完所有节点后都是NULL，则会在链表的头部添加新的一页。  
\*\*\*当第一次使用这个桶项的时候自动添加新的一页，并且对新页进程初始化。  
---------------------------------  
内存回收过程；  
1、回收块在桶中的具体位置定位；  
2、freeptr指向指定位置；  
3、指定位置指向freeptr之前指向的位置。