https://blog.csdn.net/liangzhiyang/article/details/124273428

网址最后有调优举例

**如何查看进程发生缺页中断的次数**？

用ps -o majflt,minflt -C program命令查看。

**majflt代表major fault，中文名叫大错误，minflt代表minor fault，中文名叫小错误**。

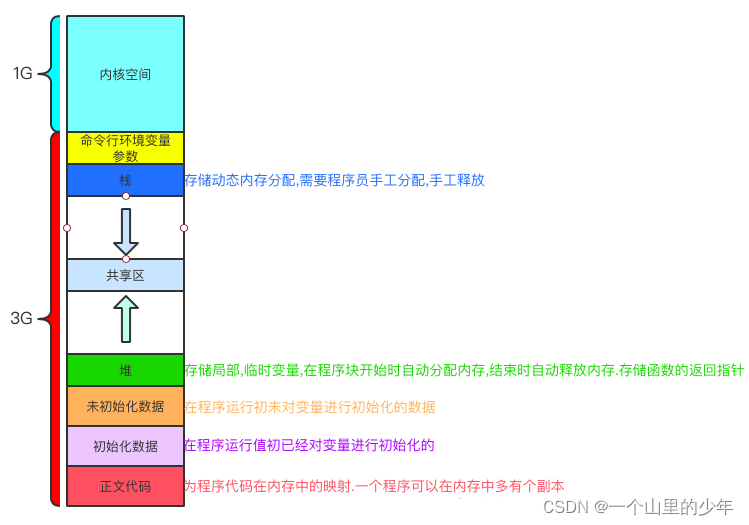
**这两个数值表示一个进程自启动以来所发生的缺页中断的次数。**

**发成缺页中断后，执行了那些操作？**

**当一个进程发生缺页中断的时候，进程会陷入内核态，执行以下操作**：   
**1、检查要访问的虚拟地址是否合法   
2、查找/分配一个物理页   
3、填充物理页内容（读取磁盘，或者直接置0，或者啥也不干）   
4、建立映射关系（虚拟地址到物理地址）**  
重新执行发生[缺页中断](https://so.csdn.net/so/search?q=%E7%BC%BA%E9%A1%B5%E4%B8%AD%E6%96%AD&spm=1001.2101.3001.7020" \o "缺页中断)的那条指令   
**如果第3步，需要读取磁盘，那么这次缺页中断就是majflt，否则就是minflt。**

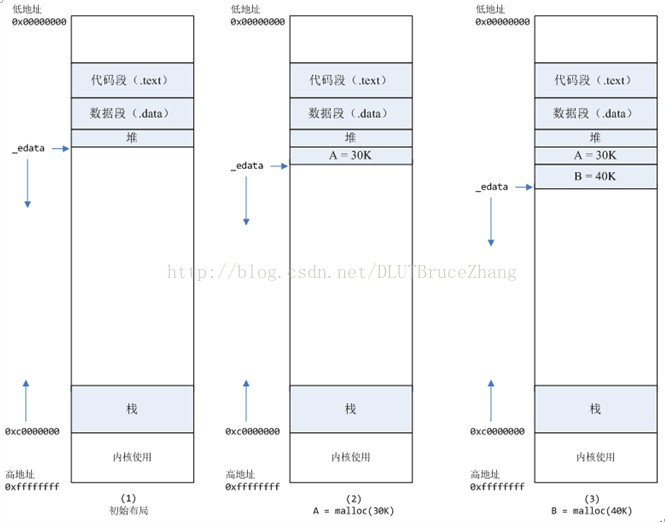
进程地址空间其中 对堆和栈的描述反了。

地址由下向上依次增高



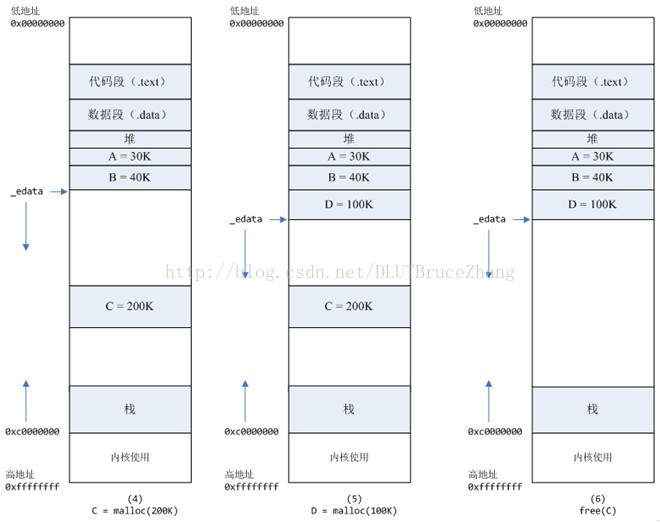
通常情况下

在malloc内存小于128k时，会调用系统内核的brk()方法，在堆内存最上面申请一块内存，基于虚拟内存中的中的\_edata指针实现；



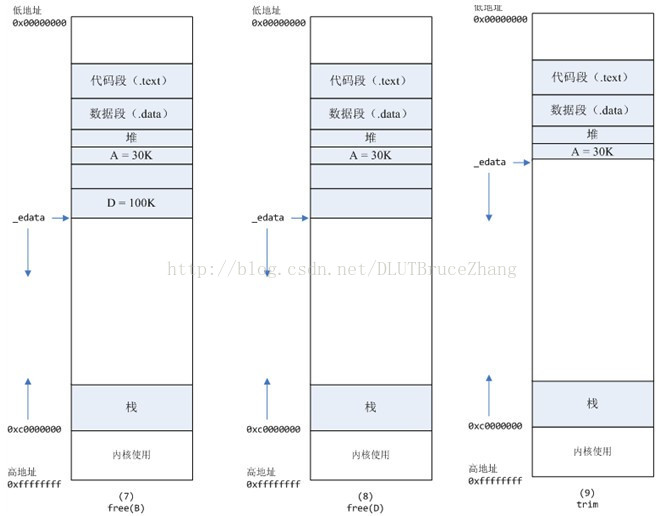
然而在申请时不会为其分配物理内存，只有在对这块虚拟内存进行第一次读写时，才会触发却缺页中断，为其分配物理内存；如果这块malloc的内存一直没有被访问，那么他将一直都没有物理内存。

默认情况下， malloc函数分配内存，如果请求内存大于128K（可由M\_MMAP\_THRESHOLD选项调节），那就不是去推\_edata指针了，而是利用mmap系统调用，从堆和栈的中间分配一块虚拟内存。



这样子做的主要原因是：

      brk分配的内存需要等到高地址内存释放以后才能释放（例如，在B释放之前，A是不可能释放的，这就是内存碎片产生的原因），而mmap分配的内存可以单独释放。



进程调用free(B)以后，如图7所示：

B对应的虚拟内存和物理内存都没有释放，因为只有一个\_edata指针，如果往回推，那么D这块内存怎么办呢？

当然，B这块内存，是可以重用的，如果这个时候再来一个40K的请求，那么malloc很可能就把B这块内存返回回去了。

进程调用free(D)以后，如图8所示：

        B和D连接起来，变成一块140K的空闲内存。

默认情况下：

       当最高地址空间的空闲内存超过128K（可由M\_TRIM\_THRESHOLD选项调节）时，执行内存紧缩操作（trim）。在上一个步骤free的时候，发现最高地址空闲内存超过128K，于是内存紧缩，变成图9所示。